A Knowledge Based Framework for Developing and Customizing Schedulers

Venkatesh Srinivasan & Leon Sterling
Department of Computer Science & Engineering,
Center for Automation and Intelligent Systems Research,
Case Western Reserve University,
Cleveland, OH 44106.
{venky,leon}@alpha.ces.cwru.edu

Abstract
This paper describes a general purpose scheduler into which one can add knowledge necessary to solve specific scheduler problems. We believe our approach typifies how software should be developed for specific classes of applications. Using a general scheduler is analogous to the use of an expert system shell to which one adds application specific rules. We have performed experiments for adding knowledge to three classes of schedulers - job shop schedulers, transportation schedulers and time tabling schedulers. We show how the framework provides flexibility in performing experiments.

1 Scheduling - A Model Domain for Knowledge Based Software
Scheduling, the allocation of resources to tasks subject to constraints [Bak74], is a common activity. Many computer programs have been and continue to be developed to perform scheduling. When several different schedulers exist within a single organization, a question arises as to whether effort expended in developing one scheduler, for example dispatching trucks to warehouses, might be useful for another, such as assigning technicians to repair jobs. This paper concerns that question and proposes an affirmative solution in the form of a software framework.

What formulation of scheduling is suitable as the basis of a computer program? Operations Research has studied scheduling problems extensively. Only special cases have efficient solutions. The general OR formulation for finding an optimal schedule is intractable in general, as it is well-known as an NP-complete problem.

In practice, optimality may not be the most important factor. It may be sufficient to produce a reasonable schedule relatively quickly. Of course, "reasonable" and "relatively quickly" depend on the application. Knowledge based scheduling methods have been proposed, largely from the AI community, as a way of producing satisfactory schedules within reasonable times. Some well known attempts have been ISIS-2 [M.S84] and OPIS-1 [OS88]. Both of these schedulers are Job Shop schedulers, and do not perform other kinds of scheduling, for example for transportation problems.

Our research has been involved in developing a generic framework to encompass a wide range of schedulers, which facilitate the use of heuristic knowledge. We have implemented three specific schedulers within the framework - for assigning mechanics to fix cars in a repair shop (an instance of job shop scheduling), for delivering oil to a number of clients (an instance of transportation scheduling) and for producing an examination timetable (an instance of time tabling scheduling).

The next section of the paper describes our general scheduling framework to construct different schedulers, and introduces the concept of schedule clusters. Sections 3,4,5 discuss how to specify schedulers within a cluster, express heuristic knowledge and the actual process of schedule generation, respectively. For each of the three schedulers we experimented with different heuristics to compare performance, and the results appear in Section 6.

We believe that developing a generic software framework is a valuable approach for knowledge based software engineering. For scheduling, our experience suggests that new scheduling applications can be developed in significantly reduced time using the framework. Furthermore, it is easy for developers to experiment with different heuristics. Other knowledge based areas such as planning and diagnosis could be addressed with an analogous framework.
2 Design of a General Scheduling Framework

The motivation for the design of our general scheduling framework is to provide a software environment where a user can enter domain specific knowledge and constraints and have a schedule generated, while a scheduler developer can produce a scheduler. Figure 1 gives an overview of a generalized scheduling environment as we envisage. It contains three specific clusters: shop floor schedulers, transportation schedulers and time tabling schedulers. Actual schedulers developed are instances of these clusters. The tasks, resources, constraints and scheduling strategy of each scheduler are specified in the environment as are customized scheduling engines as shown in Figure 1.

Figure 2 implements the concepts presented in Figure 1 by showing the design of a front end of a windowing interface for getting scheduler specific information, and also gives the functional decomposition of schedule generation.

2.1 Schedule Cluster

The basic idea in developing generalized schedulers is to develop different schedule clusters. Each schedule cluster captures the knowledge to develop a set of related schedulers. The schedule cluster contains the basic specification of tasks, resources, constraints, heuristics, cost functions, schedule specific parameters and schedule objectives of a class of schedulers. It also describes concepts unique to a class of schedulers: for example transportation schedulers need routing information during schedule development, and time tabling schedulers have a conflict matrix indicating conflicts between different sessions. Two scheduling instances e.g. ClassRoom and Examination schedulers shown in Figure 1 belong to the time tabling cluster because the specification of tasks, resource and other components are similar between these instances.

Once a particular schedule cluster has been developed, scheduler instances are built easily by customization. For example, the Oil Tanker scheduler or Truck Dispatch scheduler are instances of a transportation scheduler and are built by extensions and customizations done to the transportation schedule cluster.

2.2 Customizable Scheduling Engine

Specific schedules are generated in two phases. The first phase takes input from the user through a windowing interface and generates intermediate object code. The inputs consist of tasks, resources, constraints, heuristics, costs, schedule parameters and the scheduling objective. The basic scheduling strategy is common to all schedulers.

Our basic scheduling strategy for each schedule iteration in generating a schedule consists of choosing a task to be scheduled which is done by the task dispatcher, choosing appropriate resources to be allocated which is done by the resource allocator, allocation of resources to tasks followed by constraint propagation which determines the updated schedulable intervals for unscheduled tasks in the system. All these tasks are moderated by knowledge. Schedulable intervals of each task are intervals in the time line between which the task can be placed on a time line; this information is maintained for all schedulers developed, during each iteration. Task and resource selections are guided by selection heuristics. The scheduling strategy is a generate and test strategy, where different schedules are generated and the best schedule is finally chosen.

The core portion of each scheduler can be functionally decomposed as shown in Figure 2. Each scheduling problem is decomposed into tasks, resources and
constraints which are available to the scheduler in an object form. The schedule pre-analyzer checks the partial feasibility of a schedule. The core of the scheduler consists of a task dispatcher, resource allocator and constraint propagator. Control of the scheduler is provided by the heuristics selector, which guides the schedule by determining the task selection and resource selection heuristics.

**Task Dispatcher:** This determines the order in which different tasks are dispatched in the system. Tasks are *first ordered* purely based on task dispatch heuristics. The task which is selected for dispatch satisfies all temporal constraints.

**Resource Allocator:** The resource allocator allocates resources to a schedulable task. Since a number of resource/task allocation combinations exist for each task, the resource allocator is guided by resource selection heuristics to determine resource selection for each task. The resource allocator also determines the start time of each task, as it depends both on the temporal constraints on the task, and the periods of availability of the resource.

**Constraint Propagator:** Once a task is allocated resources and has been placed along the time line, the constraint propagator propagates constraints in the system. The constraint propagator determines the next set of tasks which contend for dispatch, and the earliest start times are determined by the constraints in the system. Schedulable time intervals for all tasks are also modified.

**Heuristics Selector:** The heuristics selector provides flow of control information to the Task Dispatcher and Resource Allocator units. The heuristics selector can use a user-defined heuristic, or one from the set of standard heuristics defined for each scheduling cluster in the system. Task dispatch ordering is determined by the task selection heuristic, while resource allocation is guided by using resource selection heuristics. Pre-defined task selection heuristics in shop floor schedulers associate dispatch weights with all the tasks which need to be scheduled. In the transportation schedulers tasks are sorted based on one of the attributes of the task object. Resource ordering heuristics also order resources by one of their attributes and the task and resource ordering is updated at the end of each schedule iteration.

3 Specifying Schedulers and Schedule Clusters

There is a language for expressing each of the schedule clusters. The input to any scheduler is obtained through a windowing interface, and is translated into an intermediate form which is used by the scheduling engine. The specification of different entities in each cluster is briefly presented now.

3.1 Task Specification

The task specification for a shop floor cluster consists of the job identifier, task identifier, earliest start time of the task, priority of the task, the schedulable time interval of the task, and other customizable fields which can be introduced for specific instances of shop floor problems. The task specification in the transportation cluster consists of the task identifier, delivery address, zonal information (a set of destinations which are proximal to each other are in the same zone), service station addresses of stations which can service the tasks, the least quantity of material required, the maximum quantity of material required, time window for delivery of material, and other customizable fields.

3.2 Resource Specification

The resource specification for the shop floor cluster consists of the resource identifier, current load on the resource, skill level of operators, resource availability intervals, and other customizable fields. The resource specification of the transportation cluster consists of a resource identifier, size of the resource (large, medium, small), actual capacity of the resource, loading constraints, number of deliveries already assigned and other customizable fields.

3.3 Constraints Specification

The constraints expressible in any of the schedule clusters can be categorized into several constraints. Temporal constraints relate the time point of activities in the schedule. The calendar constraints relate a preference calendar with the time point of activities. Resource constraints relate a resource assignment with a task. Capacity constraints relate the amount of a resource used by a task with a resource assignment. Preference constraints define preferences for alternatives in the schedule.

3.4 Cost

Costs in the shop floor schedule cluster are incurred during the allocation of a resource to a task. Unit time cost of allocation for each resource are specified in the input to the system. Costs in the transportation schedulers include the maintenance of transportation vessels and traveling cost on each route for each category of resource. Room Assignments to sessions have specific costs in the case of time tabling schedulers.
3.5 Objective

Some of the objectives for the shop floor schedulers are to minimize flow time in the system, minimize tardiness of schedule, or to minimize the cost of resource allocation. One objective for the transportation scheduler is minimizing total cost of delivery. One of the important objectives for the time tabling schedulers is to minimize the number of session conflicts, another one could be to minimize total cost of allocation of rooms to sessions.

4 Expressing Schedule Heuristic Knowledge

The task dispatcher and resource allocator shown in Figure 2 are guided by schedule heuristics. The schedule heuristics determine the next task to be scheduled, as well as the resource to be allocated to it. Each of the clusters have domain specific heuristics, and users can also define heuristics, expressed in a simple language. Some of the common task heuristics for any of the clusters dispatch the task which is the most “hard” (most constrained) first, like in the context of time tabling schedulers a session with maximum number of conflicts, or in the context of shop floor schedulers the task to which the least number of resources can be allocated.

Some of the common resource heuristics are - allocating the most constrained resource first, i.e. for a time tabling scheduler choosing the most constrained room first, or in the case of shop floor scheduling allocating the bottleneck resources first. Other resource allocation heuristics tend to allocate the resource with maximum remaining capacity first, or a resource which can complete the current task in the earliest possible time.

Research related to ours is that of Eskey and Zweben [EZ90], and that of Fox and Bakyan [FSB89]. Fox and Bakyan describe techniques for generating goodness measures based on different search techniques and at different levels of problem constraint relaxation. Eskey and Zweben’s work also deals with use of search control rules, although it concentrates more on learning plausible search control rules.

4.1 Pre-Defined Heuristics

Certain heuristics are pre-defined in the system for both order of task selection and resource allocation. Examples of some common pre-defined task selection heuristics are Weighted Shortest Task First and Weighted Longest Task First in the case of shop floor schedulers, while Largest Delivery First and Farthest

4.2 User-Defined Heuristics

The user can also define the order in which tasks should be dispatched and resources allocated to them. The primary operations in user-defined heuristics are select and sort operations, and also conjunctions of select and sort operations. The select operations allow the user to select tasks having specific attribute values, while the sort operation allows the user to arrange the objects based on attribute values. Aggregate operators like max, min are also provided. Sort sorts the input according to a specified primary
and an optional secondary key.

We feel that by providing a mechanism for customizing the control of the scheduler, a flexible scheduler has been developed.

5 Schedule Generation

In this section we flesh out the schedule generation described in Figure 2. The Pre-Processor/Pre-Analyzer analyses the basic input data for inconsistencies, for example in the case of the car shop scheduler the pre-analyzer does a partial feasibility check by verifying if there are sufficient resource hours available compared to the hours required by input tasks. The pre-processor tailors input data in the object form to be used by the scheduler, it also constructs other useful structures from the input data - like the conflict matrix for time tabling schedulers.

The algorithm presented in Figure 3 gives the essence of the schedulers developed under the generalized scheduler framework, it assumes that the pre-analysis/pre-processing is already done on the input data. The actual implementations of the algorithm are customized for specific schedulers with minor variations as mentioned in Figure 1. "routing" is an example of a problem specific calculation performed in Step 5 in a transportation scheduler. Task and resource selection are done based on the current task/resource selection heuristics. Only those tasks which satisfy all current constraints are dispatched in Step 1, resource constraints are checked at Step 2. Step 3 updates resource capacities, resource availability and the current tasklist based on the task/resource pair selected. Propagation of constraints takes place in Step 4, and the schedulable intervals of all other tasks are modified depending on the time interval during which the current task was scheduled. The best schedule in a fixed number of iterations is reported to the user.

6 Results

The generic scheduler framework has been implemented in Quintus Prolog. The shop floor cluster along with the car shop scheduling instance is described in [VF92], while details of an instance of the transportation scheduler are presented in [SS92]. Work on the time tabling cluster is also complete. Prolog is an excellent choice for an implementation language because it provides a nice way for writing grammar rules, and also facilitates the writing of "generate and test" programs[SS86], where alternative solutions are generated by backtracking.

---

**Figure 3:** "Generate and Test" Scheduling Algorithm
We now present brief results for two scheduler instances. These illustrate how different experiments can be run on each of the cluster instances in order to generate good schedules.

6.1 Car shop Scheduler Experiments

Simulation results indicate that the scheduler is able to give good schedules as solutions for a car shop with about 8-10 workers and 30-35 jobs. The complete results appear in more detail in [VF92]. The following data presented involved scheduling 19 tasks and 8 resources. DCWS is the dispatch contention window size, and tasks with their earliest start times within the dispatch contention window contend to be dispatched during each schedule iteration. (TskHs and RcsHs are task and resource heuristics, hpf is highest priority first, min.ld is minimum load first, ef.time is earliest finishing task first, wstf is weighted shortest task first, MkSpan is the makespan, Avg FLT is the average flowtime, Max Ld is the maximum load, Avg Ld is the average load of each resource, TskHS).

Best Solutions if all operators are available from (1,16):

What is interesting: The current setting allows the scheduler end user to select different heuristics and schedule using the heuristics he/she thinks will produce the best results under the current setting. From our experience, we stress the flexibility of the environment.

Good schedules for problems depend on a host of factors. A dispatch contention window size of about 25-30% of the schedule span, choosing weighted shortest task first and allocating earliest finishing operators to tasks yielded better solutions on a large range of problems. Schedules optimized with respect to cost also turned out to be optimized with respect to the makespan in cases where hours of availability of different operators did not vary by more than 30%. The schedules which allocated operators based on earliest finish time operators minimized the makespan.

6.2 TankerScheduler Experiments

The scheduling engine was implemented in Prolog and the resource and task ordering heuristics were tried on 6 sets of sample problems.

The result of one set is presented here. The problem consisted of 9 port deliveries with 6 tankers to be used for deliveries. Some of the heuristics tried for task ordering were Highest Demand First, Farthest Tasks First, Clustered Proximal tasks with highest demand first. The first heuristics tries to schedule first those tasks whose upper limit of demand is most, the third one tries to order tasks which are proximal to one other by using meta-knowledge about inter port distances from the connectivity matrix. Two resource selection heuristics used were Highest Resource Capacity First and Most Constrained Vessel First.

What is interesting: The plots show how the cost of the schedule varies with schedule iterations. As is apparent from Plot3 and Plot4, task ordering using clustered tasks with smaller or higher demand first yields
the best schedules in less than 50 iterations while other heuristics shown in Plot1 and Plot2 do not perform as well and take up to 100-150 iterations for yielding a similar result. The current framework thus allows the user to experiment and discover the most appropriate heuristics setting.

<table>
<thead>
<tr>
<th>Tasks</th>
<th>Port</th>
<th>$A^*_u$ ($10^6$ brls)</th>
<th>$A^*_l$ ($10^6$ brls)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Port a</td>
<td>40</td>
<td>50</td>
</tr>
<tr>
<td>2</td>
<td>Port b</td>
<td>30</td>
<td>44</td>
</tr>
<tr>
<td>3</td>
<td>Port c</td>
<td>30</td>
<td>60</td>
</tr>
<tr>
<td>4</td>
<td>Port d</td>
<td>30</td>
<td>60</td>
</tr>
<tr>
<td>5</td>
<td>Port e</td>
<td>35</td>
<td>55</td>
</tr>
<tr>
<td>6</td>
<td>Port f</td>
<td>20</td>
<td>30</td>
</tr>
<tr>
<td>7</td>
<td>Port g</td>
<td>25</td>
<td>40</td>
</tr>
<tr>
<td>8</td>
<td>Port h</td>
<td>30</td>
<td>35</td>
</tr>
<tr>
<td>9</td>
<td>Port i</td>
<td>30</td>
<td>35</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Resources</th>
<th>Task</th>
<th>Category</th>
<th>MaxCapacity</th>
<th>Earliest Date Available</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>t1</td>
<td>big</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>t2</td>
<td>big</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>t3</td>
<td>big</td>
<td>60</td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td>t4</td>
<td>big</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>t5</td>
<td>small</td>
<td>40</td>
<td>0</td>
</tr>
</tbody>
</table>

Grouping by Using Meta Knowledge

| Group 1 | a, d, g |
| Group 2 | b, c, e, h |
| Group 3 | c, f, i |

Connectivity Matrix

| sl | 0 | 10 | 12 | 12 | 12 | 12 | 13 | 14 | 13 | 12 |
| a  | 10 | 0  | 17 | 18 | 4  | 17 | 6  | 16 | 15 | 16 |
| b  | 12 | 17 | 0  | 14 | 16 | 4  | 14 | 17 | 6  | 16 |
| c  | 12 | 18 | 14 | 0  | 12 | 14 | 5  | 14 | 12 | 16 |
| d  | 12 | 4  | 16 | 12 | 0  | 18 | 13 | 5  | 16 | 12 |
| e  | 12 | 17 | 4  | 18 | 14 | 0  | 13 | 16 | 5  | 13 |
| f  | 13 | 14 | 15 | 13 | 13 | 13 | 0  | 12 | 14 | 4 |
| g  | 14 | 6  | 17 | 14 | 5  | 16 | 12 | 0  | 16 | 9 |
| h  | 13 | 15 | 6  | 12 | 16 | 5  | 14 | 16 | 0  | 14 |

7 Conclusions

Our experience strongly suggests that a generic scheduler framework is viable. Indeed, we believe the framework provides an affirmative answer to the question raised in the introduction as to whether effort in developing one scheduler can be useful to another.
It has certainly been true that the first author can rapidly prototype a scheduler in a new domain. Furthermore, we believe that the framework is a useful demonstration of how to incorporate heuristic knowledge into a scheduler. It seems clear that heuristic knowledge is needed for scheduling, and knowledge engineers are helped by having a structure where knowledge can be placed. The framework provides a useful, intuitive abstraction of scheduling as: choose task, choose resource/s, allocate, and lastly propagate constraints.

Finally, we believe that our experience points the way to incorporating knowledge in other software - our understanding of the conference title of Knowledge Based Software Engineering. Build a useful general framework that abstracts the problem and provide intuitive hooks for incorporating knowledge. Similar, albeit simpler, frameworks are provided for search and game playing in [SS86]. We can envisage comparable ones for planning and diagnosis.

References


