Appears in Proc. AHS International Specialists' Meeting on Unmanned Rotorcraft
Phoenix, AZ, 18-20 January 2005

Optimizing to Satisfice: Using Optimization to Guide Users

Robert P. Goldman Christopher A. Miller Peggy Wu Harry B. Funk
John Meisner
SIFT, LLC
Minneapolis, MN
{rpgoldman,cmiller,pwu,hfunk,jmeisner}@sift.info

ABSTRACT

We describe a delegation-based solution to the planning overconstrained UAV missions. SIFT's Playbook™ approach to UAV control allows single operators to request services from multiple-UAV teams, without excessive workload, specialized training, or platform-specific ground control systems. We use the metaphor of a sports team's playbook: a user "calls a play" for a mission, relying on the system to flesh out the details. As in a sports playbook, the user has the freedom to add specifics at will. We have integrated Playbook with Geneva Aerospace's VACS system ground station, to control multiple, heterogeneous, rotorcraft and fixed-wing UAVs. Using PVACS, we have discovered that overconstrained missions are particularly challenging, because it is difficult for operators find appropriate ways to relax constraints. To solve this problem we have added a "best-effort" planning mode to the Playbook. Best-effort plans may be directly useful, or can be used as a guideline in relaxing constraints.

1 Introduction

As Unmanned Air Vehicles (UAVs) become more common, several operational challenges emerge. First, current operator-to-platform ratios on the order of 4-1 will no longer be acceptable. Second, the current practice of providing a dedicated workstation for each vehicle or vehicle type will also be unacceptable when individuals must interact with multiple, heterogeneous vehicles. Operators will need a common interface for multiple vehicles. Third, new usability and training requirements will be imposed. Not all operators will spend months training to be rated for a vehicle, nor will they devote full attention to vehicle management (much less to managing a single vehicle subsystem). Instead, UAVs must be controllable with much less training and while engaged in many other activities.

We have developed an approach to UAV control that addresses these challenges [7]. We use the metaphor of a sports team's playbook to enable both quick and complex variable-initiative control with a variety of UAVs. The user of our Playbook™ "calls a play" to request a service from a team of UAVs. We have implemented this approach in the Playbook-enhanced Variable Autonomy Control System (PVACS) project, which combines SIFT's Playbook interfaces and Geneva Aerospace's Variable Autonomy Control System (VACS).

As we began to make use of the implemented PVACS system, we discovered a new, pressing challenge for users. That challenge is how to handle situations in which the operator has overconstrained the solution space. For example, an operator can now call an "overwatch" play to provide sustained surveillance of a given location, with a given radius, for some window of time. Playbook responds by attempting to develop a plan for the requested play within the user's constraints. But what should happen when the Playbook's planner *cannot* provide a plan that meets these constraints? This is no easy matter, since the planning failure could be a result of UAVs being too far from the target to reach it in time, or of having no UAV available to provide coverage in some sub-interval of the time window (perhaps neither the beginning nor the end, but some piece of the middle).

Simply notifying the user that no plan could be developed is undesirable since this gives the operator no idea what was wrong or how to revise the request. Similarly, the causes of planning failure are generally far too complex to admit any simple UI-based explanation strategy. It may seem that the operator could simply relax the constraints and ask for a different plan, but *which* constraints (surely not those that reflect physical reality)? and in what order? Worse, each attempt to generate a plan for some new, relaxed subproblem, is computationally expensive and frustrating to a human operator whose time may be needed elsewhere.

Throughout our development of the Playbook approach, we have been guided by the fact that the problem of human control of multiple complex, semi-autonomous

agents is one that humans have been wrestling with for milennia. The key difference is that up to know the "agents" controlled have generally been other humans. This long history meands that the methods humanity has developed for "supervisory control" — including delegating and managing tasks — are very familiar to us and reasonably effective. Whenever possible, we try to use human-human delegation and supervisory control as a model for human-machine interaction.

Taking this approach, we see that a subordinate who simply reports that s/he cannot do what the supervisor has requested, without provides additional explanation or alternatives, is not very helpful. On the other hand, one thing that subordinate might do, which would be seen as reasonably helpful, would be to say "I can't do what you've asked, but I can do this thing, which is close to what you wanted." While perhaps less informative than a detailed explanation, this approach has the strength of potentially saving time (if the supervisor is satisfied with the alternate plan) and of providing the basis for negotiation and plan revision if the supervisor wants to adjust it further.

To support this style of interaction, we have developed a relaxed planning method, based on cost-based optimization. Our underlying planning method [9] supports generating plans of minimum cost. Building on this, we have developed a method for generating relaxed plans for the plays in the system. Using this cost-based method, we provide users the ability to request that the system give a "best-effort" plan, when the system cannot meet all the stipulated constraints. The best-effort plan can be produced on an "anytime" basis [1], which means that we can interrupt it at any time, with a best answer so far. This meets our need for prompt response times. One ironic aspect of this is that *satisficing*[1] was originally proposed as a way of overcoming the computational demands of optimal reasoning as part of a program of "bounded rationality," but we are bringing back optimization in service of satisficing.

2 Playbook™ delegation interfaces

Human supervisors of human subordinates have been solving this problem for millennia. Humans generally don't control other human agents at the level of individual movements–roughly analogous to the current state of affairs where UAVs must be controlled via joystick commands–and in fact, there are strong reasons why this is never an effective management strategy (e.g., [1]). Instead, human supervisors delegate tasks to subordinates, or they request services that subordinates determine how best to satisfy. In either case, some objective or partial plan is communicated to the subordinate, perhaps along with constraints on how that objective may be achieved, but simultaneously some authority/autonomy for exactly how to achieve that objective in context is also given to the subordinate.

When interaction with subordinates is effective it is because the effort the supervisor requires to request, instruct, oversee and manage the subordinate(s) is less than s/he would require to do the task without them, and/or because more tasks can be accomplished faster or better via delegation than would otherwise be the case. This saving or extending of the supervisor or requestor's effort comes about specifically because the s/he does not have to plan and execute every detail of the desired behavior at the lowest levels of control available in the system. Generally, more efficiency will be achieved the greater the degree of responsibility for planning and execution the requester can reliably delegate to his/her subordinates. Where the reliability of delegation (in terms of having the desired outcome achieved via a desired method) breaks down, supervisors can actually incur greater workload than if they had done the task themselves, because not only did the supervisors have to instruct and monitor the subordinate, but now they had to repair errors as well.

Domains with a long history of attention to how to communicate intent effectively while simultaneously minimizing supervisor workload include business, construction, military command and control and sports teams. The latter two are of special interest to us for the designing of a control architecture for UAVs. Many sports teams (particularly, but not exclusively, American football teams) achieve complex forms of coordinated behavior by means of a "playbook". A playbook contains predefined patterns of behavior that are understood by all participants on the team. By means of a single, short play name or label, the quarterback or team captain can express his/her intent for a large number of independent actors to behave in a dynamically-changing yet coordinated, focused, constrained and effective fashion. Furthermore, plays can serve as a shared point of reference from which to build novel variations with minimal effort. But it is worth noting that plays also require that the actors be capable of interpreting and apply-ing the play to the context that exists when the play is called. This may be as simple as deciding whether to step left or right depending on what direction the opponent is coming from in a football play, but it precludes completely rote behavior. Actors must be allowed some autonomy about how to perform their delegated roles if there is to be any efficiency gain in the system.

Service requests are like play calling in all but the degree of authority wielded by the requestor. A supervisor is delegating tasks or goals to subordinates who are, presumably, under his or her control and have no other supervisor concurrently tasking them. A requester of service may use the same vocabulary of goals and tasks, but since his/her "commands" are not going to subordinates that are under his/her full and exclusive control, the requester has somewhat less authority than the true supervisor –

[1] Just providing a constraint-satisfying solution to a problem, rather than trying to find an optimal solution.

and somewhat less guarantee that the request will be satisfied completely. This tradeoff may be useful, however, especially if (as is the case for many proposed battlefield UAVs) the subordinates are in high demand and can be used more effectively overall if they are shared among multiple users.

3 Playbook^{TM} architecture

We have been developing a ”playbook” approach to the control of UAVs that strives to build the same kind of relationship between them and a human supervisor as exists between a captain and his/her team. Figure 1 presents the basic architecture for our Playbook. We will provide a brief discussion of the Playbook architecture here, followed by a detailed walkthrough of a specific usage example designed to illustrate both the user's experience and the internal representation and reasoning of the Playbook in the remainder of the paper.

Operators can interact with and request services from automation in highly sophisticated and flexible ways via Playbook. Like the quarterback of a football team, a PVACS operator can command a very complex, very high level ”play”-even one involving a heterogeneous mix of actors (vehicles)-via a fast and simple action. Also like the quarterback, the operator can issue more specific requests about individuals' behaviors, albeit spending more time to ”flesh out” the request, providing more instructions about specifically how it should be satisfied.

A full Playbook system consists of a user interface (UI) communicating with a Playbook server. The user interface allows operators to command automated systems via a shared model of the tasks that can be performed. This task model is both hierarchically and sequentially organized allowing the stringing together of tasks or ”plays” in commandable sequences and/or drilling down within a given play to select alternate performance methods.

3.1 User Interfaces

Decomposing the system into a server and UI allows SIFT to provide different user interfaces to meet the needs of different classes of user. There are three user interfaces that work with the current version of the server: a PDA interface, a console interface, and an engineering interface. The engineering interface is written in the scripting language Python, allowing us to rapidly test new facilities on a wide variety of platforms.

The PDA interface (see Figure 2) is aimed at time-pressured users, notionally including members of small military units, SWAT teams, etc. We expect that the primary interest of these users will be the end results of the plays (in our current scenarios, this would be streaming video of surveillance targets); and that they will not be interested in the details of platform control. Indeed, we would like to provide users who are not rated UAV operators with the benefits of UAVs. Using the PDA interface, such users can call a play to establish surveillance with three mouse clicks, in much less than 15 seconds.

Figure 2: PDA user interface.

The Console UI (see Figure 3) meets the needs of users who have more time at their disposal, and are more interested in the details of platform control. The Console UI also features an interface to the Personal Flight Planning System (PFPS) suite of tools, and can use PFPS' FalconView[2] flight mapping software to display Playbook-generated routes (see Figure 4). With FalconView, Playbook users can project the execution of mission plans using FalconView's rehearsal capability.

3.2 Playbook server

The Playbook server has three major components. The first is the server layer itself, which provides the services to clients, tracks client sessions, etc. We will not discuss this component in any detail here. The two major components are the Analysis and Planning Component (APC), which translates user play requests into executable mission plans, and the Executive. The Executive manages communication with any connected ground control station(s), and does any necessary closed-loop control: reading platform status information from the ground control system (GCS), and keeping the mission plan on track. See Figure 1. All three of these components draw upon the shared task model.

The Playbook Analysis and Planning Component (APC) evaluates the feasibility of alternate methods of satisfying requested plays. When given a high-level play request, the APC selects among various feasible methods, issues instructions to the ground control system execution environment (see below) and monitors for necessary revisions during performance. When given lower-

[2] `www.falconview.org`

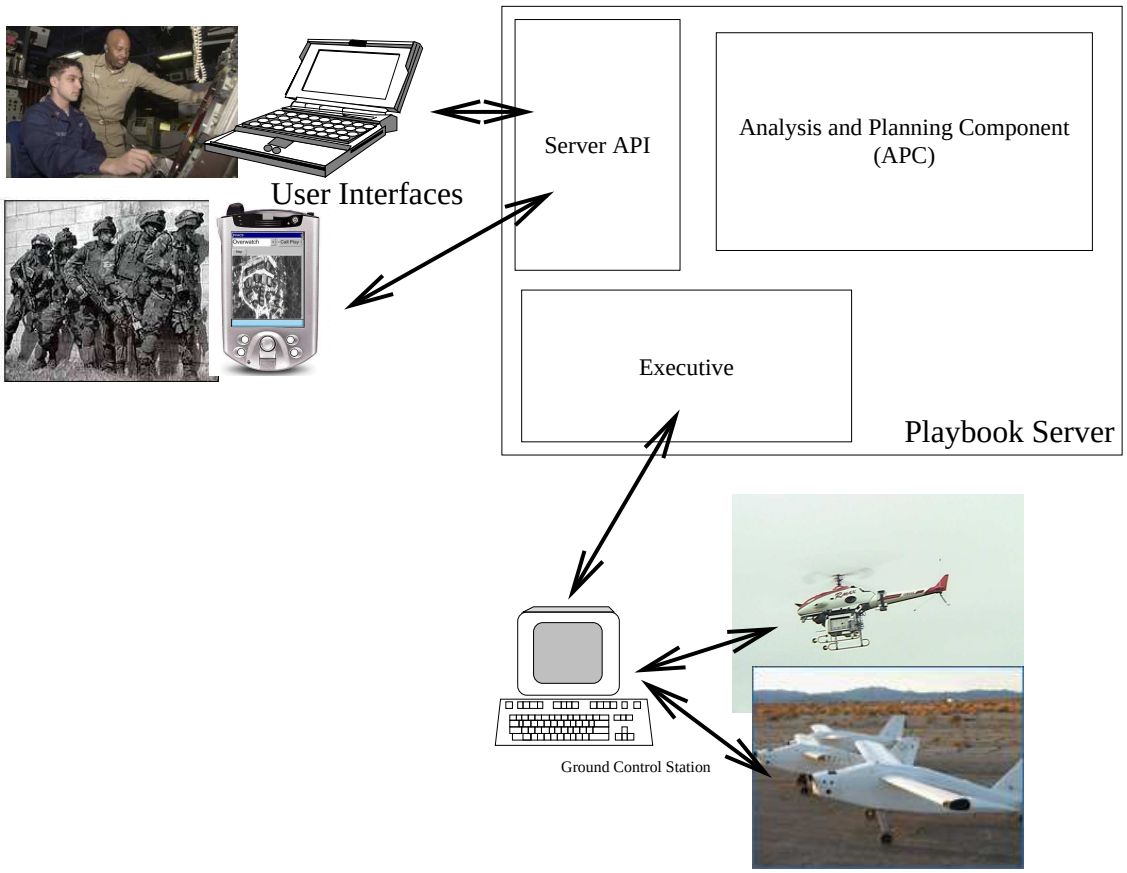

Figure 1: Playbook system architecture.

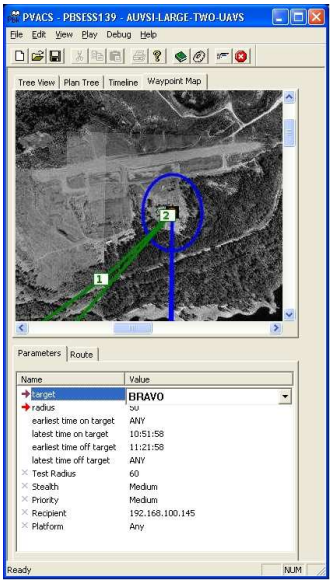

Figure 3: Console user interface.

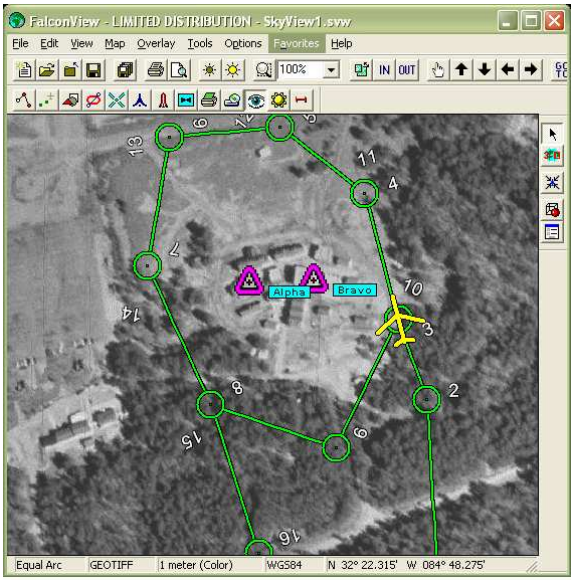

Figure 4: FalconView working with Playbook console UI.

level, more specific and detailed requests (such as a specific platform to use, a specific path to be followed, or specific scan patterns to use), the APC reviews them for feasibility and either (a) reports when requested actions are infeasible, (b) passes 'validated' user-requested plans to the execution environment and monitors their performance, or (c) fleshes out high level operator requests (like "Overwatch") to an executable level (for example, choosing among available platforms, selecting waypoints for ingress and egress, identifying sensor steering parameters, etc.) within the constraints the operator has imposed.

The APC is built on top of the SHOP2 planning system, which was developed at the University of Maryland, College Park [9]. As part of our work on Playbook, we have added facilities for temporal reasoning (scheduling), recovery of state trajectories, enhanced plan library facilities, etc. We expect to make these new facilities available through the SHOP2 open source effort (see `www.sourceforge.net/projects/shop`). SHOP2 provides a plan optimization facility, which we have employed in the work reported here. We will discuss this in greater detail later.

The APC provides pre-mission planning (and can provide in-mission replanning), however it does not close the loop around the UAV autopilots. Closed-loop control is the responsibility of the Executive component of the Playbook server.[3] The Executive tracks UAV state, as transmitted by the GCS, executes commands through the GCS during flight, and manages translation and download of mission plans. Since we are primarily concerned with the planning and user interface components of the system, we will not discuss the Executive further in this paper.

Playbook by itself does not include inner-loop control capabilities. In the program in question, we have been working with Geneva Aerospace's VACS™control system, the Playbook Executive controlling the UAVs through the VACS GCS.[4] Geneva Aerospace's Variable Autonomy Control System (VACS) provides a robust integrated control architecture enabling a single operator to control multiple UAVs[2].

The VACS architecture includes an Internet Protocol (IP) based network-centric communications package linking teams of UAVs and remote operator workstations. VACS fuses sensor and database information to autonomously adjust flight profiles to avoid terrain and mid-air collisions. Once VACS adjusts the flight profile, however, it relies on the human operator to decide if the mission plan must be revised (e.g., perhaps because time on target constraints have been violated). Further, the human user must make all mission level decisions and interact with the various control levels. VACS has recently been extended from its original fixed-wing basis to also control small rotorcraft UAVs[11].

Integration with Playbook advances VACS to higher levels of autonomy by providing automated means of developing and adjusting plans to achieve mission objectives. Playbook possesses a hierarchical understanding of the operational intent and specific target tasking, and can provide high-level commands to the vehicle and sensor control systems following the command structure already in place in the VACS. In essence, VACS provides a "library" of control execution behaviors from which plays, as well as more complex sequences of plays which form overall mission plans, can be composed. The integrated Playbook + VACS (PVACS) capabilities are particularly relevant to projected operational concepts where busy and/or non-rated operators must supervise teams of heterogeneous vehicles. PVACS' combination of very high level and variable autonomy control will allow busy operators to command sophisticated, coordinated behaviors simply and rapidly and/or allow operators with more time or training to impose highly specific commands to customize vehicle behavior to their exact needs.

More details about the Playbook architecture and various UI designs are provided in [6]. In previous work, we have developed prototype playbooks for UCAV teams [8], Tactical Mobile Robots [3], and the RoboFlag game (a robotic version of "capture the flag") [10]. The remainder of this paper will focus on issues in providing mission plan tasking with the Playbook, and more particularly on our use of optimization for this purpose.

4 The Challenge of overconstrained missions

The basic Playbook concept puts an AI planner in service of a user, attempting to emulate an intelligent subordinate. For example, in our application for mixed UAV reconnaissance teams, we allow a user to say "I would like surveillance of *this* area for *that* period of time." In response, the UI will send a request to the planner. The planner will generate a plan to meet this request, tasking one or more UAVs to fly to the surveillance area, point their cameras, etc. This frees the user from the need to identify what UAVs are available, find an acceptable flight path, etc. If the user likes the proposed plan, it can be downloaded to the GCS for automatic execution.

Until now, however, when the Playbook planner was unable to satisfy a play request, it effectively said, simply, "no." Even the system developers found it very difficult to decide why a play request had failed. We expect that a system that fails this way will generate the same sort of frustration (and inefficiencies) as would a subordinate who, when asked to perform a task, simply responds "I can't." It is certainly useful for the superior to know if his

[3] Properly speaking, the Executive does not directly command the UAVs, but works through the UAVs' GCS.

[4] We are concurrently working to add Playbook facilities to IA Tech's MIIIRO immersive multi-UAV simulation (`www.ia-tech.com`).

orders cannot be obeyed,[5] but it would be far more useful to know *why* s/he can't and know what might be feasible in moving toward the superior's goals.

In even moderately complex domains, it can be very difficult to determine why a request cannot be satisfied. Consider a surveillance request for a particular location. Here are three reasons why such a request might be infeasible:

1. There is no UAV that is able to reach the target area by the time that the user wants surveillance to start.

2. It is possible to get a UAV to the target area in time, but that UAV must leave before the end of the coverage period the user has requested (and there is no other UAV available to take over the job).

3. There is a UAV that can get to the target in time, but it must return to base before a second UAV can get to the target area to take over surveillance.

These are a few simple cases that have to do with the time bounds on the request; there could be many other reasons for mission failure (no UAV available, target too far away, etc.). Note also that there is no crisp, easily mechanizable distinction between the different explanations for failure. The additional investigation to disentangle the causes may be quite burdensome. For example, one might need to find the flight distances to the target for all available UAVs, and then fuse this information with information about the availability windows of each platform, in order to determine why no UAV could reach the target in time. Indeed, our work on this facility was prompted by difficulties we ourselves had in calling plays.

Note that there is no easy, obvious fix to the problem of search failures. There are potentially a large number of constraints on any play, and there is no obvious indication of which constraint is to be relaxed. It might be possible to address this problem, by requiring the user to supply some form of objective function over the constraints, but such objective functions are very difficult for users to provide.

To take a somewhat ridiculous example, should we relax the constraint on the time on target, or should we change the target location? Worse, there is no guarantee that relaxing a *single* constraint will be sufficient. Computationally, this is a very difficult problem. Proving plan non-existence is more difficult than proving plan existence. Furthermore, searching for a set of constraints that must be relaxed would involve searching over the already difficult planning problem. And worst of all would be to require the user to search through this space via a tedious combination of "Can you do this? No? Well, how about this?" questions.

[5]The agreeable subordinate who simply assents to every request, feasible or not, can wreak havoc in an organization.

Given these difficulties, we have turned back to our original inspiration: attempting to have our systems accept responsibilities in ways similar to the ways a human would accept delegation. Accordingly, we have built our system to be able to effectively say "I can't give you exactly what you want, but here's something I think is close that I *can* do."

5 Best-effort planning using optimization

Let us return to the three examples above, which we have tested in the current Playbook. Figure 5 shows an aerial photograph of the area of interest (around the McKenna MOUT facility), together with two UAVs, one a rotorcraft, and one fixed-wing. For simplicity, all of our examples will use these two UAVs, starting at these positions (but with varying periods of availability), and all will involve surveillance of the same target (but for different periods). Figure 6 gives details of the time constraints for the three problems.[6] Times are given in minutes:seconds (optionally with hours).

As the Playbook interface works now, if a user requests a play that is overconstrained, s/he will get a message reporting that no plan can be found that meets his or her constraints. Then the user will be offered the opportunity to get a best-effort plan for the same request. Consider our first case, where there are two UAVs suitable for performing the mission, but neither of them can get on target as quickly as the user has requested. In this case, if the user requests a relaxed plan, then the system will provide a plan where the scanning begins two minutes later than requested, but that otherwise satisfies the request: the rotorcraft will take off and fly to the target immediately and begin surveillance, until the end of the desired period, and then fly back to its starting position (the fixed-wing aircraft is not needed).

How are such "best effort" plans generated? To understand this, we must understand a little bit about the functioning of the Playbook planner (APC). The APC is a *hierarchical task network* (HTN), or *decomposition* planner. Such a planner takes task descriptions (such as the input play request in PVACS), and decomposes it, step by step, matching against *method* definitions, until it reaches a fully fleshed-out plan, where all the tasks have been specified down to a set of defined, executable primitives.

Three complications make this more than a simple exercise in tree-walking:

1. There may be multiple different methods for a single task, and the planner must choose between them to find one that is feasible (and possibly optimal).

[6]Note that the times in question are often rather short, possibly unrealistically so. These were chosen to be so short for demonstration purposes only, because our simulation runs in 1:1 time. Longer time constants would not change the way our techniques work, but would make for runs that are tedious to watch.

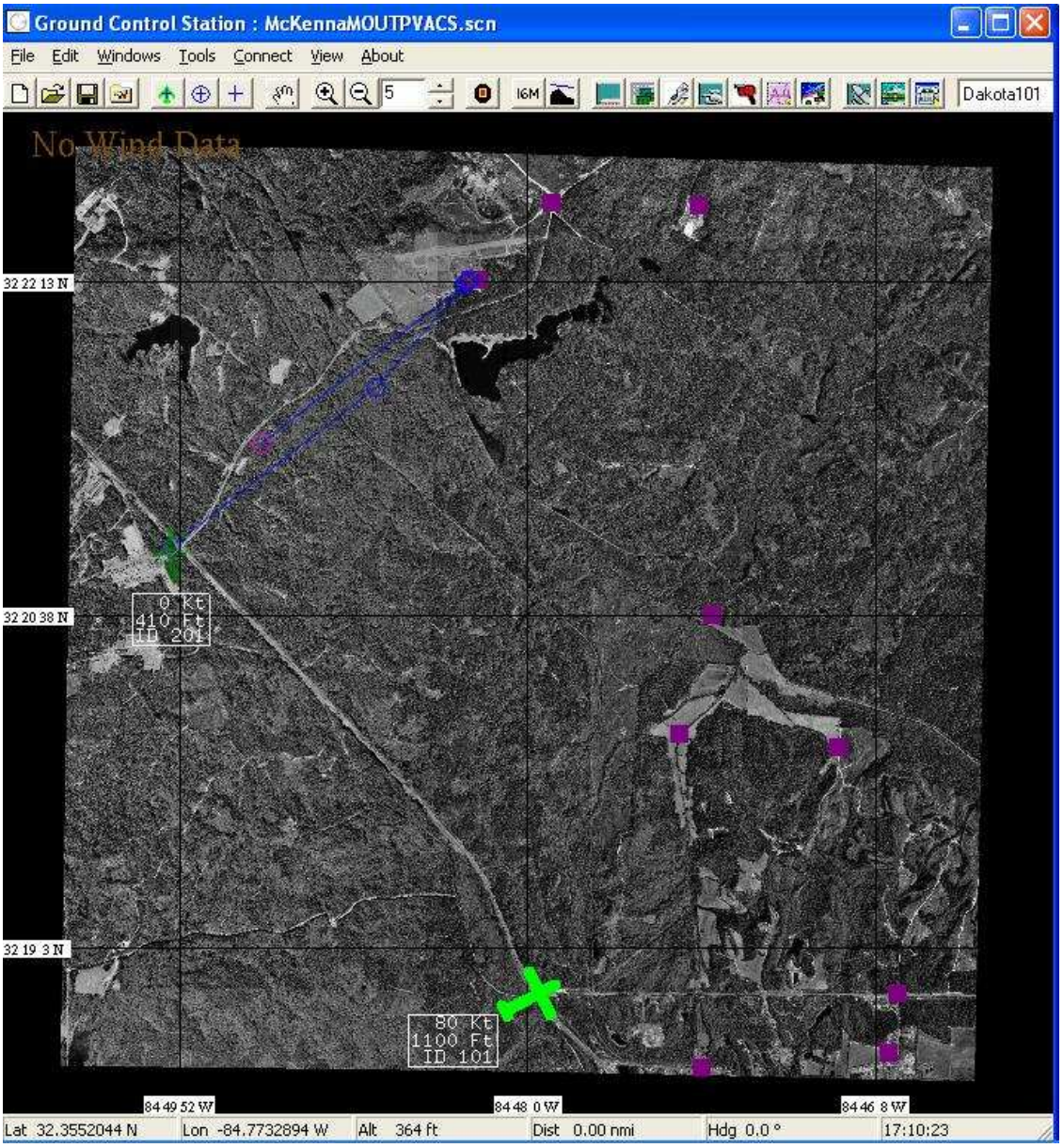

Figure 5: Area of simulated operations at the McKenna MOUT (screenshot from VACS GCS).

UAV	availability	Requested surveillance
101	$t+$ 8:20 – $t+$ 90:00	$t+$ 1:00 – $t+$ 6:00
201	t – $t+$ 16:40	

(a) No UAV available early enough

UAV	availability	Requested surveillance
101	t – t + 40:00	$t+$ 3:00 – $t+$ 40:00
201	t – $t+$ 20:00	

(b) UAVs not available long enough

UAV	availability	Requested surveillance
101	$t+$ 30:00 – t + 4:00:00	$t+$ 4:00 – $t+$ 44:00
201	t – $t+$ 30:00	

(c) UAVs available to cover the entire period, but no smooth handoff.

UAV	Travel time to target
101	approx. 3:00
201	approx. 5:00

Travel times (all problems)

Figure 6: Three overconstrained problems. UAV tail number 101 is the fixed-wing aircraft and 201 is the rotorcraft. All times are given in minutes and seconds.

2. There may be variable-binding decisions that have to be made, such as choosing between different UAVs to perform a sortie. Again, the planner must find a feasible (resp. optimal) solution.

3. The planner must check that the sequence of actions generated corresponds to a feasible state trajectory. This is complicated because actions may have side-effects, and tasks may destructively affect each other. For example, if a plan were to cause a UAV to pass over a location occupied by the enemy, it might destroy the desired element of surprise for a following UAV.

For those more familiar with numerical techniques, it is a reasonable analogy to think of a planner as a constraint solver, like a linear programming (or mixed-integer) package, but one that operates in a primarily discrete space, rather than a continuous one.

The current planner primarily operates in a constraint satisfaction, or satisficing mode. I.e., it returns the first plan that it can find that meets the user's requirements. Because the planning problem is not tractable, we time-limit the planner. In theory, this could cause us to fail to find solutions to solvable problems. In practice, we find that the problems we are working on either terminate fairly rapidly, with either success or failure, or fail after a great deal of search. For practical purposes, it is acceptable to terminate after a fixed period of time, although the precise period of time differs between applications.[7]

Our approach to best-effort planning is to relax the user-imposed constraints on the objective of a task, associate a cost function to plans, and find the best available plan (within a fixed time bound). The SHOP2 planner, which we have used as the basis for the APC is able to perform a very limited form of optimization, which minimizes the sum of a user-defined cost function, applied to each of the steps of a plan, P. I.e. $c(P) = \sum s \in Pc(s)$, and we choose $\mathrm{argmin}_{P \in \omega}\, c(P)$. This optimization is performed using branch-and-bound, and can be time-limited.

There were two challenges that faced us in providing best-effort search. The first was to overcome the limits of SHOP2's optimization functions, and the second was to identify appropriate cost functions. The current optimization functions, based on functions, are not suitable for our purposes, since our optimizations must be done over state trajectories. For example, for our overwatch play, the cost should be some function of the portion of the desired time that the target is *not* being watched. The cost is a function of the set of states, not of the actions taken by the system. Using higher-order functions, it was possible for us to modify the planning system to permit us to associate state trajectory functions with plans.

With the technical implementation out of the way, we turned our attention to composing cost functions. In gen-

[7] The Playbook integrated with MIIIRO has different time behaviors than the PVACS Playbook.

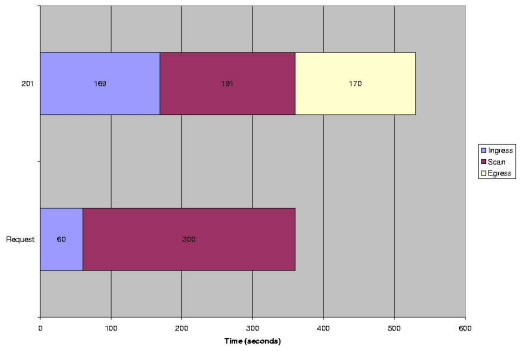

Figure 7: Relaxed plan for case with no UAV available early enough.

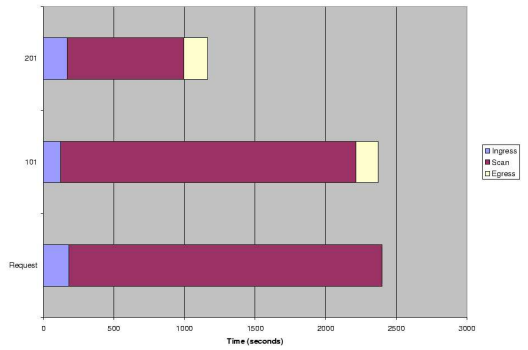

Figure 8: Relaxed plan for case with UAVs not available long enough.

eral, it is a very difficult problem to compose utility functions for complex optimization (see, for example [5]), and complex optimization systems often yield solutions unexpected to (and often unwelcome to) their users [4]. The problem at hand, however, is far more simple. We are not trying to find solutions that optimize utilities; rather we are trying to find solutions that will come close to the user's desires, or will guide the user. For example, even if the user doesn't like the plan that results from relaxation in the preceding example, it will provide some guidance in terms of how long it might take to reach the target, allowing the user to reconsider his or her tasking. Probably the most difficult problem in formulating an objective function is to handle tradeoffs beteween *competing* objectives: how much of one objective are you willing to pay to do better on another. But this problem largely does not arise in our context, since we are just trying to get as close to feasible as possible, not close to optimal. We have found that appropriate optimization functions for the reconnaissance missions involve simply penalizing the mission for parts of the surveillance window that are not covered.

Using such optimization functions, give the results shown in Figures 7, 8, and 8 for our sample problems. In these figures, the lines marked "request" show the period of surveillance requested by the user. In each case, the central band shows the surveillance part of the mission (or the surveillance request).

We find that users presented with this kind of information can readily understand why their requests could not be met, and are in a position to modify them appropriately. For example, a user who was attempting to get early coverage for five minutes, as in Figure 7, might see this, recognize that no UAV was able to reach the target soon enough, and either accept the plan or, if duration was what s/he cared about, more than time on target, modify the original request to be five minutes' surveillance start-

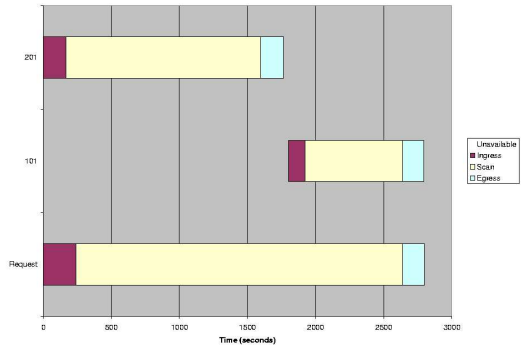

Figure 9: Relaxed plan for case with UAVs available to cover the entire period, but no smooth handoff.

ing at t + four minutes, rather than t + one minute.

6 Conclusions

In this paper, we have presented an approach to providing user support for overconstrained mission planning in the context of a Playbook-style delegation interface. Our approach uses optimization to provide feasible, close approaches to the user's preferred solution. We have incorporated this technique into a Playbook control system for UAVs (both rotorcraft and fixed-wing), that is integrated with Geneva Aerospace's Variable Autonomy Control System.

In future work, we will extend the approach and better integrate it with the user interface. Currently, while users can request best-effort plans for overconstrained problems, the user interface does not help them compare the relaxed plans with the original requests. We will design additional display elements to show how the relaxed plans deviate from requested, for example, showing desired timelines matched against the planned ones, showing actual play parameters matched against the desired choices, and highlighting the differences. We will also extend the relaxed planning approach itself to cover a wider variety of problems, including multiple concurrent plays, and more play types.

Acknowledgments

Research described in this paper was funded by The Defense Advanced Research Projects Agency (DARPA) and the U.S. Army Aviation and Missile Command (AMSAM-AC-RD-AY), through contract DAAH01-03-C-R177. The opinions expressed in this paper do not reflect the positions of the funding agencies or the U.S. government.

References

[1] T. Dean and M. Boddy, "An Analysis of Time-Dependent Planning," in *Proceedings of the Seventh National Conference on Artificial Intelligence*, pp. 49–54, 1988.

[2] D. S. Duggan. *Demonstration of an Integrated Variable Autonomy UAV Flight Control System*. Phase II SBIR Final Report, January 2001. Air Force contract number AFRL-HE-WP-TR-2001-0035.

[3] R. P. Goldman, K. Z. Haigh, D. J. Musliner, and M. J. S. Pelican, "MACBeth: A Multi-Agent Constraint-Based Planner," in *Constraints and Artificial Intelligence Planning: Papers from the AAAI Workshop*, A. Nareyek, editor, number WS-00-02 in AAAI Technical Report, pp. 11–17. American Association for Artificial Intelligence, AAAI Press, 2000.

[4] G. Jamieson and S. Guerlain, "Operator Interaction with Model-Based Predictive Controllers In Petrochemical Refining," in *Proceedings of the Human Performance, Situation Awareness and Automation Conference*, pp. 172–177, Marietta, GA, 2000, SA Technologies.

[5] R. L. Keeney and H. Raiffa, *Decisions with Multiple Objectives*, John Wiley and Sons, Inc., New York , NY, USA, 1976.

[6] C. A. Miller, R. P. Goldman, H. B. Funk, and R. Parasuraman, "Delegation as a Model for Human-Automation Interaction," in *Proceedings of the 3rd International NASA Workshop on Planning and Scheduling for Space*, October 2002.

[7] C. A. Miller, R. P. Goldman, H. B. Funk, P. Wu, and B. Pate, "A Playbook Approach to Variable Autonomy Control: Application for Control of Multiple, Heterogeneous Unmanned Air Vehicles," in *American Helicopter Society 60th Annual Forum Proceedings*, pp. 2146–2157, Alexandria, VA, June 2004, American Helicopter Society.

[8] C. A. Miller, M. J. S. Pelican, and R. P. Goldman, ""Tasking" Interfaces for Flexible Interaction with Automation: Keeping the User in Control," in *Proceedings of the Conference on Human Interation with Complex Systems*, April - May 2000.

[9] D. Nau, T.-C. Au, O. Ilgami, U. Kuter, H. Muñoz-Avila, J. W. Murdock, D. Wu, and F. Yaman, "Applications of SHOP and SHOP2," Technical Report CS-TR-4604, UMIACS-TR-2004-46, Computer Science Department, University of Maryland, College Park, MD, June 2004. `http://www.cs.umd.edu/%7Enau/papers/nau04applications.pdf`.

[10] R. Parasuraman, S. Galster, and C. A. Miller, "Human Control of Multiple Robots in the RoboFlag Simulation Environment," in *Proceedings of the 2003 Meeting of the IEEE Systems, Man and Cybernetics society*, October 2003.

[11] B. Pate, "A Rotorcraft Adaptation of Geneva Aerospace's Variable Autonomy Control System," in *American Helicopter Society 60th Annual Forum Proceedings*, pp. 2146–2157, Alexandria, VA, June 2004, American Helicopter Society.