The Design of a User-Centric Scheduling System for Multifaceted Real-World Problems

Pauline M. Berry
Artificial Intelligence Center
SRI International
berry@AI.SRI.COM

Michael D. Moffitt
IBM Austin Research Laboratory
mmoffitt@us.ibm.com

Bart Peintner and **Neil Yorke-Smith**
Artificial Intelligence Center
SRI International
{peintner,nysmith}@AI.SRI.COM

Abstract

A chief hindrance to the practical value of AI scheduling and planning tools stems from the difficulty in adequately encoding domain knowledge. Using imperfect domain knowledge, fully automated systems that abstract away the 'scruffy' real world tend to produce fragile schedules that omit important constraints and optimize artificial metrics. As a result, these systems are ultimately often rejected by the user. We describe the design of a user-centric scheduling system, *Pisces*, that assists the user in exploring the rich space of schedules in complex, real-world domains with multifaceted objectives. *Pisces* retains the strength of humans in understanding schedule quality and nuances of domain constraints, while leveraging the power and flexibility of constraint-based scheduling algorithms. The system helps the user to iteratively craft a solution by expressing both high-level guidance and low-level specific constraints and preferences.

Introduction

Considerable progress has been made in the automation of scheduling processes in domains from manufacturing to military logistics [17; 2]. At the same time, sparse is the successful adoption of AI scheduling and planning systems relative to the potential.

Smith et al. [12] identify three obstacles to a more widespread and fruitful adoption: design from a technologist's rather than a user's viewpoint; poor integration between planning, scheduling, and execution; and limited system configurability. Later, surveying the challenges addressing AI scheduling, Smith [13] again identifies one obstacle as lacking in modelling and manipulation of complex constraints, objectives, and preferences — a lack of flexibility, in both modelling and operation.

Presaged by Kramer and Smith [8], Isbell and Pierce [6] characterize an *Interface-Proactivity continuum* that ranges from zero to full automation: 'Do It Yourself', 'Tells You to Pay Attention', 'Tells You What to Pay Attention To', 'Makes Suggestions', and 'Makes Decisions'. The classical development of AI scheduling systems falls in the last category of the continuum. The user states the goal and the objective criteria, and the system generates the schedule; the process is one of full automation.

Adequately encoding domain knowledge is a longstanding difficulty in the development of AI systems, especially in complex, real-world domains. Since they operate in a one-shot fashion based on inevitably incomplete information, fully automated scheduling tools tend to produce fragile schedules that omit important constraints and optimize artificial metrics [13]. Consequently, for such domains, these systems are ultimately often rejected by the user.

Our motivating domains feature large, complex, real-world problems, such as the Air Mobility Command (AMC) problem [10]. In AMC, sets of aeroplanes are grouped into air wings. Each air wing has a resource profile that indicates how many of its planes will be available over time. The planes are used to carry out missions, defined by priority levels, durations, deadlines, earliest start times, and resource requirements. The scheduling task is to assign planes to missions, in order to schedule as many higher-priority missions as possible without violating constraints.

To solve problems such as AMC efficiently, classical AI scheduling abstracts away much of the 'scruffy' real world. By contrast, humans excel at capturing and acting on the soft constraints and complex, multifaceted objective functions. Our hypothesis is that, rather than completely automated attempts, problems in real-world domains are better served by collaborative approaches that involve the user in the scheduling process. Such systems, often called *mixed-initiative*, represent a move away from one end of the Interface-Proactivity continuum.

Hence, the philosophy of our approach is to make the user central [3], viewing the process of defining and solving a scheduling problem as collaboration between a human user and one or more automated scheduling assistants. The AI system should both involve (to the extent desired by the user) and enable the user in the scheduling process. While embedding fully automated scheduling algorithms, such a system goes beyond a decision aid, working with and in service of its user to jointly solve the scheduling problem.

This paper describes the design of a user-centric scheduling system, *Pisces*, that assists the user in exploring the rich space of schedules in complex, real-world domains with multi-faceted objectives. The system is designed from a user's rather than a technologist's viewpoint, with an emphasis on configurability. *Pisces* retains the strength of humans in understanding schedule quality and nuances of domain constraints, while leveraging the power and flexibility of constraint-based scheduling algorithms. The system helps the user to iteratively craft a solution by giving the ability to

express both high-level guidance and low-level specific constraints and preferences.

After briefly describing related scheduling systems, we report our design criteria for *Pisces*, which focus on the user experience and system configurability, rather than on the underlying scheduling technology. Next, we describe the *Pisces* system, its user interface, and its scheduling engine. We present a use case highlighting the user-centric aspects of the approach, which are designed to move us closer to practical scheduling systems. We conclude with a discussion of ongoing work.

Related Work

Despite the significant progress in AI scheduling and planning technology [17], the research energy given to developing user-centric systems and tools has been in the minority. Ferguson et al. [5] present an earlier effort to create an AI system that acts as the user's assistant, collaborating in mixed-initiative user–system fashion.

Pisces shares a similar spirit with the OZONE scheduling framework [12]. OZONE presents an object-oriented system design based around a planning and scheduling ontology and a library of problem-solving components. OZONE adopts a constraint-based [2], iterative solution paradigm where, by default, the user specifies constraints and the system determines the consequences. Developing the concept of a "continuum of automation" raised by Kramer and Smith [8], *Pisces* emphasizes the user experience in exploring the solution space, with a scheduling engine in continuous background operation; fundamental to the *Pisces* design is a set of solutions that the user can explore.

A successor of OZONE is COMIREM [14], a lightweight, interactive tool for resource management in continuous planning domains. COMIREM emphasizes iterative decision making by the user at different abstraction levels, based on multiple visualizations.

MAPGEN [1] is a mixed-initiative decision-support system for Mars rover mission planning. Based around a planning and scheduling "toolbox" and an integrated, constraint-based approach to combined planning and scheduling with time and resources, MAPGEN was successfully deployed within NASA. MAPGEN shares with OZONE a paradigm of iterative solution development, based on limited changes from an initial (system-proposed) schedule, albeit the form of system–user operation is more limited.

The commonalities in the relative success of systems such as these surveyed are instances supporting more general findings. Reports on operationalized scheduling and planning tools emphasize the importance of domain modelling; flexible levels of user–system decision making; user interface; schedule visualization; and integration with existing tools, data sources, and work practices [1; 11; 4; 15].

Design Criteria

Our motivating domains are large in scale, featuring more resources and activities than a person can reason over effectively, but also featuring knowledge and objectives too subtle, life-critical, or context-dependent for a fully automated scheduling system to produce credible solutions. Such domains include the AMC problem introduced earlier, and similar large-scale, complex logistics domains.

The features of our intended domains and the obstacles to scheduler adoption articulated by Smith et al. [12] motivate the following design criteria for *Pisces*.

User-centric Besides a source of domain expertise, the user ultimately decides what is a 'good' solution; the system assists by computing the implications of the user's guidance and managing the details of the schedule.

Solution visualization and search space exploration We view scheduling not as the search for an optimal schedule but as the process of the user–system pair exploring the space of solutions. By necessity, solutions with substantive numbers of resources and activities must be displayed in such a way that the user can quickly gain a sense of the solution's strengths and weaknesses.

Incremental operation At any time, the user should have a schedule to view and refine in iterative collaboration with the system.

Multiple criteria The user must be able to state how different criteria trade off, see how a given solution rates on each criterion and in combination, and compare solutions.

Mixed-initiative The system should enable the user to manage the schedule exploration process at multiple levels of abstraction. The user should be able to state guidance in terms of high-level strategic objectives (e.g., value robustness to resource failure over resource utilization) and low-level preferences and constraints (e.g., make an activity start as late as possible).

Oversubscribed problems The system should work equally effectively when the resources are plentiful, or insufficient, to satisfy all jobs or activities.

Scalability Because real-world problems can contain hundreds or thousands of resources, the underlying scheduling engine and visualizations should scale accordingly.

Changes to domain knowledge In many real-world situations, the problem is not completely specified before scheduling begins. For instance, an added or removed resource should be quickly factored in, and previous solutions revised accordingly.

The *Pisces* System

Three main components compose *Pisces*: the scheduling engine, the user interface (UI), and the user. The UI should be understood as being the conduit through which two autonomous, cooperative agents — the engine and the user — work on a common problem continuously and in parallel, each reacting to the operations of the other.

The engine, once started, continuously searches the solution space for optimal schedules based on the current guidance from the user. Whenever a solution is found, regardless of its level of optimality, it is added to a solution database. The engine listens for changes to the user's guidance, the resources available, and the activities to be scheduled.

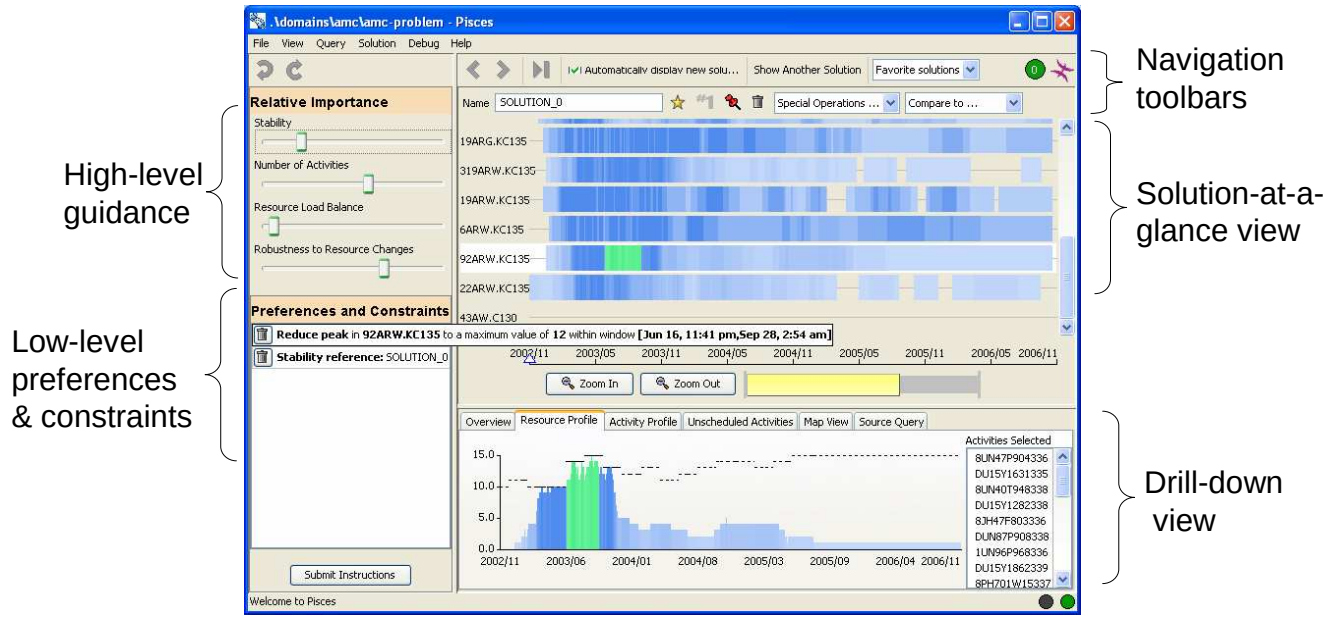

Figure 1: The *Pisces* UI gives the user the ability to specify guidance to the scheduling engine (left panes) and to view and navigate a set of solutions (right panes).

The user, once the first solution is available, directs the process of searching for and modifying solutions to the problem. For this purpose, the user can, at any time:

- Navigate through the set of solutions (e.g., to the 'best', to a favorite, to a designated reference), and direct the engine to operate from a selected solution
- Examine a solution at multiple levels of abstraction
- Annotate a solution (name it, mark it as 'best' or as poor, save it as a favorite, select it for execution)
- Change the relative importance of high-level criteria (e.g., more important to be robust to resource failure than to optimize resource usage)
- Add a low-level preference or constraint (e.g., drop usage a resource by 30% in a given time window, or force inclusion of a particular activity)
- Initiate a specialized query to the engine (e.g., flatten resource usage)

The user guides the engine by specifying from which solution to continue search (reference solution), in which direction to search (objective function), and what aspects of the solution may, may not, or must change (preferences and constraints). Combined with the ongoing iterative solution improvement process, the guidance allows the user to craft a solution in a way not possible by simply stating the weights of the objective function and invoking a fully automated scheduling algorithm.

User Interface

Figure 1 shows the *Pisces* UI toward the beginning of the collaborative solution-crafting process. It consists of three main components: a menu bar (top), panes for manipulation of user guidance (left), and panes for viewing and navigating the set of solutions (right).

Solution exploration and navigation We start with the solution navigation panes on the right, which consist of two solution navigation toolbars, a *solution-at-a-glance* (SAG) visualization, and a solution *drill-down* visualization. The solution-at-a-glance view allows the user to form a high-level assessment of a single solution, including whether there are any resource constraint violations (in red, but not shown), which resources are heavily used and at what times, and where the interesting (e.g., problematic) parts of the problem lie. For the AMC domain, each row in this pane represents the resource usage over time for a wing of aircraft. The color indicates the resource usage of the wing relative to the maximum available: grey indicates no usage, a lighter blue indicates low usage, and a darker blue indicates high usage. When usage exceeds the capacity of a resource, it is shown in red; *Pisces* attempts to eliminate such capacity constraint violations. In Figure 1, the user can tell that most resources are heavily used early in the schedule and scantly used in the later part. The user can therefore direct attention to the early part of the schedule; she might request, if possible, that additional resources be allocated to this part.

The solution drill-down view beneath the SAG view contains several panes organized in tabs. Each tab represents a detailed view of the selected part of the SAG. The Overview tab gives high-level statistics for the solution, including how well the solution scores on each objective criterion. The Resource Profile tab shows the capacity used over time for the selected resource in the SAG. In this tab, the user can add preferences to reduce the usage of a resource by a given

amount. The Activity Profile tab shows the time assigned to each activity and which activities overlap. In this tab, the user can add constraints such as "force inclusion of this activity", "lock this resource to this activity", and "lock this activity to a given time". The Unscheduled Activities tab shows which activities are excluded from the current solution. In this tab, the user can force inclusion of an activity. The Map View tab is intended to give a geo-spatial visualization of the scheduled activities. The Source Query tab contains a copy of the objective function that produced the viewed solution, and allows the user to revert back to it. Thus, the user can readily investigate the effects of varying guidance to the scheduling engine.

The solution navigation toolbars allow the user to navigate between different solutions and annotate them. The bottom bar allows the user to name the current solution, mark it as a favorite or as the 'best', mark it as the stability reference (described below), or delete it. The user's currently marked 'best' solution is the schedule that *Pisces* will pass on for execution, if the user chooses. The top bar allows the user to navigate the solution set, including browsing in the style of web browser (the leftmost 'Previous' and 'Next' buttons), and to 'Jump to best solution' (third button). A drop-down box allows access to the set of favorite solutions.

Viewing and modifying guidance Aside from specifying the solution from which the engine is to search, the user guides the solution-crafting process using two mechanisms: high-level guidance, pertaining to the composition of an objective function over a small set of criteria; and by low-level constraints and preferences. The left panes of Figure 1 show these elements.

The high-level guidance consists of a set of slider bars, specifying the weight of each criterion of the objective function. Currently, *Pisces* accommodates four criteria: *Stability*, a measure of the similarity between a candidate solution and a distinguished solution marked as the *stability reference*; *Number of Activities*; *Resource Load Balance*, a measure of how evenly distributed is the load over resources of a common type; and *Robustness to Resource Changes*, an aggregate measure of how many activities would be jeopardized for each resource that fails.

Two preferences and constraints are shown in the lower-left pane. The first says *"Reduce peak in `92ARW.KC135` to a maximum value of 12 within window [Jun 16, 11:41pm, Sep 28, 2:54pm]"*. This preference was stated by the user highlighting the resource profile of `92ARW.KC135` in the given time window (the green area in the bottom center of Figure 1, highlighted with a circle). The second preference says *"Stability reference: `SOLUTION 0`"*, which indicates that `SOLUTION 0` is the reference solution when calculating the stability metric.[1]

Although the criteria and preferences implemented to date in *Pisces* are generic, the system is architected to allow easy customization to a given domain, including specification of domain-specific guidance.

[1]Stability is computed from commonality between which activities are included in each of a pair of schedules, the resources assigned to them, and their time windows.

Scheduling Engine

The *Pisces* scheduling engine uses constraint-based, incremental reasoning [2]. Other than domain constraints, the engine supposes that all constraints, including resource capacities and user-stated guidance, are soft. Violating a preference or constraint in the user's guidance produces a smaller penalty and visual impact than violating a constraint from another source. Based on constraint violations, preference achievement, and objective function, a score is computed for each found solution to the current scheduling problem.

A ubiquitous difficulty with constraint reasoning is to find an objective function that is sufficiently expressive to capture the notion of what makes an optimal solution, but simple enough for current constraint reasoning algorithms [7]. In our motivating domains, the user's notion of 'best' leads to objective functions beyond the ability of current constraint solvers to quickly produce optimal solutions. While our weighted sum objective function approximates the user's true objective function, the *Pisces* user is enabled to craft their desired solution by the combination of the iterative collaboration with the system together with the full range of guidance (i.e., low-level preferences and constraints as well as the high-level, simplified objective function). Moreover, fortunately, in many real-world problems, absolute optimality is only important in cases where there is a single objective, such as to minimize cost or maximize throughput.

Since, for these two reasons, we do not require globally optimal solutions, the *Pisces* engine implements a local search, repair-based scheduling algorithm that uses the objective function as a heuristic [9]. The primary search step consists of inserting and removing activities in order to repair constraint violations. A tabu list helps the search escape from local optima. While the globally optimal solution is not guaranteed, the engine provides its current locally optimal solution in an anytime fashion. This ability is important for the incremental solution crafting approach. When a problem is loaded, the engine finds an initial solution by a rapid greedy construction.

Data Flow

Architecturally, the *Pisces* engine and the UI are separated. They share a common data space and react to changes in data (e.g., when the engine adds a solution to the solution set, the UI will update accordingly). This separation enables easy future extensions to both algorithms and UI.

Figure 2 shows how information flows between the user, the scheduling engine, and the UI. Because the two agents (the user and the engine) act continuously and independently, *Pisces* operates in an asynchronous manner. Starting from a designated solution, the engine continuously searches for improvements according to the current objective function; whenever an improved solution is found, it is added to the solution set. Note that the solution the user is viewing in the UI will in general differ from the solution the engine considers currently optimal; both the viewed and optimal solutions should be distinguished from the solution the user has marked as 'best'. The user can opt to have the displayed solution automatically kept synchronized with the currently optimal solution.

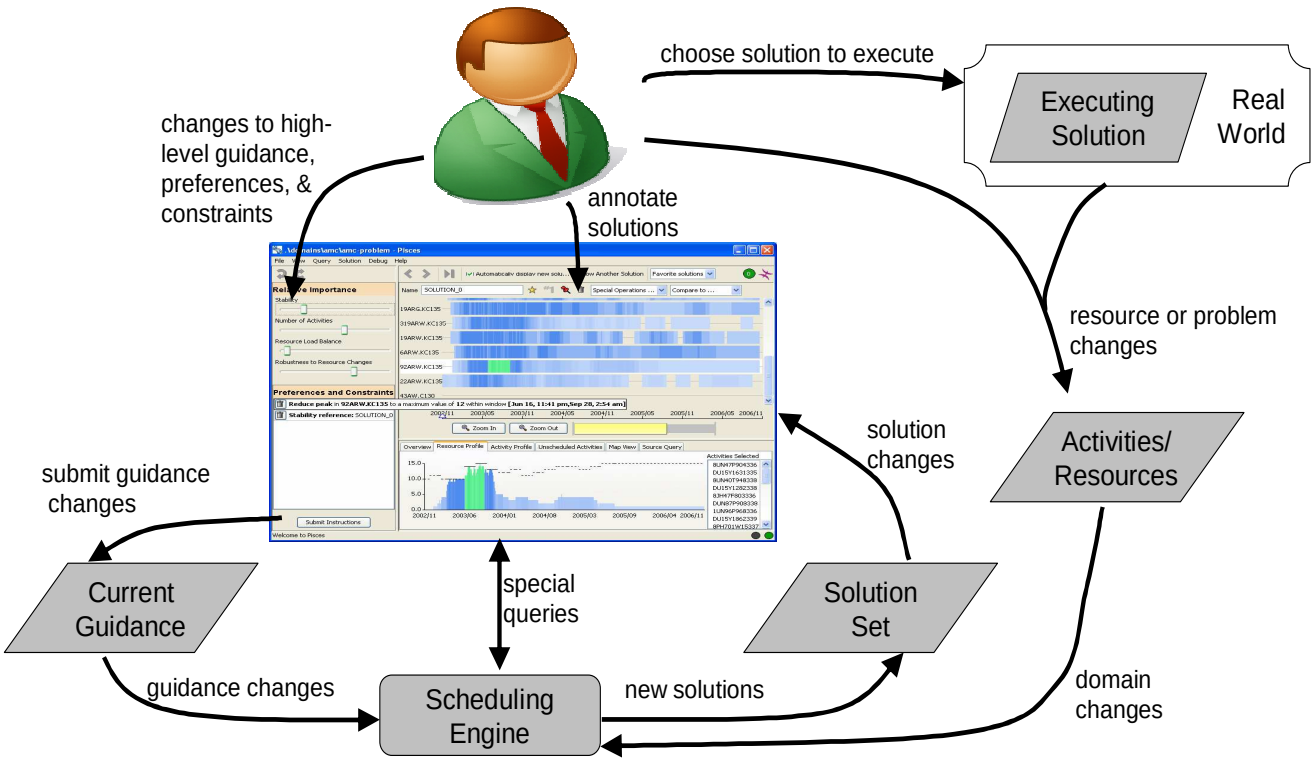

Figure 2: *Pisces* data flow. Note that during execution, resource changes and mission failures are fed into the problem representation (shown as Activities/Resources), and are immediately sent to the scheduling engine.

The engine listens for changes in the problem and changes in the objective. From the engine's point of view, these can change only when the user clicks 'Submit Instructions'. When this occurs, the engine restarts its search from the stability reference. It first re-evaluates each solution in the solution set according to the new objective function. Based on the new objective function, the solution previously considered optimal by the engine may no longer be optimal. Assuming that the user does not have a solution locked into view, the displayed solution is updated to the newly optimal solution, providing an instantaneous response to the user.

Since the database of found solutions has the potential to grow large, *Pisces* removes solutions that fare poorly on the most recent guidance, after the number exceeds a threshold.

Schedule Execution

Pisces was designed with execution in mind. Although the system has yet to be connected to a real-time schedule execution and monitoring environment, the system contains two main features that aid execution. First, *Pisces* provides the user a means to specify a 'blackout window', which specifies a range of times in which the schedule must not change.

Second, the system is responsive to any resource or mission changes that result during execution. Thus, if a set of resources suddenly becomes unavailable, *Pisces* will react to quickly generate new solutions and update those already found. The user can view the new solutions and, if she chooses, change her designation of the 'best' solution.

Illustrative Use Case

Consider a military commander, John, whose task is to schedule and allocate resources to air missions in the AMC domain. At any time, there is a large set of mission requests, each with a priority, time constraints, and resource constraints. John's subordinates monitor the mission requests and maintain the predicted availability of each resource (plane) over time.

When John loads *Pisces*, the engine immediately begins finding solutions to the problem using the default guidance and displays the first solution. John decides that he wants to maximize the number of scheduled missions, indicating this by adjusting the 'Number of Activities' slider bar. *Pisces* quickly starts displaying solutions packed with activities. Liking what he sees, John names one of the displayed solutions as `MaxMissions`, and marks it as a favorite.

Using the solution-at-a-glance view, John can see that most of his resources are at capacity in the early part of the schedule, and that — even with the engine including as many activities as possible — the later part of the schedule has only sparse resource usage. He zooms in to focus on the early part of the schedule.

Because resource usage is heavy in the early part of the schedule, John realizes that only a few failures in resources or unanticipated events during execution could wreck the likelihood of successful execution of the schedule. He therefore decides to slightly lower the importance of 'Number of

Activities' and significantly increase the importance of 'Robustness to Resource Changes'.[2] New schedules begin to appear and John marks a few as favorites. John checks the 'Unscheduled activities' tab and finds that an important mission is not included in any of these schedules. He selects 'Force inclusion of this activity'. This leads the engine to produce a solution that John marks as 'best'.

New information arrives from the field indicating changes in expected resource availability. In one case, a wing commander indicates that, while all his resources will be available, he plans to perform much needed maintenance during May. John highlights the month of May for that air wing and selects 'Reduce resource peak by 20%' (see Figure 1). He marks the current best solution as the Stability Reference and maximizes the importance of the 'Stability" criterion. The engine then finds a solution very similar to John's marked 'best', but with resource usage reduced in the key area; he marks this solution as 'best' in place of the former, and selects it for execution.

Ongoing Work

Pisces represents an exploration into ways of combining the strengths of algorithmic approaches to scheduling with the power of humans to understand schedule quality and the nuances of domain constraints. In this respect, the system occupies a range of positions on the Interface-Proactivity continuum [6]: from 'Tells You What to Pay Attention To' through 'Makes Decisions'. While *Pisces* has the ability to autonomously solve a given scheduling problem, it primarily provides the user with a mechanism to craft the solution by means of high- and low-level guidance.

The focus of *Pisces* development thus far is in the mixed-initiative user experience, and problem-solving collaboration between the user and the system. We have three main future directions. The first is expanding our scheduling engine and corresponding representation to handle domains more expressive than the AMC domain (enhanced expressiveness and configurability). This includes handling constraints between activities, uncertainty in resource usage, and uncertainty in activity duration. The second direction is developing more sophisticated visualizations, both domain dependent and domain independent. The final direction involves exploring integration with knowledge-rich planning [12; 16] and a shared problem representation with planning and execution systems.

Acknowledgments We thank Blazej Bulka and Mark Roberts for their work on *Pisces*, Laurence Kramer for providing the AMC problem data, and the anonymous reviewers for their comments.

[2] At present, these statements of high-level guidance apply to the whole schedule; the user cannot state such guidance to a subset of activities or over a designated time window.

References

[1] M. Ai-Chang, J. Bresina, L. Charest, A. Chase, J. Cheng-jung Hsu, A. Jonsson, B. Kanefsky, P. Morris, K. Rajan, J. Yglesias, B. G. Chafin, W. C. Dias, and P. F. Maldague. MAPGEN: Mixed-initiative planning and scheduling for the mars exploration rover mission. *IEEE Intelligent Systems*, 19(1):8–12, 2004.

[2] P. Baptiste, C. L. Pape, and W. Nuijten. *Constraint-Based Scheduling*. Kluwer, Boston, MA, 2001.

[3] P. Berry, B. Peintner, and N. Yorke-Smith. Bringing the user back into scheduling: Two case studies of interaction with intelligent scheduling assistants. In *Proc. of AAAI 2007 Spring Symposium on Interaction Challenges for Intelligent Assistants*, pages 10–12, Stanford University, CA, 2007.

[4] L. Castillo, J. Fdez.-Olivares, O. García-Pérez, and F. Palao. Bringing users and planning technology together:Experiences in SIADEX. In *Proc. of ICAPS'06*, Cumbria, UK, 2006.

[5] G. Ferguson, J. Allen, and B. Miller. Towards a mixed initiative planning assistant. In *Proc. of AIPS-96*, pages 70–77, Edinburgh, UK, 1996.

[6] C. L. Isbell and J. S. Pierce. An IP continuum for adaptive interface design. In *Proc. of HCI International 2005*, Las Vegas, NV, 2005.

[7] U. Junker. Preference-based search and multi-criteria optimization. In *Proc. of CP-AI-OR'02*, pages 33–47, Le Croisic, France, 2002.

[8] L. Kramer and S. Smith. Mixed-initiative resource allocation with the AMC barrel allocator. In *Proc. of 3rd Intl. NASA Workshop on Planning and Scheduling for Space*, Houston, TX, Oct. 2002.

[9] L. Kramer and S. Smith. Task swapping for schedule improvement: A broader analysis. In *Proc. of ICAPS'04*, pages 235–243, Whistler, Canada, 2004.

[10] L. Kramer and S. Smith. The AMC scheduling problem: A description for reproducibility. Technical Report CMU-RI-TR-05-75, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, 2005.

[11] M. D. Rodríguez-Moreno, D. Borrajo, and D. Meziat. An AI planning-based tool for scheduling satellite nominal operations. *AI Magazine*, 24(4):9–28, 2004.

[12] S. Smith, O. Lassila, and M. Becker. Configurable, mixed-initiative systems for planning and scheduling. In A. Tate, editor, *Advanced Planning Technology*, pages 235–241. AAAI Press, Menlo Park, CA, 1996.

[13] S. F. Smith. Is scheduling a solved problem? In *Proc. of MISTA'03*, Nottingham, UK, 2003.

[14] S. F. Smith, D. W. Hildum, and D. A. Crimm. Interactive resource management in the COMIREM planner. In *Proc. of IJCAI-03 Workshop on Mixed-Initiative Intelligent Systems*, Acapulco, Mexico, 2003.

[15] A. Tate, S. Buckingham Shum, J. Dalton, C. Mancini, and A. Selvin. Co-OPR: Design and evaluation of collaborative sensemaking and planning tools for personnel recovery. Technical Report KMI-06-07, Knowledge Media Institute, The Open University, Milton Keynes, UK, 2006.

[16] D. E. Wilkins and M. desJardins. A call for knowledge-based planning. *AI Magazine*, 22(1):99–115, 2001.

[17] M. Zweben and M. S. Fox. *Intelligent Scheduling*. Morgan Kaufmann, San Francisco, CA, 1994.