Distribution replacement for improved genetic algorithm performance on a dynamic spacecraft autonomy problem

Howard Tripp, Phil Palmer

Distribution replacement for improved genetic algorithm performance on a dynamic spacecraft autonomy problem

Journal:	*Engineering Optimization*
Manuscript ID:	GENO-2009-0007.R3
Manuscript Type:	Original Article
Date Submitted by the Author:	01-Jul-2009
Complete List of Authors:	Tripp, Howard; University of Surrey, Surrey Space Centre Palmer, Phil; University of Surrey, Surrey Space Centre
Keywords:	Genetic algorithm, Dynamic problem, Distribution replacement, Markov chain, Spacecraft autonomy

Distribution replacement for improved genetic algorithm performance on a dynamic spacecraft autonomy problem

Howard Tripp[1] **Phil Palmer**[2]

1. Abstract

This article looks at the continuous on-board optimization needed for an autonomous nanosatellite platform to perform collectively within a distributed cluster. The spacecraft needs to continuously maintain its own local behaviour (defined with orthogonal chebyshev polynomials) using a genetic algorithm. The continual arrival of tasks and the external actions of other spacecraft mean that the problem is highly dynamic in nature.

Standard genetic algorithms are based around convergence which dramatically reduces the population diversity hampering performance on both multi-modal and dynamic problems such as this. A new family of distribution replacement operators is presented which have the unique ability to explicitly (rather than probabilistically) control the population diversity in fitness (rather than genome) space. This turns out to be highly beneficial for this dynamic problem and out performs all other existing replacement operators. This result is mirrored and explained analytically using a simplified problem and a Markov model.

Keywords: Genetic algorithm, dynamic problem, distribution replacement, Markov chain, spacecraft autonomy.

2. Introduction

There is increasing interest within the space community to move towards multiple platform distributed missions. Distributed missions have many potential advantages such as signal separation (*e.g.* large synthetic apertures), signal space coverage (*e.g.* multipoint sensing) and signal combination (*e.g.* data fusion) (Clement and Barrett, 2002). Concept missions of the future point towards missions involving larger numbers of smaller platforms working collaboratively for enhanced science return (Hinchey *et al.*, 2005a). The elimination of single point failures and continuously upgradability of the factorization paradigm (Brown and Eremenko, 2006) provide further engineering and economic advantages. However, there inherently becomes a point at which there are too many spacecraft for a single ground station to control, as is the current approach. Direct ground control also breaks down on longer range exploration missions such as missions to Mars and the asteroid belt. The only feasible solution is to shift more of the ground segment tasks up to increasingly intelligent autonomous spacecraft, hence reducing the costs and demands on the ground. This idea of autonomous operation has been receiving growing attention for example (Chien *et al.*, 2004 , Muscettola *et al.*, 1997, Carrel, 2007, Richards *et al.*, 2001, Clement *et al.*, 2004, Bonnet and Tessier, 2007).

An autonomous group of spacecraft requires coordination, but standard terrestrial paradigms such as negotiation (*e.g.* (Schetter *et al.*, 2000, Zetocha, 2000)), require high levels of inter-spacecraft communication, which is nontrivial in space. This article therefore focuses on a system based around the principles of stigmergy previous detailed in (Tripp and Palmer, 2008) however usage principals of swarm intelligence for spacecraft collaboration have also been proposed elsewhere (Hinchey *et al.*, 2005b, Curtis *et al.*, 2003). Stigmergy is an agent-based, behavioural coordination mechanism introduced by Grassé in 1960 (Grasse, 1960) inspired by natural living systems such as ant colonies and schools of fish. This approach endows the system with emergent properties that can causes self-organization (Karuna *et al.*, 2004) as is the case with ant food foraging and collective fish "navigation" -- realistic models of which can be created using simple behaviours rules (White, 2005) – which allows for potential implementation on platforms that have limited capabilities such as nanosatellites (*i.e.* satellites with masses ~1kg). Coordination is achieved using infrequent communication via indirect modifications to a common (digital) environment (Bonabeau *et al.*, 1999) (rather than by direct message passing) with decisions based on local information (Parunak *et al.*, 2002). Essentially it is a compromise solution that

[1] Surrey Space Centre, University of Surrey, UK. h.tripp@surrey.ac.uk

[2] Surrey Space Centre, University of Surrey, UK. p.palmer@surrey.ac.uk

combines the benefits of minimal inter-satellite links (which are highly nontrivial in space) whilst critically freeing the ground station from the micromanagement of direct remote control. Further attractive features of the system are that it is inherently scalable to large numbers of spacecraft, and due to its simple behavioural algorithms is able to run on the small low-power nanospacecraft that are characteristic of the envisaged swarms.

The system previously proposed in (Tripp and Palmer, 2008) (Figure 1) is hierarchical and "tree-like". Tasks to be performed by the cluster flow up from the ground station (root) along with some digital environment information and global performance metrics (goals). As the environment information and goals pass through the layers they are slightly modified by intermediate nodes (supervisors) based on their own local information. At leaf nodes, worker spacecraft perform tasks basing their choices on the environment information. They return completed tasks down the hierarchy along with some feedforward information about predicted future performance. Again this downstream information and tasks are modified and monitored by intermediary nodes. When this information reaches the ground station, it is subject to more processing before being looped back up, completing the cycle. Hence as the task and environment information flows around this loop it is modified by the various nodes that it passes through. Equally, this information is used to determine local behaviour and actions, giving the two properties that are needed for the indirect coordination of stigmergy.

Discussion about the characteristics and performance of the system as a whole is beyond the scope of this article. The intention here is to investigate in detail the issues surrounding the on-board payload for a single worker spacecraft. After briefly outlining the design of the worker spacecraft architecture the remainder of this article is roughly divided into three sections. The first section looks at what an onboard "behaviour" actually is and how it's appropriateness for different goals can be evaluated. The complexity of behaviour that is needed is investigated and verification of the implementation with analytical results is demonstrated (as well as benchmarking on realistic space hardware later on).

The next section looks at the continuous search for optimal behaviour using a genetic algorithm. Current operators and genetic algorithm principles are presented along with discussions surrounding the limitations of their ability to maintain diversity when faced with his continually evolving dynamic problem. The family of distribution replacement operators is outlined that solve this problem by explicitly maintaining an arbitrary "diversity of fitness" profile.

The third section introduces a simplified representative problem that can be solved using a markov chain as an analytical model for the genetic algorithm. This problem analytically demonstrates the effectiveness of distribution replacement. The final section introduces some more detail about the actual simulation problem and demonstrates that the results see with the markov analysis are mirrored in the actual optimization problem.

3. Worker spacecraft

The worker spacecraft (

Figure 2) has two main functional blocks, a higher level *Task Ordering/Prioritization Agent* (that this article is concerned with) and the lower level *Scheduling & Operations Agent* that is detailed and analyzed in complimentary work (referred to as the NEAT architecture) (Carrel and Palmer, 2005). The lower level NEAT architecture takes a (continuously evolving) set of tasks to be performed along with an associated priority for each task. Windows of opportunity are calculated (*e.g.* using orbital parameters) and resource requirements are broken down for each task. A genetic algorithm then searches for an optimal ordering of the tasks, such that all resource and scheduling constraints are observed at all times. The ordering is continuously updated as new tasks are added and priorities changed, the final output is a low level real-time activity plan for the spacecraft to implement at any given moment. Hence the lower level NEAT architecture is responsible for enforcing all the real-time resource and scheduling constraints that are required for planning spacecraft operations, and also ensures that that the spacecraft always has "something to do" wherever possible.

The work on NEAT makes a fundamental assumption: that the spacecraft is required to perform all the tasks that it is has been assigned. Hence NEAT is effectively bin packing to try and perform all tasks as quickly as possible (given the resourcing constraints and dynamic rescheduling due to failure etc). This article proposes extending the NEAT approach for use in multi-spacecraft collaboration, where there is a set of tasks that has to be performed collectively. The individual spacecraft is now no longer expected to perform every task it is aware of - only a subset of those tasks. The choice of the subset is determined by the higher level *Task Ordering/Prioritization Agent* that is the subject of this article. One approach would be for the spacecraft to negotiate amongst themselves to partition the tasks, for example (Curtis *et al.*, 2003). In this case each of the spacecraft could then hand down their specific subset to the lower level NEAT architecture for scheduling. For reasons discussed in detail in previous work (Tripp and Palmer, 2008) this is problematic, mainly due to the high inter-spacecraft communications connectivity requirements and the complexity of

the negotiation protocol, in the dynamic environment. The approach taken here (in line with the principles of stigmergy) is that the tasks can be partitioned in a probabilistic rather than rigid manner. In this work the *Task Ordering/Prioritization Agent* generates an associated weighting level, for each of the tasks that is then used by NEAT when scheduling. In the single spacecraft case, NEAT assigned each of its tasks identical weightings - simple bin packing with no consideration for permutations. Now with the weightings assigned by the higher level, permutations become important and NEAT attempts to schedule the highest weighted tasks as early as possible[3]. This weighting effectively defines a desired ordering on the tasks, and as there are too many tasks for the spacecraft to complete in a given time only the subset of highly weighted tasks will be performed. Each of the spacecraft in the cluster can assign different local weightings to the tasks in the system so that each spacecraft attempts to perform a different subset, thereby achieving a partitioning of the tasks. This weighting function, or ordering, is referred to from here on as the *Behaviour* of the Worker spacecraft. NEAT attempts to schedule the tasks into this order as best as possible but given that there may be resource conflicts that need to be resolved, temporal dependencies and dynamic task failures this desired ordering is modified see Figure 3. This flexible approach allows the spacecraft to focus on the tasks that are "assigned" to it but also gives it the ability to perform other tasks if constraints and failures mean that it would otherwise be idle. This probabilistic weighting approach also has further advantages over a rigid negotiation approach as it allows the spacecraft to continuously evolve its plans in a dynamic environment with a low overhead. With a rigid contract style negotiation by contrast, any dynamic change requires re-negotiation between spacecraft, and hence in the worst case may result in frequent chaotic changes of the assigned tasks subset. These and other system level design decisions are discussed in previous work (Tripp and Palmer, 2008).

A suitable behaviour (a probability distribution), is selected by using a genetic algorithm to search for the optimum partitioning given a set of input goals. The optimal behaviour is not fixed, but depends upon the set of tasks to be performed combined with the information contained in the feedback and feedforward environments. Having selected a behaviour (returning to Figure 1), this behaviour is then fed-forward, down the hierarchy. Tasks completed by NEAT are also returned down the hierarchy to be collated by the ground station before it periodically rebroadcasts the updated task list completing the loop.

4. Behaviour implementation

Consider the ground station, which has a set of continuously evolving tasks that need to be performed by the cluster of spacecraft. It can be assumed that these tasks are independent (all dependencies are enforced later by NEAT see above). There are also a relatively large number of tasks in the system at any one time so as to keep the spacecrafts as busy as possible and avoid wasted "idle" time. Every task to be performed requires some spacecraft resources in the form of power, memory, propellant or simply on-board computation time etc. Hence it is possible to take every behaviour and plot it on a multi-dimensional histogram with each axis representing the particular resource being considered. The value of each of the histogram columns is the number of tasks with those specific resource requirements. In this way the task list has been transformed into density distribution over a resource space. Naturally this model can be extended from a discrete histogram to a continuous distribution.

This distribution over the resource space is a key tenant of the system. It allows for the allocation of tasks not based on the tasks themselves (as in a simple scheduling algorithm), but on the regions of the resource space that they occupy. The power of this transformation is that it frees the allocation algorithms from making decisions about individual tasks (whose complexity scales with the number of tasks). Instead, as the bounds of the resource space are static and known, making allocation decisions based on resource space regions has a constant complexity independent of the number of tasks. Additionally, a fixed resource space also provides a common environment where signs and signals can be left as is needed in a stigmergy system.

Given this distribution over the resource space, a behaviour can now be defined as a probability distribution over it. In other words, it provides a probability of performing a particular task in a given region of the space. Of course it would be possible to rearrange the regions of the space based on the probability to give a logical ordering presented previously. This ordering is the order (schedule) in which tasks will be completed given the behaviour. Naturally, if there are no tasks in a particular region, nothing can be performed. Equally if there are a large number of tasks in another region, the probability of performing one of those tasks increases. Hence, the effective behaviour is in fact the task distribution multiplied by the behaviour distribution, suitably normalized as is shown in Figure 5.

[3] This does not require significant modification to the NEAT model, as having been designed collaboratively with this approach, the weighting is already built in and hence is automatically handled by the genetic algorithm.

It is prudent to start with a simple system model to aid understanding of the fundamental processes. Hence the resource space is initially limited to a single dimension. This dimension could represent power or memory etc, but a good analogy is that of user priority. Not all tasks are of equal value to the operator and this is indicated to the system in the form of a numerical value. Without loss of generality this priority value can be normalized and restricted to the range [0,1].

In this simple model, the behaviour is now the familiar one-dimensional probability distribution. In order not to limit the flexibility of the probability distributions that can be described by the system, orthogonal polynomials are considered. Orthogonal polynomial sequences have the property that each polynomial in the sequence "does not interfere" with the other polynomials. In other words, each polynomial adds unique properties to the system, eradicating any representational redundancy. By starting with the lowest order polynomial in the sequence, adding additional polynomials improves the fidelity of the composite polynomial that can be generated, as can be seen in Figure 4 (right). The composite polynomial is simply a weighted sum of the basic polynomials in the sequence. As these basic polynomials have fixed definitions, a small vector of weighting parameters is sufficient to describe the entire system - a highly compact representation. There are many different sets of orthogonal polynomials that have been derived by mathematicians. In this case the well-known Chebyshev polynomial sequence (Figure 4, left) is used that is commonly used for polynomial approximation. (Gautschi, 2004).

4.1 Formal definition of behaviour

A standard polynomial $f_n(x)$ in the Chebyshev sequence (of degree n) has the following definition (Mason and Handscomb, 2003):

$$\begin{aligned} &f_n(x) = \cos(n.\arccos(x)) \\ &\text{Hence}: \quad -1 \le x \le 1 \\ &\text{and} \qquad -1 \le f_n(x) \le 1 \end{aligned} \tag{1}$$

Polynomials in this sequence are orthogonal to each other with respect to a specific weighting function:

$$\begin{aligned} &0 = \int_{-1}^{1} w(x) f_n(x) f_m(x) dx \quad \text{if } m \ne n \\ &\text{where}: \quad w(x) = \frac{1}{\sqrt{(1-x^2)}} \end{aligned} \tag{2}$$

Now a sum of these sequential polynomials can be defined, each multiplied by a weight a_n. This vector of weights is the crucial part of the Chebyshev sequence. It allows different polynomials to exert their influence more or less strongly, and by summing together different weights of these polynomials, arbitrary distributions can be generated. Note the introduction of the square root of the weighting function which will become useful when these values are multiplied together later.

$$\mathrm{F}(\underline{a},\mathrm{x}) = \sqrt{w(x)} \sum_{n=o}^{N} a_n f_n(x) \tag{3}$$

The behaviour of the spacecraft needs to be defined over a finite resource range and so without loss of generality this is chosen to be $0 \le r \le 1$. Equally, the behaviour is in fact a probability distribution, and as such the sum of the distribution within the range must always equal 1. Hence we can define the behaviour $B(\underline{a},r)$ that is derived from $F(\underline{a},x)$ with these two constraints. As the Chebyshev sequence is defined on a different range (as above), a change of variable is immediately required with $x = 2r - 1$. A further restriction on our behaviour is that all values must be strictly positive at all points within the range, in order to be able to be interpreted as a valid probability distribution. This can be achieved by squaring the $F(\underline{a},x)$ function:

$$
\begin{aligned}
1 &= \int_0^1 B(\underline{a},r)dr \\
&= \int_0^1 \lambda\left(F(\underline{a},2r-1)\right)^2 dr \\
&= \lambda \sum_{n,m} a_n a_m \int_{-1}^{1} w(x) f_n(x) f_m(x) dx/2
\end{aligned}
\tag{4}
$$

The normalizing constant λ has been introduced in order to satisfy the total sum equals one constraint, the value of which now needs to be determined. By using the orthogonality property of the Chebyshev polynomials with respect to their weight function (eq2) this equation can be expanded out and then substantially reduced in complexity (making sure to take special care with the first term):

$$
\begin{aligned}
1 &= \frac{\lambda}{2}\left[a_0^2 \int_{-1}^{1} w(x)(f_0(x))^2 dx + \sum_{n=1}^{N} a_n^2 \int_{-1}^{1} w(x)(f_n(x))^2 dx \right] \\
&\ldots \\
\lambda &= \frac{4}{\pi\left[2a_0^2 + \sum_{n=1}^{N} a_n^2\right]}
\end{aligned}
\tag{5}
$$

This gives the final form of the behaviour definition:

$$
\begin{aligned}
B(\underline{a},r) &= \lambda\left(F(\underline{a},2r-1)\right)^2 \\
&= \frac{4\left(F(\underline{a},2r-1)\right)^2}{\pi\left[2a_0^2 + \sum_{n=1}^{N} a_n^2\right]} \\
&= \frac{4\left[\left(1-(2r-1)^2\right)^{-1/4} \sum_{n=o}^{N} a_n f_n(2r-1)\right]^2}{\pi\left[2a_0^2 + \sum_{n=1}^{N} a_n^2\right]}
\end{aligned}
\tag{6}
$$

The introduction of the normalizing constant λ means that the polynomial weightings are no longer absolute, but relative to each other. This creates an infinite set of weighting vectors that satisfy the solution to any particular problem. In order to return to unique solutions once again, some final constraints are placed on the possible values of the weightings, determined by the inspecting the form of the normalizing constant, λ:

$$
a_0 \geq 0 \quad \text{and} \quad 1 = 2a_0^2 + \sum_{n=1}^{N} a_n^2 \tag{7}
$$

By drawing on some additional mathematics (outlined in the next section) the fitness function that can be achieved by all possible behaviours is presented for the two polynomial case (Figure 7). The infinite sets of solutions can clearly be seen extending radially outwards from (0,0). It is also possible to see the symmetry of the function for positive and negative values. The arc shown on the graph is the result of the application of the eq7 constraints, and indicates how this restricted set still covers all possible behaviours solutions.

4.2 Choosing a behaviour

So far a system has been presented that requires spacecraft to coordinate using simple behaviours (that is a mechanism for partitioning the set of tasks amongst the members of the cluster). The specifics of the problem showed that the definition of a behaviour can be abstracted to become a probability distribution. In order not to limit the flexibility of this distribution it was defined using orthogonal polynomials that provide the ability to control the fidelity of

possible behaviours by choosing the number of representative polynomials. Crucially this fidelity is delivered in a highly compact representation ideal for processing on small, hardware constrained spacecraft.

Although mathematically, higher numbers of polynomials results in a more accurate behaviour, in reality it increases the search space for the genetic algorithm (see section 5). Hence, there comes a point, at roughly 6 polynomials (after 200 generations) that, the detrimental effects of the increased search space override the minor improvements in improved flexibility of higher-order polynomial series (Figure 6, left). Adding polynomials also increases the complexity of the fitness calculations that need to be performed which increases the cpu time per generation, further worsening performance. By using the results of hardware benchmarking, this can be extrapolated to a performance vs time graph (Figure 6, right) that highlights the diminishing returns of higher-order polynomials. The benchmarking was performed on a Leon3 processor using an evaluation board to accurately replicate the available computation power on-board a small (<5kgs) CubeSat spacecraft (Vladimirova *et al.*, 2008) which is a realistic deployment platform for this system.

The next step is to determine a metric that can be used to evaluate various behaviours; and from this metric devise a mechanism to intelligently identify the optimal behaviour. The search strategy for the optimal behaviour is discussed in detail in the remainder of this article. However, the crucial choice of metrics requires a much higher level overall system view, which to a large extent, is beyond the scope of this article (see (Tripp and Palmer, 2008) with more detail to follow in forthcoming publications). In essence however, it derives from the fact that where this article is considering the autonomy required within a single spacecraft, the choice of behaviours is related to the interactions of multiple collaborating spacecrafts working together as a team. Hence, the various behaviour selection strategies within the team have an impact at the collaborative system level, however this is independent of the actual lower-level search strategy for individual behaviours.

As an overview however, each spacecraft has four "characteristics". Various weighting parameters are assigned to each of these characteristics so that the spacecraft behaviour is really a hybrid in the general case. By assigning a higher or lower degree of importance to each characteristic (possibly dynamically) the behaviour of the spacecraft changes; and by choosing different sets of behaviours for the members of the team various levels of coordination and performance can be achieved. The four characteristics are outlined as follows:

- **Greedy**:
  This approach represents the selfish mode. A greedy spacecraft will try to do what is best for itself locally, with no consideration for others. Meaning it will attempt to perform the most valuable tasks currently in the system. This characteristic takes a behaviour, *B*, and the current task map, *T*, as inputs: *G(B,T)*
- **Considerate**:
  This is the feedback mode, where significant emphasis is placed on the result of previous historical performance. The considerate spacecraft will avoid regions of previous overlap (*i.e.* multiple spacecraft redundantly performing the same task). In other words, it considerately performs tasks which no one else is interested in (and hence most likely of low value). This characteristic takes a behaviour, the current task map, and the historical overlap information, *H*, (Figure 1) as inputs: *C(B,T,H)*
- **Proactive**:
  This is the feedforward mode, where emphasis is given to predictions of future behaviour. The proactive spacecraft will attempt to avoid predicted regions of contention/overlap. In a sense this is also considerate but it is proactive rather than reactive. This characteristic takes a behaviour, the current task map, and the future predictions information, *F*, as inputs: *P(B,T,F)*
- **Obstinate**:
  This is the ignorance or inertia mode where the spacecraft makes no use of any environment information and carries on regardless. An obstinate spacecraft will attempt to do exactly what it did in the past regardless. This characteristic takes a behaviour, the current task map and the history of its previous behaviours, B_H, as inputs: $O(B,T,B_H)$

The dynamic goal parameters therefore translate into weightings for the four characteristics Greedy, Considerate, Proactive and Obstinate (g, c, p and o respectively). With suitable normalizations of the characteristic functions the overall fitness/metric of a particular behaviour can be evaluated (algebra as defined above):

$$M(B,B_H,T,H,F,g,c,p,o) = g.G(B,T) + c.C(B,T,H) + p.P(B,T,F) + o.O(B,T,B_H)$$

By setting any of the weightings to 100% a "pure" behaviour can be obtained. However generally, a behaviour is influenced by all the characteristics, usually with one dominating characteristic. How particular behaviour combinations work together, and how those collective behaviour sets can be varied over time for different collaborative performance is beyond the scope of this article.

This function (eq8) depends on the task map, feedforward and feedback environment information as well as the previous behaviour history. As such, it is clear that a particular behaviour can never be considered as good, only good in relation to a particular system state. Hence a behaviour is not a fixed shape but changes dynamically in response to the system so as to maintain its particular characteristics. Unfortunately the introduction of this extra information into the behaviour fitness calculation means that the equations no longer conform to Parseal's thereom (eq2) about orthogonal polynomials and hence a general analytical solution is not feasible. Without an analytical solution, the only remaining avenue is some kind of intelligent search for the optimal behaviour.

5. Searching for a behaviour

The external feedback, feedforward and task information are not under the control of spacecraft (to a first approximation). Equally, the weightings applied to each of the characteristics of the behaviour are assigned by the supervisor/ground station and the history of previous behaviours is a matter of record rather than a variable. Hence the only remaining free parameter for the spacecraft to vary is the behaviour polynomial, which is controlled by the Chebyshev polynomial weightings. Choosing a behaviour effectively means searching this parameter space for the optimal solution[4]. As the parameter space can be large and the evaluation of the fitness function requires significant processing, exhaustive search is immediately ruled out, leaving some kind of heuristic/intelligent search technique.

As the problem is multimodal, the simplest hill climbing search (Pearl, 1984) is not suitable, due to its inability to escape from local optimum. The main difficulty however is the dynamic nature of the problem. New task arrivals, changing environment information, updated goals and recovering from failure all affect how the problem changes over time. Hence techniques such as neural networking that attempt to "learn" solutions to specific static problems are not readily applicable. Equally methods like tabu-search (Glover, 1989) that attempt to restrict revisiting of the search space are fundamentally based on assumptions of a static problem. If the optimal solution shifts to a previously visited area, it will never be found using tabu search. Simulated annealing (Kirkpatrick *et al.*, 1983), although not limited to assumptions of static problems require a fixed "cooling" time with the final solution only available at the end. There would certainly be a need to extend the model to continuous "re-heating" and cooling cycle as the problem changes. It is likely that such extensions could be made to work, although it would seem to work against the fundamental characteristics of the technique and in the author's opinion seriously degrade it's elegance.

The approach taken here is based on the principles of evolutionary computation that has been studied for many years inspired by evolution in the natural world. There are many branches to evolutionary computing such as evolutionary strategies, artificial life and indeed the principles of the stigmergy and emergent self organization that form the basis of this entire spacecraft cluster collaborative architecture. For the purposes of this behaviour search however, genetic algorithms (GA) (Golberg, 1989) seem to provide techniques that can be most readily applied.

5.1 Onboard genetic algorithms

The first major advantage of a GA is that it is a continuous (and interruptible) process. For the real-time constraints that spacecraft systems frequently have, this ensures that the current "best solution" is always available on demand without having to wait for long blocking calls to complex calculations. Having extracted a solution the GA can resume its search for improved solution from the point it was interrupted rather than restarting from the beginning which is a further advantage. Such a system benefits from the fact it can operate continuously as a low priority background process without significant impact to real-time operations. The second major advantage is its natural ability to handle dynamic problems, due to the fact that each fitness function evaluation is effectively independent. Moreover, the system is memoryless (unlike tabu search) so that fitness evaluation has no dependency on the past.

GAs maintain a population of candidate solutions. Each solution is assigned a fitness value according to some objective function (as was outlined above in section 4.2) and a new generation of the population is then created using selection, crossover and mutation operators to splice, mutate and/or copy individuals from the old population; this process is repeated until some termination condition is triggered.

[4] A behaviour that has the highest metric score *i.e.* is the best realisation of the desired goal characteristic weightings.

As Wiegand (Wiegand, 2003) points out, there are essentially two different kinds of selection in GAs, *parent selection* for breeding new candidate solutions and *survival selection* to choose a subset of the population to be carried forward to the next generation. The importance of this distinction is almost never recognized and it is standard practice to perform both functions using a single *Selection* operator. Attempting to do two jobs with one operator results in the enviable trade-off that that neither job is performed optimally, and usually understood in terms of selection pressure (convergence to the best solution) being inversely proportional to population diversity (divergence to the best parents) (Lozano *et al.*, 2004, Ortiz-Boyer *et al.*, 2005). Furthermore, as will be seen in subsequent sections, for problems that are dynamic in nature, the diversity of the population is a crucial parameter. This can be understood by considering a population converging to a solution, which leaves it occupying only a small subset of the problem space. If the problem now transforms to a different fitness landscape, the optimal solution may now lie outside the region occupied by the population. Hence population diversity is crucial for countering the effects of a dynamic problem. For dynamic problems such as the one considered here, is it therefore sensible to explicitly "break apart" the classical genetic algorithms selection operator so that it's "selection" and "diversity" performances can be chosen independently. From here on, *parent selection* (choosing individuals from the population to read and mutate) is referred to as **selection**. *Survival selection* (choosing individuals to form the subsequent generation) will be referred to as **replacement.** In essence, the separation of responsibilities is such that *replacement* seeks to maintain the optimal set of *parents*; *selection* seeks to match parents into breeding pairs optimally; and *mutation/crossover* seek to recombine those breeding pairs into new *children* optimally.

The replacement operator is already a necessary part of a steady state GA (in each generation children are added to the existing population, before it is then pruned), see

Figure 8. In fact it is the explicit use of a replacement operator (such as replace worst, replace parent, DeJong crowding etc.) that defines the difference between steady state and classical GAs. The reason that the steady state GA is not the "standard" GA is likely down to a lack of analysis on the replacement operator. Without the insight of the separation of responsibility between selection and replacement, the implementation of a replacement operator can seem an unnecessary overhead.

5.2 Standard Replacement Strategies

Consider

Figure 8 where a steady state GA has a population of size N. In a single generation the selection, mutation and crossover operators splice and mutate individuals from the population to generate a set of children of size C. These children are added back resulting in an expanded population of size N+C. It is now the replacement operator's task to choose C individuals from expanded population to delete, and once again return the population to size N. Standard approaches to this are as follows (Smith and Vavak, 1999):

- **Random** – Remove C individuals randomly.
- **Truncation** – Remove the C individuals with the lowest fitness scores.
- **Parent** – Remove C parents chosen by the previous selection operator.
- **Oldest** – Remove the C individuals with the longest time since evaluation.

It can be seen that none of these operators explicitly control population diversity, or take any account of the child solution generated by the selection operator.

5.3 Diversity driven replacement

Some operators have been developed to try to maintain population diversity by considering the similarity of the individuals in the population rather than fitness. This similarity/diversity is normally determined by the Hamming distance between the two individuals (*i.e.* the number of bit flips required to transform one individual into the other in the binary representation):

- **DeJong Crowding** – For each child, randomly select a subset of the old population (subset size defined as the crowding factor). Remove the individual from the subset that is the most similar to the child (De Jong, 1975).
- **Contribution to Diversity / Remove Worst (CDRW)** – For each child, consider the subset of the old population with lower fitnesses. Remove the member of this subset with the lowest contribution to diversity (defined as minimum average Hamming distance from all other members of the population). Or, in the case

that the child has the lowest contribution to diversity, simply remove the individual with the lowest fitness (Lozano *et al.*, 2004).

A disadvantage of these two approaches is the additional computation involved in calculating the similarity between two individuals. This is a relatively minor overhead for DeJong Crowding (scales linearly with crowding factor), but the effect is much larger for CDRW which scales with the square of population size. In CDRW every child must compare itself with every individual in the population, and further, every individual in the population must have its contribution to diversity updated every time any individual is added or removed, which is a considerable overhead. Formal benchmarking work on realistic target hardware (not presented here) has shown that CDRW requires nearly 2 orders of magnitude more computation time than the other operators, effectively ruling it out of further consideration.

5.4 Distribution Replacement

Distribution replacement (defined in more detail in previous work (Tripp and Palmer, 2007)) is a mechanism to explicitly control the fitness distribution of the population. Fundamentally distribution replacement controls diversity with respect to the fitnesses of population members, whereas the previous diversity driven replacements control it with respect to genome diversity (*i.e.* in hamming space). Distribution replacement is therefore able to guarantee relative fitness characteristics of the population, something that is not possible with other operators. As search is primarily interested in fitness, intuitively, this mechanism allows the maintenance of "useful" diversity rather than simply diversity per se.

As the global maximum and minimum are unknown, the distribution of fitnesses is relative to maximum and minimum values currently discovered. During the course of the GA run as a range of fitnesses encountered increases, the distribution "stretches" to accommodate this increased diversity. The important point to note is that the fitness profile of the population is explicitly maintained and guaranteed, something that is not true of any other replacement operator. It is possible to select any distribution to conform the population to, for example, a uniform distribution where the probability of a discovered individual being in the population is independent of its fitness. Equally, a highly skewed distribution (that approximates Truncation) is possible where the probability of remaining in the population is strongly correlated to its fitness. For this work, these two distributions and a range of intermediary distributions are considered (Figure 9) chosen by using the highly flexible beta distribution[5].

There are a number of benefits of explicitly controlling the population profile for a dynamic problem. Firstly it is possible to avoid convergence (which effectively reduces the size of the population) and hence easily escape from local minima. Secondly the population always remains "well formed" for the subsequent selection operation, which helps to avoid some pathological cases[6]. Thirdly the shape of the population can be dynamically adjusted to match the characteristics of the problem (if appropriate feedback metrics are introduced).

6. Analytic Markov modelling

To explain the range of problems to which distribution replacement is suited, it is useful to be able to discuss it with in the context of a simplified problem. Furthermore, the argument can be made more powerful by developing an analytic model rather than relying purely on simulation.

A suitable model used here is that of a Markov chain. Markov chains have the property that they are "memoryless", that the behaviour of the system is entirely determined by the current state. In the classic birth/death Markov chain the system state is an integer, n, representing the current number of individuals. Given this value, there is a probability of an individual dying which is a state transition to n-1. Equally there is a probability of a birth where the state transitions to the new state n+1. The equations for these state transitions determine the complete system. With appropriate mathematics, statistical measures of the system can be determined, such as the expected number of individuals and whether or not it is stable.

The genetic algorithm can be modelled as a Markov chain (Nix and Vose, 1992) where the state in this case is the characteristics of the population. Given a particular population the probability of reaching an arbitrary child individual can be determined by considering the operators used in the GA. The system is memoryless in that given the population, the probability of jumping to the new child individual is the same irrespective of how we arrived at this population in the first place.

[5] The beta distribution is controlled by two parameters α and β and hence is written as $\beta(\alpha,\beta)$ where α and β represent the values of alpha and beta respectively. The special case of $\alpha=\beta=1$ $[\beta(1,1)]$ is the uniform distribution.

[6] Such as loss of direction on plateaus of uniform fitness.

The first thing to consider is that individuals are comprised of two dimensions. The first is the genome dimension, g, that is the binary representation of the location of this individual in the problem space. The genome dimension is identical for all problems, and is normally itself comprised of multiple dimensions[7]. For this analysis the genome parameters will be represented as a single scalar value. The second dimension is the fitness dimension, f. The specific problem, defined by the objective function $F(g)$, translates the genome into a fitness value that is to be maximized (along with its inverse):

$$f = F(g)$$
$$g = F^{-1}(f) \quad (9)$$

The mutation and crossover operators work on individuals in the population modifying them in the genome dimension to generate new children individuals. It is therefore possible to define the probability of jumping from any individual to another individual. This is called the transmission probability and is the probability of jumping to g' (or further) given an initial position (individual) g (Smith *et al.*, 2001) :

$$t(g',g) = \Pr(g' > X \mid g) \quad (10)$$

Where: X is a random variable

The transmission probability works in the genome dimension, but it is really the fitness dimension that is of interest. Transforming the dimension is not straightforward, because the fitness function is a many-to-one function (a fundamental characteristic of multimodal problems or even functions with points of inflexion). In order to transform this into a transmission function from a particular fitness, $T(f',f)$, the set of all genomes, g_i, with that particular fitness, G, is defined. From this set it is possible to determine the average transmission probability, $t_{av}(g',f)$ to an arbitrary new individual. These average transmission probabilities can then be integrated to determine the expected transmission probability from a fitness f, to a new fitness, f' (or higher).

$$\forall(g_i,f); F(g_i) = f \Rightarrow g_i \in G$$
$$t_{av}(g',f) = \frac{1}{size(G)} \sum_{g_i}^{g_i \in G} t(g',g_i) \quad (11)$$
$$T(f',f) = \int t_{av}(g,f) \frac{df}{dg} dg$$

Given an individual with a particular fitness, this function gives the probability of jumping to a new fitness. The genetic algorithm however, maintains a population of individuals. Distribution replacement has the explicit property that the fitness distribution of the population always remains constant, relative to the current maximum individual. Hence the characteristics of population, P, are entirely described by a single value, m, that is the maximum fitness of any individual in the population. In this work, the general beta distribution is used, $I_f(\alpha,\beta)$ - where α and β are free parameters controlling the shape of the distribution. This gives the probability of an individual with fitness, f, being in the population as:

$$P(f,m) = \frac{f.I_f(\alpha,\beta)}{m} \quad (12)$$

By integrating the transmission function over the members of the population it is possible to define the probability of the maximum value in the population jumping to a particular fitness (or better). Integrating over the population needs to be performed with respect to a weighting function that represents the probability of a particular individual being selected by the selection operator. Three common selection operators (that give the probability of a particular individual in the population being selected) can be defined based on this definition:

Random	$S(f,m) = P(f,m)$ (13)

[7] *i.e.* one dimension for each of the function parameters that are to be optimized

Rank	$S(f,m) = \frac{\int_0^f P(f,m)df}{\int_0^m P(f,m)df}$
Roulette Wheel	$S(f,m) = \frac{\int_0^f f.\frac{d(P(f,m))}{df}df}{\int_0^m f.\frac{d(P(f,m))}{df}df}$

It is now possible to construct a function that defines the probability distribution of a potential child solution based on the population. This is called *C(f',m)*, and is the probability of a child of fitness *f'* (or higher) being generated given the current best individual, *m*, the selection operator, the replacement operator and the transmission function probability (mutation & crossover operators):

$$C(f',m) = \int S(f,m).\left(\frac{d(T(f',f))}{df}\right)df \qquad (14)$$

This definition uses the current maximum individual (at generation *n*) in the population to determine the probabilities of generating a new child member of arbitrary fitness. It is now possible to use this to calculate the probability of obtaining a new maximum value after one generation (one child per generation). As the population never loses the current best individual (elitism), children with a fitness lower than the current maximum, means the next maximum remains unchanged:

$$m_{n+1} = \frac{\int_0^{m_n} m_n.N(f,m_n)df + \int_{m_n}^{\infty} f.N(f,m_n)df}{\int_{m_n}^{\infty} N(f,m_n)df} \qquad (15)$$

Where :

$$N(f,m_n) = \frac{d(C(f,m_n))}{df}$$

This formula can now form the basis of a continuous Markov chain as it depends solely on the "memoryless" value of maximum, *m*, (using fixed selection, replacement and mutation/crossover operators).

6.1 Simplified 1D random walk problem

The Markov chain can now be put to work on a purely mathematical fitness function that has characteristics similar to the real behaviour choice problem that is of interest. This simplified problem is defined as follows:

$$f(g,a) = \begin{cases} g/a & \text{if } g \le a \\ \frac{1-g}{1-a} & \text{if } g > a \end{cases} \qquad (16)$$

This represents a simple linear unimodal peak with the maximum value location defined by parameter *a* (see Figure 12). In order to make this problem dynamic, the value of *a* is altered so that the peak location continuously jumps to different values. The location of the peak can be considered as a bounded (reflected) random walk in the range [0,1]. The distance (and direction) of peak movement is defined to be a normally distributed random variable centred at $\mu=0$ and with the standard deviation, σ, defining how chaotically the peak jumps.

6.2 Effect of Replacement on Performance

Putting it all together it is now possible to generate empirical performance measures. A suitable mutation rate (eq10) is defined by modelling the output of the real genetic algorithm mutation operator. The random selection operator (eq13.1) is used for the following results (although results are similar for the others). The Markov chain is also "discretized" for runtime implementation purposes. The results of Figure 10 show the performance of several different distribution replacement operators (previously defined in Figure 9) against a varying random walk standard deviation and number of generations between each jump (*i.e.* how rapidly it changes). As this is a simple problem, explicit values of performance, number of generations etc are not of significant interest. Rather it is the relative difference in performance that indicates the strength or weakness of the operators for various parameter combinations.

The explanation for Truncation (actually it's approximation of the β(10,1) distribution) replacement's performance is reasonably intuitive - Figure 10 (top left). As more time (generations) passes, Truncation's performance improves, as is the case for all properly designed genetic algorithms. Equally, performance improves as the standard deviation of the random walk drops. Hence for a static problem (stdev = 0) Truncation does well, exactly the situation that it is has been designed for. As the standard deviation increases however, and the peak jumps around more significantly, Truncation falls into the loss of diversity trap.

The loss of diversity trap can be explained by considering Figure 12. Take a single interval during which the problem remains static, represented by the (solid) red peak. Truncation replacement quickly "hill climbs" to the top of this peak pruning away the lowest individuals from the population. Given enough time the entire population is likely to converge to this single point (Thierens and Goldberg, 1994). Now, when the problem random walks (jumps) to the second, (dashed) blue peak, the entire population finds itself far away from the new global optimum. Hence, as the standard deviation of the peak jump distance increases, so does the probability that the converged population will find itself further away from the new peak after each jump. As it takes time (generations) to move up towards the new peak, it can be seen that the average probability of being at the global maximum rapidly falls off as the jump distance between subsequent peaks increases. Hence, although Truncation is good at finding instantaneous peaks, it does so at the cost of loss of diversity. This loss of diversity means that on the subsequent jump, it finds itself in the wrong place, and so on average over all the jumps, its probability of being at the global maximum is reduced.

Uniform distribution replacement Figure 10 (bottom left) has the opposite property to Truncation. Uniform distribution maintains maximal (fitness) diversity at all times so its population is spread right from the peak to the base. Hence, after each jump the probability of one of the members of the population being close to the new peak is much higher. The downside is that if the problem remains static, maintaining this distribution reduces the selection pressure of the genetic algorithm. Or in other words it reduces the rate at which the population climbs the hill. This explains the significant drop in performance, at the front of the graph, on the static problem (stdev = 0). This is exactly the opposite argument to Truncation. Uniform distribution has sacrificed its ability to find the instantaneous best solution, but it's maximal diversity means that on average it always finds itself close to the global maximum after each peak jump.

What is especially interesting, is that this improved performance of distribution replacement requires only a small average jump distance to really pay off. Only when the problem becomes highly static does Truncation become a more successful replacement strategy. This indicates that only a small amount dynamism is needed in an optimization problem before it is beneficial to use distribution replacement over Truncation.

The graphs on the right-hand side of Figure 10 serve to highlight how this is not a binary choice but a continuous spectrum. Distribution replacement can be tuned to include more or less diversity, and inversely less or more selection pressure (as can be seen by reading round clockwise as the distribution transforms from Truncation to Uniform through the intermediate steps). It is even possible to see how this diversity profile can be optimized for different situations. For example replacement β(1.3,1) (bottom right) has maximal performance at roughly a standard deviation of 0.2. This is when the average jump distance becomes close to the mean distance from the global maximum of the distribution. *I.e.* the population is concentrated exactly where the peak is likely to jump to next. This fact serves to emphasize how distribution replacement is a tunable operator and can be subtly adjusted based on the characteristics of the problem, something no other replacement operator can provide.

7. Simulation Results

For more realistic simulations spacecraft perform tasks within the system, using the behaviour to guide their selection. Every task entering the system has an associated resource requirement. The number of tasks at each resource value forms a distribution in resource space, referred to as the task map. For implementation, the resource space is discretized into (100) bins and for simplicity of initial analysis only a single resource dimension is used. This resource can be considered as a user priority for particular tasks. However, this could equally be considered as the task power

requirements or perhaps the storage capacity in megabytes needed for various Earth observation images. For simulation purposes, a stochastic arrival pattern is defined for new tasks entering the system. These simulations consider resource as priority, so as a rough approximation to real satellite operations there is a skewed distribution of priorities so that there are larger numbers of low priority (*i.e.* background) tasks that continuously arrive (in fact a β(2,1) distribution). This is maintained to ensure that spacecraft are never idle, which is a significant waste of time and money. There is also a fixed set of repeated medium priority tasks that represents station keeping and other routine operations. As one of the fixed tasks is completed, it is immediately re-added into the system. Occasionally there is a burst of high priority tasks in response to events that are of interest to the satellite operators (such as volcano activity). All tasks also need to be done in reasonable time. To ensure this and avoid task starvation, whilst tasks remain in the system, their priority is slowly increased, so that the task map appears to drift towards higher priority. There are a finite number of tasks that the cluster can maintain. As tasks are completed and removed there is a periodic broadcast by the ground station to "top up" the system with a new set of tasks to be added, based on the arrival pattern (which is called a task update). As such, over time, the taskmap falls into steady-state (in probabilistic terms, although in practice it is highly noisy). This "steady-state" is further disturbed by changes in spacecraft behaviour or bursts of tasks etc.

The simulation runs for 100 task updates (up to 15% of the tasks completed in between updates). Each spacecraft is allowed 50 generations to choose its behaviour each update (250 fitness function evaluations). This limited amount of search time clearly means that the evolved behaviour is highly unlikely to be the instantaneous optimum. The effect of using these sub-optimal "transient" behaviours is to introduce additional dynamic noise into the problem. Although it is important to attempt to reduce this noise (by improving the performance of genetic algorithm operators), it is only a small contribution to the overall dynamism of the problem. This is because fundamentally the behaviour only provides a probabilistic ordering rather than a deterministic one anyway. There may be underlying scheduling and operational constraints in addition to simple task failures that affect which tasks are actually performed (see section 3). Additionally, the behaviour is intended to be continuously evolved in the background so that a particular instantaneous behaviour will only be used to select a small number of tasks. By the time that tasks that are of lesser significance to the spacecraft come to be performed, the underlying environment will have changed and the behaviour will have further opportunities to evolve. Hence, it is not fruitful to spend large amounts of time tweaking the parts of the behaviour covering tasks of lower significance, as this additional refinement optimization will add little if any benefit in practical terms. What is important is to choose operators that can rapidly evolve the behaviour towards a near optimal solution even if the performance in later stages subsequently plateaus.

7.1 Distribution Replacement

The simplified problem helps to explain the difference in performance of the different replacement operators. This result is also evident in the actual behaviour choice problem with the performance of different distribution replacement operators (those outlined in Figure 9) shown as the percentage differences between the performance of the specific operator and the Truncation operator, averaged over all 100 updates, shown with 95% confidence error bars. The results discussed below are for a single spacecraft attempting to be as greedy as possible (see section 4.2) in a cluster with four other spacecraft all of whom have a range of different behaviour characteristics. The real scenario results (Figure 11) match those of the analytical model (Figure 10), which show (for this highly dynamic problem) optimal performance for a uniform distribution, β(1,1). Performance also drops off smoothly as the distribution becomes a better and better approximation to Truncation, β(10,1), exactly as was seen in the Markov analysis.

Figure 11 is a specific (although representative) instance of a simulation configuration as outlined above. Extensive coverage of different spacecraft cluster configurations is beyond the scope of this article but this relative performance is consistent across different numbers of generations, different goals and different task maps. For example, this can be seen in Figure 13 where the number of tasks completed by each spacecraft per update is varied (which is a proxy for varying the communications bandwidth). This effectively changes the amount of dynamism in the problem as the task map becomes less dynamic (which corresponds to the standard deviation axis in Figure 10). However, as can be seen from the graph Uniform distribution replacement still outperforms Truncation across the board. This is due to the fact that having five spacecraft in the system still represents a large amount of noise as they attempt to avoid each other, for example. This graph would seem to imply that standard Truncation would begin to be effective only in situations with only two (perhaps three) spacecraft completing a small number of tasks per update (*i.e.* high communication bandwidth). Such a situation would effectively be ideally suited to the current approach of direct remote control (and hence represents the current state of spacecraft systems rather than future systems as this work addresses). It might appear from the graph that improvement drops when large numbers (30%) of tasks are completed per update. Care must be taken when interpreting the graph as the performance is relative to the Truncation operator. Hence, for the different types of problem the absolute value of attainable fitness changes and the reduction of this absolute value at the higher

end, feeds through into a reduced relative performance improvement. It is therefore the changing characteristics of the fitness function rather than variation in operator performance that begins to have an effect at the higher end.

Figure 14 also shows that this effect is not limited to a carefully chosen number of generations. In this case the number of tasks completed per update is once again set to 15%. The spacecraft is now given a variable number of generations in between updates in order to select the best behaviour (which corresponds to the generation axis in Figure 10). This can be considered as simple model for improving the power of the on-board processing hardware. Again, it can be seen that uniform distribution replacement outperforms standard Truncation across the feasible range of generations per update. As the number of generations increase, the selection pressure effect begins to outweigh the diversity and relative performance starts to drop off as Truncation catches up. Of course, Truncation still gets trapped in the local maxima which reduces the impact of this effect. Fundamentally though this is an engineering problem with tight constraints on available computation time. 500 generations therefore represents a top bound on feasible capability (with 50-100 generations a more realistic scenario), and so Truncation can never ever get the 1000s of generations it needs to become the operator of choice. What can also be seen from this graph is the plateauing of the operator's performance (as discussed above in section 7). Increasing the number of generations helps to enhance the performance in the early stages, but as more and more generations are allowed, the later stages of the search turn into minor refinement. This refinement is in many ways rendered meaningless by the dynamic nature of the problem and so performance plateaus.

7.2 Other Operators

Up to this point only distribution replacement and Truncation operators have been considered. Figure 15 now compares some other standard replacement operators for the same problem as well as a pure implementation of a classic genetic algorithm (that has no explicit replacement operator). Uniform distribution is clearly the best performer. Of the rest, replace the oldest performance well, as would be expected of an operator that has limited "memory". It's low inertia allows it to rapidly respond to the dynamic problem. CDRW also performs well as it is effectively an extension to Truncation that adds diversity if improved fitness's cannot be found. Of course CDRW is an extraordinarily time-consuming operator that would not be suitable for implementation on a small spacecraft.

The performances of the DeJong Crowding (irrespective of the crowding factor chosen) and the Random replacement operator is surprisingly low. The explanation can be found by looking at the fact that although both are good at maintaining diversity, this is not guaranteed or necessarily useful. Generally the diversity is high, but it has highly variability as it is not being explicitly controlled (unlike distribution replacement). This effect is illustrated in Figure 16 with Uniform distribution along with DeJong Crowding and Random replacement analyzed in a single update (an instantaneous static problem). Each GA is run for 1000 generations after the global optimum has been found[8]. In each case, the fitness distribution of the population is averaged across the 1000 generations and plotted along with its standard deviation as error bars. Uniform distribution comes out as a diagonal line conforming exactly to its CDF. Moreover, the standard deviation of this distribution is almost invisible on the graph. The population explicitly maintains a uniform and maximal distribution of fitness's without wavering as it has been designed to do. Random and the DeJong Crowding by contrast have an average fitness profile that roughly corresponds to a Gaussian distribution CDF. This is the central limit theorem at work, sometimes the population is skewed one way, sometimes the other way but on average it mainly maintains mediocre mid-range individuals. This randomly varying skew is an effective strategy for escaping from local minima in a static multimodal problem. However in a dynamic problem, the added dynamism of the population distribution simply means that the probability of fortuitous alignment is further reduced. Uniform distribution replacement by contrast has a uniform probability irrespective of where the dynamic problem jumps to. This is the crucial difference between diversity and useful diversity, as distribution replacement controls diversity based on the fitness (rather than hamming distance like the others) it inherently includes more information about the problem space.

A further point to note is that whereas Uniform distribution maintains individuals with even the minimal value, the other two operators have a very low probability of including these individuals in their population. This is the effect of the selection operator working against their diversity maintenance capabilities and hence reducing the diversity of the population, which in turn reduces their performance on the dynamic problem further.

There is also a third factor contributing to their lower performance. At each update, the dynamic problems becomes a static optimization problem. At this point-high selection pressure is desirable, such as can be found with the Truncation and CDRW operators. However, DeJong Crowding and Random, with their Gaussian bell shaped population distribution, are effectively applying selection pressure to maintain mediocre individuals. Uniform distribution

[8] As no further progress can be made the GA can now be considered to be in a steady-state condition

replacement by contrast not only has a better diversity, but also a larger number of higher fitness individuals in the population, which allows the selection operator to work more effectively *i.e.* a GA with a higher selection pressure.

In effect, the Truncation operator loses out on the dynamic problem (due to its lack of diversity), but nevertheless it's improved selection pressure during the static period recovers some of the ground. The stochastic diversity driven replacement operators miss out during the static period as they have the minimum selection pressure of all the operators.

Returning to Figure 15 the right-hand side shows three classical genetic algorithms using the Roulette Wheel (proportional), Rank and Random selection operators. These have been matched for number of fitness function evaluations along with crossover/mutation rates for a fair comparison with the other steady state genetic algorithms. The classic genetic algorithms are far worse at maintaining diversity - especially Roulette Wheel which is an extreme case of Truncation with the inherent massive loss of diversity. A classic Roulette wheel genetic algorithm has sacrificed performance in subsequent updates which outweighs any benefit of improvement in a single update. Rank selection has a lower selection pressure (and hence more diversity) and so performs much better. This in turn is followed by random selection (basic random search) which has the lowest selection pressure of all. Comparing the random classical GA with the random replacement operator demonstrates the impact of the selection and replacement operators working together. The classical Random GA is effectively random search. However, the steady state genetic algorithm with the Random replacement operator (and Rank selection operator working in tandem) has significantly worse performance and slightly lower diversity. This is due to the fact that they are effectively working against each other with the selection operator reducing the diversity of the population.

Although not shown in this article's results, a random restart strategy was also considered. In such a scheme the population (or a subset) is reinitialized after each problem jump. This reinitialization can either be to random locations in the problem space or to defined points. In principle, uniform random restart, with a replacement operator such as Truncation performs similarly to that of Random replacement. The problem is that although the restart can be controlled to be uniformly distributed in the diversity of its population. It cannot be controlled to be uniformly distributed in its fitness distribution (as that requires knowledge of the problem). Hence, restart is never able to control diversity based on the fitness, and faces similar problems to the other diversity driven replacement operators. Restart also falls down in situations where the problem only jumps a small distance, so that by chance it is similar to the previous update. In this case, distribution replacement can capitalize on the work it has done in the previous update, and spend its time "refining" rather than searching. Restart on the other hand always loses out in this case, and so overall its performance is reduced.

8. Verification

Nanosatellite cluster missions that are prohibitively expensive to deploy (for academic research purposes), hence in orbit verification has clearly not been possible. However it has been possible to benchmark the performance of the algorithms on realistic flight hardware to gain confidence in the feasibility of the proposed approach. Specific details of the simulations and evaluation is beyond the scope of this article (to appear in forthcoming publications). However, in summary the algorithm was benchmarked on the LEON3 microprocessor running at 40MHz[9] under the RTEMS real-time operating system (RTOS).

The widely available LEON3 microprocessor has existing space heritage as part of the Venus Express imaging payload controller and has been adopted by ESA as the main CPU for future on-board computers (Fiethe *et al.*, 2007). It incorporates multiple processor usage, fast AMBA2 AHB bus, 8/16/32-bit memory controller, 16-bit I/O and many peripherals and can run at up to 200 MHz, consuming around 250 mW. This processor is entirely suitable for implementation on the familiar CubeSat nanosatellite platform (Vladimirova *et al.*, 2007). A further advantage being that this processor is commercially available technology and hence is economic well suited for implementation on budget constrained future nanosatellite cluster missions.

The summary of the benchmarking tests showed that for realistic scenarios (100 generations, 5 children per generation and population of 50, evaluated with chebyshev n=6 and 100 evaluation points using a beta distribution replacement and roulette wheel selection), gave a total evaluation time of around 15 minutes. To put that into context, a standard LEO orbit is of the order of 100 minutes (*i.e.* ~15% of a single orbit time). Given this system is intended to perform as a low priority background process over several days (*i.e.* 10s of orbits), this overhead is clearly within acceptable bounds. Moreover it must be noted that the benchmarking experiments were performed using unoptimized

[9] Well below the maximum speed for this processor, but selected to be in line with likely power budgets of the limited CubeSat platform.

"research simulation code", rather than highly optimized flight software code that would be used in practice, and hence the actual likely performance would be significantly faster than indicated.

At the system level, verification of the success of the behavioural approach to the partitioning of tasks among a spacecraft cluster has previously been reported (Tripp and Palmer, 2008) with further results to appear in forthcoming publications.

9. Conclusions

This article has looked at issues surrounding the autonomous decision-making process that an individual worker spacecraft is required to perform. When collectively working together these spacecraft form an emergent system based on local decisions. These local decisions, characterized as the behaviour of the spacecraft, provide a way to prioritize a subset of tasks to be performed locally. This behaviour was defined mathematically so as to maximize generality within a compact representation. Such a manageable search space is essential to allow intelligent search techniques the best possible chance of success, within the highly constraint and limited processing capabilities of miniature spacecraft.

A genetic algorithm was selected as a suitable search technique. As the system requires continuous optimization of dynamic variables, the choices of operators and parameters must be adapted from the standard static optimization approach. Investigation of these issues was informed by the development of an analytic Markov model. This model was applied to a simplified representative problem, demonstrating that with such dynamic problems, maintaining population diversity is of fundamental importance.

Distribution replacement was introduced and found to be more effective than other replacement operators for this problem. This was shown across a range of different problem variables that affected various characteristics of noise and available processing power. Distribution replacement is successful because it explicitly controls the population so that it has guaranteed diversity. This is in contrast to other operators whose behaviour is only guaranteed in the probabilistic sense. The second major factor in its improved performance is that it controls diversity in fitness space rather than genome space unlike the other diversity operators. This effectively means its diversity contains more information about the problem and hence is more useful for the search process. The third major benefit of distribution replacement is that for this problem with highly constrained processing limits an operator is needed that can find a good solution quickly each update. Moreover the dynamic nature of the problem means that the differences between the performance of an optimal and near optimal solutions is almost unnoticeable. Hence by way of analogy what is needed is a "hare" operator rather than a tortoise. Although the tortoise may win over the whole race, at the half-way point the hare is in the lead – and for this case performance at half-way (*i.e.* the limited cpu time) is what really matters.

Performance has been investigated for a particular engineering application of the onboard autonomy of nanospacecraft. However, the results presented here are more general to continuous optimization tracking of dynamic problems. Suitable problems are ones where the instantaneous solution is relatively straightforward to find (and hence would result in rapid convergence), but in subsequent updates the particular optimum point shifts randomly or chaotically. This approach is also only suitable in situations where the processing power (or time) available between updates is limited and not where more "brute force" approaches would become viable.

10. Acknowledgements

This work was entirely funded by the Autonomy group of the Surrey Space Centre, part of the University of Surrey, Guildford, UK.

11. References

BONABEAU, E., DORIGO, M. & THERAULAZ, G. (1999) *Swarm Intelligence: From Natural to Artificial Systems*, Oxford University Press Inc, USA.

BONNET, G. & TESSIER, C. (2007) Collaboration among a satellite swarm. *6th international joint conference on Autonomous agents and multiagent systems*. Honolulu, Hawaii, ACM New York, NY, USA.

BROWN, O. & EREMENKO, P. (2006) Fractionated Space Architectures: A Vision for Responsive Space. *4th Responsive Space Conference*. Los Angeles, CA.

CARREL, A. (2007) Adaptive Evolutionary Decision Making for Autonomous Spacecraft. *Surrey Space Centre*. Guildford, University of Surrey.

CARREL, A. R. & PALMER, P. L. (2005) An evolutionary algorithm for near-optimal autonomous resource management. *8th International Symposium on Artificial Intelligence and Automation in Space.* Munich.

CHIEN, S., SHERWOOD, R., TRAN, D., CICHY, B., RABIDEAU, G., CASTANO, R., DAVIES, A., LEE, R., MANDL, D., FRYE, S., TROUT, B., HENGEMIHLE, J., D'AGOSTINO, J., SHULMAN, S., UNGAR, S., BRAKKE, T., BOYER, D., VANGAASBECK, J., GREELEY, R., DOGGETT, T., BAKER, V., DOHM, J. & IP, F. (2004) The EO-1 Autonomous Science Agent. *Autonomous Agents and Multi-Agent Systems.* New York City, USA.

CLEMENT, B. & BARRETT, A. (2002) Coordination Challenges for Autonomous Spacecraft *International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2002).* Bologna, Italy.

CLEMENT, B., BARRETT, A. & SCHAFFER, S. (2004) Argumentation for Coordinating Shared Activities. *Proc. IWPSS*.

CURTIS, S. A., RILEE, M. L., CLARK, P. E. & MARR, G. C. (2003) Use of swarm intelligence in spacecraft constellations for the resource exploration of the asteroid belt. *3rd International Workshop on Satelllite Constellations and Formation Flying.* Pisa, Italy.

DE JONG, K. A. (1975) Analysis of the behavior of a class of genetic adaptive systems.

FIETHE, B., MICHALIK, H., DIERKER, C., OSTERLOH, B. & ZHOU, G. (2007) Reconfigurable system-on-chip data processing units for space imaging instruments. *Design, automation and test.* Europe, EDA Consortium San Jose, CA, USA.

GAUTSCHI, W. (2004) *Orthogonal Polynomials: Computation and Approximation*, Oxford University Press.

GLOVER, F. (1989) Tabu search -- Part I. *ORSA Journal on Computing,* 1**,** 190-206.

GOLBERG, D. E. (1989) *Genetic algorithms in search, optimization, and machine learning,* Addison-Wesley.

GRASSE, P. P. (1960) The automatic regulations of collective behavior of social insect and" stigmergy". *J Psychol Norm Pathol (Paris),* 57**,** 1-10.

HINCHEY, M. G., RASH, J. L., TRUSZKOWSKI, W. F., ROUFF, C. A. & STERRITT, R. (2005a) Autonomous and Autonomic Swarms. *The International Conference on Software Engineering Research and Practice (SERP 05).* Las Vegas, Nevada, USA, CSREA Press.

HINCHEY, M. G., RASH, J. L., TRUSZKOWSKI, W. F., ROUFF, C. A. & STERRITT, R. (2005b) Challenges of Developing New Classes of NASA Self-Managing Missions. *Parallel and Distributed Systems, 2005. Proceedings. 11th International Conference on,* 2.

KARUNA, H., PAUL, V., CONSTANTIN, Z., HENDRIK VAN, B., BART SAINT, G., TOM, H. & ELKE, S. (2004) *Self-Organising in Multi-agent Coordination and Control Using Stigmergy.*

KIRKPATRICK, S., GELATT JR, C. D. & VECCHI, M. P. (1983) Optimisation by simulated annealing. *Science,* 220**,** 671-680.

LOZANO, M., HERRERA, F. & CANO, J. R. (2004) Replacement strategies to preserve useful diversity in steady-state genetic algorithms. *Press, March.*

MASON, J. C. & HANDSCOMB, D. C. (2003) *Chebyshev Polynomials*, Chapman & Hall/CRC.

MUSCETTOLA, N., FRY, C., RAJAN, K., SMITH, B., CHIEN, S., RABIDEAU, G., YAN, D., DIV, C. S., CENTER, N. A. R. & FIELD, M. (1997) On-board planning for New Millennium Deep Space One autonomy. *IEEE Aerospace Conference.* Snowmass, Colorado.

NIX, A. E. & VOSE, M. D. (1992) Modeling genetic algorithms with Markov chains. *Annals of Mathematics and Artificial Intelligence,* 5**,** 79-88.

ORTIZ-BOYER, D., HERVAS-MARTINEZ, C. & GARCIA-PEDRAJAS, N. (2005) CIXL2: A Crossover Operator for Evolutionary Algorithms Based on Population Features. *Journal of Artificial Intelligence Research,* 24**,** 1-48.

PARUNAK, H. V. D., PURCELL, M. & CONNELL, R. O. (2002) Digital pheromones for Autonomous Coordination of swarming UAV's. *1st AIAA Technical Conference and Workshop on Unmanned Aerospace Vehicles, Systems, and Operations.*

PEARL, J. (1984) *Heuristics: Intelligence strategies for computer problem solving*, Addison-Wesley.

RICHARDS, R. A., HOULETTE, R. T. & MOHAMMED, J. L. (2001) Distributed Satellite Constellation Planning and Scheduling. *14th International Artificial Intelligence Research Society Conference*. Florida.

SCHETTER, T. P., CAMPBELL, M. E. & SURKA, D. M. (2000) Comparison of Multiple Agent-Based Organizations for Satellite Constellations (TechSat21). *LAIRS Conference*.

SMITH, J. & VAVAK, F. (1999) Replacement Strategies in Steady State Genetic Algorithms: Static Environments. *Foundations of Genetic Algorithms 5*.

SMITH, T., HUSBANDS, P. & O'SHEA, M. (2001) Characterising Fitness Landscapes Through Evolvability. *Cognitive Science Research Article*, 534.

THIERENS, D. & GOLDBERG, D. (1994) Convergence Models of Genetic Algorithm Selection Schemes. *Lecture notes in computer science*, 119-119.

TRIPP, H. & PALMER, P. (2007) Distribution replacement: how survival of the worst can out perform survival of the fittest. *9th annual conference on Genetic and evolutionary computation*. London, England, ACM.

TRIPP, H. & PALMER, P. (2008) Distributed Behavioural Coordination for Satellite Clusters. *International Astronautical Congress 2008*. Glasgow, UK.

VLADIMIROVA, T., WU, X. & BRIDGES, C. P. (2008) Development of a Satellite Sensor Network for Future Space Missions. *IEEE Aerospace Conference*.

VLADIMIROVA, T., WU, X., JALLAD, A. H. & BRIDGES, C. P. (2007) Distributed Computing in Reconfigurable Picosatellite Networks. *2nd NASA/ESA conference on Adaptive Hardware and Systems*.

WHITE, T. (2005) Expert Assessment of Stigmergy: A Report for the Department of National Defence. School of Computer Science, Carleton University, Canada.

WIEGAND, R. P. (2003) An Analysis of Cooperative Coevolutionary Algorithms. George Mason University.

ZETOCHA, P. (2000) Satellite cluster command and control. *IEEE Aerospace Conference, Big Sky MT*.

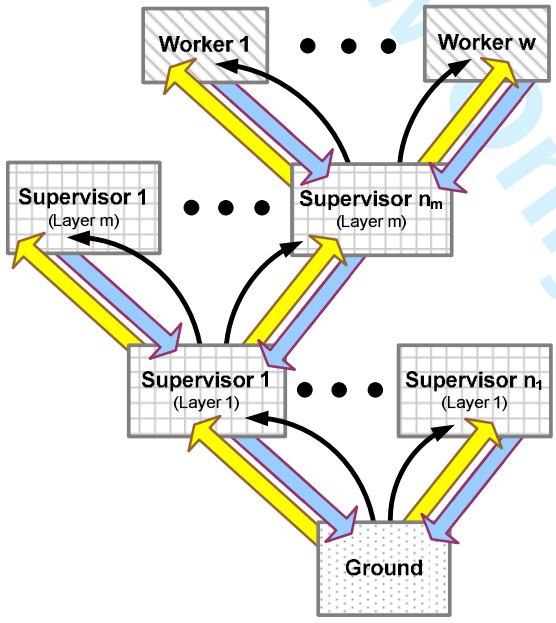

Figure 1: Schematic of the information flow in the hierarchical stigmergy task allocation architecture.

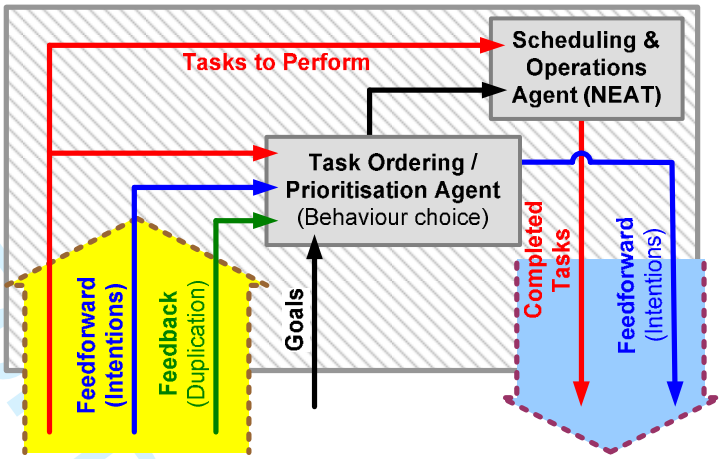

Figure 2: Information flow and main subcomponents of a Worker spacecraft

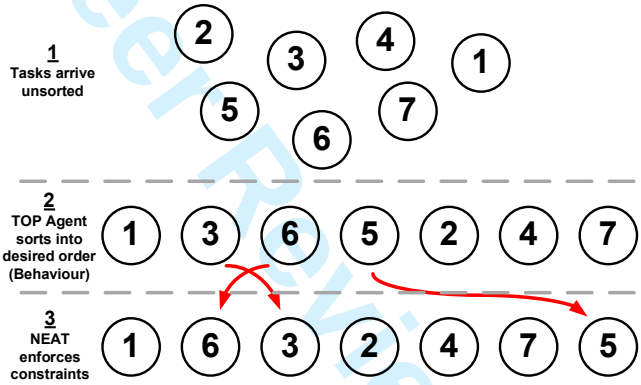

Figure 3: An overview of the tasks scheduling process on-board A Worker spacecraft.

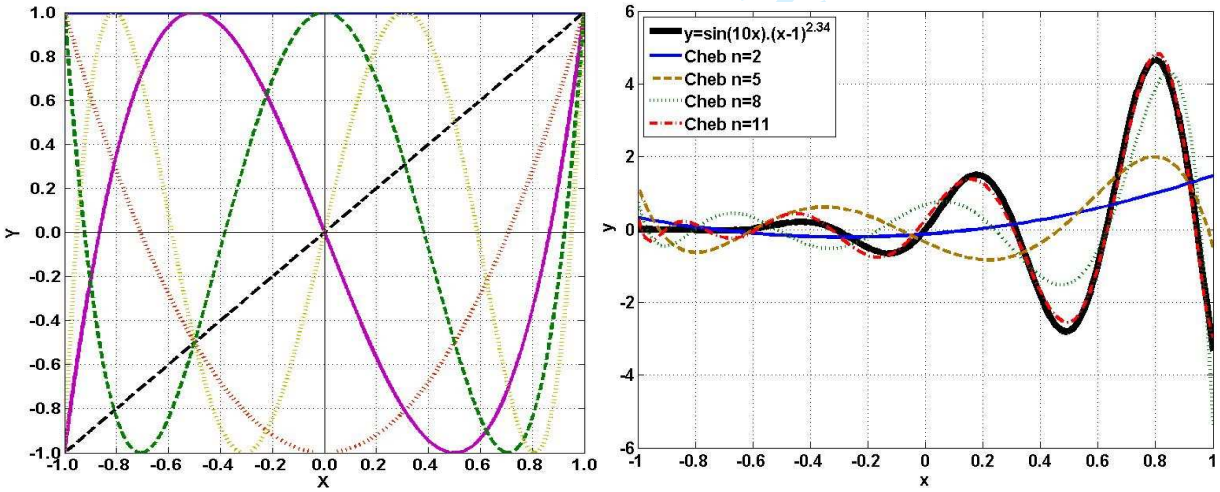

Figure 4: Left, the first six polynomials in the Chebyshev sequence. Right, the best match to a particular mathematical function using an appropriately weighted sequences of 3, 6, 9 and 12 polynomials (sequence starts at n_0 with "n=" being index of the highest order polynomial)

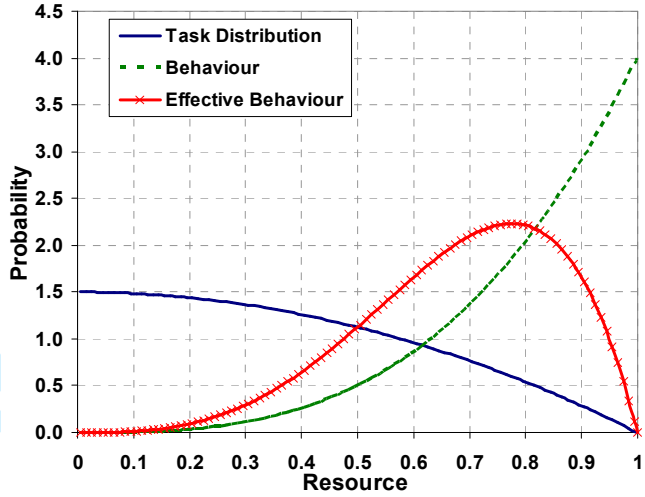

Figure 5: An example Task distribution and Behaviour, multiplied together to get a true effective probability of performing a task at a particular resource value.

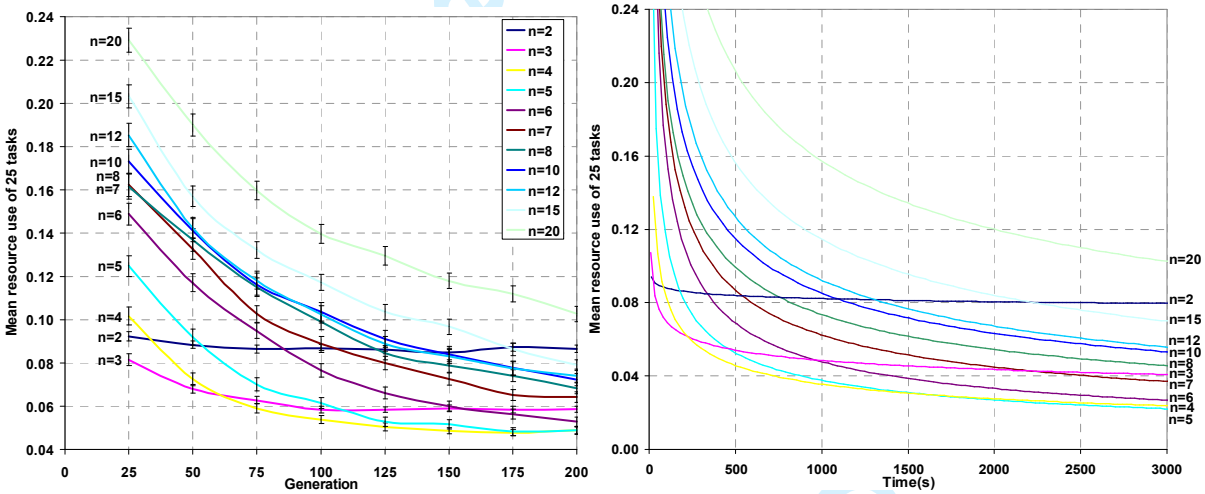

Figure 6: The performance (minimal resource use) achieved by a genetic algorithm for a given number of generations (left). This is extrapolated to performance vs time (right)

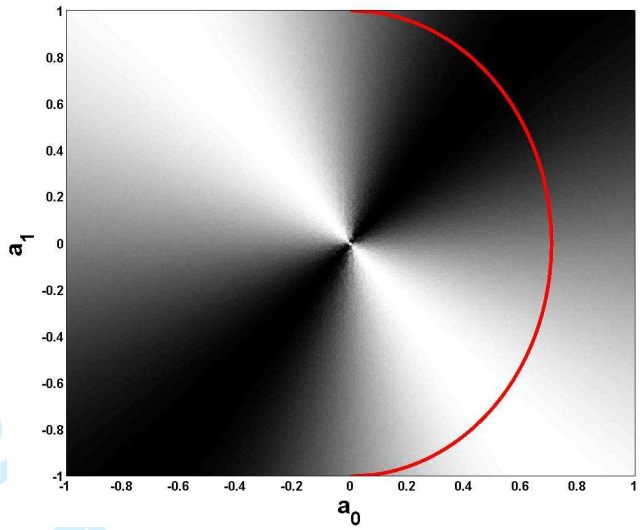

Figure 7: The fitness landscape for a two polynomial function (100% efficiency and uniform task map), with the restricted set of possible weightings shown as a continuous arc. Lighter areas indicate a higher fitness.

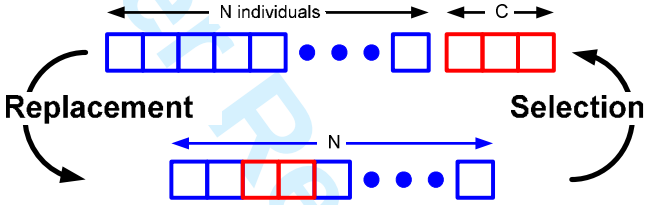

Figure 8: The Replacement-Selection loop

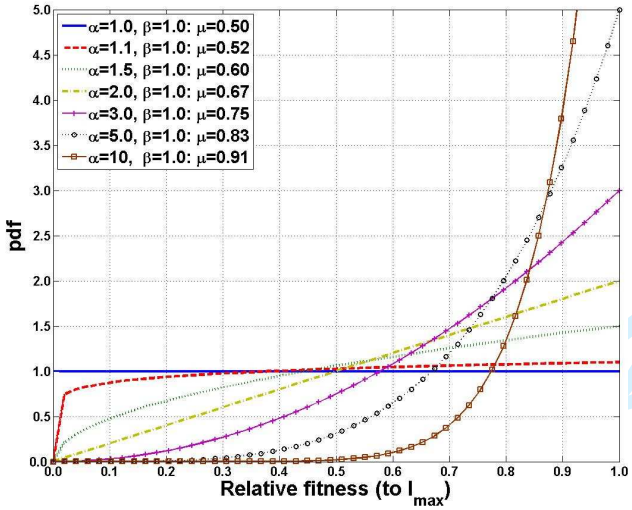

Figure 9: Population distributions used with distribution replacement in this article.

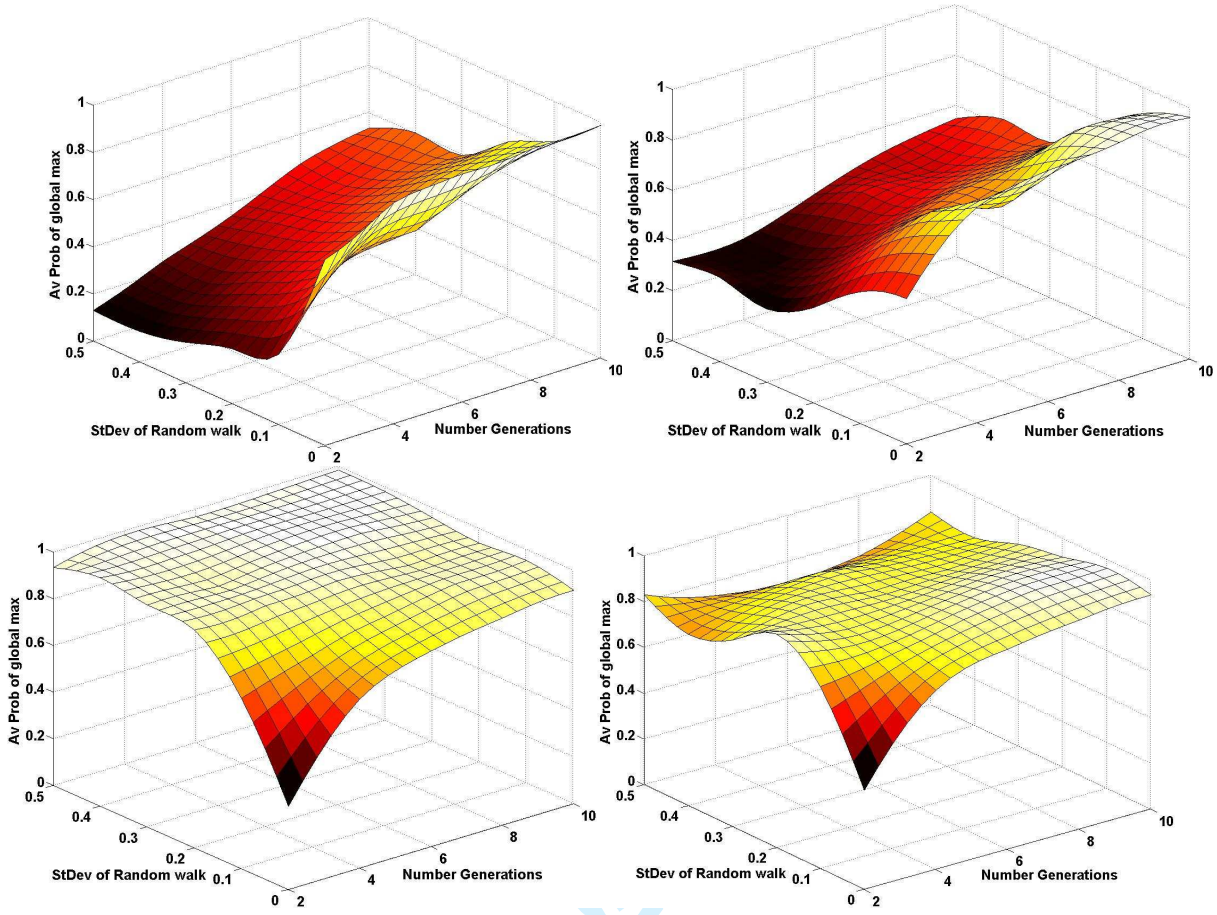

Figure 10: Performance of several distribution operators on the simplified 1D random walk problem. The vertical axis is the probability of being at the global maximum, after a certain number of generations (x-axis) for a specific random walk jump distance standard deviation (y-axis). The top left graph is for β(10,1) distribution replacement (a model for Truncation). The distributions then become closer to uniform in a clockwise direction. Hence top right is β(2,1), bottom right β(1.3,1) and bottom left β(1,1) – the uniform distribution itself.

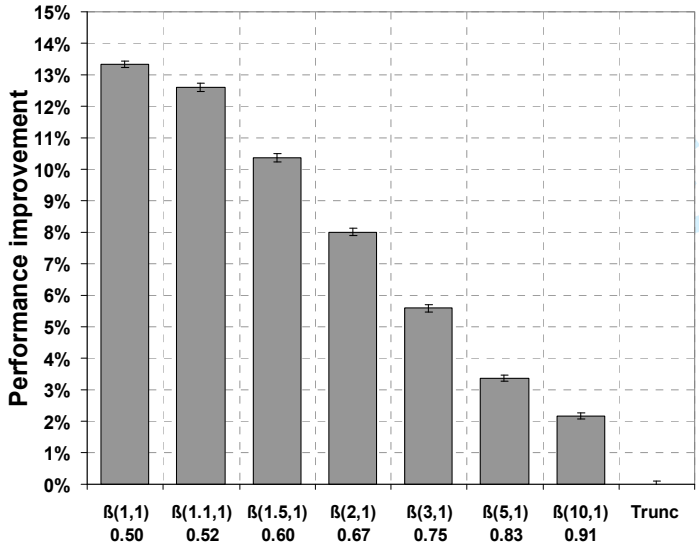

Figure 11: Percentage performance improvement over Truncation on a realistic scenario with five spacecraft. The number below each β function, is the mean of the distribution.

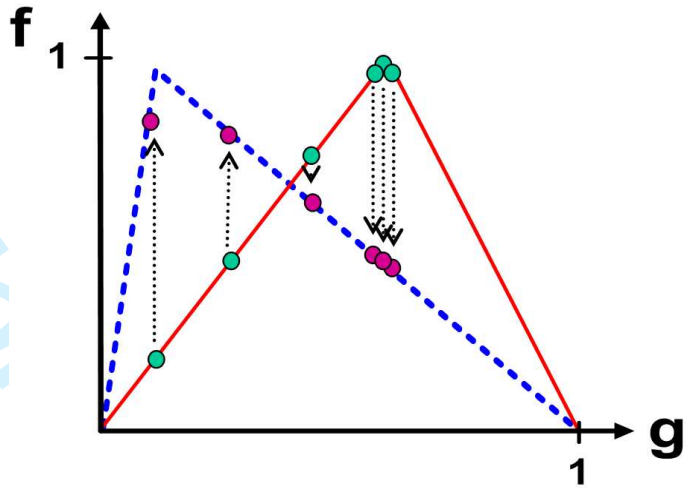

Figure 12: A schematic of how the dynamic nature of the problem alters the fitness distribution of the population at each jump.

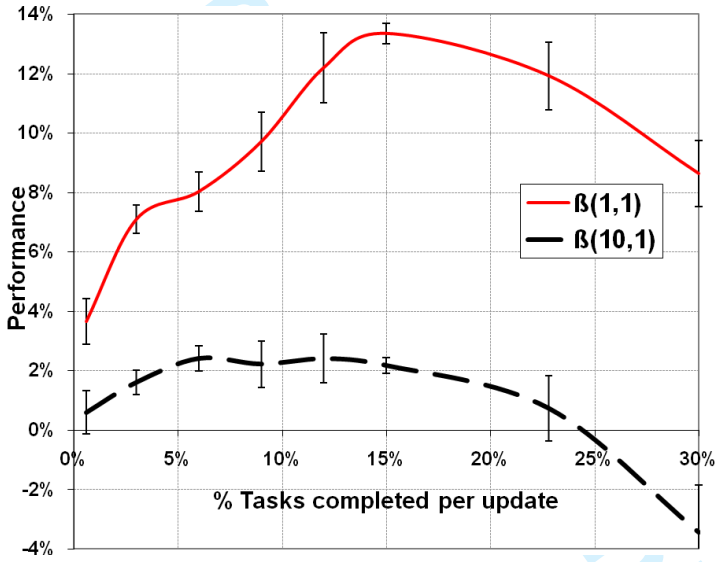

Figure 13: Varying the number of tasks per update (a proxy for communication bandwidth). Uniform distribution, β(1,1), and the more converged β(10,1), replacement operators are shown with performance as the % improvement over standard Truncation.

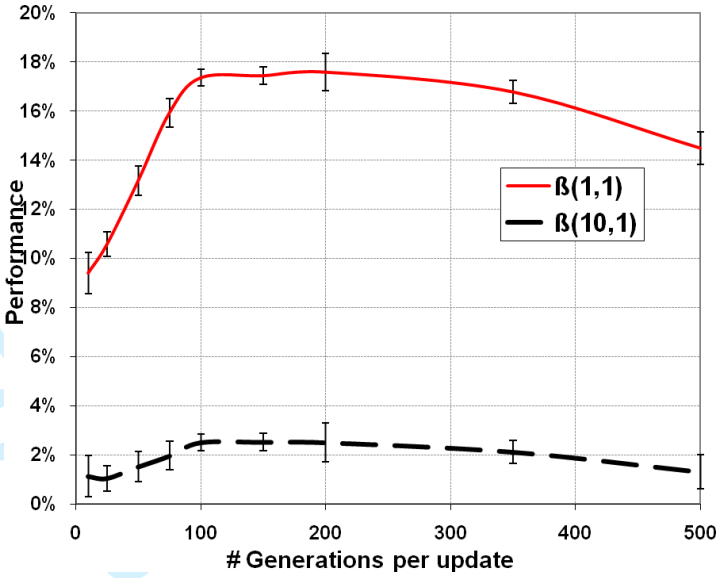

Figure 14: Varying the number of generations per update (a proxy for cpu power) using the same performance metric as Figure 13 above.

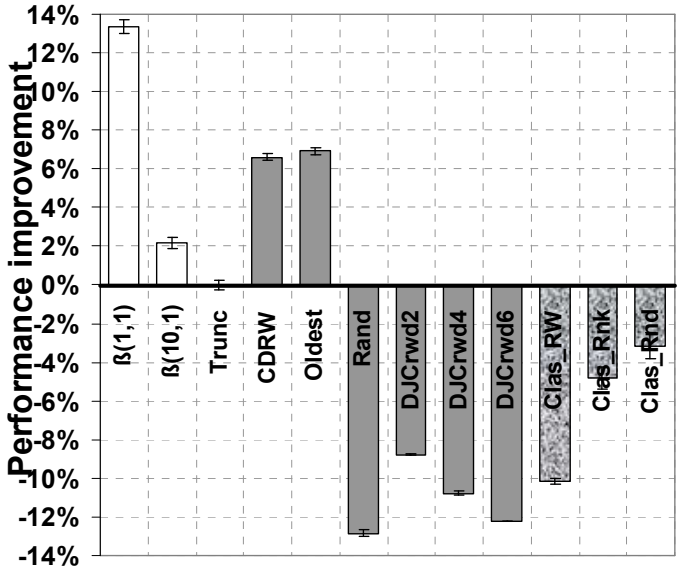

Figure 15: Percentage performance improvement over Truncation on a realistic scenario with five spacecraft

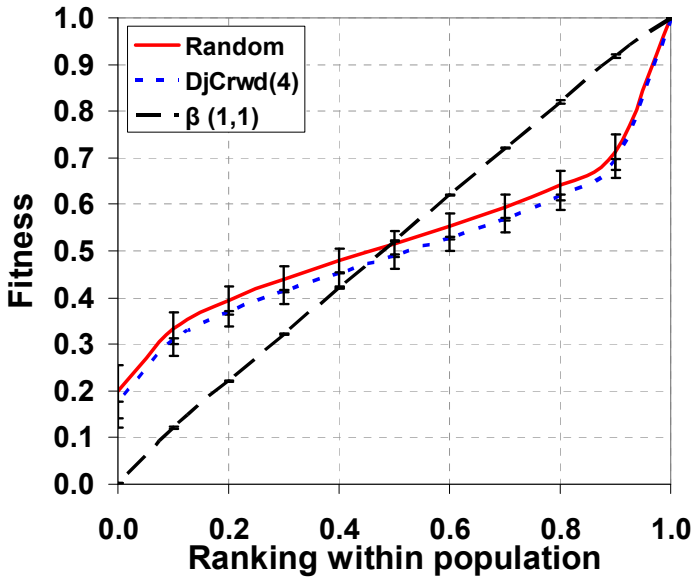

Figure 16: Distribution of fitness in a "converged" population averaged over 1000 generations for three different replacement operators.