Proceedings of the 42nd IEEE
Conference on Decision and Control
Maui, Hawaii USA, December 2003

A Framework for Conflict Resolution in Air Traffic Management[1]

Stefan Resmerita[2], Michael Heymann[3], George Meyer[4]

Abstract

We propose a distributed multi-agent framework for conflict free navigation in a discrete environment. The two-phased approach, which consists of a 'conflict-resolution' phase followed by an 'agent-accommodation' phase, guarantees maximal capacity utilization under guaranteed safety requirements. The methodology, which is motivated by the Free-Flight paradigm in Air Traffic Management (ATM), is applicable to autonomous UAVs, various ground transport settings and, with suitable modifications, also to distributed ATM.

Keywords: Air traffic management, multi-agent systems, conflict resolution.

1 Introduction

The current air traffic management (ATM) system is based on centralized control, with limited flexibility for individual aircraft to freely choose their optimal paths. The system relies on human operators in local air traffic control (ATC) centers (that are further partitioned into sectors), to track all aircraft along their nominal pathways and control their trajectories so as to insure adequate aircraft separation. Consequently, in order to keep the system manageable, there are strict limits on the number of aircraft that are allowed into a sector, creating capacity and flexibility constraints.

An intense research effort is currently under way to overcome some of the above mentioned limitations and better utilize existing technological capabilities such as Global Positioning System, data-link communication, Traffic Alert and Collision Avoidance Systems and powerful on-board communication. This infrastructure enables, among other things, aircraft to aircraft communication, predictability of trajectories of other aircraft, and on-flight access to central databases with flight plans.

One approach that has been proposed to achieve these goals and to increase individual aircraft path selection flexibility is the Free-Flight paradigm [3], in which pilots would be allowed to choose optimal trajectories, and possibly also be responsible for maintaining flight safety. In addition to the potential savings that could be achieved by airlines (in terms of fuel consumption and travel time), decentralization of control could reduce the workload at ATC's (which would then play a more supervisory role). As a consequence, it might be possible to increase the number of aircraft supervised by ATC, thereby achieving a better utilization of the airspace.

Inspired by the challenge posed by the Free-Flight idea, we propose in this paper a novel framework for distributed Traffic Management, which, in addition to safety and individual optimality, addresses explicitly the issue of efficient utilization of the common resource (airspace, tracks, roads, etc. as the case may be) . In our proposed framework, we deal with Autonomous Agents, and the above goals are accomplished through inter-agent interaction, rather than through centralized control.

A recent review of the literature on conflict resolution in ATM can be found in [2]. Most of the research regarding Free Flight deals with collision avoidance strategies [7] [4], i.e., computing deviations from nominal paths of the involved aircraft. A multi-agent approach based on negotiation is presented in [8]. A token-based framework is described in [1].

In the present paper we employ a discrete model of the system. Its relation to ATM is obtained by viewing the aircraft as agents and the 'airspace' as partitioned into cells, so that the required separation constraint is satisfied by guaranteeing that at most one agent occupies a cell (resource) at any time. Thus, the resource system is modelled by an undirected graph, where a vertex corresponds to a cell and an edge represents an adjacency relation between two cells. An agent must travel from an initial vertex to a destination vertex (both of which are specific to each agent). An agent trajectory is a timed directed path in the resource graph, start-

[1]The work of the second author was supported in part by the Technion Fund for Promotion of Research and was completed while he was visiting NASA Ames Research Center, Moffett Field, CA 94035, under a grant with San Jose State University.

[2]Department of Computer Science, Technion, Israel Institute of Technology, Haifa 32000, Israel. E-mail: stefan@cs.technion.ac.il

[3]Department of Computer Science, Technion, Israel Institute of Technology, Haifa 32000, Israel. E-mail: heymann@cs.technion.ac.il

[4]NASA Ames Research Center, Moffett Field, CA 94035. E-mail: George.Meyer-1@nasa.gov

ing at the initial vertex and ending at the destination vertex. Each transition in the path specifies the time of residence in the preceding vertex (the time of cell traversal). The number of agents in the system is not specified and may change with time. An agent can enter the system at any arbitrary (integer) instant of time and exits the system upon task completion.
An agent is required to announce the set of all its optimal paths (which are of equal quality to the agent). We call this the agent's *model*. Optimal paths are determined by a specified set of performance criteria, a computation not further discussed in the present paper. We assume that an agent can do that. To satisfy safety, an agent's movement is restricted to a *legal* subset of its model, called *legal plan*, such that legal paths of different agents are conflict-free. An agent follows an arbitrary path in its legal plan. An *incoming* agent enters the resource system upon determining a nonempty legal plan, at which time it becomes *active*. An active agent cannot be stopped or suspended while executing its task. Thus, a *liveness* specification, that insures that an active agent always has a nonempty legal plan, must be satisfied.

At the heart of the proposed framework is the mechanism by which agents select their legal paths, so as to insure safety, liveness, and efficient resource utilization. The proposed methodology consists of two algorithmic phases, preceded by an initialization phase. First, an incoming agent determines the subset of its optimal paths that are conflict-free with the legal paths of the active agents (already in the system). Then, in the *conflict resolution* phase, the agent resolves potential conflicts with paths of all other incoming agents. If no legal path is obtained, then the *accommodation* phase is executed, where the agent can request and obtain resources owned by active agents (who will try to accommodate an incoming agent).

In the sequel we describe the conflict resolution phase (fully described in [5]), and then present in some detail the accommodation procedure.

2 Agent Models and Conflict Detection

Consider the portion of airspace depicted in Figure 1. There are two airplanes, hereby denoted by R and S. R needs to fly from cell g to h and it can use any of the three routes depicted with dashed lines. S has to fly from d to a and it can follow two routes. Each agent has specific times of cell traversal. The models are represented in Figure 2 (the thick paths will be explained shortly). For example, it takes three units of time for agent R to arrive at cell c, and two units to enter f. Notice that several conflicts may occur, involving: c at time 3, f at time 3, b at time 6 and e at time 9. We use the symbol ϵ to signify flight termination (ϵ is not a node of the resource graph).

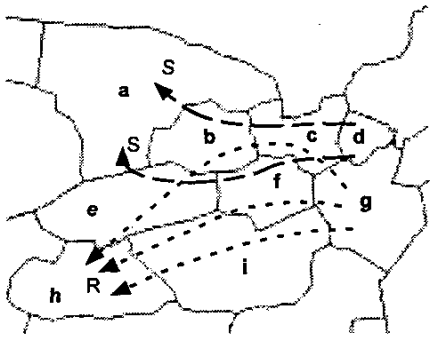

Figure 1: Cells of airspace

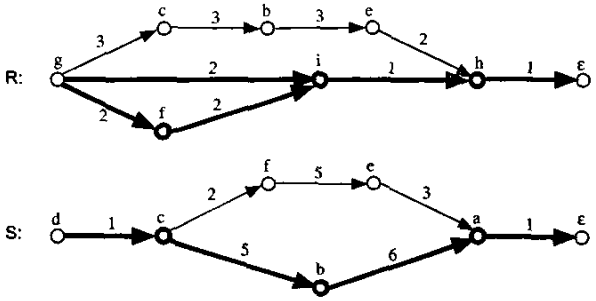

Figure 2: Agent models

Let V be the set of vertices of the resource graph. An *agent automaton* is a triple $A = (Q, q_0, E)$, as follows. Q is a set of vertices: $Q = Q' \cup \{\epsilon\}$, where $Q' \subseteq V$ and where ϵ is the task termination node (that occupies no resource of V). $q_0 \in Q'$ is the initial node which, if the agent is active, represents the currently occupied resource, and if the agent is incoming, is the first resource occupied upon becoming active. E is a set of transitions (edges) of the form $e : q \xrightarrow{\tau(e)} q'$, where $q \in Q'$, $q' \in Q$, $q' \neq q_0$, and $\tau(e)$ is a positive integer representing the time (relative to the moment of entry at q) at which e can be executed. Thus, the agent may move from q to q' (by executing transition e) precisely $\tau(e)$ units of time after entering q. In addition, if $q' \neq \epsilon$, then (q, q') is an (undirected) edge in the resource graph. The following structural conditions must be satisfied: the directed transition graph of A is acyclic, and every vertex $q \in Q'$ is contained in an agent path. Notice that the undirected transition graph of A is a subgraph of the resource graph.

By an agent path, or simply a *path*, we always understand a complete path in the agent automaton, starting at q_0 and ending at ϵ. A *disputed resource*, or *conflict*, between paths p_i of agent i and p_j of agent j is a pair $(\tau, q) \in \mathbb{N} \times V$ such that both agents i and j would occupy q at time τ if they followed p_i and p_j respectively and at most one of i, j would occupy q at $\tau - 1$. Thus,

τ is an instant of conflict occurrence between i and j at q. For example, in Figure 2, $(3, c)$ is a disputed resource between the lower path of S and the uppermost path of R.

Conflicts can be detected by computing the parallel composition of the involved automata [6]. For example, the composite of automata R and S from Figure 2 is shown in Figure 3. Observe that conflict $(3, c)$ corresponds to the composite node $< c, c >$.

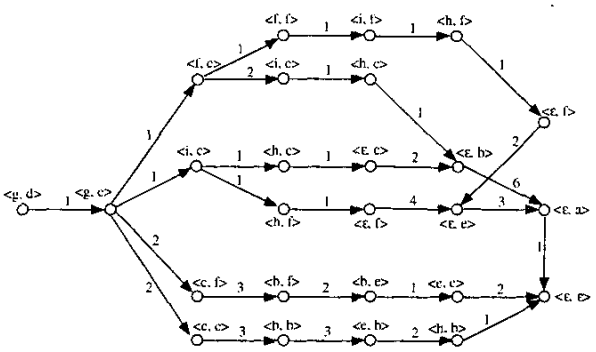

Figure 3: The composite of automata R and S

3 Conflict Resolution

Consider the situation when several incoming agents want to enter the system at the same time. They have access to a common database, containing the models and current legal plans of all active agents in the system. They must also register their optimal paths in the database. Thus, after registration, an incoming agent will know the models of all other incoming agents. Initially, an incoming agent determines the subset of its paths that are conflict-free with the legal plans of the active agents. However, this subset may have resources that are disputed with the corresponding subsets of other incoming agents. Thus, an incoming agent must determine which of these paths are ultimately legal, so that legal paths of different incoming agents are conflict-free. This is the *conflict resolution* phase. If the resultant legal plan of the incoming agent contains at least one legal path, it becomes active. Otherwise, it executes the *accommodation* phase.

A solution of the conflict resolution problem for a given set of incoming agents is a collection of legal plans that are conflict free. A solution S is *less restrictive* than another solution S' if, for each agent, the legal plan in S' is included in the legal plan given by S, and the inclusion is strict for at least one agent. A solution is *least restrictive*, or *maximal*, if no other less restrictive solution exists. While maximal solutions need not be unique, a maximal solution means that no agent can unilaterally improve its legal plan without creating a conflict with some other agent's legal plan (and thus violating the safety constraint). An algorithm that always finds a maximal solution is called *optimal*.

Our algorithmic approach to conflict resolution does not require communication among agents. The resolution is based on a prioritization of agents over the disputed resources. The prioritization can be achieved by applying a set of rules, so that if rule (i) does not give a suitable prioritization, rule (i+1) is applied. For example, the following rules can be used to prioritize the agents in Figure 2. The agent that has priority is the one which: (1) Arrives first at the disputed resource; (2) Has the shortest time to its destination from the moment of arrival at the disputed resource. Thus, agent R has priority for $(3, f)$ and $(6, b)$, while S has priority for $(3, c)$ and $(9, e)$. In our framework, we assume that the prioritization is given, and we focus on how to use it for conflict resolution. For the remaining of the paper, unless otherwise stated, by *resource* we shall mean *disputed resource*.

In [5], we present resolution algorithms for the two-agent case, as well as for the general case, under different conditions on the knowledge available to an agent with respect to the other agents' models and prioritization. The general principle of optimal resolution is that an agent reserves a resource if and only if the following conditions are both met:
(*) The agent has the *highest priority* to the resource among all agents that *have access* to the resource, and
(**) The agent can make successful use of the resource by *completing a legal execution*. This, in particular, implies that an agent is not permitted to reserve a resource (to which it may have priority) if the reservation cannot be applied toward a successful task completion.

For the example in Figure 2, the resolution is done as follows: because S has priority for c, b is inaccessible to R. Hence, S can claim b and complete a legal path. On the other hand, R has priority for f from which it can complete a legal execution. Therefore S cannot reserve e (for which it has priority). The maximal solution is outlined by the thick paths in the figure. Note that even in a maximal solution, there may be disputed resources that remain in the public domain (e.g., e in Figure 2).

In [5], we define a spectrum of possible approaches, as follows. At one end, the most that an agent can do is to take into consideration all the agents and conflicts in the system, including those in which it is not directly involved (assuming it knows the prioritization for all of them). For this case, we give an optimal resolution algorithm. At the other end of the spectrum, the least that an agent should do is to take into consideration only the agents with which it has conflicts, and to ignore the others. Conflicts are resolved with each of the

competing agents, pairwise, the final result being the intersection of the partial results. Since the number of agents conflicting with a given one is usually much lower than the number of all agents in the system, the pairwise approach is computationally efficient, but the result is, in general, not maximal.

4 Accommodation

In the accommodation phase, an incoming agent that obtained an empty legal plan in the conflict resolution phase, can request resources from active agents. Using also public resources (remained in the public domain after conflict resolution), the incoming agent tries to put together at least one legal path to its destination. Active agents cannot reclaim now-public resources, which they could not previously obtain in the resolution phase.

The proposed accommodation algorithm is based on a principle that an active agent must relinquish resources at the request of an incoming agent, provided that the active agent retains at least one legal path to its destination. The accommodation algorithm stops as soon as the incoming agent secures a complete path to the destination, or if it is determined that no solution exists. An accommodation algorithm is *optimal* if it always finds a solution (a legal path for the incoming agent), if one exists. In other words, given an initial condition (i.e., the legal plans of active agents), if any mechanism finds a solution, then an optimal accommodation will also yield a solution under the same circumstances. It can be readily seen that the less restrictive the initial condition is (i.e., the larger the legal plans of the active agents are), the more permissive an optimal accommodation is. That is, if an accommodation is not found for a given initial condition, a solution may still be found for a less restrictive initial condition (for the same model of the incoming agent). Consequently, a maximal solution of the conflict resolution offers greater flexibility for accommodating an incoming agent than a non-maximal solution.

Protocol outline. We confine our attention to the case of a single incoming agent, which tries to obtain a legal path in a system where all the other agents are active. We assume that an agent knows the models and legal plans of the other agents. In our accommodation protocol, called *ACCP*, the incoming agent tries to secure a legal path in its model, by requesting the disputed resources along the path. After soliciting a resource, the agent waits for the owner's response. If the answer is positive, then the agent proceeds to requesting the next disputed resource. If all disputed resources on the path are obtained, then the path is declared legal, and the accommodation succeeds. If a certain resource cannot be obtained, then the current path is abandoned and the agent tests another path in its model. If no path can be obtained, the accommodation fails, and the agent does not enter the system. When an active agent receives a request for a disputed resource contained in its current legal plan, it tries first to remove the resource from the legal plan, in order to satisfy the request. If the curtailed legal plan is non-empty, then the answer is positive. Otherwise, the agent tries to find a new legal path among the previously illegal ones in its model, by employing the same procedure outlined above for the incoming agent with respect to the other active agents. If a path is secured, then it becomes the new legal plan and the answer is positive. If no illegal path could be made legal, then the request is denied. The procedure may be repeated to any needed or specified search depth.

Illustrative example. Consider the incoming agents and prioritization depicted in Figure 4, where the highest priority is 1. For simplicity, transition times are not represented. In phase one, the following conflict resolution is executed:

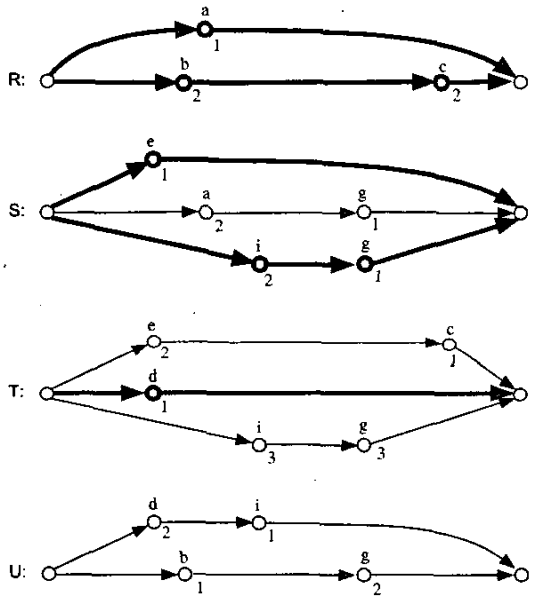

Figure 4: R, S, and T must accommodate U

R, S, and T reserve a, e, and d, respectively. Since i is inaccessible to U (which is blocked at d), S reserves i (it has priority over T), and also g, for which it has priority. Consequently, U cannot use b and therefore R can reserve it, together with c, which is inaccessible to T (which is blocked at e). The legal plans are depicted by thick transitions. Ownership of a resource is outlined by a thick circle. Observe that the solution is maximal. Agents R, S, and T become active, while U executes the accommodation phase, as follows.

U asks first for d. To answer the request, T tries to secure its lowermost path, by requesting i. This is

granted by S, which can still use e. Next, T asks for g, which is also relinquished by S. Thus, T will give d to U. Having obtained d, U asks next for i. Notice that ownership of i has changed from S to T. In general, a resource request is broadcast, because the resources's owner may not be known to the solicitor. In order to give up i, agent T will try its uppermost path, by requesting e. This will be granted by S, after obtaining a from R and g from T. The legal plans and resource ownership at this stage are depicted in Figure 5.

Next, T asks for c. To grant c, R requests a, which

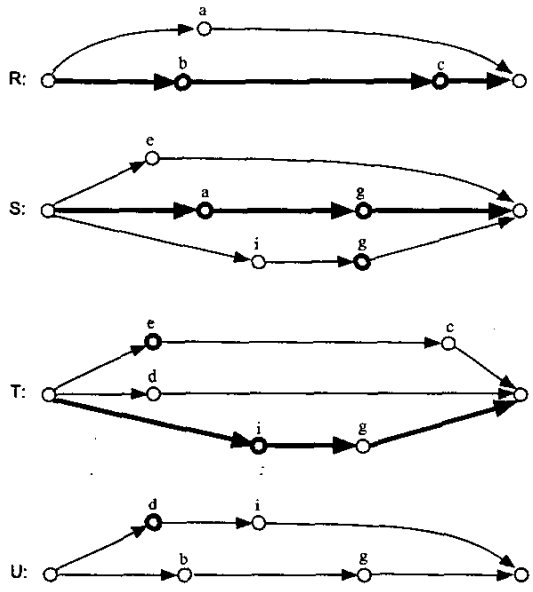

Figure 5: T asks now for c

S cannot relinquish, because neither e nor i can be obtained by S from T: T cannot give up e in order to obtain c and it cannot give up i, which it is currently requested by U. Hence, R fails to get a, and consequently denies c to T, which must abandon its quest for the uppermost path. Moreover, T cannot secure the middle path because it cannot get d from U. Thus, T's answer to U (with respect to i) is negative. Observe that, at this step, T owns e, but it doesn't own g (although g is on a current legal path). Therefore, before sending the answer to U, T must re-acquire g from S. It can be shown ([6]) that such requests are always satisfied. In our case, S will relinquish g to T after obtaining first e from T.

Being refused i, agent U abandons the upper path and tries the lower one. It asks for b, which is given by R after obtaining a from S. Then, U requests g. T needs first to get d from U, and then sends a positive answer to U. The accommodation succeeds, with the following new legal plans. R: the upper path, S: the uppermost path, T: the middle path, and U: the lower path.

The protocol procedures. *ACCP* has three procedures: (1) *RequestHandler*, which processes a resource inquiry, (2) *FindLegalPath*, which selects a path to be secured before giving an answer to a resource request, and (3) *ClaimPath*, wherein an agent tries to obtain the disputed resources along the path selected by *FindLegalPath*. These procedures use a database of variables, as follows. LP is the set of current legal paths. TR is the set of *tentatively reserved* resources. Whenever the agent tries to secure a legal path, TR holds all the disputed resources that have been requested and obtained by the agent on the tested path, until the current computational step. CO is the set of currently owned resources (which are initially those contained in the legal paths). The resources for which the agent must answer, if (and when) it receives a request, are those in CO and TR.

Procedure *RequestHandler* is executed by an active agent upon reception of a request for a disputed resource q, provided that the agent is not already waiting for an answer to a request that it has previously made (see *ClaimPath* below). Let $\widetilde{LP}$ denote the set of all (disputed) resources in LP. If $q \in CO \setminus \widetilde{LP}$, then the answer is positive. If $q \in \widetilde{LP}$, then a set $newLP$ is determined, of all paths in LP which do not contain q. If $newLP \neq \emptyset$, then the answer is positive. Otherwise, procedure *FindLegalPath* is called to determine a new legal path (to be stored in $newLP$). If *FindLegalPath* succeeds, then the answer is positive, otherwise it is negative. In case of a positive answer, the variables change as follows: $CO := CO \setminus \{q\}$, $LP := newLP$, and $CO := CO \bigcup \widetilde{newLP}$.

For an active agent, procedure *FindLegalPath* successively selects agent paths which do not contain the requested resource (q) and a resource in the public domain. It tries to claim each of these paths by calling *ClaimPath* until either a path is secured, or all paths are tested, but none could be obtained. In the latter case, the agent enters a recuperating phase, where the disputed resources in $\widetilde{LP} \setminus CO$ are re-acquired. This is necessary because resources on paths in LP may have been relinquished while trying to secure "illegal" paths. For example, this was the case for resource g in Figure 5. It can be shown ([6]) that such a request is always satisfied, without the need to relinquish other resources in $\widetilde{LP}$. Therefore, if a resource in $\widetilde{LP} \bigcap CO$ is requested by another agent at this stage, then the answer is negative. For the incoming agent, *FindLegalPath* selects paths in the agent's model arbitrarily, until either a legal path is found, in which case the accommodation succeeds, or no path can be obtained, and the accommodation fails.

Procedure *ClaimPath* sends request messages for disputed resources on a path selected by *FindLegalPath*,

until either a request is denied, or all resources are obtained. After sending a request, the agent waits for one reply. If this is positive, then the resource is memorized in TR and the next resource is picked up. If the answer is negative, then $CO := CO \bigcup TR$, $TR := \emptyset$, and the procedure returns. While waiting for an answer, the agent may receive request messages. These messages are answered immediately, without making further requests, as follows. Suppose that at the current computational step agent R is waiting for the answer to the request of resource q_R, currently owned by agent S. Also, S is waiting for the answer to its request of another resource q_S. Suppose that S receives now from some agent T a request for a resource q_T. Note that T cannot be R. S answers in one of the following cases:

1. If $q_T \in TR_S$, then the answer is negative. Notice that all resources in TR_S have been previously obtained by S along the currently tested path, up to q_S. Therefore, it is of no use for S to relinquish a resource in TR_S in order to (possibly) obtain q_S.
2. If $q_T = q_R$, then again the answer is negative. S is trying to accommodate the request for q_R in order to give it to R, not to T.
3. If $q_T \in CO \setminus \{q_R\}$, then the answer is positive and $CO := CO \setminus \{q_T\}$.

The incoming agent never executes *RequestHandler*. It starts the accommodation by calling *FindLegal Path*. When the procedure returns, a message is broadcast, signaling the end of the accommodation. If it was not successful, then all agents must switch back to their initial legal plans. The protocol does not specify the order in which candidate paths are selected by *FindLegalPath*, or resources on a selected path are requested in *ClaimPath*. While the computational and communication efficiency may depend on a particular selection strategy, the correctness and optimality of the protocol do not. We state next the main properties of the *ACCP* (for proofs, see [6]).

Theorem 1 The solution obtained by *ACCP* satisfies the requirements of safety and liveness.

Theorem 2 *ACCP* is optimal.

Implementation issues. Observe that agent models and legal plans change in time. Thus, an agent may have to make a move while being involved in an accommodation. Additional mechanisms must be used to deal with this situation. It is very likely that such a mechanism will require an agent not to relinquish some resources (which in the ideal case can be relinquished). For example, an agent may withhold resources with times within a certain lookahead range.

At a given computational step, the *depth* of the protocol execution is the number of agents waiting for answers to their requests. It can be readily seen that the maximum depth of any execution is $n - 1$, where n is the number of agents. For the incoming agent, a large depth means a long waiting time for an answer. This period can be shortened by limiting the depth to a specified value. Thus, we obtain a protocol that approximates the ideal accommodation up to a given depth.

Notice that *ACCP* does not preserve maximality of legal plans. Moreover, this can be also lost as a result of time passing. To maintain maximality, one can employ optimal conflict resolution among active agents, with respect to the freed resources only. Thus, the legal plans of active agents are augmented, such that a maximal solution is obtained.

In our framework, an agent can attempt to cheat and include non-optimal paths in its declared model, so as to retain more legal paths. The effects of such an action are beneficial to the system, because flexibility is increased, but the agent may end up having to follow a non-optimal path. Thus, an agent has a disincentive to cheat. In fact, in order to increase flexibility, agents may be required to include sub-optimal paths in their models. In this case, existing agents can be forced to deviate to sub-optimal legal paths, in order to allow new agents in the system. In other words, individual performance (path optimality) may be sometimes sacrificed in order to increase global performance.

References

[1] S. Devasia, M. Heymann and G. Meyer, 2002. Automation procedures for Air Traffic Management: a token-based approach. Proceedings of the ACC.

[2] J. K. Kuchar and L. C. Yang, 2000. A review of conflict detection and resolution modeling methods. *IEEE Trans. Intell. Transp. Syst. 1(4)*, pp. 179-189.

[3] R. Jacobsen, 2000. NASA's Free Flight Air Traffic Management Research. NASA Free Flight/DAG-ATM Workshop.

[4] P.K. Menon, G.D. Sweriduk, and B. Sridhar, 1999. Optimal strategies for Free Flight air traffic conflict resolution. *Journal of Guidance, Control and Dynamics 22(2)*, pp. 202-210.

[5] S. Resmerita and M. Heymann, 2003. Conflict Resolution in Multi-Agent Systems, CDC 2003.

[6] S. Resmerita, 2003. A multi-agent approach to control of multi-robotic systems. PhD thesis, Technion.

[7] C. Tomlin, G. Pappas, J. Lygeros, D. Godbole, and S. Sastry, 1996. A Next Generation Architecture for Air Traffic Management Systems. *Lecture Notes in Computer Science 1271*, Springer Verlag.

[8] J.P. Wangermann and R.F. Stengel, 1998. Principled negotiation between intelligent agents: a model for air traffic management. *Artificial Intelligence in Engineering (12)* pp. 177-187.