1995120962

N95-27383

A Sustainable Genetic Algorithm for Satellite Resource Allocation

47258

p. 15

R J Abbott
The Aerospace Corporation and California State University, Los Angeles

M L Campbell
The Aerospace Corporation

W C Krenz
The Aerospace Corporation

A hybrid genetic algorithm is used to schedule tasks for a satellite, which can be modelled as a robot whose task is to retrieve objects from a two dimensional field. The objective is to find a schedule that maximizes the value of objects retrieved. Typical of the real-world tasks to which this corresponds is the scheduling of ground contacts for a communications satellite.

An imortant feature of our application is that the amount of time available for running the scheduler is not necessarily known in advance. This requires that the scheduler produce reasonably good results after a short period but that it also continue to improve its results if allowed to run for a longer period. We satisfy this requirement by developing what we call a sustainable genetic algorithm.

1.0 Introduction

Planning, i.e., deciding in advance on a course of action, is a long-standing and difficult problem.[AI90] Planning for a mobile robot is yet more complex as a result of the interactions among robot dynamics, horizon planning, and task valuation. When expressed numerically, planning problems are frequently ill-conditioned due to the combination of continuous and discrete variables over which decisions must be made. Consequently, not only are standard optimization techniques (such as non-linear programming) time consuming, they generally fail to provide satisfactory results. A good robot planning algorithm should:

a) generate efficient schedules of actions;
b) abide by system-imposed constraints;
c) be flexible enough that different tasking goals can be realized without major implementation changes;
d) be implemented in an efficient computational framework either through an extremely fast serial algorithm or through parallelization;
e) provide reasonable schedules on a short timeline and better schedules when more time is available.

This paper discusses a genetic algorithm approach to robot scheduling. The remainder of the paper is organized as follows. *Problem Statement* describes the robot's problem. *Analysis for a Numerical Approach* describes the equations that would have to be maximized were the problem to be solved numerically. *A Brief Introduction to Genetic Algorithms* provides an introduction to genetic algorithms. *The Application of Genetic Algorithms to Object Retrieval* describes our use of genetic algorithms for this problem. *Diversity Management and Sustainable Genetic Algorithms*

PRECEDING PAGE BLANK NOT FILMED

PAGE 102 INTENTIONALLY BLANK

describes our techniques for allowing our genetic algorithm to find good results without population convergence. *Results* describes our results*Future Work* describes future directions. *Conclusions* discusses conclusions and further work.

2.0 Problem Statement

The problem under consideration can be modelled as the scheduling of a robot whose job it is to retrieve objects while traversing a field. The robot moves at a constant rate along a set of pre-assigned horizontal passes. The robot has an arm and a hand. The arm has a limited length (both minimum and maximum) and a limited angle at which it can operate. (Details of robot kinematics are not addressed in this paper.)

Each object has a value and a location in the field, both known in advance. To retrieve an object, the robot hand must stay at the object's location for a given time. The objective is *to retrieve the highest total value of objects.*

Additional constraints may be imposed which make the problem even more complex.

- The value of an object may depend on which other objects are retrieved.
- The robot may have a net (instead of a hand) on the end of its arm with which it may retrieve many objects at once.
- The time required to retrieve an object may be expressed as a function of the robot's arm geometry. For instance, it may take more time to retrieve an object if the arm has to reach further or if it needs to deviate from its ideal position.
- The field may contain widely-spaced objects that need to be retrieved within some specified planning horizon.
- Some of the objects may have very precise angles at which they must be accessed.
- There may be a limit to the length of time the robot may operate during each pass. The limit may derive from physical constraints on the robot's operation, such as power or thermal limitations.

3.0 Analysis for a Numerical Approach

Initial attempts to produce efficient schedules were developed in a traditional optimization framework. Figure 1 shows a test suite of data that was constructed to illustrate the problem. A random field of 100 objects was created, and the robot is given 3 parallel passes through the field. In this example, three different kinds of objects are present. They are represented by different symbols for the different values.

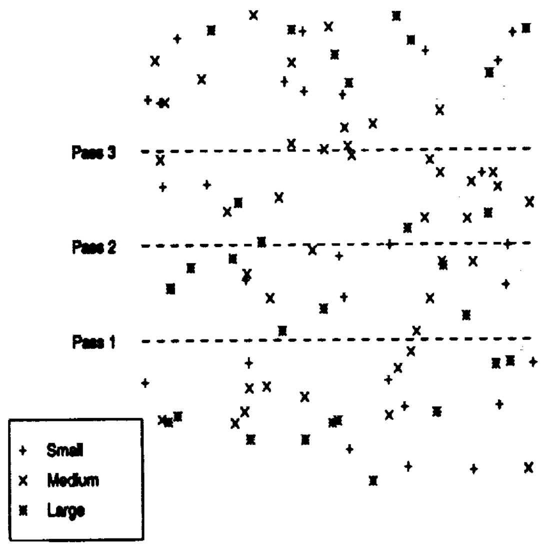

FIGURE 1. A testbed

The time between the completion of retrieving one object and the start of retrieving the next object must be greater than the time required to move the arm from one object to the next. The arm dynamics here are modeled simply as a fixed maneuver rate times the distance between subsequent objects:

(1)

$$T_{Man}(x_i, x_{i-1}) = \text{ManRate}\left\|\dot{x}_i - \dot{x}_{i-1}\right\| \quad (2)$$

where the variables are defined as:

T_i	Time required to retrieve an object and move to the next one
$T(x_i)$	Time at which object x_i is retrieved
$T_{Pickup}(x_i)$	Time required to retrieve object x_i
$T_{Man}(x_i, x_{i-1})$	Time required to move from one object to the next
ManRate	Maneuver rate of robot arm (distance/time)
$\dot{x}_i$	Vector position of point x

More complicated dynamics could be modeled without loss of applicability of the fundamental algorithms.

An added complexity is that the robot arm length and pass coordinates are such that some of the objects may be retrieved from any of a number of passes through the field. Objects in the center of the field, for example, are accessible from each of the passes; objects at the edge are accessible from only one pass.

The problem can be cast as an optimization problem (maximize the value of the objects retrieved) subject to the above constraints on robot reach and dynamics. Mathematically, this is cast in standard optimization form as:

$$\max_{t_x} f(x) \; : \; g(x) \leq 0 \quad (3)$$

where the parameter t_x over which the set will be optimized is the set of scheduled times for each point 'x.' The attributes of each point include:

$$x = \begin{bmatrix} \text{Value} \\ \text{Position} \\ \text{Accesses} \\ \dots \end{bmatrix} \quad (4)$$

where the value determines which objects are more important, the position drives dynamics constraints, and the number of accesses to an object may be used to decide which objects may be more easily postponed to a later pass.

Other attributes are also possible (for instance, preferred angles from which an object may be retrieved or length of time it takes to retrieve up a particular object). The function to be maximized is

$$f(x) = f_1(\text{Value}) + f_2(\text{Accesses}) + \dots \quad (5)$$

subject to the dynamic constraints of the robot arm (simplifying from Equation (5)):

$$g(x) = \left(\begin{matrix} \dots \\ g_i(x) = [T_{Pickup}(x_i) + T_{Man}(x_i, x_{i-1})] - T_i \\ \dots \end{matrix} \right) \quad (6)$$

Note that the access constraints are implicit in the set of *possible* scheduled points. All scheduled times outside of the feasible access time are assumed to be unscheduled points and are not considered in constraint calculations.

This optimization problem was run using a number of tools. Standard nonlinear programming algorithms[Mo92] were unsuccessful. Figure 2 shows a small field example run overnight using a nonlinear programming algorithm in which a human could easily eyeball a better schedule in a matter of seconds.

Our conclusion is that even though the problem (in simplified form) can be expressed mathematically, numerical solutions are not easy to find due to ill conditioning. Consequently a more robust and more flexible approach was explored: genetic algorithms.

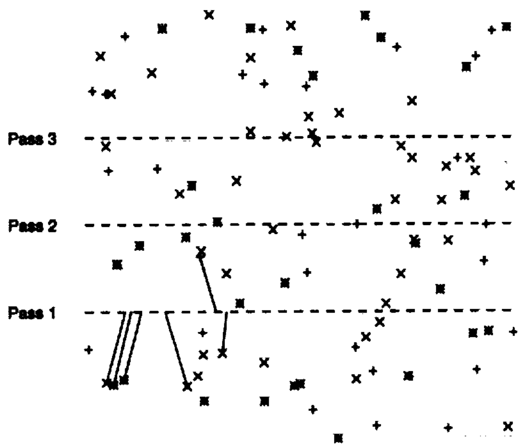

FIGURE 2. A disappointing result using non-linear programming

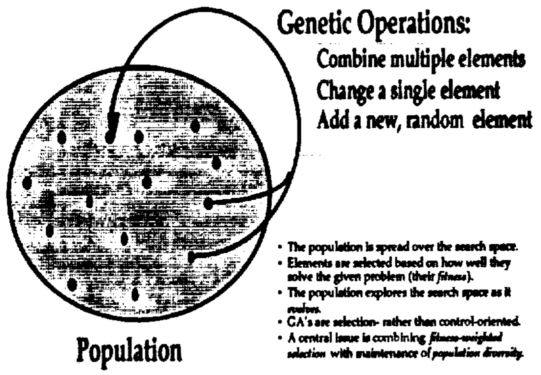
Genetic Operations:
Combine multiple elements
Change a single element
Add a new, random element
• The population is spread over the search space.
• Elements are selected based on how well they solve the given problem (their *fitness*).
• The population explores the search space as it evolves.
• GA's are selection- rather than control-oriented.
• A central issue is combining *fitness-weighted selection* with maintenance of *population diversity*.
Population

FIGURE 3. A generic genetic algorithm

4.0 A Brief Introduction to Genetic Algorithms

The term *Genetic Algorithms* [Ho75] includes a broad class of iterative optimization techniques that employ methods that are modelled after the way evolution occurs by natural selection in biological systems. The traits common to all genetic algorithms are discussed in the following algorithm schema.

1. **Populations.** Instead of iterating on a single solution (as in most iterative optimization methods), a genetic algorithm begins with a set of (suboptimal) solutions. In keeping with the biological/evolutionary theme, the set of candidate solutions is called the *population*. The initial population may be arbitrarily or randomly chosen, or it may be given as an external input.

2. **Element transformation.** One or more elements are selected from the population, modified to produce a new, possibly better solution, and then put back into the population, replacing a then current population element. (See Figure 3.)

   A distinguishing feature of genetic algorithms is the manner in which solutions are modified and in some cases combined to produce new solutions.

   - A solution may be modified to produce a new solution in a process called *mutation*. Nearly all optimization/search techniques use mutation in one form or another.
   - Two or more solutions may be combined to produce a new solution. The process of combination can create new solutions that combine the best attributes of their predecessors in ways that are very unlikely under purely random stochastic methods. This is widely considered as one of the sources of the efficiency and broad applicability of genetic algorithms.

3. **Non-determinism.** Most search techniques typically explore a search space by applying transformations to known elements to produce other elements. This is true of genetic algorithms as well. Genetic algorithms differ from other search techniques in that they are able to take better advantage of the non-determinism present in many such transformations.

Since genetic algorithms work with populations of elements, if a transformation includes a non-deterministic feature, a genetic algorithm is often able to accommodate the multiple outcomes of that transformation. This is not to say that the population is allowed to grow exponentially. But since an element is not necessarily removed from a population after a transformation is applied to it, the same element may be transformed by a non-deterministic transformation multiple times, producing multiple different results. All of those results have a chance of entering the population. The better ones are more likely to stay in the population than the worse ones.

In practice, non-determinism means that search transformations often include probabilistic elements.

4. **Fitness weighted selection.** An application-specific evaluation function is applied to each member of the population to rank the solutions according to what is known as their *fitness*.
   - When elements are selected for transformation, preference is given to selecting the higher ranking solutions.
   - When an element is selected to be eliminated, preference is given to eliminating the lowest ranked solutions.
5. **Iteration.** The selection/transformation/replacement process repeats until some termination criterion is met. The best ranked solution(s) are then produced as output.

If genetic algorithms were no more than searches based on populations and non-determinism, they would differ little from a probabilistic variant of best-first search. As mentioned above, perhaps the most important feature of genetic algorithms is that they generate new population elements by combining existing population elements. The rationale behind this feature grows out of the observation that in many search problems, good solutions often have features that are useful in many contexts.

The object retrieval problem provides some especially clear examples. If a large group of objects is close enough together to be retrieved by placing the net at a point that is within reach of all of them, that net placement is likely to be useful in many potential schedules. Similarly, if two groups of objects are sufficiently close that the second group can be retrieved by moving the arm minimally after retrieving the first group, that pair of arm placements will be a useful component in many schedules.

When a genetic algorithm allows two (or more) population elements to combine to produce a new population element, it allows useful features of the two elements to be combined in a single element.

Of course, when combining two elements, one does not necessarily know which features are useful. Even if one did, useful features are not always compatible. Non-determinism and fitness-weighted selection deal with that problem.

- Incompatible feature combinations produce poorly performing population elements, which are soon discarded.
- Useful features that are discovered independently generally survive in the population long enough to be combined in new population elements. For bit-string based genetic algorithms, such useful features are called *schemas*. Holland's Schema Theorem [Ho75] characterizes the transmission of useful schemas.

Genetic algorithms have been applied successfully to a wide range of optimization problems, such as the travelling salesman problem [Gr85], communication network design [Da87], natural gas pipeline control[Go83], image processing [Fi84], and other areas.

Genetic algorithms are a specific case of a more general concept called evolutionary computation, which includes the related fields of artificial life, artificial evolution, and complex adaptive systems. Artificial life refers to simulations of agents acting in some simulated world. The agents are typically ranked according to their success in dealing with the simulated world, and genetic algorithms are used to evolve representations of successively better agents. The agents may interact individually with the simulated world, or the agents may interact with each other in cooperative or competitive ways within the simulation.

5.0 The Application of Genetic Algorithms to Object Retrieval

In our first attempt to apply genetic algorithms to the object retrieval problem we first used the general-purpose bit-string based genetic algorithm tool Genesis [Gr84] to optimize the numerical formulation described above. A satisfactory schedule for a single pass, shown in Figure 4, was generated with this method.

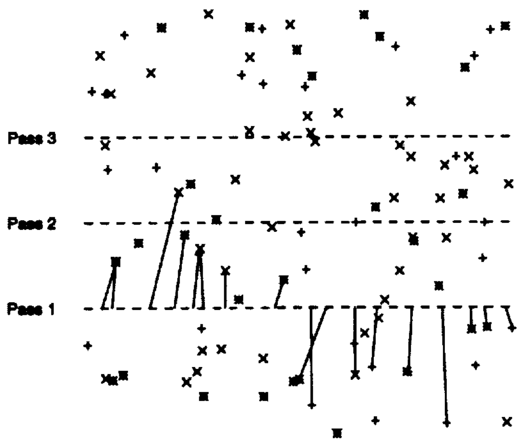

FIGURE 4. Result of applying a genetic algorithm applied to the numerical formulation

Compared to the results in Figure 2, the genetic algorithm shows marked improvement over nonlinear programming. However, extension from the single pass case to a multi-pass case was not feasible due to the increase in the number of operations, computation time, and the complexity of adjudicating retrieving decisions. Thus a more problem-specific genetic algorithm was implemented. It includes a number of distinctive features.

1. The population consists of actual schedules, rather than bit strings. Significant care was taken to represent schedules in a way that was not only intuitive, but also space efficient and computationally efficient.
2. Since the population elements are schedules, the genetic operators are all problem-specific. (This is known as a *hybrid* genetic algorithm. [Da91]) Operators are defined that transform one or two existing schedules into a new schedule.
3. A number of innovative population management strategies were employed. As discussed above, genetic algorithms serve two masters: short term optimization (hill climbing) and diversity. On the one hand, it is desirable to climb whichever hill one is on; on the other, one doesn't want the entire population to be marooned on a suboptimal peak. New population diversity techniques were combined with greedy genetic operators as a way of achieving both objectives. This is discussed below.

5.1 Schedule representation

A schedule is a sequence of *appointments,* where an appointment is a robot x-position along with an arm (x, y) position. (Appointments are given in terms of robot x-positions instead of time since the robot moves at a constant rate.)

As an illustration, Table 1 displays the beginning of one pass of a schedule.

Except perhaps for the *Object windows* column, this table should be self-explanatory. The *Object windows* column shows each *object*'s window of accessibility in the current pass, i.e., the range of robot x-positions during which the object is directly accessible to the robot hand, i.e., without a net. If, because of the robot's minimum arm length, there is an internal subwindow during which the object is not accessible, the subwindow is shown in angle brackets. This happens to be the case for the first and third object in the first appointment but for none of the other objects in the table.

5.2 The selection/replacement cycle

As the earlier discussion of probabilistic search explained, genetic algorithms generally advance through a combination of exploration and combination. Both exploration and combination require that one take one or more elements from the search space and produce a new element *in the search space*. In many domains, robot object retrieval included, that is not a trivial task. The primary difficulty is that the operations that one performs on a search space element do not always produce another valid search space element—in our case a schedule that satisfies the consistency constraints.

The primary consistency constraint on a schedule is that the robot be capable of moving its arm from one appointment to the next in the time allowed. A second consistency constraint is that no object be retrieved (or at least not be counted) twice.

Because it is not always easy to generate new population elements that satisfy the consistency constraints, production of new elements often involves two steps.

1. Generate a new element which may look like an element of the search space but which may or may not actually be a valid element of the search space.

2. Transform the new element into one that satisfies the consistency constraints.

TABLE 1. Example partial schedule (the first four appointments in one pass)

Robot *x* coordinate for this appointment	Robot net position for this appointment	Appointment value	Objects retrieved this appointment		
			Object positions	Object values	Object windows
0.000	(0.292, 0.232)	5	(0.273, 0.252) (0.275, 0.200) (0.319, 0.203)	1 2 2	[0.000 <0.185 - 0.361>0.569] [0.000 - 0.558] [0.035 <0.294 - 0.344> 0.603]
0.072	(0.090, 0.142)	8	(0.050, 0.135) (0.067, 0.130) (0.091, 0.142)	2 3 3	[0.000 - 0.301] [0.000 - 0.314] [0.000 - 0.346]
0.138	(0.247, 0.118)	7	(0.238, 0.126) (0.262, 0.150) (0.277, 0.092)	2 2 3	[0.000 - 0.482] [0.002 - 0.522] [0.061 - 0.493]
0.220	(0.257, 0.439)	6	(0.233, 0.471) (0.266, 0.426) (0.269, 0.439)	3 1 2	[0.000 - 0.480] [0.000 - 0.538] [0.004 - 0.535]
...	...	...	...	...	...

That is the process shown in Figure 5. The generation of a new element involves the following steps.

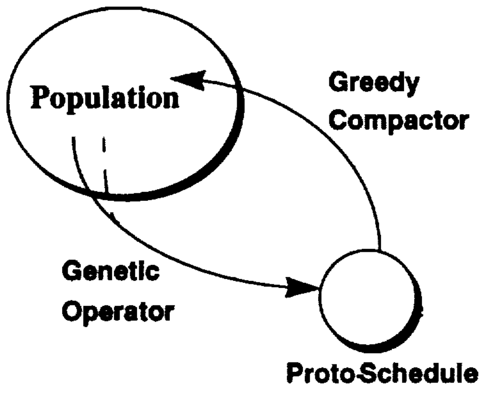

FIGURE 5. Population generation cycle

1. Select one or two elements from the population and transform it (or them) into what is called a proto-schedule, a structure that may or may not be a valid schedule.
2. Operate on the proto-schedule to produce a valid schedule.

The first transformation is the application of a genetic operator to one or two population elements. The second is the application of what we are calling a *schedule compactor.* The schedule compactor is itself a greedy scheduler. It transforms a proto-schedule, which may not satisfy the constraints, into an actual schedule, which does. We discuss the schedule compactor first and then the genetic operators.

5.3 The greedy schedule compactor

Like an actual schedule, a proto-schedule is a collection of passes, each of which is an ordered list of prospective appointments. A prospective appointment is a suggested arm position along with the list of the objects reachable from that arm position, as in Table 1. The only difference between a prospective appointment and an actual appointment is that a prospective appointment does not have an assigned robot x-position. The compactor's job is to associate robot x-positions with the given prospective appointments. It takes a table such as Table 1, but with no information in the first column, and for each prospective appointment it either assigns a robot x-position or deletes the appointment.

The schedule compactor does that job greedily. For each pass in the proto-schedule, the compactor makes a single traversal of the list of prospective appointments and throws out the ones that are incompatible with the constraints: the robot arm length, the time required to complete the previous appointment, and the maneuver time required to move the arm from the previous appointment. The compactor is greedy in the following ways.

1. Prospective appointments that are consistent with the constraints are scheduled as early as possible, i.e., the smallest possible robot x-coordinate consistent with the indicated net placement is selected.
2. Whenever possible, the compactor shifts objects from one appointment to an earlier appointment. If this involves shifting the arm position of the earlier appointment, that is acceptable as long as no objects are lost.
3. Whenever possible, the compactor adjusts an appointment's x-y arm position slightly if doing so would enable the robot to retrieve additional objects.

In addition, the compactor drops from prospective appointments any objects that have already been scheduled. Prospective appointments that are left without any objects are dropped entirely.

The compactor has a probabilistic feature built into it. The compactor must compact all passes of a proto-schedule. The order in which those passes are compacted may make a difference. If an object is reachable during multiple passes (as many are), the pass during which it is retrieved may affect other objects. To allow for all possibilities, the greedy compactor first constructs a random permutation of the passes and then compacts them in that order.

5.4 Genetic operators

The genetic operators perform two functions.

1. They are used to explore the search space. Typically, these are the so called *mutation* operators. A mutation operator takes a population element, i.e., a schedule, and transforms it into a proto-schedule. These transformations may or may not actually improve the schedule. They are simply exploration steps.
2. They combine two schedules to produce a new proto-schedule. The primary function of the combination operators is to combine features of good schedules in the hope of producing a better schedule.

The following mutation operators are defined. Most of them have a great many opportunities for non-determinism. These operators may or may not produce valid schedules. If they don't, the compactor makes the needed repairs.

- **change the pass of an appointment**. Move an appointment from one pass to another.
- **schedule an unscheduled object**. Retrieve an object that is not currently in the schedule and create an appointment for it.
- **interlard some unscheduled objects.** Sort a random selection of the unscheduled objects; allocate them to passes in which they have windows; and merge them with the current schedule. The merge process is the same as that explained below under merge two schedules. This operation is similar to schedule an unscheduled object. The difference is that it attempts to schedule collections of unscheduled objects instead of just one.
- **schedule a group.** Select all the unscheduled objects in a group and schedule appointments for them. (Recall that a group is an all-or-nothing affair. The robot does not get credit for retrieving objects in a group unless the entire group is retrieved.)

  There is no corresponding unschedule-group operation. Instead, whenever an element of the population is selected for transformation, one of the groups is (probabilistically) unscheduled.
- **exchange appointments.** The order of two adjacent appointments is switched.
- **generate a random schedule.** Generate a new, random proto-schedule. There are a number of probabilistic elements involved. The objects may first be ordered by value. In addition, the proto-schedule is generated by sorting the objects (one object per appointment) according to either x-y position or start-of-window-in-pass. If an object may be retrieved in a number of passes, the pass to which it is assigned is also deterministically probabilistically.

There is a single combination operator.

- **merge two schedules.** This operator combines and compacts two population elements. Appointments from corresponding passes of two schedules are merged, greedily. The merged result is guaranteed to conform to the constraints.

  The order condition that drives the merge is a combination of appointment x-position and appointment value. If the first available appointment from one schedule is both earlier than and more valuable than the first available appointment from the other schedule, it is selected. Otherwise, the schedule from which the next appointment is taken is selected at random.

6.0 Sustainable Genetic Algorithms

In the actual application, we sometimes want to run the genetic algorithm for an extended

d) **Evaluation functions that penalize popu-** random elements and tournament selection period. On other occasions, we need a reasonably good answer after only a relatively short run. We therefore want a genetic algorithm that can both (a) provide good results relatively quickly and (b) continue to improve if left to run for an extended time. We call a genetic algorithm with the second property *sustainable.*

One can produce reasonably good results quickly by including among our genetic operators, heuristics defined for the scheduling problem. Unchecked, however, this practice leads to population convergence at local maxima. Special techniques must be made to avoid such convergence. The following first discusses the mechanisms underlying population convergence and then describes ways to combat it.

set of search space elements that include a particular solution feature.

Traditionally, solution features that define hyperplanes have been called *schemas*. In scheduling, a schema would typically be a sequence of scheduled events, i.e., a schedule fragment. All search space elements that contain a particular schedule fragment may be considered to be on the same hyperplane. (Each search space element, i.e., a complete schedule, may lie on many intersecting hyperplanes simultaneously.) Useful schedule fragments will tend to be retained in the population. Hence, the population will tend to accumulate on the hyperplanes defined by useful features.

Since population size generally stays relatively constant over extended periods, and since ele

elements to be transformed and (b) the elements to be discarded.

We use a variant of tournament selection to make both selections. To select an element for transformation, a subset of the population, *the selection pool,* is chosen randomly and uniformly from the entire population. The best (or best two) element(s) of that pool are selected for transformation. To select an element to be discarded, we again choose a subset of the population; the worst element of the selection pool is selected for deletion.

Since elements are included in the selection pool with equal probability, the size of the selection pool is *inversely related* to the selectivity of the search. If the pool size were 1, one would be selecting (for transformation or deletion) an element uniformly from the population, i.e., with no regard for how well the element solved the problem. This would minimize convergence, but it would also minimize the likelihood that good features would be exploited.

On the other hand, were the pool to be the entire population, one would always select the best element(s) for transformation and the worst for deletion. This would maximize convergence, but it would virtually eliminate significant diversity.

Our strategy is to allow the size of the selec-

out. New entrants have an opportunity to be seen. This is comparable to local tournaments. As the season progresses, competition tightens; only the better entrants remain in the field. (Unlike sports, our population does not shrink, but the likelihood decreases that a poorly performing element will be selected for transformation.) Toward the end of the season, the selection pool is large and competition is extreme. Only "world class" elements survive. But as in competitive sports, because the entire process is probabilistic, there is always a chance that an underdog can make it to the "finals."

This seasonal cycle repeats itself continually. The selection pool size starts low, grows slowly, and then restarts at a low value for the next season. New elements with innovative features continually arise to challenge and add value to the current champions.

7.0 Results

The following plots illustrate the results of the genetic algorithm. For simplicity and consistency, these plots are based on a run with the following parameters.

Object field. The testbed example included 100 objects with values of 1, 2, and 3. The total of all objects (and hence the best possible schedule) was 201.

line from point to point) at 10 times the rate of its horizontal motion. The arm is assumed to move at a constant rate; there are no start-up or terminate arm motion penalties.

In this run, which showed typical results, approximately 40,000 schedules were considered. The best schedule had a value of 162. During the run, nearly 6000 random schedules were generated. The best of these had a value of 100. We take this as confirmation that the genetic operators added value to the search.

Figure 6 is a "fishbone" diagram. It shows the object field with the passes drawn as horizontal dashed lines. Each appointment is shown as a shadowed circle. The robot arm is shown as a line connecting the appointment to the position of the robot at the time of the appointment.

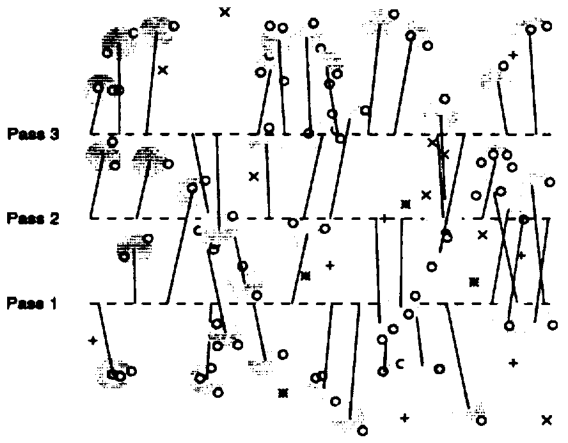

FIGURE 6. Arm positions

Retrieved objects are shown as o's; unretrieved object are shown as +'s, x's and *'s. Recall that some of the objects are in all-or-nothing groups. Four of the five groups were retrieved in their entirety. The largest group was not retrieved. Of the 11 objects in it, 3 were retrieved anyway even though they contributed no value to the schedule. Objects that appear to be easy picking but were not retrieved belong to the unretrieved group.

Figure 7 shows the trajectory of the end of the arm for this schedule. As this figure makes apparent, no effort was made to minimize arm motion. The only criterion for preferring one schedule over another was the total value of objects retrieved. If desired. such additional criteria could be added easily.

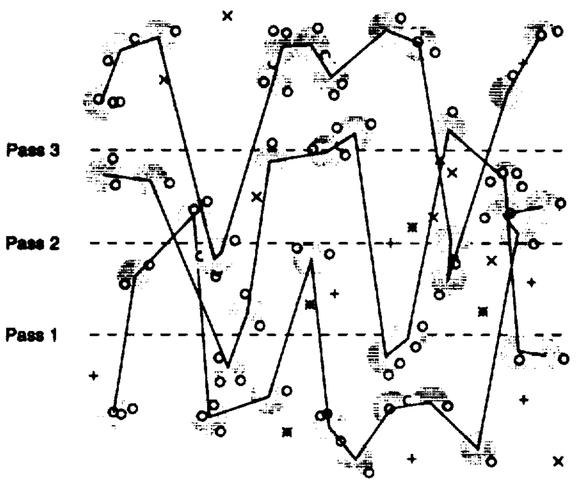

FIGURE 7. Arm trajectory

As a contrast, Figure 8 shows a schedule generated by a robot without a net: the "hand" must retrieve each object individually. Not surprisingly, this schedule is less effective in retrieving objects. More importantly, generating schedule variations of this sort turns out to be quite simple once the hybrid genetic algorithm framework is in place.

For each schedule, a partial family history is maintained. In particular, the values of the schedule's parent (or parents) is kept along with the history of the better parent. This associates with each schedule a record of that schedule's best parent back to the time when it was created as a random schedule.

Figure 9 compares the family history of the eventual Best Schedule with the best schedule in the population (Best of Population) at the time. This plot illustrates two features of this run.

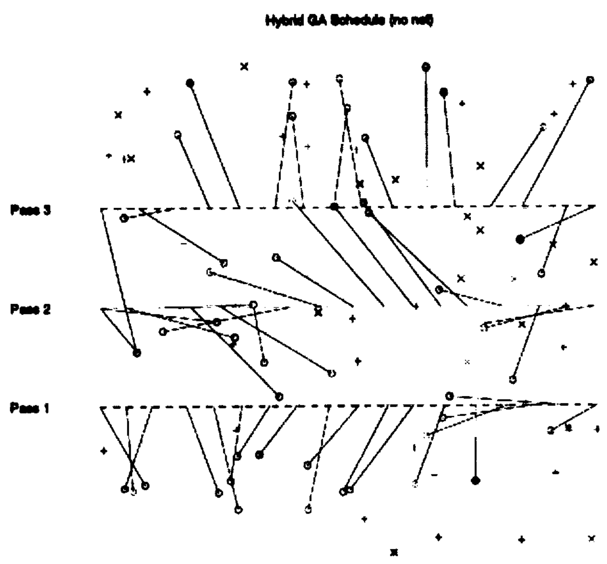

FIGURE 8. A robot without a net

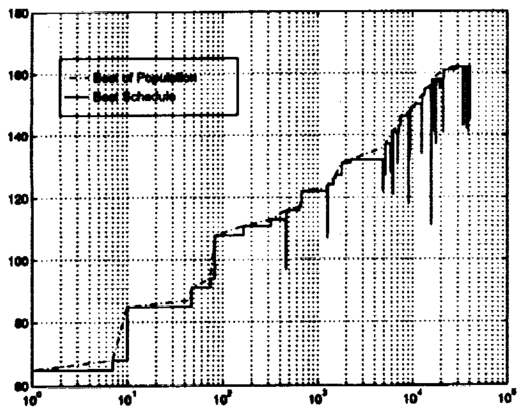

FIGURE 9. Best schedule genealogy

1. The eventual best schedule was close to the best during its entire genealogical history. (Had it not been, it would probably have gone extinct.)
2. Significant schedule degradations occurred before many of the advances. (Unfortunately, information regarding the nature of the degradations and whether or not they we prerequisite to the subsequent advances is not available.)

8.0 Future Work

Currently we are continuing to explore GA-based scheduling in two primary areas: scalability studies and new genetic operators. We are also looking for additional applications of GA-based scheduling/optimization.

In order to understand the scalability of the hybrid GA-based scheduler, we are currently performing a series of timing/profiling experiments over a wide range of problem sizes. By independently varying the size of the field and the number of objects, these scheduling runs will enable us to determine the dependence of the scheduler on the number and density of objects vs. the length of the resulting schedules for each pass. We are currently using a simple termination condition for the experiments: the algorithm stops after 10,000 schedules have been generated and evaluated. Preliminary results show a roughly linear increase in runtime vs. problem size for most cases.

Since the **merge two schedules** operation is one of the most time consuming hybrid genetic operations, it is natural to consider a more traditional genetic crossover operator. This new genetic operator would combine two schedules to produce a new population element according to the following process. The two schedules, represented as lists, will first be aligned with respect to time (in some way still to be determined,) and then the two lists will be traversed in lockstep with respect to time. As the two lists are traversed, the genetic operator copies from them to create the child list, with random crossovers from one parent to the other. Finally, the resulting proto-schedule is processed through the compactor to produce the new population element. It will be interesting to observe the effect that this new operator has on computational efficiency, population diversity, and the resulting schedules.

9.0 Conclusions

This study has demonstrated the applicability of hybrid genetic algorithms to a difficult scheduling problem. This problem resisted solution using more traditional techniques. Yet with a hybrid genetic algorithm good schedules have been generated.

Hybrid genetic algorithms differ from traditional genetic algorithms in that they make use of knowledge representation strategies and heuristics from the problem domain. This was very important for this problem in two ways.

1. It enabled us to represent the population of schedules in an efficient manner, precluding the need to transform bitstrings back and forth to schedules.
2. It allowed us to apply heuristics from the scheduling domain. Without these heuristics it is unlikely that we would have been able to solve the problem.

The primary genetic algorithm challenge is population management: how to manage the population so that (a) no one subpopulation drives out all others and (b) the various domain heuristics all have a reasonable opportunity to be applied. Our strategies of continually introducing new random elements and varying the competition level appear to have achieved these objectives.

The genetic algorithm paradigm provides a framework for selection-based search. As such it avoids many of the problems inherent in control-based search strategies. The inclusion as genetic operators of domain-specific heuristics from control-based algorithms allows one to combine the best of both approaches.

Acknowledgment

This work was performed at the Aerospace Corporation under sponsorship of its Aerospace Sponsored Research Program.

References

[Al90]. Allen, J, J Hendler, A Tate, (eds.), *Readings in Planning*, Morgan Kaufmann, San Mateo, Ca. 1990.

[Da87]. Davis, L and S Coombs, "Genetic algorithms and communication link speed design: theoretical considerations," *Proceedings of the Second International Conference on Genetic Algorithms*, pp 252-256, 1987.

[Da91]. Davis, L (Ed), *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New York, 1991.

[Fi84]. Fitzpatrick, JM, JJ Grefenstette, and D Van Gucht, "Image registration by genetic search," *Proceedings of the IEEE Southeast Conference*, pp 460-464, 1984.

[Go83]. Goldberg, DE, *Computer-Aided Gas Pipeline Operation Using Genetic Algorithms and Rule Learning*, Doctoral Dissertation, University of Michigan, 1983.

[Gr84]. Grefenstette, JJ, "GENESIS: a system for using genetic search procedures," *Proceedings of a Conference on Intelligent Systems and Machines*, pp 161-165, Rochester, MI, 1984.

[Gr85]. Grefenstette, JJ, R Gopal, B Rosamita, DV Gucht, "Genetic algorithms for the traveling salesman problem," *Proceedings of International Conference on Genetic Algorithms and their Application*, pp 160-165, Lawrence Erlbaum & Assoc., NJ 1985.

[Ho75]. Holland, JH, *Adaption in Natural and Artificial Systems.*The University of Michigan Press, Ann Arbor, Michigan, 1975.

[Mo92]. Moler, C, J Little, and S Bangert, *PRO_MATLAB for Sun Workstations: User's Guide,* The MathWorks, Sherborn, Ma 1993.