Tactical Intelligence Tools for Distributed Agile Control of Air Operations

Shashi Phoha
Applied Research Laboratory
The Pennsylvania State University
University Park, PA 16802
sxp26@psu.edu

Natarajan Gautam
Industrial Engineering
The Pennsylvania State University
University Park, PA 16802
ngautam@psu.edu

Alan Horn
Applied Research Laboratory
Pennsylvania State University
University Park, PA 16802
ajh14@psu.edu

Abstract

This paper presents an innovative architecture for engineering an agile distributed system of interacting agents, which is modeled as a set of interacting automata[11]. By slight modification of Goranson's definition of agility [5], we define agile control as the collective ability of agents to continually adapt to expected and unexpected changes. The intelligent control of the enterprise, then, consists of (i) continual observation of events and identification of relevant changes on the battlefield (ii) controllers implementing a dynamic C^2 strategy as a control specification for each agent in the enterprise, and (iii) tactical intelligence tools which process necessary transient and situational information for the agents to execute the control decisions in the battlefield. A testbed implementing these components and their interactions has been established as shown in Figure 1. Methods for engineering agile control specifications for a hierarchy of agents in the air operations enterprise are discussed in a companion paper in these proceedings [14]. This paper presents the overall architecture and its testbed implementation for agile control of air operations. In addition, a variety of tactical intelligence tools are discussed which provide situational and transient information for executing control decisions. Algorithms for dynamic assignment of targets to aircraft is an example of a tactical intelligence tool. These algorithms require current and situational knowledge of resources and positions of identified targets to continually reconfigure assignments. Both centralized and onboard tactical intelligence algorithms are discussed and compared for relative performance. The testbed is being used for more detailed experimental verification of C^2 strategies for air operations as described in another companion paper in these proceedings [4].

1. Intelligent Control of Dynamic Operations

The air operations enterprise can be represented as a dynamic hierarchy of intelligent agents who change their internal states in response to interactions with other agents or the environment. The dynamic time-evolution of complex interactions of agents in the enterprise is inherently different from the continuous variable dynamics of physical processes. Unlike the evolution of physical processes which are constrained by physical laws, agent interactions are triggered by discrete events and are characterized by a large number of interacting processes under predictable and unpredictable disturbances. Hence there are no inherent physical laws to constrain system configurations other than natural limitations of human/machine comprehension, resources and ergonomics. Therefore, an accurate plant model, based on physical laws, cannot easily be formulated. Concurrent dynamic processes, embedded at each node of the system, interact in highly non-linear, time-varying stochastic and possibly inconsistent ways. Hence model based conventional control methods are inadequate. Alternate methods of designing controllers whose structure and outputs are determined by empirical evidence through observed input/output behavior rather than by reference to a plant model are necessary.

Several techniques for such non-linear controller design have recently been proposed in recent literature on Intelligent Control [6, 7, 10, 2, 9, 12]. Meystel [8] has proposed a nested hierarchical control architecture for the design of Intelligent Controllers. Albus [2] has also developed the Real Time Control Architecture in which sensor and processing, value judgment, world modeling and behavior generation subsystems interact to adaptively generate appropriate response behaviors to sensor observations and knowledge of mission goals. Brooks' subsumption architecture [3] for intelligent control is based on achieving increasing pre-specified levels of competence in an intelligent system by examining outputs of lower levels.

An innovative behavior based architecture for intelligent agile control of agents in the air operations enterprise is presented here. The mathematical representation of agent interactions as a cellular space of interacting automata was formulated in [11]. The uniqueness of this approach is that it allows a heterogeneous group of agents to self organize and coordinate composite behaviors to support the execution of desired global behaviors of the system. Desired system behaviors are enabled through the hierarchical control architecture of the system and represented as control specifications. Methods for the control specifications for an agent hierarchy and for checking the controllability and hierarchical consistency of the resulting system are discussed in [14].

In section 2 we present an overall architecture for implementing agile enterprise control. Section 3 discusses centralized versus distributed tactical intelligence. In Section 4, we present four different algorithms for dynamic target scheduling and routing of aircraft in a limited SEAD Scenario [13] for corridor clearance. Section 5 discusses resource bounded optimization and performance comparisons for the four algorithms. More detailed experimental verification of distributed C^2 strategies on this testbed is discussed in [4].

2. Agile Control Architecture

The essential components of Agile Control are shown in Figure 1. These are:

(i) Identification and communication of relevant change in the enterprise to appropriate agents.

(ii) Hierarchical controller for specifying and implementing a control strategy, and

(iii) Tactical intelligence tools for providing intelligent decision support for executing the control strategy.

The description and interaction of these components will be described in detail in this paper.

An enterprise simulator, representing air operations in the battlefield, generates events which are observed and responded to by intelligent agents. The design of the hierarchical controller assumes the strategic knowledge an experienced commander may have in specifying a control strategy for his agents for a particular mission. For example, in the corridor clearing scenario described later in this paper, the control strategy will be based on mission planning information which may not be specific in terms of the exact location of targets. In general, sufficient planning information is assumed so that our overall control strategy can be implemented.

Control specifications define the execution level behavior-modification rules for each agent in response to observed changes in the enterprise. These specifications are *ill-posed*, however, for the execution level dynamics

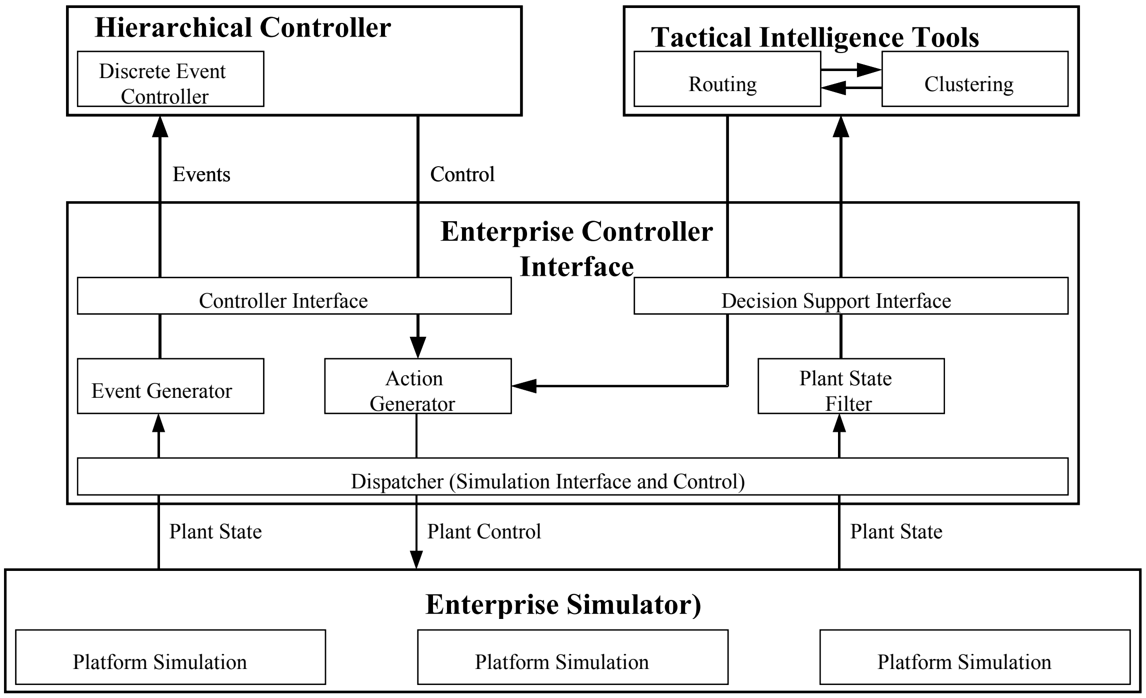

Fig 1. Components of the Agile Control Architecture

in the enterprise is not yet defined. Tactical intelligence tools are developed for providing situational and transient information derived from event observations (including status/position report) to support the execution of the control strategy. For example, upon identification of an unanticipated threat, the on-board controller may enable the 'escape' behavior for a fighter aircraft. The tactical intelligence tools on-board the aircraft must then provide move-to-coordinates (x, y, z) for safe escape. In general, the tactical intelligence tools identify observable change in the enterprise and provide situational decision support to the controller to enable an appropriate response. Together, the hierarchical controller and tactical intelligence tools define the response mechanism for an intelligent agent.

The agent interface to the enterprise allows the observation of change events in the enterprise by the agent and enables the control actions to be implemented in the enterprise.

This modular construction of the testbed allows multiple control algorithms to be tested for a given enterprise. It also allows variations in the plant simulator to evaluate control performance of a particular control strategy for various possible evolutions of the enterprise.

The following is a detailed description of the components in Figure 1:

Enterprise Simulator (ES) - The ES generates raw data for all forces, interactions among platforms, and environmental conditions. It also responds to inputs from the Dispatcher.

Dispatcher (D) – D receives data from the ES and sends it to the Event Generator (EG) and Plant State Filter (PSF). It acts as a bridge between the controllers and the enterprise simulator (ES).

Event Generator (EG) - The EG abstracts discrete events from the continuous data supplied by D. It also sends an event list to the hierarchical controller.

Action Generator (AG) - The AG takes an event vector from the controller and determines what actions should be taken. The AG, sometimes using tactical intelligent tools, determines what changes in the plant (continuous world) are required to enact each command (discrete events).

Plant State Filter (PSF) – The PSF reads pertinent data from the plant for use by the Tactical Intelligence Tools.

Controller Interface – This is a bridge between the EG software and the controller software.

Decision Support Interface – This is a bridge between the PSF software and the Tactical Intelligence Tools.

Hierarchical Controllers (HC) – The HC interpret the list of events from the EG and from this list they send an action vector to the action generator. An action vector is a control request to enable or disable specific controllable events.

The HC also abstract from event lists, sending these "higher-level" events to higher-level controllers. These higher-level controllers will also send commands back to the lower-level (original) controller through an action vector.

Tactical Intelligence (TI)– These are a set of tools used by the Action Generator to make optimized decisions. These contain algorithms for aircraft routing and creating target lists for aircraft, called clusters.

3. Centralized Vs. Distributed Tactical Intelligence

Some of the algorithms for tactical intelligence need to be performed in a centralized fashion, such as the planning algorithms before the start of a mission. However once the mission starts there is a choice of running the tactical intelligence algorithms either centrally or in a distributed fashion. The advantages of a centralized algorithm are potentially superior solutions due to global optimization. The disadvantages of the centralized algorithms are: the local information regarding enemy targets cannot be used, the communication overheads, security problems during communications, failure of communications, complexity of problems, large solution time for problems, potential inability to recover from multiple back-to-back failures, etc. Since the merits of the distributed algorithms are compelling for air combat command and control which are initially developed as centralized, are distributed over the platforms for autonomous, unsynchronized execution.

4. Algorithms for Tactical Intelligence

The tactical intelligence module uses several algorithms to assist the controller in making intelligent decisions. One of the algorithms to facilitate dynamic target scheduling and routing is explained here. It is a distributed algorithm that each friendly aircraft uses to autonomously decide its target schedule and routes. Some of the instances when this algorithm is used during the mission are: when an unknown enemy target is spotted, when the enemy attacks, during a mechanical failure, when fuel runs out, when weapons run out, when new aircrafts are added, etc.

4.1 Dynamic target scheduling and routing methods

Given the location of the friendly aircraft (e.g. Wild Weasel) and the expected locations of the enemy targets (with types such as fixed SAMS, mobile SAMS and radars) the objective of the dynamic target scheduling and routing algorithm is to compute a least-cost path to destroy all known enemy targets. The cost of a path is a function of the expected time to traverse a path and the expected risk incurred by the friendly aircraft. The following are the notations used in the algorithms:

N	:	Set of enemy targets
(x_i, y_i)	:	Coordinates of enemy target i (i 0 N)
(x_0, y_0)	:	Location of friendly aircraft at the time of determining route
C_{ij}	:	Cost for directly traveling from enemy target i to enemy target j (i, j 0 N)
R	:	An ordered set corresponding to the route of friendly aircraft
Z	:	Total cost for traversing route R
D	:	Dummy subset of enemy targets

Note that C_{ij} is computed using $(x_i,y_i),(x_j,y_j)$, speed of the aircraft, and, a risk factor (r_{ij}) such that among all paths between i and j, C_{ij} is cost associated with the path corresponding to the minimum of the product of the risk factor and travel time. We assume in this paper that C_{ij} has been pre-computed and is available to use in the algorithms. In a follow up paper we will describe algorithms to compute costs C_{ij} as well as the paths between targets i and j. Some algorithms for dynamic target scheduling and routing are now explained.

Greedy Algorithm: Given a set of enemy targets to destroy and the current location of the Wild Weasel, a greedy algorithm called the nearest neighborhood search algorithm is developed to obtain a sequence and route to destroy the targets. The greedy algorithm begins with the current location of the Wild Weasel and destroys the enemy target that can be reached by traversing the least cost path. Among the undestroyed targets, the algorithm next selects the enemy target that can be reached by traversing the least cost path. This process continues until all known enemy targets are destroyed. Then the Wild Weasel patrols its assigned region. The main advantage of using the greedy algorithm is to obtain high responsiveness for the tactical intelligence module. Hence the sequence and route to destroy the targets can be obtained extremely fast. However the price to pay for this high-speed response is the quality of the response. In most cases this response from the tactical intelligence is far from optimal, hence the overall objective of minimizing the cost of destroying the targets will not be accomplished.

<u>Algorithm Greedy</u>

```
1.  Set $i = 0$, $Z = 0$, $R := \varnothing$ and $D := N$
2.  While $D$ □ $\varnothing$, do
        a. $Z = Z + \min_{j \in D} C_{ij}$ and $k = \arg\min_{j \in D} C_{ij}$
        b. $R = R$ " $k$ and $D = D$ ! $k$
        c. $i = k$
```

Resource Bounded Optimization (RBO) Algorithm: Given a set of enemy targets to destroy and the current location of the Wild Weasel, an RBO algorithm based on the 2-opt search for the well-known graph theoretic problem, the traveling salesperson problem, is developed. Note that the problem of obtaining a sequence and route to destroy the targets can be stated as a Hamiltonian path problem as explained in the algorithm that follows. In the RBO algorithm, the solution from the greedy algorithm is taken and improvised. At each iteration, a random-pair-wise interchange to the current sequence and route to destroy the targets is performed. After several iterations, the algorithm will converge to the optimal solution. The number of iterations the tactical intelligence will perform will depend on the time available to respond. The solution quality will improve with the number of iterations. This algorithm can be stopped at any iteration and a solution can be obtained. The numerical examples indicate a vast improvement in the solution quality (as compared to the greedy algorithm and the optimal algorithm) in very few iterations. However the disadvantage is that it may take a long time to obtain the optimal solution especially in ill-posed problems. We require a few definitions for the algorithm. Consider an element i in the ordered set R. Define $P(i)$ and $S(i)$ as the elements preceding and succeeding i respectively in the ordered set R. In other words $P(i)$ and $S(i)$ are respectively the enemy targets destroyed before and after destroying target i. Note that if i is the first or last target respectively, then $P(i)$ or $S(i)$ are null sets. Also define an operation $U(A)$ on a set A that results in a 2-tuple denoting 2 elements randomly selected from set A. Denote the binary variable $Y(i,j)$ such that it is 1 if arc (i,j) is in route R and, 0 otherwise.

<u>Algorithm RBO(I)</u>

```
1. Set $I$ as the number of desired iterations
2. Obtain $R$ using Algorithm Greedy
3. Update $R$ to include the initial location of friendly
   aircraft, i.e., $R = \{0\}$ χ $R$
4. For $x = 1$ to $I$, do
     a. $(i, j) = U(R)$ [Note: $i$ and $j$ are 2 nodes randomly
        selected from the route]
     b. if $i$ □ $j$, $P(i)$ □ ι, $P(j)$ □ ι, $S(i)$ □ ι, $S(j)$ □ ι, $P(i)$ □ $j$,
        and $P(j)$ □ $i$
     i.   $n1 = i, m1 = S(i)$
     ii.  $n2 = j, m2 = S(j)$
     iii. if $C_{n1,m1} + C_{n2,m2} > C_{n1,n2} + C_{m1,m2}$,
          then modify $R$ such that
             $Y(n1,m1)=0, Y(n2,m2)=0, Y(n1,n2)=1$ and
             $Y(m1,m2)=1$
          elseif $i$ □ $j$, and $S(i)$ □ ι,
             $n1 = i, m1 = 0$ and $n2 = j, m2 = S(j)$
          if $C_{n2,m2} > C_{n1,2}$,
          then modify $R$ such that
```

```
            $Y(n2,m2)=0$ and $Y(n1,n2)=1$
        elseif $i$ □ $j$, and $S(j)$ □ ι,
            $n1 = i, m1 = S(i)$ and $n2 = j, m2 = 0$
        if $C_{n1,m1} > C_{n1,n2}$,
        then modify $R$ such that
            $Y(n1,m1) = 0$ and $Y(n1,n2)=1$
    else
go to a.
```

Note that other "elseif" conditions can be incorporated into the algorithm. For presentation reasons those conditions have been omitted.

Hamiltonian Path: The path traversed by the Wild Weasel from the current location to the last destroyed target is referred to as the Hamiltonian path in graph theory literature (see Ahuja [1]). The algorithm uses a network representation such that the known enemy targets are the nodes of the network. There is an arc from every node to every other node denoting the ability to go from any target to any other target. The cost on the arc represents the cost of traversing from one target to another and is computed using the expected travel time and expected risk. The algorithm begins by solving the minimal spanning tree (another well-known graph theoretic problem) of the network. If the spanning tree is not a Hamiltonian path, then it has arcs that violate the Hamiltonian path requirements. The solution of the minimal spanning tree acts as the lower bound. Then at every iteration, the lower bound is improved using a branch-and-bound technique where one of the violating arcs of the spanning tree is set to a high cost. The algorithm stops when a Hamiltonian path is obtained, i.e., there are no more branches to consider. This algorithm can guarantee optimal solution after a sufficiently large number of iterations. The major drawback is that if the algorithm is stopped during any iteration, no solution will exist that can be responded by the tactical intelligence.

We introduce a few notations before illustrating the algorithm. Let $C = [C_{ij}]$ be a cost matrix denoting the arc cost between every pair of nodes. Define $T(C)$ as an operation on C that results in a minimum spanning tree of the network. In particular, let $X = T(C)$ such that $X = [X_{ij}]$ and X_{ij} is a binary variable such that it is 1 if arc (i,j) is in the spanning tree and, 0 otherwise. For minimum spanning tree algorithms, see [1] to determine if a spanning tree is a Hamiltonian path is to check if the degree of all nodes in the spanning tree is not greater than 2, except node 0 whose degree should not be greater than 1. In essence, the following algorithm iterates through several spanning trees until a Hamiltonian path is obtained.

Algorithm Hamiltonian

```
1. Set $W = 0$ and $X = T(C)$
2. If $X$ is a Hamiltonian path, $W = 1$
3. While $W = 0$, do
   a. $Z = 0.5\,XC$, the total cost of the spanning tree
   b. For every node with degree greater than allowed:
      i. Obtain $X = T(C)$ assuming one of arc cost of
         the node is infinite
      ii. If $X = T(C)$ is not a Hamiltonian path, go to
          b. Else $Z_0 = 0.5XC$
   c. Choose Hamiltonian path with smallest $Z_0$
4. $Y = X$ and obtain the route $R$ via $Y$.
```

Note that it is possible to improve the above branching procedure by doing a branch-and-bound procedure. This would require fathoming all minimum spanning trees whose costs are greater that the current Hamiltonian path cost during an iteration. If the algorithm is stopped in the middle, to avoid the situation of not having a route we do the following: if a (current) Hamiltonian path is available it can be used, otherwise, the solution from the greedy algorithm can be used.

Optimal Algorithm: This algorithm uses complete enumeration of the entire solution space to obtain the optimal solution. Therefore every possible sequence and routes to destroy the targets are considered and the best is chosen. This is a very time-consuming algorithm and takes $n!$ computations if there are n targets to be destroyed. Since this algorithm guarantees an optimal solution, it is used for benchmarking other algorithms.

Define $\overline{R}$ as a set of all possible routes and r as a candidate route. Also let $D(r)$ be the cost of the candidate route r.

Algorithm Optimal

```
1. Set Z = 4
2. For all $r \in \overline{R}$, if $D(r) < Z$, then $R = r$ and $Z = D(r)$
```

5. Resource Bounded Optimization Performance Comparisons

The algorithms developed in the previous section are tested here for different numerical values. In particular, the performance of the new RBO algorithm developed is tested against the other algorithms for an air operation scenario described below.

5.1 Generic Air Ops Scenario

The scenario considered here is a limited SEAD scenario. A bombing mission is to be attempted against an enemy airbase. For the bombers to be able to perform the mission, enemy air defenses must be disabled in two corridors leading to the base. In the scenario, the corridors are given. They are four miles wide at their narrowest, to insure the safety of aircraft flying down the middle of the corridor. The enemy has three types of entities: (1) fixed SAM sites, (2) mobile SAM launchers, and (3) fixed radar sites. The mobile SAM launchers perform a random walk on the terrain. They stop wandering and prepare to attack at random points in the walk. Any target that has been hit is disabled for a random period of time.

Friendly forces are limited to Wild Weasels, which search for and destroy SAMs. Each Wild Weasel has its own discrete event controller. The local controller has access only to local information. Another discrete event controller coordinates activities among the Wild Weasels. The coordinator interacts with the system by receiving ISR inputs and sending radio messages to the aircraft. Each aircraft starts with an initial mission to be completed. The aircraft's controller determines the decisions that are taken as events occur. Missions will be to patrol parts of the corridor and destroy enemy entities. Aircraft communicate with the supervisory controller as needed for coordination. It will adjust the regions covered and targets aircraft in response to changing conditions.

The tactical intelligence module explained in Section 2 is responsible for (1) allocating platforms to targets and regions (2) allocating routes to platforms, and, (3) allocating patrolling pattern after destroying known SAMs in a region. In this paper we have explained in detail the algorithms only for the second task, i.e. allocating routes to platforms. The other algorithms are explained in [4]. It is important to note that these algorithms are executed both during the initial planning phase as well as en-route during the attack phase. Therefore it is critical to obtain an algorithm that produces reasonably good results in a short period of time. At this time, we only compare the performance of the algorithms running independently. However, in a future paper, we will provide the results based on battlefield simulations.

The coordinator assigns regions for aircraft to cover. This is a centralized algorithm that uses a clustering algorithm (K-Means algorithm) and regions are created by a Vornoi process which partitions a corridor into disjoint regions. The individual aircraft choose their own strategies for destroying known targets based on a decentralized algorithm. Once all known targets are destroyed, another decentralized algorithm (such as a lawnmower-type algorithm) to patrol for new threats is used. Regions must be reassigned and strategy for destroying targets must be reformulated as aircraft are destroyed, unknown enemy targets are spotted, aircraft run out of fuel or weapons, or new aircrafts are added.

5.2 Performance Metrics

The dynamic target scheduling and routing methods are considered here. It is assumed that regions have been described and targets have been assigned to each aircraft. In order to compare the four algorithms in Section 4.1, we use two performance metrics: solution quality and number of floating-point operations. The *solution quality* is benchmarked against the best possible solution. Therefore the ratio between the optimal solution (produced using the optimal algorithm) and the solution produced by an algorithm is the measure considered for solution quality. The *number of floating-point operations* is a measure of the number of operations that will be required on a computer to obtain the given solution. Then based on the type of computer installed on the aircraft, this metric can be used to determine the time to respond to the controller with a route.

5.3 Performance Evaluation

To evaluate the performance of the different algorithms, using 30 sets of enemy target locations to destroy and current location of the Wild Weasel, for each set, the following algorithms were considered and average performance metrics were obtained: greedy algorithm, RBO(I) algorithm with I=5, 10, 25, 50 and 100 iterations, Hamiltonian path algorithm (which is stopped after a sufficient number of iterations), and, optimal algorithm. The performance metrics are tabulated in Table 1, below. This table is based on 8 enemy targets assigned to an aircraft. Any value larger than 8 targets would require very large computational time for the optimal solution. However, for the other algorithms we could use many more targets. Also, the table is obtained by running the algorithms off-line. From the table note that the RBO algorithm with just 10 or 25 iterations results in a good solution. Therefore, if the RBO algorithm needs to be aborted after, say 5000 floating point operations, the solution obtained is very good. On the other hand, the Hamiltonian algorithm after 5000 floating point operations would not have produced any solution. Also, the greedy algorithm would have used an inferior solution. The RBO algorithm produces significantly better results than the greedy algorithm in a very few extra iterations. Therefore for the SEAD scenario it would be most appropriate to use the RBO algorithm and depending on the time available to solve the RBO, the algorithm can be stopped at a suitable time.

	Greedy	RBO (5 itns)	RBO (10 itns)	RBO (25 itns)	RBO (50 itns)	RBO (100 itns)	Hamilt.	Optimal
Solution quality	0. 9575	0.9683	0. 9727	0.9824	0. 9845	0. 9868	0. 9907	1.0000
Floating pt. Ops.	992	2589	4095	8601	15898	30395	49048	351769

Table 1. Performance metrics

Acknowledgements and disclaimers

This effort is sponsored by the Defense Advanced Research Projects Agency (DARPA) and Air Force Research Laboratory, Air Force Materiel Command, USAF, under agreement number F30602-99-1-0547 (JFACC). The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency (DARPA), the Air Force Research Laboratory, or the U.S. Government.

References

[1] R.K. Ahuja, T.L. Magnanti, and, J.B. Orlin, *Network Flows: Theory, Algorithms and Applications*, Prentice-Hall Inc., 1993.

[2] J.S. Albus, "A Reference Model Architecture for Intelligent Systems Design," in *An Introduction to Intelligent and Autonomous Control*, pp 27-56. Kluwer Academic Publishers, 1993.

[3] R.A. Brooks, "A Robust Layered Control System for a Mobile Robot," *IEEE Transactions on Robotics and Automation*, 2(3): pp 14-23, 1986.

[4] R. R. Brooks, C. Griffin, P. Dicke, M. Byrne, M. Edwards, S. Phoha, D. Friedlander, B. Button and E. Grele, "Experimental Verification of Distributed C^2 Strategies," proceedings of 2nd DARPA JFACC *Symposium on Advances in Enterprise Control*, Minneapolis, MN, July 10-11, 2000.

[5] H. T. Goranson, *The Agile Virtual Enterprise Cases, Metrics, Tools*, Quorum Books, 1999.

[6] C. J. Harris, ed., *Advances in Intelligent Control*, Taylor & Francis, Bristol, PA, 1994.

[7] A. H. Levis, "Modeling and Design of Distributed Intelligence Systems, " in *An Introduction to Intelligent and Autonomous Control*, pp 109-128, Kluwer Academic Publishers, Boston, M, 1993.

[8] A. Meystel, "Autonomous Mobile Robots: Vehicles with Cognitive Control, "*Proceedings of the World Scientific*, Singapore, 1991.

[9] S. Phoha, S. Sircar, A. Ray, and I. Mayk," Discrete Event Control of Warfare Dynamics," The Technical Proceedings of the *1992 Symposium on Command and Control Research and the 9th Annual Decision Aids Conference*, Monterey, CA, June 8-12, 1992.

[10] S. Phoha, E. Peluso, P.A. Stadter, J. Stover, and R. Gibson, "A Mobile Distributed Network of Autonomous Undersea Vehicles," Proceedings of the *24th Annual Symposium and Exhibition of the Association for Unmanned Vehicle Systems International*, Baltimore, MD, June 3-6, 1997.

[11] S. Phoha and R. Brooks, "A Constructivist Theory of Distributed Intelligent Control of Complex Dynamic Systems," DARPA JFACC *Symposium on Advances in Enterprise Control*, San Diego, CA, November 15-16, 1999.

[12] P. J. Ramadge, W. M. Wonham, "Supervisory Control of a Class of Discrete Event Processes," *SIAM J. Control and Optimization*, Vol. 25, No. 1, January 1987.

[13] SEAD Scenario. http://www.cgi.com/web2/govt/seadystorm, April 7, 2000.

[14] W. Xi, A. Ray, S. Phoha and W. Zhang "Hierarchical Consistency of Supervisory Command and Control of Aircraft Operations," proceedings of the 2nd DARPA JFACC *Symposium on Advances in Enterprise Control*, Minneapolis, MN, July 10-11, 2000.