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 $\rightarrow$ Task-Type

axiom uniqueness

$\forall$(t:Task, u:Task)

t=u $\Leftrightarrow$ release-date(t)=release-date(u) $\wedge$ due-date(t)=due-date(u) $\wedge$
weight(t)=weight(u) $\wedge$ type(t)=type(u)

2.1.6 Sort Resource-Type

% not defined here – problem specific

Sort Resource

op type: Resource $\rightarrow$ Resource-Type
op compatible-task-types: Resource $\rightarrow$ set(Task-Type)
op processing-times: Resource $\rightarrow$ (Task-Type $\rightarrow$ Duration)
op setup-times: Resource $\rightarrow$ (Task-Type, Task-Types $\rightarrow$ Duration)

axiom uniqueness

$\forall$(r:Resource, s:Resource)

r=s $\Leftrightarrow$ type(r)=type(s) $\wedge$ compatible-task-types(r)=compatible-task-types(s) $\wedge$
processing-times(r)=processing-times(s) $\wedge$
setup-times(r)=setup-times(s)

op compatible-task?: Resource, Task $\rightarrow$ Boolean
def compatible-task?(r,t) = Task.type(t) $\in$ compatible-task-types(r)

op processing-time: Resource, Task|compatible-task? $\rightarrow$ Duration
def processing-time(r,t) = processing-times(r)(Task.type(t))

op two-compatible-tasks?: Resource, Task, Task $\rightarrow$ Boolean
def two-compatible-tasks?(r,t,u) = compatible-task?(r,t) $\wedge$ compatible-task?(r,u)

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

2.1.7 Sort Hard-Constraint

% An arbitrary boolean-valued function
op global-constraint: (Schedule, set(Resource), set(Task) $\rightarrow$ Boolean)
$\rightarrow$ Hard-Constraint
% This op lifts a single-reservation constraint to a schedule constraint
op pointwise-constraint: (Reservation $\rightarrow$ Boolean) $\rightarrow$ Hard-Constraint
% This op lifts a constraint on neighboring reservations to a schedule constraint
op consecutive-constraint: (Reservation, Reservation $\rightarrow$ Boolean)
$\rightarrow$ Hard-Constraint

op observes?: Hard-Constraint, Schedule, set(Resource), set(Task) → 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), ∧, λ r→f(r))
def observes?(consecutive-constraint(f),s,R,T) =
∀(r ∈ resources(s); p,q ∈ reservations(s,r)) consecutive?(s,p,q) ⇒ f(p,q)

2.1.8 Sort Penalty = Real

2.1.9 Sort Soft-Constraint

% An arbitrary function
op global-constraint: (Schedule → Penalty) → Soft-Constraint
% Apply a penalty pointwise to each reservation that violates a constraint, and sum
op pointwise-constraint:
(Reservation → Boolean), (Reservation → Penalty) → Soft-Constraint
% Apply a penalty pointwise to each reservation that violates a constraint,
% and take the maximum
op max-constraint:
(Reservation → Boolean), (Reservation → Penalty) → Soft-Constraint
% Apply a pair-wise penalty to each pair of consecutive reservations that violates a
% constraint, and sum
op consecutive-constraint: (Reservation, Reservation → Boolean),
(Reservation, Reservation → Penalty) → Soft-Constraint

op penalty: Soft-Constraint, set(Resource), set(Task), Schedule → Penalty
def penalty(global-constraint(f),s,R,T)) = f(s,R,T)
def penalty(pointwise-constraint(p,f),s,R,T)
= reduce(all-reservation(s), +, λ r→if p(r) then 0.0 else f(r))
def penalty(max-constraint (p,f),s,R,T)
= reduce(all-reservation(s), max, λ r→if p(r) then 0.0 else f(r))
def penalty(consecutive-constraint (p,f),s,R,T)
= reduce({(r,t) | r, t ∈ all-reservations(s) ∧ consecutive?(s,r,t)}, +,
λ r,t → if p(r,t) then 0.0 else f(r,t))

2.1.10 Sort Precedence-Constraint

op relation: Precedence-Constraint → set((Task, Task))
axiom uniqueness
∀(c:Precedence-Constraint, d:Precedence-Constraint)
c=d ⇔ relation(c)=relation(d)

% A precedence relation must be a strict partial order
% (i.e., a partial order with anti-reflexivity instead of reflexivity)
axiom anti-reflexive
∀(c:Precedence-Constraint, x:Task)

$\neg$ (x,x)$\in$relation(c)

axiom transitive

$\forall$(c: Precedence-Constraint, x:Task, y:Task, z:Task)

(x,y)$\in$relation(c) $\wedge$ (y,z)$\in$relation(c) $\Rightarrow$ (x,z)$\in$relation(c)

axiom anti-symmetric

$\forall$(c: Precedence-Constraint, x:Task, y:Task)

(x,y)$\in$relation(c) $\Rightarrow$ $\neg$ (y,x)$\in$relation(c)

op observes?: Precedence-Constraint, Schedule $\rightarrow$ Boolean

def observes?(c,s) = $\forall$(p:Reservation, q:Reservation $\in$ reservations(s))

relation(c)(p,q) $\Rightarrow$ completion-time(p)<start-time(q)

2.1.11 Sort Reservation

op task: Reservation $\rightarrow$ Task

op resource: Reservation $\rightarrow$ Resource

op start-time: Reservation $\rightarrow$ Time

op completion-time: Reservation $\rightarrow$ Time

axiom uniqueness

$\forall$(p:Reservation, q:Reservation)

p=q $\Leftrightarrow$ task(p)=task(q) $\wedge$ resource(p)=resource(q) $\wedge$

start-time(p)=start-time(q) $\wedge$ completion-time(p)=completion-time(q)

op duration: Reservation $\rightarrow$ Duration

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

op precedes?: Reservation, Reservation $\rightarrow$ Boolean

def precedes?(r,t) = completion-time(r)$\leq$start-time(t)

op lateness: Reservation $\rightarrow$ Duration

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

op tardiness: Reservation $\rightarrow$ Duration

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

op release-date-observed?: Reservation $\rightarrow$ Boolean

def release-date-observed?(r) = start-time(r)$\geq$release-date(task(r))

op due-date-observed?: Reservation $\rightarrow$ Boolean

def due-date-observed?(r) = completion-time(r)$\leq$due-date(task(r))

op compatible-resource-and-task?: Reservation $\rightarrow$ Boolean

def compatible-resource-and-task?(r) = compatible-task?(resource(r), task(r))

op matching-processing-time?: Reservation → Boolean
def matching-processing-time?(r) = duration(r)≥processing-time(resource(r), task(r))

op same-resource?: Reservation, Reservation → Boolean
def same-resource?(r,t) = resource(r)=resource(t)

op sufficient-setup-time?: Reservation, Reservation|same-resource? → Boolean
def sufficient-setup-time?(r,t) =
  start-time(t) - completion-time(r) ≥ setup-time(resource(r), task(r), task(t))

2.1.12 Sort Schedule

op all-reservations: Schedule → set(Reservation)

axiom uniqueness
 ∀(s:Schedule, t:Schedule)
  s=t ⇔ all-reservations(s)=all-reservations(t)

op reservations: Schedule, Resource → set(Reservation)
def reservation(s,r) = {m | m ∈ all-reservations(s) ∧ resource(m)=r}

op consecutive?: Schedule, Reservation, Reservation → Boolean
def consecutive?(s,p,q) =
  same-resource?(p,q) ∧
  precedes?(p,q) ∧
  ¬ ∃(r ∈ reservations(s, resource(p)) precedes?(p,r) ∧ precedes?(r,q)

op closure-time: Schedule → Time
def closure-time(s) = max{completion-time(r) | r ∈ all-reservations(s)}

op makespan: Schedule → Time
def makespan(s) = closure-time(s)

op maximum-tardiness: Schedule → Time
def maximum-tardiness(s) = max{tardiness(r) | r ∈ all-reservations(s)}

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

op all-tasks-scheduled?: Schedule, set(Task) → Boolean
def all-tasks-scheduled?(s,T) = ∀(t ∈ T) size{r | r ∈ all-reservations(s) ∧ 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.
op check-feasibity: Schedule, set(Resource), set(Task),
set(Precedence-Constraint), set(Hard-Constraint)
→ Boolean

def check-feasibility(s,R,T,P,H) = ∀(p∈P) observes?(p, s) ∧ ∀(h∈H) 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
op total-penalty: Schedule, set(Resource), set(Task), set(Soft-Constraint) → Penalty
def total-penalty(s, R, T, P) = reduce(P, +, λ p→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(λ s,R,T→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 → 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(
                λ r→completion-time(r)≤due-date(task(r)),
                λ r→lateness-penalty-factor(r)*priority(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)