The Air Crew Scheduling System: The Design of a Real-world, Dynamic Genetic Scheduler

Soraya Rana-Stevens
BBN Technologies
Cambridge, MA 02138
email: sstevens@bbn.com

Benjamin Lubin
BBN Technologies
Cambridge, MA 02138
email: blubin@bbn.com

David Montana
BBN Technologies
Cambridge, MA 02138
email: dmontana@bbn.com

Abstract

The primary benefit of genetic schedulers is that they can often construct high quality schedules for long term scheduling. A common cost associated with genetic scheduling is that it may take tremendous amounts of time and computing power to produce high quality schedules, and these costs can make genetic scheduling impractical for many dynamic environments. The Air Crew Scheduler (ACS) system is built to crew tours for multiple squadrons in the United Stated Air Force. This system is built around a genetic scheduling engine that is designed to work alongside human schedulers to make crew assignments to tours. The crew scheduling problem itself is constraint-laden and requires rapid schedule updates for changes to crew member information and tour changes. This paper will provide a high-level overview of the problem and its constraints. We will provide a design overview of the genetic scheduler that has been constructed to rapidly schedule in this dynamic setting.

1 Introduction

The Air Force faces a number of challenging scheduling problems as well as other resource allocation problems. The Unit-Level Planning and Scheduling System (ULPS)[1] is a system designed to support the construction of tours and to manage and coordinate the personnel assigned to tours for the Air Force. For the purposes of this paper, we consider a **tour** is an ordering of flights such that the departure point of the first flight is the same as the arrival point of the last flight.

[1]Developed under prime contract by Unisys Corp.

The ULPS system is divided into two components: Mission Builder (MB) and the Air Crew Scheduler (ACS). The Mission Builder (MB) system [2] constructs tours and provides details on departure and arrival points, activities to take place, necessary qualifications for the tour, type of crew to assign to the tour, the aircraft type to be used, as well as other tour related information. The second component of ULPS is the ACS System[3]. The ACS system is designed to process the tours constructed by MB and allocate a set of appropriate personnel (crew) assignments so the tour can fly. The ACS system is both an automatic scheduler as well as a scheduling tool for the human schedulers.

The focus of this paper is the scheduling engine inside the ACS system. The ACS System uses a genetic scheduler customized for the Air Crew scheduling environment, where it must rapidly reschedule based on changes coming in from Mission Builder, in the form of tour additions, changes, and deletions as well as from several human schedulers, who can simultaneously manipulate the crew assignments. The genetic scheduler was carefully designed to work in conjunction and non-intrusively with the human schedulers to optimize and crew assignments. This paper will outline the ACS tool architecture in Section 2, provide details on the scheduling problem in Section 3, and present the design of the scheduling engine in Section 4.

2 The ACS Tool

As motivation for the genetic scheduler design, it is necessary to explain the context in which the scheduler is used. This section provides a brief overview of the ACS System design and describes how the system is intended to be used. We will also describe the GUI screens and buttons that allow the human schedulers

[2]Developed by Federated Software Group
[3]Developed by BBN Technologies

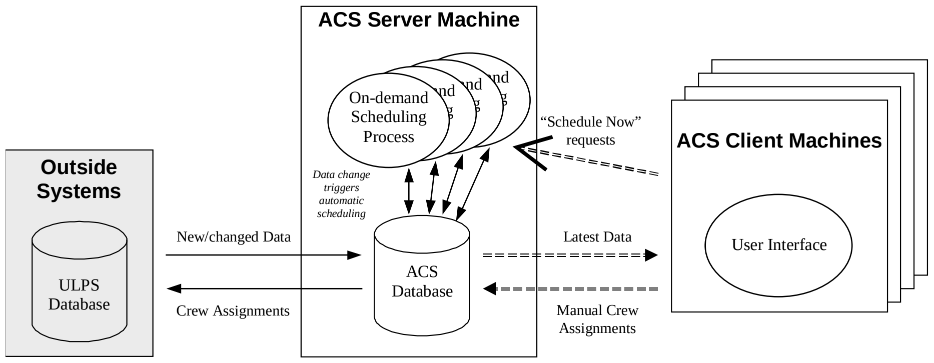


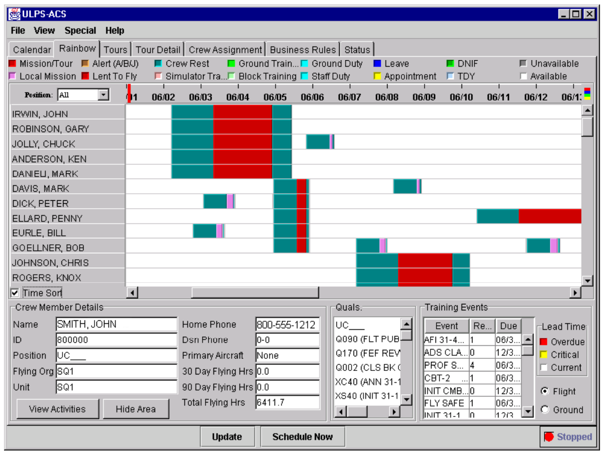
Figure 1: Overview of the ACS Architecture.

to pass information to the genetic scheduler.

ACS is a distributed system, consisting of a set of processes running on one or more server machines and a set of GUI processes running on multiple client machines. Figure 1 presents an overview of the system. Data flows from an external data source (the ULPS database) into the ACS database process. The data is partitioned into squadrons of crewmembers and tours and each squadron is scheduled independently: each squadron gets its own scheduling process and its own set of GUI-clients. When a given scheduling process recognizes that its data has changed, it begins an optimization cycle. This cycle ends when the scheduler determines that it is only making marginal improvements to the schedule. At this point the computed schedule is written back to the ACS database and the scheduling process waits until more changes are committed to the database.

At the same time that the optimization is occurring, multiple GUI client-processes may be interacting with the ACS Database process. The GUI clients permit human schedulers to view the existing schedule and then either approve or modify portions of it. These approvals are are sent back to the ACS database and the ULPS database so other systems may view the assignment information. Any changes to the schedule, either made by a single GUI-client or the automated scheduler, are posted to the ACS database and made available to other GUI-clients and the automated scheduler.

The automated scheduler is designed to not interrupt the human schedulers' activities. The automated scheduler runs passively in the background, which we will explain in more detail in Section 4.7. If changes to the database occur, it prepares to start running, but waits some (parameterized) time before automatically starting. This is to allow the GUI-clients to detect and make additional changes before the automated scheduler makes its suggested assignments. If the human schedulers want to force the scheduler to start running


ULPS-ACS
File View Special Help
Calendar Rainbow Tours Tour Detail Crew Assignment Business Rules Status
🟥 Mission/Tour ▨ Alert (A/B/J) 🟦 Crew Rest 🟩 Ground Train... 🟦 Ground Duty 🟦 Leave 🟩 DNIF ▨ Unavailable
▨ Local Mission 🟥 Lent To Fly ▨ Simulator Tra... ▨ Block Training 🟦 Staff Duty 🟨 Appointment ▨ TDY ☐ Available
Position: All
01 06/02 06/03 06/04 06/05 06/06 06/07 06/08 06/09 06/10 06/11 06/12 06/1
Time Sort
View Activities
Hide Area
Update
Schedule Now
Stopped

IRWIN, JOHN
ROBINSON, GARY
JOLLY, CHUCK
ANDERSON, KEN
DANIELI, MARK
DAVIS, MARK
DICK, PETER
ELLARD, PENNY
EURLE, BILL
GOELLNER, BOB
JOHNSON, CHRIS
ROGERS, KNOX

Crew Member Details

Name	SMITH, JOHN	Home Phone	800-555-1212
ID	800000	Dsn Phone	0-0
Position	UC___	Primary Aircraft	None
Flying Org	SQ1	30 Day Flying Hrs	0.0
Unit	SQ1	90 Day Flying Hrs	0.0
		Total Flying Hrs	6411.7

Quals.

UC___
Q090 (FLT PUB
Q170 (FEF REV
Q002 (CLS BK (
XC40 (ANN 31-1
XS40 (INIT 31-1

Training Events

Event	Re...	Due
AFI 31-4...	1	06/3...
ADS CLA...	0	12/3...
PROF S...	4	06/3...
CBT-2 ...	1	06/3...
INIT CMB...	0	12/3...
FLY SAFE	1	06/3...
INIT 31-1	0	12/3

Lead Time
🟥 Overdue
🟨 Critical
☐ Current

(● Flight) ○ Ground

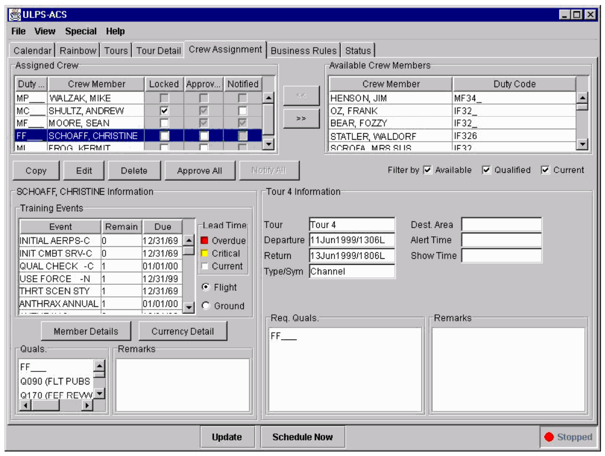
Figure 2: The ACS Rainbow Screen for track and edit all crewmember activities.

rather than waiting for it to time-out and start running automatically, they can press the *Schedule Now* button that tells the automated scheduler that the human scheduler is ready for it to build and post its recommended schedule.

2.1 The Graphical User Interface

The ACS system has a tab based interface, where each tab represents a different view of the current schedule data. The two GUI tabs that allow the human scheduler to indirectly interact with the automated scheduler are the **Rainbow** and **Crew Assignment** tabs.

The **Rainbow** (Figure 2) tab displays a resource-centric view of the schedule. Down the left hand-side of the screen a list of crewmembers is displayed. Clicking on one of the names displays details about that individual in the panel on the bottom of the screen. To the right of each name is a chart showing time-blocks for each activity that person is scheduled to do. Several different types of time-blocks can be displayed according to the color-coding indicated at the



Assigned Crew

Duty ...	Crew Member	Locked	Approv...	Notified
MP___	WALZAK, MIKE			
MC___	SHULTZ, ANDREW	✓	✓	
MF___	MOORE, SEAN		✓	✓
FF___	SCHOAFF, CHRISTINE			
MI	FROG, KERMIT			

Available Crew Members

Crew Member	Duty Code
HENSON, JIM	MF34_
OZ, FRANK	IF32_
BEAR, FOZZY	IF32_
STATLER, WALDORF	IF326
SCROFA, MRS SUS	IF32

SCHOAFF, CHRISTINE Information

Training Events

Event	Remain	Due
INITIAL AERPS-C	0	12/31/69
INIT CMBT SRV-C	0	12/31/69
QUAL CHECK -C	1	01/01/00
USE FORCE -N	1	12/31/99
THRT SCEN STY	1	12/31/69
ANTHRAX ANNUAL	1	01/01/00

Lead Time: 🟥 Overdue 🟨 Critical ⬜ Current

[✓ Flight] Ground

Quals.: FF___, Q090 (FLT PUBS, Q170 (FEF REVW

Remarks:

Tour 4 Information

Tour: Tour 4
Departure: 11Jun1999/1306L
Return: 13Jun1999/1806L
Type/Sym: Channel
Dest. Area:
Alert Time:
Show Time:

Req. Quals.: FF___

Remarks:

Figure 3: The ACS Crew Assignment Screen edit and viewing details on scheduled tour assignments.

top the screen. Most of the blocks represent a position and tour to which a given crewmember is currently assigned. However, the human scheduler can add and edit tour and other activities on this screen. Any activities added here will change a crewmember's availability and may cause the automated scheduler to reschedule.

The **Crew Assignment** (Figure 3) tab displays an assignment-centric view of the schedule. A given assignment can be generated either by the automated scheduler, or by manual human intervention. This screen allows the human scheduler to manipulate the set of assignments by manually assigning (or unassigning) crewmembers to positions, locking down assignments to positions (making them ineligible for automatic scheduling), approving an assignment and notifying a crewmember of his/her assignment. In addition, the human scheduler can add new positions or change requirements for the positions, such as changing a regular pilot position to an Instructor pilot position (meaning that only Instructor pilots can be assigned to that position).

The Rainbow screen, the Crew Assignment screen and the *Schedule Now* button are the only means for the human scheduler to interact with the automated scheduler. The Rainbow screen maintains availability information for the crewmembers (resources) and the Crew Assignment screen maintains the status of the tour assignment. When the human schedulers make changes on those screens by adding/removing availabilities or by manually assigning or unassigning a crewmember to a position, the automated scheduler needs to incorporate the human scheduler's changes into the current schedule to make any new assignments that it can or verify that the human schedulers are working with a near optimal or optimal set of assignments. To expedite the process or if they are simply not satisfied with the current assignment, the human schedulers are always given the option to press the *Schedule Now* button to force the automated scheduler to start the rescheduling process.

3 The ACS Scheduling Problem

Genetic algorithms are typically designed as black box optimizers that use an evaluation function to provide feedback that directs the search [2]. In scheduling domains, this evaluation function is usually a model of the problem that is applied to different candidate solutions to determine how effective each solution is with respect to the model. Modelling scheduling domains can be done through simulation that tracks particular domain-specific metrics that are used to distinguish high-quality solutions from low-quality solutions [3, 5]. Real-world domains can also involve constraints that dictate what is or is not a feasible schedule. The Air Crew Scheduling problem uses both simulation and constraints to evaluate candidate schedules.

At the highest level, the Air Crew Scheduling (ACS) problem is to match positions on pre-defined tours to qualified and available personnel. The *fitness* of a schedule is measured by the degree to which the schedule meets constraints; thus, the multi-objective evaluation function is designed to assign a single score to indicate how well a particular schedule meets several (potentially conflicting) constraints. The constraints to which a schedule must adhere in order to be feasible are called *hard constraints* referring to the fact that these constraints cannot be violated. A different set of constraints, known as soft constraints, may be violated, but doing so incurs a penalty that is incorporated into the evaluation function. Many of these hard and soft constraints are not entirely fixed but partially dependent on data and logic that potentially vary with the aircraft type and squadron. Editable **business rules** capture this variable data and logic in such a way that it is easy to modify them without changing the software.

3.1 Hard Constraints

The ACS system has a number of hard constraints that must be applied to all schedules to ensure feasibility. We ensure that the hard constraints are not violated during the scheduling process by using many of the hard constraints to dynamically construct the initial scheduling problem configuration. We also ensure that the remaining hard constraints are never violated during the scheduling process. The following list describes the eight hard constraints:

Crew Complement: A crew complement is the set of positions to be filled on a particular tour. The Mission Builder system sends in tours and their details, from which ACS deduces the skeleton crew that each tour needs in order to fly. A sample crew complement may be five positions that require a Pilot, a Co-Pilot, a Flight Engineer, a Boom Operator, and a Loadmaster. Associated with each position is a set of qualifications that personnel must have in order to be assigned to that particular task during the scheduling process.

Locked Assignments: The automated scheduler is not allowed to change the crewmember(s) locked to a given position by a human scheduler.

One Person Per Position: The automated scheduler can assign only one crewmember per position.

Availability: A crewmember cannot be assigned to a tour if the tour conflicts with other activities.

Crew Rest: Before and after each mission, a required period of time is designated, during which a crewmember cannot be assigned to other duties.

Qualifications: The automated scheduler can only assign a qualified crewmember to a given position.

Currency: A currency task is a training activity that crewmembers must do periodically in order to be considered *current*. A person can be qualified to perform a particular duty, but if they are not *current* for that duty (due to not performing the task recently), then they cannot be scheduled until they regain currency. A crewmember will not be assigned to a tour if the crewmember is non-current.

Person-Person and Person-Tour: These are editable rules, called business rules, that can potentially specify soft or hard constraints on who can fly together and who can fly on which tours. When a rule is designated as *Impossible* it becomes a hard constraint that the automated scheduler does not violate.

When a scheduling problem is constructed from the set of positions and resources that need to be assigned, most hard constraints can be used to filter the set of possible crewmembers that are eligible for assignment to a specific position. We use the Crew Complement, Locked Assignments, Availability, Qualifications, Currency, and Person-Tour constraints to determine which resources could, hypothetically, be assigned to a particular position. The remaining hard constraints are maintained during the scheduling process.

3.2 Soft Constraints

Since the hard constraints are being used to configure/formulate the problem, the fitness of a schedule is computed based on the soft constraints (or preferences). We currently compute and track five variables, S_1 through S_5, to provide the estimates for how well different soft constraints are being met by a particular schedule. The variables represent penalties that need to be minimized by the genetic algorithm. The overall score calculated by the evaluation function is a weighted sum of five penalties:

$$S = w_1 S_1 + w_2 S_2 + w_3 S_3 + w_4 S_4 + w_5 S_5$$

The weights in the equation are parameters that we are setting to assign an importance to each of the variables. We now discuss these five subscores:

Crew the Tours: The primary scheduling goal is to assign complete crews to all Tours, where a complete crew is a set of assignments to each position in the tour. The score also considers priority of different tours.

Reduce Schedule Turbulence: This score computes how much a schedule has changed from what a human scheduler had approved or notified. Once a human scheduler has approved of a schedule, the genetic algorithm attempts to keep approved assignments over replacing them with similar, but unapproved, assignments. A considerably higher penalty is given if the human scheduler has actually notified the crewmember of his/her duty assignment. The notification penalty is further weighted (nonlinearly) depending on how soon the assigned tour was scheduled to depart.

Maintain Training Currency: Crewmembers should be placed on tours that help them to fulfill currency requirements. Each tour has opportunities for different training events. When crewmembers are assigned to positions on these tours, they can be given credit for these training activities to help them meet their currency needs for a specified period of time. A penalty is given for each crewmember who has currency requirements to fill. As the deadlines for fulfilling the currency requirements get closer, the penalty increases nonlinearly.

Schedule Equity: Crewmembers should be scheduled evenly based on their availabilities.

Person-Person, Person-Tour, Qualifications: These editable rules capture preferences from *Smith should not fly with Jones* to *an inexperienced pilot should not fly on a tour that lands at a particular destination*. Each user-defined preference has a qualitative value for the associated penalty (very bad, bad, good, very good). An extra penalty of *Impossible* translates the preference into a hard constraint. A Qualification penalty is applied when someone is overqualified to perform an activity, such as a pilot filling a co-pilot position.

3.3 The Simulation

The simulation step of the scheduling process involves allocating training events to crewmembers. This step is necessary to evaluate the soft constraint penalty for Maintaining Currencies. For each currency item needed, we partition the number of unfilled currency events into *needs* and *wants*, where a need is the number of events due in the next 30 days and a want is anything else. In most cases, the currency tasks will fall into the want category. This simply establishes what absolutely needs to get done this scheduling period and what we would like to get done this scheduling period.

For each tour, we maintain information for each training event to indicate how many training event slots are available and allocated. The training event allocation process is a two-pass calculation. First, crewmembers will be allocated as many of their needed currency items as possible. A first come first served resolves any ties when training events need to be shared between crewmembers. Second, crewmembers must share wants on tours. If there are more wanted currency events than available training event slots, the slots are divided based on how many positions there are on the tour so each crewmember assigned to those positions gets some of their wanted currency items.

4 The Genetic Scheduler

Choosing the right representation and search operators for a scheduling problem or any other optimization problem can dictate search performance, both in terms of solution quality and computation time. The most common representation for a chromosome in genetic scheduling is a permutation of tasks [4, 5] to which a subsequent assignment of resources is made. How the subsequent assignment of resources is made can be either knowledge-intensive (i.e., specialized heuristics for choosing the best resource for the task) or knowledge poor (i.e., a purely greedy, random or first-come first-served approach). Using task permutations requires specialized crossover and mutation operators. Several generic crossover operators have been developed for permutations [4, 5]. The permutation representation is applicable when tasks and resources both need to be scheduled; however, the ACS scheduler uses prescheduled and fixed task start and end times. Therefore, the ACS scheduler uses an alternative representation and operators to directly manipulate the task-to-resource mapping rather than manipulating a task ordering.

4.1 Representational Issues

Two factors that are often overlooked when developing representations for genetic schedulers are *representational isomorphisms* and *feasibility maintenance*. These factors can dramatically increase the size of the search space and make optimization much more difficult.

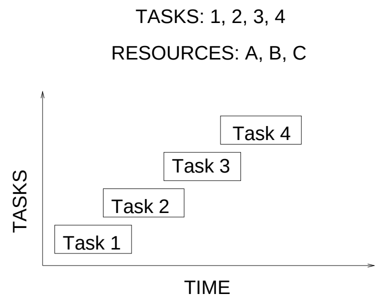


Figure 4: Sample four task three resource scheduling problem.

Representational isomorphisms occur when different permutation chromosomes represent exactly the same schedule. A potential problem with using task permutations at the chromosome level is that independent tasks can be arbitrarily reordered with respect to one another and still compete for the same resources. Consider Figure 4 in which we have four non-related scheduling tasks to perform and three resources capable of performing those tasks. Suppose we have a resource assignment algorithm that treats our task permutation as a priority queue and assigns an available resource, either deterministically or non-deterministically, to the next task in line. Given the time ordering of the tasks, a valid schedule is $\{(1, A), (2, B), (3, A), (4, B)\}$ meaning Resource A is assigned to Tasks 1 and 3 and Resource B is assigned to Tasks 2 and 4. If we use a non-deterministic resource assignment strategy, it is possible to produce this same schedule using 4! different task permutations. Even with a deterministic resource assignment strategy we can still have multiple task permutations that result in identical schedules.

Feasibility maintenance means that we require that the genetic algorithm work only with feasible schedules. The feasibility maintenance issue can arise when we use a genetic algorithm (or simulated annealing [6]) to simultaneously optimize the task ordering and resource assignment. If we are explicitly working with task to resource mappings, then we may need to repair schedules to ensure that they are feasible.

4.2 The Multi-Set Representation

The ACS genetic scheduler uses a multi-set to represent the mapping of resources to tasks, where a multi-set is a set that contains duplicate elements. We augment our set of resources with a *null* resource (ϕ) to represent that no resource is assigned to a particular task. This representation is a mapping and a direct encoding of a schedule, so it is unnecessary to permute the scheduling tasks. This explicit mapping prevents representational isomorphisms. Using our example from the previous section, there is only one schedule that represents $\{(1,A),(2,B),(3,A),(4,B)\} \rightarrow ABAB$ (assuming a fixed ordering of the tasks).

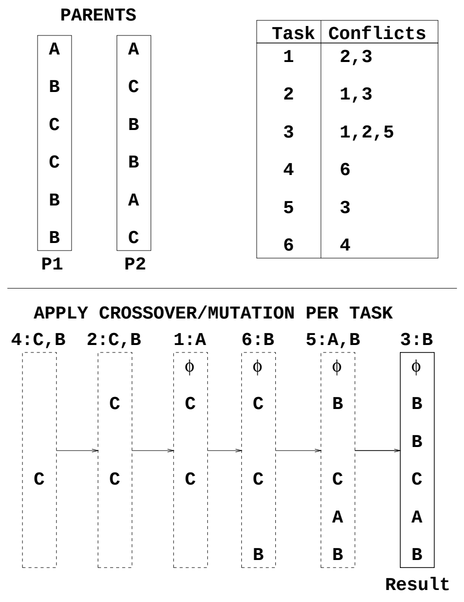
4.3 Fill

The **Fill** algorithm is a non-deterministic algorithm that randomly assigns qualified, current, and available resources to tasks that are assigned the *null* resource. Fill is called to postprocess all chromosomes before the schedule is evaluated and maintains any additional hard constraints. A random initialization procedure consists of applying the Fill algorithm to chromosomes that have the *null* assignment to all tasks. The Fill algorithm then randomly fills tours and creates fast initial feasible schedules. The Fill procedure ensures that any positions that can be assigned will be assigned before a schedule is evaluated. It fills the assignments directly onto the GA chromosome so no added representational overhead is incurred.

4.4 Constraint Based Crossover

The Constraint Based Crossover is a variant of Uniform Crossover that is applied to feasible task to resource mappings in a random order. As we are restricting crossover to take two valid schedules and produce a single new valid schedule, we need to ensure that no hard constraints are violated during crossover. To avoid this situation, we use task overlap constraints to process each crossover assignment to verify that no conflicts occurred. We use an adjacency matrix to represent conflicting tasks, where entries in the matrix indicate which tasks overlap. We use the matrix to restrict choices at each locus in the offspring chromosome based on the values that have been set. As an assignment is made, we use the task overlap information to restrict future choices at each locus that has yet to be set in the offspring chromosome.

In general, the schedules that are produced by



Task	Conflicts
1	2,3
2	1,3
3	1,2,5
4	6
5	3
6	4

Figure 5: Example constraint based crossover.

crossover are *underspecified*, meaning that there may be positions that could not be assigned due to conflicts in each parent, but may be assignable in the offspring as other conflicts may have been eliminated during crossover. It is not the role of crossover to exploit these obvious holes in the schedule. The Fill algorithm will perform that task in a subsequent procedure.

4.5 Mutation By Omission

The motivation for Crossover is to exploit commonalities between parent strings while the motivation for Mutation is to ensure that exploration continues throughout search. The postprocessing step using the Fill algorithm means that we can mutate a locus by assigning it to the *null* resource and leave that position unassigned until Fill postprocesses the chromosome. Thus, when mutation occurs, that means that we have not made an assignment to a task.

4.6 Example of Crossover and Mutation

A simple example of our crossover and mutation operators is given in Figure 5. The task conflicts are set once when the scheduling problem is read from the database. When parent strings are recombined, the task conflict information is applied to restrict the set of possible resource assignments. In the example, task 4 is chosen as the first crossover point. The parent strings have resource assignments of C and B; thus, these values, along with ϕ are valid resource choices. Since C was chosen, task 6 can no longer be assigned

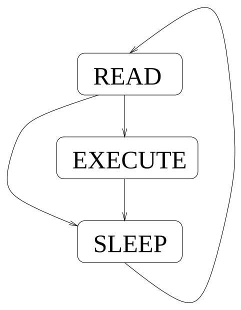


Figure 6: Genetic Algorithm controller.

to resource C since task 4 conflicts with task 6. Similarly task 2 is assigned to C, so C is eliminated as a choice for task 3, but task 1 does not have C as a choice so no further restrictions need to be made. For task 1, however, a mutation was chosen, so the ϕ resource was assigned and will be filled in by the Fill procedure after crossover is complete.

4.7 The Dynamic Execution

The ACS genetic scheduler is working in a dynamic environment where changes are constantly occurring due to tour and crewmember changes coming in from the ULPS database and from the human schedulers. This means the genetic scheduler must constantly revise schedules to fill in any assignments and to optimize the schedules over the long term scheduling window. At the same time, the genetic scheduler needs to be cautious in how it makes changes so the changes do not intrude on the human schedulers' activities.

The dynamic execution requires rapid (on the order of seconds or a few minutes) schedule production, so users are not waiting long for new schedules to arrive. We use a genetic algorithm controller to control how and when the genetic algorithm runs and we designed the genetic algorithm to produce a schedule quickly, within 1000 evaluations. An outline of the controller for the genetic scheduler is given in Figure 6.

The controller loop controls when the genetic algorithm enters and exits the Read, Execute, Sleep states. The Read state queries for any updates to the ACS database. The database sends all up-to-date scheduling information from which the genetic algorithm configures its scheduling problem. If there is no scheduling to be done, the controller enters the Sleep state. Otherwise, the genetic algorithm caches the current set of assignments to *seed* the initial population of the genetic algorithm. This means each execution step of the genetic algorithm can be thought of as a partial restart that is initialized from a valid solution. The Execute

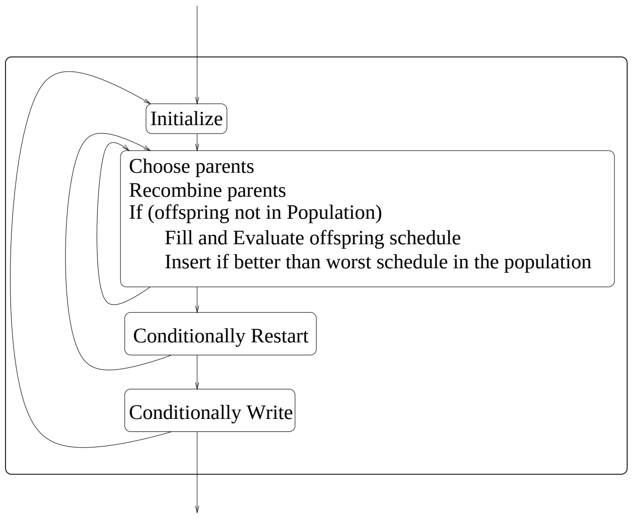

Initialize
Choose parents
Recombine parents
If (offspring not in Population)
Fill and Evaluate offspring schedule
Insert if better than worst schedule in the population
Conditionally Restart
Conditionally Write

Figure 7: Genetic Algorithm execution algorithm.

state of the controller runs the genetic scheduler and periodically posts changes to the ACS database. When the genetic scheduler has run to completion, the automated scheduler enters its Sleep state. The sleep state will be exited immediately if changes happened while the genetic algorithm had been running, or if a user pressed the *Schedule Now* button. Otherwise, the scheduler will wait for changes and wake up upon detecting changes to the database (with a parameterized delay). This Sleep state is the key to giving the genetic scheduler a passive behavior, which keeps it from interfering with the human scheduler's activities. The GUI-screens have a color-coded status indicator that reports whether the genetic algorithm is preparing to run, running, or sleeping so the user is aware that a new schedule might be arriving soon.

4.8 The Execute State

A more detailed outline of the Execute state of the ACS genetic scheduler is shown in Figure 7. The design of the population and execution was inspired by the CHC genetic algorithm [1]. Our population size is 30 individuals, but we do not allow duplicate chromosomes into the population. The innermost loop in the diagram represents a single execution of the genetic algorithm, using selection, crossover, mutation, Fill and evaluation. Naturally, using such a small population results in fast convergence. Like CHC, we perform a partial restart upon convergence, which is represented by the second loop in the diagram. A partial restart occurs by using the best member of the population to *seed* the remainder of the population. There are a limited number of partial restarts that can take place before the genetic algorithm is forced to write its solution to the ACS database; however, the genetic algorithm will automatically stop and write if it converges to the same solution repeatedly (the cutoff is specified by a parameter). When the genetic algorithm is finished with a single run, it attempts to write its solution back to the database. The write will not take place if the database has changed since the problem was read or if the cached schedule has the same fitness as the best solution in the population. Finally, the outermost loop is used to perform a larger scale restart. This loop determines, based on a parameter, whether the difference between two consecutive solutions is large enough to continue searching. If the difference is above the tolerance, search is restarted, otherwise, the controller moves into the Sleep state.

5 Conclusion

The ULPS ACS scheduling system is a dynamic genetic scheduler designed to function in the real world. The design of the scheduler is motivated by the need for rapid reaction to changes in the system. To this end, we use special genetic operators that ensure that the genetic algorithm is always manipulating valid feasible schedules and our representation guarantees that a chromosome decodes to a unique schedule. Furthermore, the scheduler takes an iterative approach to search – using small populations that yield fast-run times on a single execution to post schedule changes quickly. However, the fact that the scheduler is running multiple times, as dictated by changes to the system or by human direction means that the schedule will be refined over the long term. Furthermore, since the genetic scheduler is always starting from the current schedule that is being used by the human schedulers, it will tend to produce schedules that incorporate those assignments into it. This design is one approach to making genetic scheduling more computationally tractable for rapidly changing scheduling environments.

6 Acknowledgements

The authors wish to acknowledge the support of the United States Air Force AMC and Unisys Corporation. This work was funded under a Task Order through US Government Contract DCA100-96-D-0048. The authors wish to thank the Unisys domain experts, John Irwin and Gary Robinson and the BBN ULPS development team: Ken Anderson, Mark Danieli, Mark Davis, Peter Dick, Penny Ellard, Bill Euerle, Bob Goellner, Chris Johnson, Chuck Jolly, Knox Rogers, Christine Schoaff, and Mike Walczak.

References

[1] L. Eshelman. The CHC adaptive search algorithm. how to have safe search when engaging in non-traditional genetic recombination. In G. Rawlins, editor, *Foundations of Genetic Algorithms*, pages 265–283. Morgan Kaufmann, 1991.

[2] D. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley Publishing, Co., Reading, MA, 1989.

[3] T. Starkweather, D. Whitley, K. E. Mathias, and S. McDaniels. Sequence scheduling with genetic algorithms. In Springer-Verlag, editor, *New Directions for Operations Research in Manufacturing*, New York, 1991.

[4] G. Syswerda and J. Palmucci. The application of genetic algorithms to resource scheduling. In L. Booker and R. Belew, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms.* Morgan Kaufmann, 1991.

[5] D. Whitley, T. Starkweather, and D. Fuquay. Scheduling problems and traveling salesmen: The genetic edge recombination operator. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms.* Morgan Kaufmann, 1989.

[6] M. Zweben, B. Daun, E. Davis, and M. Deale. Scheduling and rescheduling with iterative repair. In Monte Zweben and Mark Fox, editors, *Intelligent Scheduling*, pages 241–255. Morgan Kaufmann, 1994.