Constraint Programming
An Appetizer

Christian Schulte
schulte@imit.kth.se

Laboratory of Electronics and Computer Systems
Institute of Microelectronics and Information Technology
KTH – Royal Institute of Technology
Stockholm, Sweden

KTH Microelectronics and Information Technology

Constraint Programming

- Solving combinatorial problems

start with a first toy problem

Send More Money (SMM)

- Find distinct digits for letters, such that

```
    SEND
+   MORE
= MONEY
```


Constraint Model for SMM

- Variables:
  S,E,N,D,M,O,R,Y ∈ {0,…,9}
- Constraints:

```
  distinct(S,E,N,D,M,O,R,Y)
                 1000×S+100×E+10×N+D
   +             1000×M+100×O+10×R+E
   =  10000×M+1000×O+100×N+10×E+Y
  S≠0      M≠0
```


Solving SMM

- Find values for variables

  such that

all constraints satisfied

Finding a Solution

- Enumerate assignments: poor!
- Constraint programming
  - compute with possible values
  - prune inconsistent values
    constraint propagation
  - search
    - branch: define search tree
    - explore: explore search tree for solution

Overview

Overview

- Constraint propagation
- Search
- Summary: Applications & Principles
- Application examples
  - resource scheduling
  - instruction scheduling
- Constraint Programming 2G1515

Constraint Propagation

Important Concepts

- Constraint store
- Basic constraint
- Propagator
- Non-basic constraint
- Constraint propagation

Constraint Store

$x \in \{3,4,5\}$ $y \in \{3,4,5\}$

- Stores basic constraints

  map variables to possible values

Constraint Store

finite domain constraints

$x \in \{3,4,5\}$ $y \in \{3,4,5\}$

- Stores basic constraints

  map variables to possible values

Constraint Store

x∈{3,4,5} y∈{3,4,5}

- Stores basic constraints
  - map variables to possible values
- Domains: finite sets, real intervals, trees, …

Propagators

- Implement non-basic constraints

$\texttt{distinct}(x_1,\ldots,x_n)$

$x + 2 \times y = z$

Propagators

x≥y y>3

x∈{3,4,5} y∈{3,4,5}

- Amplify store by constraint propagation

Propagators

x≥y y>3

x∈{3,4,5} y∈{3,4,5}

- Amplify store by constraint propagation

Propagators

x≥y y>3

x∈{3,4,5} y∈{4,5}

- Amplify store by constraint propagation

Propagators

x≥y y>3

x∈{3,4,5} **y∈{4,5}**

- Amplify store by constraint propagation

Propagators

x≥y y>3

x∈{4,5} y∈{4,5}

- Amplify store by constraint propagation

Propagators

x≥y y>3

x∈{4,5} y∈{4,5}

- Amplify store by constraint propagation
- Disappear when done (entailed)
  - no more propagation possible

Propagators

x≥y

x∈{4,5} y∈{4,5}

- Amplify store by constraint propagation
- Disappear when done (entailed)
  - no more propagation possible

Computation Space

x≥y y>3

x∈{4,5} y∈{4,5}

- Store with connected propagators

Propagation for SMM

- Results in store

```
S=9   E∈{4,…,7} N∈{5,…,8} D∈{2,…,8}
M=1   O=0       R∈{2,…,8} Y∈{2,…,8}
```

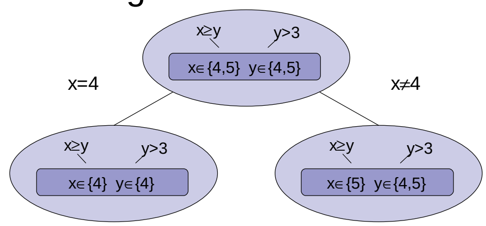
- Propagation alone not sufficient!
  - create simpler sub-problems
  - branching

Search

Important Concepts

- Branching
- Exploration
- Branching heuristics
- Best solution search

Branching



- Yields spaces with additional constraints
- Enables further constraint propagation

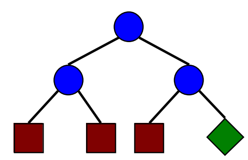
Branching Strategy

- Pick variable x with at least two values
- Pick value n from domain of x
- Branch with

  x=n and x≠n

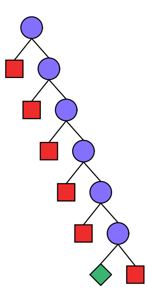
- Part of model

Search



- Iterate propagation and branching
- Orthogonal: branching ⇆ exploration
- Nodes:
  - **Unsolved** • **Failed** • **Succeeded**

SMM: Solution



```
   SEND
+  MORE
= MONEY

   9567
+  1085
= 10652
```

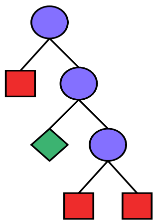

Heuristics for Branching

- Which variable
  - least possible values (first-fail)
  - application dependent heuristic
- Which value
  - minimum, median, maximum

    x=m or x≠m
  - split with median m

    x<m or x≥m
- Problem specific

SMM: Solution With First-fail



```
   SEND
+  MORE
= MONEY

   9567
+  1085
= 10652
```


Send Most Money (SMM++)

- Find distinct digits for letters, such that

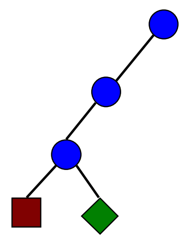
```
   SEND
+  MOST
= MONEY
```

and MONEY maximal

Best Solution Search

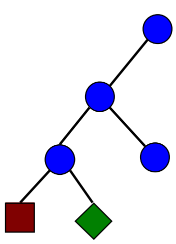
- Naïve approach:
  - compute all solutions
  - choose best
- Branch-and-bound approach:
  - compute first solution
  - add “betterness” constraint to open nodes
  - next solution will be “better”
  - prunes search space

Branch-and-bound Search



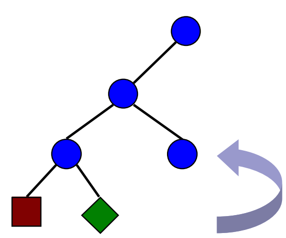
- Find first solution

Branch-and-bound Search



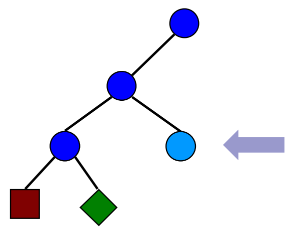
- Explore with additional constraint

Branch-and-bound Search



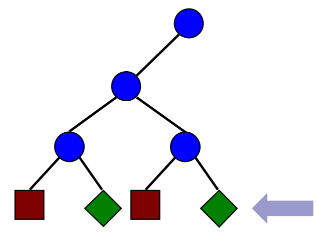
- Explore with additional constraint

Branch-and-bound Search



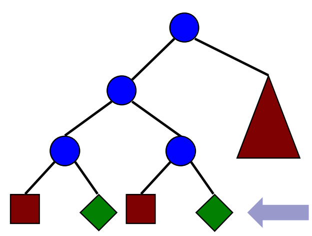
- Guarantees better solutions

Branch-and-bound Search



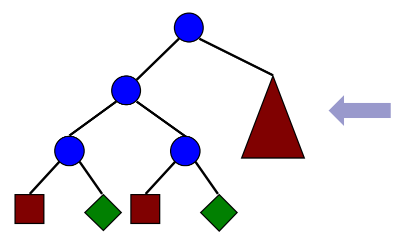
- Guarantees better solutions

Branch-and-bound Search



- Last solution best

Branch-and-bound Search



- Proof of optimality

Modelling SMM++

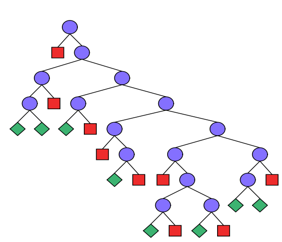
- Constraints and branching as before
- Order among solutions with constraints
  - so-far-best solution S,E,N,D,M,O,T,Y
  - current node S,E,N,D,M,O,T,Y
  - constraint added

    10000×M+1000×O+100×N+10×E+Y

    <

    10000×M+1000×O+100×N+10×E+Y

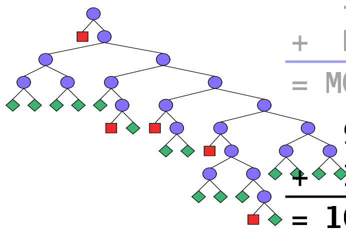
SMM++: Branch-and-bound



```
   SEND
+  MOST
= MONEY

   9782
+  1094
= 10876
```

SMM++: All Solution Search



```
   SEND
+  MOST
= MONEY

   9782
+  1094
= 10876
```


Summary

Summary

applications

principles

- Modeling
  - variables with domain
  - constraints to state relations
  - branching strategy
  - solution ordering
- Solving
  - constraint propagation
  - constraint branching
  - search tree exploration

Application Areas

- Timetabling
- Scheduling
- Crew rostering
- Resource allocation
- Workflow planning and optimization
- Gate allocation at airports
- Sports-event scheduling
- Railroad: track allocation, train allocation, schedules
- Automatic composition of music
- Genome sequencing
- Frequency allocation
- …

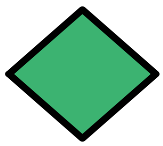
Application Examples

- Scheduling resources
  - machines, personal, …
  - constraint programming showcase
- Instruction scheduling
  - used in compilation
  - for modern microprocessors with latencies

Principles

- Models for constraint propagation
  - properties and guarantees
- Strong constraint propagation
  - global constraints with strong algorithmic methods
  - mantra: search kills, search kills, search k…
- Branching strategies
- Exploration strategies

SMM: Strong Propagation



```
    SEND
+   MORE
= MONEY

    9567
+   1085
= 10652
```


Used Techniques

- Artificial intelligence
- Operations research
- Algorithms
- Programming languages

Why Does CP Matter?

- Middleware for combining smart algorithmic components
  - scheduling
  - graphs
  - flows
  - …

  plus
  - essential extra constraints

Significance

- Constraint programming identified as a strategic direction in computer science research
  [ACM Computing Surveys, December 1996]
- Applications are ubiquitous
  - knowledgeable people are not!

Relation to Logic Programming

- Logic programming
  horn clauses + trees with unification
- Constraint logic programming
  horn clauses + constraint domain
- Constraint programming
  X + constraint domain
  - with focus on constraints
  - refined search language

Scheduling Resources

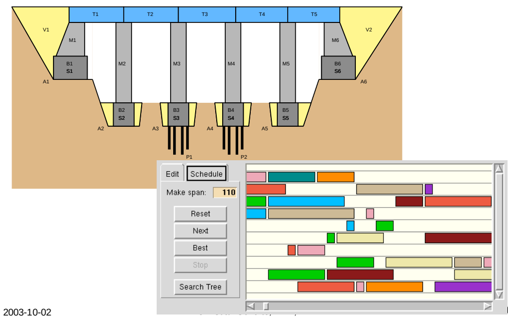
- Modeling
- Propagation
- Strong propagation

Scheduling Resources: Given

- Tasks
  - duration
  - resource
- Precedence constraints
  - determine order among two tasks
- Resource constraints
  - at most one task per resource

    [disjunctive, non-preemptive scheduling]

Scheduling: Bridge Example



Scheduling: Solution

- Start time for each task
- All constraints satisfied
- Earliest completion time
  - minimal make-span

Scheduling: Model

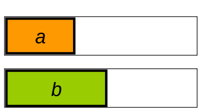
- Variable for start-time of task *a*

  start(*a*)
- Precedence constraint: *a* before *b*

  start(*a*) + dur(*a*) $\leq$ start(*b*)

Propagating Precedence

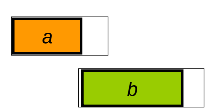
a before *b*



start(*a*)$\in\{0,\ldots,7\}$
start(*b*)$\in\{0,\ldots,5\}$

Propagating Precedence

a before *b*



start(*a*)$\in\{0,\ldots,7\}$
start(*b*)$\in\{0,\ldots,5\}$

start(*a*)$\in\{0,\ldots,2\}$
start(*b*)$\in\{3,\ldots,5\}$

Scheduling: Model

- Variable for start-time of task *a*
  - start(*a*)
- Precedence constraint: *a* before *b*
  - start(*a*) + dur(*a*) ≤ start(*b*)
- Resource constraint:
  - *a* before *b*
  - or
  - *b* before *a*

Scheduling: Model

- Variable for start-time of task *a*
  - start(*a*)
- Precedence constraint: *a* before *b*
  - start(*a*) + dur(*a*) ≤ start(*b*)
- Resource constraint:
  - start(*a*) + dur(*a*) ≤ start(*b*)
  - or
  - *b* before *a*

Scheduling: Model

- Variable for start-time of task *a*
  - start(*a*)
- Precedence constraint: *a* before *b*
  - start(*a*) + dur(*a*) ≤ start(*b*)
- Resource constraint:
  - start(*a*) + dur(*a*) ≤ start(*b*)
  - or
  - start(*b*) + dur(*b*) ≤ start(*a*)

Reified Constraints

- Use control variable $b \in \{0,1\}$
  - $c \quad \leftrightarrow \quad b=1$
- Propagate
  - c holds $\Rightarrow$ propagate $b=1$
  - $\neg c$ holds $\Rightarrow$ propagate $b=0$
  - $b=1$ holds $\Rightarrow$ propagate c
  - $b=0$ holds $\Rightarrow$ propagate $\neg c$

Reified Constraints

- Use control variable $b \in \{0,1\}$
  - $c \quad \leftrightarrow \quad b=1$
- Propagate
  - c holds $\Rightarrow$ propagate $b=1$
  - $\neg c$ holds $\Rightarrow$ propaga
  - $b=1$ holds $\Rightarrow$ propagate
  - $b=0$ holds $\Rightarrow$ propagate $\neg c$

not easy!

Reification for Disjunction

- Reify each precedence
  - [start(*a*) + dur(*a*) ≤ start(*b*)] $\leftrightarrow b_0=1$
  - and
  - [start(*b*) + dur(*b*) ≤ start(*a*)] $\leftrightarrow b_1=1$
- Model disjunction
  - $b_0 + b_1 \geq 1$

Model Is Too Naïve

- Local view
  - individual task pairs
  - $O(n^2)$ propagators for n tasks
- Global view
  - all tasks on resource
  - single propagator
  - smarter algorithms possible

Edge Finding

- Find ordering among tasks ("edges")
- For each subset of tasks $\{a\}\cup B$
  - assume: a before B
    deduce information for a and B
  - assume: B before a
    deduce information for a and B
  - join computed information
  - can be done in $O(n^2)$

Summary

- Modeling
  - easy but not always efficient
  - constraint combinators (reification)
  - global constraints
  - smart heuristics
- More on constraint-based scheduling
  Baptiste, Le Pape, Nuijten. Constraint-based Scheduling, Kluwer, 2001.

Instruction Scheduling

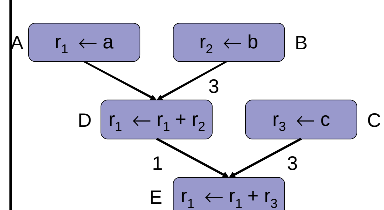
Instruction Scheduling

- Optimized object code by compiler
- Minimum length instruction schedule
  - precedence
  - latency
  - resources

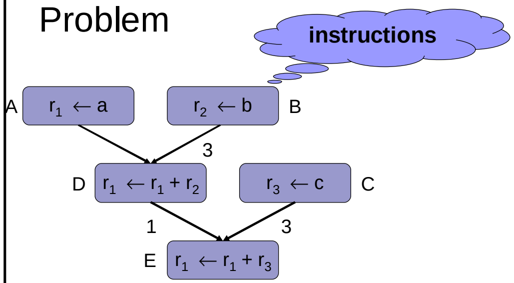
per basic block

Best paper CP 2001, Peter van Beek and Kent Wilken, Fast Optimal Scheduling for Single-issue Processors with Arbitrary Latencies, 2001.

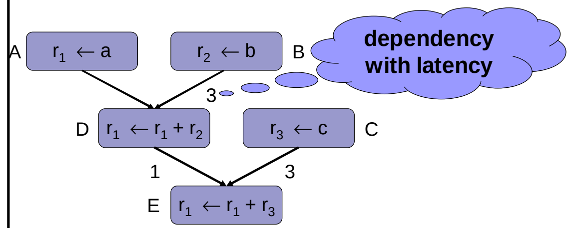
Problem



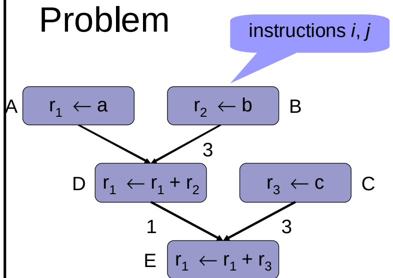
Problem



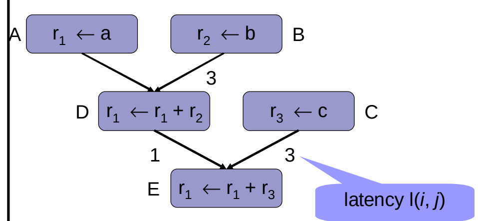
Problem



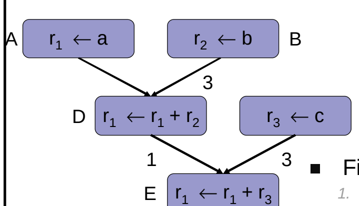
Problem



Problem

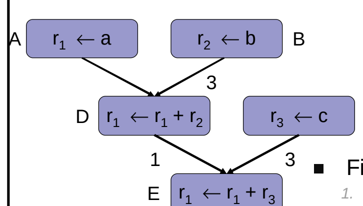


Problem



- Find issue time s(*i*)
  1. $i \neq j \Rightarrow s(i) \neq s(j)$
  2. $s(i)+l(i,j) \leq s(j)$
- Minimize
  max s(*i*)

Problem Is also Model



- Find issue time s(*i*)
  1. $i \neq j \Rightarrow s(i) \neq s(j)$
  2. $s(i)+l(i,j) \leq s(j)$
- Minimize
  max s(*i*)

Model

- All issue times must be distinct
  - use single `distinct` constraint (as in SMM)
  - is resource constraint or unit duration
- Latency constraints
  - precedence constraints (as before)
  - difference: duration ⇆ latency

Making It Work

- Only propagate bounds information
  - relevant for `distinct`
- Add redundant constraints
  - regions: additional structure in DAG
  - successor and predecessor constraints
    [special case of edge-finding]

Results

- Tested with gcc
- SPEC95 FP
- Large basic blocks
  - up to 1000 instructions
- Optimally solved
  - less than 0.6% compile time increase
  - limited static improvement (< 0.1%)
  - better dynamic impact (loops)
- Far better than ILP approach

2G1515

Course Overview 2G1515

- As to be expected, no surprises:
  - applications
  - principles
  - pragmatics
  - limitations

Organization

- 12 lectures
- Assignments (practical + principles)
- Exam
- Material
  - to be developed as we go…
  - with some textbook excerpts + research papers

More Info

- Check my webpage frequently…

  `www.imit.kth.se/~schulte/`