Hybrid Behavioral-Based Multiobjective Space Trajectory Optimization

Massimiliano Vasile

Department of Aerospace Engineering, University of Glasgow, James Watt South Building, G12 8QQ, Glasgow, UK
m.vasile@aero.gla.ac.uk

In this chapter we present a hybridization of a stochastic based search approach for multi-objective optimization with a deterministic domain decomposition of the solution space. Prior to the presentation of the algorithm we introduce a general formulation of the optimization problem that is suitable to describe both single and multi-objective problems. The stochastic approach, based on behaviorism, combined with the decomposition of the solutions pace was tested on a set of standard multi-objective optimization problems and on a simple but representative case of space trajectory design.

1 Introduction

The design of a space mission steps through different phases of increasing complexity; generally, the first step is a mission feasibility study. In order to be successful, the feasibility study phase 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 trajectories that have to be optimal with respect to one or more criteria. In mathematical terms, the problem can be formulated as a search for multiple local minima, or as a multi-objective optimization problem.

In both cases, it is desirable to have a collection of several optimal solutions. Normally in literature, single objective and multi-objective optimization are treated as two distinct problems with different algorithms developed to address one or the other (see [1, 3, 4, 5, 6, 7, 8] for some examples of algorithms for global single objective optimization and [9, 10, 11, 12] for some examples of algorithms for multi-objective optimization). In most of the cases Evolutionary Algorithms seem to be the preferred method and many examples exist of their use to address both single objective and multi-objective problems in space trajectory design: Gage et al. has shown the effectiveness of genetic algorithms with niching technique compared to a simple grid search for the optimization of bi-impulsive transfers [13], Coverstone et al. used genetic algorithms for low-thrust trajectory design [15, 16], Gurfil et al. used niching genetic algorithms for the characterization of geocentric orbits [14], Vasile proposed a hybridization of evolutionary algorithms with SQP methods for the design of weak stability transfers [17] and an hybridisation with branch and bound for low-thrust trajectory design [18], and Dachwald et al. proposed the combination of a neurocontroller and of Evolutionary Algorithms for

C.-K. Goh, Y.-S. Ong, K.C. Tan (Eds.): Multi-Objective Memetic Alg., SCI 171, pp. 231–253.
springerlink.com

the design of low-thrust trajectories[19]. More recently, an comparison of several global optimization methods applied to the optimization fo space trajectories showed that Differential Evolution outperforms GAs on some trajectory design problems [20, 21].

In general, all evolutionary-based approaches for global optimization implement some heuristic derived from 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 chapter 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 composed of the solution space. The combination of individualistic and social behaviors aims at an optimal balance between global search and local convergence (or exploration versus exploitation).

Furthermore, a unified formulation is proposed that can be applied to the solution of both multi-objective and single objective problems in which the aim is to find a set of optimal solutions, rather than a single one. In order to improve the exploration of the search space and to collect as many local minima as possible, the proposed metaheuristic was hybridized with a domain decomposition technique.

2 General Problem Formulation

The general problem both for single and multi-objective optimization is to find a set X, contained in a given domain D, of solutions $\mathbf{x}$ such that the property $P(\mathbf{x})$ is true for all $\mathbf{x} \in X \subseteq D$,

$$X = \{\mathbf{x} \in D \mid P(\mathbf{x})\} \tag{1}$$

where the domain D is a hyper-rectangle defined by the upper and lower bounds on the components of the vector $\mathbf{x}$,

$$D = \left\{x_i \mid x_i \in [b_i^l \; b_i^u] \subseteq \Re, \; i = 1, ..., n\right\} \tag{2}$$

All the solutions satisfying property P are here defined to be optimal with respect to P, or P-optimal, and X can be said to be a P-optimal set. Now, the property P might not identify a unique set, for example if P is Pareto optimality, X can collect all the points belonging to a local Pareto front. Therefore we can define a global optimal set X_{opt} such that all the elements of X_{opt} dominate the elements of any other X,

$$X_{opt} = \left\{\mathbf{x}^* \in D \mid P(\mathbf{x}^*) \wedge \forall \mathbf{x} \in X \Rightarrow \mathbf{x}^* \prec \mathbf{x}\right\} \tag{3}$$

where $\mathbf{x}^* \prec \mathbf{x}$ represents the dominance of the $\mathbf{x}^*$ solution over the $\mathbf{x}$ solution.

If we are looking for local minima, the property P is to be a local minimiser or a solution $\mathbf{x}^*$ can be said to dominate solution $\mathbf{x}$ if the associated value of the objective function $f(\mathbf{x}^*) < f(\mathbf{x})$. In this case X_{opt} would contain the global optimum or a set of global optima all with the same value of f.

In the case of multiple objective problems, given a set of solution vectors we can associate to each one of them a scalar dominance index I_d such that:

$$I_d(\mathbf{x}_j) = |\{i \mid i \wedge j \in N_p \wedge \mathbf{x}_i \prec \mathbf{x}_j\}| \tag{4}$$

where the symbol $|.|$ is used to denote the cardinality of a set and N_p is the set of the indices of all the given solution vectors. Here and in the following, a solution vector $\mathbf{x}_i$ is said to be dominating a solution vector $\mathbf{x}_j$ if the values of all the components of the objective vector $\mathbf{f}(\mathbf{x}_i)$ are lower than or equal to the values of all the components of the objective vector $\mathbf{f}(\mathbf{x}_j)$ and at least one component is strictly lower. In this case, for the $j-th$ solution, $P(\mathbf{x}_j)$ simply defines the property of being not-dominated by any other solution in the set N_p, thus:

$$X = \{\mathbf{x}_j \in D \mid I_d(\mathbf{x}_j) = 0\} \tag{5}$$

For constrained problems, the property P is to be optimal, either locally or Pareto, and feasible at the same time. The property P can be expressed through a single scalar value or through a set of values and relationships. It can be a bolean value, a real number or a fuzzy expression (e.g. bad, average, good).

3 A Behavioral Prospective

The search for a set of solutions can be broken down to a three steps process: collecting information, making decision, taking action. For a black-box problem the collection of information is generally performed by sampling the solution space. The aim in this respect is to minimize the number of samples required to find the desired solution or set of solutions. The decision making process consists of deciding what action to take at every step of the search, selecting who does what in the case of multiple entities and deciding when to stop the search. The action step consists of implementing the selected actions by the selected entity. The three steps are required to be automatic with minimum human intervention, which means that the decision to orient the search in one or another direction should not require the human judgment (e.g. restrict the search space, increase the number of samples in a specific region).

Let us assume that a virtual agent is endowed with the ability of collecting pieces of information, making decisions and implementing actions. The decision making process could involve a long term planning of actions a closed-loop control mechanism or some sort of action-selection process in response to stimuli. For example in particle swarm optimization (PSO)[6] the velocity of each particle i is computed with a close-loop control mechanism:

$$\mathbf{v}_{i+1} = w\mathbf{v}_i + \mathbf{u}_i \tag{6}$$

where w is a weight and given the random numbers r_1,r_2 and the weights a_1 and a_2 the control $\mathbf{u}_i$ has the form:

$$\mathbf{u}_i = a_1 r_1(\mathbf{x}_i - \mathbf{x}_{gi}) + a_2 r_2(\mathbf{x}_i - \mathbf{x}_{go}) \tag{7}$$

The search is continued till the decision to stop is taken. The control function requires a piece of information collected by the particle $\mathbf{x}_{gi}$ and one collected by another particle $\mathbf{x}_{go}$. In this case, the decision making process includes the selection of the particle in $\mathbf{x}_{go}$.

Let us assume that the virtual agent is equipped with a set of actions and an action-selection process. If more than one agent exists then some of the actions can be regarded

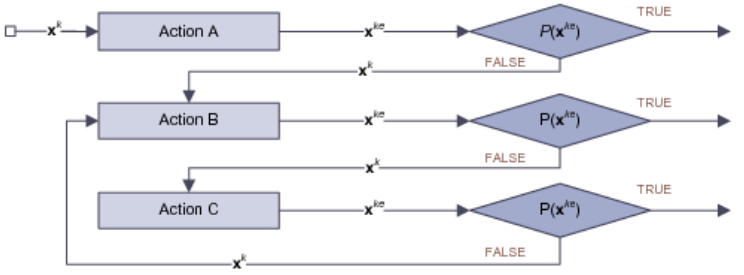

Fig. 1. Example of action selection mechanism

as individualistic, since involve only one agent at time, while others as social since require interaction among multiple agents. A collection of actions and the action selection process define a behavior. When at step k an agent $\mathbf{x}^k$ implements an action, it produces an outcome $\mathbf{x}^{k_e}$ according to:

$$\mathbf{x}^{k_e} = \mathbf{x}^k + \beta(\mathbf{x}^k, \mathbf{x}^{k-1}, \Pi_k, A_l) \tag{8}$$

where β is a function of the current state of the agent $\mathbf{x}^k$, the current state of the population Π_k, the past state of the agent $\mathbf{x}^{k-1}$ and the information about the past state of the population stored in an archive A_l. Note that β is not analytical but is an algorithm that selects an action, assigns a value and return a variation $\Delta\mathbf{x}^k$.

For example, for individualistic behaviors (see Fig. 1 and the next section), each agent can perform three types of actions, **A**, **B** and **C**. This general scheme accommodates two types of heuristics: Action **A** generates always the same outcome every time is performed once a solution vector $\mathbf{x}$ is given (e.g. inertia in PSO), while Actions **B** and **C** generate different values for the same $\mathbf{x}$ every time they are performed (e.g. mutation in EA[1]). These last two actions are repeated until an improvement is registered or a maximum number of attempts is reached. The index k_e is increased by one every time an action is performed, and every action makes use of the agent status $\mathbf{x}$, the status of other agents and of the outcome of the proceeding actions.

4 MultiAgent Collaborative Search

A population of virtual agents (i.e., points within H)is deployed in the search space:

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

Each agent is associated to a solution vector $\mathbf{x}$ and endowed with a set of basic actions forming a behavior. The entire population evolves, through a number of steps, toward the set X. At each step, the agents collect clues about the environment and implement actions according to an action selection mechanism. Some of the actions are devoted to acquiring new information (sampling the solution space), others to displacing agents, other actions are instead to exchange information among the agents. We implement a set

of actions, derived from PSO, EA and Differential Evolution (DE)[4] and very simple, basic action selection mechanisms. The general scheme both for a single agent and for a group is: select actions, implement actions, evaluate actions, make decision.

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 Section 4.2.1). The intersection $S^{\mathbf{x}} \cap H$ basically represents the local region around agent $\mathbf{x}$ which we want to explore. We also associate an *effort* value $s(\mathbf{x})$ to each agent $\mathbf{x}$, which specifies the amount of computational effort we want to dedicate to the exploration of $S^{\mathbf{x}}$. This value is updated at each iteration (see, again, Section 4.2.1).

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

Multiagent Collaborative Search

Step 0. Initialization. Generate an initial population of agents Π_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 [22]). A hyperrectangle $S^{\mathbf{x}_j^0}$ is associated to the j-th agent $\mathbf{x}_j^0 \in \Pi_0$. 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 $\mathbf{x}_j^0 \in \Pi_0$ is fixed to the same value $s_{\max}$ (equal to n in the computations) for all agents in Π_0. Set $k = 0$.

Step 1. Social Behavior. A set of social actions, specified by a *social behavior* are applied to the population Π_k. In particular two sets of social actions are implemented:

- Collaboration. The agents exchange information with each other. Each set of communication actions gives rise to new sampled points. Some of them identify the new location of a subset of the collaborating agents. See Section 4.1.1.
- Repulsion. If two or more agents are too crowded one or more are reallocated in the search space. See Section 4.1.2.

Step 2. Filtering. A *filter* partitions population Π_k into two subsets Π_k^{in}, the population within the filter, and Π_k^{out}, the population outside the filter. See Section 4.3.

Step 3. Individualistic Behavior. A set of individualistic actions, specified by an *individualistic behavior*, are applied to each agent $\mathbf{x} \in \Pi_k$.

- Local Exploration. These set of actions allows 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 Π_k is updated by replacing $\mathbf{x}$ with its improvement. See Section 4.2.
- Hyperrectangle and effort update. The size parameter ρ and the effort parameter s associated to each agent within the filter are updated according to some rule. See Section 4.2.1.

Step 4. Archive update. Apply filtering and update archive A_l (see Section 4.4).

Step 5. Stopping rule. A stopping rule is checked (see Section 4.5). If it is not satisfied, then set $k = k+1$ and go back to Step 1. If it is satisfied, then update the archive A_l by adding the current population, i.e., set $A_l = A_l \cup P_k$.

4.1 Social Behavior

Social behavior is defined through a set of communication actions (collaboration), a decision making process to select the outcome of the communication actions, a repulsion mechanism to limit crowding and increase diversity and a shared memory mechanism to exploit the social knowledge acquired during the search.

4.1.1 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 $\mathbf{x}_1$ considered to be the better one according to property P, three different actions are defined. Two of them are defined by adding to $\mathbf{x}_1$ a step $\Delta\xi$ defined as follows

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

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 $\Delta\xi$ 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 a 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 the three operations give rise to four new samples, denoted by $\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3, \mathbf{y}_4$.

The first of the two parent agents is selected at random in the worst half of the current population (from the point of view of the property P), while the second parent is selected at random from the whole population. Selecting the first parent from the best half generally reduces the diversity of the population and may cause premature convergence.

Note that here all the communication actions are selected and implemented sequentially. However, according to the general scheme mentioned above, a different action selection mechanism can adaptively choose the most appropriate subset of actions at every step. The decision making process is used to select which of the four samples will be used. Each pair of parent agents $\mathbf{x}_1$ and $\mathbf{x}_2$ generates four samples $\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3$ and $\mathbf{y}_4$. Then, a tournament, based on the property P, is started between the worst of the two parents and the best of the four samples. The winner of the tournament will be the new location in the solution space of the worst parent agent. When an agent is displaced no update of ρ and s is performed.

4.1.2 Repulsion

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

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

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.

4.1.3 Shared Memory

The archive A_l is used to direct the movements of those agents for which P is false. For all agents for which the property P is not true at step k the inertia component is recomputed as:

$$\Delta\xi = r(\mathbf{x}_{A_l} - \mathbf{x}^k) \tag{9}$$

where $\mathbf{x}_{A_l}$ is an element taken from the archive. The elements in the archive are ranked according to their relative distance or crowding factor. Then, every agent for which P is false picks the least crowded element $\mathbf{x}_{A_l}$ not already picked by any other agent.

[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.

4.2 Individualistic Behavior

The individualistic behavior is defined through a set of local exploration actions, an action selection mechanism, a decision making process to select the outcome of the exploration and an adaptive update of the resources and of the regions $S^{\mathbf{x}}$.

At every generation, a behavior β is used to generate the set of exploration actions. 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:

$$\beta = \{\Delta\xi \ \mid \mathbf{x}_j^k + \Delta\xi \in H \text{ and } \Delta\xi = z_\beta(\mathbf{x}_j^k, \mathbf{x}_j^{k-1}, \mathbf{w}, \mathbf{r}, \Pi_k)\} \tag{10}$$

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 $\mathbf{x}_j^k + \Delta\xi$ is called a *child* of agent j. In what follows we describe the different kinds of actions employed in this chapter.

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:

$$\Delta\xi = \bar{\lambda}(\mathbf{x}_j^k - \mathbf{x}_j^{k-1}) \tag{11}$$

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., [8, 4]). 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

$$\Delta\xi = \mathbf{x}_j^k - (\mathbf{x}_{i_1}^k + (\mathbf{x}_{i_3}^k - \mathbf{x}_{i_2}^k)) \tag{12}$$

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

Random-Walk. 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:

$$\Delta\xi_i = w_1 r^t(\ell_i - x_i) + (1 - w_1) r^t(u_i - x_i) \tag{13}$$

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 (13) with $t = 1$ and $p_i = 0.5$;
- a component i is selected at random and perturbed according to (13) with $t = 1$ and $p_i = 0.5$;
- a component i is selected at random and perturbed according to (13) 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:

$$\Delta\xi = \alpha_2 r^t(\mathbf{y}-\mathbf{x}) + \alpha_1(\mathbf{y}-\mathbf{x}). \tag{14}$$

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.

4.2.1 Size and Effort Update

Given an agent $\mathbf{x}_j^k \in \Pi_k^{in}$, 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 Π_k (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.

4.3 Filtering

Given a population Π_k, a filter simply subdivides the population into two parts, Π_k^{in} and Π_k^{out}. Π_k^{in} contains the best members of the population Π_k, i.e., those with the best fitness values, while Π_k^{out} contains all the other individuals in Π_k. The main difference between agents inside and outside the filter is that on agents outside the filter, only mutation actions (see equation (13) 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 Π_k^{out}), while also other actions, allowing a deeper local exploration, are performed on agents inside the filter (see the following Section 4.2). Moreover, values ρ and s are only updated for agents in Π_k^{in} (see the following Section 4.2.1). In the 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 Archive Update

Let ρ_{tol} be a small threshold value, $\mathbf{x}_i$ be an agent whose size parameter is below the threshold value, i.e., $\rho(\mathbf{x}_i) < \rho_{tol}$ and 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 in L_i (from the point of view of P), then all agents in L_i are randomly regenerated within the current domain H and $\mathbf{x}_i$ is inserted in archive A_l, 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.

4.5 Stopping Rule

The stopping rule is quite simple: the search within a subdomain is stopped when a prefixed number N_e of function evaluations is reached.

4.6 Definition of P for Multiobjective Optimization

Although the problem formulation through the definition of P is general and applicable to both single objective and multiple objective optimization problems, either constrained or not, the actual property is substantially different depending on the type of problem.

For box constrained multi-objective problems the property P can be defined by the value of the scalar dominance index I_d, thus:

$$I_d(\mathbf{x}) > I_d(\mathbf{x}_{k_e}) + \varepsilon \Rightarrow P(\mathbf{x}_{k_e}) = true \tag{15}$$

where ε is now the minimum expected improvement in the computation of the dominance. Note that this easily accommodates the concept of ε dominance.

Now, when multiple outcomes with the same dominance index are generated by either social or individualistic actions, the one that corresponds to the longest vector difference in the criteria space with respect to $\mathbf{x}$ is considered. Note that in many situations the action selection scheme in Fig. 1 generates a number of solutions that are dominated by the agent $\mathbf{x}$. Many of them can have the same dominance value; therefore in order to rank them, we use the modified dominance index:

$$I_d(\mathbf{x}) = \left|\left\{j \mid f_j(\mathbf{x}_{k_e}) = f_j(\mathbf{x})\right\}\right| \kappa + \left|\left\{j \mid f_j(\mathbf{x}_{k_e}) > f_j(\mathbf{x})\right\}\right| \tag{16}$$

where κ is equal to one if there is at least one component of $\mathbf{f}(\mathbf{x}) = [f_1, f_2, ..., f_{N_f}]^T$ which is better than the corresponding component of $\mathbf{f}(\mathbf{x}_{k_e})$, and is equal to zero otherwise.

Now, if for the $k_e{}^{th}$ outcome, the dominance index in Eq.16 is not zero but is lower than the number of components of the objective vector, then the agent $\mathbf{x}$ is only partially dominating the $k_e{}^{th}$ outcome. Among all the partially dominated outcome with the same dominance index we chose the one that satisfies the condition:

$$\min_{k_e} \left\langle \left(\mathbf{f}(\mathbf{x}) - \mathbf{f}(\mathbf{x}_{k_e})\right), \mathbf{e} \right\rangle \tag{17}$$

where $\mathbf{e}$ is the unit vector of dimension N_f, $\mathbf{e} = \frac{[1,1,1,...,1]^T}{\sqrt{N_f}}$, and N_f is the number of objective functions.

Since the partially dominated outcomes of one agent could dominate other agents or the outcomes of other agents at the end of every evolution cycle all the outcomes are added to the archive. Then, the dominance index in Eq.4 is computed for all the elements in A_l and only the non-dominated ones are preserved.

4.7 Hybridization with Domain Decomposition

The Multiagent Collaborative Search described in the previous section is a stochastic process. In order to improve its robustness (repeatability of the result) and increase the

exhaustiveness of the search, MACS is combined with a deterministic domain decomposition technique. The search space D, is a hyperrectangle and the subdomains into which it is subdivided are also hyperrectangles. The stochastic algorithm searches on the subdomains in order to evaluate them. In this section we will give the details of the deterministic method.

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

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

$$\overline{H} \in \arg\min_{H \in \mathscr{F}} \theta(H), \tag{18}$$

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 $\mathscr{F}$ as follows:

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

Step 4. Node deletion Delete nodes from $\mathscr{F}$ according to some rule.
Step 5. Stopping rule If $\mathscr{F} = \emptyset$, 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.

4.7.1 Node Evaluation

The evaluation of a subdomain H is done by running the MultiAgent Collaborative Search (MACS) algorithm within H. The MACS algorithm explores the subdomain H and stores in a local archive A_l all the promising points in H. The local archive A_l is then compared to the global archive A_g containing all the points in the search space for which P is true. The points in $E_v(H) = (A_l \cap A_g) \cap H$ represent the evaluation of the subdomain.

4.7.2 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 a point midpoint $\tilde{\mathbf{x}}$. 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. We can now subdivide the interval $b_i - a_i$ into a number n_j of subintervals and look for the one that contains the majority of the points in H. Now if the subinterval has boundaries b_{ik} and a_{ik} with $k = 1, \ldots, n_j$, the cutting point $\tilde{x}_j$ is defined as:

$$\tilde{x}_j = \begin{cases} b_{ik} \text{if } (b_i - b_{ik}) > (a_{ik} - a_i) \\ a_{ik} \text{if } (b_i - b_{ik}) \leq (a_{ik} - a_i) \end{cases}$$

Then, we define the two new subdomains

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

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

4.7.3 The Game of Exploration

The selection of the subdomain H on which to perform a new search with MACS depends on the outcome of a simple game between two players: explorer and exploiter.

Before presenting the game and selection process, 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:

$$\omega(H) = \frac{\max\{N(H), 1\}}{N} \frac{\ell(D)}{\ell(H)} \tag{19}$$

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 (19) 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:

$$\varphi(H) = |\{i | I(\mathbf{x}_i) = 0\}| \tag{20}$$

Now, the player *explore* always tries to maximize the volume of the explored space and therefore selects the subdomain with the lowest value of ω. The player *exploiter*

always tries to maximize convergence and thus selects always the subdomain with the highest value of φ. If both players play explore (explore-explore strategy), then the algorithm will select the subdomain with the smallest ω for further exploration, if both players play converge (converge-converge strategy), then the algorithm will select the subdomain with the highest φ. If *explorer* plays explore first and *exploiter* plays converge (explore-converge strategy) then, the algorithm will rank the subdomain according to ω and then, among the first n_ω of them will take the one with the largest number of elements in A_l. Vice versa, if *exploiter* plays first (converge-explore strategy), the algorithm will rank the subdomains according to φ and among the first n_φ will select the one with the lowest ω. The selection of the strategy to play depends on the outcome of the game.

If both players play converge then exploiter gets a reward only if it finds an improvement while explorer gets no reward whatsoever. Since we are interested in collecting as many different elements of a set as possible, this strategy is not convenient for any of the two players. If both players play explore then the explorer gets a reward while the exploiter gets half of the reward of the explorer only if an improvement is registered. This strategy is convenient if no improvements are registered with any other strategy or if no information is available. Thus, it is the first strategy that both players play at the beginning. The outcome of the exploration of a subdomain could be: a) a number of points belonging to the set X higher than the one already available, b) a number of points belonging to the set X lower or equal to the number already available, c) no points belonging to the set X. In the last case the algorithm plays explore-explore, in case a) though the subdomain looks promising the higher number of points could suggest an over-exploitation of the subdomain and the algorithm then plays explore-converge. Finally, in case b) the algorithm plays converge-explore.

4.8 Discussion

The hybrid behavioral-based approach presented in the previous sections is one possible implementation of the concept expressed by Eq.8. In particular we used a very simple action selection mechanism for both social and individualistic actions. More sophisticated mechanisms may allow for a reduction of the number of function evaluations or could include a learning mechanism. Furthermore, in the present implementation there is a limited use of the past history of the search process. The behavior Eq.8 depends on the archive A_l, which works as a repository of the social knowledge, and on the state of the agent at step $k-1$, therefore only a partial history is preserved and used.

4.9 Preliminary Optimization Test Cases

The proposed optimization approach, combining MACS with deterministic domain decomposition, was implemented in a software code in Matlab called EPIC, and tested on a number of standard problems, well known in literature. In a previous work [24, 25], EPIC was tested on single objective optimization problems related to space trajectory design, showing good performances.

Here we initially used a set of test functionsthat can be found in [10, 9]. In the next section EPIC will be tested on a typical space trajectory optimization problem.

Table 1. Multiobjective test functions

Scha	$f_2 = (x-5)^2 ; f_1 = \begin{cases} -x & if\ x \le 1 \\ -2+x & if\ 1 < x < 3 \\ 4-x & if\ 3 < x \le 4 \\ -4+x & if\ x > 4 \end{cases}$	$x \in [-5,10]$
Deb	$f_1 = x_1$ $f_2 = (1+10x_2)\left[1-\left(\frac{x_1}{1+10x_2}\right)^{\alpha} - \frac{x_1}{1+10x_2} sin(2\pi q x_1)\right]$	$x_1, x_2 \in [0,1]$ $\alpha = 2;.\ q = 4$
T4	$g = 1+10(n-1)+\sum_{i=2}^{n}[x_i^2 - 10cos(2\pi q x_i)];$ $h = 1-\sqrt{\frac{f_1}{g}}$ $f_1 = x_1; f_2 = gh$	$x_1 \in [0,1];$ $x_i \in [-5,5];$ $i = 2,\ldots,n$

The test case, *T4*, is commonly recognized as one of the most challenging problems since it has 21^9 different local Pareto fronts of which only one corresponds to the global Pareto-optimal front. In this case the exploration capabilities of each single agents are enough to locate the correct front with a very limited effort. In fact even with just five agents it was possible to reconstruct (see Fig. 2b) the correct Pareto front 20 times over 20 different runs. The total number of function evaluations was fixed to 20000 for each of the runs, though already after 10000 function evaluations EPIC was always able to locate the global front (see Fig. 2a).

Despite the small number of agents the sampled points of the Pareto are quite well distributed with just few and limited interruptions. The use of a limited number of agents instead of a large population is related also to the complexity of the algorithm. In fact the complexity of the procedure for the management of the global archive is of order $n_A(n_p+n_A)$, where n_A is the archive size and n_p is the population size, while the complexity of the exploration-perception mechanism is of order $n_p(n+n_p)$, therefore, even

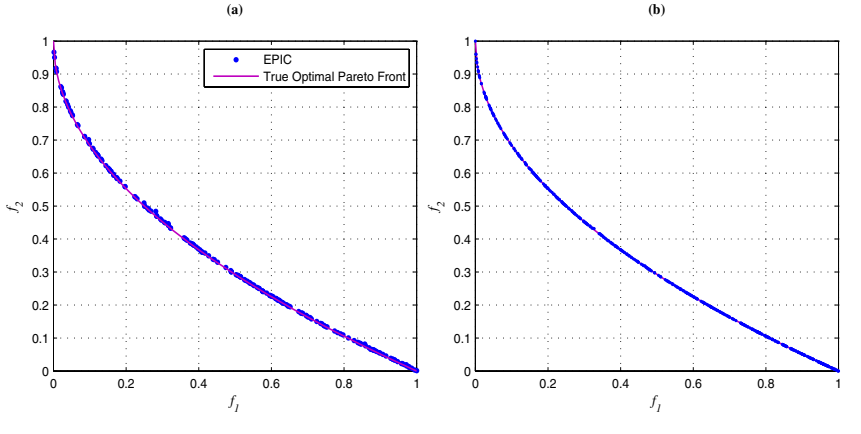

Fig. 2. Pareto front for the test case *T4*: a) 10000 function evaluations, b) 20000 function evaluations

Table 2. Comparison of the average Euclidian distances between 500 uniformly space points on the optimal Pareto front for various optimization algorithms

Approach	T4	Scha	Deb
EPIC	1.542e-3 (5.19e-4)	5.4179e-4 (1.7e-4)	1.4567e-4 (3.61e-4)
NSGA-II	0.513053 (0.118460)	0.002536 (0.000138)	0.001594 (0.000122)
PAES	0.854816 (0.527238)	0.002881 (0.00213)	0.070003 (0.158081)
MOPSO	0.0011611 (0.0007205)	0.002057 (0.000286)	0.00147396 (0.00020178)

if the algorithm is overall polynomial in population dimension, the computational cost would increase quadratically with the number of agents.

As an additional proof of the effectiveness of MACS, we compare the average Euclidean distance of 500 uniformly spaced points on the true optimal Pareto front from an equal number of points belonging to the solution found by EPIC, NSGA-II, PAES and MOPSO (see Table 2).

5 Application to Space Trajectory Design

In this section we present an apparently very simple example of space trajectory optimization. It is a two impulse transfer from the Earth to the asteroid Apophis. As it often happens, the goal is to minimize the propellant consumption and the time of flight. The cost of the mission, in fact, increases proportionally to both quantities.

The propellant consumption is a function of the velocity change, or Δv[23], required to depart from the Earth and to rendezvous with Apophis. Both the Earth and Apophis are point masses, with the only source of gravity attraction being the Sun. Therefore, the spacecraft is assumed to be initially at the Earth, flying along its orbit. The first velocity change, or Δv_1, is used to leave the orbit of the Earth and put the spacecraft into a transfer orbit to Apophis. The second change in velocity, or Δv_2, is then used to inject the spacecraft into Apophis' orbit.

The two Δv's are a function of the positions of the Earth and Apophis at the time of departure t_0 and at the time of arrival $t_f = t_0 + T$, where T is the time of flight. The contour lines of the sum of the two Δv is represented in Fig. 3 for $t_0 \in [3675, 10500]^T$ MJD2000 and $T \in [50, 900]$days.

As can be seen in the specified solution space D there is a large number of local minima. Each minimum has a different value but some of them are nested, very close to each other with similar values. For each local minimum, there can be a different front of locally Pareto optimal solutions. The global Pareto front should contain the best transfer with minimum total Δv and the fastest transfer with minimum TOF.

The best known approximation of the global Pareto front is represented in Fig. 4. It is a disjoint front corresponding to two basins of attraction of two minima as can be seen in Fig. 5. The lower front is made of solutions with a very low transfer time, the upper front, instead, is made of solutions with a much longer transfer time but a total Δv similar to the one of the solutions belonging to the lower front.

Besides containing local minima with similar Δv, the two basins of attraction present similar values of the first objective function. Converging to the upper front is therefore

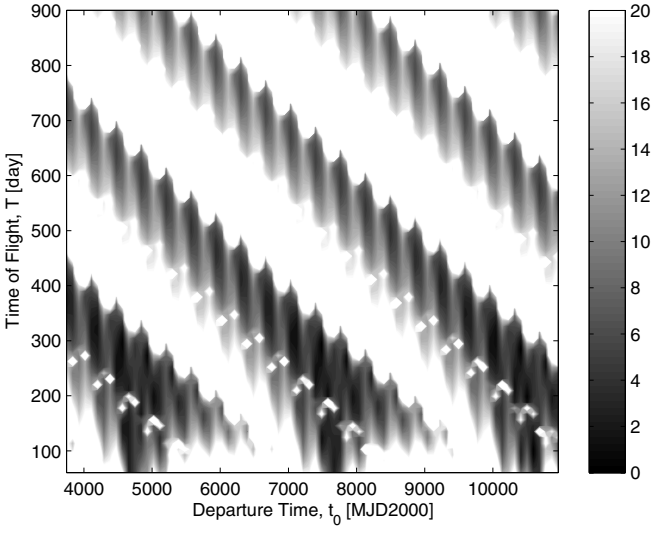

Fig. 3. Earth-Apophis search space

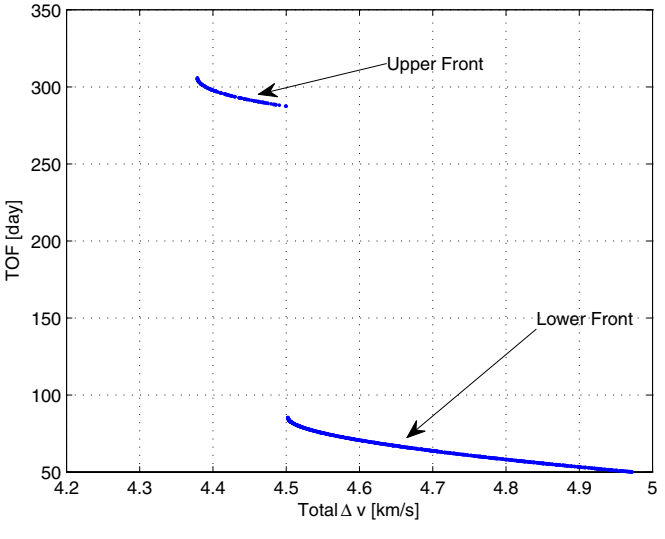

Fig. 4. Earth-Apophis transfer: dual front

quite a challenge since the lower front has a significantly lower value of the second objective function. It is only when the optimizer converges to the a vicinity of the local minimum of the upper front that the latter becomes not dominated by the lower front. The upper front contains the global minimum with a total $\Delta v = 4.3786$ k/s while the lower front contains only a local minimum.

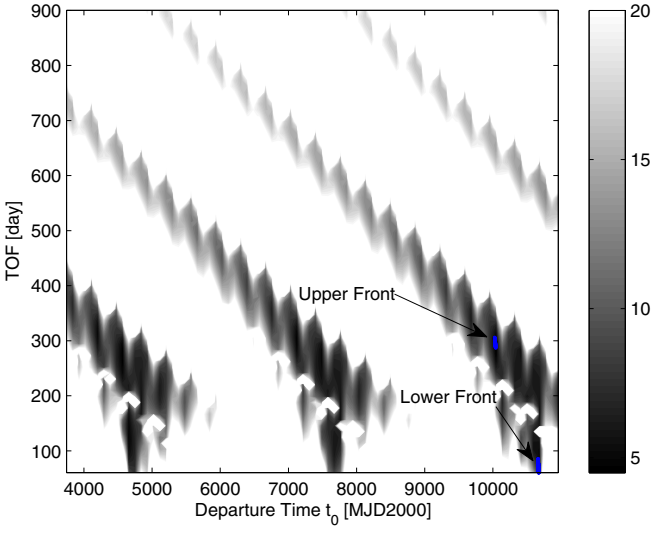

Fig. 5. Earth-Apophis transfer: distribution of the solutions in the search space

Table 3. Comparison of the metrics M_1 and M_2 ont he Earth-Apophis case

Approach	Metric	3000	6000	9000
EPIC	M_1	4% (39.00)	38% (22.28)	51% (17.53)
	M_2	1.89 (5.9)	1.66 (5.14)	0.66 (3.22)
NSGA-II	M_1	10% (26.51)	19% (27.54)	24% (25.63)
	M_2	20.96 (27.78)	11.99 (16.20)	10.12 (12.82)

In order to test the multi-objective optimizers with this simple but typical space trajectory design problem, we define two metrics:

$$M_1 = \frac{1}{M_p}\sum_{i=1}^{M_p}\min_{j\in N_p} 100\left\|\frac{\mathbf{f}_j-\mathbf{f}_i}{\mathbf{f}_i}\right\| \tag{21}$$

$$M_2 = \frac{1}{N_p}\sum_{i=1}^{N_p}\min_{j\in M_p} 100\left\|\frac{\mathbf{f}_j-\mathbf{f}_i}{\mathbf{f}_j}\right\| \tag{22}$$

Although similar, the two metrics are measuring two different things: M_1 is the sum, over all the elements in the global Pareto front, of the minimum distance of all the elements in the Pareto front N_p from the the i^th element in the global Pareto front. M_2, instead, is the sum, over all the elements in the Pareto front N_p, of the minimum distance of the elements in the global Pareto front from the i^th element in the Pareto front N_p.

Therefore, if N_p is only a partial representation of the global Pareto front but is a very accurate partial representation, then metrics M_1 would give a high value and metrics M_2

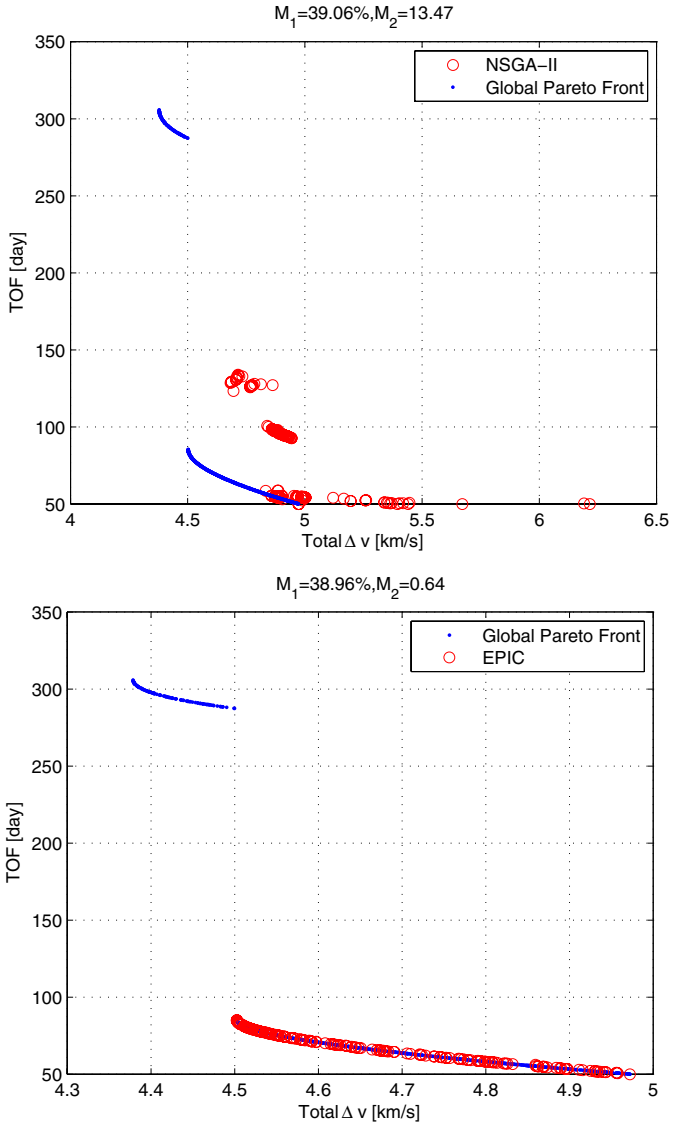

Fig. 6. Earth-Apophis transfer: comparison of two not-converged runs

a low value. If both metrics are high then the Pareto front N_p is partial and poorly accurate. In Table 3 we represented metrics M_1 and metrics M_2, in brackets, for an increasing number of function evaluations and for two different optimizers.

We compared EPIC against the optimization algorithm that displayed the best performances on this case, NSGA-II. NSGA-II was run several times with crossover probability ranging from 0.5 to 0.9 in order to tune the main parameters, in particular we

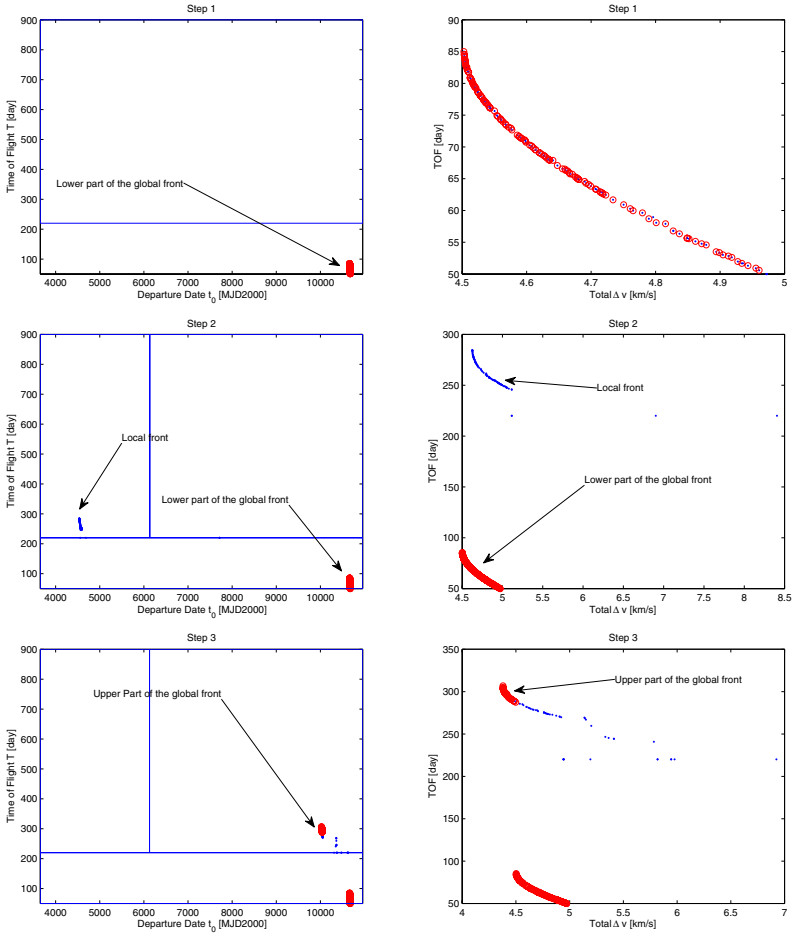

Fig. 7. Earth-Apophis transfer: domain decomposition process. The red circles are the non-dominated solutions at each step while the blues dots are the whole set of solutions3.

performed several tests to find a good population size. The results in Table 3 were the best obtained over all the runs. For 3000 function evaluations we used a population of 200 individuals while for the 6000 evaluations and the 9000 evaluations test we used a population of 300 individuals since it was returning better results.

On the other hand EPIC was run with a very small population of 10 agents, with a filter size of 5 agents. For, 3000 evaluations we did not use the domain decomposition. For 6000 evaluations we used a decomposition in 2 subintervals. For 9000 evaluations, we tested a 3 subdomain decomposition and a 2 subdomain decomposition. In both cases the first cut is always along the TOF coordinate.

For each number of function evaluations we performed 100 independent runs. The table reports the percentage of times the metric M_1 is below 2%, and in brackets the average value of M_1 over the 100 runs. It should be noted that a value of M_1 larger than 2% up to 10% does not necessary correspond to a fully unsuccessful run. The 2% tolerance, on the other hand, guarantees that the algorithm was able to identify both parts of the Pareto front with a good distribution of the points.

In the Table 3 we also reported the average value and the standard deviation (in brackets) of metric M_2. This second metric measures the accuracy of the convergence to even a portion of the whole Pareto front.

As can be seen for a low number of function evaluations, NSGA-II performs better than EPIC, though EPIC achieves a better value of M_2, which means a better local convergence on average. Conversely, when NSGA-II is not converging to the global front, is converging to a local front, while when EPIC is not converging to the whole global front is converging to a portion of it. Figs. 6 are showing two typical cases in which the metric M_1 is over 30% for both the optimization algorithms. In both cases the number of function evaluations is 3000.

Fig. 7 shows an example of the domain decomposition process. At step 1 MACS is run on the entire search space D and in this example identifies only the lower part of the global front. The search space is then partitioned in two subdomains and MACS is run on the unexplored one. The second step leads to the identification of a local front. The third step explores the unexplored subdomain and identifies the upper part of the global front. At each step MACS was run for 3000 function evaluations.

For a higher number of function evaluations, NSGA-II progressively increases the number of successes, though the accuracy remains lower than for EPIC. The large population of NSGA-II, in fact, samples the solution space better than the small population of EPIC. On the other hand the decomposition of the solution space allows EPIC to increase the exploration even with a small number of agents. This is demonstrated by the number of successful runs which is more than double than the one of NSGA-II.

6 Conclusions

In this chapter we presented a hybrid behavioral-based search algorithm for multiobjective optimization problems. We showed its effectiveness on a set of standard problems and in particular on a space trajectory design problem. The latter, though very simple, well illustrates some typical difficulties in the use of global methods for the design of space trajectories.

Though some of them, like NSGA-II, perform statistically well, still on a small number of trials the result could be only a partial reconstruction of the full Pareto front or a full but inaccurate reconstruction of it. The proposed hybridization increases the robustness (i.e. repeatability of the result) and the convergence accuracy at the same time. Even the stochastic part of the algorithm, based on a multiagent system, performs better compared to known optimizer. This is mainly due to good mixture of actions performing both local and global search and to the adaptivity of the search.

References

1. 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)
2. Horst, R., Tuy, H.: Global optimization: deterministic approaches, 3rd edn. Springer, Berlin (1996)
3. Stephens, C.P., Baritompa, W.: Global optimization requires global information. Journal of Optmization Theory and Applications 96, 575–588 (1998)
4. Storn, R., Price, K.: Differential Evolution – a Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. Journal of Global Optimization 11(4), 341–359 (1997)
5. Torn, A., Zilinskas, A.: Global Optimization. Springer, Berlin (1987)
6. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: Proceedings of the IEEE International Conference on Neural Networks, pp. 1942–1948 (1995)
7. Perttunen, C.D., Stuckman, B.E.: Lipschitzian Optimization Without the Lipschitz Constant. JOTA 79(1), 157–181 (1993)
8. Price, K.V., Storn, R.M.: Jouni A. Lampinen. Differential Evolution: A Practical Approach to Global Optimization, 1st edn. Springer, Heidelberg (December 22, 2005)
9. Deb, K., Pratap, A., Meyarivan, T.: Fast elitist multi-objective genetic algorithm: NGA-II. KanGAL Report No. 200001 (2000)
10. Deb, K., Pratap, A., Meyarivan, T.: Constrained test problems for multi-objective evolutionary optimization. KanGAL Report No. 200002 (2002)
11. Sierra, M.R., Coello, C.A.C.: A Study of Techniques to Improve the Efficiency of a Multi-Objective Particle Swarm Optimizer. Evolutionary Computation in Dynamic and Uncertain Environments, 269–296 (2007)
12. Coello, C.A.C., Lamont, G., Van Veldhuizen, D.: Evolutionary Algorithms for Solving Multi-Objective Problems, 2nd edn. Springer, New York (2007)
13. Gage, P.J., Braun, R.D., Kroo, I.M.: Interplanetary trajectory optimisation using a genetic algorithm. Journal of the Astronautical Sciences 43(1), 59–75 (1995)
14. Gurfil, P., Kasdin, N.J.: Niching genetic algorithms-based characterization of geocentric orbits in the 3D elliptic restricted three-body problem. Computer Methods in Applied Mechanics and Engineering 191(49-50), 5673–5696 (2002)
15. Hartmann, J.W., Coverstone-Carrol, V.L., Williams, S.N.: Optimal Interplanetary Spacecraft Trajectories via Pareto Genetic Algorithm. Journal of the Astronautical Sciences 46(3) (1998)
16. Rauwolf, G., Coverstone-Carroll, V.: Near-optimal low-thrust orbit transfers generated by a genetic algorithm. Journal of Spacecraft and Rockets 33(6), 859–862 (1996)
17. 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
18. De Pascale, P., Vasile, M.: Preliminary Design of Low-Thrust Multiple Gravity Assist Trajectories. Journal of Spacecraft and Rockets 43(5), 1065–1076 (2006)
19. Dachwald, B.: Optimization of interplanetary solar sailcraft trajectories using evolutionary neurocontrol. Journal of Guidance, and Dynamics (January/ February 2004)
20. 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)
21. 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)

22. Dam, R.E., van Husslage, B.G.M., den Hertog, D., Melissen, J.B.M.: Maximin Latin hypercube designs in two dimensions. Operations Research 55, 158–169 (2007)
23. 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
24. Vasile, M.: A Behavioral-based Meta-heuristic for Robust Global Trajectory Optimization. In: IEEE Congress on Evolutionary Computing (CEC 2007) Proceedings, pp. 2056–2063 (2007)
25. Vasile, M., Locatelli, M.: A hybrid multiagent approach for global trajectory optimization. Journal of Glaobal Optimization (accepted) (to appear, 2007)