

# Multi-resolution modelling and Pseudo-Boolean encoding of the SNAP scheduling problem

Alejandro Bugacov<sup>†</sup>, Pedro Szekely<sup>†</sup>, Geoff Pike<sup>†</sup>, Donghan Kim<sup>†</sup>,  
Carmel Domshlak<sup>‡</sup>, Carla Gomes<sup>‡</sup> and Bart Selman<sup>‡</sup>

(<sup>†</sup>) Information Sciences Institute, University of Southern California  
4676 Admiralty Way Marina del Rey, California 90292

(<sup>‡</sup>) Department of Computer Science, Cornell University, Ithaca, NY 14853

(bugacov,szekely,donghank)[@isi.edu](mailto:isi.edu), (selman,gomes)[@cs.cornell.edu](mailto:cs.cornell.edu)

November 14, 2003

## Abstract

We describe a Pseudo-Boolean encoding of a time-discretized model of the large and complex scheduling problem coming from the SNAP system. This problem involves the precise scheduling (with 1-minute resolution) of flight missions with multiple requirements of variable duration over large (several weeks at least) planning horizons. The problem is actually a combination of scheduling and planning, where the scheduling of some tasks in the right order might enable resources to perform additional tasks, thus improving the overall quality of the schedule. In order to maintain the scalability of our approach to large planning horizons, we introduce a time discretization of the planning horizon that enable us to solve the problem at variable resolutions. The encoding is fully integrated into the SNAP system and uses the OPARIS (CIRL) Pseudo-Boolean search engine to solve the resulting Pseudo-Boolean *formula*.

# 1 Introduction

There are two major trends in methodologies to solve complex combinatorial optimization problems. One is building specialized solvers containing sophisticated heuristics derived from precise knowledge of the domain. The other method is to encode the problem into a set of variables and constraints and use general state-of-the-art search techniques to find the assignments of variables that satisfies or optimizes the system of constraints. This second approach can be competitive and sometimes outperform the specialized solvers and provides some additional long term benefits in terms of maintenance of the solver and performance. However, the burden of the overall efficiency of the solver may sometimes be on finding a proper encoding of the problem.

The last ten years have seen tremendous advances in the efficiency of SAT solvers and they can now solve systems with up to  $10^5$  variables within minutes. This have motivated some researchers to extend these advanced search techniques to other similar representations that are more appropriate for certain type of problems. Of particular interest to our problem is the Pseudo-Boolean (PB) representation. Since PB is a much more expressive and compact representation than SAT for problems involving arithmetic reasoning and there are very efficient general-purpose PB solvers available [Oparis from CIRL, WsatOIP by J.P. Walser, Backbone guided search by Weixiong Zhang], we adopted this technique to solve a time-discretized model of the SNAP scheduling problem. The encoding of this problem result in a very large number of cardinality constraints (e.g., the ones associated to exclusion of a resource to be assigned to more than a mission at a time) that can be written in PB in a form that is much more compact than its equivalent in SAT. In addition to the compactness of the encoding in the PB formalism, these solver engines also provide for optimization capabilities. This makes these solvers very attractive for our domain of real-time resource allocation where in most situations the users are interested in finding the best possible schedule that can be computed within a given execution time.

In the PB formalism, constraints are expressed as linear inequalities over a set of integers variables with domain  $\{0, 1\}$ . Constraints are weighted and can be either hard or soft. The PB engine searches for an assignment of variables that satisfies all hard constraints and minimizes the sum of the weights of violated soft constraints.

The remaining part of the report is organized as follows. Section 1 describes the time-discretized model of the SNAP scheduling problem and the variable-resolution scheme we adopted to discretize the planning horizon. Section 2 describes the Pseudo-Boolean encoding of the time-discretized problem.

## 2 Problem Definition

In order to solve the SNAP Scheduling problem using sophisticated general-purpose search engines we introduced a time-discretized model of the problem that captures the most relevant features of the domain but that is general enough that results obtained using this model problem can be useful in other domains sharing the main features of the scheduling problem.

To use finite domain encoding techniques we discretize time by dividing the length of the planning horizon,  $L$ , in a finite set of  $S$  time slots of equal duration  $w = L/S$ . In our implementation the problem can be discretized at variable time-resolution with time-step  $w \geq 1$  minute. The same time-step is used to discretize all time intervals of the tasks and availability of resources.

In the problem definition all time intervals are described as an *Availability Slot Range* which is represented as:

- Start slot: The first slot in the range
- Length: Number of slots in the range
- Capacity: Capacity during that time interval

Thus, a problem is characterized by the following entities:

- $\mathbf{T} = T_1, T_2, \dots, T_N$  : The set of  $N$  Tasks
- $\mathbf{R} = R_1, R_2, \dots, R_M$  : The set of  $M$  Resources
- $L$  : The duration of the planning horizon
- $w$  : The duration of the time-discretization step
- $s_0$  : The time (date) of the beginning of the planning horizon

The dependencies between all entities of a problem can be seen in Figure 1. In Figure 2 we sketch example of the definition of a task with 6 requirements in a problem with 10 resources and a planning horizon of 21 slots.

### 2.1 Resources

Each Resource in the set  $\mathbf{R}$  is described by:

- Name
- Resource Type (E.g., Pilot, Aircraft, Simulator)
- A set of Availability Slot Ranges. This set represent the times during which a resource is available and its capacity.

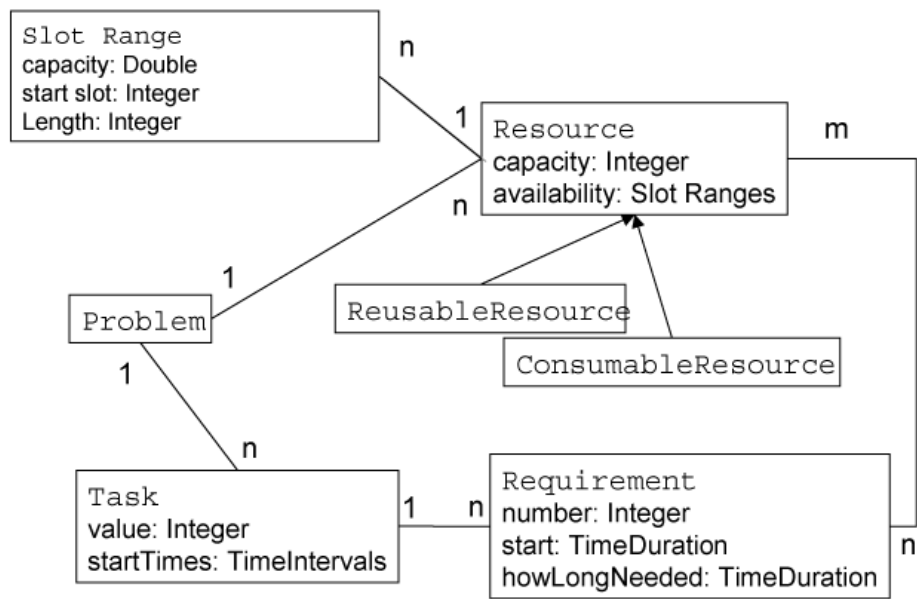


Figure 1: Dependencies between all entities of a defining the time-discretized model of the SNAP scheduling problem

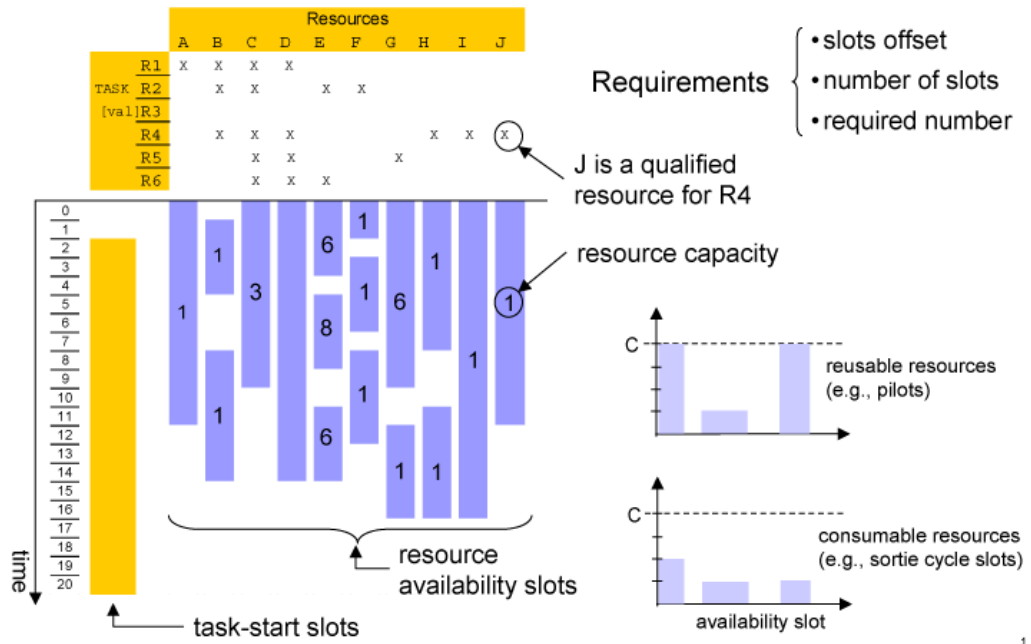


Figure 2: Example of a task with 6 requirements in a problem with 10 resources and a planning horizon of 21 slots

There are two subclasses of resources:

- Reusable Resources: These resources can be used up to their capacity at each individual time slot.
- Consumable Resources: The usage of Consumable resources adds up and they can be used up to their capacity for the duration of the whole Availability Slot Range.

## 2.2 Tasks

Each Task is described by:

- Name
- Value
- A set of Start-Time Range Slots
- A set of Requirements

## 2.3 Requirements

Each requirement is described by:

- Name
- Required Number
- Segment Length
- Offset from task start slot
- A set of Desired Resources
- A set of Qualified Resources
- A set of Required Skills
- A set of Earned Skills

### 3 Pseudo-Boolean Encoding

In the Pseudo-Boolean (PB) formalism, constraints are expressed as linear inequalities over a set of integers variables with domain  $\{0, 1\}$ . Constraints are weighted and can be either hard or soft. The PB engine searches for an assignment of variables that satisfies all hard constraints and minimizes the sum of the weights of violated soft constraints.

To encode into PB form the problem describe in the previous section, we introduce the following set of parameters and notation:

- $N$  = Number of Tasks
- $M$  = Number of Resources
- $\mathbf{R}_{reusable}$  = Set of Reusable Resources
- $\mathbf{R}_{consumable}$  = Set of Consumable Resources
- $\mathbf{R}$  = Set of all resources.  $\mathbf{R} = \mathbf{R}_{reusable} \cup \mathbf{R}_{consumable}$
- $X$  = Name or unique identifier of a resource, ( $X \in \mathbf{R}$ )
- $R_{ij}^*(k)$  = Subset of  $\mathbf{R}$  with all resources qualified for Requirement  $j$  of Task  $i$  that is available for the whole duration of the requirement segment if the task starts at slot  $k$  and has initial capacity greater or equal than the required number for that task-requirement, i.e.,  $c(X, k) \geq n_{ij}$ .
- $s_{ij}$  = Offset for Requirement  $j$  of Task  $i$
- $d_{ij}$  = Number of slots for Requirement  $j$  of Task  $i$
- $n_{ij}$  = Number of required resources for Requirement  $j$  of Task  $i$
- $V_i$  = Value of Task  $i$
- $r_i$  = Number of Requirements in Task  $i$
- $c(X, k)$  = Capacity of Resource  $X$  for its availability interval containing the slot  $k$ .
- $\mathbf{S}_{T_i}$  = Set of Starting Intervals for Task  $T_i$ .  $\mathbf{S}_{T_i} = \mathbf{S}_{T_i}^{(1)}, \mathbf{S}_{T_i}^{(2)}, \dots, \mathbf{S}_{T_i}^{(L(T_i))}$
- $\mathbf{S}_{T_i}^{(l)}$  = Set of possible starting-slots of interval  $l$  of Task  $i$

In the following subsections we describe the variables and types of constraints that we needed to introduced in order to obtain high quality valid solutions of the model problem that can then be used to generate full solutions of the SNAP scheduling problem.

### 3.1 Variables

In our encoding we introduced a hierarchy of variables where top-level variables act like *switches* or activation variables for the low-level variables. These low level variables are the ones that actually book or schedule the resources for a given task-requirement pair at a given time slot of the planning horizon. The activation variables or switches have the characteristics that when they are *turned off* (i.e., evaluated to 0) the constraints in which they appear are trivially satisfied.

For the basic set of constraints we introduce the following 4 types of variables:

Name	Description
$T_i$	<i>Task Activation Variables</i>
$T_{ik}$	<i>Task Starting-Slot Activation Variables</i>
$X_{i,j}$	<i>Resource Task Requirement Variable</i> Represents the assignment of Resource $X$ to Requirement $j$ of Task $i$ Used both for Reusable and Consumable Resources
$X_{i,j,k}$	<i>Reusable Resource Slot Variable</i> Represents the assignment of Slot $k$ of the Reusable Resource $X$ to Requirement $j$ of Task $i$

In other sections dealing with crew day constraints and resource enabling constraints we introduce additional variables but that are self-contained within those constraints.

### 3.2 Soft constraints

Currently we only introduce soft constraints to activate the top-level tasks activation variables. We introduce one soft constraint per task and weight it with the task value so that the solver will try to find the solution that maximizes the sum of the value of the scheduled tasks.

$$[soft] \quad V_i T_i \geq 1 \quad for \quad i = 1, 2, \dots, N. \quad (1)$$

In our implementation we have some options that flatten the tasks values so that the solution will correspond to the scheduling of the maximum number of tasks.

### 3.3 Task-start slot selection constraints

We introduce one of these constraints per task and they are used to select at least one of the possible task starting slots if the task activation variable is turned on and none otherwise.

$$\bar{T}_i + \sum_{k \in \mathbf{S}_{T_i}} T_{ik} = 1 \quad \text{for } i = 1, 2, \dots, N. \quad (2)$$

Here  $\mathbf{S}_{T_i}$  indicates the set of slots  $k$  that are feasible starting slots for the task considering the offsets and length of the segments for all requirements in the task  $i$ .

### 3.4 Task-requirement resource selection constraints

Once a starting slot has been selected by the constraints above, we use the following constraints to pick a resource among the set of possible resources for that starting slot  $k$ . At this point we don't distinguish between what instance of resource  $X$  we are selecting. We only worry that in its initial availability intervals the resource has enough capacity to accommodate that requirement.

$$\bar{T}_{ik} + \sum_{X \in R_{ij}^*(k)} X_{i,j} \geq 1 \quad \text{for } i = 1, 2, \dots, N \quad (3)$$

$$j = 1, 2, \dots, r_i \quad (4)$$

In this equation we should have  $k \in \Sigma(T_i)$  and  $X \in R_{ij}^*(k)$  where  $\Sigma(T_i)$  is the subset of  $\mathbf{S}_{T_i}$  that are feasible starting times for that task. I.e., starting times for which  $k \in \mathbf{S}_{T_i}$  and  $R_{ij}^*(k) \neq \emptyset \forall j$  in the task.  $R_{ij}^*(k)$  is the subset of qualified resources for that requirement with initial availability to accommodate the requirement in terms of capacity and segment duration.

In the previous expression, it is possible to select more than one resource for that requirement of a task. To exclude that possibility we introduce the following additional condition

$$\bar{T}_i + \sum_{X \in \Omega_{ij}} X_{i,j} = 1 \quad \text{for } i = 1, 2, \dots, N \quad (5)$$

$$j = 1, 2, \dots, r_i \quad (6)$$

Where  $\Omega_{ij} = \bigcup_{\forall k} R_{ij}^*(k)$

### 3.5 Reusable resources slots selection constraints

This constraints are used for *reusable resources only* to turn on  $d_{ij}$  consecutive slots (of its availability interval) starting from the  $k + s_{ij}$  slot, where  $s_{ij}$  is the offset for the requirement's segment

$$d_{ij} \bar{T}_{ik} + d_{ij} \bar{X}_{i,j} + \sum_{l=0}^{d_{ij}-1} X_{i,j,k+s_{ij}+l} \geq d_{ij} \quad (7)$$

$$i = 1, 2, \dots, N$$

$$j = 1, 2, \dots, r_i$$

(8)

In this equation we should have  $k \in \Sigma(T_i)$  and  $X \in R_{ij}^*(k)$  where  $\Sigma(T_i)$  is the subset of  $\mathbf{S}_{T_i}$  that are feasible starting times for that task. I.e., starting times for which  $k \in \mathbf{S}_{T_i}$  and  $R_{ij}^*(k) \neq \emptyset \forall j$  in the task.  $R_{ij}^*(k)$  is the subset of qualified resources for that requirement with initial availability to accommodate the requirement in terms of capacity and segment duration.

This condition alone will in principle enable some spurious slots form the resource availability to be also turned on. To avoid this effect we impose an additional condition to say that we don't want more than  $d_{ij}$  slots on for a selected resource and we want all slots turned off if that resource wasn't selected.

$$d_{ij} \bar{X}_{ij} + \sum_{k \in \mathbf{K}(i,j)} X_{i,j,k} \leq d_{ij} \quad (9)$$

$$i = 1, 2, \dots, N$$

$$j = 1, 2, \dots, r_i$$

(10)

Where  $\mathbf{K}(i, j)$  is the whole set of feasible slots for the scheduling of resource  $X$  to the requirement  $j$  of task  $i$ , taking into consideration the initial resource availability and capacity of  $X$ , the possible task start slots for task  $i$  and the segment length, offset and required capacity of requirement  $j$ .

### 3.6 Capacity Exclusion Constraints

In order to have correct solutions we need to preclude the assignment of a resource slot to more requirements than its initial capacity. Since reusable and consumable resources are modelled differently in terms of their slots assignments (i.e., we don't represent consumable resources at the atomic slot level, but only at the range availability level) we have two different expressions for reusable and consumable resources.

$$\sum_{i,j \in X(i,j,k)} n_{ij} X_{i,j,k} \leq c(X, k) \text{ for } X \in \mathbf{R}_{reusable} \quad (11)$$

$$\sum_{i,j \in X(i,j,k)} n_{ij} X_{i,j,k} \leq c(X) \text{ for } X \in \mathbf{R}_{consumable} \quad (12)$$

(13)

where  $X(i, j, k)$  is the set of task-requirement pairs in which  $X$  is a possible resource for start-slot  $k$ .

### 3.7 Crew Day Constraints

For a given problem we can specify a particular kind of crew day constraints via the following parameters:

- $c$  = crew day length (e.g., 12 hours)
- $p$  = period (e.g., 24 hours)
- $s$  = maximum shift per day (e.g., 2 hours)
- = maximum total shift (e.g., 20 hours)
- $\gamma$  = maximum number of tasks per crew day
- = number of “crew day slots” per slot (positive integer)

In addition, the problem specifies what kind of resources must adhere to crew day rules (typically pilots). The rest of this section describes encoding the crew day constraints for a single resource, a pilot. The same encoding is replicated if there are multiple pilots.

We will also use the following notation:

- $\rho$  = maximum number of crew days that could overlap the planning horizon
- $c_l$  = the start of crew day  $l$  ( $0 \leq l < \rho$ )
- $\alpha_l$  = the smallest legal value of  $c_l$
- $\beta_l$  = the largest legal value of  $c_l$
- =  $\lceil (\alpha_l + \beta_l) / 2 \rceil$
- = first slot in the planning horizon
- = last slot in the planning horizon

All times and durations in this section are measured in “crew day slots,” which are integer multiple of the slots used for resource availability. So if the problem is discretized to 10 minutes for resource availability, crew day slots would be 10 minutes.

The crew day constraints can be summarized as follows:

- $-c < -c_0 - \leq p - c$
- $(\forall l) p - s \leq c_{l+1} - c_l \leq p + s$
- $(\forall l) - \leq c_l - c_0 - lp \leq$
- No work happens outside of a crew day, and no pilot does more than  $\gamma$  tasks in a single crew day.

In order to enforce these constraints, we introduce the following pseudoboolean variables:

$d_{l,b}$	one bit of the difference $c_l -$
$A_k$	true iff pilot is at rest in slot $k$
	true iff crew day $l$ contains slot $k$
$Y_{i,k}$	true iff pilot does task $i$ in slot $k$
	true if (not iff) this pilot does task $i$ on crew day $l$

The quantity  $c_l$  is not a pseudoboolean variable, but it may be expressed as

$$-2^n d_{l,n} + d_{l,0} + 2d_{l,1} + \cdots + 2^{n-1}d_{l,n-1}$$

where the value of  $n$  is determined by how large a range must be representable (i.e., by  $\beta_l - \alpha_l$ ).

Given that, we can trivially encode

$$\begin{aligned} -c < -c_0 - \leq p - c, \\ p - s \leq c_{l+1} - c_l \leq p + s, \\ - \leq c_l - c_0 - lp \leq, \text{ and} \\ (c_l >) \rightarrow (c_l - c_{l-1} = p). \end{aligned}$$

The last of those is merely to reduce the number of redundant, equivalent solutions.

For the  $A$ 's and  $B$ 's we encode

$$\begin{aligned} A_k + + \cdots = 1 \text{ and} \\ = (c_l \leq k < c_l + c). \end{aligned}$$

(The latter expands into a few pseudoboolean constraints.) These constraints ensure that each slot is assigned to exactly one crew day or to "rest."

If crew day slot  $k$  corresponds to resource slots  $x$  through  $x + -1$  then we encode

$$\begin{aligned} Y_{i,k} \leftrightarrow (\sum X_{i,j,x} + \sum X_{i,j,x+1} + \cdots + \sum X_{i,j,x+-1} > 0) \text{ and} \\ Y_{i,k} \rightarrow \overline{A_k} \end{aligned}$$

to ensure that no work occurs when  $A_k$  is true.

Finally, to limit the number of tasks per crew day, we encode

$$+ + \cdots \leq \gamma$$

and

$$\wedge Y_{i,k} \rightarrow$$

### 3.8 Resource Constraints Enabling – Problem Statement

The flight tasks provided to SNAP have several requirements that in particular require pilots as their main resources. Unlike other resources, not every pilot can be assigned to a requirement of a given task, but only pilots that fulfill some *essential skills*, making them qualified to fulfill the mission. These essential skills are a static property of the task requirements and they are drawn from a static set of different skills  $Q = \{q_1, \dots, q_m\}$ . Let  $T_{i,j}$  represent the requirement  $j$  of task  $i$ . A set of skills that a pilot should have in order to be counted as qualified for  $T_{i,j}$  is a (possibly empty) set  $\text{Pre}(T_{i,j}) \subseteq Q$ .

In order to reason about pilots' qualifications, each pilot  $X$  (recall that pilots is modeled as a resources) is annotated with a (similarly defined) set  $\text{Quals}(X) \subseteq Q$  describing the qualification history of  $X$ . In the SNAP flight operations scheduling problem, *no pilot can be assigned to a task without having the skills required by the task*. More formally, a pilot  $X$  can be assigned to  $T_{i,j}$  if and only if  $\text{Pre}(T_{i,j}) \subseteq \text{Quals}(X)$ .

The crucial part of the flight operations scheduling problem is that *the tasks performed by the pilots provide them with additional qualifications*. Note that the generalization of this property is not unique to SNAP, but appears in many real-life scheduling problems (e.g., long-term students/courses allocation). More specifically, each task requirement  $T_{i,j}$  is associated with a (possibly empty) set of skills  $\text{Provide}(T_{i,j}) \subseteq Q$ , such that if a pilot  $X$  is assigned to perform  $T_{i,j}$ , then after performing  $T_{i,j}$  the qualification history of  $X$  is updated as follows:

$$\text{Quals}(X) := \text{Quals}(X) \cup \text{Provide}(T_{i,j})$$

In what follows, by  $\text{Quals}(X, k)$  we denote the content of  $\text{Quals}(X)$  at the time slot  $k$ .

In addition to the variables that are used for the core part of encoding, resource enabling use an additional set of variables: For each pilot  $X$ , each skill  $q \in Q$ , and each time slot  $k$ , the variable  $\llbracket X, q \rrbracket_k$  encodes the proposition that at the time slot  $k$ ,  $X$  has the skill  $q$  in his qualification history, i.e.,  $q \in \text{Quals}(X, k)$ . (In fact, a variable  $\llbracket X, q \rrbracket_k$  will be used in the encoding only if  $X$  is available at the time slot  $k$ , i.e.,  $c(X, k) = 1$ .)

The first axiom of resource enabling conditions assignment of pilots on the skills that are required by the corresponding task requirements. More formally, for each task requirement  $T_{i,j}$ , for each pilot  $X$ , and for each time slot  $k$ , we have:

$$\bigvee_{q \in \text{Pre}(T_{i,j})} \overline{\llbracket X, q \rrbracket_k} \longrightarrow \overline{X_{i,j,k}} \quad (14)$$

encoded as:

$$\alpha_{i,j} \overline{X_{i,j,k}} + \sum_{q \in \text{Pre}(T_{i,j})} \llbracket X, q \rrbracket_k \geq \alpha_{i,j} \quad (15)$$

where  $\alpha_{i,j} = |\text{Pre}(T_{i,j})|$ . Informally, Eq. ?? encodes an intuitive axiom that if  $X$  does not have the skills essential to perform  $T_{i,j}$ , then  $X$  should not be assigned to  $T_{i,j}$ .

The subsequent two axioms correspond to the planning part of the problem, and in particular they embed the planning "frame axioms". The first planning axiom is that for each pilot  $X$ , for each skill  $q$ , and for each time slot  $k$ , we have:

$$\llbracket X, q \rrbracket_k \vee \bigvee_{T_{i,j} \in \text{Provide}(q)} X_{i,j,k} \longrightarrow \llbracket X, q \rrbracket_{k+1} \quad (16)$$

where  $\text{Provide}(q)$  stands for the set of all task requirements  $T_{i,j}$  such that  $q \in \text{Provide}(T_{i,j})$ . This axiom is encoded as:

$$\beta_q \llbracket X, q \rrbracket_{k+1} + \overline{\llbracket X, q \rrbracket_k} + \sum_{T_{i,j} \in \text{Provide}(q)} \overline{X_{i,j,k}} \geq \beta_q \quad (17)$$

where  $\beta_q = |\text{Provide}(q)| + 1$ . Informally, this axiom states that "if the pilot already has a certain skill, or currently he performs something that provides this skill, then at the next time slot he will have this skill".

The second planning axiom accomplish the first planning axiom, and it states that for each resource  $X$ , for each skill  $q$ , and for each time slot  $k$ , we have:

$$\overline{\llbracket X, q \rrbracket_k} \wedge \bigwedge_{T_{i,j} \in \text{Provide}(q)} \overline{X_{i,j,k}} \longrightarrow \overline{\llbracket X, q \rrbracket_{k+1}} \quad (18)$$

encoded as:

$$\overline{\llbracket X, q \rrbracket_{k+1}} + \llbracket X, q \rrbracket_k + \sum_{T_{i,j} \in \text{Provide}(q)} X_{i,j,k} \geq 1 \quad (19)$$

Informally, this axiom states that "if the pilot does not have a certain skill, and currently he is not doing something that will provide him this skill, then at the next time slot still he will not have this skill".

Finally, for each pilot  $X$ , and for each skill  $q \in Q$ , we have  $\llbracket X, q \rrbracket_0 = 1$  if the pilot has the skill  $q$  prior to the period in scheduling, and  $\llbracket X, q \rrbracket_0 = 0$ , otherwise. This set of axioms (added to the core scheduling constraints) provides necessary and sufficient condition for correct resource enabling within the CAMERA tool and the SNAP system.