Scheduling Background: Specifications

Stephen Fitzpatrick (fitzpatrick@kestrel.edu)
Kestrel Institute, 3260 Hillview Avenue, Palo Alto, CA 94304
http://www.kestrel.edu/home/projects/ants/

This document is one of a set that attempt to bring together background information on scheduling. In writing these documents, we have not attempted to be comprehensive, but rather have concentrated on information useful for Kestrel’s ANTs project. The contents of the set of documents are:

1. Glossary
   Informal definitions of terms generic to scheduling.

2. Specifications
   Formal definitions of sort and operations for scheduling.

3. Algorithms
   Informal descriptions of classes of algorithms used in scheduling.

If you do not have any of the other documents in the set, you should be able to find them at our web site: http://www.kestrel.edu/home/projects/ants/scheduling/

Caveat: this is a work in progress.

2 The Specifications

The specifications below define sorts and operators (“op”) for scheduling. Most of the terms in the glossary have formal definitions below. The syntax is fairly standard for operator signatures and axiomatic definitions. A “def” term defines an operation in a functional style (i.e., constructively).

2.1 Sorts and Operators

2.1.1 Sort Time = Real

2.1.2 Sort Duration = Real

2.1.3 Sort Weight = Real

2.1.4 Sort Task-Type
   % not defined here – problem specific

2.1.5 Sort Task
   op release-date: Task → Time
   op due-date: Task → Time
   op weight: Task → Weight
op type: Task → Task-Type

axiom uniqueness
\[ \forall (t: \text{Task}, u: \text{Task}) \]
\[ t = u \iff \text{release-date}(t) = \text{release-date}(u) \land \text{due-date}(t) = \text{due-date}(u) \land \]
\[ \text{weight}(t) = \text{weight}(u) \land \text{type}(t) = \text{type}(u) \]

2.1.6 SORT RESOURCE-TYPE
% not defined here – problem specific

SORT RESOURCE
op type: Resource → Resource-Type
op compatible-task-types: Resource → set(Task-Type)
op processing-times: Resource → (Task-Type → Duration)
op setup-times: Resource → (Task-Type, Task-Types → Duration)

axiom uniqueness
\[ \forall (r: \text{Resource}, s: \text{Resource}) \]
\[ r = s \iff \text{type}(r) = \text{type}(s) \land \text{compatible-task-types}(r) = \text{compatible-task-types}(s) \land \]
\[ \text{processing-times}(r) = \text{processing-times}(s) \land \]
\[ \text{setup-times}(r) = \text{setup-times}(s) \]

op compatible-task?: Resource, Task → Boolean
def compatible-task?(r,t) = Task.type(t) ∈ \text{compatible-task-types}(r)

op processing-time: Resource, Task|compatible-task? → Duration
def processing-time(r,t) = \text{processing-times}(r)(\text{Task.type}(t))

op two-compatible-tasks?: Resource, Task, Task → Boolean
def two-compatible-tasks?(r,t,u) = compatible-task?(r,t) \land \text{compatible-task?}(r,u)

op setup-time: Resource, Task, Task|two-compatible-tasks? → Duration
def setup-time(r,t,u) = \text{setup-times}(r)(\text{Task.type}(t),\text{Task.type}(u))

2.1.7 SORT HARD-CONSTRAINT
% An arbitrary boolean-valued function
op global-constraint: (Schedule, set(Resource), set(Task) → Boolean)
→ Hard-Constraint
% This op lifts a single-reservation constraint to a schedule constraint
op pointwise-constraint: (Reservation → Boolean) → Hard-Constraint
% This op lifts a constraint on neighboring reservations to a schedule constraint
op consecutive-constraint: (Reservation, Reservation → Boolean)
→ Hard-Constraint
\textbf{op observes?: Hard-Constraint, Schedule, set(Resource), set(Task) \rightarrow Boolean}

def observes?(global-constraint(f),s,R,T) = f(s,R,T)
def observes?(pointwise-constraint(f),s,R,T) = reduce(all-reservations(s), \land, \lambda r \rightarrow f(r))
def observes?(consecutive-constraint(f),s,R,T) =
\forall (r \in \text{resources}(s); p,q \in \text{reservations}(s,r)) \text{ consecutive?}(s,p,q) \Rightarrow f(p,q)

\subsection*{2.1.8 Sort Penalty = \text{REAL}}

\subsection*{2.1.9 Sort Soft-Constraint}

\text{% An arbitrary function}
\textbf{op global-constraint: (Schedule \rightarrow Penalty) \rightarrow Soft-Constraint}

\text{% Apply a penalty pointwise to each reservation that violates a constraint, and sum}
\textbf{op pointwise-constraint:}
\text{(Reservation \rightarrow Boolean), (Reservation \rightarrow Penalty) \rightarrow Soft-Constraint}

\text{% Apply a penalty pointwise to each reservation that violates a constraint,}
\text{% and take the maximum}
\textbf{op max-constraint:}
\text{(Reservation \rightarrow Boolean), (Reservation \rightarrow Penalty) \rightarrow Soft-Constraint}

\text{% Apply a pair-wise penalty to each pair of consecutive reservations that violates a}
\text{% constraint, and sum}
\textbf{op consecutive-constraint: (Reservation, Reservation \rightarrow Boolean),}
\text{(Reservation, Reservation \rightarrow Penalty) \rightarrow Soft-Constraint}

\textbf{op penalty: Soft-Constraint, set(Resource), set(Task), Schedule \rightarrow Penalty}

def penalty(global-constraint(f),s,R,T)) = f(s,R,T)
def penalty(pointwise-constraint(p,f),s,R,T)
\quad = reduce(all-reservation(s), +, \lambda r \rightarrow \text{if } p(r) \text{ then } 0.0 \text{ else } f(r))
def penalty(max-constraint (p,f),s,R,T)
\quad = reduce(all-reservation(s), max, \lambda r \rightarrow \text{if } p(r) \text{ then } 0.0 \text{ else } f(r))
def penalty(consecutive-constraint (p,f),s,R,T)
\quad = reduce(\{(r,t) \mid r, t \in \text{all-reservations}(s) \land \text{consecutive?}(s,r,t)\}, +,
\quad \lambda r,t \rightarrow \text{if } p(r,t) \text{ then } 0.0 \text{ else } f(r,t))

\subsection*{2.1.10 Sort Precedence-Constraint}

\textbf{op relation: Precedence-Constraint \rightarrow set((Task, Task))}

\textit{axiom uniqueness}
\forall (c:\text{Precedence-Constraint}, d:\text{Precedence-Constraint})
\quad c = d \Leftrightarrow \text{relation}(c) = \text{relation}(d)

\text{% A precedence relation must be a strict partial order}
\text{% (i.e., a partial order with anti-reflexivity instead of reflexivity)}
\textit{axiom anti-reflexive}
\forall (c:\text{Precedence-Constraint}, x:\text{Task})
¬ (x,x) ∈ relation(c)

**axiom transitive**

∀(c: Precedence-Constraint, x:Task, y:Task, z:Task)
(x,y) ∈ relation(c) ∧ (y,z) ∈ relation(c) ⇒ (x,z) ∈ relation(c)

**axiom anti-symmetric**

∀(c: Precedence-Constraint, x:Task, y:Task)
(x,y) ∈ relation(c) ⇒ ¬ (y,x) ∈ relation(c)

**op observes?: Precedence-Constraint, Schedule → Boolean**

def observes?(c,s) = ∀(p:Reservation, q:Reservation ∈ reservations(s))
relation(c)(p,q) ⇒ completion-time(p) < start-time(q)

2.1.11 **SORT RESERVATION**

**op task: Reservation → Task**

**op resource: Reservation → Resource**

**op start-time: Reservation → Time**

**op completion-time: Reservation → Time**

**axiom uniqueness**

∀(p:Reservation, q:Reservation)
p=q ⇔ task(p)=task(q) ∧ resource(p)=resource(q) ∧
start-time(p)=start-time(q) ∧ completion-time(p)=completion-time(q)

**op duration: Reservation → Duration**

def duration(r) = completion-time(r) – start-time(r)

**op precedes?: Reservation, Reservation → Boolean**

def precedes?(r,t) = completion-time(r) ≤ start-time(t)

**op lateness: Reservation → Duration**

def lateness(r) = completion-time(r) - due-date(task(r))

**op tardiness: Reservation → Duration**

def tardiness(r) = max(0.0, lateness(r))

**op release-date-observed?: Reservation → Boolean**

def release-date-observed?(r) = start-time(r) ≥ release-date(task(r))

**op due-date-observed?: Reservation → Boolean**

def due-date-observed?(r) = completion-time(r) ≤ due-date(task(r))

**op compatible-resource-and-task?: Reservation → Boolean**

def compatible-resource-and-task?(r) = compatible-task?(resource(r), task(r))
**op matching-processing-time**: Reservation $\rightarrow$ Boolean

def matching-processing-time?(r) = duration(r) $\geq$ processing-time(resource(r), task(r))

**op same-resource**: Reservation, Reservation $\rightarrow$ Boolean

def same-resource?(r,t) = resource(r)=resource(t)

**op sufficient-setup-time**: Reservation, Reservation|same-resource? $\rightarrow$ Boolean

def sufficient-setup-time?(r,t) = 

\[ \text{start-time(t)} - \text{completion-time(r)} \geq \text{setup-time(resource(r), task(r), task(t))} \]

2.1.12 **SORT SCHEDULE**

**op all-reservations**: Schedule $\rightarrow$ set(Reservation)

*axiom uniqueness*

\[ \forall (s:\text{Schedule}, t:\text{Schedule}) \]  

\[ s=t \iff \text{all-reservations}(s)=\text{all-reservations}(t) \]

**op reservations**: Schedule, Resource $\rightarrow$ set(Reservation)

def reservation(s,r) = \{ m | m $\in$ all-reservations(s) $\land$ resource(m)=r \}

**op consecutive**: Schedule, Reservation, Reservation $\rightarrow$ Boolean

def consecutive?(s,p,q) = 

\[ \text{same-resource?}(p,q) \land \text{precedes?}(p,q) \land \neg \exists (r \in \text{reservations}(s, \text{resource}(p)) \text{ precedes?}(p,r) \land \text{precedes?}(r,q) \]

**op closure-time**: Schedule $\rightarrow$ Time

def closure-time(s) = max \{ completion-time(r) | r $\in$ all-reservations(s) \}

**op makespan**: Schedule $\rightarrow$ Time

def makespan(s) = closure-time(s)

**op maximum-tardiness**: Schedule $\rightarrow$ Time

def maximum-tardiness(s) = max \{ tardiness(r) | r $\in$ all-reservations(s) \}

**op total-weighted-tardiness**: Schedule $\rightarrow$ Real

def total-weighted-tardiness(s) = reduce(all-reservations(s), +, λ r→ tardiness(r)*weight(task(r)))

**op all-tasks-scheduled**: Schedule, set(Task) $\rightarrow$ Boolean

def all-tasks-scheduled?(s,T) = \forall (t \in T) \text{ size}\{ r | r \in \text{all-reservations}(s) \land \text{task}(r)=t \} = 1
% Function for checking hard and precedence constraints
% Run a schedule through a set of constraints to check if it observes each constraint.

\textbf{op check-feasibility: Schedule, set(Resource), set(Task),}
\textbf{set(Precedence-Constraint), set(Hard-Constraint)}
\rightarrow \textbf{Boolean}

\textbf{def check-feasibility(s,R,T,P,H) = }\forall (p \in P) \text{ observes}(p, s) \land \forall (h \in H) \text{ observes}(h, s, R, T)

% Function for checking soft constraints
% Compute the total penalty on a schedule as the weighted sum of penalties arising
% from soft constraints

\textbf{op total-penalty: Schedule, set(Resource), set(Task), set(Soft-Constraint) \rightarrow Penalty}
\textbf{def total-penalty(s, R, T, P) = }\text{reduce}(P, +, \lambda p \rightarrow \text{penalty}(p, s, R, T))
2.2 **Examples**

2.2.1 **Typical Constraints**

% ____________________________________________________
% Hard constraints

const release-dates-constraint: Hard-Constraint
    = pointwise-constraint(release-date-observed?)
const due-dates-constraint: Hard-Constraint
    = pointwise-constraint(due-date-observed?)
const compatibility-constraint: Hard-Constraint
    = pointwise-constraint(compatible-resource-and-task?)
const processing-times-constraint: Hard-Constraint
    = pointwise-constraint(matching-processing-time?)
const setup-times-constraint: Hard-Constraint
    = consecutive-constraint(sufficient-setup-time?)
const complete-schedule-constraint: Hard-Constraint
    = global-constraint(\(\lambda s, R, T \rightarrow \text{all-tasks-scheduled}(s, T)\))

% ____________________________________________________
% Soft constraints

% Define a penalty factor that starts at 0 and increases to 1 with the lateness, % relative to a scale defined by the expected processing time of the task.

**op lateness-penalty-factor**: Reservation \(\rightarrow\) Real

def lateness-penalty-factor(r) = 1-exp(-lateness(r)/processing-time(resource(r), task(r)))

% Constraint: completion times come before due dates
% Penalty for violation increases with lateness and is weighted by task priority
const weighted-tardiness-constraint: Soft-Constraint
    = pointwise-constraint(
        \(\lambda r \rightarrow \text{completion-time}(r) \leq \text{due-date}(\text{task}(r)),\)
        \(\lambda r \rightarrow \text{lateness-penalty-factor}(r) \times \text{priority}(\text{task}(r))\))
2.2.2 *Find a Feasible Schedule*

% Collect the constraints that are to apply
const Hard-Constraints: set(Hard-Constraint)
  = \{ release-dates-constraint, 
    due-dates-constraint, 
    compatibility-constraint, 
    processing-times-constraint, 
    setup-times-constraint, 
    complete-schedule-constraint \}

const Precedence-Constraints: set(Precedence-Constraint)
% not defined

% Define the function that finds a feasible schedule
op find-feasible-schedule: set(Resource), set(Task) → Schedule

axiom feasible and complete
∀(R,T) check-feasibility(find-feasible-schedule(R,T), R, T, 
  Precedence-Constraints, Hard-Constraints)
2.2.3 **Find an Optimal Schedule**

% Collect the constraints that are to apply
const Hard-Constraints: set(Hard-Constraint)
    = { release-dates-constraint,
         compatibility-constraint,
         processing-times-constraint,
         setup-times-constraint,
         complete-schedule-constraint}

const Precedence-Constraints: set(Precedence-Constraint)
% not defined

const Soft-Constraints: set(Soft-Constraint)
    = { weighted-tardiness-constraint }

% Define the function that finds an optimal schedule,
% i.e., one with a minimal penalty
op find-optimal-schedule: set(Resource), set(Task) → Schedule

axiom feasible and complete
∀(R,T) check-feasibility(find-optimal-schedule(R,T), R, T,
    Precedence-Constraints, Hard-Constraints)

% No other feasible schedule has a lower penalty
axiom optimality
∀(R,T) ∀(s’:schedule)
    s≠s’ ∧ check-feasibility(s’, R, T, Precedence-Constraints, Hard-Constraints)
    ⇒ total-penalty(s’, R, T, Soft-Constraints)≥total-penalty(s, R, T, Soft-Constraints)