Constraint Programming: An Oz Perspective

A PRICAI'98 Tutorial

Martin Henz

Overview

Part I: Getting Started

Part II: Constraint Propagation

Part III: Tree Search

Part IV: Constraint Programming Techniques

Part V: Programming Search Engines

Overview Part I: Getting Started

- Combinatorial Problems
- Modeling
- Solvers
- Propagate & Distribute
- Degrees of Freedom

3

Overview Part I: Getting Started

- Combinatorial Problems
- Modeling
- Solvers
- Propagate & Distribute
- Degrees of Freedom

4

Combinatorial Problems

- Discrete assignment problems
- numerical and symbolic constraints

- Combinatorial Search Problem:
  Find an assignment
- Combinatorial Optimization Problem:
  Find an *optimal* assignment

5

Examples

- SAT
- Timetabling
- production scheduling
- puzzle solving

6

SAT

- Given: Set of variables $V = \{v_1, \ldots, v_n\}$
  Set of clauses C over V
- Wanted: Assignment $V \rightarrow \{T, F\}$
  such that every clause in C holds.

- Example: $V = \{v_1, v_2\}$
  $C = \{\overline{v_1} \vee v_2, v_1 \vee \overline{v_2}, v_1 \vee v_2\}$
- Solution: $v_1 \mapsto T, v_2 \mapsto T,$

Timetabling

- Given: Courses: CS101, ..., CS545
  Instructors: Dr. Ang, ..., Dr. Zhao
  Students: Ms Aw, ..., Mr. Zhang
  Hours: Monday 9-10, ..., Friday 16-17
  Instructor Assignment: Courses $\rightarrow$ Instructors
  Student Selection: Students $\rightarrow 2^{\text{Courses}}$
  Constraints: overlap constraints,
  idiosyncratic constraints
- Wanted: Course Timetable: Courses $\rightarrow$ Hours

Scheduling

- Given: Tasks: T_1,..., T_n
  Duration: Tasks →
  Machines: M_1,..., M_m
  Constraints: precedence constraints,
  capacity constraints
- Wanted: Schedule: Task → , *minimize* overall duration

9

Puzzle Solving: Example

SEND + MORE = MONEY

Assign distinct digits to the letters

S, E, N, D, M, O, R, Y

such that

```
    S E N D
+   M O R E
-----------
= M O N E Y
```

holds.

10

SEND + MORE = MONEY

Assign distinct digits to the letters

S, E, N, D, M, O, R, Y

such that

```
  S E N D
+ M O R E
---------
= M O N E Y
```

Solution

```
  9 5 6 7
+ 1 0 8 5
---------
= 1 0 6 5 2
```

holds.

Overview Part I: Getting Started

- Combinatorial Problems
- Modeling
- Solvers
- Propagate & Distribute

Modeling

Formalize the problem as a *Constraint Problem*:

- number of variables: n
- constraints: $c_1, \ldots, c_m \subseteq \mathbb{Z}^n$
- problem: Find $a = (v_1, \ldots, v_n) \in \mathbb{Z}^n$ such that $a \in c_i$, for all $1 \leq i \leq m$ (minimize $cost(a)$)

A Model for **MONEY**

- number of variables: 8
- constraints:

$$c_1 = \{(S,E,N,D,M,O,R,Y) \in \mathbb{Z}^8 \mid 0 \leq S,\ldots,Y \leq 9\}$$

$$c_2 = \{(S,E,N,D,M,O,R,Y) \in \mathbb{Z}^8 \mid$$
$$1000*S + 100*E + 10*N + D$$
$$+ 1000*M + 100*O + 10*R + E$$
$$= 10000*M + 1000*O + 100*N + 10*E + Y\}$$

A Model for **MONEY** (continued)

- more constraints

$c_3 = \{(S,E,N,D,M,O,R,Y) \in \quad ^8 \mid S \neq 0 \}$
$c_4 = \{(S,E,N,D,M,O,R,Y) \in \quad ^8 \mid M \neq 0 \}$
$c_5 = \{(S,E,N,D,M,O,R,Y) \in \quad ^8 \mid S \ldots Y$ all different$\}$

Solution for **MONEY**

$c_1 = \{(S,E,N,D,M,O,R,Y) \in \quad ^8 \mid \mathbf{0 \leq S,\ldots,Y \leq 9} \}$
$c_2 = \{(S,E,N,D,M,O,R,Y) \in \quad ^8 \mid$
$\mathbf{1000*S + 100*E + 10*N + D}$
$\mathbf{+ 1000*M + 100*O + 10*R + E}$
$\mathbf{= 10000*M + 1000*O + 100*N + 10*E + Y}\}$
$c_3 = \{(S,E,N,D,M,O,R,Y) \in \quad ^8 \mid \mathbf{S \neq 0} \}$
$c_4 = \{(S,E,N,D,M,O,R,Y) \in \quad ^8 \mid \mathbf{M \neq 0} \}$
$c_5 = \{(S,E,N,D,M,O,R,Y) \in \quad ^8 \mid \mathbf{S \ldots Y}$ **all different**$\}$

Solution: $\mathbf{(9,5,6,7,1,0,8,2) \in \quad ^8}$

A Different Model for **MONEY**: Idea

```
      S E N D
+     M O R E
+   C3 C2 C1
-------------
= M O N E Y
```

17

A Different Model for **MONEY**: Idea

```
     S        E        N        D
     +        +        +        +
     M        O        R        E
     +        +        +
     C3       C2       C1
     =        =        =        =
  10*M    10*C3    10*C2    10*C1
     +        +        +        +
     O        N        E        Y
```

18

A Different Model for **MONEY**

- number of variables: 11
- constraints

$c_1 = \{(S,E,N,D,M,O,R,Y,C1,C2,C3) \in \mathbb{Z}^{11} \mid 0 \le S,\ldots,Y \le 9\}$

$c_2 = \{(S,E,N,D,M,O,R,Y,C1,C2,C3) \in \mathbb{Z}^{11} \mid$
$S + M + C3 = 10*M + O \}$

$c_3 = \{ \ldots \mid E + O + C2 = 10*C3 + N \}$

$c_4 = \{ \ldots \mid N + R + C1 = 10*C2 + E \}$

$c_5 = \{ \ldots \mid D + E = 10*C1 + Y \}$

A Different Model for **MONEY** (continued)

- more constraints

$c_6 = \{(S,E,N,D,M,O,R,Y,C1,C2,C3) \in \mathbb{Z}^{11} \mid S \neq 0 \}$

$c_7 = \{(S,E,N,D,M,O,R,Y,C1,C2,C3) \in \mathbb{Z}^{11} \mid M \neq 0 \}$

$c_8 = \{(S,E,N,D,M,O,R,Y,C1,C2,C3) \in \mathbb{Z}^{11} \mid S \ldots Y$ all different $\}$

Variation: Combinatorial Optimization

SEND + MOST = MONEY

Assign distinct digits to the letters

S, E, N, D, M, O, S, T, Y

such that

```
  S E N D
+   M O S T
---------
= M O N E Y
```

holds and M O N E Y is maximized.

Overview Part I: Getting Started

- Combinatorial Problems
- Modeling
- Solvers
- Propagate & Distribute

SAT

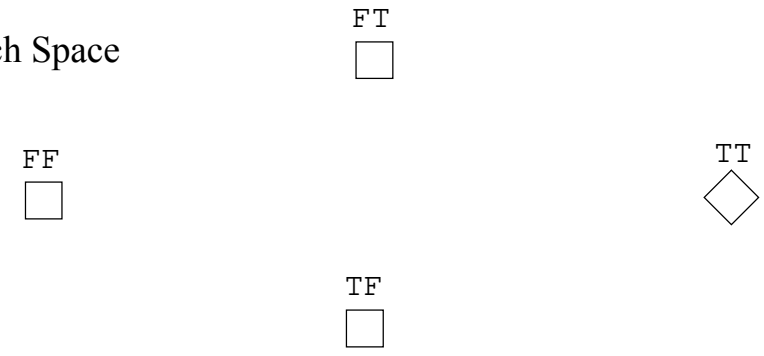
- Given: Set of variables $V = \{v_1, \ldots, v_n\}$
  Set of clauses C over V
- Wanted: Assignment $V \rightarrow \{T, F\}$
  such that every clause in C holds.

- Example: $V = \{v_1, v_2\}$
  $C = \{\overline{v_1} \vee v_2,\ v_1 \vee \overline{v_2},\ v_1 \vee v_2\}$
- Solution: $v_1 \mapsto T,\ v_2 \mapsto T$

Local Search

$V = \{v_1, v_2\}$
$C = \{\overline{v_1} \vee v_2, v_1 \vee \overline{v_2}, v_1 \vee v_2\}$

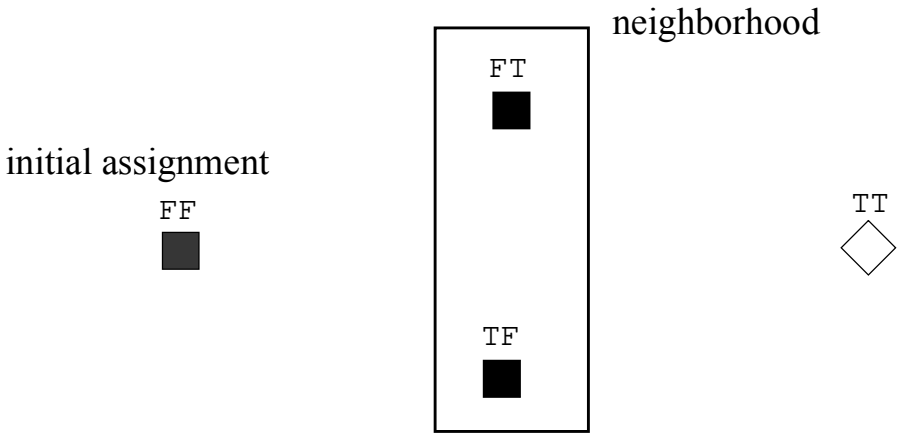
Search Space



Local Search

$V = \{v_1, v_2\}$

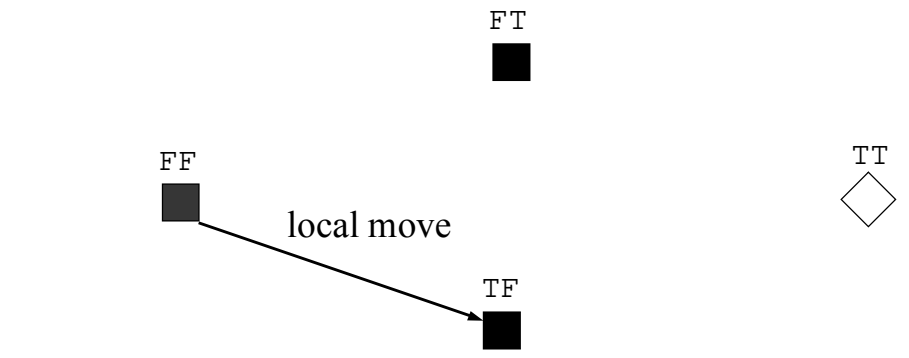
$C = \{\overline{v_1} \vee v_2, v_1 \vee \overline{v_2}, v_1 \vee v_2\}$



Local Search

$V = \{v_1, v_2\}$

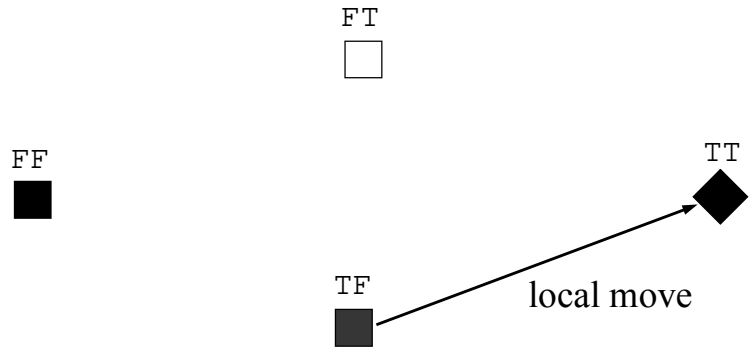
$C = \{\overline{v_1} \vee v_2, v_1 \vee \overline{v_2}, v_1 \vee v_2\}$



Local Search

$V = \{v_1, v_2\}$

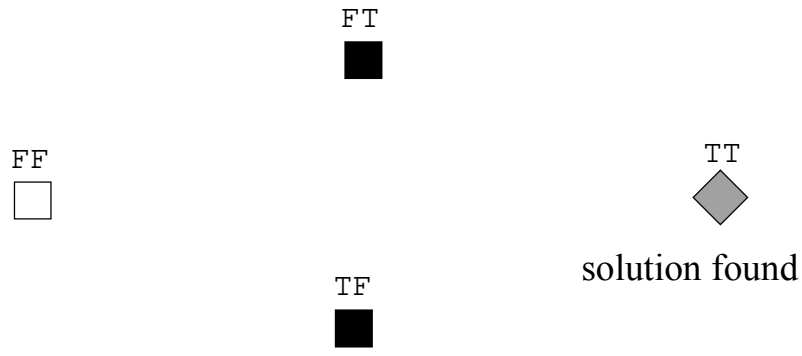
$C = \{\overline{v_1} \vee v_2, v_1 \vee \overline{v_2}, v_1 \vee v_2\}$



Local Search

$V = \{v_1, v_2\}$

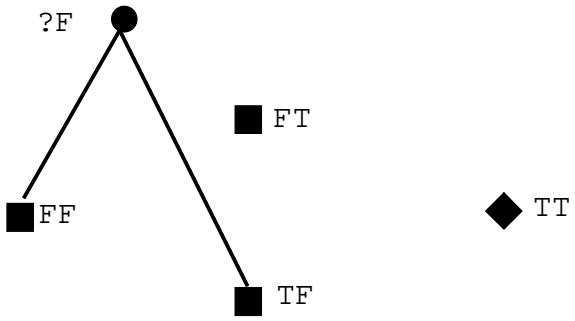
$C = \{\overline{v_1} \vee v_2, v_1 \vee \overline{v_2}, v_1 \vee v_2\}$



Global Search

$V = \{v_1, v_2\}$

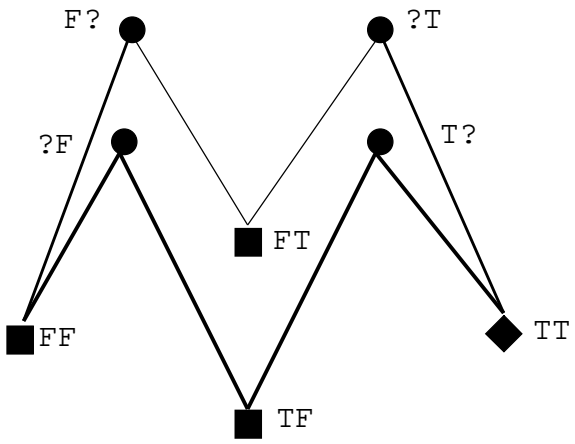
$C = \{\overline{v_1} \vee v_2, v_1 \vee \overline{v_2}, v_1 \vee v_2\}$



Global Search

$V = \{v_1, v_2\}$

$C = \{\overline{v_1} \vee v_2, v_1 \vee \overline{v_2}, v_1 \vee v_2\}$

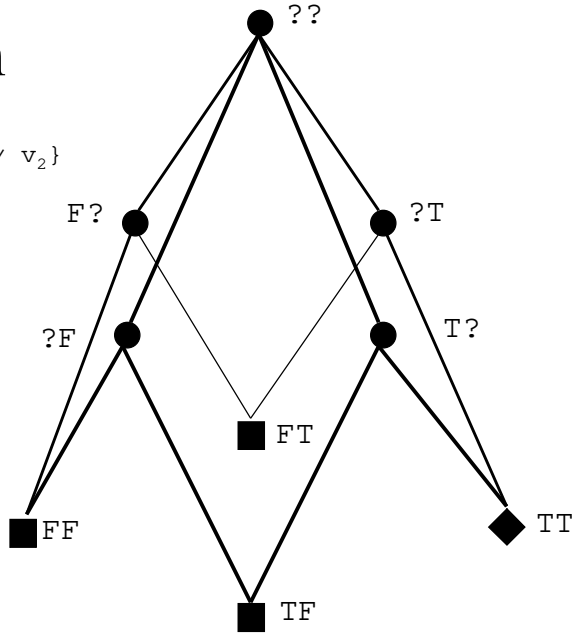


Global Search

$V = \{v_1, v_2\}$

$C = \{\overline{v_1} \vee v_2, v_1 \vee \overline{v_2}, v_1 \vee v_2\}$

Search Graph



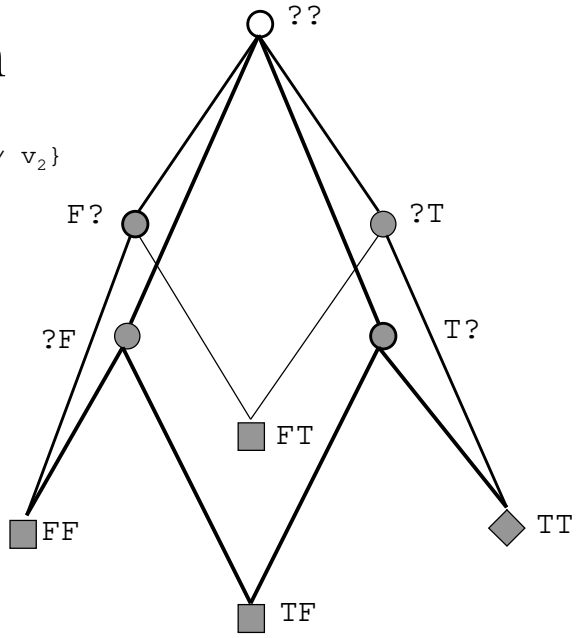
Search Graph

- Information increases down the graph
- graph contains all solutions (by construction)
- graph not unique (here variable fixing is used)

Global Search

$V = \{v_1, v_2\}$

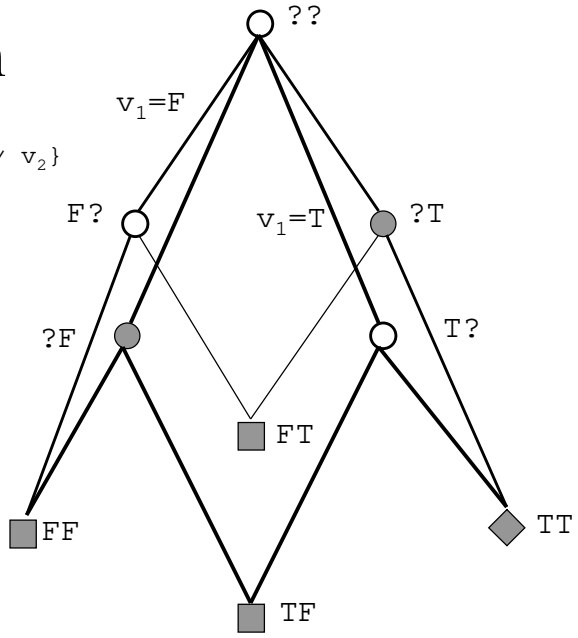
$C = \{\overline{v_1} \vee v_2, v_1 \vee \overline{v_2}, v_1 \vee v_2\}$



Global Search

$V = \{v_1, v_2\}$

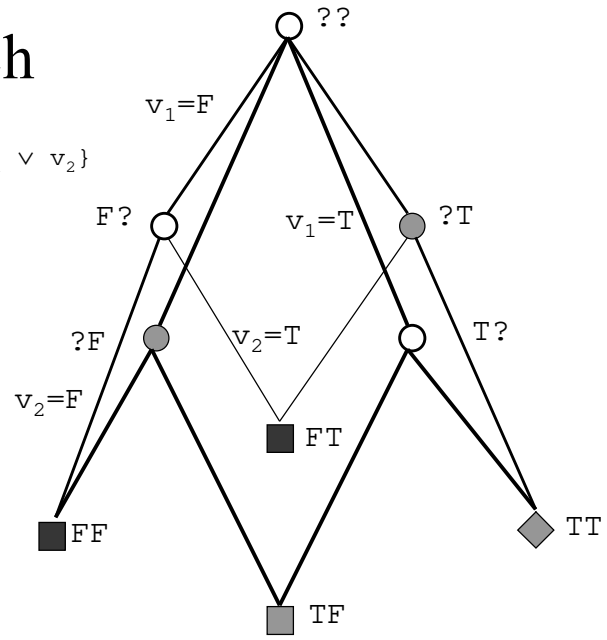
$C = \{\overline{v_1} \vee v_2, v_1 \vee \overline{v_2}, v_1 \vee v_2\}$



Global Search

$V = \{v_1, v_2\}$

$C = \{\overline{v_1} \vee v_2, v_1 \vee \overline{v_2}, v_1 \vee v_2\}$

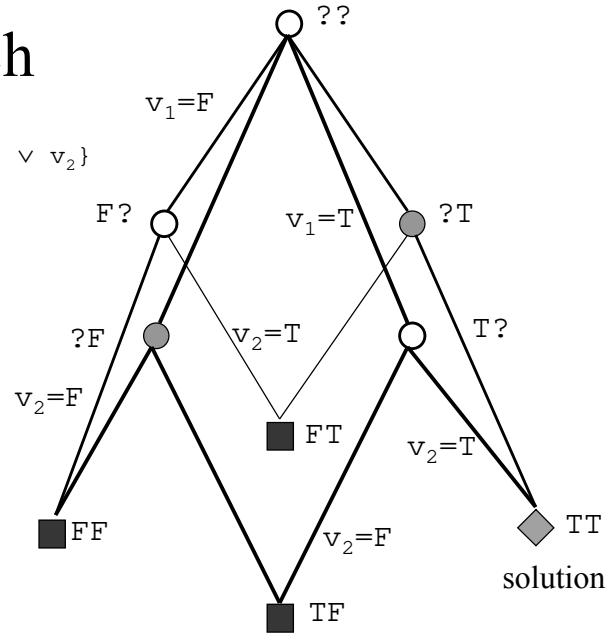


Global Search

$V = \{v_1, v_2\}$

$C = \{\overline{v_1} \vee v_2, v_1 \vee \overline{v_2}, v_1 \vee v_2\}$

Search Tree



Search Tree

- Subgraph of search graph
- complete if it contains all solutions
- tree not unique (here determined by first trying v_1 and then v_2)

37

Exploration of Search Tree

- Top-down
- complete if every solution is reached
- exploration not unique (which value to try first? depth-first or breadth-first?...)

38

Overview Part I: Getting Started

- Combinatorial Problems
- Modeling
- Solvers
- Propagate & Distribute

39

Constraint Programming

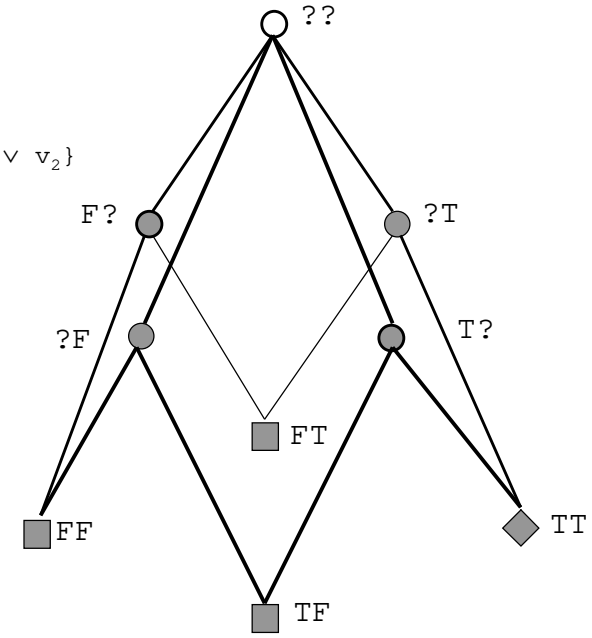
Exploiting constraints during tree search

- Domain restriction
- pruning
- constraint-based distribution
- constraint-based optimization
- constraint-based exploration

40

$V = \{v_1, v_2\}$

$C = \{\overline{v_1} \vee v_2, v_1 \vee \overline{v_2}, v_1 \vee v_2\}$

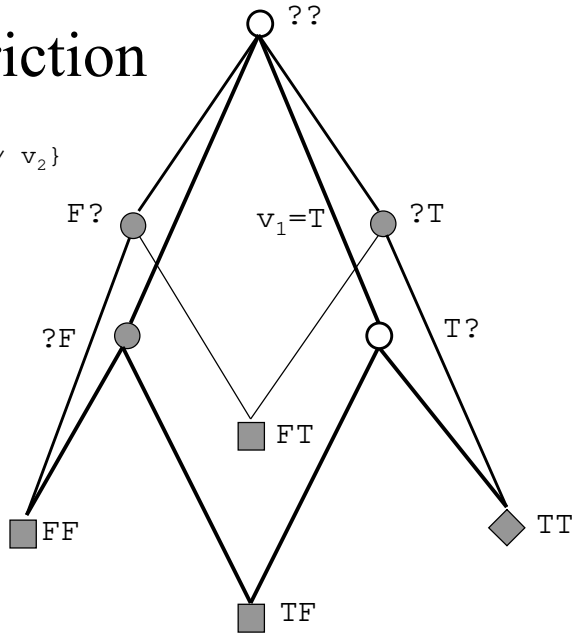


41

Domain Restriction

$V = \{v_1, v_2\}$

$C = \{\overline{v_1} \vee v_2, v_1 \vee \overline{v_2}, v_1 \vee v_2\}$

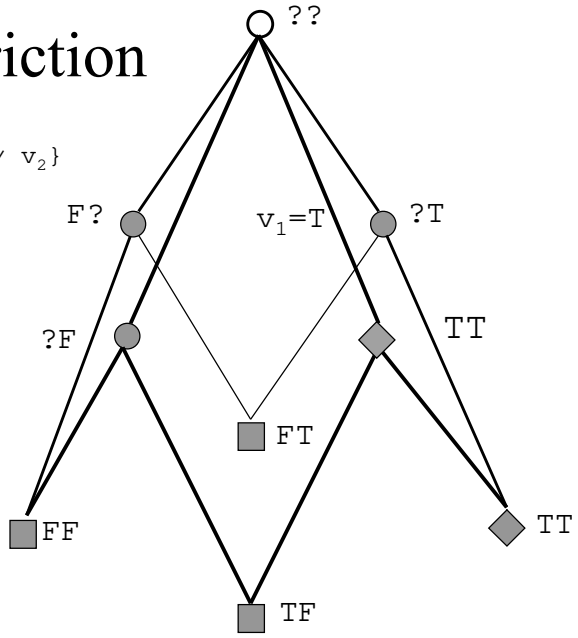


42

Domain Restriction

$V = \{v_1, v_2\}$

$C = \{\overline{v_1} \vee v_2, v_1 \vee \overline{v_2}, v_1 \vee v_2\}$

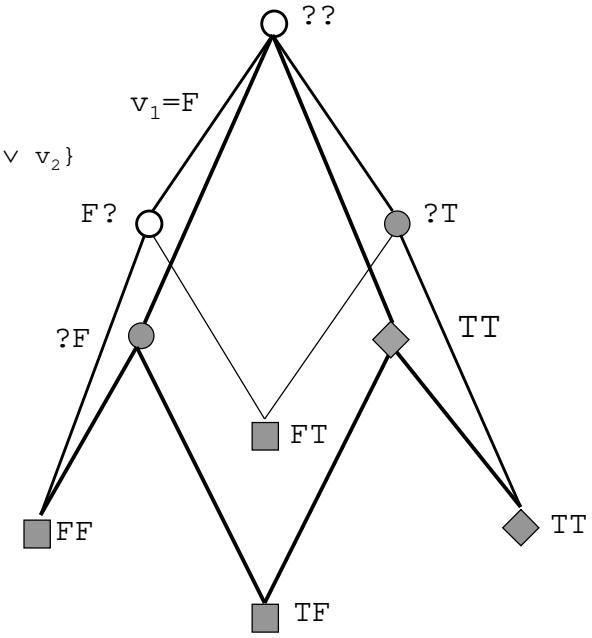


43

Pruning

$V = \{v_1, v_2\}$

$C = \{\overline{v_1} \vee v_2, v_1 \vee \overline{v_2}, v_1 \vee v_2\}$

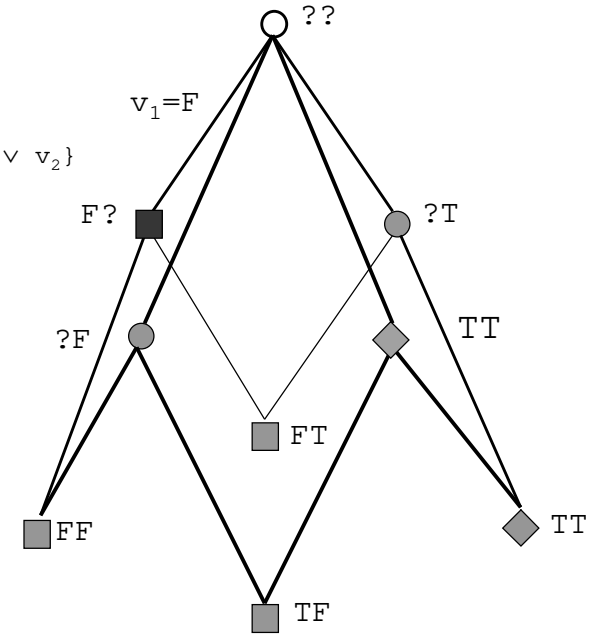


44

Pruning

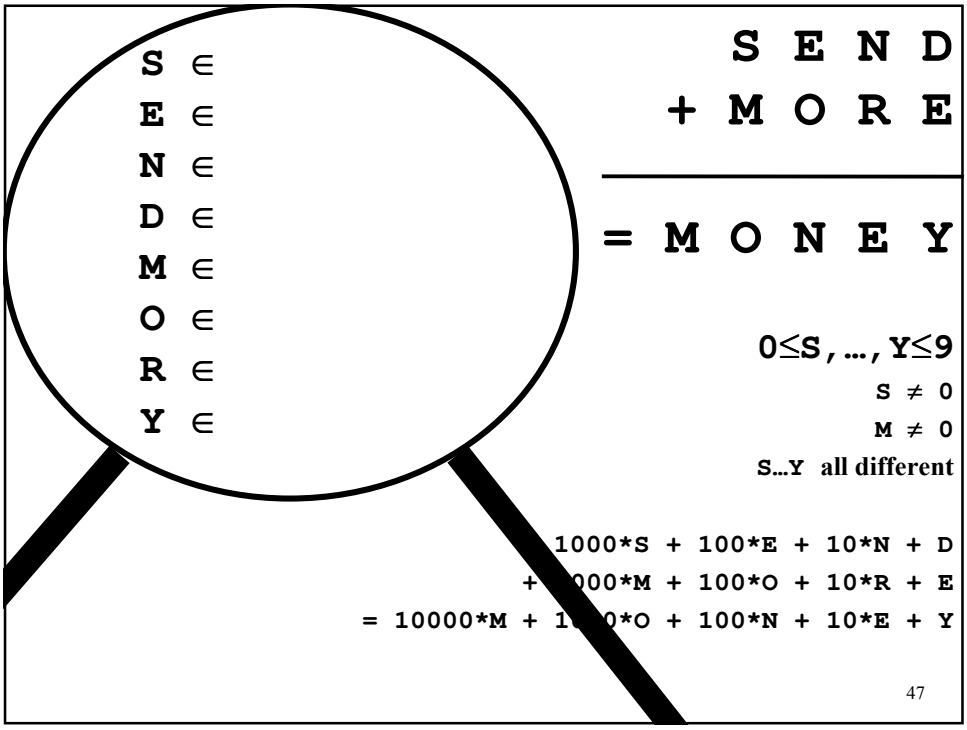
$V = \{v_1, v_2\}$

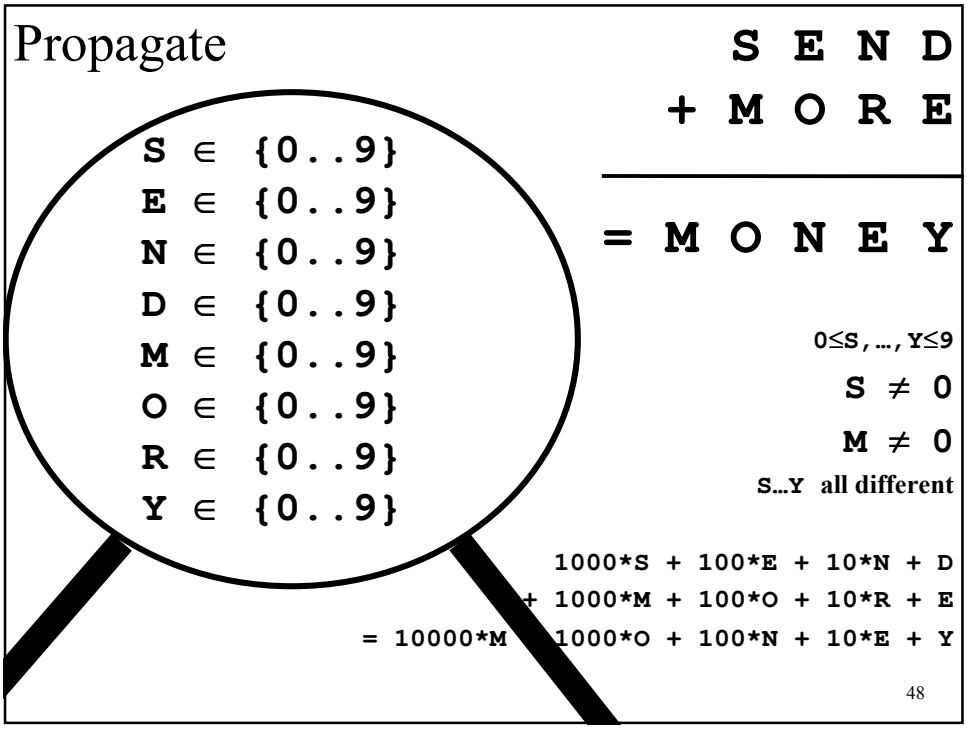
$C = \{\overline{v_1} \vee v_2, v_1 \vee \overline{v_2}, v_1 \vee v_2\}$



The Constraint Programming Principle

Propagate and *then* Distribute


S ∈
E ∈
N ∈
D ∈
M ∈
O ∈
R ∈
Y ∈
S E N D
+ M O R E
= M O N E Y
0≤S,…,Y≤9
S ≠ 0
M ≠ 0
S…Y all different
1000*S + 100*E + 10*N + D
+ 000*M + 100*O + 10*R + E
= 10000*M + 1 0*O + 100*N + 10*E + Y
47


Propagate
S ∈ {0..9}
E ∈ {0..9}
N ∈ {0..9}
D ∈ {0..9}
M ∈ {0..9}
O ∈ {0..9}
R ∈ {0..9}
Y ∈ {0..9}
S E N D
+ M O R E
= M O N E Y
0≤S,…,Y≤9
S ≠ 0
M ≠ 0
S…Y all different
1000*S + 100*E + 10*N + D
+ 1000*M + 100*O + 10*R + E
= 10000*M 1000*O + 100*N + 10*E + Y
48

Propagate

```
  S E N D
+ M O R E
---------
= M O N E Y
```

$S \in \{1..9\}$
$E \in \{0..9\}$
$N \in \{0..9\}$
$D \in \{0..9\}$
$M \in \{1..9\}$
$O \in \{0..9\}$
$R \in \{0..9\}$
$Y \in \{0..9\}$

$0 \le S, \ldots, Y \le 9$
$S \neq 0$
$M \neq 0$
S…Y all different

```
  1000*S + 100*E + 10*N + D
+ 1000*M + 100*O + 10*R + E
= 10000*M + 1000*O + 100*N + 10*E + Y
```


Propagate

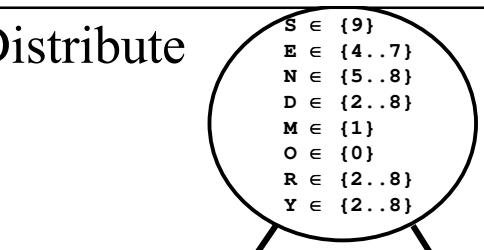
```
  S E N D
+ M O R E
---------
= M O N E Y
```

$S \in \{9\}$
$E \in \{4..7\}$
$N \in \{5..8\}$
$D \in \{2..8\}$
$M \in \{1\}$
$O \in \{0\}$
$R \in \{2..8\}$
$Y \in \{2..8\}$

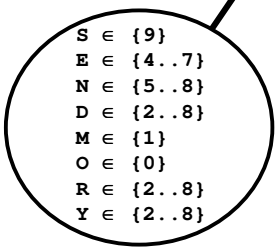
$0 \le S, \ldots, Y \le 9$
$S \neq 0$
$M \neq 0$
S…Y all different

```
  1000*S + 100*E + 10*N + D
+ 1000*M + 100*O + 10*R + E
= 10000*M + 1000*O + 100*N + 10*E + Y
```

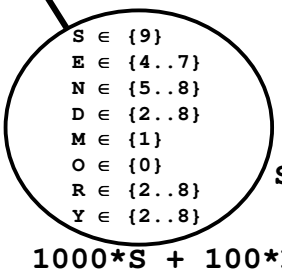
Distribute



E = 4



E ≠ 4




```
  S E N D
+ M O R E
---------
= M O N E Y
```

$0 \le S, \ldots, Y \le 9$

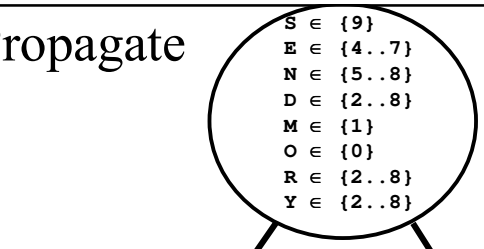
$S \neq 0$

$M \neq 0$

S…Y all different

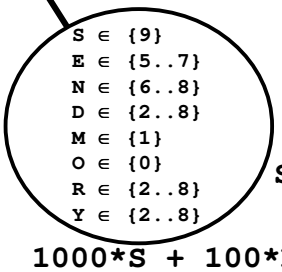
$$1000*S + 100*E + 10*N + D$$
$$+ 1000*M + 100*O + 10*R + E$$
$$= 10000*M + 1000*O + 100*N + 10*E + Y$$

Propagate



E = 4

E ≠ 4




```
  S E N D
+ M O R E
---------
= M O N E Y
```

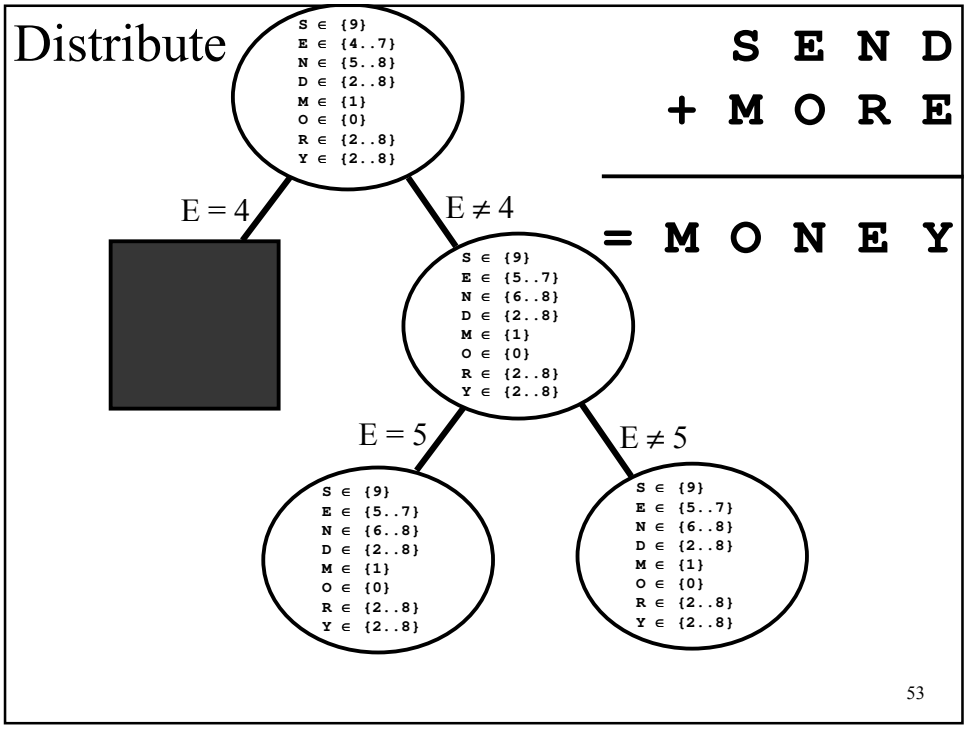
$0 \le S, \ldots, Y \le 9$

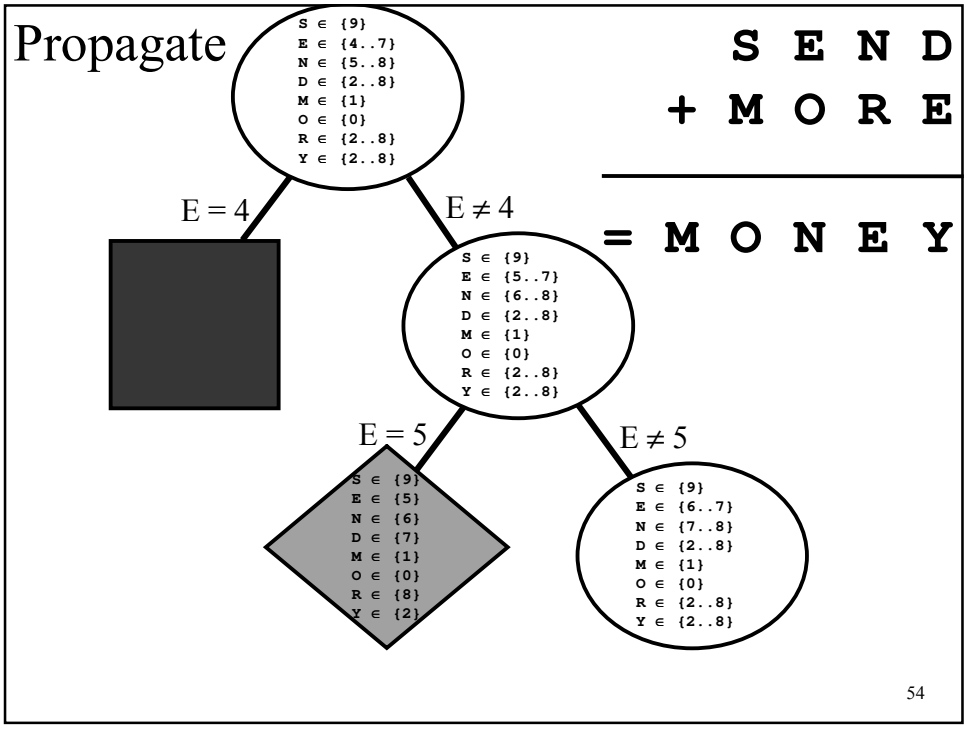
$S \neq 0$

$M \neq 0$

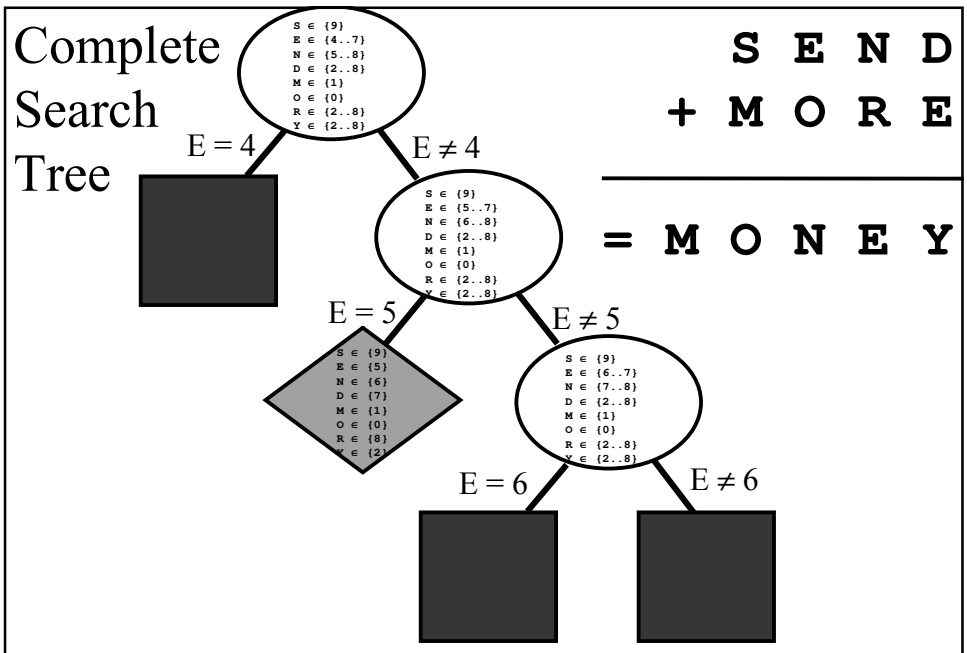
S…Y all different

$$1000*S + 100*E + 10*N + D$$
$$+ 1000*M + 100*O + 10*R + E$$
$$= 10000*M + 1000*O + 100*N + 10*E + Y$$


Distribute
S ∈ {9}
E ∈ {4..7}
N ∈ {5..8}
D ∈ {2..8}
M ∈ {1}
O ∈ {0}
R ∈ {2..8}
Y ∈ {2..8}
S E N D
+ M O R E
= M O N E Y
E = 4
E ≠ 4
S ∈ {9}
E ∈ {5..7}
N ∈ {6..8}
D ∈ {2..8}
M ∈ {1}
O ∈ {0}
R ∈ {2..8}
Y ∈ {2..8}
E = 5
E ≠ 5
S ∈ {9}
E ∈ {5..7}
N ∈ {6..8}
D ∈ {2..8}
M ∈ {1}
O ∈ {0}
R ∈ {2..8}
Y ∈ {2..8}
S ∈ {9}
E ∈ {5..7}
N ∈ {6..8}
D ∈ {2..8}
M ∈ {1}
O ∈ {0}
R ∈ {2..8}
Y ∈ {2..8}
53


Propagate
S ∈ {9}
E ∈ {4..7}
N ∈ {5..8}
D ∈ {2..8}
M ∈ {1}
O ∈ {0}
R ∈ {2..8}
Y ∈ {2..8}
S E N D
+ M O R E
= M O N E Y
E = 4
E ≠ 4
S ∈ {9}
E ∈ {5..7}
N ∈ {6..8}
D ∈ {2..8}
M ∈ {1}
O ∈ {0}
R ∈ {2..8}
Y ∈ {2..8}
E = 5
E ≠ 5
S ∈ {9}
E ∈ {5}
N ∈ {6}
D ∈ {7}
M ∈ {1}
O ∈ {0}
R ∈ {8}
Y ∈ {2}
S ∈ {9}
E ∈ {6..7}
N ∈ {7..8}
D ∈ {2..8}
M ∈ {1}
O ∈ {0}
R ∈ {2..8}
Y ∈ {2..8}
54

Complete Search Tree




```
  S E N D
+ M O R E
---------
= M O N E Y
```


Where Do We Go From Here?

- Demos
  - SAT
  - `MONEY`
  - Timetabling
  - Scheduling
- Constraint Programming Systems

Part II: Constraint Propagation

Part III: Tree Search

Part IV: Constraint Programming Techniques

Part V: Programming Search Engines

Demos

Starting Demo

57

Constraint Programming Systems

support constraint programming with high-level constructs

- Constraint Programming Library
  - ILOG Solver (Puget 1993)
- Constraint Programming Languages
  - CONSTRAINTS (Steele, Sussman 1980)
  - CHIP (Dincbas, Hentenryck, Simonis, Aggoun 1988)
  - CLP(R) (Jaffar, Maher, Stuckey, Yap 1992)
  - Oz (Smolka and others 1995)
  - OPL (Hentenryck 1998)

58

Constraint Programming Language Oz

- Functional programming
  - first-class procedures
  - symbolic computation
- concurrent programming
  "constraints are concurrent"
- built-in constraint systems
  - integers ("finite domains")
  - sets of integers
  - feature trees
- built-in computation spaces

59

Programming System DFKI Oz

- Tools for constraint programming
  - Oz Explorer
  - Oz Browser
- constraint programming libraries (including scheduling)
- constraint programming interface (C++)
- efficient implementation (Unix, Windows9x/NT)
- public domain
- full documentation
- many existing public-domain application prototypes

60

Further Reading

- Programming with Constraints: Marriott, Stuckey, MIT Press, 1998
- DFKI Oz documentation:
  `http://www.ps.uni-sb.de/oz/documentation`
- Oz papers: `http://www.ps.uni-sb.de/Papers`
- An Exercise in Concurrent Object-Oriented Programming: Oz Browser, WOz'95
- Oz Explorer: A Visual Constraint Programming Tool, ICLP'97
- Objects for Concurrent Constraint Programming, Martin Henz, Kluwer Academic Publishers, 1998

Overview

Part I: Getting Started

→ Part II: Constraint Propagation

Part III: Tree Search

Part IV: Constraint Programming Techniques

Part V: Programming Search Engines

Overview Part II: Constraint Propagation

- Issues
- Basic Constraints versus Propagators
- Domain versus Interval Consistency
- Propagator Classes
- Case Study: Domain-Consistent Sum

63

Overview Part II: Constraint Propagation

- Issues
- Basic Constraints versus Propagators
- Domain versus Interval Consistency
- Propagator Classes
- Case Study: Domain-Consistent Sum

64

Issues

- Expressivity: What kind of information can be expressed as propagators?
- Completeness: What behavior can be expected from propagation?
- Efficiency: How much computational resources does propagation consume?

Completeness

- General arithmetic constraints are undecidable (Hilbert's Tenth Problem).
- Propagation may not exhibit all inconsistencies.
- Example:

  c_1: $n > 2$

  c_2: $a^n + b^n = c^n$

Overview Part II: Constraint Propagation

- Issues
- Basic Constraints versus Propagators
- Domain versus Interval Consistency
- Propagator Classes
- Case Study: Domain-Consistent Sum

Basic Constraints vs. Propagators

- Basic constraints
  - are conjunctions of constraints of the form
    - `X = Y`
    - `X ∈ S`, where `S` is a finite set of integers
  - enjoy complete constraint solving
- Propagators
  - can be arbitrarily expressive (arithmetic, symbolic)
  - implementation typically fast but incomplete

Current Domain

Let C be basic constraint. For a given variable X, the maximal set S such that $C \wedge X \in S$ is consistent is called the *current domain of* X *in* C, denoted $dom_C(X)$.

Example:

C: X∈{1..10}, X=Y, Y∈{9..20}

$dom_C(X) = dom_C(Y) = \{9,10\}$

69

Example Session: Basic Constraints vs. Propagators

Basic Demo

70

Overview Part II: Constraint Propagation

- Issues
- Basic Constraints vs. Propagators
- Domain vs. Interval Consistency
- Propagator Classes
- Case Study: Domain-Consistent Sum

Domain vs. Interval Consistency

- Domain Consistency: Check full domains of all variables known to the propagator
- Interval Consistency: Check only the boundaries of the domains of the variables

Domain Consistency

Assume a constraint `d` over variables `X` and `Y`.
A basic constraint `C` is *domain consistent in* `X` *with respect to* `d`, if for every $i \in dom_C(X)$ there is a $j \in dom_C(Y)$ such that $(i,j) \in d$.

Example:

```
d: X + Y = 10
C: X∈{7,9}, Y∈{1..20}
```

73

Domain Consistency

Assume a constraint `d` over variables `X` and `Y`.
A basic constraint `C` is *domain consistent in* `X` *with respect to* `d`, if for every $i \in dom_C(X)$ there is a $j \in dom_C(Y)$ such that $(i,j) \in d$.

Example:

```
d: X + Y = 10
C: X∈{7,9}, Y∈{1,3}
```

74

Interval Consistency

Assume a constraint `d` over variables `X` and `Y`.
A basic constraint `C` is *interval consistent in* `X` *with respect to* `d`, if for `i=min(dom`$_C$`(X))` and for `i=max(dom`$_C$`(X))` there is a `j`$\in$`dom`$_C$`(Y)` such that `(i,j)`$\in$`d`.

Example:

```
d: X + Y = 10
C: X∈{7,9}, Y∈{1..20}
```


Interval Consistency

Assume a constraint `d` over variables `X` and `Y`.
A basic constraint `C` is *interval consistent in* `X` *with respect to* `d`, if for `i=min(dom`$_C$`(X))` and for `i=max(dom`$_C$`(X))` there is a `j`$\in$`dom`$_C$`(Y)` such that `(i,j)`$\in$`d`.

Example:

```
d: X + Y = 10
C: X∈{7,9}, Y∈{1..3}
```

Example Session: Domain vs. Interval Consistency

Domain Demo

77

Overview Part II: Constraint Propagation

- Issues
- Basic Constraints versus Propagators
- Domain versus Interval Consistency
- Propagator Classes
- Case Study: Domain-Consistent Sum

78

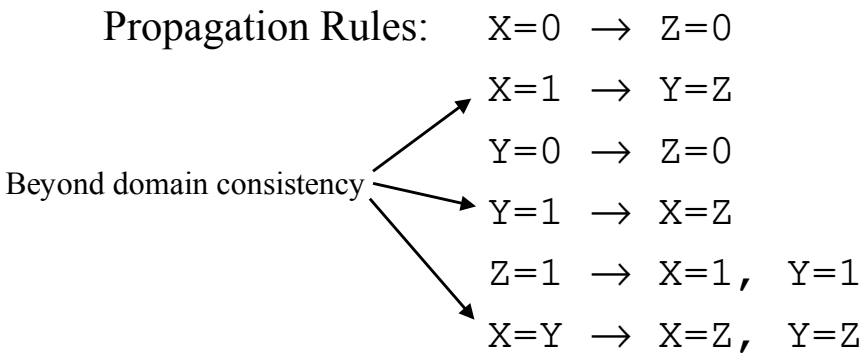
Propagator Classes

- 0/1 Propagators
- Symbolic Propagators
- Arithmetic Propagators
- Scheduling Propagators
- Reification

0/1 Propagators

Example: `{FD.conj X Y Z}`

Propagation Rules:



All 0/1 Propagators

```
{FD.conj X Y Z}
{FD.disj X Y Z}
{FD.exor X Y Z}
{FD.impl X Y Z}
{FD.equi X Y Z}
{FD.nega X Y}
```


Symbolic Propagators

Example: The "Element" Propagator

Call: `{FD.element I L X}`

Intuition: `X` is the `I`th element of `L`

Example: `{FD.element I [5 6 7 8] X}`

$dom_c(I)=\{1,3\} \rightarrow X \in \{5,7\}$

$dom_c(X)=\{6,8\} \rightarrow I \in \{2,4\}$

More Symbolic Propagators

```
{FD.element X L Y}
{FD.distinct Xs}
{FD.distinctOffset Xs Is}
{FD.distinct2 Xs Is Ys Js}
{FD.atMost X Ys I}
    atLeast
    exactly
```


Arithmetic Propagators

General Arithmetic Equations:

$I_1*X_{11}*\ldots*X_{1m_1} + \ldots + I_n*X_{n1}*\ldots*X_{nm_n}$ `=: 0`

`<: >: =<: >=: \=:`

Others:

```
{FD.power X I Y}
{FD.divI X I Y}                modI,divD,modD
{FD.max X Y Z}                 min
{FD.distance X Y '=:' Z}       <: >: =<: >=: \=:
```

Scheduling Propagators

Details in Part IV

Issues:

- resource constraints
- precedence constraints
- edge finding
- task intervals

Reified Constraints

Reflecting the validity of a constraint in a 0/1 variable

Example: `{FD.reified.int [1 2 3 4] X B}`

Intuition: `B` reflects whether $X \in \{1,2,3,4\}$

Propagation:

$B=1 \rightarrow X \in \{1,2,3,4\}$

$B=0 \rightarrow X \notin \{1,2,3,4\}$

$dom(X) \subseteq \{1,2,3,4\} \rightarrow B=1$

$dom(X) \cap \{1,2,3,4\} = \emptyset \rightarrow B=0$

More Reified Constraints

General Arithmetic Equations:

$X = I_1*X_{11}*\ldots*X_{1m_1} + \ldots + I_n*X_{n1}*\ldots*X_{nm_n}$ `=: 0`

`<: >: =<: >=: \=:`

Others:

```
{FD.reified.distance X1 X2 I X3 B}
{FD.reified.card X1 Xs X2 B}
```


Overview Part II: Constraint Propagation

- Issues
- Basic Constraints versus Propagators
- Domain versus Interval Consistency
- Propagator Classes
- Case Study: Domain-Consistent Sum

Case Study: Domain-Consistent Sum

- C++ Propagator Interface
- Implementing Domain-Consistent Sum
- Using Domain-Consistent Sum

Propagator Interface

- Efficiency critical for implementation of propagators
- imperative programming style adequate

Provide a programming interface to implement propagators in C++:

Constraint Propagator Interface (CPI) of DFKI Oz

CPI of DFKI Oz

- Object-oriented structure, C++ classes for
  - propagators
  - variables
  - domains
- All library propagators are available through CPI
  - consistency
  - rich set of examples
- many convenient C++ macros and operators predefined

Domain-Consistent Sum: Idea

```
{MYLIB.add X Y Z}
```

enforces domain-consistency in `X`, `Y`, `Z`.

Implementation: Loop through the domains of `X` and `Y`. Keep only those values in the domains of `X`, `Y`, `Z` for which a consistent assignment exists.

Defining a Class AddProp

```
class AddProp : public OZ_Propagator {
private: OZ_Term _x, _y, _z;
public:
  AddProp(OZ_Term a,OZ_Term b,OZ_Term c)
   : _x(a), _y(b), _z(c) {}
virtual OZ_Return propagate(void);
virtual size_t sizeOf(void);
virtual void updateHeapRefs(OZ_Boolean);
virtual OZ_Term getParameters(void);
};
```


Implementing Propagation for AddProp

```
OZ_Return AddProp::propagate(void)
{
  OZ_FDIntVar x(_x), y(_y), z(_z);
  OZ_FiniteDomain x_aux(fd_empty),
                  y_aux(fd_empty),
                  z_aux(fd_empty);
  // actual propagation code goes here
  if ((*x &= x_aux)==0) goto failure;
  if ((*y &= y_aux)==0) goto failure;
  if ((*z &= z_aux)==0) goto failure;
  return (x.leave() | y.leave() | z.leave())
    ? OZ_SLEEP : OZ_ENTAILED;
  failure: x.fail(); y.fail(); z.fail();
           return OZ_FAILED;
}
```

Actual Propagation Code for `AddProp`

```
for (int i = x->getMinElem(); i != -1;
     i = x->getNextLargerElem(i))
  for (int j = y->getMinElem(); j != -1;
       j = y->getNextLargerElem(j))
    if (z->isIn(i + j)) {
      x_aux += i;
      y_aux += j;
      z_aux += (i + j);
    }
```


Using Domain-Consistent Sum

Sum Demo

Case Study Domain-Consistent Sum: Summary

- New (application-specific) propagators can be implemented using CPI
- implementation language C++ for efficiency
- propagators can be implemented in Oz using reflection primitives (rapid prototyping)

Further Reading

- Foundations of Constraint Satisfaction, Edward Tsang, Academic Press 1993
- The Constraint Propagator Interface of DFKI Oz, Müller, Würtz

  `http://www.ps.uni-sb.de/oz/documentation`

Overview

Part I: Getting Started

Part II: Constraint Propagation

Part III: Tree Search

Part IV: Constraint Programming Techniques

Part V: Programming Search Engines

Overview Part III: Tree Search

- Encapsulating a Constraint Store
- Operations on Spaces
- Choice Points
- Depth-First Search
- Distributors
- Branch-And-Bound
- Case Study: Timetabling

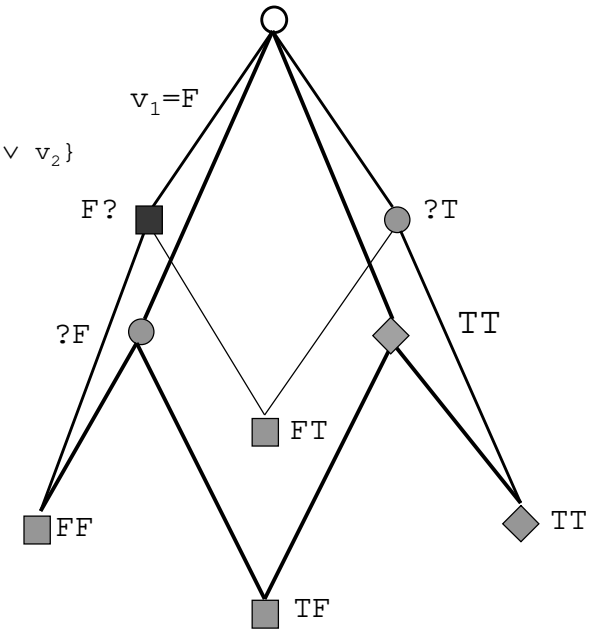
Overview Part III: Tree Search

- Encapsulating a Constraint Store
- Operations on Spaces
- Choice Points
- Depth-First Search
- Distributors
- Branch-And-Bound
- Case Study: Timetabling

Tree Search Demo

$V = \{v_1, v_2\}$

$C = \{\overline{v_1} \vee v_2, v_1 \vee \overline{v_2}, v_1 \vee v_2\}$



Overview Part III: Tree Search

- Encapsulating a Constraint Store
- Operations on Spaces
- Choice Points
- Depth-First Search
- Distributors
- Branch-And-Bound
- Case Study: Timetabling

Timetabling: Problem

- Given: Courses: CS101, ..., CS545
  Instructors: Dr. Ang, ..., Dr. Zhao
  Students: Ms Aw, ..., Mr. Zhang
  Hours: Monday 9-10, ..., Friday 16-17
  Instructor Assignment: Courses → Instructors
  Student Selection: Students → 2^{Courses}
  Constraints: overlap constraints
  Preferences: Instructors → Hours
- Wanted: Course Timetable: Courses → Hours
  Optimize fulfillment of preferences

Timetabling: Model

- Variables: For each course `C` the variable T_C represents the time slot in which it is given.
- Constraints: Non-overlap constraints for instructor assignment and student selection are represented by `\=:` constraints.
- Preferences: Optimization function is constructed as a sum of reified `\=:` constraints

Timetabling: Implementation

Timetabling Demo

Case Study Timetabling: Summary

- Compiling Data: Often symbolic computation is needed to translate
- Experimentation crucial
- Efficient solution often appears after gaining a thorough understanding of the problem instances.

Further Reading

- Using Oz for College Timetabling, Henz, Würtz, PATAT'96,

  `http://www.comp.nus.edu.sg/~henz/papers`

Overview

Part I: Getting Started

Part II: Constraint Propagation

Part III: Tree Search

Part IV: Constraint Programming Techniques

Part V: Programming Search Engines

Overview Part IV: Constraint Programming Techniques

- Symmetries
- Parameterized Scripts
- Multi-level Distribution
- Redundant Constraints
- User-defined Distributors
- Case Study: Scheduling

Overview Part IV: Constraint Programming Techniques

- Symmetries
- Parameterized Scripts
- Multi-level Distribution
- Redundant Constraints
- User-defined Distributors
- Case Study: Scheduling

Symmetries

Often, the most efficient model admits many different solutions that are essentially the same ("symmetric" to each other).

This technique tries to improve the performance of search by eliminating such symmetries.

Example: Grocery Puzzle

A kid goes into a grocery store and buys four items. The cashier charges $7.11, the kid pays and is about to leave when the cashier calls the kid back, and says "Hold on, I multiplied the four items instead of adding them; I'll try again… Gosh, with adding them the price still comes to $7.11"! What were the prices of the four items?

Model for Grocery Puzzle

- Variables `A`, `B`, `C`, `D` represent prices of items in cents.
- Constraints:
  - `A + B + C + D = 711`
  - `A * B * C * D = 711 * 100 * 100 * 100`

Solution to Grocery Puzzle

Grocery Demo

Performance of Symmetry Breaking

- All Solution Search: Symmetry breaking usually improves performance; often dramatically
- One Solution Search: Symmetry breaking may or may not improve performance

Overview Part IV: Constraint Programming Techniques

- Symmetries
- Parameterized Scripts
- Multi-level Distribution
- Redundant Constraints
- User-defined Distributors
- Case Study: Scheduling

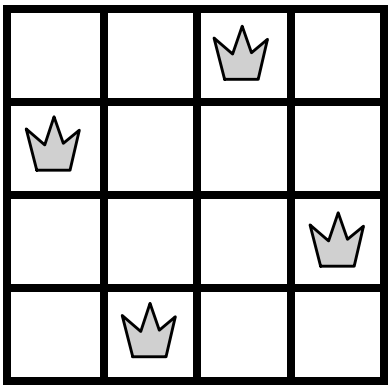
Parameterized Scripts

Often scripts are used multiple times.

This technique uses features of Oz to achieve reusable scripts.

Example: N Queens

Arrange `N` queens on an `N×N` chess board such that no queen can attack any other.



Model for N Queens: Idea

Observe that each column must hold exactly one queen. Introduce a variable R_i for each column `i` that expresses in which row its queen goes.

Model for N Queens

- Variables $R_1 \ldots R_N$
- vertical no-attack: by construction
- horizontal no-attack:

  `{FD.distinct [`$R_1 \ldots R_N$`]}`
- diagonal no-attack
  - `{FD.distinctOffset [`$R_1 \ldots R_N$`] [1...N]}`
  - `{FD.distinctOffset [`$R_1 \ldots R_N$`] [~1...~N]}`

Solution to N Queens

N Queens Demo

Parameterized Scripts

- Need to reuse scripts with different parameters
- Use first-class procedures to produce scripts with desired parameter settings

Overview Part IV: Constraint Programming Techniques

- Symmetries
- Parameterized Scripts
- → Multi-level Distribution
- Redundant Constraints
- User-defined Distributors
- Case Study: Scheduling

Multi-level Distribution

Often a problem can be naturally modeled in form of a hierarchy of levels. Such hierarchies can be implemented by using multiple distributors in a script.

Example: Map Coloring

Find the minimal number of colors to color the following map of western Europe:

Austria	: Germany Italy Switzerland
Belgium	: France Germany Luxembourg Netherlands
France	: Italy Germany Luxembourg Spain
Germany	: Luxembourg Netherlands
Italy	: Switzerland
Luxembourg	:
Netherlands	:
Portugal	: Spain
Spain	:
Switzerland	:

A Model for Map Coloring

- variable `N` denotes number of available colors
- variables C_i denote the colors of countries `i`
- constraints: not equal according to adjacency

Solution Strategy for Map Coloring

Level 1:

enumerate `N`, starting with `1`.

Level 2:

- create variables C_i ranging from `1` to `N`
- set up propagators
- eliminate symmetries
- enumerate variables C_i

Solution to Map Coloring

Map Coloring Demo

Multi-level Distribution

- problems often hierarchical
- each level corresponds to a distributor
- multiple distributors in one script

Overview Part IV: Constraint Programming Techniques

- Symmetries
- Parameterized Scripts
- Multi-level Distribution
- → Redundant Constraints
- User-defined Distributors
- Case Study: Scheduling

Redundant Constraints

- Pruning often not sufficient
- Adding redundant constraints sometimes helps

Example: Fractions

Find distinct non-zero digits such that the following equation holds:

$$\frac{A}{BC} + \frac{D}{EF} + \frac{G}{HI} = 1$$

Model for Fractions

- One variable for each letter, similar to `MONEY`
- Constraint

```
A*E*F*H*I +
D*B*C*H*I +
G*B*C*E*F = B*C*E*F*H*I
```

Additional Constraints for Fractions

- Symmetries

$$\frac{A}{B\ C} \geq \frac{D}{E\ F} \geq \frac{G}{H\ I}$$

- Redundant Constraints

$$3 * \frac{A}{B\ C} \geq 1$$

$$3 * \frac{G}{H\ I} \leq 1$$

Fractions Demo

- Constraint

```
A*E*F*H*I +
D*B*C*H*I +
G*B*C*E*F = B*C*E*F*H*I
```

- Symmetries

```
A*E*F >= D*B*C
D*H*I >= G*E*F
```

- Redundant Constraints

```
3*A >= B*C
3*G =< H*I
```

Redundant Constraints

Adding redundant constraints sometimes results in dramatic performance improvements.

Overview Part IV: Constraint Programming Techniques

- Symmetries
- Parameterized Scripts
- Multi-level Distribution
- Redundant Constraints
- User-defined Distributors
- Case Study: Scheduling

User-defined Distributors

- Distributors provided by the library do not capture problem-specific properties.
- Problem-specific distributors can be built using choice points.
- Distributors are often recursive procedures.

Examples of Distributors

- Naive distribution:
  - choose first variable with non-singleton domain
  - choose lowest value
- Domain-splitting distribution:
  - choose first variable with non-singleton domain
  - try lower half of its domain first

Implementation of Distributors

Distributor Demo

Overview Part IV: Constraint Programming Techniques

- Symmetries
- Parameterized Scripts
- Multi-level Distribution
- Redundant Constraints
- User-defined Distributors
- → Case Study: Scheduling

Problem Characteristics

- Precedence Constraints
- Capacity Constraints
- Optimization

Techniques for Handling Precedence

- Distribution imposes precedence instead of enumerating variables: Serializers
- resource-oriented serializers

Techniques for Handling Capacity

- Edge finding
- task intervals

145

Techniques for Handling Optimization

- Branch-and-Bound
- Lower Bounds
- Upper Bounds

146

Demo: Building A Bridge

- Problem description
- enumeration
- naive serializer
- resource-oriented serializer

Demo: MT10

- Edge finding
- task intervals
- lower/upper bounds

Scheduling Demo

Scheduling Demo

149

Summary Scheduling

- killer application of constraint programming
- integration of OR techniques
- development of domain-specific propagation and distribution techniques

150

Further Reading

- Finite Domain Constraint Programming in Oz, A Tutorial, Smolka, Schulte, Würtz, `http://www.ps.uni-sb.de/oz/documentation`
- Oz Scheduler: A Workbench for Scheduling Problems, Würtz, ICTAI'96

  `http://www.ps.uni-sb.de/Papers`

Overview

Part I: Getting Started

Part II: Constraint Propagation

Part III: Tree Search

Part IV: Constraint Programming Techniques

→ Part V: Programming Search Engines

Overview Part V: Programming Search Engines

- Depth-First Search
- Branch-And-Bound
- Recomputation-based Search
- Limited Discrepancy Search

First Solution Depth-First Search

```
fun {DFS S}
   case {Space.ask S}
   of succeeded then S
   [] failed then false
   [] alternatives(2) then
      C={Space.clone S} in
      {Space.commit S 1}
      case {DFS S}
      of false then {Space.commit C 2} {DFS C}
      elseof SolS then SolS
      end
   end
end
```

All Solution Depth-First Search

```
fun {DFS S}
   case {Space.ask S}
   of succeeded then [S]
   [] failed then nil
   [] alternatives(2) then
      C={Space.clone S} in
      {Space.commit S 1} {Space.commit C 2}
      {Append {DFS S} {DFS C}}
end
```


Branch-And-Bound

Idea: At each choice-point, solutions to the right are constrained to be better than solutions to the left.

Branch-and-Bound: Ordering

We assume a pruning procedure `Ordering`

```
proc {Constrain S SolS Ordering}
   {Inject S
    proc {$ NR}
       OR={Space.peek SolS}
    in
       {Ordering OR NR}
    end}
end
```


Branch-and-Bound: Implementation

```
fun {BAB S SolS Ord}
   case {Space.ask S} of failed then SolS
   [] succeeded then S
   [] alternatives(2) then
      C={Clone S}
      {Space.commit S 1} {Space.commit C 2}
      NewS={BAB S SolS Ord}
   in
      case NewS==SolS then skip
      else {Constrain C NewS Ord}
      end
   end
end
```

Recomputation-based Search

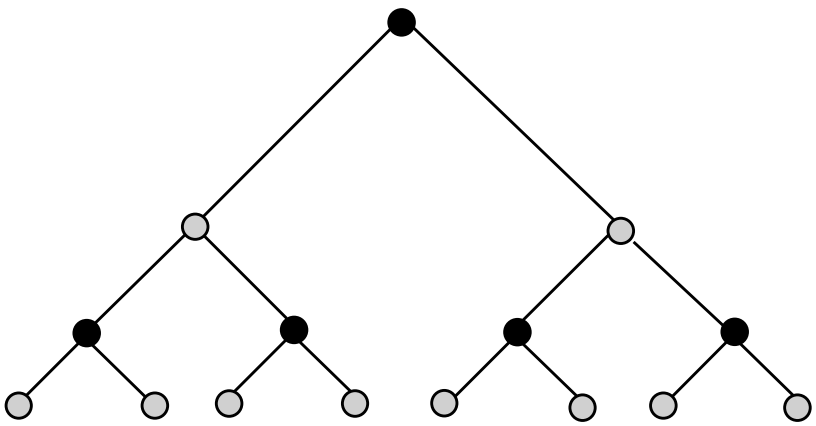
Observation: Cloning of spaces is a memory-intensive operation

Trade space for time by recomputing a space from an ancestor node

Describe path to ancestor with a list of integers

159

Recomputation-based Search



160

Recomputation-based Search

```
fun {Recompute S Is}
   case Is of nil then
      {Space.clone S}
   [] I|Ir then
      C={Recompute S Ir}
   in
      {Space.commit C I}
      C
   end
end
```


Recomputation-based Search: Demo

Recomputation Demo

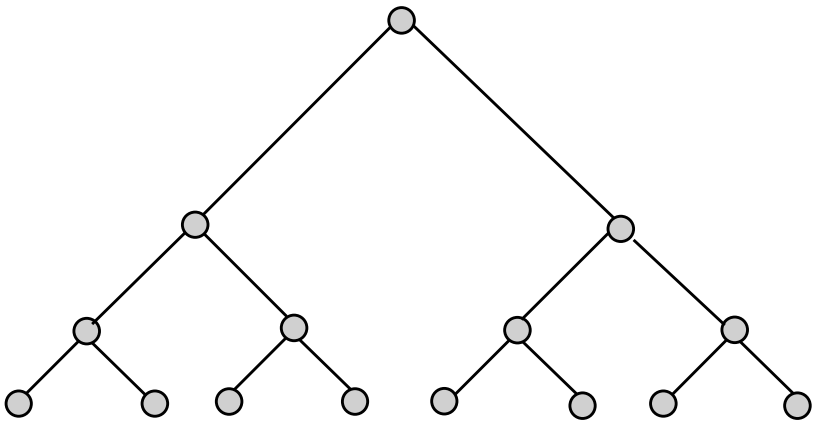
Limited Discrepancy Search

Assume a heuristic that rarely goes wrong.

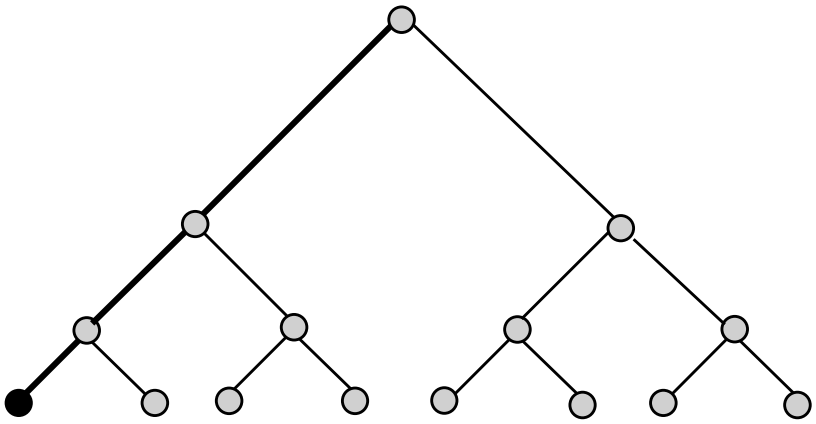
Choosing the right branch is a *discrepancy*.

LDS explores the search tree in increasing number of discrepancies.

Limited Discrepancy Search

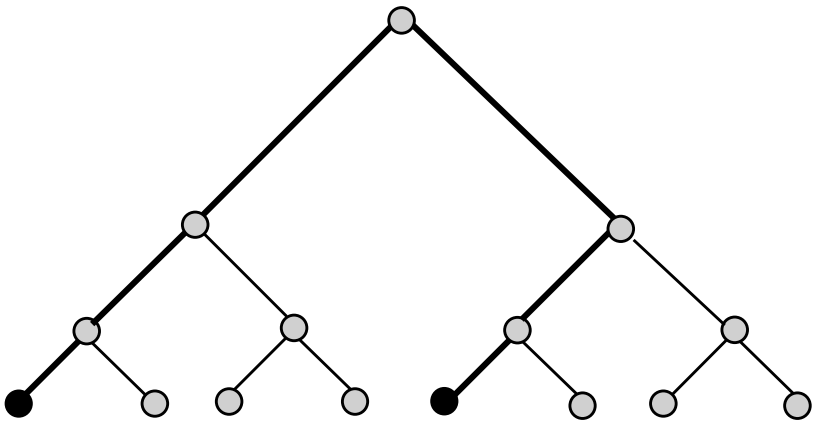


Limited Discrepancy Search



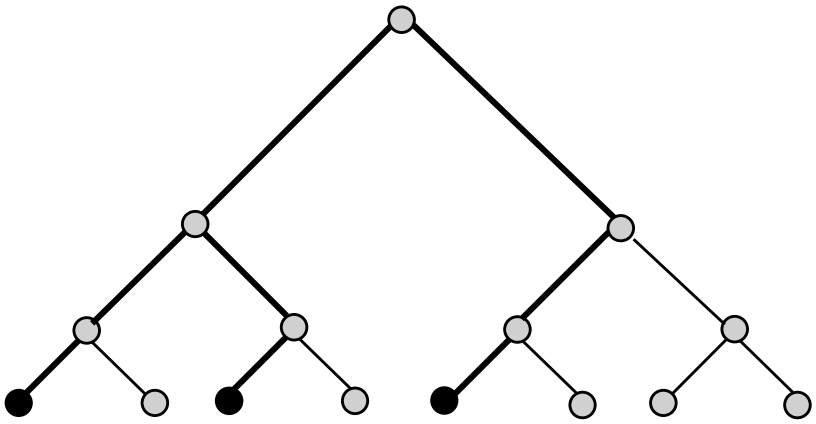
165

Limited Discrepancy Search



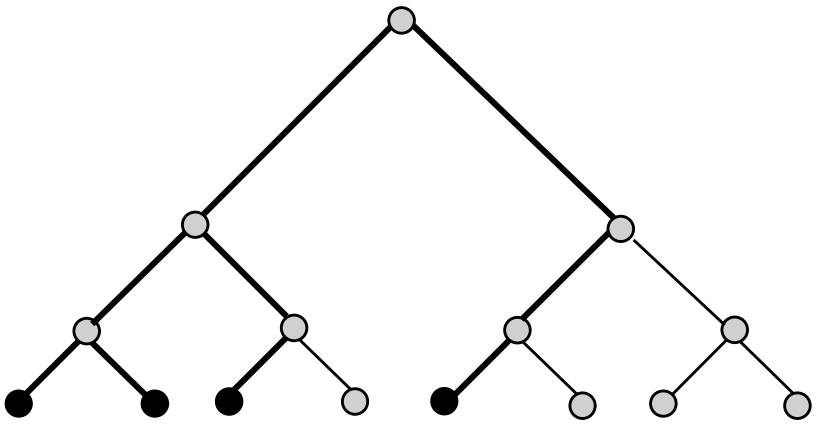
166

Limited Discrepancy Search



167

Limited Discrepancy Search



168

LDS: Implementation

```
fun {Probe S N}
   case {Space.ask S} of failed then nil
   [] succeeded then [S]
   [] alternatives(2) then
      case N>0 then C={Space.clone S} in
         {Space.commit S 2}
         case {Probe S N-1} of nil
         then {Space.commit C 1} {Probe C N}
         [] [T] then [T]
         end
      else {Space.commit S 1} {Probe S 0}
      end
   end
end
```


Programming Search Engines: Summary

- Spaces provide powerful and flexible abstraction for programming search engines
- Optimization and heuristic driven engines
- Trading space for time using recomputation

Further Reading

- Programming Constraint Inference Engines, Schulte, CP'97

  `http://www.ps.uni-sb.de/Papers`

Overview

Part I: Getting Started

Part II: Constraint Propagation

Part III: Tree Search

Part IV: Constraint Programming Techniques

Part V: Programming Search Engines

→ Concluding Remarks

Personal Remarks on Constraint Programming in Oz

- main "competitors"
  - ILOG Solver/OPL
  - CHIP (Cosytec)
  - Eclipse (IC-PARC)
- programming language approach
  - high threshold (syntax, paradigms)
  - rapid prototyping
- research spirit
  - open community of users
  - DFKI Oz 2.0 is public domain software

Research Directions

- distributed problem solving
- software patterns for constraint programming
- constraint programming applications

Further Information

- Oz and DFKI Oz
  - download system and documentation from
    `http://www.ps.uni-sb.de/oz`
  - research publications
    `http://www.ps.uni-sb.de/Papers`
  - research system Mozart, a distributed implementation of Oz
    `http://www.ps.uni-sb.de/mozart`
- Tutorial slides, source code of programs
  Martin Henz, `http://www.comp.nus.edu/~henz`