Proceedings of the
47th IEEE Conference on Decision and Control
Cancun, Mexico, Dec. 9-11, 2008

Team Task Allocation and Routing in Risky Environments under Human Guidance

David A. Castañón and Darryl K. Ahner

Abstract— **In this paper we design coordination policies for unmanned vehicles that select and perform tasks in uncertain environments where vehicles may fail. We develop algorithms that accept different levels of human guidance, from simple allocation of priorities through the use of task values to more complex task partitioning and load balancing techniques. The goal is to maximize expected value completed under human guidance. We develop alternative algorithms based on approximate dynamic programming versions appropriate for each level of guidance, and compare the resulting performance using simulation results.**

I. Introduction

The purpose of this paper is to develop approaches for adaptive replanning and routing of unmanned vehicles to spatially distributed tasks in scenarios where there is risk of vehicle loss. Such situations arise typically in complex environments where adversarial actions will try to prevent vehicles from pursuing tasks. We consider a class of problems where vehicles follow paths to perform a sequence of tasks, and the choice of path affects the probability that a vehicle may be destroyed before completing its tasks. The presence of risk in these problems may require the assignment of multiple vehicles to important tasks. Of particular interest is the development of algorithms that can accept a varying level of human guidance, allowing for detailed control of the individual vehicles. In particular, we are interested in two modes of human guidance: a nearly autonomous mode where values are assigned to tasks, and a second mode where tasks are partitioned across vehicles and values are assigned to tasks.

From a dynamic perspective, traversal of a segment of a path is an observable event with two random outcomes: vehicle destruction or arrival at the end of the segment. This suggests the use of stochastic dynamic programming techniques that observe outcomes and adapt assignments and routes accordingly. However, the resulting stochastic dynamic program is cumbersome to solve, and not suitable for real-time replanning.

In this paper, we propose an alternative approach based on an expected value formulation that incorporates the effects of risk. Our approach mimics model-predictive control [9] in that this expected value formulation is used to develop a planned course of action. Once information is collected concerning vehicle and task status, the plan will be recomputed using the most recent information. The resulting optimization problem is NP-hard, so our proposed algorithms are based on approximate solution of the resulting problem using approximate dynamic programming techniques.

Much of the previous work on stochastic scheduling problems has focused on scheduling of vehicles subject to random demands. The extensive work of Powell and his colleagues [1], [2], [3] illustrates these approaches. In these formulations, vehicles are reliable carriers that respond to random demands. In contrast, we consider scenarios where vehicles are lost, and other vehicles must compensate for these losses. In our previous work [4], we considered routing and scheduling of a single vehicle in risky networks, and developed a class of approximate dynamic programming techniques known as rollout algorithms that compensated for vehicle failures. The problem considered in this paper is an extension of the formulation in [4] to include multiple vehicles.

Other related work includes a class of stochastic resource assignment problems that explicitly model the risk of asset loss [5], [6]. In these models, multiple assets are assigned to individual tasks to hedge against potential loss of assets. Our problem generalizes these models to include decisions concerning routing as well as assignment.

The rest of this paper is organized as follows. Section II presents a formulation of the risky vehicle routing problem that will be used for algorithm development. Section III introduces two classes of approximate algorithms to solve the models in Section II. Section IV discusses the implementations of these algorithms under different levels of human guidance. Section V presents experimental evaluations of the alternative algorithms, and discusses further experiments. Section VI includes concluding remarks.

II. Problem Statement

Define a directed graph $G = (N, A)$, where the nodes correspond to possible vehicle waypoints or tasks to be done, and the arcs represent segments connecting these waypoints. Associated with each node i is a value V_i of reaching the node, representing completion of the task associated with the node. Each arc (i, j) has a probability p_{ij} that a vehicle traversing it will successfully reach the destination node j. For simplicity, this probability is assumed independent of the vehicle. We assume that the events of successful vehicle traversals of arcs are mutually independent across vehicles and arcs. Furthermore, if the same vehicle traverses the same arc twice, these events are also independent. We also assume

This work was supported by AFOSR under grant FA9550-07-1-0361 and by ODDR&E MURI Grant FA9550-07-1-0528

D. Castañón is with the Department of Electrical and Computer Engineering, Boston University, Boston, MA dac@bu.edu

D. Ahner is with the Department of Mathematical Sciences, United States Military Academy, West Point, NY darryl.ahner@usma.edu

for simplicity that the time required to traverse each arc is normalized to 1. We can introduce additional nodes as required to enforce this.

Assume that there are K vehicles present, initially located at known nodes $n^1, \ldots, n^K$. For each vehicle k, one must select a path P^k on the directed graph, which will include a specified number of arcs M. A path P^k will be denoted by a sequence of nodes $(n_0^k, \ldots, n_M^k)$ where $(n_i^k, n_{i+1}^k) \in A$ for $i = 0, \ldots, M-1$ and $n_0^k = n^k$. Associated with the path for vehicle k is the probability that the vehicle will complete the first s segments, defined as

$$p_s^k = \prod_{l=0}^{s-1} p_{n_l^k, n_{l+1}^k} \tag{1}$$

As notation, we denote $p_0^k = 1$, and $p_{-1}^k = 0$.

Note that it is desirable to assign multiple paths to traverse a valuable node, because each vehicle is not guaranteed to reach a node to complete the task. Consider a task at node n. For each path P^k, denote $q^k(n)$ to be the first position in path P^k that the node is visited, where $q^k(n) = -1$ is used to indicate that path P^k does not visit node n. The independence assumptions lead to the following result:

Lemma 2.1: Given paths $P^k, k = 1, \ldots, K$, the probability that node n will be visited by a vehicle is given by

$$P(n) = 1 - \prod_{k=1}^{K} (1 - p_{q^k(n)}^k) \tag{2}$$

The objective of the risky path planning problem is to select K paths of length M so as to maximize the expected node value completed, given by

$$J = \sum_{n \in N} V_n P(n) \tag{3}$$

The above problem formulation is an optimization problem over deterministic paths, as opposed to a feedback formulation where the choice of paths would depend on observation of successful traversals. Note that the objective is to have at least one vehicle visit each node, with the assumption that when a vehicle visits a node, the node's value is collected, and subsequent visits to the same node collect no additional value. The resulting deterministic optimization problem is complex, as detailed next.

Lemma 2.2: The risky path planning problem is NP-hard.

Proof: Instances of this problem with $M = 1$ and dense graphs become instances of the weapon target assignment problem which was shown to be NP-hard by Lloyd and Witsenhausen [8]. ■

There are simpler instances of the problem, but even these instances tend to be hard to solve. For instance, if risk is removed, so that $p_{ij} = 1$ for all arcs $(i, j) \in A$, the resulting problem is a deterministic problem. However, the constraint that only one vehicle can collect the value of a task over all time creates nonlinear couplings that prevent the use of network optimization techniques with polynomial complexity. This emphasizes that the computational complexity arises from both coupling the performance of multiple vehicles, and hedging against the possibility of vehicle loss.

Note that there is no explicit cost associated with the loss of a vehicle, but simply an opportunity loss. The formulation can be easily extended to include the cost of vehicle loss.

The single vehicle version of this problem is similar to the stochastic scheduling problems studied in [4]. For dense graphs with probabilities of success that depend only on the end node, the optimal single vehicle path is given by the quiz heuristic that maximizes at step the node n with maximal index

$$Q(n) = \frac{p_{in} V_n}{1 - p_{in}}$$

where i is the current node of the vehicle, n are indices of the nodes that have not been visited yet which can be directly reached from node i, and p_{in} is the probability of successfully traversing link (i, n). Note that this heuristic is an optimal strategy for dense graphs where the probability of successful traversal depends only on the end node n, as noted in [4]. For sparse graphs, or for graphs where the success probability depends on both start node and end node of arcs, [4] describes approximate dynamic programming techniques based on rollout algorithms to solve these problems, and shows that the average performance of these algorithms is within 5% of the optimal stochastic dynamic programming solution.

III. Solution Algorithms

Given the complexity of exact solution of the risky vehicle routing problem, we focus on developing approximate solutions using the rollout approaches introduced in [4]. Note that each of these algorithms is intended to be a centralized algorithm, where knowledge of the full information about the problem details is assumed. Although distributed algorithms are of interest, they are beyond the scope of the current paper. We describe two solution algorithms below, originally introduced in [7].

A. Coordinate Ascent Algorithms

The first approach we propose is a direct extension of the single vehicle rollout algorithms described in [4]. Our approach is to optimize (3) by varying the path selected by a single vehicle, keeping the paths of the other $K-1$ vehicles fixed. Alternating over the various vehicles, one defines a coordinate ascent approach that improves the objective at each iteration, and stops when a full cycle of optimizations fails to improve the objective. The resulting single vehicle optimization problems are solved using a direct extension of the rollout algorithms of [4], as follows.

For solving the subproblem for vehicle j, define the reduced values of nodes $\tilde{V}_n$ as follows

$$\tilde{V}_n = V_n \prod_{k=1, k \neq j}^{K} (1 - p_{q^k(n)}^k) \tag{4}$$

This is the expected value of node n that is not completed by any of the vehicles other than j.

Our approach will be based on approximate dynamic programming using rollout techniques. Consider a single

vehicle j. The problem of selecting a path for vehicle j, given that the other vehicles' paths are known, becomes a dynamic programming problem. Selection of the first link out of the initial node n_0^j is done following Bellman's equaton:

$$n_1^j \in \arg \max_{(n_0^j,n)\in A} p_{n_0^j,n}(\tilde{V}_n + R(n, M-1))$$

where $R(n, M-1)$ is the maximum expected reward (in terms of reduced values) that vehicle j can collect in the subsequent $M-1$ transitions. The rollout algorithm is an approximate dynamic programming technique, that computes an approximation $\tilde{R}(n, M-1)$ by using a suboptimal policy. The rollout algorithm for vehicle j is described below:

1) Enumerate the possible next nodes n_1 out of node n_0^j, the initial node for vehicle j, examining the arcs in A.
2) For each possible next node n_1, let $s = 1$ and compute the approximate reward-to-go $\tilde{R}(n_1, M-1)$ as follows:
   - Given n_s, select n_{s+1} to maximize over the feasible next nodes in A, the index
   $$Q(n) = \frac{p_{n_s,n}\tilde{V}_n}{1-p_{n_s,n}}$$
   where $\tilde{V}_n$ is set to zero if n is already in the evaluated path.
   - Set $s = s+1$, and repeat the above step until $s = M$.
   - Evaluate the value attained by the rollout path out of n_1, $\tilde{R}(n_1, M-1)$, recursively backwards as
   $$\tilde{R}(n_M, 0) = \tilde{V}_{n_M} \quad (5)$$
   $$\tilde{R}(n_s, M-s) = \tilde{V}_{n_s} + p_{n_s,n_{s+1}}\tilde{R}(n_{s+1}, M-s-1) \quad s = M-1, \ldots, 1 \quad (6)$$
3) Select the next node n_1^j from the feasible next nodes n_1 to maximize the expected rollout value from the initial position, given as
$$R(n_0^j) = \tilde{V}_{n_0^j} + p_{n_0^j,n_1}\tilde{R}(n_1, M-1) \quad (7)$$
4) We now have the path for the first step, (n_0, n_1). To extend this path to the end, set s = 1, 2, ..., M-1, and repeat the above procedure starting from n_s^j with $M-s$ steps to go, setting $\tilde{V}_{n_s^j} = 0$.
5) Compute the overall reward of all K vehicles using (1)-(3).

The above procedure is initialized by selecting a vehicle, assuming no other vehicle is used, so all the node values are set to the original values. The algorithm is then repeated for each vehicle j, until a full pass of all K vehicles yields no further improvement in overall reward achieved.

Note that, for each vehicle, there is no further value in revisiting a node that was previously visited by the same vehicle. This is because the event that the vehicle is still present in the second visit requires that the first visit was successful (i.e. the vehicle was not destroyed before the first visit), and so the second visit is superfluous. This is why the reduced values are set to 0 once a path is extended to include a node.

The algorithm presented above is a vehicle-by-vehicle improvement algorithm, where a subset of decision variables are modified at a time. As such, it converge to solutions where no single vehicle modification can improve the solution, but multi-vehicle perturbations would result in improvement. Furthermore, the convergence value may depend on the order in which vehicles are considered. In our limited experiments, we have found little difference in convergence value based on the vehicle order explored.

B. Multi-vehicle Rollout Algorithms

The algorithm in the previous subsection considered modifying complete paths for each vehicle, one vehicle at a time, while keeping the paths of other vehicles the same. An alternative approach is to use approximate dynamic programming across all vehicles simultaneously, obtaining individual trajectories for each vehicle.

The idea for the multi-vehicle rollout algorithms is to select the transtion arc for each vehicle in a round-robin manner, using approximate dynamic programming based on rollout techniques. We artificially order the decisions of each vehicle, and consider the sequential decision problem one vehicle at a time for each time step, across M time steps. Thus, we create $M*K$ decision steps, for K vehicles, where vehicle k acts at decision steps t such that $k = (t-1) \mod K + 1$. We summarize the algorithm below:

1) Number each vehicle $1, \ldots, K$. Start with step $t = 1$, time $s = \lfloor t-1/K \rfloor + 1$ and vehicle $j = (t-1) \mod K + 1$. Initialize the reduced values $\tilde{V}_n = V_n$.
2) Enumerate the possible next nodes n_t out of node n_{s-1}^j, the current node for vehicle j at time $s-1$.
3) For each possible next node n_t, compute the approximate reward $\tilde{R}(n_t)$ as follows:
   a) Set $n_s^j = n_t$, and decrement the value $\tilde{V}_{n_t} = \tilde{V}_{n_t} * (1 - p_{n_{s-1}^j,n_t})$.
   b) Select the next vehicle to move as $k = t \mod K + 1$, and select its move n_s^k at stage $s = \lfloor t/K \rfloor + 1$ to maximize over the feasible next nodes in A, the index
   $$Q(n) = \frac{p_{n_{s-1}^k,n}\tilde{V}_n}{1-p_{n_{s-1}^k,n}}$$
   where $\tilde{V}_n$ is set to zero if n is already in the evaluated path for vehicle k.
   c) Decrement the value $\tilde{V}_{n_s^k} = \tilde{V}_{n_s^k}(1 - p_{n_{s-1}^k,n_s^k})$. Increment $t = t+1$, and repeat the above two steps until $t = K*M$.
   d) We have now selected complete paths for each vehicle. Evaluate the value attained by the rollout path out of n_t, $\tilde{R}(n_t)$, using (1)-(3).
4) Select the next node n_s^j from the feasible next nodes n_t to maximize the expected rollout value from the initial position, $\tilde{R}(n_t)$.
5) Increment $t = t+1$. Set time $s = \lfloor t-1/K \rfloor + 1$ and vehicle $j = (t-1) \mod K + 1$. Initialize the reduced values $\tilde{V}_n$ using the partial paths computed

for nodes $n_r^k, r < s$ and $n_s^k, k < j$ using (1)-(2) to compute $P(n)$ for each node n. Set the reduced values as $\tilde{V}_n = V_n(1 - P(n))$

6) Repeat steps 2-5 above until $t > K * M$. Compute the overall reward of all K vehicles using (1)-(3).

The multi-vehicle rollout algorithm described above has significant computation advantages for approximate dynamic programming. Although the algorithm has been described to compute the full rollout paths $n_s^k, k = 1, \ldots K; s = 1, \ldots, M$, the algorithm can be stopped once there is a single stage of decisions available, corresponding to a single stage of the approximate dynamic programming algorithm. The current decisions would be implemented, and the status information of each task and vehicle would be updated. Subsequently, a new rollout approximation to the cost to go can be used to determine the decisions for each vehicle at the next decision time.

The coordinate ascent algorithms do not have this property. For each vehicle, one must determine the full path of decisions in order to evaluate the reduced values that these decisions imply for other vehicles. This is the computation cost that goes along with reducing the problem to a vehicle-by-vehicle optimization iteration.

C. Stochastic Dynamic Programming

The previous two algorithms solve a deterministic approximation of the multi-vehicle risky planning problem, which is useful in a model-predictive control approach where real-time recomputation of decisions takes place in response to observed events. An alternative approach is to solve the full stochastic feedback version of the multi-vehicle risky planning problem, using stochastic dynamic programming. In this formulation, events of successful or failed link transitions at each stage are observed, and the solution is expressed in terms of feedback strategies from the observed state into admissible decisions. Note that the state space for the dynamic programming formulation is

$$S = \prod_{k=1}^{K} (N \cup \{0\}) \times 2^N$$

where the first product terms correspond to the vehicle states (state 0 indicates a destroyed vehicle), and the last term 2^N is the set of subsets of tasks that have not been completed yet.

Denote the state $s_t \in S$ to be the state at time t. A decision $u_t \in F(s_t)$ at time t is a set of feasible moves ($F(s_t)$) of each of the surviving vehicles in s_t, according to the arcs in A. Associated with each set of feasible moves is a transition probability kernel, $Q(s_{t+1}|s_t, u_t)$, which is readily computed given the independence assumption of the transitions described in Section II.

Given the above setup, the stochastic dynamic programming algorithm can be used to determine the optimal expected performance for any initial condition, as follows: Initialize the terminal cost as

$$V(s, M+1) = \sum_{n \in N, n \notin s} V_n \tag{8}$$

where one sums the values of the tasks that have been completed. The stochastic dynamic programming iteration becomes

$$V(s,t) = \max_{u \in F(s)} \sum_{s' \in S} Q(s'|s,u) V(s', t+1) \tag{9}$$

$$t = 1, \ldots, M; s \in S \tag{10}$$

The above procedure computes the optimal value completed for the initial condition where no task has been completed and all vehicles are in their starting positions.

Unfortunately, exact solution of the above dynamic programming problem is possible only for small numbers of tasks and vehicles, as the cardinality of the state S grows exponentially with the number of tasks and the number of vehicles. In the experimental results, we show comparisons with the performance obtained by the rollout algorithms and the optimal performance as computed by stochastic dynamic programming in small examples.

IV. Operation under Variable Human Guidance

There are two modes that we consider in this paper. In the *autonomous mode*, the operator specifies values for the different task locations. The vehicles use either the multi-vehicle rollout algorithm or the coordinate ascent algorithm to select paths among the tasks. After each step is complete, the vehicle and task states are recomputed, and the same algorithms are used to determine a new path with a smaller horizon.

In the *partition* mode, tasks are assigned to individual vehicles, and vehicles are constrained to act on the assigned tasks. Values are also assigned to tasks, and it is possible to assign the same task to multiple vehicles. Solution for the routing of each vehicle can be done with the same algorithms as before, constrained to the constraints that vehicles be restricted to their subset of designated nodes.

Since both of the proposed algorithmic techniques are suitable for opertion in either of the modes, it is important to determine which of the two techniques provides superior performance. That is the purpose of the next section, our experiment results.

V. Experiments

In our experiments, we compare the performance of the multi-vehicle rollout algorithm (MVRA), the coordinate ascent rollout algorithm (CARA) and the full stochastic dynamic programming solution whenever possible.

For our first results, We consider a set of test problems consisting of three basic problems and a variable number of tasks assigned. The first test problem consists of 3 vehicles and 36 tasks distributed uniformly in a square area of size 60×60. Quantized task values are integers from 2 to 10. The second test problem had 3 vehicles and 48 tasks in the same 60×60 area, with 6 values at 10 and 42 values at

Problem	Hor.	Risk	MVRA	CARA
1	7	Low	83.43	88.31
1	14	Low	101.47	105.38
1	20	Low	104.41	112.41
1	7	High	68.21	82.49
1	14	High	86.82	105.36
1	20	High	101.57	110.90
2	7	Low	88.16	89.33
2	14	Low	129.19	125.32
2	20	Low	134.99	143.51
2	7	High	77.58	85.30
2	14	High	113.10	123.22
2	20	High	123.73	141.14
3	7	Low	178.26	179.71
3	14	Low	245.82	249.83
3	20	Low	249.07	249.97
3	7	High	168.5	177.99
3	14	High	229.73	249.03
3	20	High	242.58	249.78

TABLE I

PERFORMANCE OF MVRA AND CARA ALGORITHMS ON TEST PROBLEMS WITH VARIABLE HORIZON AND RISK LEVELS

2. The third problem also had 3 vehicles and 41 tasks, in a narrow rectangular region of size 60×2, and values as integers between 2 and 10.

For each problem, arcs were introduced between every pair of tasks. The probabilities p_{ij} of successfully traversing an arc between tasks i and j were selected uniformly in $[L_{ij}, 1]$, where L_{ij} is a lower bound on the probability of successful traversal. Two different conditions were used to determine this bound for each of the test cases, resulting in different levels of risk. The high risk version had

$$L_{ij} = 1 - \frac{d_{ij}}{170}$$

where d_{ij} corresponded to the distance between tasks i and j. The low risk version had

$$L_{ij} = 1 - \frac{d_{ij}}{1000}$$

Additional details on these test problems are provided in [7].

Table I contains the results of our experiments. For each tested condition, the table reports the expected value obtained by the paths selected for each vehicle, averaged over ten instances of the random problems. Across all of these problems, the maximum computation time was under 17 seconds in a 1.2 GHz Intel workstation running LINUX. In general, CARA took about 3 times longer to compute a solution than MVRA, due to the number of iterations required for convergence.

The results in Table I show that the CARA algorithm yields superior performance to that of the MVRA algorithm across all the tested conditions. These results suggest that it is superior to construct vehicle paths one vehicle at a time, instead of using a rollout approach to compute a cost to go across vehicles and times. Based on these results, we propose to use the CARA algorithm as the basis for our future experiments involving human supervisory control.

Hor.	Risk	MVRA	CARA	SDP
5	Low	38.7	39.8	42.3
7	Low	39.5	42.1	44.3
9	Low	40.1	43.2	46.8
5	High	26.8	33.2	36.5
7	High	32.1	35.2	39.1
9	High	34.5	38.4	43.3

TABLE II

PERFORMANCE OF MVRA, CARA AND SDP ALGORITHMS ON SMALL PROBLEMS WITH VARIABLE HORIZON AND RISK LEVELS

The above results do not show whether the performance of the CARA algorithm is close to the optimal performance as computed by the stochastic dynamic programming algorithm. To evaluate this, we consider a smaller problem, which has a total of 12 tasks and 2 vehicles. As before, the 12 tasks were distributed uniformly in a square area of size 30×30, with quantized task values as integers from 2 to 10. The probability of successful traversing an arc from node i to node j was sampled uniformly as in the low-risk and high risk versions discussed above, in the interval $[L_{ij}, 1]$ where d_{ij} is the Euclidean distance between nodes i and j. To keep the computations under control, we use three horizons, corresponding to 5, 7 and 9 units.

Table II contains the results of our experiments with the three algorithms, where the stochastic dynamic programming algorithm is denoted SDP. As in the previous experiments, the results reported are the average of 10 randomly selected problems generated according to the problem description above. For the CARA and MVRA algoritms, the value reported is the value computed by the expected cost according to (3). For the SDP algorithm, the cost reported is the optimal value computed by the SDP recursion in (9). Note that this represents comparing the value achieved by a closed-loop feedback algorithm versus open-loop sequences of nodes to visit. Thus, there should be a significant performance difference between the optimal SDP performance and the open-loop performance achieved by MVRA and CARA.

The results again confirm that the CARA approach yields improved solutions over the MVRA approach, and is closer to the optimal SDP algorithm. As the horizon is increased, the algorithms find less risky paths and exploit the additional stages to collect additional value. For short time horizons, the algorithms attempt the higher-valued tasks, even if the risks are high, and reduce the total value collected.

VI. CONCLUSION

In this paper, we considered the problem of task assignment and scheduling for a team of unmanned vehicles in a risky environment where vehicles incur the risk of destruction. The possibility of vehicle loss requires that selected

task assignments hedge against risk by assigning multiple vehicles to valuable objects. We presented an expected value formulation for computing the expected task value completed given deterministic paths for each vehicle. We developed two extensions of our previous results on rollout algorithms for stochastic scheduling to multivehicle scenarios, and described their implementation. We also presented a stochastic control formulation of the problem, and discussed the dynamic programming solution.

We evaluted the two rollout algorithms and the stochastic dynamic programming algorithm over a set of randomly generated problems. Our results indicate that the slower of the two approaches, the Coordinate Ascent Rollout Algorithm, achieved superior performance in all of the test conditions.

Our goal in this study is to develop feedback scheduling algorithms that hedge against and adapt to vehicle loss. The approximate algorithms presented in this paper solve an open-loop version of the problem. To generate adaptive versions of these algorithms, we will use the rollout algorithms in a model-predictive control framework, where new observations will trigger re-optimization of selected trajectories based on the most recent information available.

The results in this paper illustrate the expected value achieved by the different algorithms when the trajectories selected by the CARA and MVRA algorithms are used without adaptation. Future work will include evaluations of the closed-loop performance of these algorithms when adaptation is introduced using model-predictive control. We will also be implementing this paradigm for experiments in Boston University's Mechatronics Laboratory in order to introduce alternative ways of exercising human guidance and control of the unmanned vehicles.

VII. ACKNOWLEDGMENTS

The authors would like to thank the Conference Editorial Board reviewers for their thoughtful comments that improved the final presentation in the paper.

REFERENCES

[1] W. B. Powell, "A Stochastic Model of the Vehicle Allocation Problem," *Transportation Science*, vol. 20, pp.117-129, 1986.

[2] W. B. Powell and T. A. Carvalho, "Dynamic Control of Logistics Queueing Networks for Large-scale Feel Management," *Transportation Science*, vol. 32, pp.161-175, 1998.

[3] K. P. Papadaki and W. B. Powell, "An Adaptive Dynamic Programming Algorithm for a Stochastic Multiproduct Batch Dispatch Problem," *Naval Research Logistics Quarterly*, vol. 50, pp.742-769, 2003.

[4] D. P. Bertsekas and D. A. Castañón, "Rollout Algorithms for Stochastic Scheduling," *Heuristics,* V. 5, pp.89–108, April, 1999.

[5] G. G. denBroder, R. E. Ellison and L. Emerling, "On Optimum Target Assignment," *Operations Research,* V. 7, 1959.

[6] D. A. Castañón and J. Wohletz "Model Predictive Control for Dynamic Unreliable Resource Allocation," *Proc. 41st IEEE Conference on Decision and Control,* Las Vegas, NV, Dec. 2002.

[7] D. K. Ahner, *Planning and Control of Unmanned Air Vehicles in a Dynamic Stochastic System,* Ph. D. Thesis, Systems Engineering, Boston University, May 2005.

[8] S. P. Lloyd and H. S. Witsenhausen, "Weapons Allocation is NP-Complete," *Proc. 1986 Summer Conf. on Simulation,* Reno, NE 1986.

[9] D. Q. Mayne, J. B. Rawlings, C. V. Rao and P. O. M. Skocaert, "Constrained Model Predictive Control: Stability and Optimality," *Automatica*, V. 36, 2000.