A Generic library of problem-solving methods for scheduling applications

How to cite:
A Generic Library of Problem Solving Methods for Scheduling Applications

Dnyanesh Rajpathak, Enrico Motta, and Zdenek Zdrahal
Knowledge Media Institute, The Open University, Walton Hall, Milton Keynes, MK7 6AA, United Kingdom.
{d.g.rajpathak, e.motta, z.zdrahal}@open.ac.uk

Rajkumar Roy
Cranfield University, School of Industrial & Manufacturing Science, Cranfield, Bedford, MK43 0AL, United Kingdom.
r.roy@cranfield.ac.uk

ABSTRACT
In this paper we describe a generic library of problem-solving methods (PSMs) for scheduling applications. Although, some attempts have been made in the past at developing libraries of scheduling methods, these only provide limited coverage: in some cases they are specific to a particular scheduling domain; in other cases they simply implement a particular scheduling technique; in other cases they fail to provide the required degree of depth and precision. Our library is based on a structured approach, whereby we first develop a scheduling task ontology, and then construct a task-specific but domain independent model of scheduling problem-solving, which generalises from specific approaches to scheduling problem-solving. Different PSMs are then constructed uniformly by specialising the generic model of scheduling problem-solving. Our library has been evaluated on a number of real-life and benchmark applications to demonstrate its generic and comprehensive nature.

Categories and Subject Descriptors
I.2.8 Problem solving, control methods, and search

General Terms
Algorithms, Theory, Design

Keywords
Scheduling, Ontologies, Problem-solving methods, Knowledge acquisition, and Knowledge reuse

INTRODUCTION
Scheduling is a hard problem both in theory and in practice. As a first approximation, we can say that scheduling deals with the assignment of jobs and activities to resources within a specific time window. Theoretical approaches to scheduling strive to search for an optimal solution; however, these approaches suffer from combinatorial complexity that can be proved NP-hard [5]. The complex nature of the scheduling task has attracted attention from researchers in artificial intelligence for many years and several intelligent scheduling systems were developed in the 80s and 90s [13]. However, these systems tend to be domain specific and not easily reusable across scheduling domains.

In this paper we describe a comprehensive library of problem-solving components for scheduling, which aims both at providing practical, engineering support to build scheduling applications, as well as a principled framework to analyse and compare alternative approaches to scheduling. This work is based on the knowledge modelling paradigm [10] [17] that moves away from implementation-level analysis of knowledge-based systems (e.g., forward-chaining vs. backward-chaining behaviours of rule-based systems) to focus on the knowledge embodied by a performance system (e.g., how a diagnostic system discriminates between hypotheses on the basis of clinical tests). In parallel with this attention to knowledge-level reasoning, knowledge modelling has traditionally focused on generic knowledge components [1] [3] [10] [17], in an attempt to provide a more robust basis to the analysis and engineering of knowledge-based systems. The generic reasoning methods defined by knowledge modelling researchers are often called Problem-Solving Methods (PSMs) [1] [10]. A PSM can be seen as the abstract reasoning process underlying a KBS, and can be used to provide model-based templates to direct the knowledge acquisition (KA) process [18] and to support robust and maintainable applications by reuse [9] [10]. PSMs are usually categorised into the following two types: task-specific PSMs, which tackle specific classes of generic tasks like diagnosis, design, parametric design, etc. [1] [3] [10]; and task-independent PSMs, which do not subscribe to any particular task, but rather provide reasoning steps in terms of a generic paradigm, such as search [12].

The scheduling library proposed in this paper subscribes to the Task-Method-Domain-Application (TMDA) [10] knowledge modelling framework. This can be seen as a four-tier architecture, whereby we first formalise the scheduling task by means of the appropriate task ontology, and then we develop a generic model of scheduling problem-solving (henceforth gen-model), by instantiating
the search paradigm and in terms of the appropriate concepts in the task ontology. In total, seven knowledge-intensive PSMs were developed in our library by reusing the high-level tasks from gen-model. These are meant to cover all the validation activities carried out during scheduling problem-solving, such as completion, constraint violation, requirement violation, and optimisation.

Our library has been validated on a various real-life and benchmark applications to confirm its generic nature and its practical utility.

The content of the paper is organised as follows. The next section describes the construction of gen-model and various PSMs in our library. Then we describe an evaluation of our library in a particular application from space scheduling domain. Finally, we compare our work with existing approaches and re-iterate the main results from our research.

GENERIC TASK ONTOLOGY FOR SCHEDULING TASK

The task ontology formalises the nature of a scheduling task independent of any particular applications domains or the way problems can be solved. The task ontology and the rest of the library are specified by using Operational Conceptual Modelling Language (OCML) [10].

Formal Specification of the Scheduling Task

In our task ontology, the scheduling task is formalised in terms of the ten-dimensional space \{J, A, R, C, Req, Tr, P, Cf, Sc, Cr\}. These parameters are described as follows:

- **Jobs**, \( J = \{j_1, \ldots, j_M\} \). A set of jobs to be assigned to a set of resources for their execution.
- **Activities**, \( A \). For each job, \( j_m \), there are \( N \) uniquely associated activities. The set of all such activities is denoted as, \( A = \{a_{j_1}, \ldots, a_{j_M}\} \).
- **Resources**, \( R = \{r_1, \ldots, r_I\} \). A set of resources to which the jobs and activities can be assigned for their execution. The constraint and requirement specific knowledge relevant to the resources must be obeyed while assigning the jobs and activities over resources.
- **Constraints**, \( C = \{c_1, \ldots, c_L\} \). A set of constraints that must not be violated by a solution schedule. The typical examples of the constraints in scheduling can be a limited capacity of resources, a temporal precedence among jobs, due date of jobs, etc.
- **Requirements**, \( Req = \{req_1, \ldots, req_K\} \). A set of requirements that describe the desired properties of a solution schedule. For instance, in the manufacturing domain, to execute the milling operation a ‘milling-machine-A’ must be present along with other tools.
- **Schedule time range**, \( Tr \). The time horizon in which the schedule takes place. It is represented in terms of a start time and an end time.
- **Preferences**, \( P = \{p_1, \ldots, p_T\} \). A set of criteria for choosing among competing solution schedules. Each preference defines a partial order over the set of solution schedules. The preferences are deemed to be the choice points to choose the specific resources, although two resources have the same function.

- **Cost function**, \( Cf \). A function that computes the cost of a solution schedule.
- **Schedule**, \( Sc = \{s_1, \ldots, s_W\} \). A schedule, \( Sc \), represents all possible schedules those can be generated as an output by the task ontology. Each schedule, say, \( S_w \), is a set of quadruples of the form: \(<j_m, a_{jmN}, r_i, j_{tr_{m,n,i}}^i>\), where \( j_m \) is a job, \( a_{jmN} \) is an \( N \)th activity associated with \( j_m \), \( r_i \) is a resource, and \( j_{tr_{m,n,i}}^i \) is a job time range associated with the assignment of \( j_m \) and \( a_{jmN} \) to resource \( r_i \). The job time range is represented in terms of the earliest and latest start and end time, and each \( j_{tr_{m,n,i}}^i \) is a sub-interval of \( Tr \).
- **Solution criterion**, \( Cr \). A mapping from \( S_w \) to \{True, False\}, which determines whether a candidate schedule is a solution. The minimal set of conditions imposed by a solution criterion on a schedule \( S_w \) is, \( S_{sol} \subseteq S_w \). Usually requires \( S_{sol} \) to be consistent, complete, consistent, and feasible. See below for the definitions of these properties. More restrictive solution criteria may impose optimality condition based on the application-specific preferences and cost function.

Below we define various criteria to check the validity of a schedule.

- A schedule, say \( S_w \), is **correct**, if the pair \( j_m, a_{jmN} \) in a schedule appears no more than once.
- A schedule, say \( S_w \), is **complete**, if for each activity \( a_{jmN} \) in \( A \), associated with job \( j_m \) in a schedule, there exists a quadruple ‘q’ in \( S_{w} \), such that \( q = \langle j_m, a_{jmN}, r_i, j_{tr_{m,n,i}}^i \rangle \). In other words, all the jobs and activities in a schedule are assigned to the resources and time ranges.
- A schedule, say, \( S_w \), is **consistent**, if it does not violate any applicable constraints, \( C \cup S_w \not\models \perp \).
- A schedule, say \( S_w \), is **feasible**, if it satisfies all the applicable requirements, \( S_w \models Req \).
- A solution schedule, say \( S_{sol-opt} \), is **optimal**, if no other solution schedule has a lower cost than that of \( S_{sol-opt} \).

A more detailed discussion of the task ontology component of our library can be found in [11].

GENERIC MODEL OF PROBLEM-SOLVING

At the problem-solving level, we subscribe to search as our fundamental problem-solving mechanism. While the task ontology developed in the previous section aims at formalising the scheduling task, here, we develop the generic method ontology that provides the vocabulary necessary to characterise the search based problem-solving behaviour of the scheduling task.

During problem-solving, the set of all schedules, \( Sc \), is realised in terms of the schedule-space, and each schedule-state, \( s_w \), in a schedule-space has a unique association
with a corresponding schedule, \( s_n \). We define the relation \textit{state-transition} that enables a scheduling agent to transit from an initial schedule state to the goal state. The transition through the \textit{schedule-space} is achieved by applying the \textit{schedule-extension-resource-operator} and the \textit{schedule-extension-time-range-operator} that assign jobs to the resources and time ranges respectively. Both the operators are defined as a subclass of the class \textit{schedule-extension-operator}. The following box shows the OCML definition of the \textit{schedule-space} and the \textit{schedule-state}.

\begin{verbatim}
(def-class Schedule-Space () ?x
  ((associated-with-task :type scheduling :
    :cardinality 1)
   (has-states :type set :
    :cardinality 1))
  :constraint (=> (member ?x
                   (the ?set (has-states ?x ?set))))
   (schedule-state ?x))

(def-class Schedule-State () ?s
  ((has-schedule-model :type schedule)
   (has-states :type set :
    :cardinality 1))
  :constraint (=> (member ?s
                   (the ?set (has-states ?x ?set))))
   (schedule-state ?s))
\end{verbatim}

While constructing a schedule, it can be imagined that an assignment of one job may depend upon other jobs, and therefore, it may affect their assignments. To make such a job dependency explicit, we construct the job-dependency network by defining the following types of relations: \textit{job-depends-on}, \textit{job-affects}, and \textit{job-assignable}. The first relation says that any job \( j_1 \) may depend on any other job \( j_2 \) while constructing a schedule. The second relation is an inverse of the first relation, which states that, the assignment of job \( j_1 \) may affect any other job \( j_2 \) that depends on it. Finally, the last relation states that, if a job is an \textit{unassigned} one, then it is a potential candidate for the assignment. The function \textit{all-assignable-jobs} retrieve all the unassigned jobs in a schedule. The following box shows the OCML definitions of the relation \textit{job-depends-on}, \textit{job-affects}, and the function \textit{all-assignable-jobs}.

\begin{verbatim}
(def-relation JOB-DEPENDS-ON (?j1 ?j2)
  :constraint (and (job ?j1) (job ?j2)))

(def-relation JOB-AFFECTS (?j1 ?j2)
  :constraint (and (job ?j1) (job ?j2))
  :iff-def (job-depends-on ?j1 ?j2))

(def-function ALL-ASSIGNABLE-JOBS (?js ?sc)
  :body (setofall ?x
      (and (member ?x ?js)
           (unassigned-job ?x ?sc)
           (job-assignable ?x ?sc))))
\end{verbatim}

Our method ontology comprises about 47 definitions. Although, this method ontology is still very coarse-grained; it provides an initial basis to characterise a generic model of scheduling problem-solving.

**Generic Problem-Solving Model of Scheduling**

\textit{Gen-model} decomposes the top-level \textit{scheduling task} hierarchically into a number of (-sub) tasks and proposes (-sub) methods to achieve these tasks. These tasks and methods represent the inferences that are necessary to execute the reasoning actions for constructing a schedule. Such a breakdown is not only instrumental in identifying all the generic tasks required to characterise the \textit{scheduling task}, but also provides a generic base structure for the entire library. The problem-solving process in \textit{gen-model} is initialised by invoking the method independent control regime \textit{Gen-schedule-control}. The following box shows an informal specification of the \textit{Gen-Schedule-Control}.

\begin{verbatim}
Generic-Task Gen-Schedule-Control
Inputs: Schedule-operators, Scheduling-task
Output: Schedule-state
Control: Schedule-space
Goal: "State that satisfy goal of Scheduling-task"
Subtasks: Generate-Space, Choose-Schedule-State, Schedule-from-State
Body: Generate-Space (scheduling-task)
  \rightarrow Schedule-space
  Repeat
  \hspace{1cm}Choose-Schedule-State
  \hspace{1cm}(Schedule-space)\rightarrow Schedule-state
  \hspace{1cm}IF "Choose-schedule-state fails"
  \hspace{1cm}then Return () \rightarrow \text{fails}
  \hspace{1cm}else
  \hspace{1cm}IF "Schedule-state that satisfies a goal"
  \hspace{1cm}then Return () \rightarrow \text{Success}
  \hspace{1cm}else
  \hspace{1cm}do Schedule-from-State
  \hspace{1cm}(schedule-state)
\end{verbatim}

\textit{Gen-schedule-control} takes as an input the list of \textit{schedule-extension-operators} and the \textit{scheduling task}, and first it invokes the task \textit{generate-space} for constructing the \textit{schedule-space} associated with the scheduling task. Having created the \textit{schedule-space}, the task \textit{new-schedule-state} is invoked to create a root state associated with the \textit{schedule-space}. Within the task \textit{new-schedule-state} first we apply the \textit{downstream consistency enforcement} heuristic [16]. This heuristic propagates earliest start time of the jobs to avoid violation of the downstream cascading constraints. The complexity of this heuristic is linear, and, in the absence of resource conflict, guarantees backtrack-free search. Each newly generated state is evaluated by the task \textit{evaluate-schedule-state}. It is crucial to remember that these evaluation criteria are independent of each other. The following mechanisms describe the methods used to evaluate a schedule state.

- **Evaluate-completeness**: checks whether a schedule associated with a state is already completed;
- **Evaluate-consistency**: checks whether any of the constraints associated with a state are violated;
- **Evaluated-feasibility**: checks whether all the requirements imposed on a state are maintained;
- **Evaluate-cost**: this calculates a cost of a state by using the \textit{cost-function} from the scheduling task ontology.
- **Evaluate admissibility**: checks whether a current, consistent state lays on a solution path. For this we implemented the following \textit{look ahead heuristics: full looking ahead} and \textit{partial looking ahead} [6]. The former heuristic checks the compatibility between the value requirements (i.e., resources and time ranges) of any two unassigned jobs as well as between an unassigned job with assigned and currently selected job. The latter heuristic checks the compatibility of the value requirements between any two \textit{unassigned} jobs.

Having evaluated a schedule state, the following two tasks are invoked: \textit{choose-schedule-state} and \textit{schedule-from-}
state. The former task provides a default criterion to select a correct state by using the method that subsumes the following conditions: a) a state that does not violate any constraints; b) a state that satisfies all the requirements; and c) a state that provides a maximal extension to a schedule.

The task schedule-from-state provides a method specific control regime for gen-model, which is achieved by the generic method generate-new-state-successor. This method takes as an input the state selected by the task choose-schedule-state, and expands it iteratively by applying the schedule-extension-operators. The operator selection in gen-model is achieved based on the context and the focus knowledge. The context in gen-model is to extend a partial schedule and the focus is one of the unassigned jobs. However, it is crucial to remember that the different PSMs in our library specialise the notions of context and focus.

Once a correct context is abstracted, then all the foci (i.e., unassigned jobs) are collected in terms of the task, collect-state-foci. Selection of a correct focus (i.e., a job) is one of the most important tasks in scheduling, as it improves the efficiency of a schedule construction by reducing undue backtracking. The task propose-schedule-from-context is a high-level control regime that selects a focus by calling the task select-schedule-focus. We have developed seven alternative methods that select a correct job judiciously based on different circumstances in scheduling. All these methods are constructed by using the job selection heuristics, both from the existing scheduling literature and from the real-world domain. To elicit the heuristics from the real-world scheduling domain, we conducted KA interviews with a scheduler in a steel-manufacturing plant in the UK, and their natural language description is given below:

1) If two jobs say, $j_i$ and $j_u$, are conflicting with each other for their resource requirements then a job with the earliest due-date is selected;
2) The jobs that are consuming bottleneck resources are always given priority, as it provides better control maintaining the global stability of a schedule;
3) A job with least number of activities is selected first; as such jobs guarantee to finish early with less chances of conflict among their values.

If an application fails to provide information to select the candidate focus, then the focus is selected by subscribing to the method that is constructed based on the dynamic search rearrangement (DSR) [4]. A focus selection preference among different applications is determined by the relation schedule-focus-order. Having selected the candidate focus, the tasks collect-focus-operators and sort-focus-operators are invoked that first collects and then sorts all schedule-extension-operators applicable to the selected focus.

Finally, the selected focus is assigned to the resources and time ranges by the tasks generate-value-from-focus and propose-schedule-from-focus. Once an assignment of the current focus is completed, then the task new-schedule-state is invoked to repeat an entire cycle until all the jobs are assigned. A Gen-model consists of 135 reusable definitions that can be instantiated by the domain specific knowledge, and more importantly all the PSMs in our library are constructed simply by reusing these definitions.

**THE PROBLEM-SOLVING METHODS**

Here, we describe how the different PSMs in our library are engineered by reusing the tasks in gen-model. Because all the PSMs are constructed uniformly by specialising generate-new-state-successor, it allows us to compare and contrast their knowledge requirements. Our library comprises of the following seven PSMs: Hill-Climbing, Propose & Backtrack (P&B) [15], Propose & Revise (P&R) [9], Propose & Exchange (P&E) [14], Propose & Genetic-Exchange (P&GE), which is a variation of P&E and based on genetic algorithms, Propose & Restore-feasibility (P&Rf) that deals with the requirement violations (RV), and Propose & Improve (P&I) [10], which aims at optimising a schedule. These PSMs covers and reason about all the areas necessary to validate the scheduling task: completion, constraint violation (CV), RV, optimisation.

**Engineering of Propose & Genetic-Exchange**

In this section, we show how P&GE is engineered by reusing gen-model. Because the propose phase of P&GE is derived uniformly from gen-model; we focus our discussion on the construction of Genetical-Exchange phase.

To enable fixing the CVs within the Genetical-Exchange phase, we define the new type of operator genetical-operator. This operator takes as an input the set of flawed assignments and generates as an output the set of assignments in which the CVs are fixed. The control regime for P&GE named generation-of-P&GE is developed by refining the method specific control regime of gen-model on various dimensions. The following box shows an informal specification of generation-of-P&GE.

```
Decomposition-Method Generation-of-P&GE
Inputs: Schedule-state
Output: Successor-state
Control: Schedule-space
Subtasks: Generate-New-State-Successor Initial-Crossover Final-Crossover
Tackles: Schedule-from-State
Body: If "Schedule-state violates requirements" then Return () -> Nothing else If "Schedule-state is a solution" then Return () -> Schedule-state else do Generate-New-State-Successor (Schedule-state, Schedule-context = :Extend) If "Schedule state violates constraints" then Initial-Crossover (schedule-state) else do Final-Crossover (schedule-state)
```

Analogous to gen-model, the propose phase of the P&GE method first extends an incomplete schedule by invoking the task generate-new-state-successor in the extend context and by selecting one of the unassigned jobs as the focus.
The state selection policy of P&GE select that schedule state which gives maximal extension to a schedule, violate no constraints, and has a minimum cost. If any constraints are violated during schedule construction, then they are ignored until a complete schedule is devised. Then the task initial-crossover is invoked in the genetical-exchange context where the focus is one of the CVs. This task is achieved by the new method default-initial-crossover, which is a local improvement strategy. It takes as an input the set of complete but inconsistent assignments generated during schedule extension, and then, it perform exchanges among the assignments of the jobs that are involved in the CVs to produce a schedule with either no or at least less number of CVs. At the end of each iteration, the task initial-crossover invokes the relation schedule-violates-constraint from the task ontology to check if any improvement is achieved in a schedule. If the CVs cannot be fixed with the limited efforts, then the new task final-crossover is invoked that tries to fix the CVs globally. The new method default-crossover is defined to achieve the task final-crossover. It is an exhaustive loop that takes as an input partially corrected set of assignments by the task initial-crossover, and to improve the performance of a schedule globally by optimising the CVs the body of this method calls itself until all the CVs are fixed. The body of this method collects all the outstanding CVs as the foci. A focus selection is achieved by the new method select-candidate-constraint, which achieves the task select-schedule-focus from gen-model. The candidate focus during P&GE is selected by complying with the application-specific knowledge, but if an application fails to provide enough knowledge to achieve a focus selection, then the first CV in the list of collected foci is selected non-deterministically as the candidate focus. In compliance with the selected focus, all the genetical-operators are collected by defining the new method genetical-operator-collection, which achieves the task collect-focus-operators from gen-model. The following box shows the OCML definition of genetical-operator-collection.

The order over an application of all the selected genetical-operators is determined by instantiating the relation schedule-operator-order. The first operator from the sorted list is selected and applied to exchange the assignment of the job(s) involved in the selected focus. This process is repeated exhaustively until either a schedule with no CVs is devised, or no further removal of the CVs is possible, that is to say a schedule is optimised. Finally, the last task that is invoked in the body of the task final-crossover is the evaluate-fitness-function. This task evaluates the quality of a schedule for the tardiness of the jobs. Following formula shows the evaluation function used to check job tardiness:

\[
\sum_{j=1}^{n} \text{jtardi} = (0.1 / \text{Maximum Lateness}) \times 100
\]

Generally speaking, all the PSMs in our library are developed by specialising the method specific control regime of gen-model, the notions of operator, context, and focus. On average, less than two dozen definitions were required to be defined to engineer each new PSM. Table 1 shows a synoptic description of all the PSMs in our library. The row ‘problem-solving knowledge’ represents the type of problem-solving knowledge required by a PSM to achieve its functionality. The row ‘global properties’ represents the different validation areas of the scheduling task that can be tackled by the application of a PSM.

<table>
<thead>
<tr>
<th>Knowledge Requirement</th>
<th>Gen-Model</th>
<th>Hill-Climbing</th>
<th>P&amp;B</th>
<th>P&amp;R</th>
<th>P&amp;Rf</th>
<th>P&amp;I</th>
<th>P&amp;E</th>
<th>P&amp;GE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Context</td>
<td>Extend</td>
<td>Extend</td>
<td>Extend</td>
<td>Extend, Revise</td>
<td>Extend, Feasibility</td>
<td>Extend, Improve</td>
<td>Extend, Exchange</td>
<td>Extend, Genetical-Exchange</td>
</tr>
<tr>
<td>Focus</td>
<td>Job</td>
<td>Job</td>
<td>Job</td>
<td>Job, rv</td>
<td>Job, most expensive job</td>
<td>Job, rv</td>
<td>Job, ev</td>
<td>Job, cv</td>
</tr>
<tr>
<td>Operators</td>
<td>Schedule extension operator</td>
<td>Schedule extension operator</td>
<td>Schedule extension operator</td>
<td>Schedule extension operator, fixes</td>
<td>Schedule extension operator, feasibility-operator</td>
<td>Schedule extension operator, improvement-operator</td>
<td>Schedule extension operator, Exchange-operator</td>
<td>Schedule extension operator, Genetical-operator</td>
</tr>
<tr>
<td>Problem solving knowledge</td>
<td>Focus &amp; operator selection, available values</td>
<td>Uses a detailed cost function</td>
<td>Preference knowledge for the available values</td>
<td>Operator cost, feasibility-operator &amp; available values</td>
<td>Operator cost, feasibility-operator &amp; available values</td>
<td>Focus, operator selection, available values &amp; cost function</td>
<td>Operator cost, exchange-operator &amp; available values</td>
<td>Operator cost, genetical-operator &amp; available values</td>
</tr>
<tr>
<td>Global properties</td>
<td>Complete</td>
<td>Local optimality</td>
<td>Complete &amp; optimise operator selection</td>
<td>Complete &amp; consistent</td>
<td>Complete &amp; feasible</td>
<td>Complete &amp; optimal</td>
<td>Complete, consistent &amp; globally optimal</td>
<td>Complete, consistent &amp; globally optimal</td>
</tr>
</tbody>
</table>
EVALUATION STUDY

The main purpose of our evaluation study is to validate the generic nature of our library. It consists of three steps: 1) instantiating classes from the task ontology with application specific knowledge to formalise an application; 2) selecting and configuring a domain-independent PSM to devise a valid schedule; and 3) evaluating the performance of an application and the extent to which a selected PSM satisfied the need of an application. In total, our library has been validated on the five applications consisting of real-life and benchmark applications from the following domains: space scheduling, resource allocation, manufacturing scheduling, and joint scheduling. Because space precludes, here, we discuss only the satellite-scheduling application in detail.

Satellite-scheduling application

The satellite-scheduling application is from the domain of space scheduling. It exemplifies the complex nature of the scheduling application due to the non-monotonic nature of the constraints and varying degrees of requirements.

Construction of a Task Model

The satellite-scheduling application consists of the assignment of 5 satellites over 3 antennas to ensure communication among them at different times during a period of 24 hrs. In accordance with the task ontology the satellites are modelled as the jobs, the antennas as the limited supply resources, and the communications within each satellite as the activities. They are formalised by defining the following application-specific classes: satellite-job, antenna-resource, and satellite-communication. In extension to these concepts, the attributes of each satellite, antenna, and communication activity, such as job-time-range, availability period, and duration are modelled by defining their application-specific classes. The following box shows how Nimbus-1 satellite is formalised in terms of the OCML definitions.

```
(def-instance NIMBUS-1-JOB (satellite-job))
(def-instance NIMBUS-1-JOB (satellite-job))
(def-class SATELLITE-JOB (job))
(def-instance NIMBUS-1-TOL-LOW-RANGE-ANTENNA
  ((applicable-to-jobs '(setofall
    ?x (nimbus-1-job ?x)))
   (has-costs 6)
   (has-body (lambda (?x ?schedule)
     (the ?low-range-antenna
      (and (handles-job
        ?low-range-antenna ?x)
       (activity
        ?x ?nimbus-1-comm)
       (< (length
         ?nimbus-1-comm) 4))))))
```

Each operator has a specific cost associated with it, which represents the cost incurred by the assignment of each satellite. The cost associated with each operator is represented by the slot has-costs in the definition of operators. We define a new function satellite-state-cost-function to calculate the cost incurred by the assignment of each satellite, and the cost of a schedule is calculated by defining an application specific satellite-cost-function.

The P&B method is configured based on the focus and the operator selection knowledge. Because the satellite-scheduling application did not provide any concrete knowledge for selecting the candidate focus, we subscribed to the method job-selection-based-on-lowest-degrees-of-freedom from gen-model. This method is constructed by using the DSR heuristic, and in each iteration it selected a satellite that is left with the least number of antennas and time ranges for its assignment. The total number of antennas associated with each satellite is represented by instantiating the relation possible-resource-for-job. Having selected the focus, all the operators that can be applied to achieve the assignment of the selected focus are collected. The order over their application is determined by instantiating the relation schedule-operator-order as shown in the above box while definition ‘Nimbus-1-to-Low-Range-Antenna’ operator. The same process is iterated.

Construction of a schedule

Because the primary objective of the satellite-scheduling application was to construct a complete and a consistent schedule, we applied the P&B method from the library. We defined the following two operators: satellite-resource-operator and satellite-time-range-operator to assign antennas and time ranges to the satellites respectively. These operators are defined as a subclass of schedule-extension-resource-operator and schedule-extension-time-range-operator respectively. The satellite-resource-operator is defined by complying with application-specific knowledge which ensures that the appropriate antennas are assigned to the satellites, and it also makes sure that the ‘number of communication’ constraint imposed on each satellite is maintained as well. The following box shows the OCML definition of satellite-resource-operator defined to assign Nimbus-1 satellite to Low-Range-Antenna. The operators for the other satellites can be realised on the same lines.

```
{def-instance NIMBUS-1-TO-LOW-RANGE-ANTENNA
  satellite-resource-operator
  ((applicable-to-jobs '(setofall
    ?x (nimbus-1-job ?x)))
   (has-costs 6)
   (has-body (lambda (?x ?schedule)
     (the ?low-range-antenna
      (and (handles-job
        ?low-range-antenna ?x)
       (activity
        ?x ?nimbus-1-comm)
       (< (length
         ?nimbus-1-comm) 4))))))
```

A satellite-scheduling application is formulated based on the following constraints and requirements elicited from the application: the antenna visibility constraints states that each antenna has a limited visibility period to communicate with the assigned satellites and all the communication activities within each satellite must be completed by complying with the visibility period of antennas. The Number of communications requirement states that each satellite must have at least four communication slots per day. The Communication duration requirement states that the duration of each communication slot must be of fifteen minutes. Finally, the communication gap requirement states that the gap between any two communication slots for a given satellite should not be greater than five hours.
Until all the satellites are assigned. A complete schedule for
the application is constructed by generating 509 schedule
states. Due to the correct focus selection policy, the
schedule is constructed without any backtracking and
therefore 100% efficiency is achieved. By the completion
of a schedule, the aggregate cost of a schedule is the three-
element vector (000 120 000). Although, the schedule
generated by P&B was of a ‘good’ quality (by good quality
we mean that a schedule violated no constraints and all the
requirements are maintained), it was not an optimal one
and, therefore, we tried the hill climbing method to try and
optimise the cost of a schedule.

Analogous to the P&B method, the hill climbing method
could not achieve an optimal schedule solution. The main
reason behind this is that the assignments of the satellites
Nimbus-1 and Nimbus-2 were competing with each other,
and consequently their assignments were leading towards a
schedule state with the same cost. The hill climbing method
did not have enough competence to choose a schedule state
that could lead towards an optimal solution.

Finally, we applied the P&I method to devise an optimal
schedule. The propose phase of the method first devised a
complete schedule, and having constructed a complete
schedule, the improve phase is invoked to optimise the cost.
Within the improve phase, the satellite-improvement-
operators are defined such that they can optimise a
schedule by improving the cost of the assignment of the
most expensive satellite(s). We used the class job-cost-
function for calculating the cost associated with the
assignment of each satellite, and the most expensive
satellite is selected as the candidate focus. First all the
assigned satellites from the propose phase are collected as
the candidate foci and then they are sorted according to the
cost of their assignments. Because the assignment of
Nimbus-1 and Nimbus-2 satellites was competing with each
other, obviously these two satellites levied highest
cost, and therefore, we decided to swap the order in which
they perform their communication with the respective
antennas. The Nimbus-1 satellite is selected as a candidate
focus and the time window of Nimbus-1 is swapped with
Nimbus-2 satellite by applying the satellite-improvement-
operator. Because of the change in their execution order,
the ‘locking period performance’ between the satellites and
their respective antennas was improved by 10%. As a
result, the cost achieved by the P&I method is the three-
dimensional vector (000 108 000). Because no benchmark
application existed against which we could have checked
our cost, we tried to optimise the cost of the application
schedule as much as possible.

COMPARISON WITH RELATED WORK
Here, we compare our work with the following existing
libraries of scheduling: the domain-specific library of
scheduling [7], the constraint-based resource scheduler
(ILOG) [8], and the CommonKADS library [17].

The major difference between our approach over that of
Hori and Yoshida’s [7] is that, we subscribe to a top-down
approach of schedule construction. It starts with the generic
template (i.e., gen-model) whose components can be reused
and refined by a configuration process to construct more
specialised PSMs. As opposed to our approach, their library
follows a bottom-up approach whereby all the problem-
solvers are constructed by identifying and subscribing to
the knowledge requirements of the production scheduling
domain. Such a type of domain specificity restricts the
possible reusability of their library within a single domain
of scheduling. Another important difference between these
two approaches is that, while the PSMs in their library can
cover and reason about only completion and CV issues of
scheduling, but fail to provide any accountability for the
RV and optimality. In contrast to their approach, our library
provides a comprehensive repertoire of PSMs that tackles
all the validation areas of scheduling. Moreover, the gen-
model component of our library offers a much richer and
quicker way to construct a new PSM simply by reusing its
high-level tasks and by specialising the notions of context,
focus, state selection and operator construction knowledge.
This uniformity allows us to compare and contrast the
knowledge requirements of these PSMs. Because, the ‘gen-
model’ component is absent in their library architecture, it
does not offer a modularity for constructing a new PSM.
From a scheduling perspective their library discusses only
two job selection criteria, i.e. earliest start time and down to
the due-date, as compared to the broad job-selection criteria
propose in our library. The job selection heuristic that deals
with the assignment of the jobs and activities over the
bottleneck resources is consistent with [2].

The ILOG framework subscribes to the CS approach as
their problem-solving technique, in contrast with the
knowledge-intensive approach of our library. In spite of the
uniform approach to modelling, CS fails to provide a fine-
grained epistemological framework required to analyse
various knowledge-intensive tasks involved in the schedule
construction process. It is essentially an implementation
technique. Because their library subscribes to CS, it aims at
constructing sophisticated but domain-independent
algorithms, but such domain independence fails to support
the important function of KA. Another primary difference
between these two approaches is that, ILOG focuses on the
resource allocation class of the scheduling task as
compared to the generic class of the scheduling task tackled
by our library.

CommonKADS is a comprehensive methodology which
also tackles the assignment and scheduling tasks. However,
their library does not provide a clean separation between
the problem-solving structure and the domain description.
In other words, the problem-solvers in the CommonKADS
library are directly associated with the domain-specific
knowledge. In our viewpoint, it makes it difficult to
abstract the generic components associated with PSMs for
their reuse. More importantly, the CommonKADS library
comprises of only one method, i.e. P&R. As a result, the
CommonKADS library tackles only the completion and CV
issues of scheduling, but does not have any accountability
for validating the RV and optimisation issues. In contrast with the CommonKADS library, our library comprises a wide range of seven different PSMs that allow us to tackle all the validation areas of scheduling. Moreover, the library framework of CommonKADS is opaque, as it fails to provide the required level of detail to construct a new PSM. In contrast with CommonKADS, our library provides a wide range of methods for selecting and evaluating a schedule state by considering different scenarios, various job selection heuristics that help to improve the efficiency of a schedule construction, etc. Finally, our library offers a much richer framework to construct a new PSM simply by reusing the generic tasks developed in gen-model and by specialising the notions of context, focus, and state selection policy.

CONCLUSION
In this paper we have proposed a generic library of PSMs for the scheduling task. It is based on the TMDA knowledge modelling framework and follows a top-down approach. Because our library has drawn from the various KBS technologies, like ontologies, PSMs, search, and KA, our organisation not only allows construction of different PSMs quickly, but also provides a way to compare and contrast their knowledge requirements. Our work is important for scheduling research both from theoretical and engineering perspectives. Theoretically, it exhibits a nice integration of the various techniques that have been developed in the scheduling research and also provides an insight into the various components which can be used in scheduling. From the engineering perspective, our library offers a support for the rapid construction of scheduling applications from different domains. Because our library provides a comprehensive repertoire of seven different PSMs, it allows us to cover and reason about all the validation areas that are crucial to the scheduling task, such as completion, constraint violation, requirement violation, and optimization. Finally, our library now has hundreds of reusable definitions, and it has been validated on a number of real-life and benchmark applications to confirm its generic nature.

ACKNOWLEDGMENT
We would like to thank the anonymous referees of the K-CAP conference for providing us with valuable comments that helped to improve the quality of our paper.

REFERENCES