2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)
Banff Center, Banff, Canada, October 5-8, 2017

Applying a Heuristic-Based Scheduling Framework in Manufacturing, Service, and Communication Domains

Jeremy Ludwig, Rob Richards, Annaka Kalton, Dick Stottler
Stottler Henke Associates, Inc.
San Mateo, CA
ludwig; richards; kalton; stottler @stottlerhenke.com

Abstract— **The work presented in this paper describes an intelligent scheduling software framework that utilizes domain-specific heuristics. The customizability of the framework and heuristics allows the software to develop a valid schedule that reflects each domain's specific preferences and constraints. Four distinct examples of this are presented in the areas of prototype vehicle testing, pharmaceutical packaging, mortgage audit scheduling, and satellite communication. In each example, the software solves a complex scheduling problem in less than five minutes and produces a schedule that is significantly better than those reached by previous methods. The primary contribution of this brief is describing how the customization of a general scheduling framework quickly generates solutions for specialized and highly constrained problems in a variety of domains.**

*A. **Keywords—scheduling framework; heuristics; prototype vehicle testing; pharmaceutical packaging; satellite communication; mortgage auditing***

II. Introduction

Scheduling, at its most basic, is the process of assigning tasks to resources over time, with the goal of optimizing the result according to one or more objectives [1]. Scheduling is heavily used in construction, manufacturing, defense, and service industries to minimize the time and cost associated with the completion or production of large and complex projects.

The Aurora scheduling framework is one example of a general-purpose scheduler that has been successfully applied to a variety of domains [2], [3]. Aurora combines graph analysis techniques with heuristic scheduling techniques to quickly produce an effective schedule based on a defined set of *tasks* and *constraints*. This typically includes the following:

- **Temporal:** Tasks must be scheduled between the project start and end dates; each task has duration and an optional start date and an optional end date.
- **Calendar:** Tasks can only be scheduled during working shifts; tasks cannot be scheduled on holidays.
- **Ordering:** Tasks can optionally be assigned to follow either immediately after/before another task or sometime after/before another task; optionally with a specific lag time in between.
- **Resource:** Each task can require that resources be available for the task to be scheduled.

The framework distills the various operations involved in creating a schedule that respects all of these constraints into reconfigurable modules that can be exchanged, substituted, adapted, and extended. This framework is used as a foundation to create domain-specific scheduling tools that respect the constraints specific to domains. Additionally, heuristics are tuned on a domain-specific basis to ensure a high-quality schedule for a given domain.

The remainder of this brief will describe related work, followed by an overview of the scheduling framework. Following this, four use cases demonstrating the application of the framework will be discussed along with a short conclusion.

A. Related Work

There is a body of prior work outside of Aurora towards creating a general-purpose scheduling framework that forms the basis of domain-specific scheduling tools. The OZONE Scheduling Framework [4] is one example. [5] describes the validation of the OZONE concept through its application to a diverse set of real-world problems, such as transportation logistics and resource-constrained project scheduling. [6] presents a design for a general scheduling framework for manufacturing. [1] presents an overview of several modern general purpose scheduling systems such as the SAP Production Planning and Detailed Scheduling System, ASPROVA Advanced Planning and Scheduling, Preactor Planning and Scheduling Systems, and ORTEM Agile Manufacturing Suite. Each of these modern systems has a distinct feature set while sharing much in common with Aurora and each other. The primary contribution of this brief is to describe one specific framework and how it is customized to develop schedules in a variety of domains.

III. Intelligent Scheduling Framework

The scheduling framework consists of two primary components: the engine and the user interface. Both components will be customized to create a domain-specific

scheduling tool, However, this paper focuses on engine customization.

A. Scheduling Engine

The scheduling engine is designed to be highly flexible and easily customizable, by allowing a software developer to specify what components to use for different parts of scheduling. The steps in the scheduling process are described in detail below and all configurable elements are shown in bold.

1) Schedule Initialization

First, Aurora applies the **Preprocessor** to the schedule information. Second, Aurora uses the **Queue Initializer** to set up the queue that will be used to run the scheduling loop. A standard Queue Initializer puts some or all the *schedulable elements*—tasks and resources—onto the queue. Third, the queue uses the **Prioritizer** to determine the priority of each element. The Prioritizer may be re-applied within the scheduling loop.

2) Scheduling loop

First, the **Scheduler** calls constraint propagation on the highest- priority schedulable to be sure that all its requirements and restrictions are up to date. Second, the Scheduler looks at the element and considers any **Scheduling Method** that is associated with it (e.g., Forward, Backward). The Scheduler also selects which **Quality Criterion** to associate with the selected scheduling method; the Quality Criterion determines what makes an assignment "good." Third, the Scheduler calls the Schedule Method on the schedulable. The result is that the schedulable element is assigned to a time window and has resources selected to satisfy any resource requirements. It also returns a list of the conflicts resulting from the given assignment. Fourth, the Scheduler calls constraint propagation on the schedulable (again) to update all the neighbors so that they are appropriately restricted by the newly scheduled element. This process may result in additional conflicts; if so, these are added to the list of conflicts from scheduling., Fifth, the Scheduler asks the **Conflict Manager** to resolve those conflicts.

3) Schedule Finalization

When the queue is empty, Aurora goes through a final conflict management step, this time at the global level. Aurora calls the **Postprocessor** on the schedule, so that any additional analysis may be done before Aurora returns the schedule results.

IV. EXAMPLE DOMAINS

Two different types of modifications are made to the intelligent scheduling framework to create a domain-specific tool. First, the components in the scheduling engine are updated specifically for the domain. Second, the user interface is modified to import, display, and edit domain-specific properties and functionality. This customization results in a tool that includes heuristics that are currently used by planners to solve the problem with existing tools and technology and that respects the constraints particular to each domain. Four domain-specific tools created with the framework and their results are briefly discussed.

A. Protype Vehicle Testing

Prototype vehicle testing is an essential part of building models of cars and trucks in the automotive industry. This can involve carrying out hundreds of tests on expensive hand-built prototype vehicles. As part of creating a schedule, the primary objectives in this domain are to minimize the number of prototype vehicles required and to complete the project in the allotted time window. This domain also includes additional constraints: **Vehicle Build Dates:** Vehicles are resources that are not available for tests until the date they are created; creation dates follow a given calendar; the scheduling system should assign creation dates to vehicles such that the objective functions are minimized; **Exclusive:** Tests indicated as exclusive must be the first test on the selected vehicle; and **Destructive:** Tests indicated as destructive must be the last test on the selected vehicle.

While most of the scheduling engine components were customized [7], [8], the Prioritizer contained the bulk of the domain specific heuristics. In general, if "difficult" tasks are scheduled earlier in the process, the schedule tends to avoid subsequent conflicts that would be difficult to repair. Several heuristics were developed to identify these difficult tasks—those tasks that are exclusive, are significantly longer in duration, are destructive, have significant follow-on work, and have fewer options with respect to resources and/or time windows.

In the end, the customized system created a testing schedule that met all of the constraints, making use of over 100 vehicles and over 30 vehicle configurations to complete over 4000 days of testing [9]. A conservative estimate suggests the schedule includes a 6% reduction in the number of vehicles over the previous scheduling method, resulting in cost savings in the millions of dollars.

B. Pharmaceutical Packaging

Pharmaceutical packaging is a critical step in the production and distribution of pharmaceutical drugs, where it is imperative that all packages are free from contamination, properly labeled according to where they will be distributed, and, most important, contain only the desired product in the correct dosage. A schedule for a packaging plant assigns products to packaging lines and determines the order in which they are packaged.

There are several important factors in producing a good schedule: **Changeover Times:** How long it takes to set up for the next packaging operation depends on the previous packaging operation and the selected line; **Line Options:** Each product has a subset of lines it can use in packaging, **Throughput:** each product has a certain packaging rate for each line; and **Consistency:** All packaging for a specific product should occur on the same line, and packaging order should be consistent from month to month.

Changeover times, line options, and throughput all feed into the line selection criteria—effectively a customized resource selection criteria. In each case, the optimal selection is balanced against resource availability in the schedule at large. Even with heuristic scheduling, analyzing the resource selection tradeoffs is a computationally expensive process. However, the consistency requirement actually reduces the scalability issues.

Because the factory needs to generally do things in the same order from one month to the next, the Preprocessor in this domain rigorously explores the problem space using utilization projections and resource selection criteria customized to use changeover times and throughput to find a good product ordering and equipment selection allocation for a single month. Having found that order, the scheduling system can reuse the order for the other months of the year, at linear computational cost. This is a slightly unusual situation because the scheduling step's decisions are effectively pre-determined by the customized preprocessor, but the preprocessor uses customized versions of the scheduler's helper classes to determine the best product allocations for the month, which will then become the template for the whole schedule.

The system provides automated scheduling across any number of packaging lines in a plant, while finding a balance that minimizes changeover time and maximizes overall equipment effectiveness. This efficiency increase allows for lower inventories and provides the plant an improved ability to accommodate change. The automated scheduling, in conjunction with the speed at which scheduling is performed, allows the human planners to better adapt to changing circumstances.

C. Mortgage Audit Scheduling

Mortgage auditing is routinely performed on lenders to guarantee that mortgage approvals are appropriate and unbiased. A large mortgage auditing company may perform thousands of audits for dozens of clients in a given week. Each audit goes through multiple synchronized steps, and all steps must be completed by a hard deadline. There are a number of constraints on how those audits should be allocated to auditors to create a schedule: **Training:** There are a wide variety of mortgage types, and audits must be assigned to personnel with the correct training; **Consistency:** Assignment of a consistent, minimal subset of auditors is advantageous; and **Thoroughness:** At least two auditors are required.

Because some of these constraints are soft (e.g., using consistent auditors for a client, or preferring a small number of auditors), while others are hard (e.g., training requirements or deadline satisfaction), a flexible scheduling strategy is required. Backtracking once tasks are formally scheduled is slow, so instead the Preprocessor has been modified to construct a less precise but more nimble projection. The Preprocessor models a queue for each auditor, with logic to determine on which day a given audit will be completed. By populating this queue in due-date order, starting with the most preferred formulation but shifting work based on a variety of heuristics, Aurora is able to quickly find a solution that maximizes the soft constraint satisfaction while satisfying the hard constraints.

The customized system allows automated scheduling of thousands of audits, a process that used to require a human scheduler to devote a person-day to each week. Because it is automated, the system can update much more frequently to support rapid adaptation to changing circumstances.

D. Satellite Communication

The Air Force commands and controls a variety of satellites through a global network of antennas and ground support equipment. Each constellation of satellites (e.g. the GPS satellites) is commanded from a separate satellite operations center. Each constellation's controlling organization makes satellite communication support requests for the antennas and other ground support equipment (including limited bandwidth for each multi-antenna site as a whole) independently of the others to a central scheduling organization which must deconflict the competing requests. The most obvious constraint on this process is that there must be **line of sight** between the antenna and the satellite. In general, the scheduling organization tries to meet the original requests as closely as possible. In a typical single day, there are about 600 or more support requests and usually more than half are in conflict with each other. Many of the conflicts are seemingly unsolvable, e.g. if there is only one antenna at a site and two requests for that antenna at the same time, the conflict is seemingly unsolvable. Yet this organization produces a conflict-free schedule daily, while meeting all requests. Meeting all (or as many as possible) support requests as closely as possible is the main objective.

The solution is a two-step process. The first step applies the bottleneck avoidance algorithm [10] to meet as many of the requests as possible with the existing resources, without relaxing any constraints. The bottleneck avoidance algorithm involves the Preprocessor to derive a global perspective by determining which resources are bottlenecks (most overly-contended-for) and at which times. This explained more fully in [11] but very briefly, this involves "spreading" each request pseudo-probabilistically across all resources that it might use. (E.g. if a support request needs one of two antennas it is pseudo allocated 50% to each one and similarly the request's needed minutes are spread across the full possible time window). The Prioritizer uses this information to put requests that need the most overly-contended-for resources at the most overly contended for times at the front of the queue to be scheduled first. The ScheduleMethod uses the bottleneck information to make resource and time window selections to avoid the worst bottlenecks by making the assignment which most reduces the bottleneck problem. That is, in making this local decision it considers the global perspective. Bottleneck avoidance solves about half of the conflicts but the remaining

ones are typically unsolvable without relaxing some aspect of the requests.

The second step of the process iteratively examines each remaining conflict, and makes suggested changes to one or more support requests. For example, a specific support may request 10 minutes of preparation time before the support will actually commence. The scheduler may know that this constellation's manager will accept 5 minutes, if there is no other choice. The suggested change to that manager is to reduce his preparation time to 5 minutes. Other changes relate to moving the support out of its requested time window or to a different site or replacing ground support equipment with alternatives or even dropping certain hardware requirements all together. Some of these changes are more suggestable if the other satellite in the conflict is from the same constellation. The scheduler annotates the schedule with symbols and notes for the suggested change, appending his initials. With dozens of constellations, and each constellation having dozens of these rules of thumb, there were hundreds of undocumented rules that the expert schedulers used to resolve effectively all the remaining conflicts. Within each set of rules, there were preferences for which to use before others. Combinations and domino effects (e.g. solving a conflict by creating another, then solving that one) had to also be considered. This knowledge was elicited and implemented in constellation-specific, user-editable rule bases which were incorporated into Aurora's Postprocessor. The application of each rule also made the necessary note annotations and appended the software's initials.

Over a thirty-year period, dozens of researchers have worked on this specific problem and the Air Force had previously invested tens of millions of dollars to develop various solutions, but all of them were considered operationally unacceptable (primarily because the relaxation rules had never been elicited before). In 2017, this application of Aurora passed high-stakes testing so that it could be operationally implemented and it demonstrated a 20-fold improvement in the time required to deconflict a 24-hour schedule.

V. Conclusion

This paper illustrates the application of a general scheduling framework to four distinct domains. While scheduling problems share much in common as described in the introduction, each of the four domains also contains additional constraints and objectives that define what constitutes a 'good' schedule in that particular domain. For each domain, we describe how the customizability of the framework and heuristics allows the software to develop a valid schedule that reflects each domain's specific objectives and constraints. Finally, the four systems highlighted in this paper are all deployed and in-use. This allows us to briefly describe the impact the scheduling system has had in each domain. In all cases, the scheduling system solves a complex scheduling problem in less than five minutes and produces a schedule that is significantly better than those arrived at by previous scheduling methods.

References

[1] M. L. Pinedo, *Scheduling*. Cham: Springer International Publishing, 2016.

[2] A. Kalton, "Applying an Intelligent Reconfigurable Scheduling System to Large-Scale Production Scheduling," presented at the International Conference on Automated Planning & Scheduling (ICAPS) 2006, Ambleside, The English Lake District, U.K., 2006.

[3] R. Richards, "Critical Chain: Short-Duration Tasks & Intelligent Scheduling in e.g., Medical, Manufacturing & Maintenance," presented at the 2010 Continuous Process Improvement (CPI) Symposium, Cal State University, Channel Islands, 2010.

[4] S. F. Smith, O. Lassila, and M. Becker, "Configurable, Mixed-Initiative Systems for Planning and Scheduling," in *Advanced Planning Technology*, A. Tate, Ed. Menlo Park, CA: AAAI Press, 1996.

[5] M. A. Becker, "Reconfigurable Architectures for Mixed-initiative Planning and Scheduling," Carnegie Mellon University, Pittsburgh, PA, USA, 1998.

[6] J. M. Framinan and R. Ruiz, "Architecture of manufacturing scheduling systems: Literature review and an integrated proposal," *Eur. J. Oper. Res.*, vol. 205, no. 2, pp. 237–246, Sep. 2010.

[7] J. Ludwig, A. Kalton, R. Richards, B. Bautsch, C. Markusic, and J. Schumacher, "A Schedule Optimization Tool for Destructive and Non-destructive Vehicle Tests," in *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, Québec City, Québec, Canada, 2014, pp. 2998–3003.

[8] J. Ludwig, B. Presnell, L. Loomis, D. Decker, and K. Light, "Deploying an Intelligent Pairing Assistant for Air Operations Centers," presented at the Maritime/Air Systems & Technologies (MAST) Europe Conference 2016, Amsterdam, Netherlands, Jun-2016.

[9] J. Ludwig, A. Kalton, R. Richards, B. Bautsch, C. Markusic, and C. Jones, "Deploying a Schedule Optimization Tool for Vehicle Testing," in *Proceedings of the 10th Scheduling and Planning Applications woRKshop (SPARK)*, 2016, pp. 44–51.

[10] D. Stottler and K. Mahan, "Automatic, Rapid Replanning of Satellite Operations for Space Situational Awareness (SSA)," presented at the Advanced Maui Optical and Space Surveillance Technologies Conference, 2015.

[11] K. Mahan, R. Stottler, and R. Jensen, "Bottleneck Avoidance Techniques for Automated Satellite Communication Scheduling," in *Infotech@Aerospace 2011*, American Institute of Aeronautics and Astronautics.