Scheduling

Planning with Actions that Require Resources

Literature

Planning and Scheduling

- solution to planning problem:
  - plan: partially ordered set of actions
  - actions: fully instantiated operators
    - require resources
- resources:
  - can be modelled as parameters of an action
    - problem: planning algorithms tries out all possibilities (inefficient)
  - alternative approach:
    - allow unbound resource variables in plan (planning)
    - find assignment of resources to actions (scheduling)

Overview

- Scheduling Problems and Schedules
  - Searching for Schedules
Actions and Resources

- resources: an entity needed to perform an action
  - state variables: modified by actions in absolute ways
    - example: move(r,l,l'):
      - location changes from l to l'
  - resource variables: modified by actions in relative ways
    - example: move(r,l,l'):
      - fuel level changes from f to f-f'

Actions with Time Constraints

- Let a be an action in a planning domain:
  - attached time constraints:
    - earliest start time: $s_{min}(a)$ – actual start time: $s(a)$
    - latest end time: $s_{max}(a)$ – actual end time: $e(a)$
    - duration: $d(a)$
  - action types:
    - preemptive actions: cannot be interrupted
      - $d(a) = e(a) - s(a)$
    - non-preemptive actions: can be interrupted
      - resources available to other actions during interruption
**Actions with Resource Constraints**

- Let $a$ be an action in a planning domain:
  - attached resource constraints:
    - required resource: $r$
    - quantity of resource required: $q$
  - reusable: resource will be available to other actions after this action is completed
  - consumable: resource will be consumed when action is complete

**Reusable Resources**

- resource availability:
  - total capacity: $Q_r$
  - current level at time $t$: $z_r(t)$
- resource requirements:
  - $\text{require}(a, r, q)$: action $a$ requires $q$ units of resource $r$ while it is active
- resource profile:

\[
\begin{align*}
Q_r & \quad q_1 \quad q_2 \\
\text{a}_1: \text{require}(a_1, r, q_1) \quad \text{a}_2: \text{require}(a_2, r, q_2)
\end{align*}
\]
**Consumable Resources**

- resource availability:
  - total reservoir at $t_0$: $Q_r$
  - current level at time $t$: $z_r(t)$
- resource consumption/production:
  - consume($a, r, q$): action $a$ requires $q$ units of resource $r$
  - produce($a, r, q$): action $a$ produces $q$ units of resource $r$
- resource profile:

```
Q_r: \[ a_1: \text{consume}(a_1, r, q_1) \]
    \[ a_2: \text{consume}(a_2, r, q_2) \]
    \[ a_3: \text{produce}(a_3, r, q_3) \]
```

**Other Resource Features**

- discrete vs. continuous
  - countable number of units: cranes, bolts
  - real-valued amount: bandwidth, electricity
- unary
  - $Q_r = 1$; exactly one resource of this type available
- sharable
  - can be used by several actions at the same time
- resources with states
  - actions may require resources in specific state
Combining Resource Constraints

- **conjunction:**
  - action uses multiple resources while being performed

- **disjunction:**
  - action requires resources as alternatives
  - cost/time may depend on resource used

- **resource types:**
  - resource-class(s) = \{r_1, \ldots, r_m\}: require(a, s, q)
  - equivalent to disjunction over identical resources

Cost Functions and Optimization Criteria

- **cost function parameters**
  - quantity of resource required
  - duration of requirement

- **optimization criteria:**
  - total schedule cost
  - makespan (end time of last action)
  - weighted completion time
  - (weighted) number of late actions
  - (weighted) maximum tardiness
  - resource usage
Machine Scheduling

- machine: resource of unit capacity
  - either available or not available at time $t$
  - cannot process two actions at the same time
- job $j$: partially ordered set of actions $a_{j1}, \ldots, a_{jk}$
  - action $a_j$ requires
    - one resource type
    - for a number of time units
  - actions in same job must be processed sequentially
  - actions in different jobs are independent (not ordered)
- machine scheduling problem:
  - given: $n$ jobs and $m$ machines
  - schedule: mapping from actions to machines + start times

Example: Scheduling Problem

- machines:
  - $m_1$ of resource type $r_1$
  - $m_2, m_3$ of resource type $r_2$
- jobs:
  - $j_1$: $\langle r_1(3), r_2(3), r_1(3) \rangle$
    - three actions, totally ordered
    - $a_{11}$ requires 3 units of resource type 1, etc.
  - $j_2$: $\langle r_2(3), r_1(5) \rangle$
  - $j_3$: $\langle r_1(3), r_1(2), r_2(3), r_1(5) \rangle$
Example: Schedules by Job

- **machines:**
  - $m_1$ of type $r_1$
  - $m_2$ of type $r_2$

- **jobs:**
  - $j_1$: $\langle r_1(1), r_2(2) \rangle$
  - $j_2$: $\langle r_1(3), r_2(1) \rangle$

Example: Schedules by Machine

- **machines:**
  - $m_1$ of type $r_1$
  - $m_2$ of type $r_2$

- **jobs:**
  - $j_1$: $\langle r_1(1), r_2(2) \rangle$
  - $j_2$: $\langle r_1(3), r_2(1) \rangle$
Overview

- Scheduling Problems and Schedules
- Searching for Schedules

Assignable Actions

- Let $P$ be a machine scheduling problem. Let $S$ be a partially defined schedule.
- An action $a_{ji}$ of some job $j_i$ in $P$ is unassigned if it does not appear in $S$.
- An action $a_{ji}$ of some job $j_i$ in $P$ is assignable if it has no unassigned predecessors in $S$. 
Example: Assignable Actions

- problem $P$:
  - machines:
    - $m_1$ of type $r_1$
    - $m_2$ of type $r_2$
  - jobs:
    - $j_1$: $\langle r_1(1), r_2(2) \rangle$
    - $j_2$: $\langle r_1(3), r_2(1) \rangle$
    - $j_3$: $\langle r_1(3), r_2(1), r_1(3) \rangle$
- partial schedule $S$:
  - assignable:
    - $a_{22}, a_{31}$
  - unassigned:
    - $a_{22}, a_{31}, a_{32}, a_{33}$

Earliest Assignable Time

- Let $a_{ji}$ be an assignable action in $S$. The earliest assignable time for $a_{ji}$ on machine $m$ in $S$ is:
  - the end of the last action currently scheduled on $m$ in $S$, or
  - the end of the last predecessor ($a_{j0} \ldots a_{ji-1}$) in $S$,
whichever comes later.
Example: Earliest Assignable Time

- problem $P$:
  - machines:
    - $m_1$ of type $r_1$
    - $m_2$ of type $r_2$
  - jobs:
    - $j_1$: $(r_1(1), r_2(2))$
    - $j_2$: $(r_1(3), r_2(1))$
    - $j_3$: $(r_1(3), r_2(1), r_1(3))$

- partial schedule $S$:

```
<table>
<thead>
<tr>
<th>m1</th>
<th>0 2 4 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>a11</td>
<td></td>
</tr>
<tr>
<td>a21</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>m2</th>
<th>0 2 4 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>a12</td>
<td></td>
</tr>
</tbody>
</table>
```

- earliest assignable time for $a_{22}$ on $m_2$: 4
- earliest assignable time for $a_{31}$ on $m_1$: 4

Heuristic Search

```python
def heuristicScheduler(P, S):
    assignables ← P.getAssignables(S)
    if assignables.isEmpty() then return S
    action ← assignables.selectOne()
    machines ← P.getMachines(action)
    machine ← machines.selectOne()
    time ← S.getEarliestAssignableTime(action, machine)
    S ← S + assign(action, machine, time)
    return heuristicScheduler(P, S)
```
Using Local Search

• issues:
  • representing schedules
  • generating a random initial schedule
  • generating neighbours
  • evaluating neighbours (schedules)

Schedule Representation

• representation:
  • totally ordered list of all actions with assigned machines
  • example: \( \langle (a_{11}, m_1), (a_{21}, m_1), (a_{12}, m_2), (a_{22}, m_2) \rangle \)

• schedule:
  • assign actions in sequence to given machines at earliest assignable times
  • example:

\[
\begin{align*}
m_1 & \quad a_{11} & \quad a_{21} \\
m_2 & \quad a_{12} & \quad a_{22}
\end{align*}
\]
Initial Schedule and Evaluation

- generating random schedules:
  - randomly choose an assignable action
  - randomly choose a machine of the right resource type for that action
  - append the action-machine pair to the list of assignments
  - do this until all actions are assigned
- evaluating schedules:
  - generate schedule from list
  - apply optimization criterion

Generating Neighbours

- machine neighbours:
  - change the machine assigned to an action to any other machine
- position neighbours:
  - change the position of an action $a$ in the list:
    - $a_{\text{min}}$: the latest predecessor of $a$ in the current list
    - $a_{\text{max}}$: the earliest successor of $a$ in the current list
    - move $a$ anywhere between $a_{\text{min}}$ and $a_{\text{max}}$
LocalSearchScheduler: Pseudo Code

function LocalSearchScheduler(P)
    best ← randomSchedule(P)
    loop MAXLOOP times
        S ← randomSchedule(P)
        do
            succs ← S.getBestNeighbours(P)
            next ← succs.selectOne()
            if S.evaluate() < next.evaluate() then
                S ← next
        while S = next
        if S.evaluate() > best.evaluate() then
            best ← S
    return best

Overview

- Scheduling Problems and Schedules
- Searching for Schedules