Constraint-based Scheduling: Search Strategies

Hana Rudová*, Rong Qu’

*Fakulty of Informatics, Masaryk University
Brno, Czech Republic
http://www.fi.muni.cz/~hanka

’School of Computer Science
University of Nottingham, UK
http://www.cs.nott.ac.uk/~rxq
Outline

1 Search

2 Search Strategies for Scheduling
Constraint propagation techniques are (usually) incomplete
⇒ search algorithm is needed to solve the "rest"

**Labeling**
- depth-first search (DFS/BT)
  - assign value to variable
  - propagate = make the problem locally consistent
  - backtrack in case of failure

\[
\begin{align*}
  X \text{ in } 1..5 & \equiv X=1 \lor X=2 \lor X=3 \lor X=4 \lor X=5
\end{align*}
\]

Generally search algorithm solves remaining disjunctions
- \(X=1 \lor X \neq 1\) standard assignment
- \(X < 3 \lor X \geq 3\) domain splitting
- \(X < Y \lor X \geq Y\) variable ordering (scheduling: tasks ordering)
Search: variable ordering

Which variable should be assigned first?

First-fail principle

- prefer variable with the hardest assignment
- for example variable with the smallest domain:
  domain can be emptied easily
- or variable with the most constraints:
  assignment of other variables constrain and
  make the domain smaller easily

Variable ordering defines shape of the search tree

- selection of variable with small domain size:
  small branching on this level ... more options left for later
- selection of variable with large domain size:
  large branching on this level ... less options left for later
Search: value ordering

What value should be chosen first?

Succeed-first principle
- prefers values with probably belongs to the solution
- for example the values with most supports in neighbouring variables
- this heuristic is usually problem specific

Value ordering defines the order how the branches are explored
Branching schemes in scheduling

Branching = resolving disjunctions

Traditional scheduling approaches

- take the critical decisions first
  - resolve bottlenecks, ...
  - defines the shape of the search tree
  - recall the first-fail principle

- prefer an alternative leaving more flexibility
  - defines order of branches to be explored
  - recall succeed-first principle

How to describe criticality and flexibility formally?
Slack

- **Slack** is a formal description of flexibility
- Slack for **a given order of two activities**
  "free time for shifting the activities"

\[
\text{slack}(A << B) = \max(\text{end}(B)) - \min(\text{start}(A)) - p(A) - p(B)
\]

- Slack for **two activities** (without any ordering)
  \[
  \text{slack}(\{A, B\}) = \max(\text{slack}(A << B), \text{slack}(B << A))
  \]

- Slack for **a group of activities**
  \[
  \text{slack}(\Omega) = \max(\text{end}(\Omega)) - \min(\text{start}(\Omega)) - p(\Omega)
  \]
Order branching

\[ A << B \lor \neg A << B \]

What activities \( A, B \) should be ordered first?
- the most critical pair (first-fail)
- the pair with the minimal slack(\( \{A, B\} \))

What order of activities \( A \) and \( B \) should be selected?
- the most flexible order
- the order with \textbf{with the maximal slack} \( A\??B \)

\( O(n^2) \) choice points
First/last branching

\[(A << \Omega \lor \neg A << \Omega) \lor (\Omega << A \lor \neg \Omega << A)\]

Should we look for the first or last activity?

- look to the set of possible candidates for first activity and to the set of possible candidates for last activities
- select a **smaller set** from these (first-fail)
  - smaller number of candidates means that it is harder to find a suitable candidate

What activity should be selected?

- if first activity is being selected then the activity with the **smallest** \(\min(\text{start}(A))\) is preferred
- if last activity is being selected then the activity with the **largest** \(\max(\text{end}(A))\) is preferred

\(O(n)\) choice points
Resource slack is defined as a slack of the set of activities processed by the resource.

How to use a resource slack?

- choosing a resource on which the activities will be ordered first
  - resource with a minimal slack (bottleneck) preferred
- choosing a resource on which the activity will be allocated
  - resource with a maximal slack (flexibility) preferred