Human Factors in Design, Engineering, and Computing, Vol. 238, 2026, 1640–1650
https://doi.org/10.54941/ahfe1008392

An Intelligent Scheduling Capability for ISR Collection Assignment

Jeremy Ludwig[1], Annaka Kalton[1], Forest Finnigan[1], Katelyn Clements[2], and Jerred Holt[2]

[1]Stottler Henke Associates, Inc., San Mateo, CA 94402, USA
[2]USAF AFMC 711 HPW/RHW, USA

ABSTRACT

Intelligence, Surveillance, and Reconnaissance (ISR) operations require rapidly assigning prioritized collection requests to available assets while adapting to changing operational conditions. The increasing volume of collection requests and the need for asset coordination make automated scheduling essential for effective ISR operations. By automating the assignment and dynamic reallocation of collection requests, such systems can reduce planner workload while also enabling autonomous mission management capabilities. This paper focuses on the specific problem of assigning prioritized ISR collection requests to planned air missions, where the objective is to maximize the number of assigned collections while favoring higher-priority requests. This ISR assignment problem differs from standard scheduling problems due to domain-specific collection feasibility constraints and the need for rapid rescheduling in response to evolving threats, route adjustments, and weather changes. The primary contribution of this work is the design and evaluation of a domain-specific scheduling capability for ISR collection assignment that addresses collection feasibility constraints and the dynamic replanning requirements. The paper describes the system architecture and scheduling methods and presents results comparing standard (baseline) and bottleneck scheduling approaches across both static and dynamic scenarios. The resulting capability can both support human planners through decision support and operate as an autonomous scheduling component.

Keywords: Intelligent scheduling, Decision support systems, Autonomous mission management

INTRODUCTION

Intelligence, Surveillance, and Reconnaissance (ISR) is a pivotal component of military and security operations, providing critical information for strategic decision-making and tactical execution. The demands placed on ISR capabilities have intensified due to the increasing volume of data generated by advanced sensors, the complexity of modern operational environments, and the need to coordinate a diverse array of collection assets effectively. Consequently, the manual planning and scheduling of ISR collections is becoming increasingly challenging, necessitating the development and implementation of automated scheduling solutions. Such capabilities improve planning efficiency and responsiveness for human operators while also supporting autonomous mission management systems.

Received August 6, 2026; Revised September 25, 2026; Accepted October 3, 2026; Available online December 1, 2026

© 2026 The Authors. This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 License.
For more information, see https://creativecommons.org/licenses/by-nc-nd/4.0/

This paper addresses one important aspect of ISR planning: assigning ISR collection requests (collections) to existing air missions (Coman & Veum, 2014). Each collection request includes a location, the type of data to collect, and its relative priority. Air missions include one or more aircraft, the sensors available to the aircraft, assigned routes, and timing estimates along the route. The goal of the scheduler is to assign collection requests (tasks) to air missions (resources) at feasible locations and times within each aircraft's on-station period. A high-quality schedule maximizes the number of assigned collections while favoring higher-priority collections. What makes the ISR assignment problem different from standard scheduling problems is the combination of domain-specific collection feasibility constraints and the need for rapid dynamic replanning.

Collection feasibility depends on both the capabilities of an aircraft and its ability to reach a location from which the requested collection can be performed at an appropriate time. The aircraft must be equipped with a sensor that matches the requirements of a collection request. Given the planned route and expected timing, there are only limited locations and times at which the collection can potentially be performed due to sensor limitations, geography, weather, and other operational constraints. Allocating a collection request along an aircraft's route requires a feasibility map describing the valid collection windows for each aircraft and sensor. Additional collection constraints, such as earliest or latest start times, may further reduce valid assignment windows.

The ISR collection problem is also characterized by rapidly changing priorities and emerging collection requests. Aircraft may require maintenance or be reassigned to higher-priority missions, planned routes may change, and deteriorating weather may affect sensor capabilities. The scheduler must be able to rapidly reschedule collections across aircraft before or during mission execution.

The primary contribution of this paper is the design, implementation, and evaluation of a domain-specific automated scheduling capability for ISR collection assignment. The overall system combines a mission management front end, a backend feasibility service, and a scheduling engine. The scheduler is built from a general-purpose intelligent scheduling framework. We extend this framework with ISR-specific constraints and heuristics to prioritize collection requests while respecting feasibility along planned mission routes. The resulting capability provides decision support to human planners and could also be integrated as an autonomous scheduling component within larger mission management systems. The system is evaluated using standard (STD) and bottleneck (BN) scheduling modes across scenarios involving both static planning and dynamic replanning.

RELATED WORK

The scheduler developed in this work builds on an existing intelligent scheduling framework designed as a configurable foundation for constructing domain-specific schedulers across a wide range of domains and problem types (Ludwig et al., 2017). The framework combines graph-based analysis with heuristic scheduling to rapidly generate high-quality solutions.

Its configurable representations of tasks, resources, constraints, and objectives support efficient adaptation to new domains with limited custom engineering. This makes it well suited to the ISR scheduling problem.

General-purpose planning and scheduling frameworks such as OZONE, EUROPA, and ASPEN demonstrate how reusable architectures can be specialized through domain-specific constraints, heuristics, and resource models (Barreiro et al., 2012; Fukunaga et al., 1997; Smith et al., 1996). More recent work continues to apply a range of exact, constraint-based, heuristic, and metaheuristic methods to resource-constrained scheduling problems. Examples include CP-SAT, which combines constraint programming, satisfiability, and integer programming (Perron & Didier, 2025), and recent surveys of multi-project and dynamic scheduling methods (Ding et al., 2023; Gómez Sánchez et al., 2023; Reijnen et al., 2025).

Although many scheduling approaches could be applied to ISR collection scheduling, the objective of this work was not to compare competing algorithms. Instead, we selected an existing general-purpose scheduling framework as a practical and cost-effective basis for rapid system development.

METHODS

This section describes the overall system and scheduling framework, the integration among the three system components, the test data and scenarios, and the scheduler customizations developed for automated ISR collection assignment.

System Overview

The system combines three containerized components connected through REST APIs: a collection-management application, an ISR feasibility service, and a scheduling service. The management application provides missions, routes, and collection requests; the feasibility service identifies valid collection windows; and the scheduler assigns collections to aircraft and times while respecting priority, timing, and feasibility constraints.

Scheduling Framework

The scheduler is built on a configurable intelligent scheduling framework that combines graph analysis with heuristic scheduling. The framework represents tasks, resources, constraints, and objectives through reusable components that can be adapted to specific domains. For ISR collection scheduling, it was extended with domain-specific preprocessing, prioritization, feasible-window search, resource-selection criteria, explanations, and conflict repair.

System Integration

The round-trip integration between the collection management application, the ISR feasibility engine, and the scheduler operates sequentially. The process begins within the collection management application, where users select desired missions and collection requests and then initiate scheduling by pressing the designated button.

Following this user action, the collection management application sends the chosen missions, associated routes, and collection requests to the ISR feasibility engine. The purpose of this step is to obtain a detailed feasibility report from the engine, which outlines the viability of each potential collection for every aircraft/sensor/route combination on a minute-by-minute basis.

Next, the collection management application transmits the original missions, routes, and collection requests, now accompanied by the generated feasibility report, to the scheduling tool. Upon receiving this data set, the scheduler automatically generates an optimized schedule. This involves assigning the collection requests to specific missions at designated times, ensuring all constraints are respected, including the feasibility windows. Finally, completing the integration loop, the scheduler returns the generated schedule back to the collection management application. This allows the schedule to be displayed to the user, along with a list detailing any issues that may have been encountered during the automated scheduling process.

Test Data

The data contained four missions, which include aircraft along with available sensors, detailed routes, and mission timing. There were 80 standard collection requests and 30 additional collections representing late-breaking requests that require dynamic replanning. Each collection included a latitude/longitude location, type, and an assigned priority of 1, 2, or 3 (1 being highest). Of the 110 collections, 30 were Priority 1, 53 were Priority 2, and 27 were Priority 3. This data was programmatically entered into a database, making changes as needed for mission realism that included aircraft speeds, altitudes, and time on station. Each collection is assumed to be two minutes in length, though this is an artifact of the test data and not a limitation of the scheduler.

The collection management software supports scheduling any combination of the available missions and collections, where the user selects the missions and collections to pair and presses the schedule button. Once scheduling is complete, each collection request is paired with the mission at a particular time. Users can either unpair collections to reschedule or select new collections that will be scheduled around the previously paired items. While this UI provides open ended option for planning and replanning, this paper focuses on four specific test scenarios. Scenario 1 tested static planning by scheduling 80 collections across four aircraft. Scenarios 2–4 tested dynamic replanning: adding 30 collections to an existing schedule, rescheduling all 110 collections across four aircraft, and rescheduling all 110 collections across three aircraft after the loss of one resource.

Schedule Algorithm Updates

The scheduler developed for ISR collection request scheduling builds on the algorithms supplied by the underlying framework. However, the collection domain requires additional logic to change the default behavior in some cases and to add enforcement of constraints that do not exist in the

default scheduling system (e.g. limited feasible windows per task/resource combination). This subsection describes the changes to the default scheduling algorithms. To reconcile scheduling terminology vs. the collection domain, it is helpful to remember that a collection request is a *task* and an aircraft is a *resource*.

Preprocessor for Previously Scheduled Tasks: During schedule repair (i.e., adding more tasks to an existing schedule), there is a prescheduling step that runs as each previously scheduled task is being added into the scheduling model. The scheduler tries to allocate each of these tasks to the previously scheduled time and resource combination before any new collections are scheduled. This step is intended to give the prescheduled tasks the highest chance of scheduling to their designated time and resource combination. If the prescheduled time and resource combination is not valid (e.g., the aircraft previously allocated to is no longer available), the task is added to the normal scheduling queue, where it will be allocated a time and resource as part of normal processing. An exception message is generated to notify the user of this change.

Prioritizer: Note that because of the prescheduling enforcement during preprocessing, any tasks with prescheduled information are effectively allocated at the highest priority – but that is done via a different mechanism. For things that will be scheduled dynamically, the scheduler first looks at the priority supplied in the data (e.g. priority 1, 2, 3). This allows enforcement of the high-level request tiers. The scheduler then analyzes feasible start times. Feasible start times are the number of different times a task could feasibly start, given the feasibility map. Tasks with fewer feasible start times are prioritized first, to give more restricted tasks a higher likelihood of succeeding. If both prioritizers are tied, the scheduler uses the default identifier-based prioritization to ensure a consistent scheduling order from one run to another. These deterministic results make analysis easier and more robust. All data used in the prioritization process is visible in the UI for debugging purposes.

Scheduling Method: If a task is not prescheduled, or if the prescheduling fails, the task is scheduled in an order determined by the prioritization logic discussed above. The scheduling logic uses a modified resource search that restricts the search to the feasible windows for each task/resource combination. It is also different from the default logic in that it considers all feasible windows when running the comparison using the Resource Selection Criteria; the default logic only evaluates the first valid option for each resource.

The scheduler can operate in two modes – standard and bottleneck. Standard scheduling assigns tasks to the earliest feasible time slot. In priority order, tasks are assigned to the first available resource & time combination that meets all constraints (e.g., feasibility). This represents baseline scheduler performance. In bottleneck scheduling, tasks are assigned (in priority order) to an available resource and time combination that meets all constraints and that has the least amount of combined demand. Bottleneck scheduling tries to push tasks out of high-demand slots if possible, keeping them open for later tasks.

Resource Selection Criteria: When selecting the best time window/resource combination for a task, the scheduler first considers the resource demand captured in the analytical histogram constructed by the bottleneck analysis. This allows it to preferentially schedule away from time windows/resources that are projected to be congested. In case of ties, it then prefers the earliest available window. If there are multiple windows on different resources with the same earliest availability, the final tiebreaker is the identifier associated with the task and resource. This ensures deterministic results.

Explanations: For any tasks with prescheduled information, the scheduler adds a note providing transparency into the prescheduling results. This is especially important if the prescheduled time/resource was invalid. Explanations are also given for anything that does not schedule and why, and there are a limited number of possible explanations. These include that a collection was never feasible (and why), that there is a conflict where there was not enough space in the schedule to accommodate a task, and a catch-all error for items that do not schedule for any other reason. A conflict indicates that there are spaces in the schedule where this collection could have been scheduled, but all those spaces are taken. An error indicates a data model or scheduling issue that needs to be fixed.

Conflict Repair: If a collection request does have feasible time window/resource combinations, but all such combinations are already occupied by other tasks at the time of scheduling, the scheduler invokes a conflict repair process. This attempts to reschedule one or more other tasks to make space for the problem task. The Conflict Repair component analyzes each possible time window/resource combination to see what task(s) are already occupying the possible allocation. It then analyzes each of those tasks to see whether they could be reallocated to a different time window/resource combination (one not available to the problem task), to make space for the problem task. If such an option is found, the other task is moved, and the problem task is allocated to the newly available location. This is a limited conflict repair search to keep the algorithm as fast as possible. Note that this strategy will never result in a lower-priority task being scheduled instead of a higher-priority task; a higher-priority task may be moved, but only if it has another valid location.

User Interface Updates: Although the ISR scheduler operates as a backend service, the framework's user interface was adapted to support visualization and debugging. As shown in Figure 1, the resource plot displays time on the x-axis and aircraft on the y-axis. Gray indicates aircraft unavailability, white indicates available on-station time, and blue indicates assigned collections. The red line shows aggregate scheduling demand, while the green line shows the feasible collection windows for the selected task across all aircraft. The linked table displays collection data and highlights the corresponding item in the plot when selected.

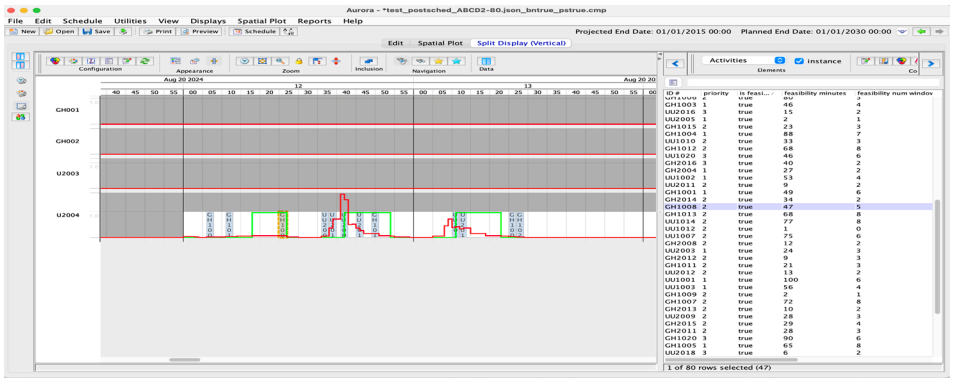


Figure 1: Assignments for aircraft U2004 with resource plot and tabular data.

RESULTS

This section presents the scheduling results generated by the scheduler on the four scenarios, as well as compares the results from the two scheduling methods. Discussion is included in line with the results.

The results from scheduling these four scenarios are shown in Table 1. This table contains the scenario and scheduling algorithm in column 1, the collections attempted to be scheduled in column 2, and the number of collections not scheduled in column 3. Column 4, Due to Feasibility, contains collections that either have no feasible collection times or do not have two minutes of contiguous feasibility. That is, these items could only be scheduled if something else changes (e.g., route, sensor) that makes the collection feasible. Column 5, Due to Conflicts, indicates collections that were not scheduled due to conflicts. This means there are spaces in the schedule where this collection could have been scheduled, but all those spaces are already full. Bolded rows indicate a difference between the STD and BN scheduling modes, where the * highlights when the BN algorithm outperformed the STD algorithm. Conflicts, when they occurred, included only Priority 2 and 3 items as shown in the three right columns.

Table 1: Initial results for STD and BN scheduling.

Scenario	Collections	Not Scheduled	Due to Feasibility	Due to Conflicts	P1 Conflicts	P2 Conflicts	P3 Conflicts
1 STD	80	21	21	0			
1 BN	80	21	21	0			
2 STD	110	26	26	0			
2 BN	110	26	26	0			
3 STD	110	28	26	2	0	1	1
3 BN	110	26	26	0*			
4 STD	110	34	28	6	0	3	3
4 BN	110	31	28	3*	0	0	3

Taken together, these results are extremely positive. The scheduler was easily able to schedule the standard collection plan (Scenario1) using either algorithm. Scenario2 and Scenario3 are two distinct versions of the same dynamic rescheduling problem. In Scenario2 the first 80 collections (a mix of priorities) are scheduled followed by adding 30 additional collections (also a mix of priorities) but keeping in place the original 80. This is schedule repair, adding in the new tasks to the existing schedule. In Scenario3, all 110 collections are scheduled at once, a complete rescheduling. The scheduler generates two different solutions based on this difference, where the STD algorithm performs worse for Scenario3 than Scenario2. This is an example where the exact nature of the data can affect the solution. Regardless, the BN algorithm solved both problems and the STD algorithm came close. Finally, the scheduler also showed strong results on Scenario4, where an aircraft dropped out and all 110 collections were dynamically rescheduled onto the remaining 3 missions. The BN algorithm managed to fit in all the feasible collections except for three Priority 3s.

Returning to the notion of conflicts, they may or may not be resolvable. For example, in an earlier version of the test data there were two collections that both had exactly one feasible time slot, and both collections had the exact same time slot. This conflict could only be resolved by changing the routes, sensors, etc. to adjust the underlying feasibilities so there were more options to place one of the collections somewhere else. Other conflicts could have been avoided by making different choices during scheduling as we saw with the STD algorithm and Scenario2 and Scenario3. As a heuristic scheduler the software aims to rapidly produce a high-quality schedule, which means there are times when it will not find the optimal schedule. As the resource allocation problem is NP-complete, this represents a pragmatic tradeoff.

However, we would like the scheduler to use a limited amount of search to fix the easy to repair conflicts. Towards this end, we implemented a Conflict Repair component that analyzes each possible time window/resource combination to see what task(s) are already occupying the possible allocation. It then analyzes each of those tasks to see whether they could be reallocated to a different time window/resource combination (one not available to the conflicted task), to make space for the conflicted task. Applying conflict repair further improved the results as shown in Table 2. This fixed the 2 conflicts left by the STD algorithm in Scenario3 and reduced the number of conflicts for both BN and STD algorithms in Scenario4.

Table 2: Updated schedule results with conflict repair.

Scenario	Collections	Not Scheduled	Due to Conflicts	Previous Conflicts
3 STD	110	26	0	2
4 STD	110	31	3	6
4 BN	110	30	2	3

In our unit tests, all these schedules are created in ~1 second on a standard desktop computer. This includes reading a stored JSON message from a file, translating the mission, collection, and feasibility data to a model, creating

a schedule, and then printing the schedule to console. We have found that communication between the various components takes longer than the scheduling itself.

COMPARISON OF SCHEDULING MODES

Standard scheduling will try to assign collections to earlier collection windows across the missions to complete the work as soon as possible. Bottleneck scheduling assigns collections to times away from the areas of greatest demand. Higher demand is when a set of collections have limited feasibilities and those limited feasibilities overlap.

Figure 2 presents a side-by-side comparison of schedules created by these two methods, with BN on top and STD on bottom. Trying to complete work as early as possible, the STD schedule is more compact and scheduled more collections to this mission than BN. By comparison, the BN schedule is more spread out. This can easily be seen in Table 3, which shows a distribution of collections across aircraft where the first available mission is assigned the bulk of the work in STD mode while BN is more evenly distributed.

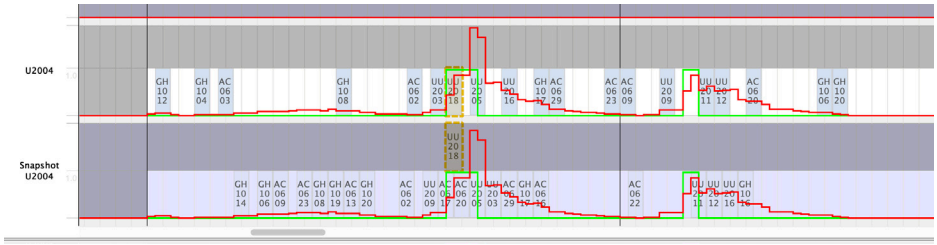


Figure 2: Comparison of BN (top) vs. Std (bottom) scheduling results.

Table 3: Distribution of collections across aircraft for Scenario1.

	STD	BN
U2003	32	17
U2004	14	15
GH001	3	10
GH002	10	17
Unassigned	21	21
	80	80

BN has a slight edge in that it will likely schedule more collections than STD, even with conflict repair, as seen in Table 2 and Table 3. Based on our experience with a large variety of scheduling problems, we have found this to be generally true across domains. The advantages and disadvantages of spreading out tasks as seen with BN vs early as possible in STD depend on the situation. Sometimes there is value in completing tasks earlier rather than later and sometimes it is better to spread out the work and leave room in

the schedule for last-minute additions or a second attempt of a failed task. A reasonable course of action is to support both modes, allowing schedulers to switch as needed.

CONCLUSION AND FUTURE WORK

Assigning ISR collections to aircraft while accounting for priorities, sensor limitations, timing, route feasibility, and current conditions is a difficult and time-consuming task. The system addresses this problem by integrating collection management, feasibility analysis, and automated scheduling into a single system.

Across four realistic scenarios, the scheduler generated high-quality minute-by-minute schedules that respected operational constraints, supported dynamic replanning when collections were added or missions removed, and provided explanations for unscheduled collections. The results also showed that bottleneck scheduling can improve performance in constrained cases by preserving scarce scheduling opportunities.

The primary benefits of automated scheduling within a collection management system are rapid generation of high-quality schedules that maximize fulfilled collections while respecting feasibility and timing constraints, transparent reporting of scheduling exceptions that helps planners iteratively improve the plan, and on-demand dynamic replanning in response to changing priorities, collection requests, missions, and conditions. Together, these capabilities can increase the number of collections completed with the same or fewer resources while reducing planning effort.

ACKNOWLEDGMENT

This material is based upon work supported by the United States Air Force Research Laboratory under Contract No. FA8650-22-C-6450. The views, opinions, and/or findings contained in this article/presentation are those of the author/presenter and should not be interpreted as representing the official views or policies, either expressed or implied, of AFRL. DISTRIBUTION A. Approved for public release: distribution unlimited.

REFERENCES

Barreiro, J., Boyce, M., Do, M., Frank, J., Iatauro, M., Kichkaylo, T., Morris, P., Ong, J. C., Remolina, E., Smith, T. B., & Smith, D. E. (2012). *EUROPA: A Platform for AI Planning, Scheduling, Constraint Programming, and Optimization*. 4th International Competition on Knowledge Engineering for Planning and Scheduling (ICKEPS).

Coman, C., & Veum, K. (2014). Automation within the ISR Collection Management domain. *IEEE International Conference on Communications*, 1–4. https://doi.org/10.1109/ICComm.2014.6866721

Ding, J., Chen, M., Wang, T., Zhou, J., Fu, X., & Li, K. (2023). A Survey of AI-enabled Dynamic Manufacturing Scheduling: From Directed Heuristics to Autonomous Learning. *ACM Comput. Surv.*, *55*(14s), 307:1-307:36. https://doi.org/10.1145/3590163

Fukunaga, A., Rabideau, G., Chien, S., & Yan, D. (1997). Towards an application framework for automated planning and scheduling. *1997 IEEE Aerospace Conference*, *1*, 375–386 vol.1. https://doi.org/10.1109/AERO.1997.574426

Gómez Sánchez, M., Lalla-Ruiz, E., Fernández Gil, A., Castro, C., & Voß, S. (2023). Resource-constrained multi-project scheduling problem: A survey. *European Journal of Operational Research*, *309*(3), 958–976. https://doi.org/10.1016/j.ejor.2022.09.033

Ludwig, J., Richards, R., Kalton, A., & Stottler, D. (2017). Applying a heuristic-based scheduling framework in manufacturing, service, and communication domains. *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 1–4. https://doi.org/10.1109/SMC.2017.8122568

Perron, L., & Didier, F. (2025). *CP-SAT* (Version v9.12) [Computer software]. Google. https://developers.google.com/optimization/cp/cp_solver/

Reijnen, R., Smit, I. G., Zhang, H., Wu, Y., Bukhsh, Z., & Zhang, Y. (2025). *Job Shop Scheduling Benchmark: Environments and Instances for Learning and Non-learning Methods* (arXiv:2308.12794). arXiv. https://doi.org/10.48550/arXiv.2308.12794

Smith, S. F., Lassila, O., & Becker, M. (1996). Configurable, Mixed-Initiative Systems for Planning and Scheduling. In A. Tate (Ed.), *Advanced Planning Technology*. AAAI Press.