2009 American Control Conference
Hyatt Regency Riverfront, St. Louis, MO, USA
June 10-12, 2009

An Event Driven Decision Support Algorithm for Command and Control of UAV Fleets

Oktay Arslan and Gokhan Inalhan

Abstract— **In this work, we focus on solving large-scale UAV fleets scheduling problem in dynamically changing (i.e. external event-driven or operator induced selection) scenarios. This autonomous scheduling of planned tasks and allocation of resources is designed to provide real-time decision support to the operator for problem sizes that is intractable or infeasible by one or a set of operators. We begin by analyzing the computational complexity of a well-known Solve & Robustify approach that generates robust and flexible schedules and propose the temporal space partition approach for decreasing the computationally expensive solve step. The improved algorithm, which is refereed as Earliest Start Time Algorithm with Partitioning (ESTA$_P$), divides the larger problem into smaller subproblems by partitioning the temporal space and then iteratively solves the subproblems. Benchmark problem comparisons with the classical ESTA formulation for two hundred tasks indicates that the proposed temporal space partitioning approach improves the computation time forty-fold while only incurring five percent increase in the total completion of the tasks.**

INTRODUCTION

During the last decade, there has been a remarkable increase in usage of Unmanned Air Vehicles (UAVs) in military command and control applications. Growing involvement of UAVs in complex application areas (such as dynamically changing urban rescue operations), the types and the number of tasks easily outgrow one vehicle's (or a set of UAV operators' command and control) limited capabilities and thus it is required a fleet of UAVs to work in a collaborative fashion to achieve desired operational goals. Despite the current trend reducing the role of human in aerial systems, human operators are still needed for supervisory control and high level decision making [6]. In this paper, we provide an event driven decision support algorithm for temporal scheduling of tasks across UAV assets based on Solve & Robustify approach [13], [14]. Specifically, the algorithm is designed and tested to provide hard real-time decision support (i.e. on the order of seconds) to the operator for scenarios involving over hundred tasks across multiple assets.

A key part of such an event-driven process hinges on the coordination of the target/task assignments and distributions across UAV assets through supervisory commanding. However, it is not always feasible to apply basic supervisory command and control for a large number of unmanned vehicles performing complex missions with strict time constraint. Therefore, a decision support or autonomous decision-making system can be designed for the commander in order to reduce the workload. Figure 1 illustrates the overall process of such a decision support system. Such a decision support system requires integration of two important operations: planning - the problem of determining which activities to perform, and scheduling - the problem of temporal allocation of resources to these activities.

Basically, the steps of decision making can be tracked as follows: planning, scheduling and low-level mission specific task planning. The planning process is triggered by external monitored events or requests. Following the taxonomy as presented in [10], the planning step proceeds by selecting suitable actions (UAV missions) from a predefined action sets by an expert system. Then, resource allocation is done for these action sets regarding operation constraints (time, environment) and a group of target is selected regarding to the selected actions and resources. The second process is scheduling of these action sets under resource constraints by satisfying specified temporal constraints. In this work, we assume that the set of activities requiring resources are specified in advance and focus on temporal allocation of a set activities regarding resource and strict time constraints as fast as possible. Specifically, Solve & Robustify approach is used as a base algorithm in order to handle executional uncertainty in the dynamic mission environment. The algorithm used in the Solve step, Earliest Start Time Algorithm (ESTA)[12], is modified with temporal space partitioning to provide real-time solutions to the operator. Benchmark problem comparisons with the classical ESTA formulation for two hundred tasks indicates that the proposed temporal space partitioning approach improves the computation time forty-fold while only incurring five percent increase in the total completion of the tasks. After finding a feasible schedule, the low level task planning problem is solved by the algorithm given in [8]. An experimental illustration of this within a mission simulator can be found in [1].

The organization of this work is as follows: In Section 1, the scheduling step of the decision support system (DSS) is formulated as a classical Resource Constraint Project Scheduling with Maximum Lag (RCPSP/max) problem. A brief literature survey about different approaches for solving this classic problem is given. Specifically, Solve & Robustify approach is introduced for robust and flexible schedule generation. Following a brief computational analysis of Solve & Robustify, the temporal space partitioning is proposed to reduce computational load of solve step. In Section 2,

This work is supported by TUBITAK Graduate Fellowship Funds

O. Arslan - Research Assistant, Controls and Avionics Laboratory, Istanbul Technical University, Maslak, Istanbul, Turkey oktay.arslan@itu.edu.tr

G. Inalhan - Corresponding Author, Assistant Professor, Faculty of Aeronautics and Astronautics, Istanbul Technical University, Maslak, Istanbul, Turkey inalhan@itu.edu.tr

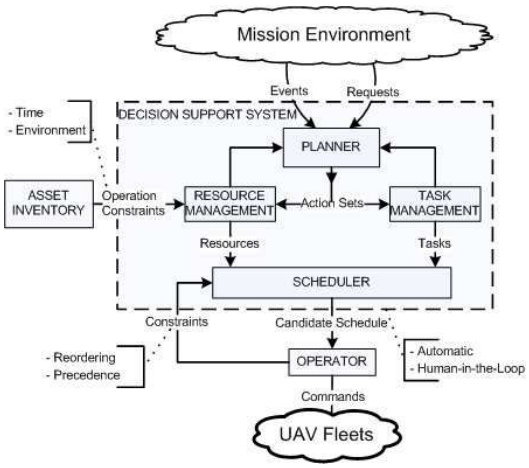

Fig. 1. Event-Driven Decision Support Architecture

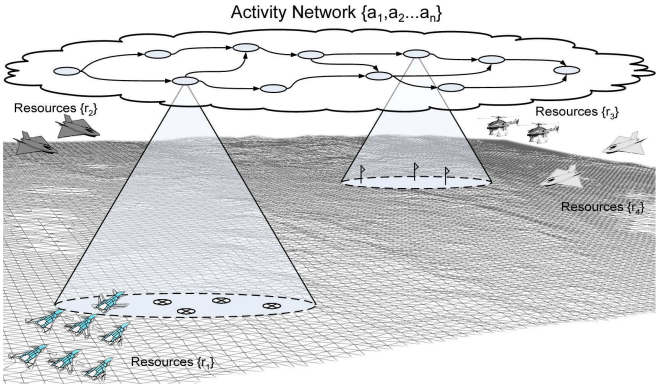

Fig. 2. Mission Environment

the basics of the ESTA algorithm is reminded. Specifically, construction of an infinite capacity solution and leveling of resource demand by posting precedence are used in the proposed greedy scheduling algorithm (ESTA$_P$). Then, the partitioning heuristics used in the algorithm and the overall greedy algorithm are described step by step. Finally, in the last section, experimental evaluations and results are given to demonstrate the efficiency of the proposed algorithm across a range of benchmark problems of increasing activity size.

I. The Scheduling Problem - RCPSP/max and Solution Approaches

Figure 2 illustrates an overview of distributed of command and control (C2) problem in which missions are accomplished by human operators and UAV fleets. From the operators' perspective, the C2 problem involves scheduling of several missions consistently regarding to the number of UAVs and temporal constraints between the missions. This problem can be defined as a scheduling problem by using the following definitions:

- Each mission corresponds to an activity which has to be scheduled and has an estimated duration
- UAV fleets correspond to a set of renewable resources and each UAV with different capabilities (i.e. combat, imaging) define a new type of resources
- Each mission requires a number of UAVs for all its duration
- Structural dependencies between missions, physical and logistic constraints define a set of temporal constraints. For example, a target must be destroyed within some periods of time immediately after its designation, the total completion time of set of missions assigned to a UAV can not exceed its endurance.

This scheduling problem can be stated as finding a temporal allocation and synchronization of a set of renewable resources $R = \{r_1...r_m\}$ for given a set of activities (missions) $V = \{a_1...a_n\}$ over time regarding a set of temporal constraints. In this work, we use the Resource-Constrained Project Scheduling Problem with Minimum and Maximum time lags (RCPSP/max) as a reference [2] and each activity must be performed regarding the following constraints:

- each activity a_j has a duration dur_{a_j},a start time s_{a_j} and an end time e_{a_j} such that $e_{a_j} = s_{a_j} + dur_{a_j}$;
- each activity a_i requires the use of req_{ik} units of the resource r_k for all of dur_{a_i}
- a set of temporal constraints c_{ij} each defined for a pair of activities (a_i, a_j) and of the form of $c_{ij}^{min} \leq s_{a_j} - s_{a_i} \leq c_{ij}^{max}$
- each resource r_k has an integer capacity $max_k \geq 1$;

The objective is to determine the starting times of all activities in V , $S = (s_1, s_2, ..., s_n)$, which are temporally consistent and satisfy resource capacity constraints.

There are different models and approaches for solving RCPSP/max in Operations Research community. One the most used approach is solving the problem in branch and bound schema. An early analysis focusing on mathematical properties of the problem was given in [2] and a branch and bound approach was proposed which is based on extending the the set precedence relations in order to resolve conflicts in the problem. In that work, the solution space was modeled as a network of temporal constraints and the concept of *forbidden sets* and *reduced forbidden sets* to define resource conflicts and resource conflicts with a minimal number of activities respectively. Then a systematic Branch and Bound (B&B) was formalized as posting precedence relations in the problem to remove all reduced forbidden sets in an initial, time feasible solution step by step.

Another approach for solving scheduling problem is formulating RCPSP/max as constraint satisfaction problem and there are two main models in the CSP scheduling literature: start time assignment [15], [11] and precedence constraint posting model [5]. In the first model, time points that indicate the start times of activities are defined as decision variables of the problem and aim of the CSP search is finding a consistent (both time and resource feasible) assignment of start time values. In the second case, various ordering decisions between sets of activities that are competing for the same resources are considered as decision variables rather than start times of activities and the aim of the CSP search is posting a consistent set of precedence constraints

that removes all the resource peaks in the resource profiles of the resources. Since,in this approach, there is no any specific start time assignment for each activity during the search process or in the final solution, it is less commitment than the first model. As a result, in the uncertain dynamic environment, this model has advantages as having capability of generating more robust and flexible schedules than the first model.

Due to requirement of robust and flexible schedules to handle executional uncertainty in the dynamic mission environment, one of the successful CSP based algorithm Solve & Robustify approach given in [14] is used as base algorithm and the proposed temporal space partitioning method is implemented within the Solve & Robustify approach.

The Temporal Space Partitioning

During the implementation of the Solve & Robustify approach, it is observed that the first step which is computation of a fixed-time solution increasingly dominates the overall solution time. The most of computation done in this step is contributed by leveling resource demand by posting precedence. The leveling resource demand operation basically consists of two steps, namely, determination of which precedence to post and finally propagating the precedence constraint along the temporal network.

First, all contention peaks[1] are collected for a given earliest start solution. Then, Minimal Critical Sets[2] (MCSs) are computed for each peak. Since the complete analysis of MCSs has exponential computation in nature, a *sampling strategy* of polynomial complexity given in [4] is used to collect some subset of MCSs for each peak by abstaining from combinatorial explosion of the search. Unfortunately, as the size of the problem increases regarding to the both number of resource types and number of temporal variables in the network, the number of peaks for a given earliest start solution increases too fast in most cases and this tends to increasingly spend too much time for collecting of MCSs even if using sampling strategies of polynomial complexity.

Second, after selecting the MCS to be resolved, it is solved by posting a simple precedence constraint between pair of competing activities. In order to maintain the consistency of the problem's temporal information, that constraint is propagated along the temporal network by using all pairs shortest path algorithms, which are $O(n^2)$ in space and $O(n^3)$ in time according to the number of temporal variables in the network. Unfortunately, as problem size increases, computation of propagation of the constraints dramatically takes too much time.

Thus, in order to solve the UAV mission inherent large-scale scheduling problem in real-time, we propose an approach that divides the larger problem into smaller subproblems by partitioning the temporal space and then iteratively solving the subproblems. To obtain balanced partitioning, a temporal space flattening method is developed. This method eliminates overlapping of conflicting activities through temporal sequencing in an expanded time space. To obtain a consistent integration of subproblem solutions, a binding method is introduced. This method propagates the maximum horizon constraint over the whole network to generate the main solution. These methods are explained in the following section in detail.

[1]Contention peak is a set of activities which are competing for the same resources and their simultaneous execution exceeds the resource capacity

[2]A Minimal Critical Sets, is a contention peak such that any subset of activities belonged to MCS is not a contention peak

II. A Greedy Algorithm Based on the Temporal Space Partitioning

Temporal space partitioning approach consists of three basic steps: (a) a temporal partitioning algorithm is first used to divide the problem into subproblems; (b) the first subproblem is solved recursively by using precedence posting scheme; then the second problem is redefined with regarding the updated partial solution and finally the second problem is solved in the same way; (c) the solutions obtained from first and second subproblems are integrated to maintain consistency in temporal space.

The proposed approach is based on the Earliest Start Time Algorithm (ESTA), given first in [3] and further improved in [4]. The steps of ESTA are summarized as follows and these steps are separately used in the proposed algorithm.

Summary of ESTA Procedure

Construct an infinite capacity solution: In this step, the problem is formulated as an STP [7] temporal constraint network and a time feasible solution that ignores resource constraints is computed by propagating constraints on the underlying temporal constraint network.

Level resource demand by posting precedence: Resource demand profile for each type of resources is computed for all time values and the time intervals that resource demands exceed resource capacity are detected. Finally, these detected resource conflicts are resolved by iteratively posting simple precedence constraints between pairs of competing activities.

Separating Steps of ESTA

The steps of ESTA procedure are separately used in the proposed algorithm which is referred as $ESTA_P$, where the suffix P indicates the partitioning. As shown in Algorithm 1, before solving the problem recursively, an infinite capacity solution is computed by the first step taken from ESTA procedure. Then, the resource demand of this partial solution is leveled until there is no any resource conflict in its earliest start solution by calling $ESTA_{PR}$ procedure.

In the ESTA procedure, the resource demand of earliest start solution is leveled by posting simple precedence constraints until eliminating all resource conflicts. However, this is required only when the size of the subproblem is small than predefined minimum size value in the proposed approach. In other cases, as it is shown in Algorithm 2, the leveling step is repeated for a given number of times as required in special circumstances.

Algorithm 1 $ESTA_P\ (P, h_{max})$

Input: a problem P and horizon of the problem h_{max}
Output: a solution S

1. $S^P \leftarrow CreateCSP(P)$
2. $S^P \leftarrow S^P \cup \{t_e - t_s \leq h_{max}\}$
3. $S^P \leftarrow ESTA_{PR}(S^P, h_{max})$
4. **if** Exists failure in S^P **then**
5. **return** FAILURE
6. **else**
7. $S \leftarrow ComputeESS(S^P)$
8. **end if**
9. **return** S

Algorithm 2 $LevelResourceDemand\ (S^P, n \leftarrow \infty)$

Input: a partial solution of problem S^P
Output: a partial solution of problem S^P

1. **for** $i = 1$ to n **do**
2. $C_S \leftarrow$ Select-Conflict-Set(S^P)
3. **if** $C_S = \emptyset$ **then**
4. **return** S^P
5. **else if** Exist an unresolvable conflict in C_S **then**
6. **return** FAILURE
7. **else**
8. $(a_i \prec a_j) \leftarrow$ Select-Leveling-Constraint(C_S)
9. $S^P \leftarrow S^P \cup (a_i \prec a_j)$
10. **end if**
11. **end for**
12. **return** S^P

Solving the Problem by the Temporal Space Partitioning

Posting precedence constraints and propagation of them are fundamental operations in ESTA and propagation of constraints over the temporal network sometimes does not cause an update in the temporal information of some activities. Obviously, the effect of propagation of constraints over temporal networks is closely correlated with how deeply the activities of temporal networks are interconnected. The distinction of sensitive and less-sensitive parts of temporal networks to constraint propagation are more evident in the temporal networks which are sufficiently flexible and have enough temporal slacks among their activities. The temporal space partitioning approach is based on exploiting such properties of temporal networks by dividing them into subnetworks.

Algorithm 3 shows the $ESTA_{PR}$ procedure where P and R indicate partitioning and recursive, respectively in more detail. It takes two parameters as input: (1) a partial solution of the problem P S^P, and (2) maximal horizon of the problem P h_{max}.

First, the number of activities in the problem P is compared with the minimum size value. If it is less than minimum size, then all the resource conflicts detected in the partial solution S_P are eliminated by the *LevelResourceDemand* procedure at Step 2. Otherwise, the problem is divided into subproblems by partitioning the its temporal space.

The temporal space of problem P is divided into 2 subregions by cutting it from the time point t_c, where suffix c indicates critical within the loop at Step 6. First, the critical time point t_c is computed by *CreatePartition* procedure at Step 7 and then the first subproblem is created by collecting activities from problem P such that their earliest start time value is less than t_c. If there is any activity in the first subproblem and the size of the first subproblem is less than the size of the problem P, then the inner loop is broken. Otherwise, this means that configuration of the current temporal space of problem P is too shrunk and it needs to be flattened. Flattening of the temporal space increases the divisibility and it is flattened by posting one precedence constraint by *LevelResourceDemand* procedure at Step 12. The inner loop repeats until dividing the problem P into smaller subproblems.

After successfully dividing the problem P, the current state of the partial solution S^P is saved by assigning it to S_0^P at Step 14 for further restore operation. Then, a dummy finish time point t_{de} is included in subproblem P_1 at Step 15 and the horizon constraint between t_s and t_{de} is posted in the next step. Finally, the partial solution S^{P_1} is solved by calling the same procedure $ESTA_{PR}$, recursively. Later, the $ESTA_{PR}$ begins to create second subproblem by collecting activities from problem P such that their earliest end time value is greater than t_c. Then, a dummy start time point t_{ds} is included in the subproblem P_2 at Step 22 and the horizon constraint between t_{ds} and t_e is posted. As in the case of the first subproblem, the partial solution S^{P_2} is solved by calling the same procedure $ESTA_{PR}$, recursively. If there is no any failure in both computed solutions, then S_{P_1} and S_{P_2} are integrated by simply propagating the updated temporal information along the temporal network of problem P at Step 28. If a time feasible solution for problem P is obtained, then $ESTA_{PR}$ procedure returns this solution. Otherwise, the S_P is assigned to its previously saved state S_0^P at step 34, then temporal space of problem P is flattened by posting a simple precedence constraint by *LevelResourceDemand* procedure at step 35. The outer loop is repeated infinitely until a time feasible solution exists or a failure is encountered in the solution of the subproblems.

Partitioning Heuristic

The partitioning heuristic is based on dividing the temporal space of the problem by a critical time point t_c such that the resulted subproblems are balanced with regard to the number of activities. The first subproblem consists of activities such that their earliest start time value is less than earliest time value of the t_c and the certain number of activities in the second subproblem only can be determined by checking the $est(e_{a_i}) > t_c$ for all activities in the problem after solving the first subproblem. Therefore, the normalized percentage of the domains of the time values respect to the t_c is calculated for the second subproblem instead of number of activities. The calculated number n_2 is just an approximate value about the number of activities of second subproblem. After calculating n_1 and n_2 a cost value is calculated for

Algorithm 3 ESTA$_{PR}(S^P, h_{max})$

Input: a partial solution of problem S^P and horizon of the problem h_{max}
Output: a partial solution S^P

if $size(P) <$ minimum size **then**
  $S^P \leftarrow LevelResourceDemand(S^P)$
  return S^P
else
  loop
    loop
      $t_c \leftarrow CreatePartition(S^P)$
      $S^{P_1} \leftarrow \forall a_i | s_{a_i} \in P \wedge est(s_{a_i}) < t_c$
      if $0 < size(P_1) < size(P)$ **then**
        break
      end if
      $S^P \leftarrow LevelResourceDemand(S^P, 1)$
    end loop
    $S_0^P \leftarrow S^P$
    $P_1 \leftarrow P_1 \cup t_{de}$
    $S^{P_1} \leftarrow S^{P_1} \cup \{t_{de} - t_s \leq h_{max}\}$
    $S^{P_1} \leftarrow ESTA_{PR}(S^{P_1}, h_{max})$
    if Exists failure in S^{P_1} **then**
      return FAILURE
    else
      $S^{P_2} \leftarrow \forall a_i | e_{a_i} \in P \wedge est(e_{a_i}) > t_c$
      $P_2 \leftarrow P_2 \cup t_{ds}$
      $S^{P_2} \leftarrow S^{P_2} \cup \{t_e - t_{ds} \leq h_{max}\}$
      $S^{P_2} \leftarrow ESTA_{PR}(S^{P_2}, h_{max})$
      if Exists failure in S^{P_2} **then**
        return FAILURE
      else
        $S^P \leftarrow S^{P_1} \cup S^{P_2}$
        if S^P is time feasible **then**
          return S^P
        end if
      end if
    end if
    $S^P \leftarrow S_0^P$
    $S^P \leftarrow LevelResourceDemand(S^P, 1)$
  end loop
end if

Algorithm 4 CreatePartition (S^P)

Input: a partial solution of problem S^P
Output: a time point t_c

$n_1 \leftarrow 0$, $n_2 \leftarrow 0$
for all time point $t_j \in P$ **do**
  for all activity $a_k \in P$ **do**
    if $est(s_{a_k}) < est(t_j)$ **then**
      $n_1 \leftarrow n_1 + 1$
    end if
    if $d(t_j, e_{a_k}) > 0$ **then**
      $n_2 \leftarrow n_2 + \frac{d(t_j, e_{a_k}) + min(d(e_{a_k}, t_j), 0)}{d(t_j, e_{a_k}) + d(e_{a_k}, t_j)}$
    end if
  end for
  $c_j \leftarrow Cost(n_1, n_2, n)$
end for
$t_c \leftarrow t_j | c_j = \min_{t_i}(c_i)$
return $est(t_c)$

t_c and for the simplicity the cost function is selected as $\frac{n_1^3 + n_2^3}{n^3}$ inspired from the propagation complexity ($O(n^3)$) of precedence constraints over temporal network. Finally, *CreatePartition* procedure shown in Algorithm 4 returns the earliest time value of t_c with the minimum cost values.

III. Experimental Results and Evaluations

Both ESTA$_P$ and ESTA algorithms are implemented in C++ and we have compared their performance on the three sets of benchmark problems taken from the RCPSP/max problem repository [9], [16]. These sets are *UBO50*, *UBO100* and *UBO200* of 90 instances of problem of different size 50×5, 100×5 and 200×5 (number of activities $\times$ number of resources). The numbers of solvable instances are 73, 78, and 80 respectively. The rest of the problems have provably infinite lower bounds for makespan. The parameters of MCSs sampling strategy are set to $\delta \leftarrow 2$ and $s_f \leftarrow 100$ values. The maximal horizon h_{max} is set to 5000 in order to find a solution quickly by searching within a sufficiently large horizon.

Evaluation criteria

The following performance measures are calculated for comparative analysis of both algorithm on different problem sets:

- $N_{feas}\%$ the percentage of problems feasibly solved for each benchmark set
- t_{mks} average makespan of the solutions
- t_{cpu} average CPU-time in seconds spent to solve instances of the problem set
- N_{pc} the number of leveling precedence constraints posted to solve a problem
- $\Delta_{LB}\%$ the average of percentage relative deviation from known lower bound

First, both algorithms are not able to solve all the problem instances in the benchmark set due to subset of instances which their infeasibility is proven by branch and bound algorithm. However, they solve all the problems which have a finite lower bound found by several algorithms [16].

Second, the average makespan of the solutions found by the ESTA$_P$ algorithm is slightly larger than the ones found by ESTA algorithm and this results in slightly larger deviation from lower bound of the solutions. However, there is a major improvement in the average computation time and the number of posted constraints in ESTA$_P$ over ESTA and the ESTA$_P$ algorithm dominates the ESTA algorithm across all problem sets for t_{cpu} and N_{pc} metrics as the size of the problem increases.

The ESTA$_P$ is tested on the benchmark set UBO50 for different values of the minimum partition size and it improves

TABLE I

PERFORMANCE OF THE ALGORITHMS (UBO50)

UBO50	t_{mks}	$\Delta_{LB}\%$	t_{cpu}	$N_{feas}\%$	N_{pc}
ESTA$_{P11}$	217.671	28.757	0.360	77.778	54.471
ESTA$_{P15}$	217.099	28.247	0.383	78.889	55.141
ESTA$_{P16}$	217.306	27.531	0.400	80.000	56.694
ESTA$_{P17}$	**218.973**	**27.459**	**0.373**	**81.111**	**56.904**
ESTA$_{P20}$	218.466	27.462	0.452	81.111	60.480
ESTA$_{P21}$	218.699	27.595	0.462	81.111	59.384
ESTA	**213.603**	**24.455**	**4.004**	**81.111**	**74.890**

TABLE II

PERFORMANCE OF THE ALGORITHMS (UBO100)

UBO100	t_{mks}	$\Delta_{LB}\%$	t_{cpu}	$N_{feas}\%$	N_{pc}
ESTA$_{P12}$	**423.167**	**30.970**	**3.075**	**86.667**	**120.077**
ESTA$_{P15}$	419.705	29.833	3.121	86.667	123.538
ESTA$_{P20}$	418.436	29.697	3.166	86.667	128.910
ESTA$_{P25}$	419.141	30.100	3.345	86.667	132.974
ESTA	**407.286**	**25.645**	**79.214**	**86.667**	**195.753**

the computation time ten-fold by decreasing it from 4.004 to 0.373 seconds. As seen in Table I, the performance of the algorithm is closely correlated with the minimum partition size parameter. If the value of the parameter decreases until a certain value, then the algorithm has a better performance in the sense of the computation time and number of the posting constraints. But, if the parameter is set to too small values, then the $ESTA_P$ can not solve some instances of problems because of the inconsistency of time constraints, while the ESTA can.

As seen in the Table II ESTA$_P$ has made great improvement in computation time by decreasing it from 79.214 to 3.075 seconds, while the resulted solutions have 5 percent larger deviation than the ones found by ESTA.

The ESTA$_P$ algorithm solves the problems of the benchmark set UBO200 in average 31.782, while the ESTA solves in 1462.83 seconds as shown in the Table III. Thus, ESTA$_P$ algorithm reaches the highest rate of improvement of computation time, forty-fold, over the ESTA algorithm.

IV. CONCLUSIONS AND FUTURE WORKS

In this work, a large-scale UAV fleet-task scheduling algorithm is given for an event driven decision support system. Both the proposed algorithm ESTA$_P$ and one of the well-known algorithm ESTA are implemented in C++ and a comparative analysis is given on sets of benchmark problems with increasing number of activities at the end of the paper. The results show that the proposed algorithm provides a significant improvement in computation time but founds solutions of slightly larger makespan. However, the enormous improvement on computation time provide us to solve the problem of large sized in realtime and this compensate the negative results of slightly larger makespan of the solutions.

As seen in the tables, the performance of the algorithm is sensitive to the minimum partition size parameter. A good analysis of selecting value of the minimum partition size and development of the expert system to solve the planning problem are left as future works.

TABLE III

PERFORMANCE OF THE ALGORITHMS (UBO200)

UBO200	t_{mks}	$\Delta_{LB}\%$	t_{cpu}	$N_{feas}\%$	N_{pc}
ESTA$_{P18}$	770.974	29.970	26.761	85.556	254.961
ESTA$_{P20}$	765.481	28.987	33.866	85.556	261.091
ESTA$_{P25}$	**763.474**	**28.603**	**31.782**	**86.667**	**275.897**
ESTA$_{P30}$	759.766	27.987	33.159	85.556	281.169
ESTA	**751.962**	**26.946**	**1462.83**	**86.667**	**461.436**

REFERENCES

[1] O. Arslan, B. Armagan, and G. Inalhan. *Development of a Mission Simulator for design and testing of C2 Algorithms and HMI Concepts across Real and Virtual Manned-Unmanned Fleets*. Lecture Notes in Control and Information Sciences. Springer, 2008. (To appear).

[2] M. Bartusch, R. H. Möhring, and F. J. Radermacher. Scheduling project networks with resource constraints and time windows. *Annals of Operations Research*, 16(1):199–240, 1988.

[3] A. Cesta, A. Oddi, and S. F. Smith. Profile-based algorithms to solve multiple capacitated metric scheduling problems. In *In Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems (AIPS-98*, pages 214–223. AAAI Press, 1998.

[4] A. Cesta, A. Oddi, and S. F. Smith. An Iterative Sampling Procedure for Resource Constrained Project Scheduling with Time Windows. *International Joint Conference On Artificial Intelligence*, 16:1022–1031, 1999.

[5] C. C. Cheng and S. F. Smith. Generating feasible schedules under complex metric constraints. *Proceedings of the 12th national conference on Artificial intelligence (vol. 2) table of contents*, pages 1086–1091, 1994.

[6] M. L. Cummings, S. Bruni ans S. Mercier, and P. J. Mitchell. Automation architecture for single operator, multiple uav command and control. *The International Command and Control Journal*, 1(2), 2007.

[7] R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49(1-3):61–95, 1991.

[8] S. Karaman and G. Inalhan. Large-scale task/target assignment for uav fleets using a distributed branch and price optimization scheme. In *Int. Federation of Automatic Control World Congress (IFAC WC'08)*, Seoul, South Korea, jun 2008.

[9] R. Kolisch, C. Schwindt, and A. Sprecher. Benchmark instances for project scheduling problems. *Project Scheduling: Recent Models, Algorithms, and Applications*, 1999.

[10] C. Nehme, M. L. Cummings, and J. Crandall. UAV Mission Hierarchy, 2006.

[11] W. Nuijten and C. Le Pape. Constraint-Based Job Shop Scheduling with IILOG Scheduler. *Journal of Heuristics*, 3(4):271–286, 1998.

[12] N. Policella. *Scheduling with uncertainty: A proactive approach using Partial Order Schedules*. PhD thesis, University of Rome "La Sapienza", Rome, March 2005.

[13] N. Policella, A. Oddi, S. F. Smith, and A. Cesta. Generating Robust Partial Order Schedules. *Lecture Notes in Computer Science*, pages 496–511, 2004.

[14] N. Policella, S. F. Smith, A. Cesta, and A. Oddi. Generating Robust Schedules through Temporal Flexibility. In *Proceedings of the* 14^{th} *International Conference on Automated Planning & Scheduling, ICAPS'04*, 2004.

[15] N. Sadeh. Look-ahead techniques for micro-opportunistic job shop scheduling. *Report CS91-102, Carnegie Mellon Univ., Pittsburg*, 1991.

[16] C. Schwindt. Project generator progen/max and psp/max-library. http://www.wior.uni-karlsruhe.de/LSNeumann/Forschung/ProGenMax/rcpspmax.html.