Chapter 1: Introduction

Helmut Simonis

Cork Constraint Computation Centre
Computer Science Department
University College Cork
Ireland

ECLiPSe ELearning Overview

Licence

Outline

What we want to introduce

- Constraint Programming
- Using ECLiPSe Language
- With Saros Eclipse IDE

Constraint Programming (CP)

- Solve hard combinatorial problems
- With minimal programming effort
- Exploit strategies and heuristics
- Understand and control problem solving

ECLiPSe Language

- Open source constraint programming language
- Flexible toolkit to develop/use constraints
- Contains different constraint solvers
- Here: Use of finite domains/(mixed) integer programming

Aims and Outcomes

- Understand what constraint programming is
- How constraint programs can be applied to a problem
- Which application problems are good candidates for CP
- How to write/run/analyze simple ECLiPSe programs

You should already know about...

- No hard requirements
- Basic understanding of programming assumed
- Useful to have some background in one of:
  - Network Management
  - Integer Programming
  - Combinatorial Optimization

Choices of materials

Slides PDF files for computer viewing
- Contains animations of visualization
- Large file sizes

Handout PDF files for printing
- 2 slides per page
- Does not contain all animations

Transcript Text of presentation as articles

Video Video presentation with audio (640x480 pixels)

iPhone Video presentation tuned for iPhone display (480x320 pixels)

Chapters

Introduction (You are here)	Video	iPhone	Slides	Handout
First Steps - Hello World	Video	iPhone	Slides	Handout
Application Overview	Video	iPhone	Slides	Handout
Basic Constraint Reasoning	Video	iPhone	Slides	Handout
Global Constraints	Video	iPhone	Slides	Handout
Search Strategies	Video	iPhone	Slides	Handout
Optimization	Video	iPhone	Slides	Handout
Symmetry Breaking	Video	iPhone	Slides	Handout
Choosing the Model	Video	iPhone	Slides	Handout
Customizing Search	Video	iPhone	Slides	Handout
Limits of Propagation	Video	iPhone	Slides	Handout
Systematic Development	Video	iPhone	Slides	Handout
Visualization Techniques	Video	iPhone	Slides	Handout
Finite Set and Continuous Variables	Video	iPhone	Slides	Handout
Network Applications	Video	iPhone	Slides	Handout
More Global Constraints	Video	iPhone	Slides	Handout
Adding Material	Video	iPhone	Slides	Handout

Applications

Application Overview	Video	iPhone	Slides	Handout
SEND+MORE=MONEY	Video	iPhone	Slides	Handout
Sudoku	Video	iPhone	Slides	Handout
N-Queens	Video	iPhone	Slides	Handout
Routing and Wavelenght Assignment	Video	iPhone	Slides	Handout
Balanced Incomplete Block Designs	Video	iPhone	Slides	Handout
Sports Scheduling	Video	iPhone	Slides	Handout
Progressive Party	Video	iPhone	Slides	Handout
Costas Array	Video	iPhone	Slides	Handout
SONET/SDH Ring Design	Video	iPhone	Slides	Handout
Network Applications	Video	iPhone	Slides	Handout
Car Sequencing	Video	iPhone	Slides	Handout

Cork Constraint Computation Centre

Introduction

- Aims and Outcomes
- Overview of chapters
- Hyperlinks to all materials

Video iPhone Slides Handout

Cork Constraint Computation Centre

First Steps - Hello World

- How to install ECLiPSe and Saros
- Writing a first program
- Running the program
- Where to find information

Video iPhone Slides Handout

Application Overview

- Why constraint programming is interesting
- Solving industrial problems with CP
- Main application areas
  - Assignment
  - Scheduling
  - Network problems
  - Transportation
  - Personnel Assignment

Video iPhone Slides Handout

Basic Constraint Reasoning - SEND+MORE = MONEY

- Finite Domain variables
- CP: Variables + Constraints + Search
- Bounds reasoning on arithmetic constraints
- Simple visualizers

Video iPhone Slides Handout

Global Constraints - Sudoku

- Modellimg the Sudoku puzzle
- One model, different behaviours
- Global constraint: `alldifferent`
- Bounds and domain consistency
- A domain consistent `alldifferent`

Video iPhone Slides Handout

Search Strategies - N Queens

- How to search for a solution
- Variable and value choice
- How to avoid deep backtracking
- Partial search strategies

Video iPhone Slides Handout

Optimization - Routing and Wavelength Assignment

- Optimization
- Graph algorithms library
- Integer Programming with `eplex`
- Problem decomposition
- Routing and Wavelength Assignment in Optical Networks

Video iPhone Slides Handout

Symmetry Breaking - Balanced Incomplete Block Designs

- Balanced Incomplete Block Designs
- Planning Experiments and Testing Features
- Problems with highly symmetrical structure
- Symmetry Breaking with `lex` constraints

Video iPhone Slides Handout

Constraint Programming
Chapter Overview
Chapter Details

Choosing the Model - Sports Scheduling

- Complex sports scheduling problem
- How to decide which model to use
- Improving reasoning by channeling

Video iPhone Slides Handout

Customizing Search - Progressive Party

- Scheduling Meetings between Teams
- Teams only meet once
- Capacity Limits
- Build customized search routines tailored to problem
- Problem decomposition: decide which problem to solve

Video iPhone Slides Handout

Limits of Propagation - Costas Array

- Antenna/Sonar Design
- Hard Benchmark Problem
- Naive Enumeration works best
- When clever reasoning doesn't pay off
- Cautionary Tale

Video iPhone Slides Handout

Systematic Development

- Developing Programs
- Testing
- Profiling
- Documentation

Video iPhone Slides Handout

Visualization Techniques

- How to visualize constraint programs
- Variable Visualizers
- Understanding Search Trees
- Constraint Visualizers
- Complex Visualizations

Video iPhone Slides Handout

Finite Set and Continuous Variables - SONET Design Problem

- Finite set variables
- Continuous domains
- Optimization from below
- Advanced symmetry breaking
- SONET design problem without inter-ring traffic

Video iPhone Slides Handout

Network Applications

- Overview of Network Applications
- Traffic Placement
- Capacity Management
- Network Design
- Using Advanced Techniques

Video iPhone Slides Handout

More Global Constraints - Car Sequencing

- New global constraints: `gcc` and `sequence`
- Choosing a better search strategy

Video iPhone Slides Handout

Adding Material

- How to add new chapters
- Copying template files
- Configuring templates
- Adding frames to body
- Integrating with other chapters

Video iPhone Slides Handout

To continue

- Branch from here to all materials
- Choose presentation form which suits you

Chapter 2: First Steps

Helmut Simonis

Cork Constraint Computation Centre
Computer Science Department
University College Cork
Ireland

ECLiPSe ELearning Overview

Licence

Outline

What we want to introduce

- How to install ECLiPSe
- Installing Saros
- Writing a first program
- Running the program
- Where to find information

Chapter 3: Application Overview

Helmut Simonis

Cork Constraint Computation Centre
Computer Science Department
University College Cork
Ireland

ECLiPSe ELearning Overview

Licence

Outline

What is the common element amongst

- The production of Mirage 2000 fighter aircraft
- The personnel planning for the guards in all French jails
- The production of Belgian chocolates
- The selection of the music programme of a pop music radio station
- The design of advanced signal processing chips
- The print engine controller in Xerox copiers

They all use constraint programming!

Constraint Programming - in a nutshell

- Declarative description of problems with
  - *Variables* which range over (finite) sets of values
  - *Constraints* over subsets of variables which restrict possible value combinations
  - A *solution* is a value assignment which satisfies all constraints
- Constraint propagation/reasoning
  - Removing inconsistent values for variables
  - Detect failure if constraint can not be satisfied
  - Interaction of constraints via shared variables
  - Incomplete
- Search
  - User controlled assignment of values to variables
  - Each step triggers constraint propagation
- Different domains require/allow different methods

Constraint Satisfaction Problems (CSP)

- Different problems with common aspects
  - Planning
  - Scheduling
  - Resource allocation
  - Assignment
  - Placement
  - Logistics
  - Financial decision making
  - VLSI design

Characteristics of these problems

- There are no general methods or algorithms
  - NP-completeness
  - Different strategies and heuristics have to be tested.
- Requirements are quickly changing:
  - Programs should be flexible enough to adapt to these changes rapidly.
- Decision support required
  - Co-operate with user
  - Friendly interfaces

Benefits of CLP approach

- Short development time
  - Fast prototyping
  - Refining of modelling
  - Same tool used for prototyping/production
- Compact code size
  - Ease of understanding
  - Maintenance
- Simple modification
  - Changing requirements
  - No need to understand all aspects of problem
- Good performance
  - Fast answer
  - Good results
  - Optimal solutions rarely required

Overview

- Production sequencing
- Production scheduling
- Satellite tasking
- Maintenance planning
- Product blending
- Time tabling
- Crew rotation
- Aircraft rotation
- Transport
- Personnel assignment
- Personnel requirement planning
- Hardware design
- Compilation
- Financial problems
- Placement
- Cutting problems
- Stand allocation
- Air traffic control
- Frequency allocation
- Network configuration
- Product design
- Production step planning

Tools Used (Prolog Based Constraint Languages)

- CHIP
  - 1986-1990 ECRC, Munich, Germany
  - 1990-today COSYTEC, Orsay, France
- ECLiPSe
  - 1984-1996 ECRC
  - 1996-2004 IC-Parc, PTL, London
  - 2004-today Cisco Systems
  - a.k.a. Sepia (ECRC)
  - a.k.a. DecisionPower (ICL)

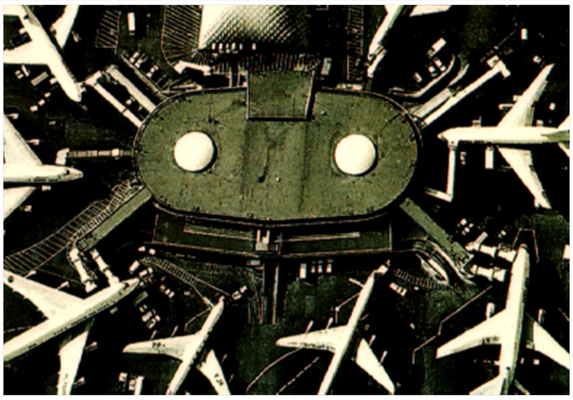
Five central topics

- Assignment
  - Parking assignment
  - Platform allocation
- Network Configuration
- Scheduling
  - Production scheduling
  - Project planning
- Transport
  - Lorry, train, airlines
- Personnel assignment
  - Timetabling, Rostering
  - Train, airlines

Stand allocation

- HIT (ICL)
  - Assign ships to berths in container harbor
  - Developed with ECRC's version of CHIP
    - Then using DecisionPower (ICL)
    - Early version of ECLiPSe
  - First operational constraint application (1989-90)
- APACHE (COSYTEC)
  - Stand allocation for airport
- Refinery berth allocation (ISAB/COSYTEC)
  - Where to load/unload ships in refinery

APACHE - AIR FRANCE (COSYTEC)

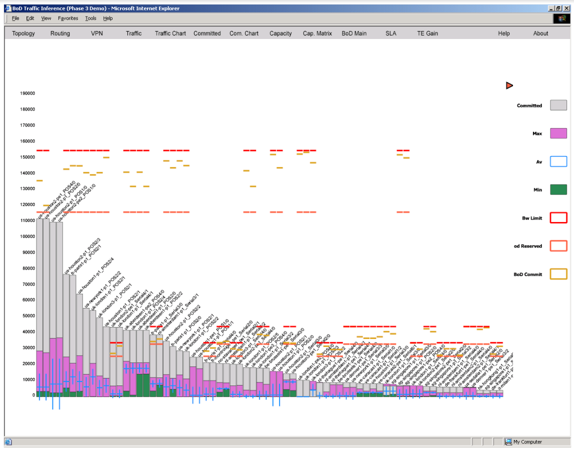


- Stand allocation system
  - For Air Inter/Air France
  - Roissy, CDG2
  - Packaged for large airports
- Complex constraint problem
  - Technical constraints
  - Operational constraints
  - Incremental re-scheduler
- Cost model
  - Max. nb passengers in contact
  - Min. towing, bus usage
- Benefits and status
  - Quasi real-time re-scheduling
  - KAL, Turkish Airlines

Network configuration

- BoD (PTL)
- Locarim (France Telecom, COSYTEC)
  - Cabling of building
- Planets (UCB, Enher)
  - Electrical power network reconfiguration
- Load Balancing in Banking networks (ICON)
  - Distributed applications
  - Control network traffic
- Water Networks (UCB, ClocWise)

BoD - Schlumberger (IC-Parc/PTL)



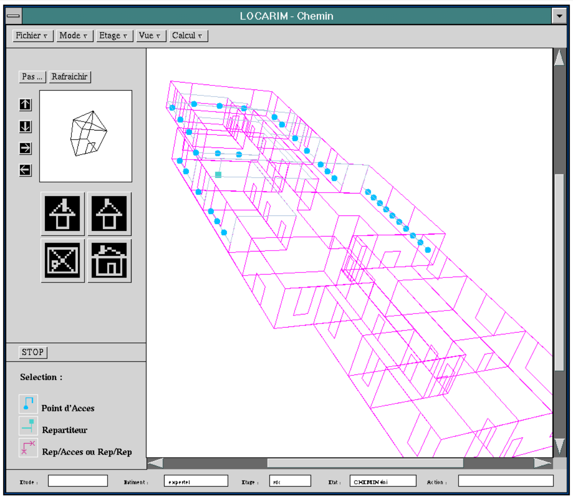
- Bandwidth on Demand
  - Provide guaranteed QoS
  - For temporary connections
  - Video conferences
  - Oil well logging
- World-wide, sparse network
- Bandwidth limited
- Do not affect existing traffic
- Uses route generator module for MPLS-TE
  - Model extended with temporal component
- First version delivered February, 2003

ISC-TEM - Cisco Systems

- Traffic Engineering in MPLS
- Find routes for demands satisfying bandwidth limits
- Path placement algorithm developed for Cisco by PTL and IC-Parc (2002-2004)
- Internal, competitive selection of approaches
- Strong emphasis on stability
- Written in ECLiPSe
- PTL bought by Cisco in 2004
- Part of team moved to Boston

LOCARIM - France Telecom

- Intelligent cabling system
  - For large buildings
  - Developed by
    - COSYTEC
    - Telesystemes
- Application
  - Input scanned drawing
  - Specify requirements
- Optimization
  - Minimize cabling, drilling
  - Reduce switches
  - Shortest path
- Status
  - Operational in 5 Telecom sites
  - Generates quotations



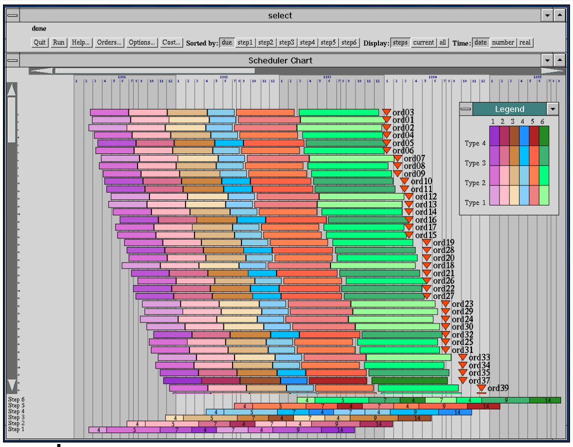
Cork Constraint Computation Centre

Production Scheduling

- Amylum (OM Partners)
  - Glucose production
- Cerestar (OM Partners)
  - Glucose production
- Saveplan (Sligos)
  - Production scheduling
- Trefi Metaux (Sligos)
  - Heavy industry production scheduling
- Michelin
  - Rubber blending, rework optimization

Cork Constraint Computation Centre

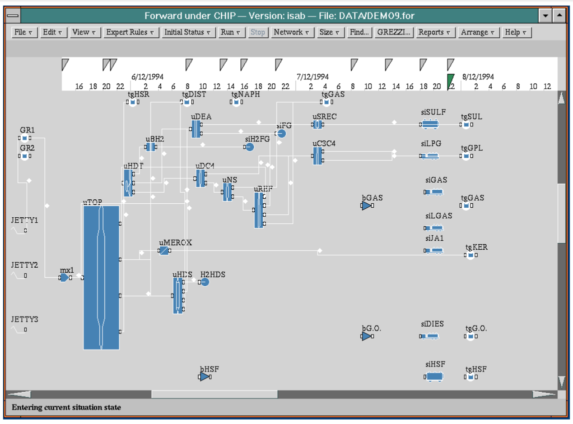
PLANE - Dassault Aviation



- Assembly line scheduling
  - Mirage 2000 Fighter
  - Falcon business jet
- Two user system
  - Production planning 3-5 years
  - Commercial what-if sales aid
- Optimisation
  - Balanced schedule
  - Minimise changes in production rate
  - Minimise storage costs
- Benefits and status
  - Replaces 2 week manual planning
  - Operational since Apr 94
  - Used in US for business jets

Cork Constraint Computation Centre

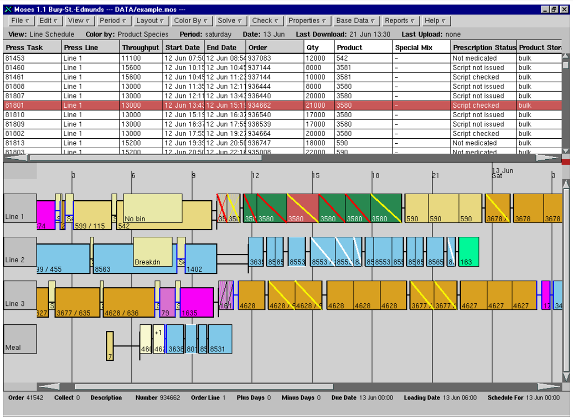
FORWARD - Fina



- Oil refinery scheduling
  - Developed by
    - TECHNIP
    - COSYTEC
  - Uses simulation tool
    - Forward by Elf
- Schedules daily production
  - Crude arrival →
  - Processing → Delivery
  - Design, optimize and simulate
- Product Blending
  - Explanation facilities
  - Handling of over-constrained problems
- Status
  - Operational since June 94
  - Operational at FINA, ISAB, BP

Cork Constraint Computation Centre

MOSES - Dalgety



- Animal feed production
  - Feed in different sizes/
  - For different species
  - Human health risk
    - Contamination
    - BSE
  - Strict regulations
- Constraints
  - Avoid contamination risks
  - Machine setup times
  - Machine choice (quality/speed)
  - Limited storage of finished products
  - Very short lead times (8-48 hours)
  - Factory structure given as data
- Status
  - Operational since Nov 96

Transport

- By Air
  - AirPlanner (PT)
  - Daysy (Lufthansa)
  - Pilot (SAS)
- By Road
  - Wincanton (IC-Parc)
  - TACT (SunValley)
  - EVA (EDF)
- By Rail
  - CREW (Servair)
  - COBRA (NWT)

AirPlanner (IC-Parc)

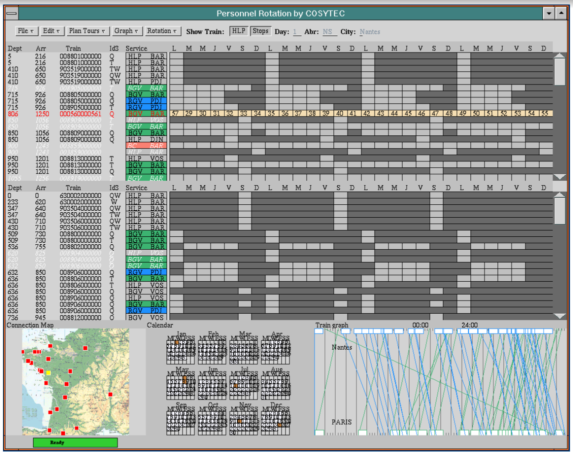
- Based on the Retimer project for BA
- Consider fleet of aircraft
- Shifting some flights by small amount may allow better use of fleet
- Many constraints of different types limit the changes that are possible

Wincanton (IC-Parc)

- Large scale distribution problem
- Deliver fresh products to supermarkets
- Direct deliveries/warehousing
- Combining deliveries
- Capacity constraints
- Tour planning
- Workforce constraints

CREW - Servair

- Crew rostering system
  - Assign service staff to TGV
  - Bar/Restaurant service
  - Joint design COSYTEC/GSI
- Problem solver
  - Generates tours/cycles
  - Assigns skilled personnel
- Constraints
  - Union, physical, calendar
- Status
  - Operational since Mar 1995
  - Cost reduction by 5%



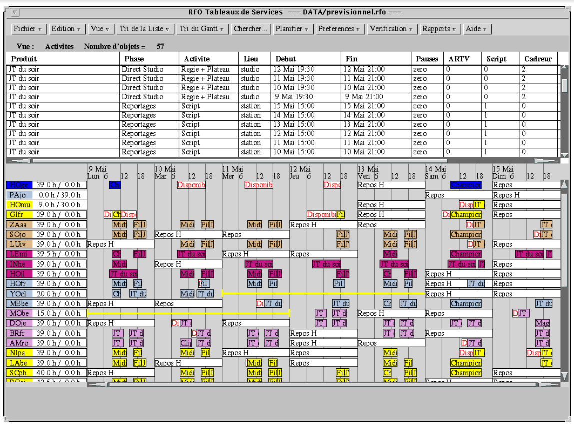
Personnel Planning

- RAC (IC-Parc)
- OPTISERVICE (RFO)
- Shifter (ERG Petroli)
- Gymnaste (UCF)
- MOSAR (Ministère de la JUSTICE)

RAC

- Personnel dispatching
- On-line problem
  - Change plan as new requests are phoned in
- Typical constraints for workforce
  - Duty time
  - Rest periods
  - Max driving time
  - Response time
- Operational/Strategic use

OPTI SERVICE - RFO



- Assignment of technical staff
  - Overseas radio/TV network
  - Radio France Outre-mer
  - Joint development:
    - GIST and COSYTEC
  - 250 journalists and technicans
- Features
  - Schedule manually,
  - Check, Run automatic
  - Rule builder to specify cost formulas
  - Minimize overtime, temporary staff
  - Compute cost of schedule
- Status
  - Operational since 1997
  - Installed worldwide in 8 sites
  - Developed into generic tool

Nurse Scheduling

- GYMNASTE
- Time tabling
- Personnel assignment
- Provisional and reactive planning (1-6 weeks)
- Developed by COSYTEC with partners
  - PRAXIM/Université Joseph Fourier de Grenoble
- Pilot site Grenoble
- Also used at hôpital de BLIGNY (Paris)
- Advantages :
  - Plan generation in 5 minutes
  - User/personnel preferences
  - Decrease in days lost

Conclusions

- Constraint Programming useful for many domains
- Large scale industrial use in
  - Assignment
  - Network Management
  - Production Scheduling
  - Transport
  - Personnel Planning

Good approach for specialized, complex problems

- 3D camera control in movie animation
- Finding instable control states for robots
- Optimized register allocation in gcc

Key advantages

- Easy to prototype/develop
- Using modelling to understand problem
- Expressive power
- Add/remove constraints as problem evolves
- Customized search exploiting structure and knowledge

Chapter 4: Basic Constraint Reasoning (SEND+MORE=MONEY)

Helmut Simonis

Cork Constraint Computation Centre
Computer Science Department
University College Cork
Ireland

ECLiPSe ELearning Overview

Licence

Outline

What we want to introduce

- Finite Domain Solver in ECLiPSe
- Models and Programs
- Constraint Propagation and Search
- Basic constraints: linear arithmetic, alldifferent, disequality
- Built-in search: Labeling
- Visualizers for variables, constraints and search

Problem Definition

A Crypt-Arithmetic Puzzle

We begin with the definition of the SEND+MORE=MONEY puzzle. It is often shown in the form of a hand-written addition:

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

Rules

- Each character stands for a digit from 0 to 9.
- Numbers are built from digits in the usual, positional notation.
- Repeated occurrence of the same character denote the same digit.
- Different characters denote different digits.
- Numbers do not start with a zero.
- The equation must hold.

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

Model

- Each character is a variable, which ranges over the values 0 to 9.
- An *alldifferent* constraint between all variables, which states that two different variables must have different values. This is a very common constraint, which we will encounter in many other problems later on.
- Two *disequality constraints* (variable *X* must be different from value *V*) stating that the variables at the beginning of a number can not take the value 0.
- An arithmetic *equality constraint* linking all variables with the proper coefficients and stating that the equation must hold.

General Program Structure

```
:- module(sendmory).
:- export(sendmory/1).
:- lib(ic).
sendmory(L):-
    L = [S,E,N,D,M,O,R,Y],⇨ Variables
    L :: 0..9,
    alldifferent(L),⇨ Constraints
    S #\= 0, M #\= 0,
    1000*S + 100*E + 10*N + D +
    1000*M + 100*O + 10*R + E #=
    10000*M + 1000*O + 100*N + 10*E + Y,
    labeling(L).⇨ Search
```

Choice of Model

- This is *one* model, not *the* model of the problem
- Many possible alternatives
- Choice often depends on your constraint system
  - Constraints available
  - Reasoning attached to constraints
- Not always clear which is the *best* model
- Often: Not clear what is the *problem*

▸ Alternative 1 ▸ Alternative 2

Running the program

- To run the program, we have to enter the query
  - `sendmory:sendmory(L).`
- Result
  - `L = [9, 5, 6, 7, 1, 0, 8, 2]`
  - `yes (0.00s cpu, solution 1, maybe more)`

Question

- But how did the program come up with this solution?

Domain Definition

```
L = [S,E,N,D,M,O,R,Y],
L :: 0..9,
```

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

Domain Visualization

Columns = Values

Rows = Variables

	0	1	2	3	4	5	6	7	8	9
S										
E										
N										
D										
M			Cells= State							
O										
R										
Y										

Alldifferent Constraint

```
alldifferent(L),
```

- Built-in of `ic` library
- No initial propagation possible
- *Suspends*, waits until variables are changed
- When variable is fixed, remove value from domain of other variables
- *Forward checking*

Alldifferent Visualization

Uses the same representation as the domain visualizer

	0	1	2	3	4	5	6	7	8	9
S										
E										
N										
D										
M										
O										
R										
Y										

Disequality Constraints

```
S #\= 0, M#\= 0,
```

Remove value from domain

$$S \in \{1..9\}, M \in \{1..9\}$$

Constraints solved, can be removed

Domains after Disequality

	0	1	2	3	4	5	6	7	8	9
S										
E										
N										
D										
M										
O										
R										
Y										

Equality Constraint

- Normalization of linear terms
  - Single occurence of variable
  - Positive coefficients
- Propagation

Normalization

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

is transformed into

	1000*S+	91*E+		D
		+	10*R	
9000*M+	900*O+	90*N+		Y

Simplified Equation

$$1000*S+91*E+10*R+D = 9000*M+900*O+90*N+Y$$

Propagation

$$\underbrace{1000 * S^{1..9} + 91 * E^{0..9} + 10 * R^{0..9} + D^{0..9}}_{1000..9918} =$$
$$\underbrace{9000 * M^{1..9} + 900 * O^{0..9} + 90 * N^{0..9} + Y^{0..9}}_{9000..89919}$$

Deduction:

$$M = 1, S = 9, O \in \{0..1\}$$

Why? Skip

Consider lower bound for S

$$\underbrace{1000 * S^{1..9} + 91 * E^{0..9} + 10 * R^{0..9} + D^{0..9}}_{9000..9918} = \underbrace{9000 * M^{1..9} + 900 * O^{0..9} + 90 * N^{0..9} + Y^{0..9}}_{9000..9918}$$

- Lower bound of equation is 9000
- Rest of lhs (left hand side) $(91 * E^{0..9} + 10 * R^{0..9} + D^{0..9})$ is atmost 918
- S must be greater or equal to $\frac{9000-918}{1000} = 8.082$
  - otherwise lower bound of equation not reached by lhs
- S is integer, therefore $S \geq \lceil \frac{9000-918}{1000} \rceil = 9$
- S has upper bound of 9, so $S = 9$

Consider upper bound of M

$$\underbrace{1000 * S^{1..9} + 91 * E^{0..9} + 10 * R^{0..9} + D^{0..9}}_{9000..9918} = \underbrace{9000 * M^{1..9} + 900 * O^{0..9} + 90 * N^{0..9} + Y^{0..9}}_{9000..9918}$$

- Upper bound of equation is 9918
- Rest of rhs (right hand side) $900 * O^{0..9} + 90 * N^{0..9} + Y^{0..9}$ is at least 0
- M must be smaller or equal to $\frac{9918-0}{9000} = 1.102$
- M must be integer, therefore $M \leq \lfloor \frac{9918-0}{9000} \rfloor = 1$
- M has lower bound of 1, so $M = 1$

Problem
Program
Constraint Setup
Search
Lessons Learned

Domain Definition
Alldifferent Constraint
Disequality Constraints
Equality Constraint

Consider upper bound of O

$$\underbrace{1000 * S^{1..9} + 91 * E^{0..9} + 10 * R^{0..9} + D^{0..9}}_{9000..9918} = \underbrace{9000 * M^{1..9} + 900 * O^{0..9} + 90 * N^{0..9} + Y^{0..9}}_{9000..9918}$$

- Upper bound of equation is 9918
- Rest of rhs (right hand side) $9000 * 1 + 90 * N^{0..9} + Y^{0..9}$ is at least 9000
- O must be smaller or equal to $\frac{9918-9000}{900} = 1.02$
- O must be integer, therefore $O \leq \lfloor \frac{9918-9000}{900} \rfloor = 1$
- O has lower bound of 0, so $O \in \{0..1\}$

Propagation of equality: Result

	0	1	2	3	4	5	6	7	8	9
S		-	-	-	-	-	-	-	-	✹
E										
N										
D										
M		✹	-	-	-	-	-	-	-	-
O			✖	✖	✖	✖	✖	✖	✖	✖
R										
Y										

Propagation of alldifferent

	0	1	2	3	4	5	6	7	8	9
S										
E										
N										
D										
M										
O										
R										
Y										

$$O = 0, [E, R, D, N, Y] \in \{2..8\}$$

Waking the equality constraint

- Triggered by assignment of variables
- *or* update of lower or upper bound

Removal of constants

$$1000 * 9 + 91 * E^{2..8} + 10 * R^{2..8} + D^{2..8} = \\ 9000 * 1 + 900 * 0 + 90 * N^{2..8} + Y^{2..8}$$

$$\mathbf{1000} * \mathbf{9} + 91 * E^{2..8} + 10 * R^{2..8} + D^{2..8} = \\ \mathbf{9000} * \mathbf{1} + \mathbf{900} * \mathbf{0} + 90 * N^{2..8} + Y^{2..8}$$

$$91 * E^{2..8} + 10 * R^{2..8} + D^{2..8} = 90 * N^{2..8} + Y^{2..8}$$

Propagation of equality (Iteration 1)

$$\underbrace{91 * E^{2..8} + 10 * R^{2..8} + D^{2..8}}_{204..816} = \underbrace{90 * N^{2..8} + Y^{2..8}}_{182..728}$$

$$\underbrace{91 * E^{2..8} + 10 * R^{2..8} + D^{2..8} = 90 * N^{2..8} + Y^{2..8}}_{204..728}$$

$$N \geq 3 = \lceil \frac{204 - 8}{90} \rceil, E \leq 7 = \lfloor \frac{728 - 22}{91} \rfloor$$

Cork Constraint Computation Centre

Problem
Program
Constraint Setup
Search
Lessons Learned
Domain Definition
Alldifferent Constraint
Disequality Constraints
Equality Constraint

Propagation of equality (Iteration 2)

$$91 * E^{2..7} + 10 * R^{2..8} + D^{2..8} = 90 * N^{3..8} + Y^{2..8}$$

$$\underbrace{91 * E^{2..7} + 10 * R^{2..8} + D^{2..8}}_{204..725} = \underbrace{90 * N^{3..8} + Y^{2..8}}_{272..728}$$

$$\underbrace{91 * E^{2..7} + 10 * R^{2..8} + D^{2..8} = 90 * N^{3..8} + Y^{2..8}}_{272..725}$$

$$E \geq 3 = \lceil \frac{272 - 88}{91} \rceil$$

Cork Constraint Computation Centre

Propagation of equality (Iteration 3)

$$91 * E^{3..7} + 10 * R^{2..8} + D^{2..8} = 90 * N^{3..8} + Y^{2..8}$$

$$\underbrace{91 * E^{3..7} + 10 * R^{2..8} + D^{2..8}}_{295..725} = \underbrace{90 * N^{3..8} + Y^{2..8}}_{272..728}$$

$$\underbrace{91 * E^{3..7} + 10 * R^{2..8} + D^{2..8} = 90 * N^{3..8} + Y^{2..8}}_{295..725}$$

$$N \geq 4 = \lceil \frac{295 - 8}{90} \rceil$$

Problem
Program
Constraint Setup
Search
Lessons Learned

Domain Definition
Alldifferent Constraint
Disequality Constraints
Equality Constraint

Propagation of equality (Iteration 4)

$$91 * E^{3..7} + 10 * R^{2..8} + D^{2..8} = 90 * N^{4..8} + Y^{2..8}$$

$$\underbrace{91 * E^{3..7} + 10 * R^{2..8} + D^{2..8}}_{295..725} = \underbrace{90 * N^{4..8} + Y^{2..8}}_{362..728}$$

$$\underbrace{91 * E^{3..7} + 10 * R^{2..8} + D^{2..8} = 90 * N^{4..8} + Y^{2..8}}_{362..725}$$

$$E \geq 4 = \lceil \frac{362 - 88}{91} \rceil$$

Propagation of equality (Iteration 5)

$$91 * E^{4..7} + 10 * R^{2..8} + D^{2..8} = 90 * N^{4..8} + Y^{2..8}$$

$$\underbrace{91 * E^{4..7} + 10 * R^{2..8} + D^{2..8}}_{386..725} = \underbrace{90 * N^{4..8} + Y^{2..8}}_{362..728}$$

$$\underbrace{91 * E^{4..7} + 10 * R^{2..8} + D^{2..8} = 90 * N^{4..8} + Y^{2..8}}_{386..725}$$

$$N \geq 5 = \lceil \frac{386 - 8}{90} \rceil$$

Propagation of equality (Iteration 6)

$$91 * E^{4..7} + 10 * R^{2..8} + D^{2..8} = 90 * N^{5..8} + Y^{2..8}$$

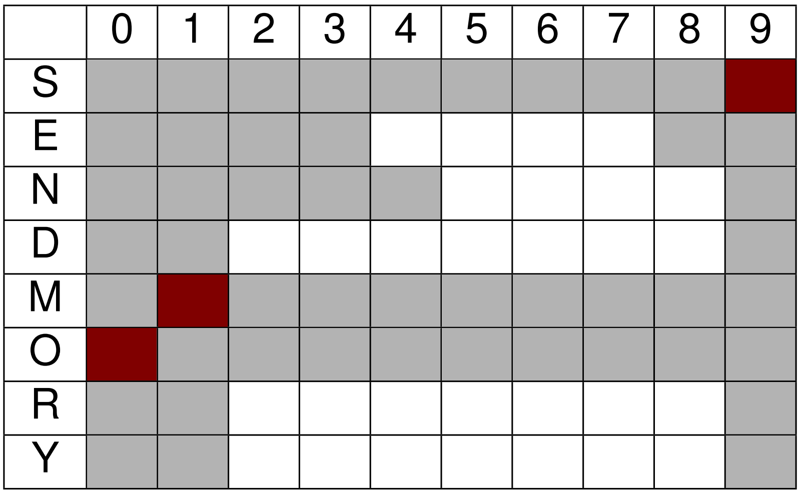
$$\underbrace{91 * E^{4..7} + 10 * R^{2..8} + D^{2..8}}_{386..725} = \underbrace{90 * N^{5..8} + Y^{2..8}}_{452..728}$$

$$\underbrace{91 * E^{4..7} + 10 * R^{2..8} + D^{2..8} = 90 * N^{5..8} + Y^{2..8}}_{452..725}$$

$$N \geq 5 = \lceil \frac{452 - 8}{90} \rceil, E \geq 4 = \lceil \frac{452 - 88}{91} \rceil$$

No further propagation at this point

Domains after setup

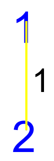


`labeling` built-in

```
labeling([S,E,N,D,M,O,R,Y])
```

- Try variable is order given
- Try values starting from smallest value in domain
- When failing, backtrack to last open choice
- *Chronological Backtracking*
- *Depth First search*

Search Tree Step 1



Variable S already fixed

Problem
Program
Constraint Setup
Search
Lessons Learned

Step 1
Step 2
Further Steps
Solution

Step 2, Alternative $E = 4$

Variable $E \in \{4..7\}$, first value tested is 4

Assignment $E = 4$

	0	1	2	3	4	5	6	7	8	9
S										
E					✸	-	-	-		
N										
D										
M										
O										
R										
Y										

Propagation of $E = 4$, equality constraint

$$91 * 4 + 10 * R^{2..8} + D^{2..8} = 90 * N^{5..8} + Y^{2..8}$$

$$\underbrace{91 * 4 + 10 * R^{2..8} + D^{2..8}}_{386..452} = \underbrace{90 * N^{5..8} + Y^{2..8}}_{452..728}$$

$$\underbrace{91 * 4 + 10 * R^{2..8} + D^{2..8} = 90 * N^{5..8} + Y^{2..8}}_{452}$$

$$N = 5, Y = 2, R = 8, D = 8$$

Result of equality propagation

	0	1	2	3	4	5	6	7	8	9
S										
E										
N						✹	-	-	-	
D			-	-	-	-	-	-	✹	
M										
O										
R			-	-	-	-	-	-	✹	
Y			✹	-	-	-	-	-	-	

Cork Constraint Computation Centre

Propagation of alldifferent

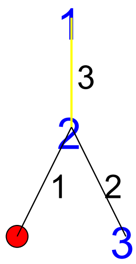
	0	1	2	3	4	5	6	7	8	9
S									|	
E									|	
N						✹	-	-	|	
D			-	-	-	-	-	-	✹	
M									|	
O									|	
R			-	-	-	-	-	-	✹	
Y			✹	-	-	-	-	-	|	

Alldifferent fails!

Cork Constraint Computation Centre

Step 2, Alternative $E = 5$

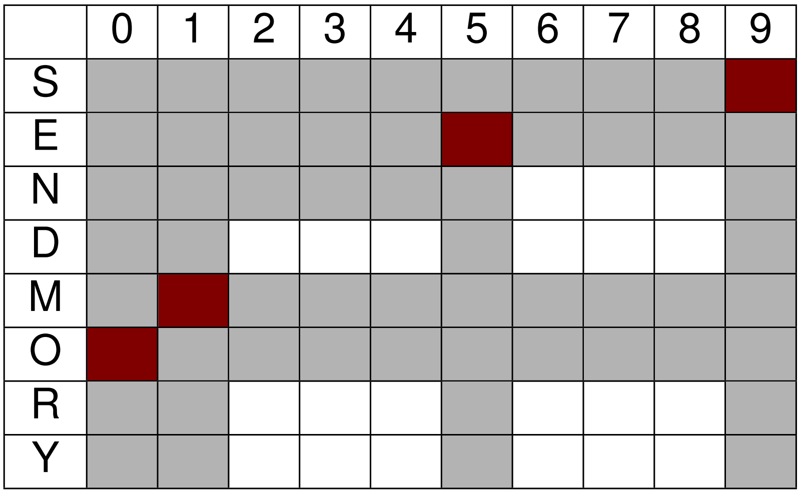
Return to last open choice, E, and test next value



Assignment $E = 5$

	0	1	2	3	4	5	6	7	8	9
S										
E					-	✸	-	-		
N										
D										
M										
O										
R										
Y										

Propagation of alldifferent



$$N \neq 5, N \geq 6$$

Propagation of equality

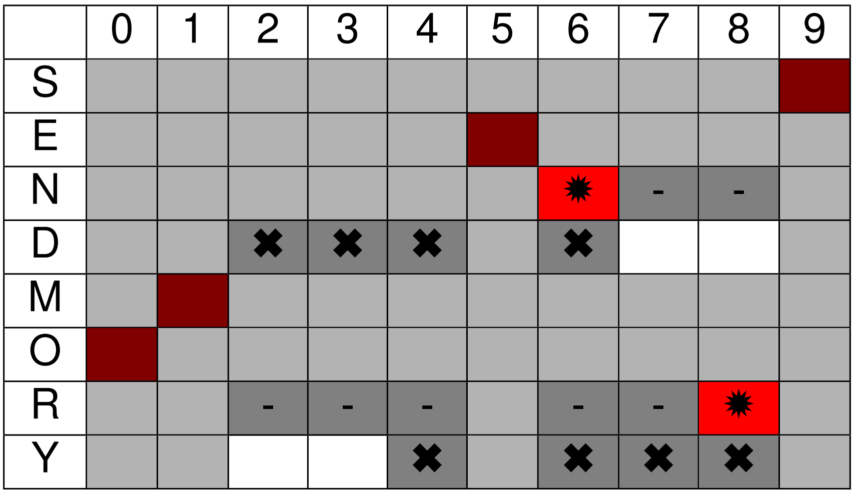
$$91 * 5 + 10 * R^{2..8} + D^{2..8} = 90 * N^{6..8} + Y^{2..8}$$

$$\underbrace{91 * 5 + 10 * R^{2..8} + D^{2..8}}_{477..543} = \underbrace{90 * N^{6..8} + Y^{2..8}}_{542..728}$$

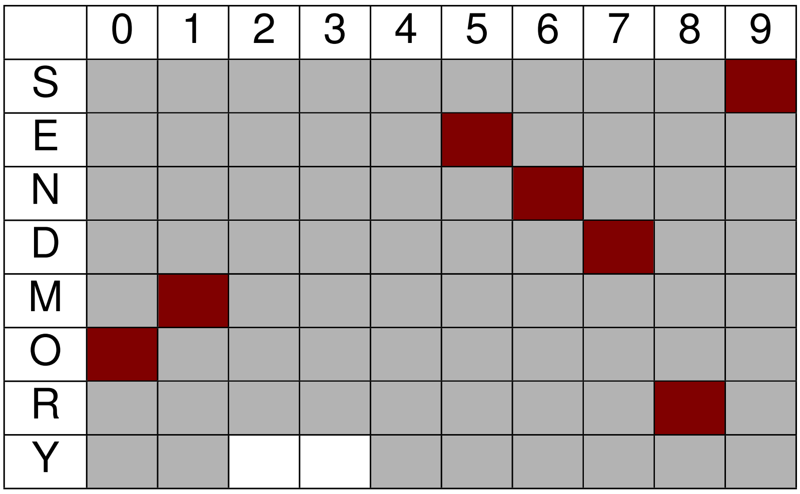
$$\underbrace{91 * 5 + 10 * R^{2..8} + D^{2..8} = 90 * N^{6..8} + Y^{2..8}}_{542..543}$$

$$N = 6, Y \in \{2, 3\}, R = 8, D \in \{7..8\}$$

Result of equality propagation



Propagation of `alldifferent`



$D = 7$

Propagation of equality

$$91 * 5 + 10 * 8 + 7 = 90 * 6 + Y^{2..3}$$

$$\underbrace{91 * 5 + 10 * 8 + 7}_{542} = \underbrace{90 * 6 + Y^{2..3}}_{542..543}$$

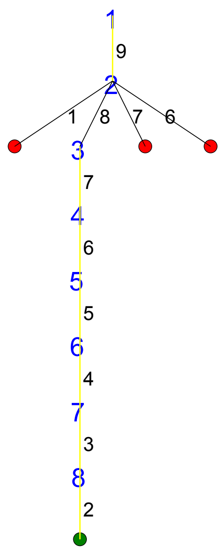
$$\underbrace{91 * 5 + 10 * 8 + 7 = 90 * 6 + Y^{2..3}}_{542}$$

$$Y = 2$$

Last propagation step

	0	1	2	3	4	5	6	7	8	9
S										
E										
N										
D										
M										
O										
R										
Y			✹	-						

Complete Search Tree



Solution

	9	5	6	7
+	1	0	8	5
1	0	6	5	2

Topics introduced

- Finite Domain Solver in ECLiPSe, `ic` library
- Models and Programs
- Constraint Propagation and Search
- Basic constraints: linear arithmetic, `alldifferent`, disequality
- Built-in search: `labeling`
- Visualizers for variables, constraints and search

Lessons Learned

- Constraint models are expressed by variables and constraints.
- Problems can have many different models, which can behave quite differently. Choosing the best model is an art.
- Constraints can take many different forms.
- Propagation deals with the interaction of variables and constraints.
- It removes some values that are inconsistent with a constraint from the domain of a variable.
- Constraints only communicate via shared variables.

Lessons Learned

- Propagation usually is not sufficient, search may be required to find a solution.
- Propagation is data driven, and can be quite complex even for small examples.
- The default search uses chronological depth-first backtracking, systematically exploring the complete search space.
- The search choices and propagation are interleaved, after every choice some more propagation may further reduce the problem.

Alternative 1

- Do we need the constraint "Numbers do not begin with a zero"?
- This is not given explicitely in the problem statement
- Remove disequality constraints from program
- Previous solution is still a solution
- Does it change propagation?
- Does it have more solutions?

Program without Disequality

Listing 1: Alternative 1

```
:-module(alternative1).
:-export(sendmory/1).
:-lib(ic).

sendmory(L):-
        L = [S,E,N,D,M,O,R,Y],
        L :: 0..9,
        alldifferent(L),
        1000*S + 100*E + 10*N + D +
        1000*M + 100*O + 10*R + E #=
        10000*M + 1000*O + 100*N + 10*E + Y,
        labeling(L).
```


After Setup without Disequality

	0	1	2	3	4	5	6	7	8	9
S										
E										
N										
D										
M										
O										
R										
Y										

Setup Comparison

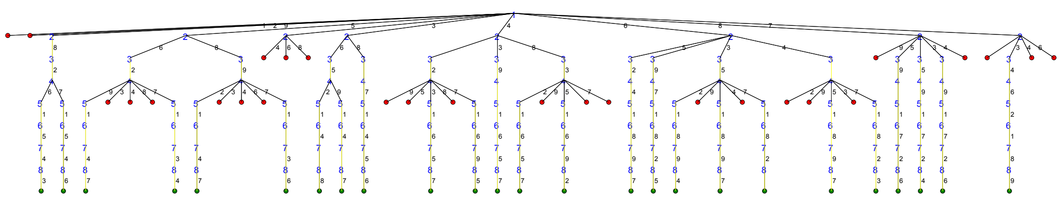
original

	0	1	2	3	4	5	6	7	8	9
S										
E										
N										
D										
M										
O										
R										
Y										

alternative 1

	0	1	2	3	4	5	6	7	8	9
S										
E										
N										
D										
M										
O										
R										
Y										

Search Tree: Many Solutions



Note:

- Not just a different model, solving a different problem!
- Often we can choose which problem we want to solve
  - Which constraints to include
  - What to ignore
- In this case not acceptable

◂ Choice of Model

Alternative 2

- Large equality difficult to understand by humans
- Replace with multiple, simpler equations
- Linked by carry variables (0/1)
- Should produce same solutions
- Does it give same propagation?

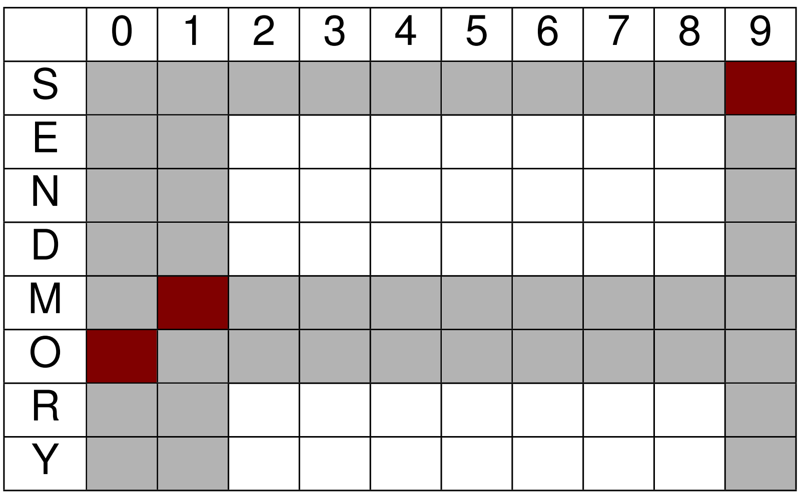
$$\begin{array}{ccccc} & S & E & N & D \\ + & M & O & R & E \\ +C5 & C4 & C3 & C2 & \\ \hline M & O & N & E & Y \end{array}$$

Carry Variables with Multiple Equations

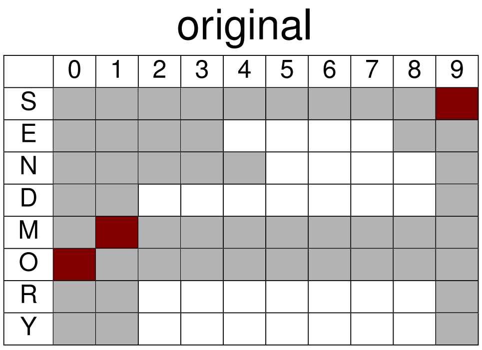
```
:-module(alternative2),export(sendmory/1),lib(ic).
sendmory(L):-⇨ same as before
    L=[S,E,N,D,M,O,R,Y],L :: 0..9,
    [C2,C3,C4,C5] :: 0..1, ⇨ new
    alldifferent(L),
    S #\= 0,M #\= 0,
    M #= C5,
    S+M+C4 #= 10*C5+O,
    E+O+C3 #= 10*C4+N,
    N+R+C2 #= 10*C3+E,
    D+E    #= 10*C2+Y,
    labeling(L).
```

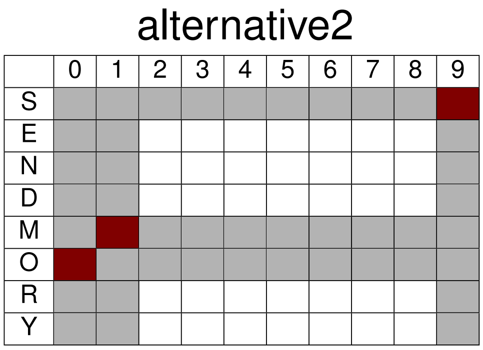
	S	E	N	D
+	M	O	R	E
+*C5*	*C4*	*C3*	*C2*	
M	O	N	E	Y

With Carry Variables: After Setup

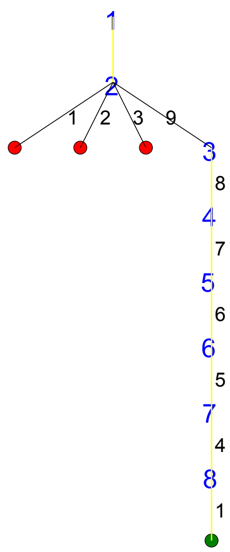


Setup Comparison



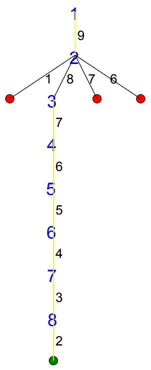


Search Tree: First Solution

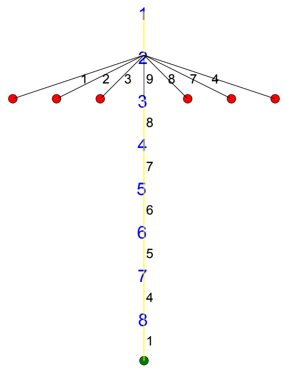


Comparison

Single Equation



Multiple Equations



Observations

- This is solving the original problem
- Search tree slightly bigger
- Caused here by missing interaction of equations
- And repeated variables
- But: Introducing auxiliary variables not always bad!

◂ Choice of Model

Exercises

1. Does the reasoning for the equality constraints that we have presented remove all inconsistent values? Consider the constraint Y=2*X.
2. Why is it important to remove multiple occurences of the same variable from an equality constraint? Give an example!
3. Solve the puzzle DONALD+GERALD=ROBERT. What is the state of the variables before the search, after the initial constraint propagation?
4. Solve the puzzle Y*WORRY = DOOOOD. What is different?
5. (extra credit) How would you design a program that finds new crypt-arithmetic puzzles? What makes a good puzzle?

Chapter 5: Global Constraints(Sudoku)

Helmut Simonis

Cork Constraint Computation Centre
Computer Science Department
University College Cork
Ireland

ECLiPSe ELearning Overview

Outline

What we want to introduce

- Global Constraints
  - Powerful modelling abstractions
  - Non-trivial propagation
- Consistency Levels
  - Tradeoff between speed and propagation
  - Characterisation of reasoning power
- Example: Alldifferent
  - 3 variants shown

Methodology

- Evaluation on Sudoku puzzle
- Comparing
  - Initial setup
  - Search
  - Performance
- Explaining reasoning inside constraint
- Link to general classification of global constraints

Problem Definition

Sudoku

Fill in numbers from 1 to 9 so that each row, column and block contain each number exactly once

1	1 2 3 4 5 6 7 8 9	2	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	3	4	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	2	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	5	6
1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	7	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	2	6	8	1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9	9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	2	1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9	5	4	7	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9
6	4	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	8	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	3	1	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	7	1 2 3 4 5 6 7 8 9	1

1	2	3	4	5	6	7	8	9
6	4	9	7	8	2	1	5	3
8	5	7	1	3	9	4	6	2
7	1	5	6	9	3	2	4	8
4	9	2	8	1	7	6	3	5
3	6	8	5	2	4	9	1	7
2	8	1	9	4	5	3	7	6
5	3	6	2	7	1	8	9	4
9	7	4	3	6	8	5	2	1

Model

- A variable for each cell, ranging from 1 to 9
- A 9x9 matrix of variables describing the problem
- Preassigned integers for the given hints
- alldifferent constraints for each row, column and 3x3 block

Reminder: alldifferent

- Argument: list of variables
- Meaning: variables are pairwise different
- Reasoning: Forward Checking (FC)
  - When variable is assigned to value, remove the value from all other variables
  - If a variable has only one possible value, then it is assigned
  - If a variable has no possible values, then the constraint fails
  - Constraint is checked whenever one of its variables is assigned
  - Equivalent to decomposition into binary disequality constraints

Declarations

```
:-module(sudoku).
:-export(top/0).
:-lib(ic).

top:-
    problem(Matrix),
    model(Matrix),
    writeln(Matrix).
```

Data

```
problem([]([](4, _, 8, _, _, _, _, _, _),
           [](_, _, _, 1, 7, _, _, _, _),
           [](_, _, _, _, 8, _, _, 3, 2),
           [](_, _, 6, _, _, 8, 2, 5, _),
           [](_, 9, _, _, _, _, _, 8, _),
           [](_, 3, 7, 6, _, _, 9, _, _),
           [](2, 7, _, _, 5, _, _, _, _),
           [](_, _, _, _, 1, 4, _, _, _),
           [](_, _, _, _, _, _, 6, _, 4))).
```

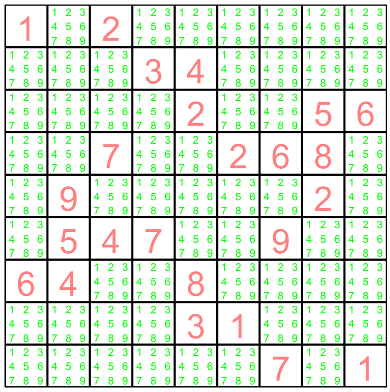

Problem
Program
Initial Propagation (Forward Checking)
Improved Reasoning
Search
Lessons Learned

Main Program

```
model(Matrix):-
     Matrix[1..9,1..9] :: 1..9,
     (for(I,1,9),
      param(Matrix) do
         alldifferent(Matrix[I,1..9]),
         alldifferent(Matrix[1..9,I])
     ),
     (multifor([I,J],[1,1],[7,7],[3,3]),
      param(Matrix) do
         alldifferent(flatten(Matrix[I..I+2,J..J+2]))
     ),
     flatten_array(Matrix,List),
     labeling(List).
```

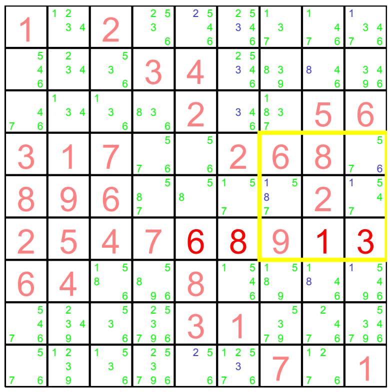
Domain Visualizer

- Problem shown as matrix
- Each cell corresponds to a variable
- Instantiated: Shows integer value (large)
- Uninstantiated: Shows values in domain

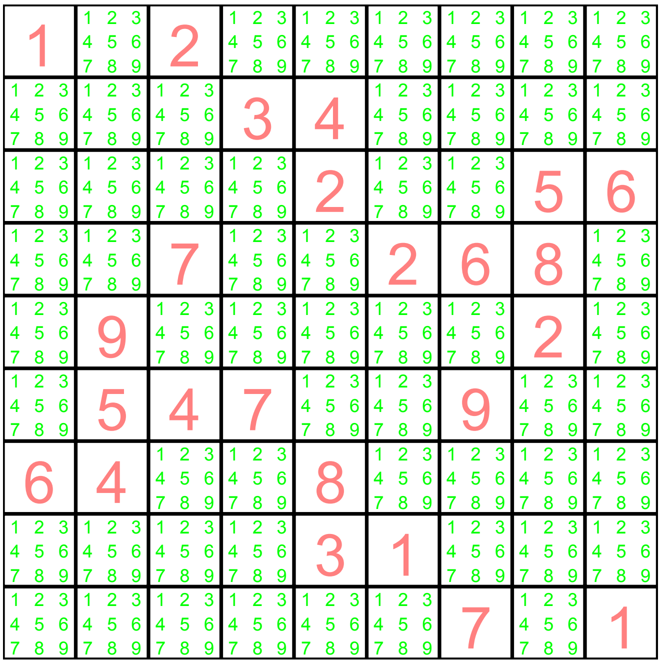


Constraint Visualizer

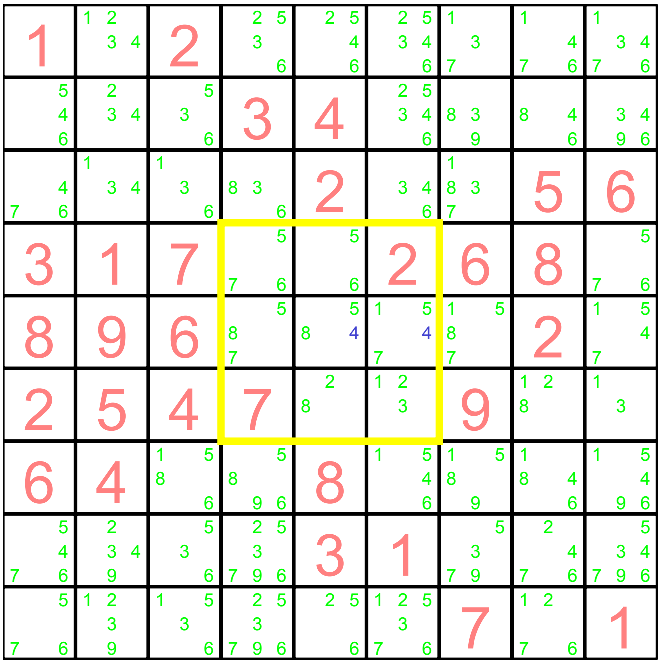
- Problem shown as matrix
- Currently active constraint highlighted
- Values removed at this step shown in blue
- Values assigned at this step shown in red



Initial State (Forward Checking)



Propagation Steps (Forward Checking)



After Setup (Forward Checking)

1	1 2 3 4	**2**	2 5 3 6	5 4 6	2 5 4 6	1 3 7	1 4 7 6	3 4 7 6
5 4 6	2 3 4	5 3 6	**3**	**4**	2 5 4 6	8 3 9	4 6	3 4 9 6
4 7 6	1 3 4	1 3 6	8 3 6	**2**	4 6	1 8 3 7	**5**	**6**
3	**1**	**7**	5 7 6	5 6	**2**	**6**	**8**	5 7
8	**9**	**6**	5 8 7	5 8	1 5 7	5 7	**2**	5 4 7
2	**5**	**4**	**7**	**6**	**8**	**9**	**1**	**3**
6	**4**	1 5 8 6	5 9 6	**8**	5 4 6	1 5 9	1 6	5 9 6
5 4 6	3 4 9	5 3 6	2 5 7 9 6	**3**	**1**	5 3 7 9	2 7 6	5 3 7 9 6
5 6	1 3 9	1 5 3 6	2 5 7 9 6	5 6	2 5 7 6	**7**	1 2 7 6	**1**

Can we do better?

- The alldifferent constraint is missing propagation
  - How can we do more propagation?
  - Do we know when we derive all possible information from the constraint?
- Constraints only interact by changing domains of variables

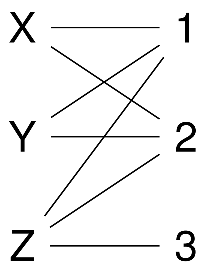
A Simpler Example

```
:-lib(ic).

top:-
    X :: 1..2,
    Y :: 1..2,
    Z :: 1..3,
    alldifferent([X,Y,Z]),
    writeln([X,Y,Z]).
```


Using Forward Checking

- No variable is assigned
- No reduction of domains
- But, values 1 and 2 can be removed from Z
- This means that Z is assigned to 3

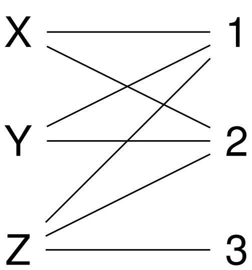
Visualization of alldifferent as Graph



- Show problem as graph with two types of nodes
  - Variables on the left
  - Values on the right
- If value is in domain of variable, show link between them
- This is called a *bipartite* graph

Cork Constraint Computation Centre

A Simpler Example

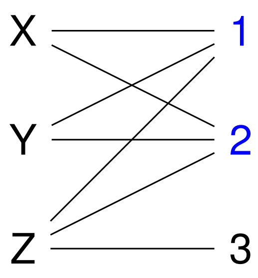


Value Graph for

```
X :: 1..2,

Y :: 1..2,

Z :: 1..3
```

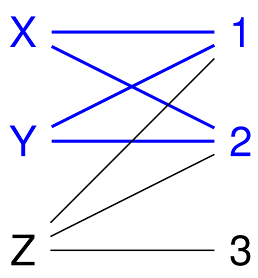
Cork Constraint Computation Centre

A Simpler Example



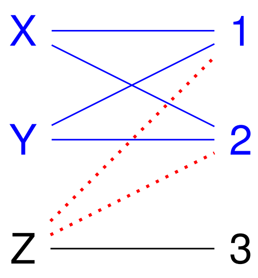
Check interval [1,2]

A Simpler Example



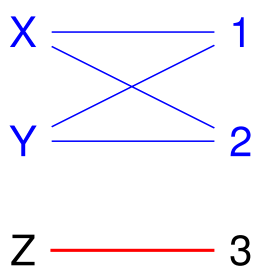
- Find variables completely contained in interval
- There are two: X and Y
- This uses up the capacity of the interval

A Simpler Example



No other variable can use that interval

A Simpler Example



Only one value left in domain of Z, this can be assigned

Idea (Hall Intervals)

- Take each interval of possible values, say size N
- Find all K variables whose domain is completely contained in interval
- If $K > N$ then the constraint is infeasible
- If $K = N$ then no other variable can use that interval
- Remove values from such variables if their bounds change
- If $K < N$ do nothing
- Re-check whenever domain bounds change

Implementation

- Problem: Too many intervals ($O(n^2)$) to consider
- Solution:
  - Check only those intervals which update bounds
  - Enumerate intervals incrementally
  - Starting from lowest(highest) value
  - Using sorted list of variables
- Complexity: $O(n \log(n))$ in standard implementations
- Important: Only looks at min/max bounds of variables

Bounds Consistency

Definition

A constraint achieves *bounds consistency*, if for the lower and upper bound of every variable, it is possible to find values for all other variables between their lower and upper bounds which satisfy the constraint.

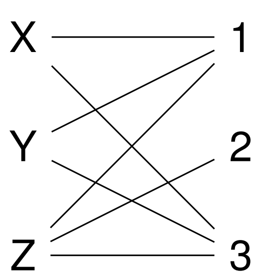
Can we do better?

- Bounds consistency only considers min/max bounds
- Ignores “holes” in domain
- Sometimes we can improve propagation looking at those holes

Another Simple Example

```
:-lib(ic).

top:-
    X :: [1,3],
    Y :: [1,3],
    Z :: 1..3,
    alldifferent([X,Y,Z]),
    writeln([X,Y,Z]).
```

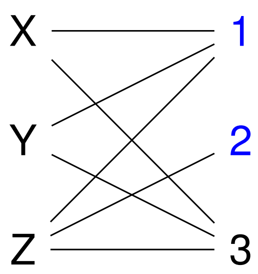

Another Simple Example



Value Graph for

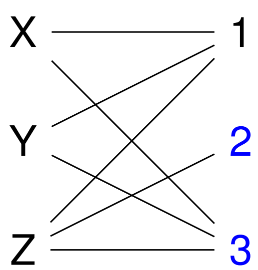
```
X :: [1,3],
Y :: [1,3],
Z :: 1..3
```

Another Simple Example



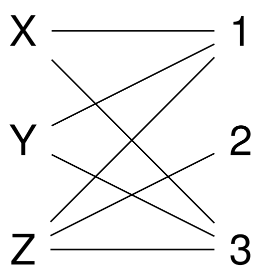
- Check interval [1,2]
- No domain of a variable completely contained in interval
- No propagation

Another Simple Example



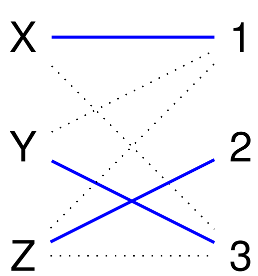
- Check interval [2,3]
- No domain of a variable completely contained in interval
- No propagation

Another Simple Example



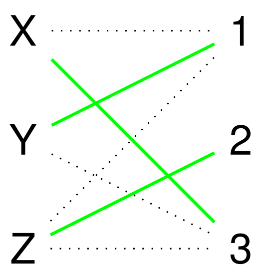
But, more propagation is possible, there are only two solutions

Another Simple Example



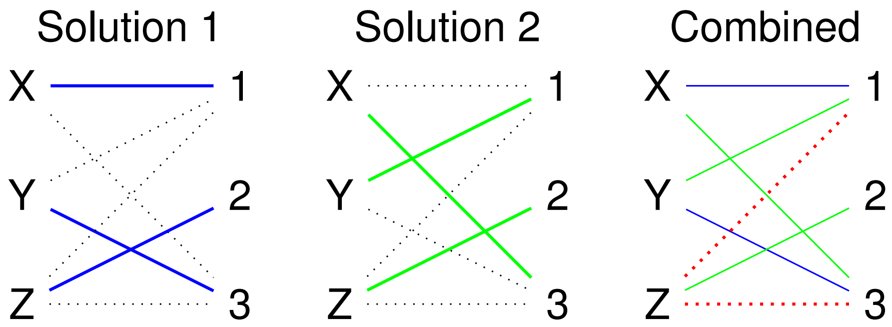
Solution 1: assignment in blue

Another Simple Example



Solution 2: assignment in green

Another Simple Example



Combining solutions shows that Z=1 and Z=3 are not possible. Can we deduce this without enumerating solutions?

Solutions and maximal matchings

- A *Matching* is subset of edges which do not coincide in any node
- No matching can have more edges than number of variables
- Every solution corresponds to a *maximal matching* and vice versa
- If a link does not belong to some maximal matching, then it can be removed

Implementation

- Possible to compute all links which belong to some matching
  - Without enumerating all of them!
- Enough to compute **one** maximal matching
- Requires algorithm for *strongly connected components*
- Extra work required if more values than variables
- All links (values in domains) which are not supported can be removed
- Complexity: $O(n^{1.5}d)$

Domain Consistency

Definition

A constraint achieves *domain consistency*, if for every variable and for every value in its domain, it is possible to find values in the domains of all other variables which satisfy the constraint.

- Also called *generalized arc consistency (GAC)*
- or *hyper arc consistency*

Can we still do better?

- NO! This extracts all information from this one constraint
- We could perhaps improve speed, but not propagation
- But possible to use different model
- Or model interaction of multiple constraints

Should all constraints achieve domain consistency?

- Domain consistency is usually more expensive than bounds consistency
  - Overkill for simple problems
  - Nice to have choices
- For some constraints achieving domain consistency is NP-hard
  - We have to live with more restricted propagation

Improved Propagation in ECLiPSe

- `ic_global` library bounds consistent version
- `ic_global_gac` library domain consistent version
- Choose which version to use by using module annotation
- Choice can be passed as parameter

Declarations

```
:-module(sudoku).
:-export(top/0).
:-lib(ic).
:-lib(ic_global).
:-lib(ic_global_gac).

top:-
     problem(Matrix),
     model(ic_global,Matrix),
     writeln(Matrix).
```


Main Program

```
model(Method,Matrix):-
     Matrix[1..9,1..9] :: 1..9,
     (for(I,1,9),
      param(Method,Matrix) do
          Method:alldifferent(Matrix[I,1..9]),
          Method:alldifferent(Matrix[1..9,I])
     ),
     (multifor([I,J],[1,1],[7,7],[3,3]),
      param(Method,Matrix) do
          Method:alldifferent(flatten(Matrix[I..I+2,
                                              J..J+2]))
     ),
     flatten_array(Matrix,List),labeling(List).
```

Initial State (Bounds Consistency)

1	1 2 3 4 5 6 7 8 9	2	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	3	4	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	2	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	5	6
1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	7	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	2	6	8	1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9	9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	2	1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9	5	4	7	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9
6	4	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	8	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	3	1	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	7	1 2 3 4 5 6 7 8 9	1

Problem
Program
Initial Propagation (Forward Checking)
Improved Reasoning
Search
Lessons Learned

Bounds Consistency
Domain Consistency
Comparison

Propagation Steps (Bounds Consistency)

1	1 2 3 4	2	3	4	2 5 3 4 6	1 3 7	1 4 7 6	1 3 4 7 6
5 4 6	2 3 4	5 3 6	5	6	2 5 3 4 6	8 3 9	8 4 6	3 4 9 6
6	4	1 3 6	1	2	7	1 8 3	8	9
5	1	4	5 7 6	5 6	2	9	3	5 7 6
3	7	9	5 8 7	5 8 4	5	1 5 8 7	2	4
2	8	6	4	9	3	7	1	5
9	6	1	5 8 9 6	3	4	1 5 8 9	1 8 4 6	1 5 4 9 6
4	2 3 4 9	5 3 6	2 5 3 7 9 6	5	1	5 3 7 9	2 4 7 6	5 3 4 7 9 6
5 6	1 2 3 9	1 5 3 6	2 5 3 7 9 6	2 5 6	6	4	1 2 7 6	1

After Setup (Bounds Consistency)

1	1 2	2	3	4	2 3	1 4	1 4 5	4 5
3 5	2 6	3 5	5	6	2 3	7 6 8	9 5	6 8 5
6	4	1 6	1	2	7	1 6	8	9
5	1	4	3 4 5	3 5	2	9	3	3 4
3	7	9	3 4	3 7	5	3 4	2	4
2	8	6	4	9	3	7	1	5
9	6	1	3 8 5	3	4	1 3 8	1 5	3 8 5
4	6 8	3 5	2 3 8 5	5	1	3 6 4 8	2 4 5	3 6 4 8 5
3 5	1 6 8	1 6	2 8	3 5	6	4	1 2	1

Initial State (Domain Consistency)

1	1 2 3 4 5 6 7 8 9	2	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	3	4	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	2	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	5	6
1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	7	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	2	6	8	1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9	9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	2	1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9	5	4	7	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9
6	4	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	8	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	3	1	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	7	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9

Propagation Steps (Domain Consistency)

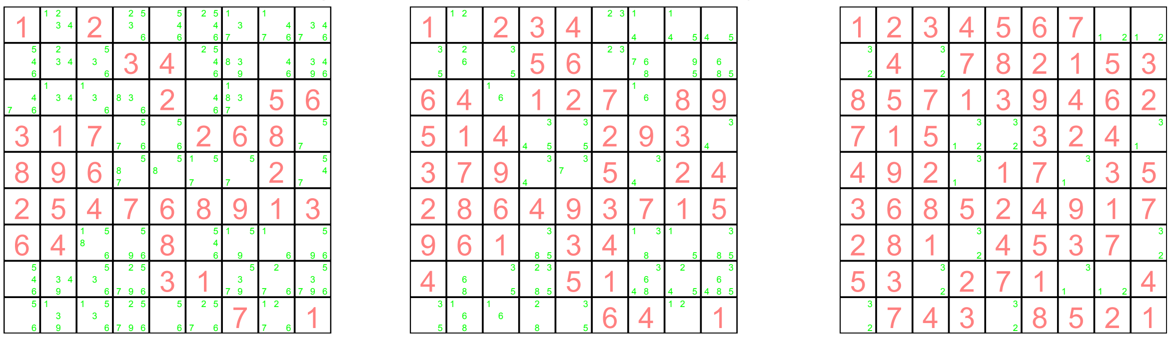
1	2	3	4	5	6	7	1 2 3 4	1 5 2 3 4
6 2 4	4	6 5 4	7	8	2	1	5	3
8	5	7	1	3	9	4	6	2
7	1	5	6 3 4	6 4	3	2	4	6 3 4
4	9	2	6 3	1	7	1 6 7 3	3	5
3	6	8	5	2	4	9	1	7
2	8	1	6 8 4	4	5	3	7	1 6 2 8 4
5	3	6 5 4	2	7	1	6 5 3 8	9 2 3 4	4
6 4	7	4	3	9 6 4	8	5	2	1

After Setup (Domain Consistency)

1	2	3	4	5	6	7	1 2	1 2
3 2	4	3 2	7	8	2	1	5	3
8	5	7	1	3	9	4	6	2
7	1	5	3 1 2	3 2	3	2	4	3 1
4	9	2	3 1	1	7	3 1	3	5
3	6	8	5	2	4	9	1	7
2	8	1	3 2	4	5	3	7	3 2
5	3	3 2	2	7	1	3 1	1 2	4
3 2	7	4	3	3 2	8	5	2	1

Comparison

Forward Checking | Bounds Consistency | Domain Consistency



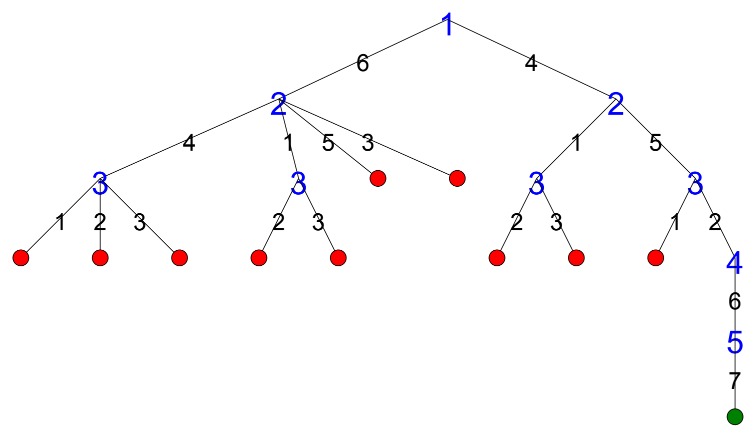
Typical?

- This does not always happen
- Sometimes, two methods produce same amount of propagation
- Possible to predict in certain special cases
- In general, tradeoff between speed and propagation
- Not always fastest to remove inconsistent values early
- But often required to find a solution at all

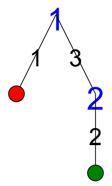
Simple search routine

- Enumerate variables in given order
- Try values starting from smallest one in domain
- Complete, chronological backtracking

Search Tree (Forward Checking)



Search Tree (Bounds Consistency)



Search Tree (Domain Consistency)

Observations

- Search tree much smaller for bounds/domain consistency
- Does not always happen like this
- Smaller tree = Less execution time
- Less reasoning = Less execution time
- Problem: Finding best balance
- For Sudoku: not good enough, should not require any search!

Solution

1	2	3	4	5	6	7	8	9
6	4	9	7	8	2	1	5	3
8	5	7	1	3	9	4	6	2
7	1	5	6	9	3	2	4	8
4	9	2	8	1	7	6	3	5
3	6	8	5	2	4	9	1	7
2	8	1	9	4	5	3	7	6
5	3	6	2	7	1	8	9	4
9	7	4	3	6	8	5	2	1

Global Constraints

- Powerful modelling abstractions
- Efficient reasoning

Problem
Program
Initial Propagation (Forward Checking)
Improved Reasoning
Search
Lessons Learned

Consistency Levels

- Defined levels of propagation
- Tradeoff speed/reasoning
- Characterisation of power of constraint

Alldifferent Variants

- Forward Checking
  - Only reacts when variables are assigned
  - Equivalent to decomposition into binary constraints
- Bounds Consistency
  - Typical best compomise speed/reasoning
  - Works well if no holes in domain
- Domain Consistency
  - Extracts all information from single constraint
  - Cost only justified for very hard problems

End of Chapter 5

Thank you!

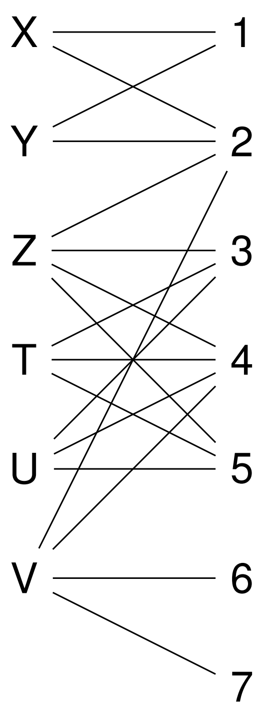
Some optional material follows

Bigger Example

```
:-lib(ic).
:-lib(ic_global_gac).

top:-
    [X,Y] :: 1..2,
    Z :: 2..5,
    [T,U] :: 3..5,
    V :: [2,4,6,7],
    ic_global_gac:alldifferent([X,Y,Z,T,U,V]).
```


Complete Example: Domain Consistent Alldifferent
Generic Model
Exercises

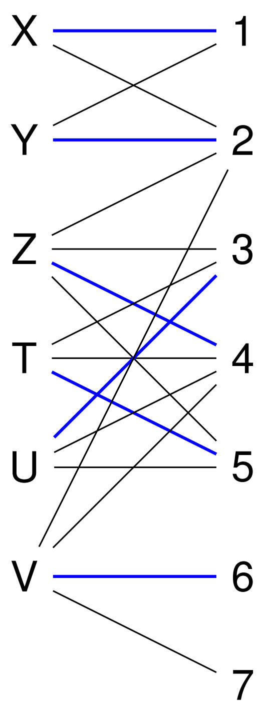
Making constraint domain consistent



Problem shown as bipartite graph

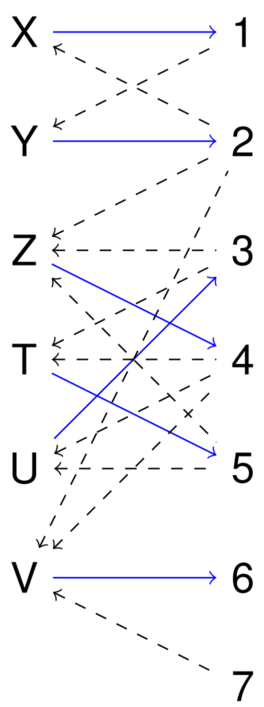
Helmut Simonis Global Constraints

Making constraint domain consistent



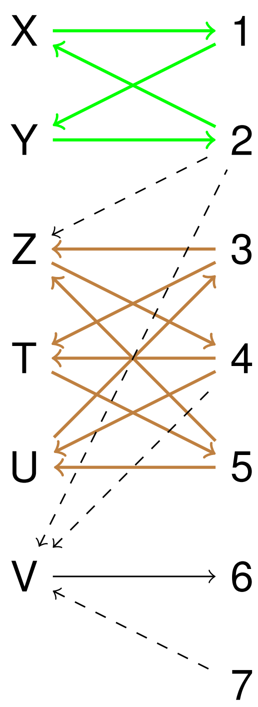
Find maximal matching (in blue)

Making constraint domain consistent



Orient graph (edges in matching from variables to values, all others from values to variables), mark edges in matching

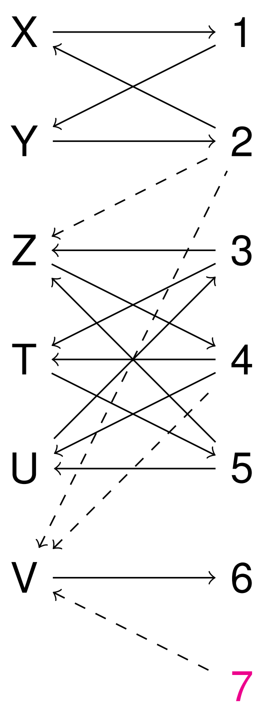
Making constraint domain consistent



Find strongly connected components (green and brown), mark their edges

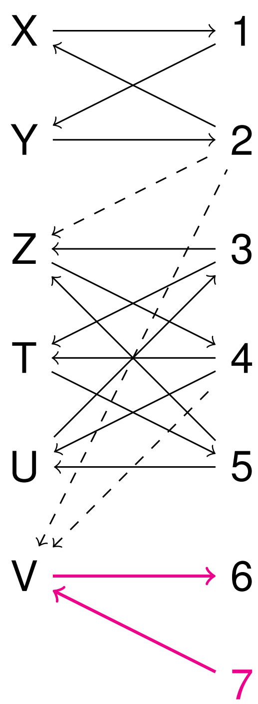
Complete Example: Domain Consistent Alldifferent
Generic Model
Exercises

Making constraint domain consistent



Find unmatched value nodes (here node 7, magenta)

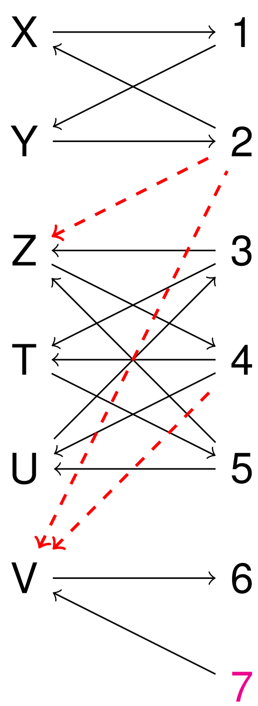
Making constraint domain consistent



Find alternating paths from such nodes (in magenta), mark their edges

Complete Example: Domain Consistent Alldifferent
Generic Model
Exercises

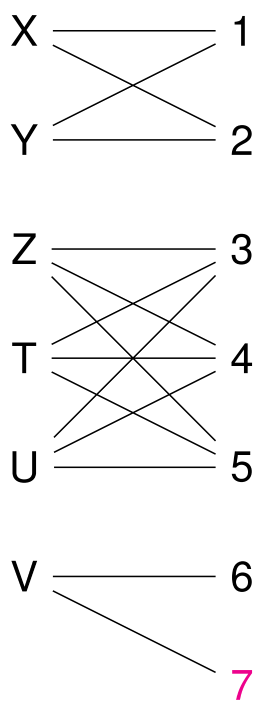
Making constraint domain consistent



All unmarked edges can be removed

Helmut Simonis Global Constraints

Making constraint domain consistent

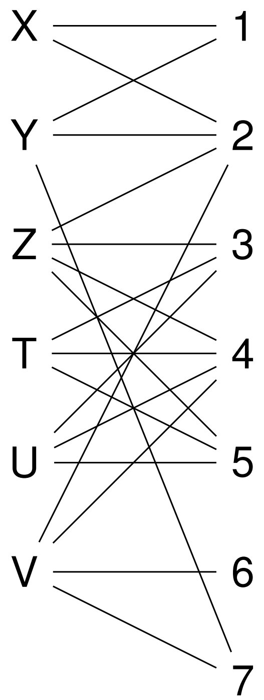


Resulting graph, constraint is domain consistent

Extended Example

```
:-lib(ic).
:-lib(ic_global_gac).

top:-
    X :: 1..2,
    Y :: [1,2,7],
    Z :: 2..5,
    [T,U] :: 3..5,
    V :: [2,4,6,7],
    ic_global_gac:alldifferent([X,Y,Z,T,U,V]).
```

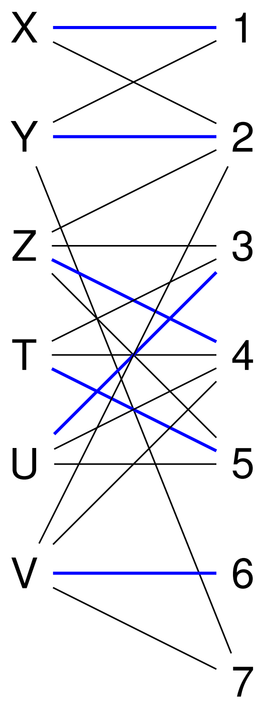
No propagation in expanded example



Problem shown as bipartite graph

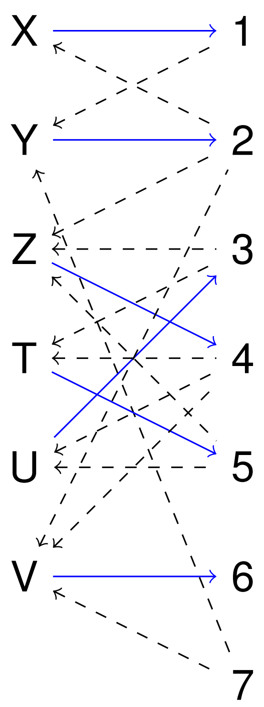
Complete Example: Domain Consistent Alldifferent
Generic Model
Exercises

No propagation in expanded example



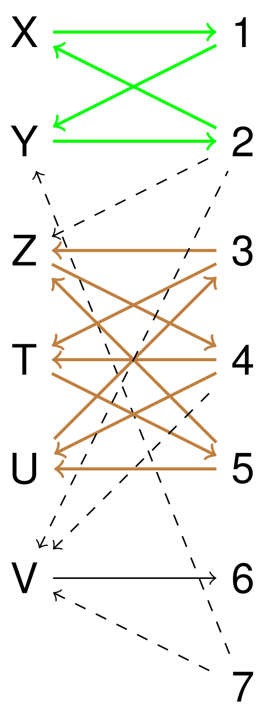
Find maximal matching (in blue)

No propagation in expanded example



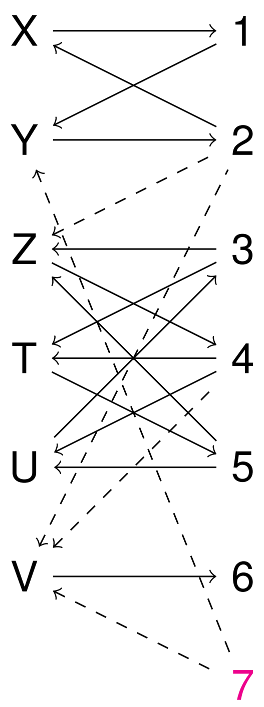
Orient graph (edges in matching from variables to values, all others from values to variables), mark edges in matching

No propagation in expanded example



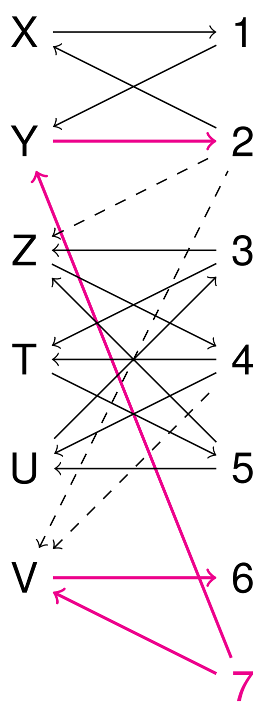
Find strongly connected components (green and brown), mark their edges

No propagation in expanded example



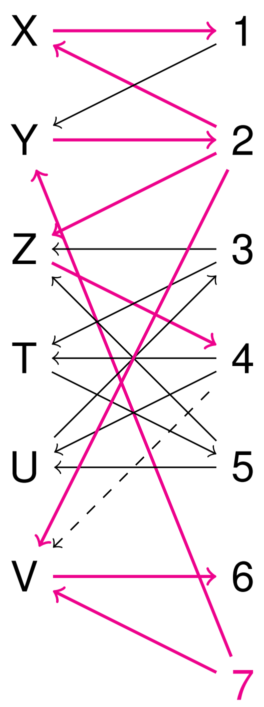
Find unmatched value nodes (here node 7, magenta)

No propagation in expanded example



Find alternating paths from such nodes (in magenta), mark their edges

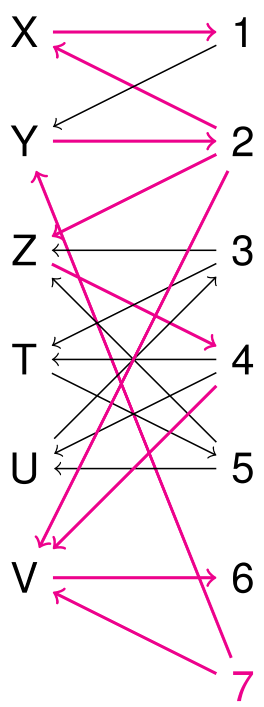
No propagation in expanded example



Continue with alternating paths

Complete Example: Domain Consistent Alldifferent
Generic Model
Exercises

No propagation in expanded example



Continue with alternating paths, all edges marked, no propagation, constraint is domain consistent

Observation

- A lot of effort for no propagation
- Problem: Slows down search without any upside
- Constraint is woken every time any domain is changed
- How often does the constraint do actual pruning?

Complete Example: Domain Consistent Alldifferent
Generic Model
Exercises

Generalize Program for different sizes

- How to generalize program for different sizes (4,9,16,25,36...)
- Add parameter R (Order, number of blocks in a row/column)
- Size N is square of R
- Remove explicit integer bounds by expressions
- Useful to do this change as rewriting of working program

Helmut Simonis Global Constraints

Main Program

```
model(R,M,Matrix):-
    N is R*R,Matrix[1..N,1..N] :: 1..N,
    (for(I,1,N),
     param(N,M,Matrix) do
        M:alldifferent(Matrix[I,1..N]),
        M:alldifferent(Matrix[1..N,I])
    ),
    (multifor([I,J],[1,1],[N-R+1,N-R+1],[R,R]),
     param(R,M,Matrix) do
        M:alldifferent(flatten(Matrix[I..I+R-1,
                                      J..J+R-1]))
    ),
    flatten_array(Matrix,List),labeling(List).
```


Exercises

1

Chapter 6: Search Strategies (N-Queens)

Helmut Simonis

Cork Constraint Computation Centre
Computer Science Department
University College Cork
Ireland

ECLiPSe ELearning Overview

Licence

Outline

What we want to introduce

- Importance of search strategy, constraints alone are not enough
- Dynamic variable ordering exploits information from propagation
- Variable and value choice
- Hard to find strategy which works all the time
- `search` builtin, flexible search abstraction
- Different way of improving stability of search routine

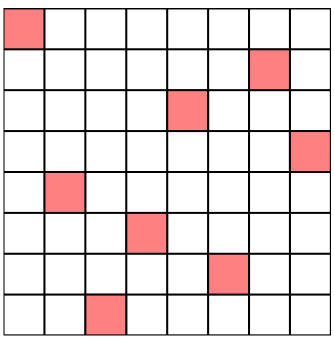
Example Problem

- N-Queens puzzle
- Rather weak constraint propagation
- Many solutions, limited number of symmetries
- Easy to scale problem size

Problem Definition

8-Queens

Place 8 queens on an 8×8 chessboard so that no queen attacks another. A queen attacks all cells in horizontal, vertical and diagonal direction. Generalizes to boards of size $N \times N$.



Solution for board size 8×8

A Bit of History

- This is a rather old puzzle
- Dudeney (1917) cites Nauck (1850) as source
- Certain solutions for all sizes can be constructed, this is not a hard problem
- Long history in AI and CP papers
- Important: Haralick and Elliot (1980) describing the first-fail principle

Basic Model

- Cell based Model
  - A 0/1 variable for each cell to say if it is occupied or not
  - Constraints on rows, columns and diagonals to enforce no-attack
  - N^2 variables, $6N - 2$ constraints
- Column (Row) based Model
  - A 1..N variable for each column, stating position of queen in the column
  - Based on observation that each column must contain exactly one queen
  - N variables, $N^2/2$ binary constraints

Model

$$\text{assign} \quad [X_1, X_2, ...X_N]$$

s.t.

$$\begin{aligned}
\forall 1 \leq i \leq N: &\quad X_i \in 1..N \\
\forall 1 \leq i < j \leq N: &\quad X_i \neq X_j \\
\forall 1 \leq i < j \leq N: &\quad X_i \neq X_j + i - j \\
\forall 1 \leq i < j \leq N: &\quad X_i \neq X_j + j - i
\end{aligned}$$

Main Program (Array Version)

```
:-module(array).
:-export(top/0).
:-lib(ic).

top:-
    nqueen(8,Array), writeln(Array).

nqueen(N,Array):-
    dim(Array,[N]),
    Array[1..N] :: 1..N,
    alldifferent(Array[1..N]),
    noattack(Array,N),
    labeling(Array[1..N]).
```

Generating binary constraints

```
noattack(Array,N):-
    (for(I,1,N-1),
     param(Array,N) do
        (for(J,I+1,N),
         param(Array,I) do
            subscript(Array,[I],Xi),
            subscript(Array,[J],Xj),
            D is I-J,
            Xi #\= Xj+D,
            Xj #\= Xi+D
        )
    ).
```

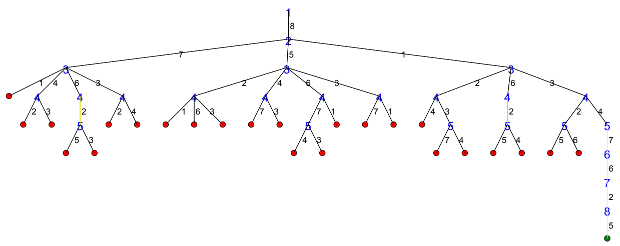

Main Program (List Version)

```
:-module(nqueen).
:-export(top/0).
:-lib(ic).

top:-
    nqueen(8,L), writeln(L).

nqueen(N,L):-
    length(L,N),
    L :: 1..N,
    alldifferent(L),
    noattack(L),
    labeling(L).
```

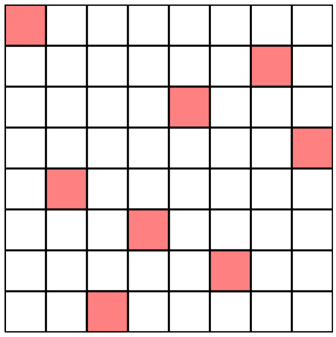
Generating binary constraints

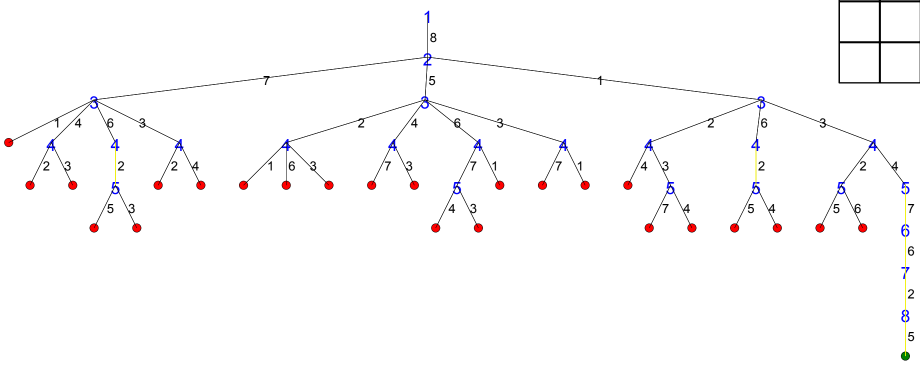
```
noattack([]).
noattack([H|T]):-
    noattack1(H,T,1),
    noattack(T).

noattack1(_,[],_).
noattack1(X,[Y|R],N):-
    X #\= Y+N,
    Y #\= X+N,
    N1 is N+1,
    noattack1(X,R,N1).
```


Default Strategy



First Solution

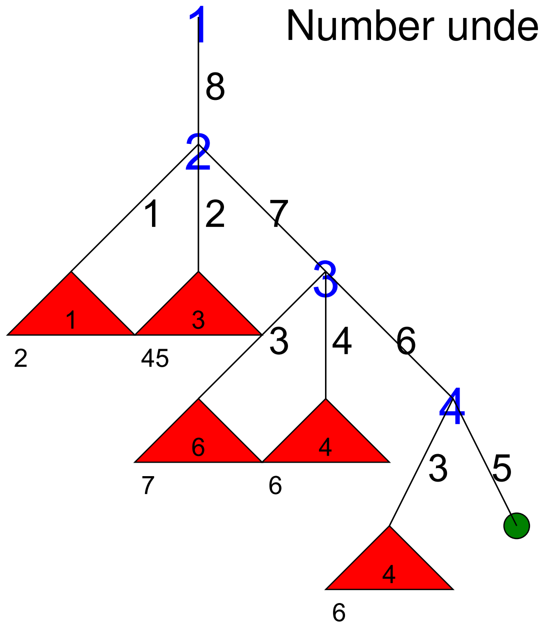




Observations

- Even for small problem size, tree can become large
- Not interested in all details
- Ignore all automatically fixed variables
- For more compact representation abstract failed sub-trees

Compact Representation

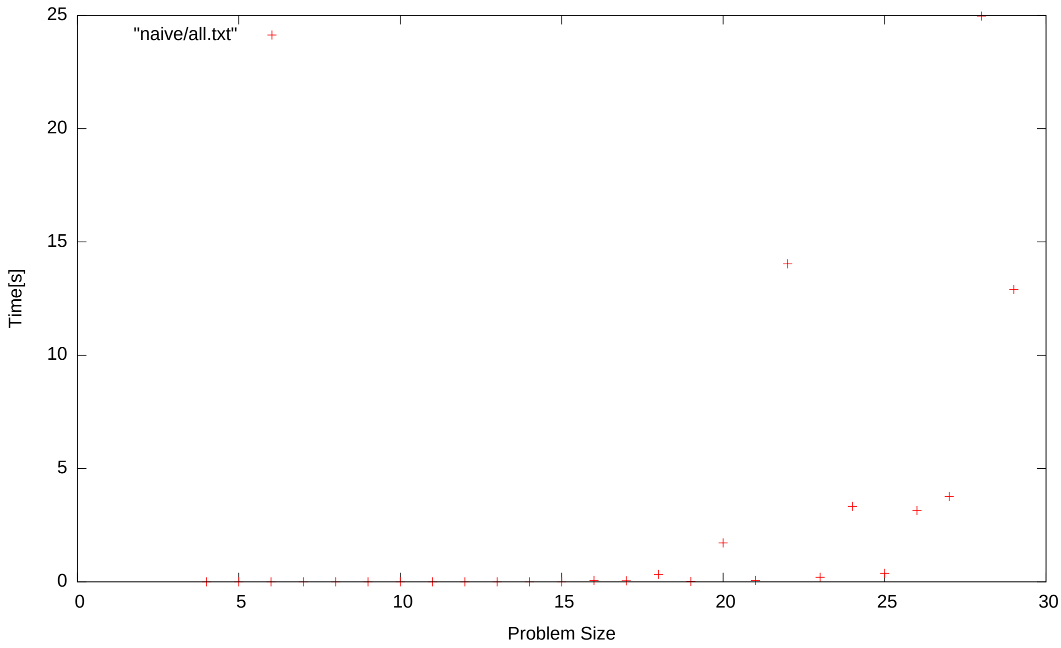


Number inside triangle: Number of choices
Number under triangle: Number of failures

Exploring other board sizes

- How stable is the model?
- Try all sizes from 4 to 100
- Timeout of 100 seconds

Naive Stategy, Problem Sizes 4-100



Observations

- Time very reasonable up to size 20
- Sizes 20-30 times very variable
- Not just linked to problem size
- No size greater than 30 solved within timeout

Possible Improvements

- Better constraint reasoning
  - Remodelling problem with 3 `alldifferent` constraints
  - Global reasoning as described before
  - Not explored here
- Better control of search
  - Static vs. dynamic variable ordering
  - Better value choice
  - Not using complete depth-first chronological backtracking

Static vs. Dynamic Variable Ordering

- Heuristic Static Ordering
  - Sort variables before search based on heuristic
  - Most important decisions
  - Smallest initial domain
- Dynamic variable ordering
  - Use information from constraint propagation
  - Different orders in different parts of search tree
  - Use all information available

First Fail strategy

- Dynamic variable ordering
- At each step, select variable with smallest domain
- Idea: If there is a solution, better chance of finding it
- Idea: If there is no solution, smaller number of alternatives
- Needs tie-breaking method

Caveat

- First fail in many constraint systems have slightly different tie breakers
- Hard to compare result across platforms
- Best to compare search trees, i.e. variable choices in all branches of tree

Modification of Program

```
:-module(nqueen).
:-export(top/0).
:-lib(ic).

top:-
    nqueen(8,L), writeln(L).

nqueen(N,L):-
    length(L,N),
    L :: 1..N,
    alldifferent(L),
    noattack(L),
    search(L,0,first_fail,indomain,complete,[]).
```


The `search` Predicate

- Packaged search library in ic constraint solver
- Provides many different alternative search methods
- Just select a combination of keywords
- Extensible by user

`search` Parameters

`search(L,0,first_fail,indomain,complete,[])`

1. List of variables (or terms, covered later)
2. 0 for list of variables
3. Variable choice, e.g. `first_fail`, `input_order`
4. Value choice, e.g. `indomain`
5. Tree search method, e.g. `complete`
6. Optional argument (or empty) list

Variable Choice

- Determines the order in which variables are assigned
- `input_order` assign variables in static order given
- `first_fail` select variable with smallest domain first
- `most_constrained` like first_fail, tie break based on number of constraints in which variable occurs
- Others, including programmed selection

Value Choice

- Determines the order in which values are tested for selected variables
- `indomain` Start with smallest value, on backtracking try next larger value
- `indomain_max` Start with largest value
- `indomain_middle` Start with value closest to middle of domain
- `indomain_random` Choose values in random order

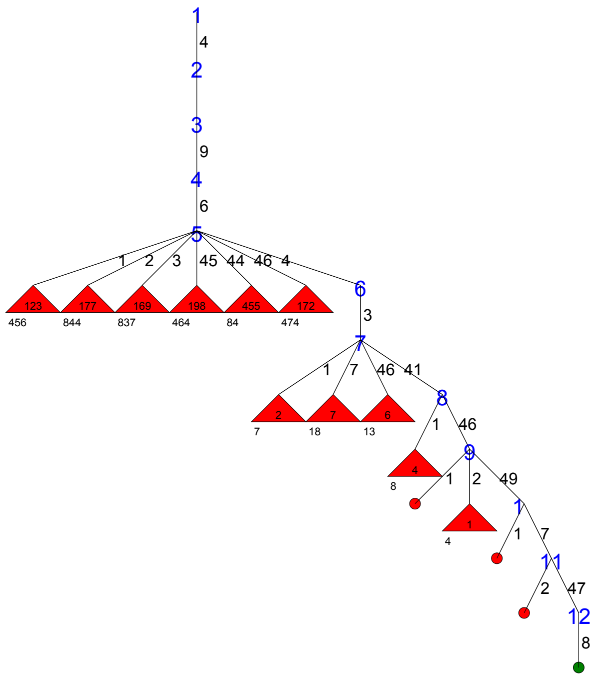
Cork Constraint Computation Centre

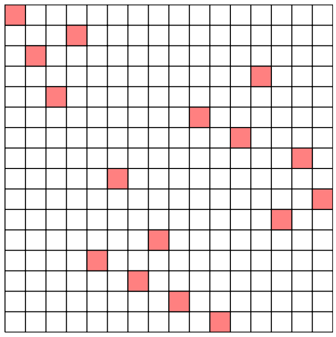
Comparison

- Board size 16x16
- Naive (Input Order) Strategy
- First Fail variable selection

Cork Constraint Computation Centre

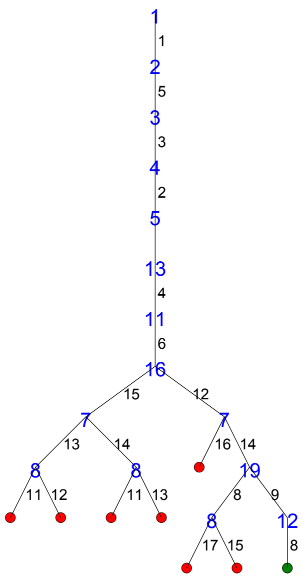
Naive (Input Order) Strategy (Size 16)





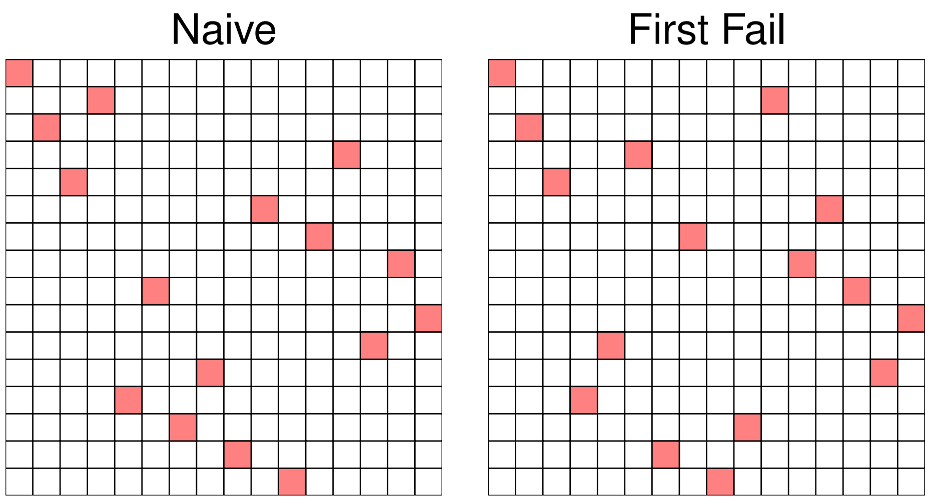
Cork Constraint Computation Centre

FirstFail Strategy (Size 16)



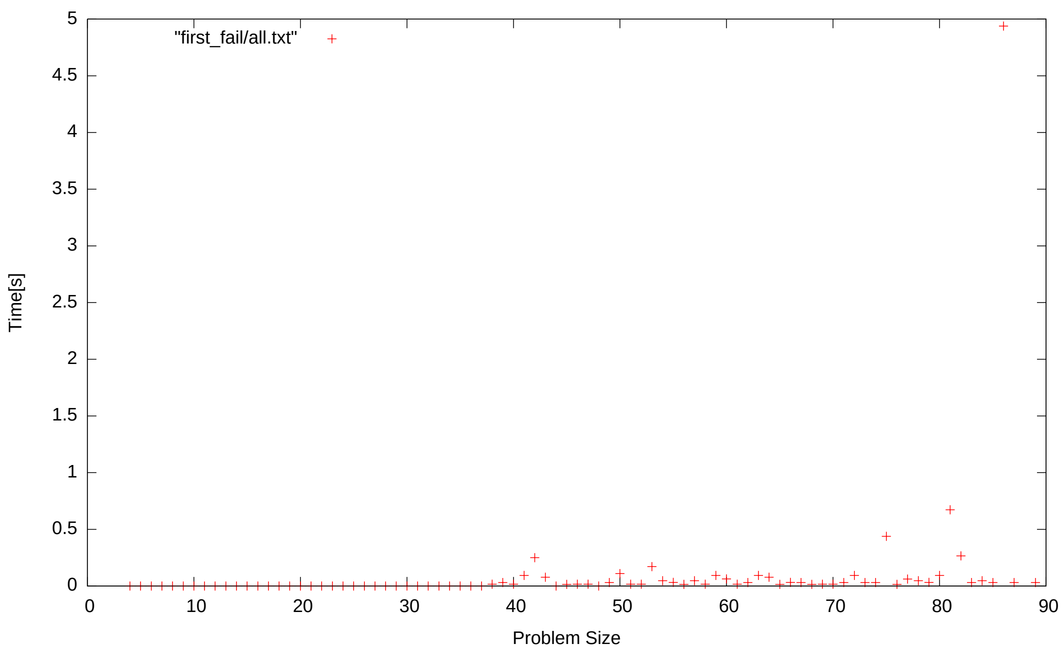
Cork Constraint Computation Centre

Comparing Solutions



Solutions are different!

FirstFail, Problem Sizes 4-100



Observations

- This is much better
- But some sizes are much harder
- Timeout for sizes 88, 91, 93, 97, 98, 99

Can we do better?

- Improved initial ordering
  - Queens on edges of board are easier to assign
  - Do hard assignment first, keep simple choices for later
  - Begin assignment in middle of board
- Matching value choice
  - Values in the middle of board have higher impact
  - Assign these early at top of search tree
  - Use `indomain_middle` for this

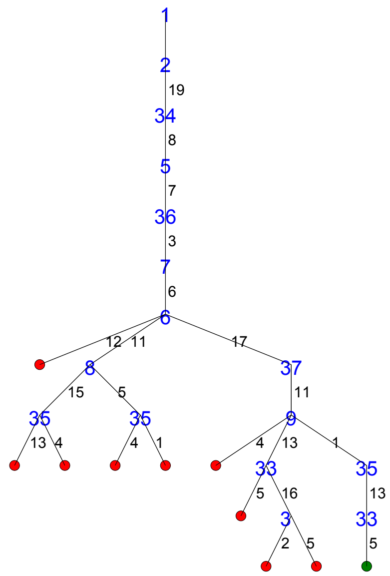
Modified Program

```
:-module(nqueen).
:-export(top/0).
:-lib(ic).
top:-
    nqueen(16,L),writeln(L).

nqueen(N,L):-
    length(L,N),
    L :: 1..N,
    alldifferent(L),
    noattack(L),
    reorder(L,R),
  search(R,0,first_fail,indomain_middle,complete,[]).
```


Reordering Variable List

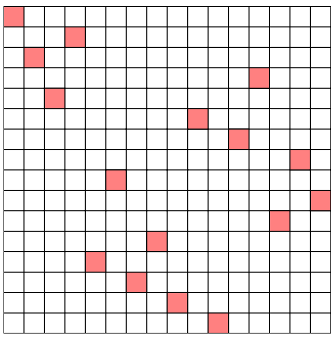
```
reorder(L,L1):-
    halve(L,L,[],Front,Tail),
    combine(Front,Tail,L1).

halve([],Tail,Front,Front,Tail).
halve([_],Tail,Front,Front,Tail).
halve([_,_|R],[F|T],Front,Fend,Tail):-
    halve(R,T,[F|Front],Fend,Tail).

combine(C,[],C):-!.
combine([],C,C).
combine([A|A1],[B|B1],[B,A|C1]):-
    combine(A1,B1,C1).
```

Start from Middle (Size 16)

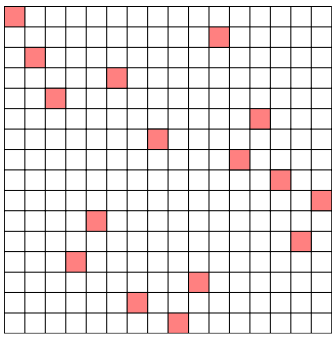


Comparing Solutions

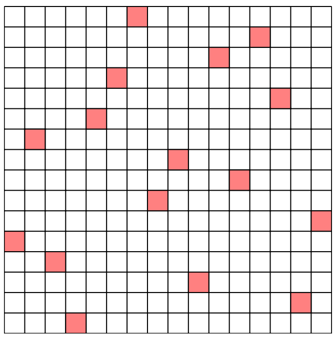
Naive



First Fail

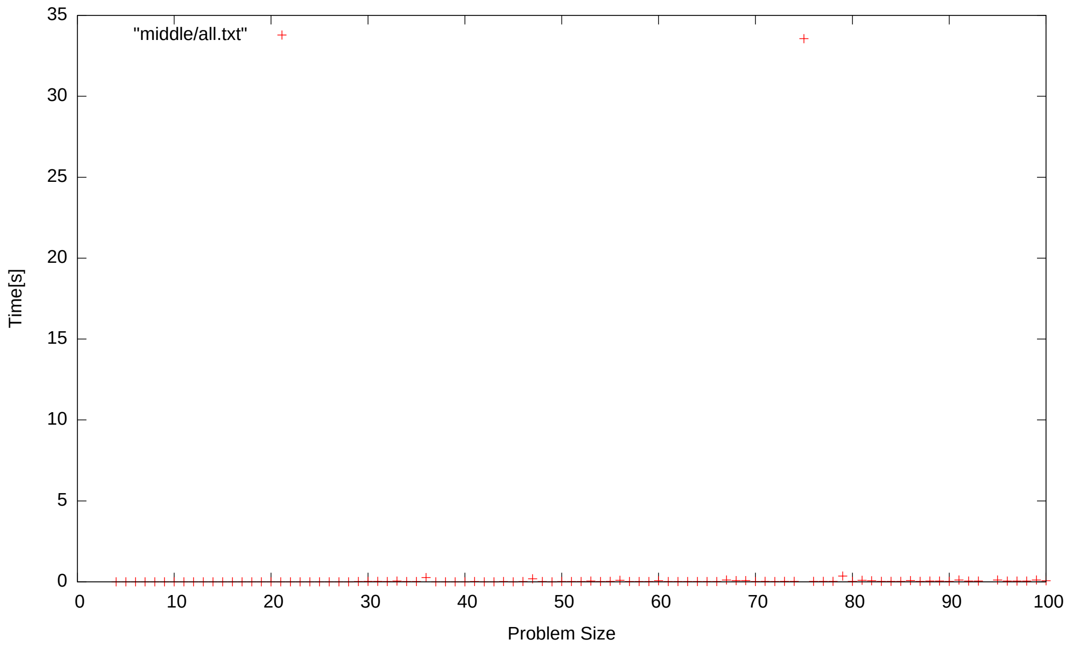


Middle



Again, solutions are different!

Middle, Problem Sizes 4-100



Cork Constraint Computation Centre

Observations

- Not always better than first fail
- For size 16, trees are similar size
- Timeout only for size 94
- But still, one strategy does not work for all problem sizes
- There are ways to resolve this!

Cork Constraint Computation Centre

Approach 1: Heuristic Portfolios

- Try multiple strategies for the same problem
- With multi-core CPUs, run them in parallel
- Only one needs to be successful for each problem

Approach 2: Restart with Randomization

- Only spend limited number of backtracks for a search attempt
- When this limit is exceeded, restart at beginning
- Requires randomization to explore new search branch
- Randomize variable choice by random tie break
- Randomize value choice by shuffling values
- Needs strategy when to restart

Approach 3: Partial Search

- Abandon depth-first, chronological backtracking
- Don't get locked into a failed sub-tree
- A wrong decision at a level is not detected, and we have to explore the complete subtree below to undo that wrong choice
- Explore more of the search tree
- Spend time in promising parts of tree

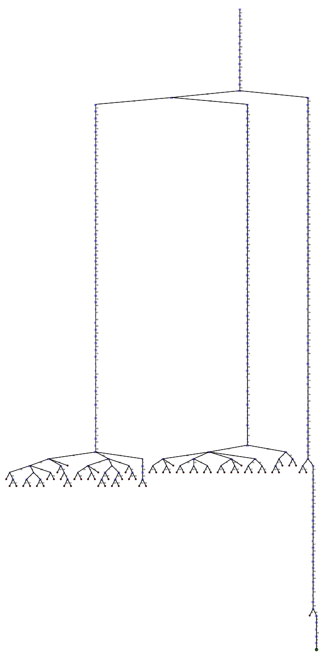
Example: Credit Search

- Explore top of tree completely, based on credit
- Start with fixed amount of credit
- Each node consumes one credit unit
- Split remaining credit amongst children
- When credit runs out, start bounded backtrack search
- Each branch can use only K backtracks
- If this limit is exceeded, jump to unexplored top of tree

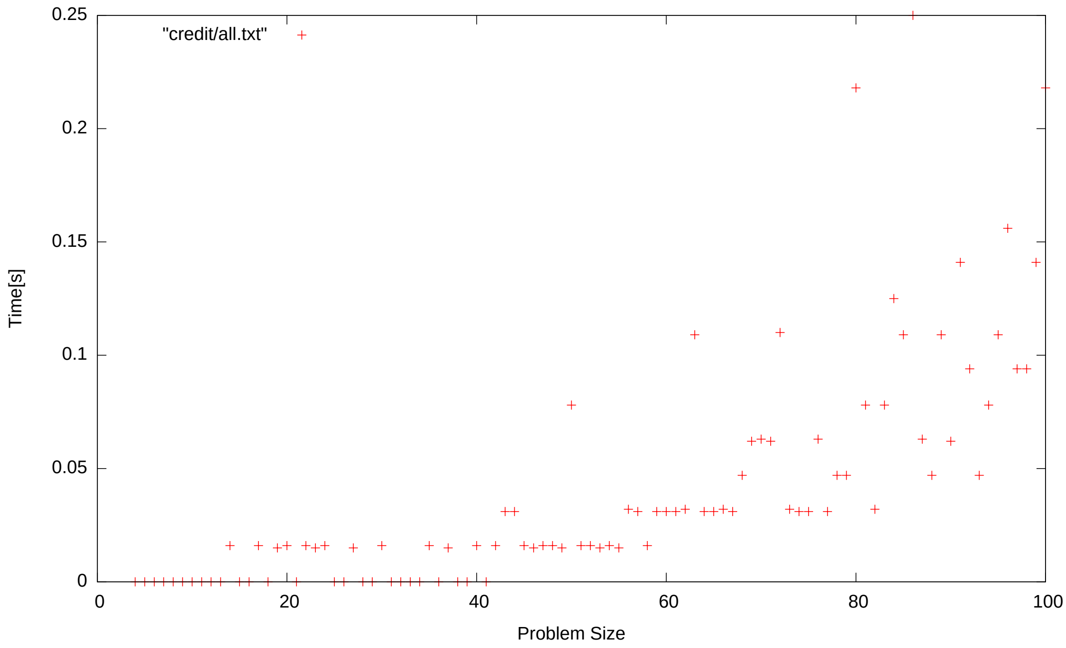
Credit based search

```
:-module(nqueen).
:-export(top/0).
:-lib(ic).
top:-
    nqueen(8,L),writeln(L).

nqueen(N,L):-
    length(L,N),
    L :: 1..N,
    alldifferent(L),
    noattack(L),
    reorder(L,R),
    search(R,0,first_fail,indomain_middle, credit(N,5), [])
```

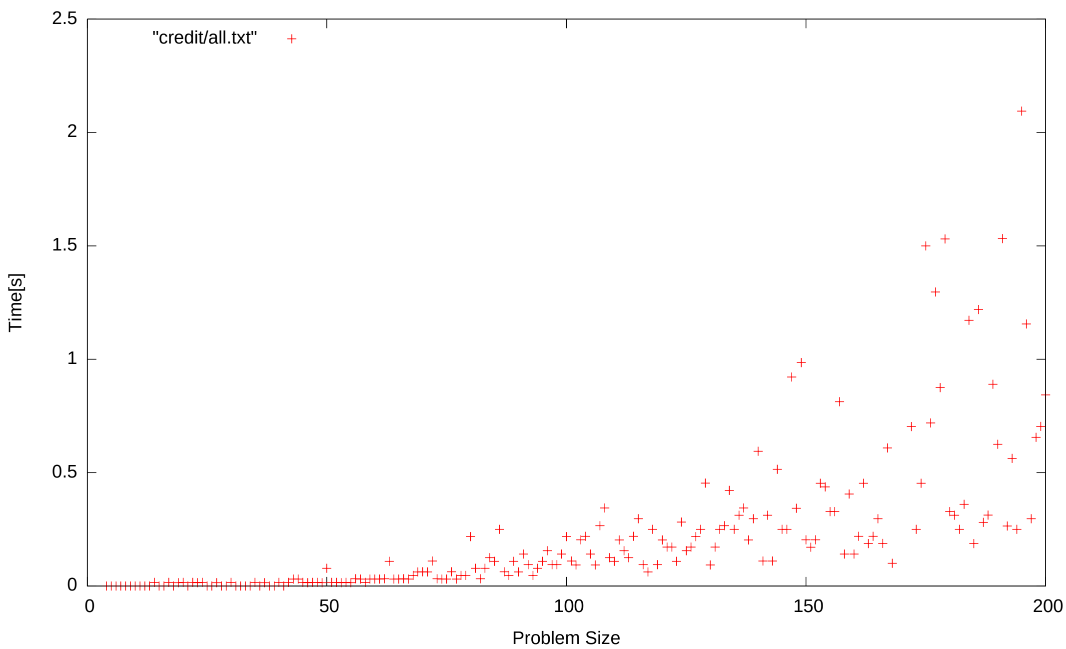

Credit, Search Tree Problem Size 94



Credit, Problem Sizes 4-100



Credit, Problem Sizes 4-200



Conclusions

- Choice of search can have huge impact on performance
- Dynamic variable selection can lead to large reduction of search space
- `search` builtin provides useful abstraction of search functionality
- Depth-first chronologicial backtracking not always best choice

Outlook

- Finite domain with good search reasonable for board sizes up to 1000
- Limitation is memory, not execution time
- Memory requirement quadratic as domain changes must be trailed
- Better results possible for repair based methods
- N-Queens not a hard problem, so general conclusions hard to draw

Exercises

1. Write a program for the 0/1 model of the puzzle as described above. Explain the problem with introducing a dynamic variable ordering for this model.
2. It is possible to express the problem with only three `alldifferent` constraints. Can you describe this model?
3. What is the impact of using a more powerful consistency method for the `alldifferent` constraint in our model? How do the search trees differ to our solution? Does it pay off in execution time?
4. Describe precisely what the `reorder` predicate does. You may find it helpful to run the program with instantiated lists of varying length.
5. The credit search takes two parameters, the total amount of credit and the extra number of backtracks allowed after the credit runs out. How does the program behave if you change these parameters? Can you explain this behaviour?

Cork Constraint Computation Centre

Chapter 7: Optimization (Routing and Wavelength Assignment)

Helmut Simonis

Cork Constraint Computation Centre
Computer Science Department
University College Cork
Ireland

ECLiPSe ELearning Overview

Licence

Outline

What We Want to Introduce

- Optimization
- Graph algorithm library
- Problem decomposition
- Routing and Wavelength Assignment in Optical Networks

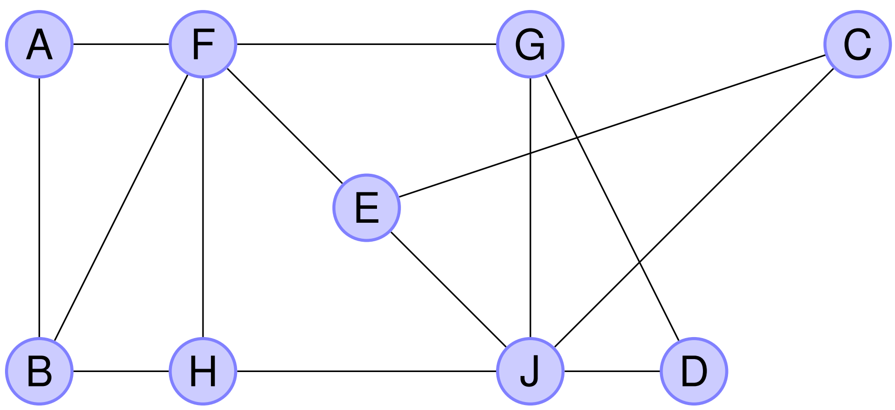
Problem Definition

Routing and Wavelength Assignment

In an optical network, traffic demands between nodes are assigned to a route through the network and a specific wavelength. The route (called *lightpath*) must be a simple path from source to destination. Demands which are routed over the same link must be allocated to different wavelengths, but wavelengths may be reused for demands which do not meet. The objective is to find a combined routing and wavelength assignment which minimizes the number of wavelengths used for a given set of demands.

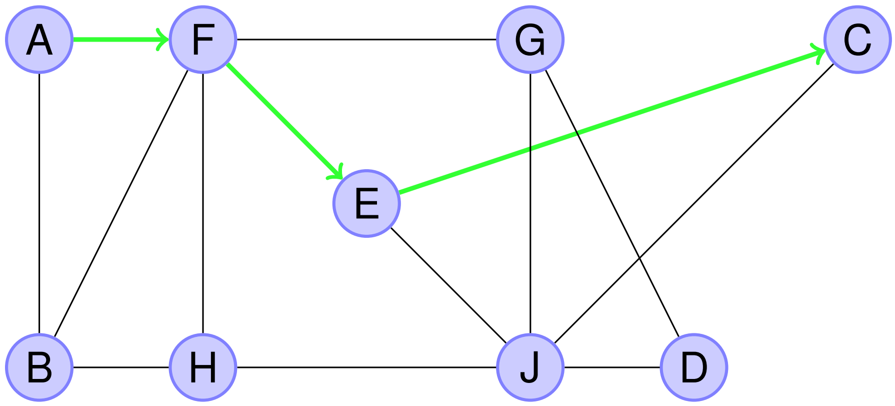
Cork
Constraint
Computation
Centre

Example Network

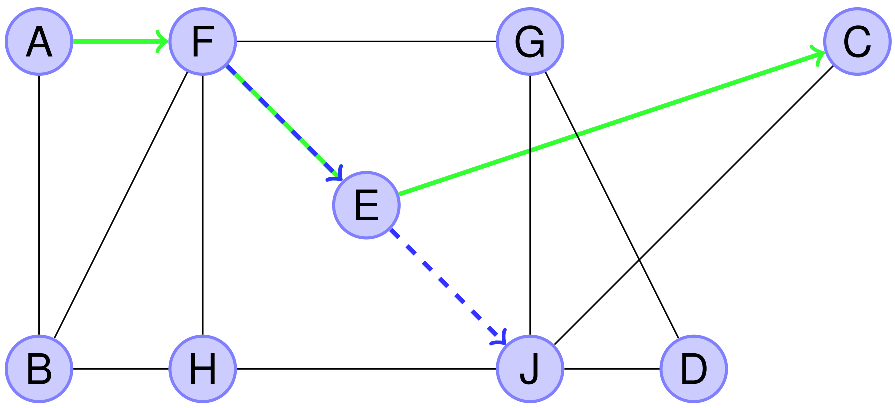


Cork
Constraint
Computation
Centre

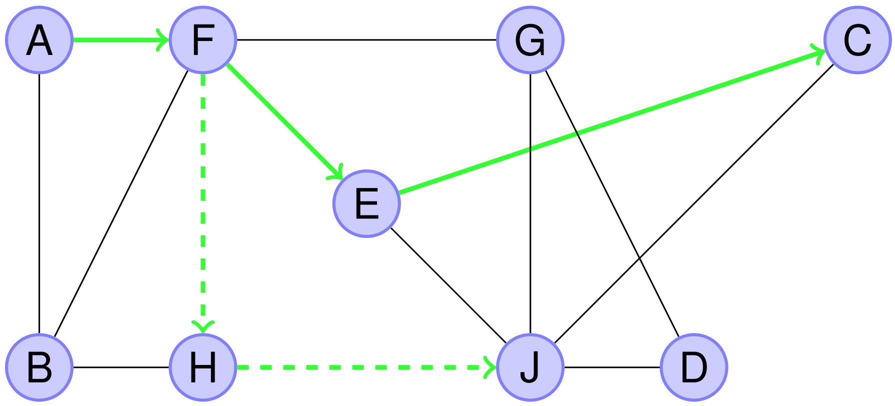
Lightpath from A to C



Conflict between demands A to C and F to J: Use different frequencies



Conflict between demands A to C and F to J: Use different paths



Solution Approaches

- Greedy heuristic
- Optimization algorithm for complete problem
- Decomposition into two problems
  - Find routing
  - Assign wavelengths

Finding Routing

- Find routing which does not assign too many demands on the same link
- Lower bound for overall problem
- Do not use arbitrarily complex paths
- Start with shortest paths

Proposed Solution

- For each demand, use a shortest path between source and destination
- Shortest path = smallest number of links used
- Good for overall network utilisation
- May create bottlenecks on some links

How to Find Shortest Paths

- Well studied, well understood problem
- Many different algorithms for particular cases
  - Positive/negative weight
  - Path between pair of nodes/between node and all other nodes/between all nodes
  - One/all shortest paths or paths which are nearly shortest paths
- Don't program this yourself!
- Library in ECLiPSe: `lib(graph_algorithms)`

Library `graph_algorithms`

- Provides different algorithms about graphs
- Based on opaque `Graph` structure created from nodes and edges
- `make_graph(NrNodes,Edges,Graph)`
- Edges are terms `e(FromNode,ToNode,Weight)`
- Directed graphs as default, undirected graphs represented by edges in both directions

Basic Shortest Path Method

- `single_pair_shortest_path(Network,-1,From,To,Result)`
- Find path from node `From` to node `To` in graph `Network`
- Second argument describes weight function
  - -1: use number of hops
- `Result` given length of path and edges as list

Problem 2: Assign Wavelength

- Demands are routed on shortest paths
- Demands routed over the same link must have different frequencies
- Minimize maximal number of frequencies used

Model

- Domain variable for every demand
- Initial domain large, e.g. number of demands
- Disequality constraint between demands routed over same link
- Alternative: `alldifferent` constraints for all demands over each link
- Feasible solution: find assignment for variables

Optimization

- We are not looking for only a feasible solution
- We want to optimize objective
- Minimize largest value used

Library `branch_and_bound`

- `bb_min(Goal,Cost,bb_options{})`
- `Goal` search goal
  - Like `search/6` or `labeling/1` call
- `Cost` objective (domain variable)
- `bb_options` optional parameters
  - `timeout:Time` timeout limit in seconds
  - `from:LowerBound` known lower bound
  - `to:UpperBound` known upper bound

Example

```
...
List :: 1..20,
...
ic:max(List,Max),
bb_min(labeling(List),Max,
       bb_options{timeout:100,from:10}),
...
```

ic Constraint max(List,Var)

- `Var` is the largest value occuring in `List`
- Similar `min(List,Var)`
- Do not confuse with `max` in core language

Main Program

```
:-module(pure).
:-export(top/5).
:-lib(ic).
:-lib(ic_global).
:-lib(graph_algorithms).
:-lib(branch_and_bound).

top(Name,NrDemands,LowerBound,Assignment,Max):-
    problem(Name,NrDemands,Network,Demands),
    route(Network,Demands,Routes),
    wave(NrDemands,Routes,
         LowerBound,Assignment,Max).
```

Routing

```
route(Network,Demands,Routes):-
     (foreach(demand(I,From,To),Demands),
      foreach(route(I,Path),Routes),
      param(Network) do
          single_pair_shortest_path(Network,-1,
                                    From,To,
                                    _-Path)
     ).
```

Cork Constraint Computation Centre

Wavelength Assignment

```
wave(NrDemands,Routes,LowerBound,Var,Max):-
     dim(Var,[NrDemands]),
     Var[1..NrDemands] :: 1..NrDemands,
     ic:max(Var,Max),
     setup_alldifferent(Routes,Var,LowerBound),
     bb_min(assign(Var),Max,
            bb_options{from:LowerBound,
                       timeout:100}).
```

Cork Constraint Computation Centre

Assignment Routine

```
assign(Var):-
    search(Var,0,most_constrained,indomain,
           complete,[]).
```


Variable Selection Method `most_constrained`

- Similar to `first_fail`
- Select vairable with smallest domain first
- For tie break, select variable in largest number of constraints

Creating `alldifferent` Constraints

```
setup_alldifferent(Routes,Var,LowerBound):-
    (foreach(route(I,Path),Routes),
     fromto([],A,A1,Pairs) do
        (foreach(Edge,Path),
         fromto(A,AA,[l(Edge,I)|AA],A1),
         param(I) do
            true
        )
    ),
    group(Pairs,1,Groups),
    ...
```

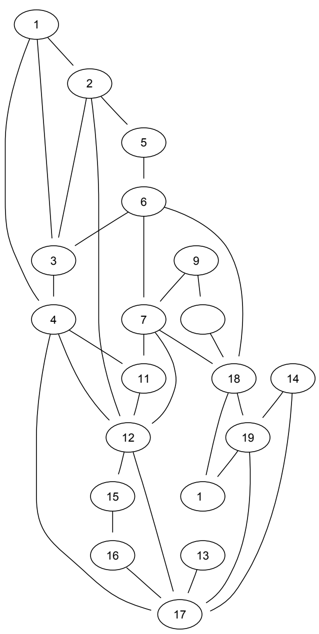

Creating `alldifferent` Constraints (II)

```
    ...
    (foreach(_-Group,Groups),
     fromto(0,A,A1,LowerBound),
     param(Var) do
        length(Group,N),
        A1 is eclipse_language:max(N,A),
        (foreach(l(_,I),Group),
         foreach(X,AlldifferentVars),
         param(Var) do
            subscript(Var,[I],X)
        ),
        ic_global:alldifferent(AlldifferentVars)
    ).
```

Generating Data

```
problem(Name,NrDemands,Network,Demands):-
    network_topology(Name,NrNodes,Edges),
    make_graph(NrNodes,Edges,Directed),
    make_undirected_graph(Directed,Network),
    (for(I,1,NrDemands),
     fromto([],A,[demand(I,From,To)|A],Demands),
     param(NrNodes) do
        repeat,
        From is 1+(random mod NrNodes),
        To is 1+(random mod NrNodes),
        From \= To,
        !
    ).
```

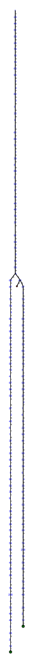

Example Network: MCI



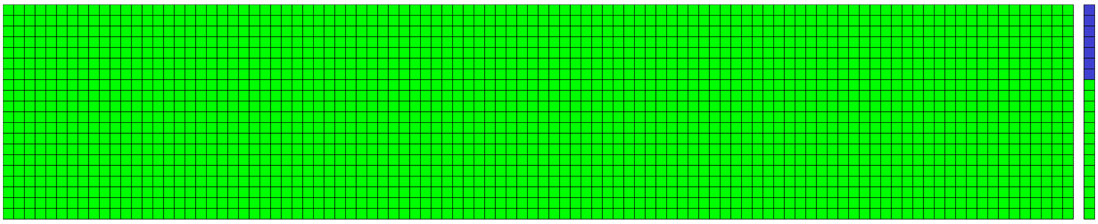
MCI Topology Data

```
network_topology(mci,19,
   [e(1,2,1),e(1,5,1),e(1,6,1),e(2,3,1),
    e(2,5,1),e(2,12,1),e(3,4,1),e(4,5,1),
    e(4,8,1),e(4,10,1),e(5,6,1),e(6,11,1),
    e(6,12,1),e(6,18,1),e(7,8,1),e(7,9,1),
    e(8,10,1),e(8,11,1),e(8,12,1),e(9,10,1),
    e(10,17,1),e(10,19,1),e(11,12,1),e(12,13,1),
    e(12,18,1),e(13,14,1),e(14,18,1),e(15,18,1),
    e(16,17,1),e(16,18,1),e(17,18,1),e(17,19,1)]).
```

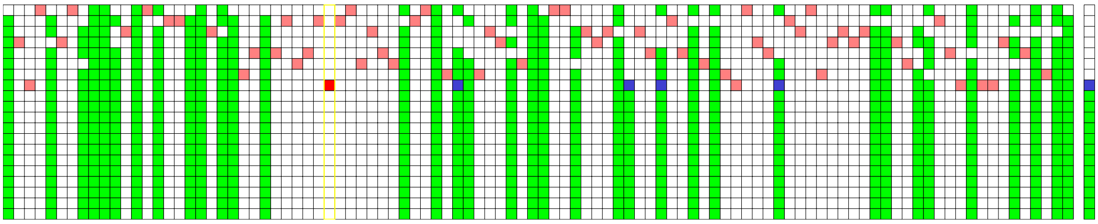

Searchtree



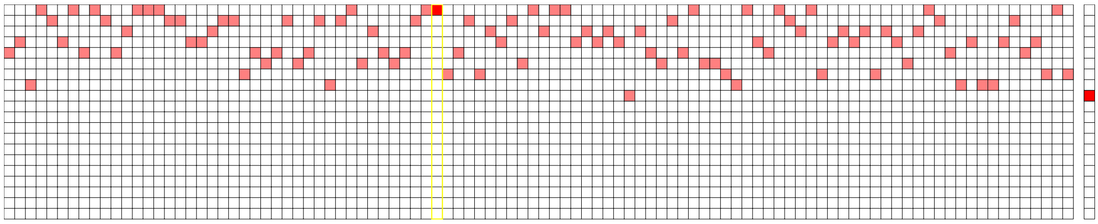
Initial State



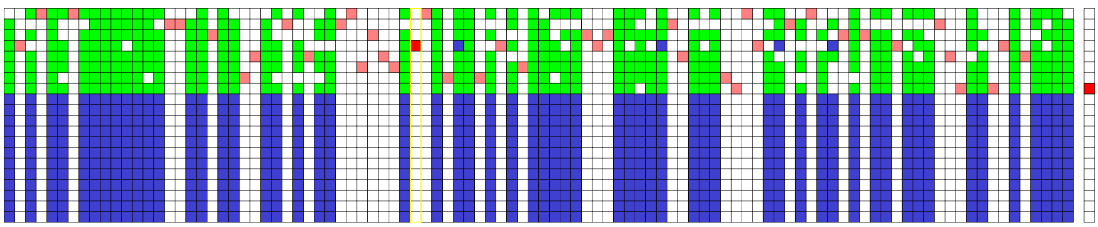
Update Cost



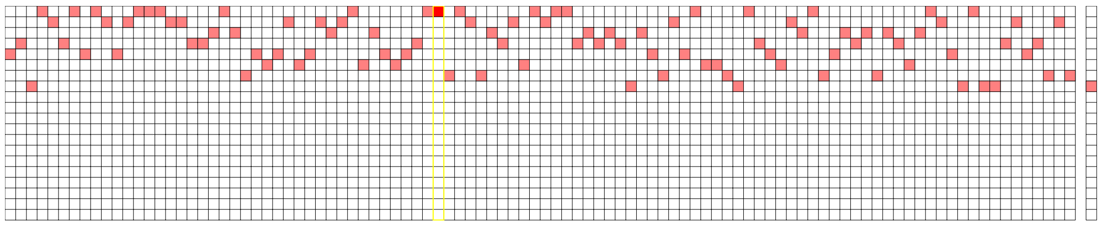
First Solution



Continue Search



Optimal Solution



Observations

- Optimal solution found with minimal backtracking
- Reaching lower bound avoids enumeration proof of optimality
- Not guaranteed to be optimal for original problem
- Given decomposition destroys flexibility in finding solution

Further Experiments

- Vary number of demands to be handled
- Make 100 runs with randomized demands

Multiple Runs (100 experiments)

Network	Nr Demands	Avg LB	Avg Sol	Σ Sol	Avg Gap
mci	20	3.71	3.71	0.711	0.00
mci	40	5.85	5.85	0.931	0.00
mci	60	7.69	7.69	1.324	0.00
mci	80	9.48	9.48	1.353	0.00
mci	100	11.34	11.34	1.687	0.00
mci	120	12.89	12.89	1.928	0.00
mci	140	14.59	14.59	2.298	0.00
mci	160	16.28	16.28	2.421	0.00
mci	180	17.89	17.89	2.656	0.00
mci	200	19.52	19.52	2.456	0.00

Conclusions

- These are not hard problem instances
- In general, graph coloring can be much more difficult
- Fast, simple solution to RWA problem
- Quality gap to be determined

Cork Constraint Computation Centre

Chapter 8: Symmetry Breaking (Balanced Incomplete Block Designs)

Helmut Simonis

Cork Constraint Computation Centre
Computer Science Department
University College Cork
Ireland

ECLiPSe ELearning Overview

Licence

Outline

What we want to introduce

- BIBD - Balanced Incomplete Block Designs
- Using lex constraints to remove symmetries
- Finding all solutions to a problem
- Using timeout to limit search

Problem Definition

BIBD (Balanced Incomplete Block Design)

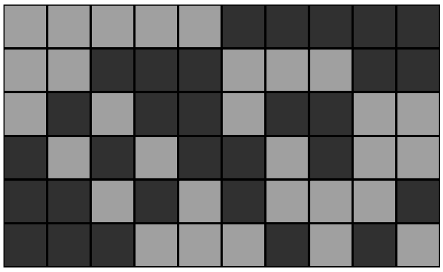
A BIBD is defined as an arrangement of v distinct objects into b blocks such that each block contains exactly k distinct objects, each object occurs in exactly r different blocks, and every two distinct objects occur together in exactly λ blocks. A BIBD is therefore specified by its parameters (v, b, r, k, λ).

Motivation: Test Planning

Consider a new release of some software with v new features. You want to regression test the software against combinations of the new features. Testing each subset of features is too expensive, so you want to run b tests, each using k features. Each feature should be used r times in the tests. Each pair of features should be tested together exactly λ times. How do you arrange the tests?

Model

Another way of defining a BIBD is in terms of its incidence matrix, which is a binary matrix with v rows, b columns, r ones per row, k ones per column, and scalar product λ between any pair of distinct rows.



A (6,10,5,3,2) BIBD

Model

- A binary $v \times b$ matrix. Entry V_{ij} states if item i is in block j.
- Sum constraints over rows, each sum equal r
- Sum constraints over columns, each sum equal k
- Scalar product between any pair of rows, the product value is λ.

Top Level Program

```
:-module(bibd).
:-export(top/0).
:-lib(ic).
:-lib(ic_global).

top:-
    bibd(6,10,5,3,2,Matrix),writeln(Matrix).

bibd(V,B,R,K,L,Matrix):-
    model(V,B,R,K,L,Matrix),➪ Set up model
    extract_array(row,Matrix,List),➪ Get list
    search(L,0,input_order,indomain,
           complete,[]).➪ Search
```


Constraint Model

```
model(V,B,R,K,L,Matrix,Method):-
    dim(Matrix,[V,B]),➪ Define Binary Matrix
    Matrix[1..V,1..B] :: 0..1,
    (for(I,1,V), param(Matrix,B,R) do
        sumlist(Matrix[I,1..B],R)
    ),➪ Row Sum = R
    (for(J,1,B), param(Matrix,V,K) do
        sumlist(Matrix[1..V,J],K)
    ),➪ Column Sum = K
    (for(I,1,V-1), param(Matrix,V,B,L) do
        (for(I1,I+1,V), param(Matrix,I,B,L) do
            scalar_product(Matrix[I,1..B],
                           Matrix[I1,1..B],L)
        )
    ).➪ Scalar product between all rows
```

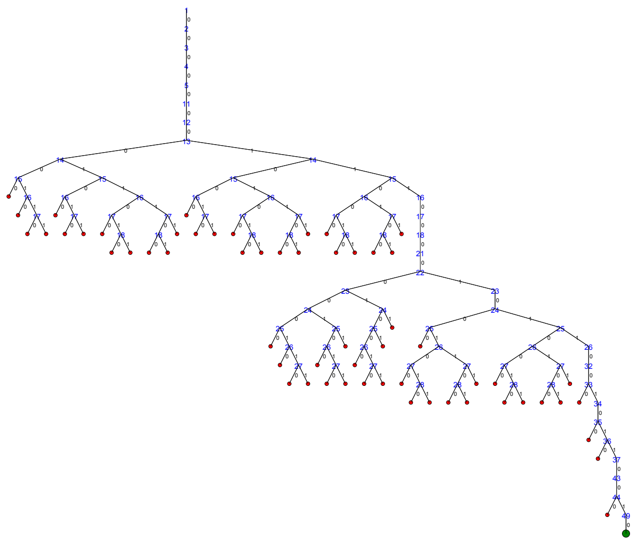
scalar_product

```
scalar_product(XVector,YVector,V):-
    collection_to_list(XVector,XList),
    collection_to_list(YVector,YList),➪ Get lists
    (foreach(X,XList),➪ Iterate over lists
     foreach(Y,YList),➪ ...in parallel
     fromto(0,A,A1,Term) do ➪ Build term
        A1 = A+X*Y➪ Construct term
    ),
    eval(Term) #= V.➪ State Constraint
```


Search Routine

- Static variable order
- First fail does not work for binary variables
- Enumerate variables by row
- Use utility predicate `extract_array/3`
- Assign with `indomain`, try value 0, then value 1
- Use simple `search` call

Basic Model - First Solution



Finding all solutions - Hack!

```
:-module(bibd).
:-export(top/0).
:-lib(ic).
:-lib(ic_global).

top:-
    bibd(6,10,5,3,2,Matrix),writeln(Matrix),
    fail.⇨ Force Backtracking

bibd(V,B,R,K,L,Matrix):-
    model(V,B,R,K,L,Matrix),
    extract_array(row,Matrix,List),
    search(L,0,input_order,indomain,
           complete,[]).
```

Finding all solutions - Proper

```
:-module(bibd).
:-export(top/0).
:-lib(ic).
:-lib(ic_global).

top:-
    findall(Matrix,bibd(6,10,5,3,2,Matrix),Sols),
    writeln(Sols).

bibd(V,B,R,K,L,Matrix):-
    model(V,B,R,K,L,Matrix),
    extract_array(row,Matrix,List),
    search(L,0,input_order,indomain,
           complete,[]).
```


`findall` predicate

- `findall(Template,Goal,Collection)`
- Finds all solutions to `Goal` and collects them into a list `Collection`
- `Template` is used to extract arguments from `Goal` to store as solution
- Backtracks through all choices in `Goal`
- Solutions are returned in order in which they are found

Problem

- Program now only stops when it has found all solutions
- This takes too long!
- How can we limit the amount of time to wait?
- Use of the `timeout` library

Finding all solutions - Proper

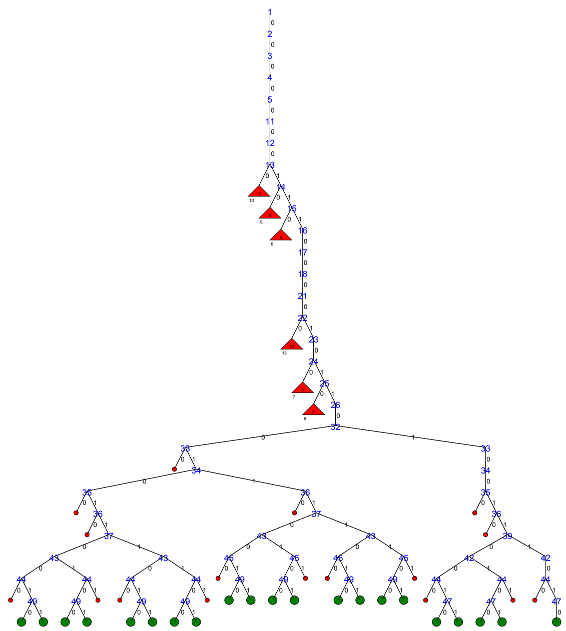
```
:-module(bibd).
:-export(top/0).
:-lib(ic).
:-lib(ic_global).
:-lib(timeout). ⇨ Load library

top:-
    findall(Matrix,timeout(bibd(6,10,5,3,2,Matrix),
                           10, ⇨ seconds
                           fail),Sols),
    writeln(Sols).
```

`timeout` library

- `timeout(Goal,Limit,TimeoutGoal)`
- Runs `Goal` for `Limit` seconds
- If `Limit` is reached, `Goal` is stopped and `TimeoutGoal` is run instead
- If `Limit` is not reached, it has no impact
- Must load `:-lib(timeout).`

Cork Constraint Computation Centre

Search Tree 200 Nodes

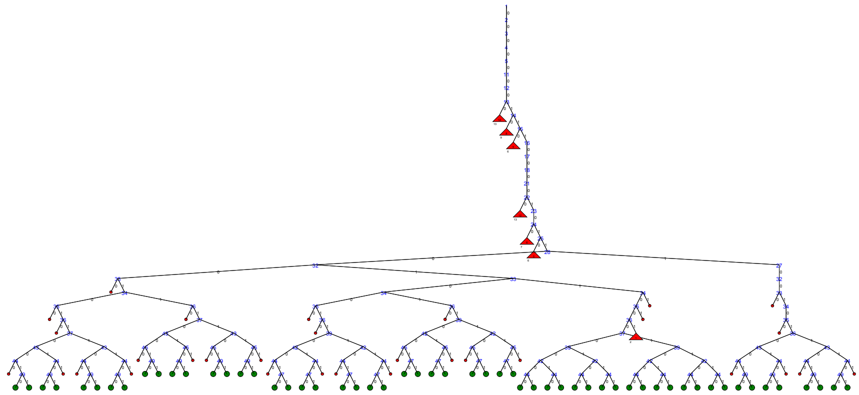


Cork Constraint Computation Centre

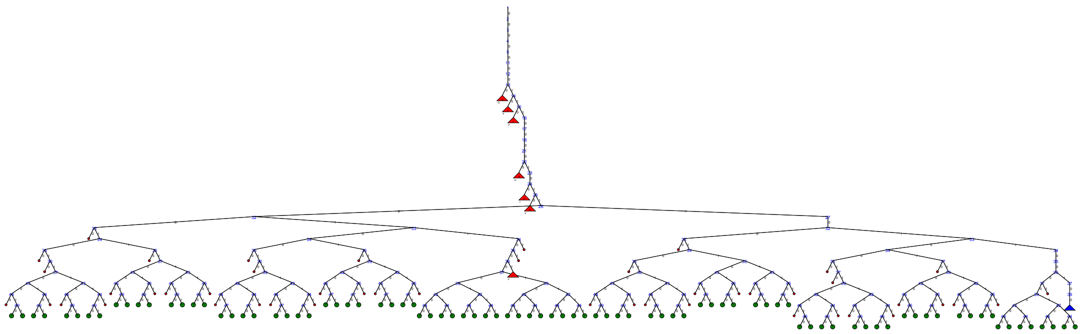
Observation

- Surprise! There are many solutions

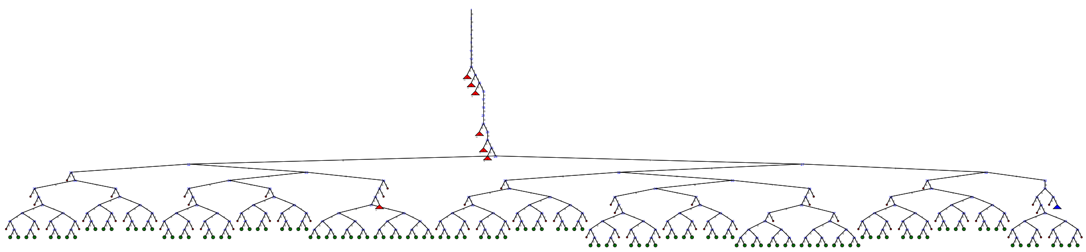
Search Tree 300 Nodes



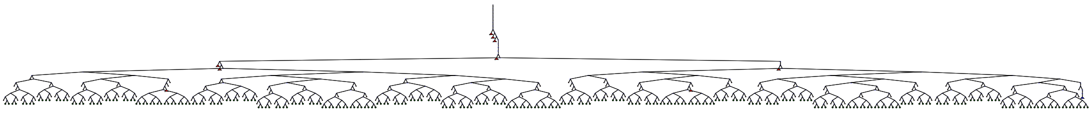
Search Tree 400 Nodes



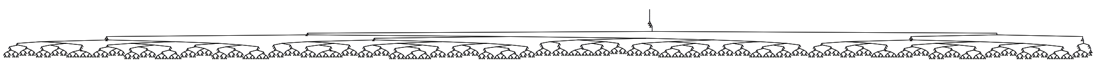
Search Tree 500 Nodes



Search Tree 1000 Nodes



Search Tree 2000 Nodes



Problem

- There are too many solutions to collect in a reasonable time
- Most of these solutions are very similar
- If you take one solution and
  - exchange two rows
  - and/or exchange two columns
- ... you have another solution
- Can we avoid exploring them all?

Symmetry Breaking Techniques

- Remove all symmetries
  - Reduce the search tree as much as possible
  - May be hard to describe all symmetries
  - May be expensive to remove symmetric parts of tree
- Remove some symmetries
  - Search is not reduced as much
  - May be easier to find some symmetries to remove
  - Cost can be low

Symmetry Breaking Techniques

- Symmetry removal by forcing partial, initial assignment
  - Easy to understand
  - Rather weak, does not affect search
- Symmetry removal by stating constraints
  - Removing all symmetries may require exponential number of constraints
  - Can conflict with search strategies
- Symmetry removal by controling search
  - At each node, decide if it needs to be explored
  - Can be expensive to check

Solution used here: Double Lex

- Partial symmetry removal by adding lexicographical ordering constraints
- Our problem has full row and column symmetries
- Any permutation of rows adn/or columns leads to another solution
- Idea: Order rows lexicographically
- Rows must be different from each other, strict order on rows
- Columns might be identical, non strict order on columns
  - This can be improved in some cases
- Constraints only between adjacent rows(columns)

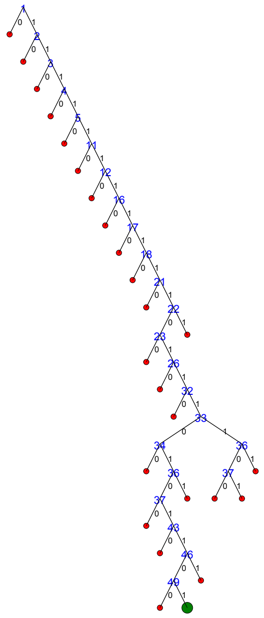
Added Constraints

```
dim(Matrix,[V,B]),
(for(I,1,V-1),
 param(Matrix,B) do
    I1 is I+1,
    lex_less(Matrix[I1,1..B],Matrix[I,1..B])
),⇨ Row lex constraints
 (for(J,1,B-1),
  param(Matrix,V) do
     J1 is J+1,
     lex_leq(Matrix[1..V,J1],Matrix[1..V,J])
 ),⇨ Column lex constraints
```


Two new global constraints

- `lex_leq(List1,List2)`
  - `List1` is lexicographical smaller than or equal to `List2`
  - Achieves domain consistency
- `lex_less(List1,List2)`
  - `List1` is lexicographical smaller than `List2`
  - Achieves domain consistency

Complete Search Tree with Double Lex

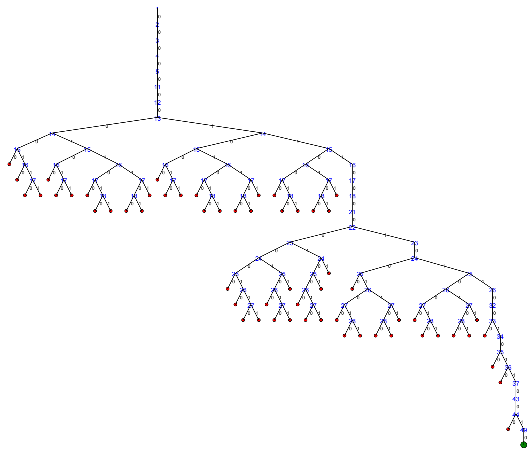


Observation

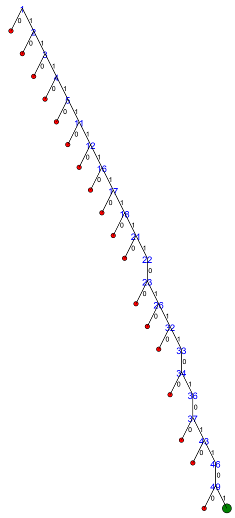
- Enormous reduction in search space
- We are solving a different problem!
- Not just good for finding all solutions, also for first solution!
- Value choice not optimal for finding first solution
- There is a lot of very shallow backtracking, can we avoid that?

Effort for First Solution

Basic Model



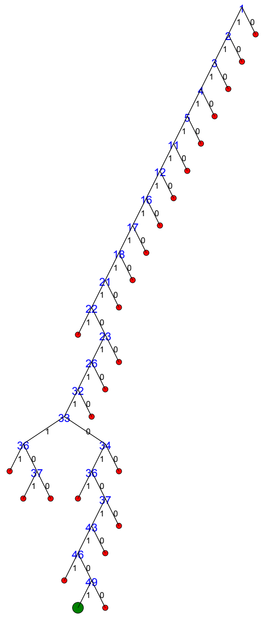
With double Lex



Alternative Value Order

```
:-module(bibd).
:-export(top/0).
:-lib(ic).
:-lib(ic_global).

top:-
    bibd(6,10,5,3,2,Matrix),writeln(Matrix).

bibd(V,B,R,K,L,Matrix):-
    model(V,B,R,K,L,Matrix),
    extract_array(row,Matrix,List),
    search(L,0,input_order,
           indomain_max,⇨ Start with 1
           complete,[]).
```

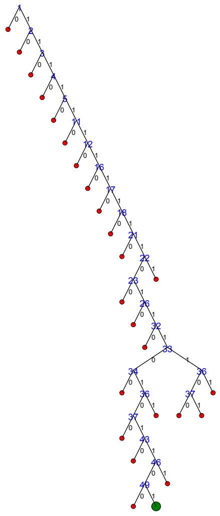
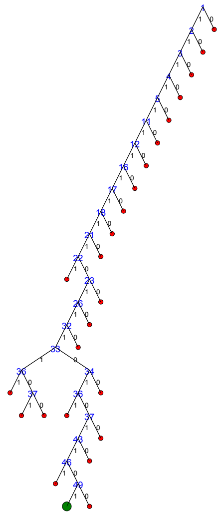
Assigning Value 1 First



Observation

- First solution is found more quickly
- Size of tree for all solutions unchanged
- Value order does not really affect search space when exploring all choices!

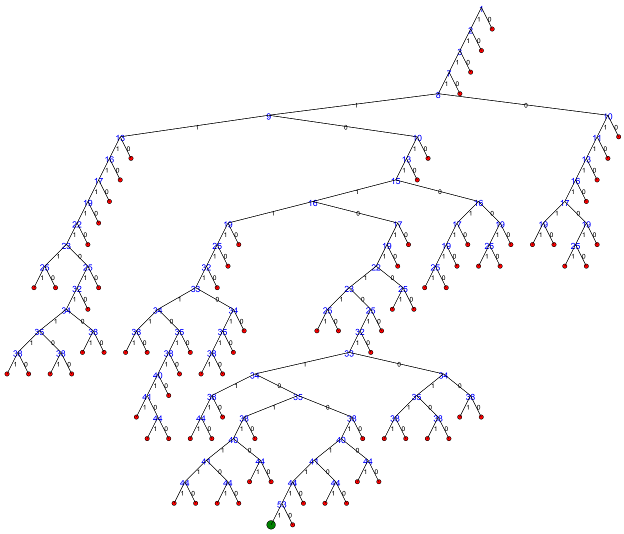
Effort for All Solutions

Assign 0, then 1	Assign 1, then 0
	

Conclusions

- Symmetry breaking can have huge impact on model
- Mainly works for pure problems
- Partial symmetry breaking with additional constraints
- Double lex for row/column symmetries
- Only one variant of many symmetry breaking techniques

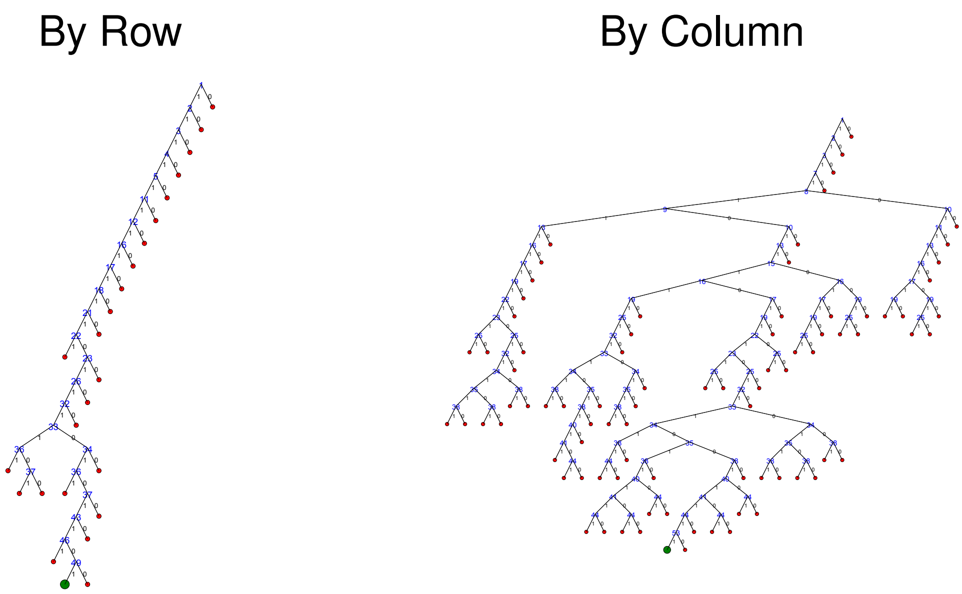
Variable Selection by Column



Observation

- Good, but not as good as row order
- Value choice unimportant even for first solution
- Changing the variable selection does affect size of search space, even for all solutions

Effort for All Solutions



Possible Explanations

- There are fewer rows than columns
- Strict lex constraints on rows, but not on columns
  - More impact of first row
- Needs more testing

Do we need binary variables?

- Consider a model with finite domain variables
- Each of *b* blocks consists of *k* variables ranging over *v* values
- The values in a block must be alldifferent (ordered)
- Each value can occur *r* times
- Scalar product more difficult
- Even better expressed with finite set variables

Exercises

1.

Chapter 9: Choosing the Model (Sports Scheduling)

Helmut Simonis

Cork Constraint Computation Centre
Computer Science Department
University College Cork
Ireland

ECLiPSe ELearning Overview

Licence

Outline

What we want to introduce

- How to come up with a model for a problem
- Why choosing a good model is an art
- Channeling
- Projection
- Redundant Constraints

Sports Scheduling

Tournament Planning

We plan a tournament with 8 teams, where every team plays every other team exactly once. The tournament is played on 7 days, each team playing on each day. The games are scheduled in 7 venues, and each team should play in each venue exactly once.
As part of the TV arrangements, some preassignments are done: We may either fix the game between two particular teams to a fixed day and venue, or only state that some team must play on a particular day at a given venue. The objective is to complete the schedule, so that all constraints are satisfied.

Example

	City 1	City 2	City 3	City 4	City 5	City 6	City 7
Day 1		8			7, 5		
Day 2	2	1, 5					
Day 3	7		8				
Day 4					2	5	1
Day 5	8					1	
Day 6				5, 4			
Day 7	4				1, 3		

Solution

	City 1	City 2	City 3	City 4	City 5	City 6	City 7
Day 1		6, 8		1, 2	5, 7		3, 4
Day 2	2, 3	1, 5			4, 8	6, 7	
Day 3	1, 7	2, 4	3, 8				5, 6
Day 4			4, 7		2, 6	3, 5	1, 8
Day 5	5, 8			3, 6		1, 4	2, 7
Day 6		3, 7	1, 6	4, 5		2, 8	
Day 7	4, 6		2, 5	7, 8	1, 3		

A More Abstract Formulation

Rooms Puzzle, (Thomas G. Room, 1955)

Place numbers 1 to 8 in cells so that each row and each column has each number exactly once, each cell contains either no numbers or two numbers (which must be different from each other), and each combination of two different numbers appears in exactly one cell.

Puzzle presented by R. Finkel

How to come up with a model

- What are the variables/what are their values?
- How can we express the constraints?
- Do we have these constraints in our system?
- Does this do good propagation?
- Backtrack to earlier step as required

Requirements

1. There are 8 teams, seven days and seven locations
2. Each team plays each other team exactly once
3. Each team plays 7 games (redundant)
4. Each team plays in each location exactly once
5. Each team plays on each day exactly once
6. A game consists of two (different) teams
7. There are four games on each day (redundant)
8. There are four games at each location (redundant)
9. In any location there is atmost one game at a time

Idea 1

- Matrix *Day* $\times$ *Game* (7×4)
- Each cell contains two variables, denoting teams
- Easy to say that team plays once on each day, `alldifferent`
- Columns don't have significance
- Model does not mention location, how to add this?
- How to express that each team plays each other once?

Idea 2, Change problem structure

- Matrix of *Day* $\times$ *Location* (7×7)
- Each cell contains two variables, each denoting a team
- How do we avoid symmetry inside cell?
- Need special value (0) to denote that there is no game
- In one cell, either both or none of the variables are 0
- Easy to say that each row and column contains each team exactly once
- Except for value 0, can not use `alldifferent`
- Link between two variables in cell to state that game needs two different teams
- How to express that each (ordered) pair occurs exactly once?

Idea 3, Add location variables

- Model as in Idea 1, matrix $Day \times Game$
- Each cell contains two variables for teams and one for location
- Easy to state that games on one day are in different locations
- How to express condition that each team plays in each location once?
- Also, how to express that each team plays each other exactly once?

Idea 4, Use variables for pairs

- Matrix $Day \times Location$
- Each cell contains one variable ranging over (sorted) pairs of teams, and special value 0 (no game)
- Each pair value occurs once, except for 0
  - Special constraint `alldifferent0`
  - Or use `gcc`
- How to state that each team plays once per day?
- How to state that each team plays in each location?

Idea 5: If all else fails, use binary variables

- Binary variable stating that team *i* plays in location *j* at day *k*
- Three dimensional matrix
- Each team plays once on each day
- Each team plays once in each location
- Each game has two (different) teams, needs auxiliary variable
- Each pair of team meets once, needs auxiliary variables

Idea 6: An even bigger binary model

- Use four dimensions
- Team *i* meets team *j* in location *k* on day *l*
- 3136 = 8*8*7*7 variables
- Constraints all linear
- Why use finite domain constraints?

Idea 7: A different mapping

- Each team plays each other exactly once, one variable for each combination (8*7/2=28 variables)
- Decide when and where this game is played, values range over combinations of days and locations (7*7=49 values)
- All variables must be different (no two games at same time and location)
- Each team plays 7 games, by construction
- How to express that each team plays once per day?
- How to express that each team plays in each location once?

Cork
Constraint
Computation
Centre

Expand Idea 7 into Full Model

Cork
Constraint
Computation
Centre

Numbering Values

	City 1	City 2	City 3	City 4	City 5	City 6	City 7
Day 1	1	2	3	4	5	6	7
Day 2	8	9	10	11	12	13	14
Day 3	15	16	17	18	19	20	21
Day 4	22	23	24	25	26	27	28
Day 5	29	30	31	32	33	34	35
Day 6	36	37	38	39	40	41	42
Day 7	43	44	45	46	47	48	49

Cork Constraint Computation Centre

Four games on each day

	City 1	City 2	City 3	City 4	City 5	City 6	City 7
Day 1	1	2	3	4	5	6	7
Day 2	8	9	10	11	12	13	14
Day 3	15	16	17	18	19	20	21
Day 4	22	23	24	25	26	27	28
Day 5	29	30	31	32	33	34	35
Day 6	36	37	38	39	40	41	42
Day 7	43	44	45	46	47	48	49

- Day 1 corresponds to values 1..7
- Four variables can take these values
- Day 2 corresponds to values 8..14, etc
- One constraint per day
- Exactly four of all variables take their value in the set ...
- Seven such constraints

Cork Constraint Computation Centre

Four games at each location

	City 1	City 2	City 3	City 4	City 5	City 6	City 7
Day 1	1	2	3	4	5	6	7
Day 2	8	9	10	11	12	13	14
Day 3	15	16	17	18	19	20	21
Day 4	22	23	24	25	26	27	28
Day 5	29	30	31	32	33	34	35
Day 6	36	37	38	39	40	41	42
Day 7	43	44	45	46	47	48	49

- City 1 corresponds to values
  - 1, 8, 15, 22, 29, 36, 43
- Four variables can take these values
- City 2 corresponds to values
  - 2, 9, 16, 23, 30, 37, 44
- One constraint per location
- Exactly four of all variables take their value in the set ...
- Seven such constraints over 28 variables each

Teams plays once on a day (at a location)

- Select those variables which correspond to Team *i*
- Exactly one of those variables takes its value in the set 1..7
- Same for all other days
- Same for all other teams
- 56 Constraints over 7 variables each
- Similar for teams and locations, another 56 constraints

Are we there yet?

- 28 variables with 49 possible values
- 1 alldifferent
- 7 exactly constraints over all variables (Days)
- 7 exactly constraints over all variables (Locations)
- 56 exactly constraints over 7 variables each (Days)
- 56 exactly constraints over 7 variables each (Locations)
- Forgotten anything?
- Check the requirements

Do we satisfy the requirements?

1. There are 8 teams, seven days and seven locations
2. Each team plays each other team exactly once
3. Each team plays 7 games (redundant)
4. Each team plays in each location exactly once
5. Each team plays on each day exactly once
6. A game consists of two (different) teams
7. There are four games on each day (redundant)
8. There are four games at each location (redundant)
9. In any location there is atmost one game at a time

What about the `exactly` constraint?

- ECLiPSe doesn't provide this constraint
  - Other system might do, could switch system
- Implement it
  - Extend `gcc` to allow multiple values
  - Should be last resort
- Emulate constraint with others

Idea 8: Mapping games to days and locations

- For each game to be played, we have two variables
  - One ranges over the days
  - The other over the locations
- Easy to state that there are four games per day an location
- Easy to state that each team plays once per day and location
- How do we express that no two games are played at the same location and the same time?
  - If we had an `alldifferent` over pairs of variables...
  - Not in ECLiPSe

We have four games on each day

- Each row value is taken four times amongst the variables
- `gcc([gcc(4,4,1),...,gcc(4,4,7)],Rows)`
- Similar for columns:
- `gcc([gcc(4,4,1),...,gcc(4,4,7)],Cols)`

Each team plays once per day

- For the seven variables which describe games of a team
- Each row value is taken exactly once amongst the variables
- Could use `gcc([gcc(1,1,1),...,gcc(1,1,7)],Vars)`
- But `alldifferent(Vars)` is more compact
- Similar for columns

How do the models differ?

Idea	Mapping
1	$D \times G \times \{f, s\} \to T$
2	$D \times L \times \{f, s\} \to T \cup \{0\}$
3	$D \times G \times \{f, s\} \to T$ $D \times G \to L$
4	$D \times L \to T \vartriangle T \cup \{0\}$
5	$T \times D \times L \to \{0, 1\}$
6	$T \times T \times D \times L \to \{0, 1\}$
7	$T \vartriangle T \to D \times L$
8	$T \vartriangle T \to D$ $T \vartriangle T \to L$

D	Days
T	Teams
L	Locations
G	Games

Requirements Capture

Idea	Requirement								
	1	2	3	4	5	6	7	8	9
1	N	?	Y	?	Y	Y	Y	?	?
2	C	?	Y	Y	Y	Y	Y	Y	Y
3	C	?	Y	?	Y	Y	Y	Y	Y
4	C	Y	Y	Y	Y	Y	Y	Y	Y
5	C	NL	L	L	L	NL	L	L	NL
6	C	L	L	L	L	L	L	L	L
7	C	C	C	E	E	C	E	E	A
8	C	C	C	A	A	C	G	G	?

Comments on models

Idea	Main point
1	missing locations, first second symmetry
2	spare value, first second symmetry
3	first second symmetry
4	spare value
5	0/1, non-linear constraints
6	0/1, large matrix
7	needs exactly constraint
8	needs alldifferent on tuples

Channeling

- Instead of expressing all constraints over one set of variables
- Use multiple sets of variables
- Decide which constraint to express over which variables
- Allows more freedom on how to express problem
- Link the different variables with *channeling* constraints

In Our Case

- Combine ideas 7 and 8
- One set of variables ranging over pairs
- Another using two variables per game for day and location
- How to combine variables?
- Minimize loss of information

Projection

	City 1	City 2	City 3	City 4	City 5	City 6	City 7
Day 1	1	2	3	4	5	6	7
Day 2	8	9	10	11	12	13	14
Day 3	15	16	17	18	19	20	21
Day 4	22	23	24	25	26	27	28
Day 5	29	30	31	32	33	34	35
Day 6	36	37	38	39	40	41	42
Day 7	43	44	45	46	47	48	49

- Link pair variables to row and column variables
- Pair variable uses cell numbers 1-49 as values
- Row and column variables indicate on which day (row) and in which location (column) the game is played
- Pair value 23 = row 4, column 2
- `element` constraint to link the variables
- Two projections from $D \times L$ space onto D and L

Mapping cells to rows and columns

	City 1	City 2	City 3	City 4	City 5	City 6	City 7
Day 1	1	2	3	4	5	6	7
Day 2	8	9	10	11	12	13	14
Day 3	15	16	17	18	19	20	21
Day 4	22	23	24	25	26	27	28
Day 5	29	30	31	32	33	34	35
Day 6	36	37	38	39	40	41	42
Day 7	43	44	45	46	47	48	49

```
element(Cell,[1,1,1,1,1,1,1,2,2,2,2,2,2,2,3,3,3,3,3,3,3,
              4,4,4,4,4,4,4,5,5,5,5,5,5,5,6,6,6,6,6,6,6,
              7,7,7,7,7,7,7],Row),
element(Cell,[1,2,3,4,5,6,7,1,2,3,4,5,6,7,1,2,3,4,5,6,7,
              1,2,3,4,5,6,7,1,2,3,4,5,6,7,1,2,3,4,5,6,7,
              1,2,3,4,5,6,7],Col),
```


Mapping cells to rows and columns

	City 1	City 2	City 3	City 4	City 5	City 6	City 7
Day 1	1	2	3	4	5	6	7
Day 2	8	9	10	11	12	13	14
Day 3	15	16	17	18	19	20	21
Day 4	22	23	24	25	26	27	28
Day 5	29	30	31	32	33	34	35
Day 6	36	37	38	39	40	41	42
Day 7	43	44	45	46	47	48	49

```
element(23  ,[1,1,1,1,1,1,1,2,2,2,2,2,2,2,3,3,3,3,3,3,3,
              4,4,4,4,4,4,4,5,5,5,5,5,5,5,6,6,6,6,6,6,6,
              7,7,7,7,7,7,7],4  ),
element(23  ,[1,2,3,4,5,6,7,1,2,3,4,5,6,7,1,2,3,4,5,6,7,
              1,2,3,4,5,6,7,1,2,3,4,5,6,7,1,2,3,4,5,6,7,
              1,2,3,4,5,6,7],2  ),
```

Channeling Constraints

- This is one common type, a *projection*
- Another common type is the *inverse*
  - Link a variable $A \rightarrow B$ to another $B \rightarrow A$
  - Typically used for bijective mappings
  - Built-in `inverse/2`
- Also used: *Boolean* channeling
  - Link variables $A \rightarrow B$ and $A \times B \rightarrow \{0, 1\}$
  - Built-in `bool_channeling/3`

Selected Model

- Two sets of variables (Req 1, 2, 3, 6, by construction)
- Pair variables ($T \triangle T \rightarrow D \times L$)
  - `alldifferent` (Req 9)
- Day and Location variables ($T \triangle T \rightarrow D$), ($T \triangle T \rightarrow L$)
  - `gcc` (Req 4, 5)
  - `alldifferent` (Req 7, 8)
- Channeling Constraints
  - `element` projection from pairs onto rows and columns
- Search only on pair variables

Handling of hints (I)

	City 1	City 2	City 3	City 4	City 5	City 6	City 7
Day 1		8			7, 5		
Day 2	2	1, 5					
Day 3	7		8				
Day 4					2	5	1
Day 5	8					1	
Day 6				5, 4			
Day 7	4				1, 3		

- This value (17) can not be used by pairs not involving team 8
- One of the pairs involving team 8 must use this value (17)

Handling of hints (II)

	City 1	City 2	City 3	City 4	City 5	City 6	City 7
Day 1		8			7, 5		
Day 2	2	1, 5					
Day 3	7		8				
Day 4					2	5	1
Day 5	8					1	
Day 6				5, 4			
Day 7	4				1, 3		

- The pair involving teams 5 and 7 must take value 5, fixes variable

Problem Data

```
hint(1,8,[2-[8],5-[5,7],8-[2],9-[1,5],15-[7],
          17-[8],26-[2],27-[5],28-[1],29-[8],
          34-[1],39-[4,5],43-[4],47-[1,3]]).
```


Main Program

```
top(Problem,L):-
    hint(Problem,N,Hints),
    N1 is N-1,
    N2 is N//2,
    NrVars is N*N1//2,
    SizeDomain is N1*N1,
    length(L,NrVars),
    L :: 1..SizeDomain,
    create_pairs(N,Contains,Names),
    ic_global_gac:alldifferent(L),
    process_hints(L,Contains,Hints),
    ...
```

Main Program (continued)

```
project_row_cols(L,N1,Rows,Cols),
limit(Rows,N2,N1),
limit(Cols,N2,N1),
separate(Contains,Rows,N,SplitRows),
separate(Contains,Cols,N,SplitCols),
(foreach(K,SplitRows) do
    ic_global_gac:alldifferent(K)
),
(foreach(K,SplitCols) do
    ic_global_gac:alldifferent(K)
),
search(L,0,input_order,indomain,
        complete,[]).
```


Create Pairs and Names

```
create_pairs(N,Contains,Names):-
    (for(I,1,N-1),
     fromto(Names,A1,A,[]),
     fromto(Contains,B1,B,[]),
     param(N) do
        (for(J,I+1,N),
         fromto(A1,[Name|AA],AA,A),
         fromto(B1,[I-J|BB],BB,B),
         param(I) do
            concat_string([I,J],Name)
        )
    ).
```

Projecting Rows and Columns

```
project_row_cols(L,N,Rows,Cols):-
    generate_tables(N,RowTable,ColTable),
    (foreach(X,L),
     foreach(R,Rows),
     foreach(C,Cols),
     param(RowTable,ColTable) do
        element(X,RowTable,R),
        element(X,ColTable,C)
    ).
```

Cork Constraint Computation Centre

Generating Projection Tables

```
generate_tables(N,RowTable,ColTable):-
    (for(I,1,N),
     fromto(RowTable,A1,A,[]),
     fromto(ColTable,B1,B,[]),
     param(N) do
        (for(J,1,N),
         fromto(A1,[I|AA],AA,A),
         fromto(B1,[J|BB],BB,B),
         param(I) do
            true
        )
    ).
```

Cork Constraint Computation Centre

Extract row variables

```
separate(Contains,Rows,Values,SplitRows):-
     (for(Value,1,Values),
      foreach(SplitRow,SplitRows),
      param(Contains,Rows) do
          (foreach(A-B,Contains),foreach(V,Rows),
           fromto([],R,R1,SplitRow),param(Value) do
               (memberchk(Value,[A,B]) ->
                    R1 = [V|R]
               ;
                    R1 = R
               )
          )
     ).
```


Set up gcc constraint

```
limit(L,Bound,Values):-
     (for(I,1,Values),
      foreach(gcc(Bound,Bound,I),Pattern),
      param(Bound) do
         true
     ),
     gcc(Pattern,L).
```

Setting up hints

```
process_hints(L,Contains,Hints):-
     (foreach(Pos-Values,Hints),
      param(L,Contains) do
          process_hint(Pos,Values,L,Contains)
     ).

process_hint(Pos,[A,B],L,Contains):- % clause 1
     !,
     match_hint(A-B,Contains,L,X),
     X #= Pos.
```

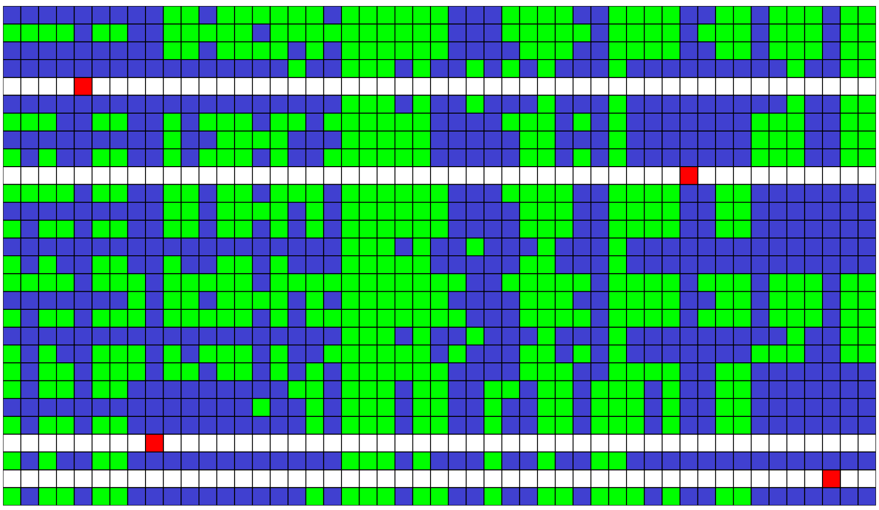

Setting up hints

```
process_hint(Pos,[Value],L,Contains):- % clause 2
     (foreach(X,L),
      foreach(A-B,Contains),
      fromto([],R,R1,Required),
      param(Pos,Value) do
          (not_mentioned(A,B,Value) ->
               X #\= Pos,
               R1 = R
          ;
               R1 = [X|R]
          )
     ),
     occurrences(Pos,Required,1).
```

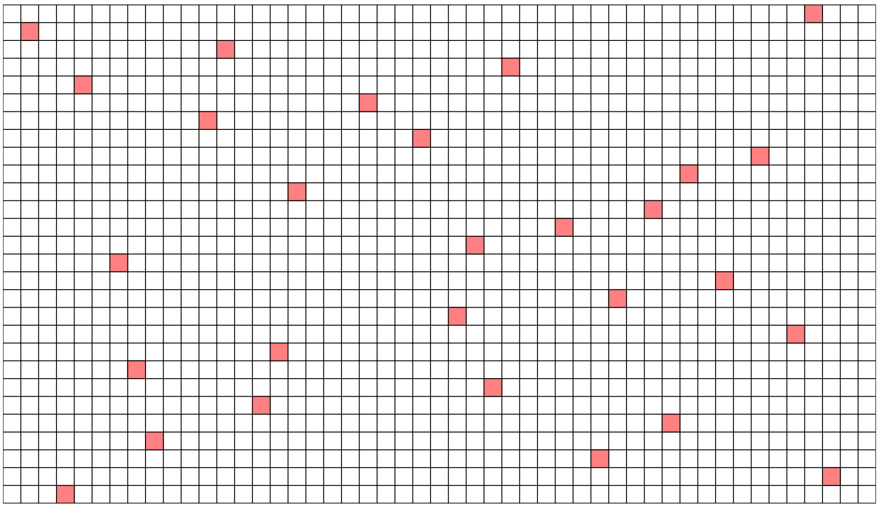
Setting up hints

```
not_mentioned(A,B,V):-
    A \= V,
    B \= V.

match_hint(H,[H|_],[X|_],X):-
    !.
match_hint(H,[_|T],[_|R],X):-
    match_hint(H,T,R,X).
```

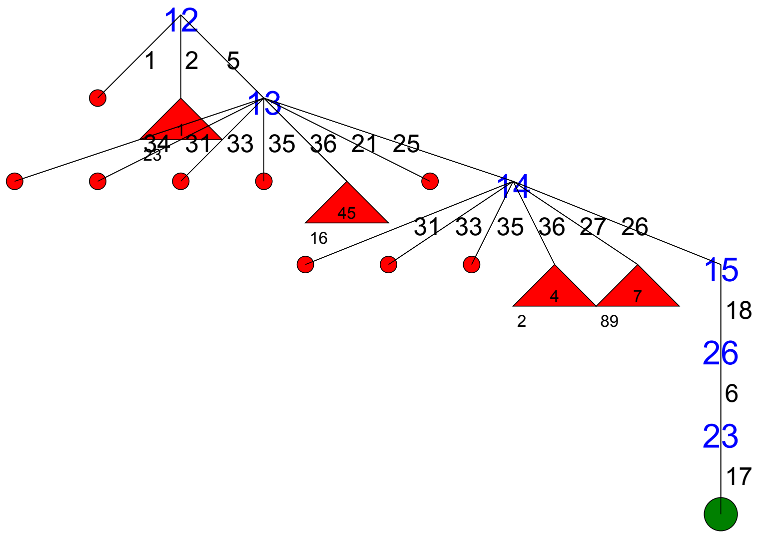

Before Search



Solution



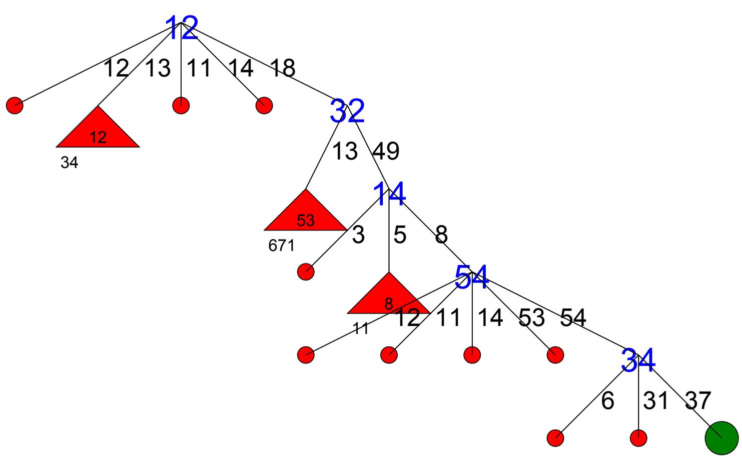
Search Tree with input order



How to improve?

- Try different search strategy
- Use `first_fail` dynamic variable selection

Search Tree with first fail

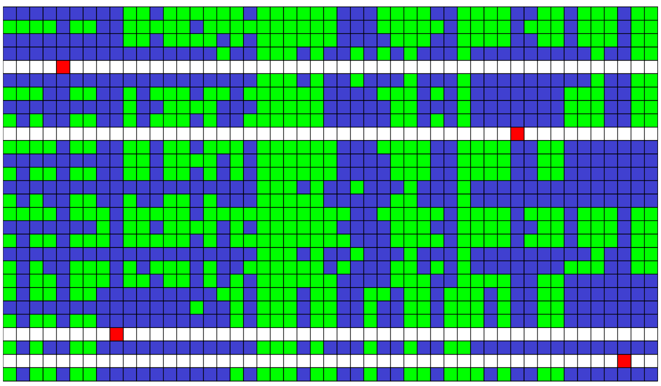


Observation

- It does not work
- Search tree is slightly larger than before!

Missing Propagation

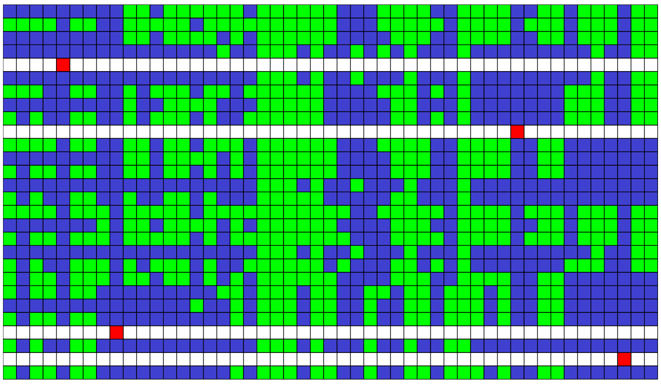
	City 1	City 2	City 3	City 4	City 5	City 6	City 7
Day 1		8			7, 5		
Day 2	2	1, 5					
Day 3	7		8				
Day 4					2	5	1
Day 5	8					1	
Day 6				5, 4			
Day 7	4				1, 3		



	City 1	City 2	City 3	City 4	City 5	City 6	City 7
Day 1	1	2	3	4	5	6	7
Day 2	8	9	10	11	12	13	14
Day 3	15	16	17	18	19	20	21
Day 4	22	23	24	25	26	27	28
Day 5	29	30	31	32	33	34	35
Day 6	36	37	38	39	40	41	42
Day 7	43	44	45	46	47	48	49

Missing Propagation

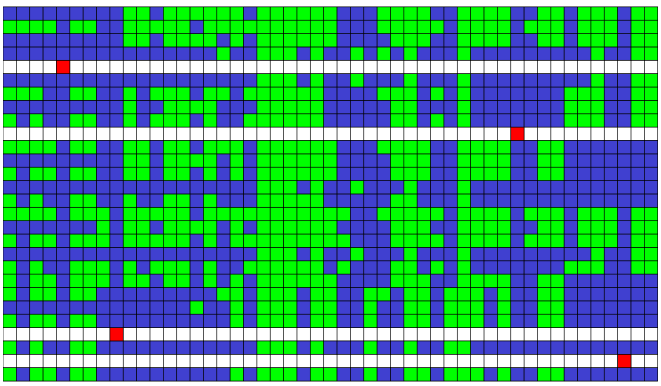
	City 1	City 2	City 3	City 4	City 5	City 6	City 7
Day 1		8			7, 5		
Day 2	2	1, 5					
Day 3	7		8				
Day 4					2	5	1
Day 5	8					1	
Day 6				5, 4			
Day 7	4				1, 3		



	City 1	City 2	City 3	City 4	City 5	City 6	City 7
Day 1	1	2	3	4	5	6	7
Day 2	8	9	10	11	12	13	14
Day 3	15	16	17	18	19	20	21
Day 4	22	23	24	25	26	27	28
Day 5	29	30	31	32	33	34	35
Day 6	36	37	38	39	40	41	42
Day 7	43	44	45	46	47	48	49

Missing Propagation

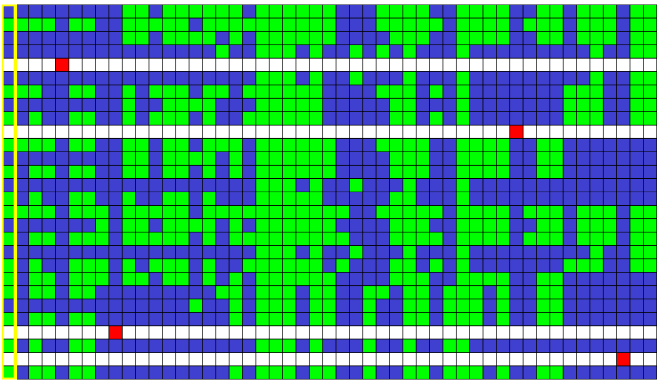
	City 1	City 2	City 3	City 4	City 5	City 6	City 7
Day 1		8			7, 5		
Day 2	2	1, 5					
Day 3	7		8				
Day 4					2	5	1
Day 5	8					1	
Day 6				5, 4			
Day 7	4				1, 3		



	City 1	City 2	City 3	City 4	City 5	City 6	City 7
Day 1	1	2	3	4	5	6	7
Day 2	8	9	10	11	12	13	14
Day 3	15	16	17	18	19	20	21
Day 4	22	23	24	25	26	27	28
Day 5	29	30	31	32	33	34	35
Day 6	36	37	38	39	40	41	42
Day 7	43	44	45	46	47	48	49

Missing Propagation

	City 1	City 2	City 3	City 4	City 5	City 6	City 7
Day 1		8			7, 5		
Day 2	2	1, 5					
Day 3	7		8				
Day 4					2	5	1
Day 5	8					1	
Day 6				5, 4			
Day 7	4				1, 3		

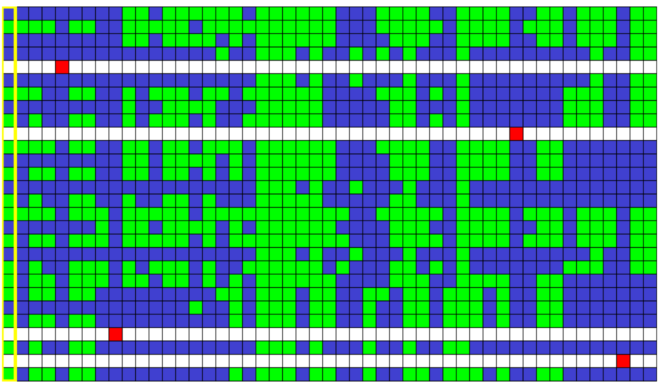


	City 1	City 2	City 3	City 4	City 5	City 6	City 7
Day 1	1	2	3	4	5	6	7
Day 2	8	9	10	11	12	13	14
Day 3	15	16	17	18	19	20	21
Day 4	22	23	24	25	26	27	28
Day 5	29	30	31	32	33	34	35
Day 6	36	37	38	39	40	41	42
Day 7	43	44	45	46	47	48	49

Cork
Constraint
Computation
Centre

Missing Propagation

	City 1	City 2	City 3	City 4	City 5	City 6	City 7
Day 1		8			7, 5		
Day 2	2	1, 5					
Day 3	7		8				
Day 4					2	5	1
Day 5	8					1	
Day 6				5, 4			
Day 7	4				1, 3		

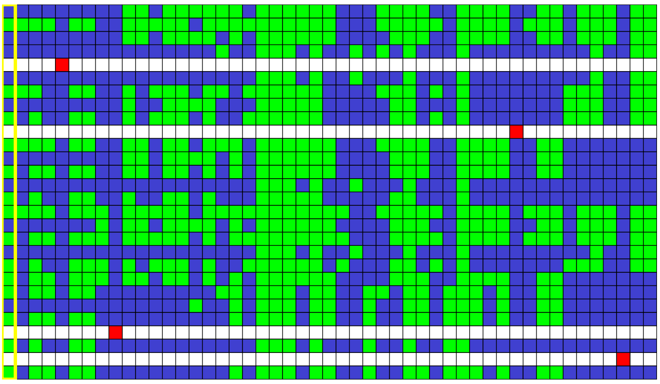


	City 1	City 2	City 3	City 4	City 5	City 6	City 7
Day 1	1	2	3	4	5	6	7
Day 2	8	9	10	11	12	13	14
Day 3	15	16	17	18	19	20	21
Day 4	22	23	24	25	26	27	28
Day 5	29	30	31	32	33	34	35
Day 6	36	37	38	39	40	41	42
Day 7	43	44	45	46	47	48	49

Cork
Constraint
Computation
Centre

Missing Propagation

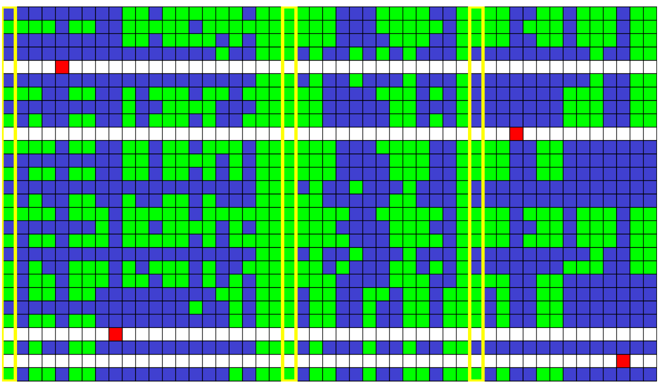
	City 1	City 2	City 3	City 4	City 5	City 6	City 7
Day 1		8			7, 5		
Day 2	2	1, 5					
Day 3	7		8				
Day 4					2	5	1
Day 5	8					1	
Day 6				5, 4			
Day 7	4				1, 3		



	City 1	City 2	City 3	City 4	City 5	City 6	City 7
Day 1	1	2	3	4	5	6	7
Day 2	8	9	10	11	12	13	14
Day 3	15	16	17	18	19	20	21
Day 4	22	23	24	25	26	27	28
Day 5	29	30	31	32	33	34	35
Day 6	36	37	38	39	40	41	42
Day 7	43	44	45	46	47	48	49

Missing Propagation

	City 1	City 2	City 3	City 4	City 5	City 6	City 7
Day 1		8			7, 5		
Day 2	2	1, 5					
Day 3	7		8				
Day 4					2	5	1
Day 5	8					1	
Day 6				5, 4			
Day 7	4				1, 3		



	City 1	City 2	City 3	City 4	City 5	City 6	City 7
Day 1	1	2	3	4	5	6	7
Day 2	8	9	10	11	12	13	14
Day 3	15	16	17	18	19	20	21
Day 4	22	23	24	25	26	27	28
Day 5	29	30	31	32	33	34	35
Day 6	36	37	38	39	40	41	42
Day 7	43	44	45	46	47	48	49

Why is this?

- Constraints involved:
  - `gcc` constraint on row: four variables can use values from this row
  - four `occurrence` constraints for hints: One of the variables must take this value
- No interaction between constraints, only between constraints and variables
- We do not detect that value 1 can not be used
- Eventual solution respects condition, model is correct
- We are concerned about propagation, not correctness

Adding redundant model

- Add constraints which do more propagation, but do not affect solutions
- Lead to smaller search tree, hopefully faster solution
- Introduction requires understanding of (lack of) propagation
- Visualization is key to detect missing propagation

Adding 0/1 model

- *Day* $\times$ *Location* matrix of 0/1 variables
- Indicates if there is a game on this day at this location
- Row/column sums: 4 games in each row/column
- Hint given for cell: Game variable is 1

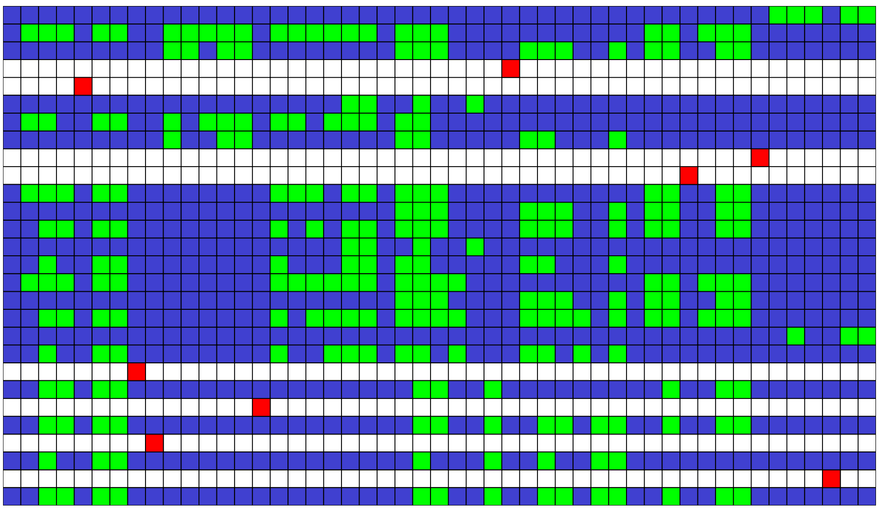
Channeling Constraint

- Link pair variables P_i to 0/1 variables Y_j as *value-index*
- $(\exists i \text{ s.t. } P_i = v) \Leftrightarrow Y_v = 1$
- Propagation:
  - $P_i = v \Rightarrow Y_v = 1$
  - $Y_v = 0 \Rightarrow \forall i: \ P_i \neq v$
  - $(\forall i: \ v \notin d(P_i)) \Rightarrow Y_v = 0$
  - $Y_v = 1 \Rightarrow \text{occurrence}(V, P_1 ... P_n, N), N \geq 1$

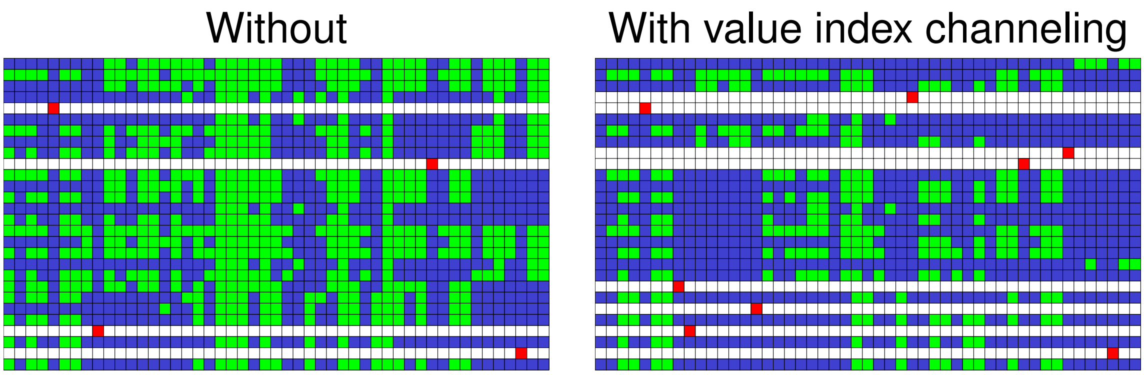
Added Program

```
value_set_channeling(L,Hints):-
    dim(Matrix,[7,7]),
    Matrix[1..7,1..7] :: 0..1,
    flatten_array(Matrix,ValueSet),
    value_set_channel(L,ValueSet,1),
    (for(I,1,7),param(Matrix) do
        sumlist(Matrix[I,1..7],4),
        sumlist(Matrix[1..7,I],4)
    ),
    (foreach(K-_,Hints),param(Matrix) do
        coor(K,I,J),
        subscript(Matrix,[I,J],1)
    ).
```

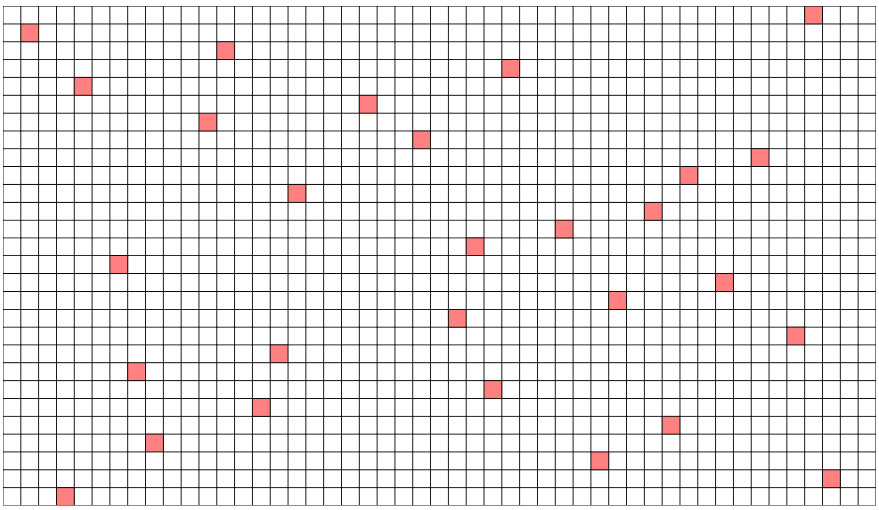

Before Search



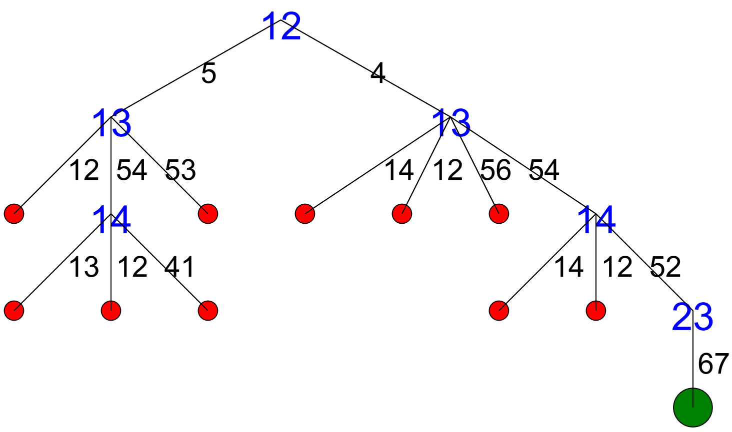
Impact of Redundant Constraints



Solution

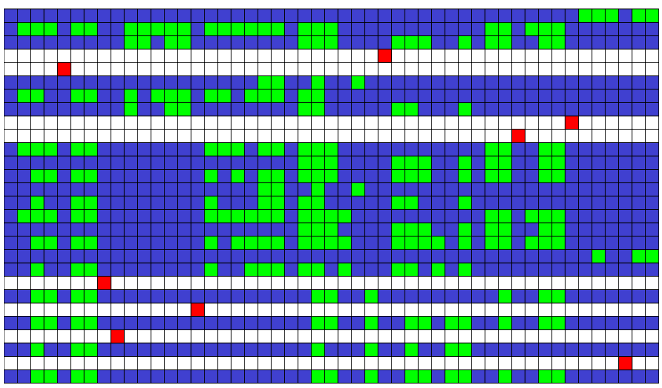


Search Tree



Still Missing Propagation

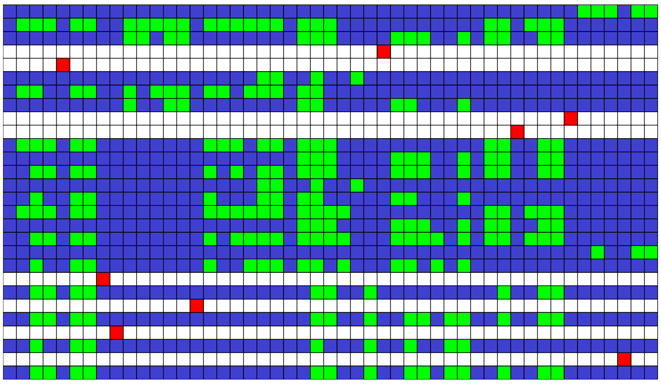
	City 1	City 2	City 3	City 4	City 5	City 6	City 7
Day 1		8			7, 5		
Day 2	2	1, 5					
Day 3	7		8				
Day 4					2	5	1
Day 5	8					1	
Day 6				5, 4			
Day 7	4				1, 3		



	City 1	City 2	City 3	City 4	City 5	City 6	City 7
Day 1	1	2	3	4	5	6	7
Day 2	8	9	10	11	12	13	14
Day 3	15	16	17	18	19	20	21
Day 4	22	23	24	25	26	27	28
Day 5	29	30	31	32	33	34	35
Day 6	36	37	38	39	40	41	42
Day 7	43	44	45	46	47	48	49

Game 12 can not be played on day 1 at locations 5 or 6

Still Missing Propagation

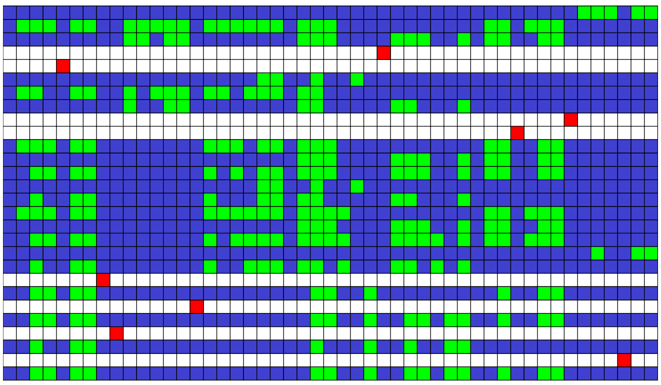


	City 1	City 2	City 3	City 4	City 5	City 6	City 7
Day 1		8			7, 5		
Day 2	2	1, 5					
Day 3	7		8				
Day 4					2	5	1
Day 5	8					1	
Day 6				5, 4			
Day 7	4				1, 3		

	City 1	City 2	City 3	City 4	City 5	City 6	City 7
Day 1	1	2	3	4	5	6	7
Day 2	8	9	10	11	12	13	14
Day 3	15	16	17	18	19	20	21
Day 4	22	23	24	25	26	27	28
Day 5	29	30	31	32	33	34	35
Day 6	36	37	38	39	40	41	42
Day 7	43	44	45	46	47	48	49

Game 12 can not be played on day 1 at locations 5 or 6

Still Missing Propagation

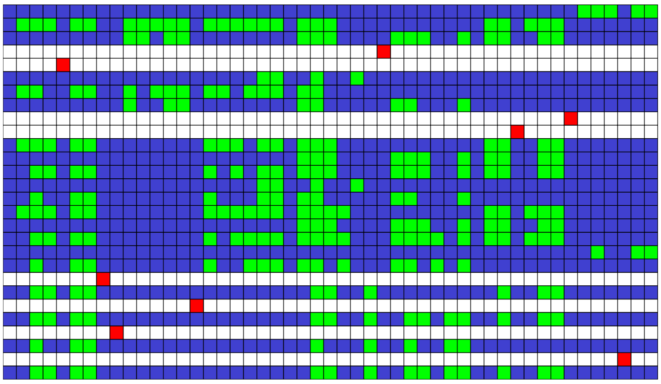


	City 1	City 2	City 3	City 4	City 5	City 6	City 7
Day 1		8			7, 5		
Day 2	2	1, 5					
Day 3	7		8				
Day 4					2	5	1
Day 5	8					1	
Day 6				5, 4			
Day 7	4				1, 3		

	City 1	City 2	City 3	City 4	City 5	City 6	City 7
Day 1	1	2	3	4	5	6	7
Day 2	8	9	10	11	12	13	14
Day 3	15	16	17	18	19	20	21
Day 4	22	23	24	25	26	27	28
Day 5	29	30	31	32	33	34	35
Day 6	36	37	38	39	40	41	42
Day 7	43	44	45	46	47	48	49

Game 12 can not be played on day 1 at locations 5 or 6

Still Missing Propagation

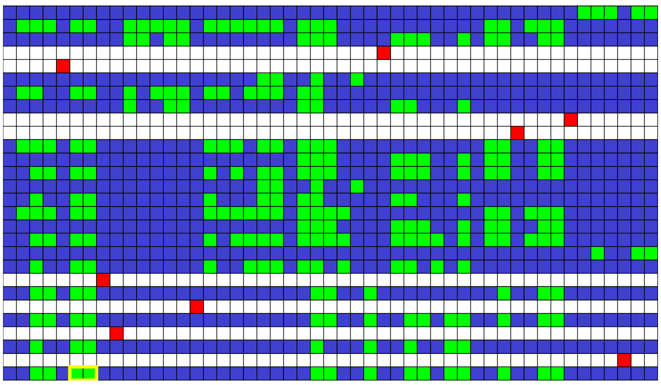


	City 1	City 2	City 3	City 4	City 5	City 6	City 7
Day 1		8			7, 5		
Day 2	2	1, 5					
Day 3	7		8				
Day 4					2	5	1
Day 5	8					1	
Day 6				5, 4			
Day 7	4				1, 3		

	City 1	City 2	City 3	City 4	City 5	City 6	City 7
Day 1	1	2	3	4	5	6	7
Day 2	8	9	10	11	12	13	14
Day 3	15	16	17	18	19	20	21
Day 4	22	23	24	25	26	27	28
Day 5	29	30	31	32	33	34	35
Day 6	36	37	38	39	40	41	42
Day 7	43	44	45	46	47	48	49

Game 12 can not be played on day 1 at locations 5 or 6

Still Missing Propagation

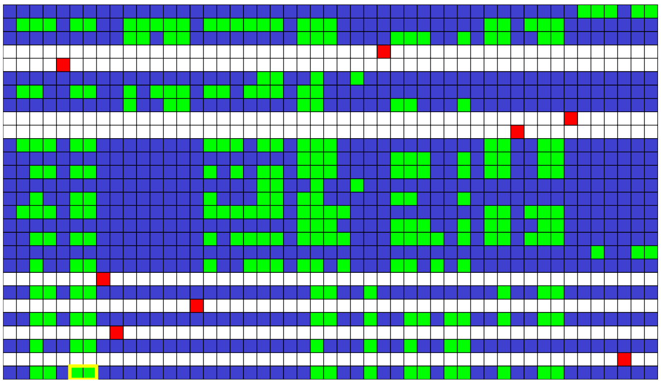


	City 1	City 2	City 3	City 4	City 5	City 6	City 7
Day 1		8			7, 5		
Day 2	2	1, 5					
Day 3	7		8				
Day 4					2	5	1
Day 5	8					1	
Day 6				5, 4			
Day 7	4				1, 3		

	City 1	City 2	City 3	City 4	City 5	City 6	City 7
Day 1	1	2	3	4	5	6	7
Day 2	8	9	10	11	12	13	14
Day 3	15	16	17	18	19	20	21
Day 4	22	23	24	25	26	27	28
Day 5	29	30	31	32	33	34	35
Day 6	36	37	38	39	40	41	42
Day 7	43	44	45	46	47	48	49

Game 12 can not be played on day 1 at locations 5 or 6

Still Missing Propagation



	City 1	City 2	City 3	City 4	City 5	City 6	City 7
Day 1		8			7, 5		
Day 2	2	1, 5					
Day 3	7		8				
Day 4					2	5	1
Day 5	8					1	
Day 6				5, 4			
Day 7	4				1, 3		

	City 1	City 2	City 3	City 4	City 5	City 6	City 7
Day 1	1	2	3	4	5	6	7
Day 2	8	9	10	11	12	13	14
Day 3	15	16	17	18	19	20	21
Day 4	22	23	24	25	26	27	28
Day 5	29	30	31	32	33	34	35
Day 6	36	37	38	39	40	41	42
Day 7	43	44	45	46	47	48	49

Game 12 can not be played on day 1 at locations 5 or 6

Our model does not deal well with hints

- Preset game is ok, leads to variable assignment
- Preset team is weak, adds new constraint
- As there is no interaction of this constraint with the other constraints, there is no initial domain restriction
- Model is correct, but lazy

Improving the handling of hints

	City 1	City 2	City 3	City 4	City 5	City 6	City 7
Day 1		8			7, 5		
Day 2	2	1, 5					
Day 3	7		8				
Day 4					2	5	1
Day 5	8					1	
Day 6				5, 4			
Day 7	4				1, 3		

- This value can not be used by pairs not involving team 8
- One of the pairs involving team 8 must use this value
- These values can not be used by any pair involving team 8

Added Program

```
improved_hint(Pos,[Value],L,Contains):-
    (foreach(X,L),foreach(A-B,Contains),
     fromto([],R,R1,Required),
     param(Pos,Value) do
         (not_mentioned(A,B,Value) ->
             X #\= Pos,R1 = R
         ;
             R1 = [X|R]
         )
    ),
    occurrences(Pos,Required,1),
     excluded_locations(Pos,Excluded),
     exclude_values(Required,Excluded).
```

Added Program

```
excluded_locations(Pos,Excluded):-
    coor(Pos,X,Y),
    (for(I,1,7),
     fromto([],A,A1,E1),
     param(Y,Pos) do
        coor(K,I,Y),
        (Pos = K ->
            A1 = A
        ;
            A1 = [K|A]
        )
    ),
    ...
```

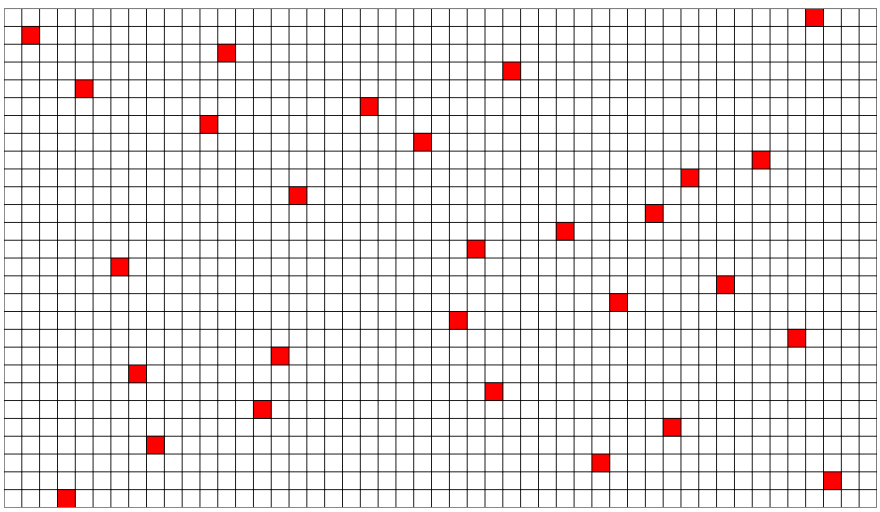

Added Program

```
    ...
    (for(J,1,7),
     fromto(E1,A,A1,Excluded),
     param(X,Pos) do
        coor(K,X,J),
        (Pos = K ->
            A1 = A
        ;
            A1 = [K|A]
        )
    ).
```

Added Program

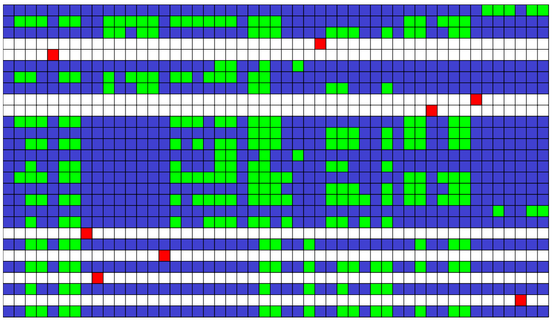
```
exclude_values(Vars,Values):-
     (foreach(X,Vars),
      param(Values) do
          (foreach(Value,Values),
           param(X) do
               X #\= Value
          )
     ).
```


Before Search

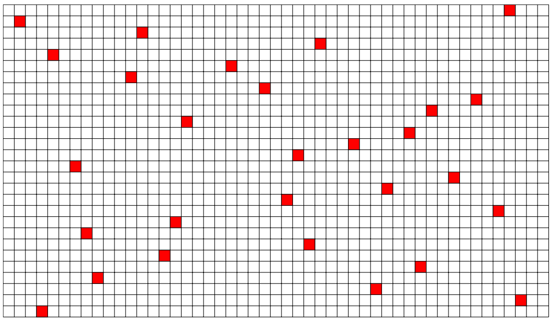


Impact of improved hint handling

With index set channeling



Improved Hints



Observation

- We don't need the value index channeling
- It is subsumed by the improved hint treatment
- Always worthwhile to check if constraints are still required after modifying model

Chapter 10: Customizing Search (Progressive Party Problem)

Helmut Simonis

Cork Constraint Computation Centre
Computer Science Department
University College Cork
Ireland

ECLiPSe ELearning Overview

Licence

Outline

What we want to introduce

- Problem Decomposition
  - Decide which problem to solve
  - Not always required to solve complete problem in one go
- Modelling with bin packing
- Customized search routines can bring dramatic improvements
- Understanding what is happening important to find improvements

Problem Definition

Progressive Party

The problem is to timetable a party at a yacht club. Certain boats are to be designated hosts, and the crews of the remaining boats in turn visit the host boats for several successive half-hour periods. The crew of a host boat remains on board to act as hosts while the crew of a guest boat together visits several hosts. Every boat can only host a limited number of guests at a time (its capacity) and crew sizes are different. The party lasts for 6 time periods. A guest boat cannot not revisit a host and guest crews cannot meet more than once. The problem facing the rally organizer is that of minimizing the number of host boats.

Data

Boat	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Capacity	6	8	12	12	12	12	12	10	10	10	10	10	8	8
Crew	2	2	2	2	4	4	4	1	2	2	2	3	4	2
Boat	15	16	17	18	19	20	21	22	23	24	25	26	27	28
Capacity	8	12	8	8	8	8	8	8	7	7	7	7	7	7
Crew	3	6	2	2	4	2	4	5	4	4	2	2	4	5
Boat	29	30	31	32	33	34	35	36	37	38	39	40	41	42
Capacity	6	6	6	6	6	6	6	6	6	6	9	0	0	0
Crew	2	4	2	2	2	2	2	2	4	5	7	2	3	4

Problem Decomposition

- Phase 1: Select minimal set of host boats
  - Manually
- Phase 2: Create plan to assign guest boats to hosts in multiple periods
  - Done as a constraint program

Idea

- Decompose problem into multiple, simpler sub problems
- Solve each sub problem in turn
- Provides solution of complete problem
- Challenge: How to decompose so that good solutions are obtained?
- How to show optimality of solution?

Selecting Host boats

- Some additional side constraints
  - Some boats must be hosts
  - Some boats may not be hosts
- Reason on total or spare capacity
- No solution with 12 boats

Cork
Constraint
Computation
Centre

Solution to Phase 1

- Select boats 1 to 12 and 14 as hosts
- Many possible problem variants by selecting other host boats

Cork
Constraint
Computation
Centre

Phase 2 Sub-problem

- Host boats and their capacity given
- Ignore host teams, only consider free capacity
- Assign guest teams to host boats

Model

- Assign guest boats to hosts for each time period
- Matrix of domain variables (size *NrGuests* $\times$ *NrPeriods*)
- Variables range over possible hosts 1..*NrHosts*

Constraints

- Each guest boat visits a host boat atmost once
- Two guest boats meet at most once
- All guest boats assigned to a host in a time period fit within spare capacity of host boat

Each guest visits a hosts atmost once

- The variables for a guest and different time periods must be pairwise different
- `alldifferent` constraint on rows of matrix

Two guests meet at most once

- The variables for two guests can have the same value for atmost one time period
- Constraints on each pair of rows in matrix

All guests assigned to a host in a time period fit within spare capacity of host boat

- Capacity constraint expressed as bin packing for each time period
- Each host boat is a bin with capacity from 0 to its unused capacity
- Each guest is an item to be assigned to a bin
- Size of item given by crew size of guest boat

Bin Packing Constraint

- Global constraint
  `bin_packing(Assignment,Sizes,Capacity)`
- Items of different sizes are assigned to bins
- Assignment of item modelled with domain variable (first argument)
- Size of items fixed: integer values (second argument)
- Each bin may have a different capacity
- Capacity of each bin given as a domain variable (third argument)

Main Program

```
top:-
    top(10,6).

top(Problem,Size):-
    problem(Problem,Hosts,Guests),
    model(Hosts,Guests,Size,Matrix),
    writeln(Matrix).
```

Data

```
problem(10,
    [10,10,9,8,8,8,8,8,8,7,6,6,4],
    [7,6,5,5,5,4,4,4,4,4,4,4,4,4,3,
     3,2,2,2,2,2,2,2,2,2,2,2,2,2]).
```


Creating Variables

```
model(Hosts,Guests,NrPeriods,Matrix):-
    length(Hosts,NrHosts),
    length(Guests,NrGuests),
    dim(Matrix,[NrGuests,NrPeriods]),
    Matrix[1..NrGuests,1..NrPeriods] :: 1..NrHosts,
    ...
```

Setting up `alldifferent` constraints

```
...
(for(I,1,NrGuests),
 param(Matrix,NrPeriods) do
    ic:alldifferent(Matrix[I,1..NrPeriods])
),
...
```


Problem
Program
Search

Setting up `bin_packing` constraints

```
...
(for(J,1,NrPeriods),
 param(Matrix,NrGuests,Guests,Hosts) do
    make_bins(Hosts,Bins),
    bin_packing(Matrix[1..NrGuests,J],
                Guests,Bins)
),
...
```

Each pair of guests meet atmost once

```
...
(for(I,1,NrGuests-1),
 param(Matrix,NrGuests,NrPeriods) do
     (for(I1,I+1,NrGuests),
      param(Matrix,NrPeriods,I) do
          card_leq(Matrix[I,1..NrPeriods],
                       Matrix[I1,1..NrPeriods],1)
     )
),
...
```


Call search

```
...
extract_array(col,Matrix,List),
assign(List).
```

Make Bin variables

```
make_bins(HostCapacity,Bins):-
    (foreach(Cap,HostCapacity),
     foreach(B,Bins) do
        B :: 0..Cap
    ).
```

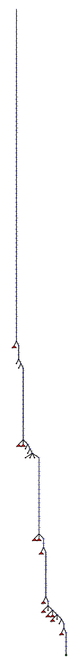

Each pair of guests meet atmost once

```
card_leq(Vector1,Vector2,Card):-
    collection_to_list(Vector1,List1),
    collection_to_list(Vector2,List2),
    (foreach(X,List1),
     foreach(Y,List2),
     fromto(0,A,A+B,Term) do
        #=(X,Y,B)
    ),
    eval(Term) #=< Card.
```

Naive Search

```
assign(List):-
    search(List,0,input_order,indomain,
           complete,[]).
```


Naive Search (Compact view)



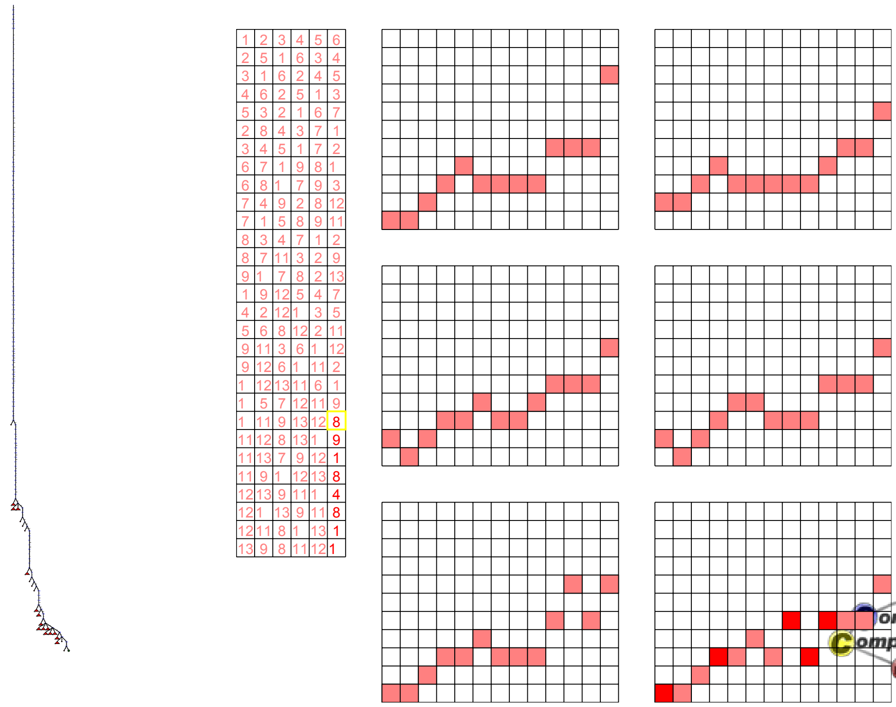
Observations

- Not too many wrong choices
- But very deep backtracking required to discover failure
- Most effort wasted in “dead” parts of search tree

First Fail strategy

```
assign(List):-
    search(List,0,first_fail,indomain,
           complete,[]).
```

First Fail Search



Observations

- Assignment not done in row or column mode
- Tree consists of straight parts without backtracking
- and nearly fully explored parts

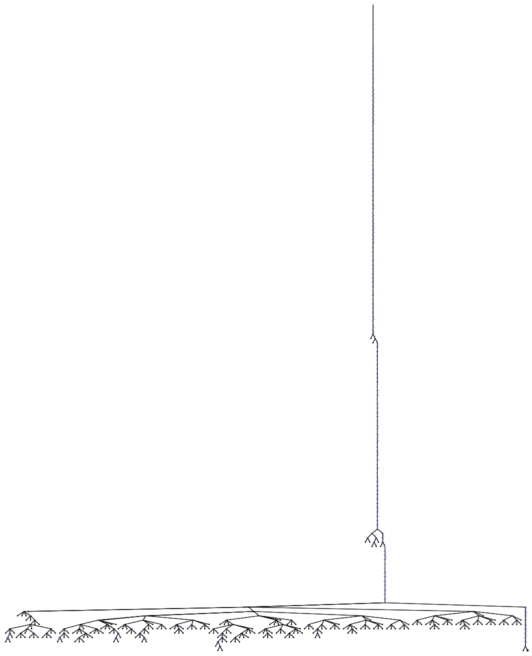
Idea

- Assign variables by time period
- Within one time period, use `first_fail` selection
- Solves bin packing packing for each period completely
- Clearer impact of disequality constraints

Layered Search

```
assign(Matrix,NrPeriods,NrGuests):-
     (for(J,1,NrPeriods),
      param(Matrix,NrGuests) do
          search(Matrix[1..NrGuests,J],0,
                 first_fail,indomain,
                 complete,[])
     ).
```

Layered Search



Observations

- Deep backtracking for last time period
- No backtracking to earlier time periods required
- Small amount of backtracking at other time periods

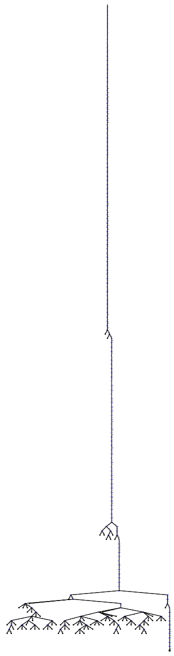
Idea

- Use credit based search
- But not for complete search tree
- Loose too much useful work
- Backtrack independently for each time period
- Hope to correct wrong choices without deep backtracking

Layered with Credit

```
assign(Matrix,NrPeriods,NrGuests):-
     (for(J,1,NrPeriods),
      param(Matrix,NrGuests) do
          NSq is NrGuests*NrGuests,
          search(Matrix[1..NrGuests,J],0,
                   first_fail,indomain,
                   credit(NSq,10),[])
     ).
```

Layered with Credit Search



Problem
Program
Search

Naive Search
First Fail Strategy
Layered Search
Layered with Credit Search
Randomized with Restart

Observations

- Improved search
- Need more sample problems to really understand impact

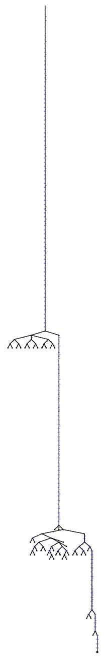
Idea

- Randomize value selction
- Remove bias picking bins in same order
- Allows to add restart
- When spending too much time without finding solution
- Restart search from beginning
- Randomization will explore other initial assignments
- Do not get caught in “dead” part of search tree

Randomized with Restart

```
assign(Matrix,NrPeriods,NrGuests):-
    repeat,
    (for(J,1,NrPeriods),
     param(Matrix,NrGuests) do
        NSq is NrGuests*NrGuests,
        once(search(Matrix[1..NrGuests,J],0,
                        first_fail,indomain_random,
                        credit(NSq,10),[]))
    ),
    !.
```

Randomized Search



Observations

- Avoids deep backtracking in last time periods
- Perhaps by mixing values more evenly
- Impose fewer disequality constraints for last periods
- Easier to find solution
- Should allow to find solutions with more time periods

Changing time periods

Problem	Size	Naive	FF	Layered	Credit	Random
10	5	0.812	1.453	1.515	0.828	1.922
10	6	14.813	2.047	2.093	1.219	2.469
10	7	79.109	3.688	50.250	3.234	3.672
10	8	-	-	141.609	55.156	6.328
10	9	-	-	-	-	10.281

Cork Constraint Computation Centre

Observations

- Randomized method is strongest for this problem
- Not always fastest for smaller problem sizes
- Restart required for size 9 problems
- Same model, very different results due to search
- Very similar results for other problem instances

Cork Constraint Computation Centre

Chapter 11: Limits of Propagation (Costas Array)

Helmut Simonis

Cork Constraint Computation Centre
Computer Science Department
University College Cork
Ireland

ECLiPSe ELearning Overview

Licence

Outline

What we want to introduce

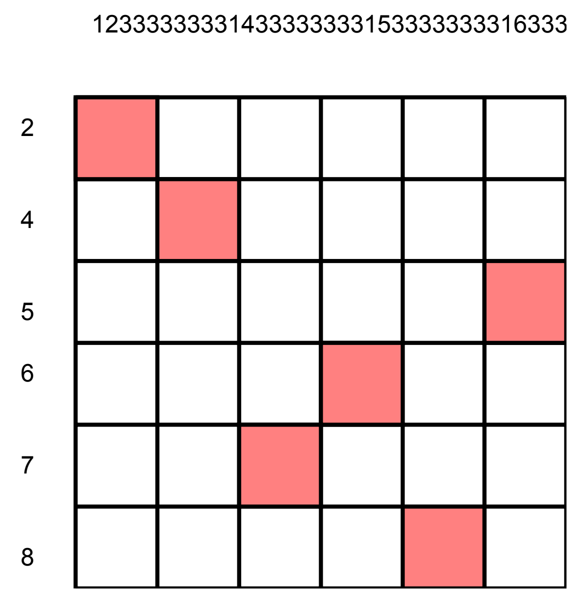
- Improving propagation does not always pay
- For some problems, simple backtracking is best
- CP may not always be the best method
- CP should always be fastest way to model problem
- Consider time to target
  - Time required to *run* program
  - Time required to *write* program
- Problem: Costas Array (Antenna design, sonar systems)

Problem Definition

Costas Array (Wikipedia)

A Costas array (named after John P. Costas) can be regarded geometrically as a set of N points lying on the squares of a NxN checkerboard, such that each row or column contains only one point, and that all of the N(N - 1)/2 vectors between each pair of dots are distinct.

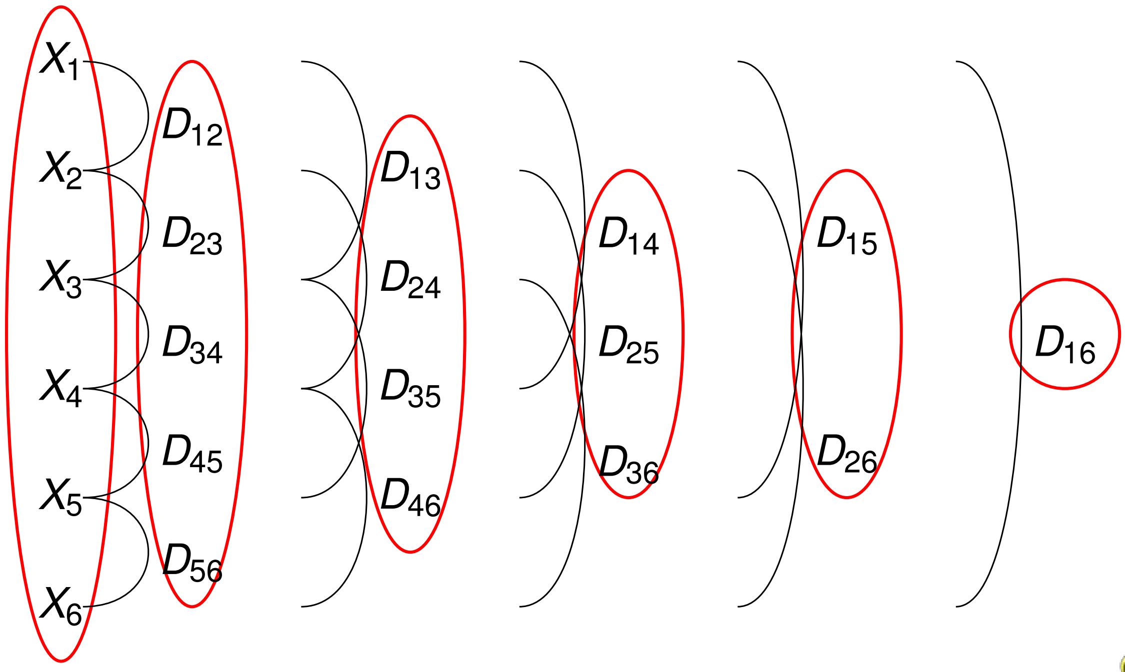
Example (Size 6)



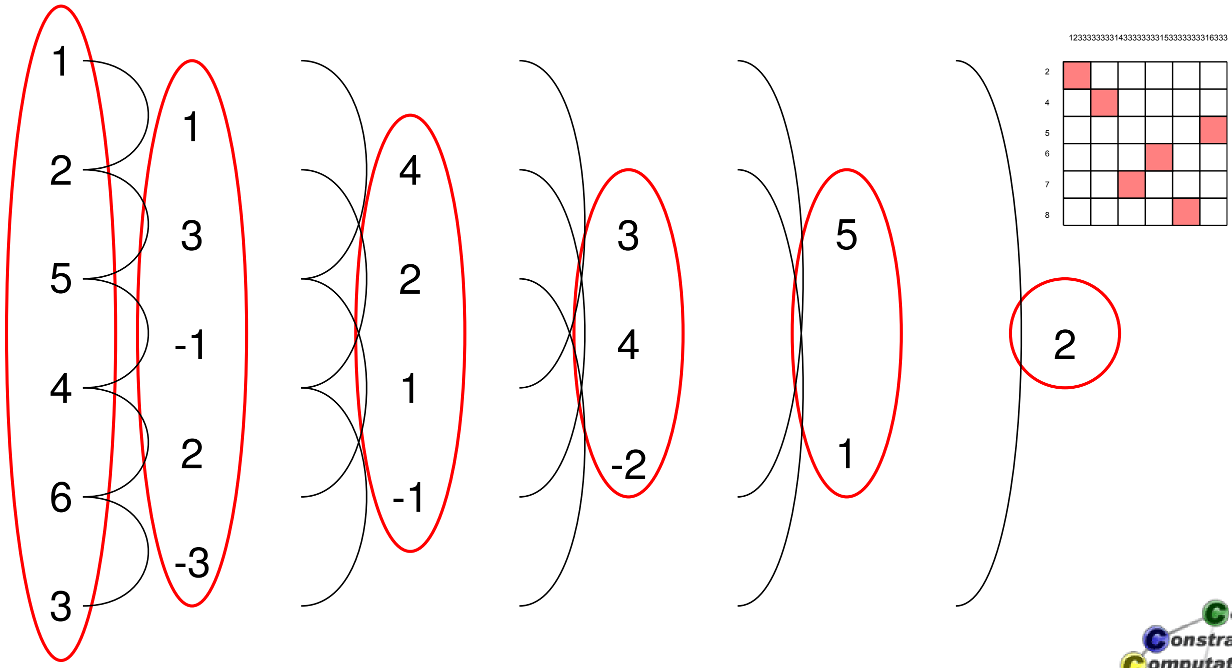
Model

- A variable for each column, ranging from 1 to N
- A list of N variables for the columns
- A difference variable between each ordered pair of variables
- `alldifferent` constraint between variables
- `alldifferent` constraints for all differences

Model



Example



Cork Constraint Computation Centre

Declarations

```
:-module(costas).

:-export(top/0).

:-lib(ic).
```

Cork Constraint Computation Centre

Main Program

```
top:-
        (for(N,3,20) do
            costas(N,_)
        ).

costas(N,L):-
        length(L,N),
        L :: 1..N,
        alldifferent(L),
        L = [_|L1],
        diffs(L,L1),
        search(L,0,first_fail,indomain,
               complete,[]).
```

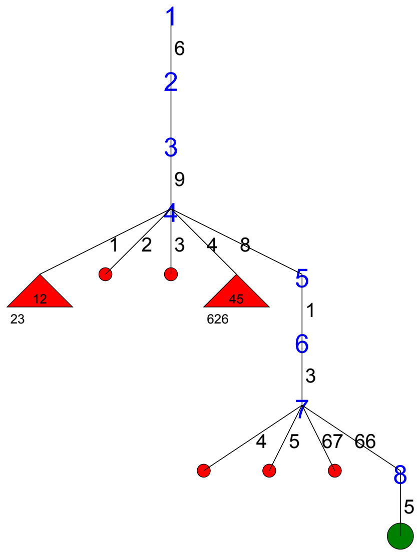
Cork Constraint Computation Centre

Differences

```
diffs(_,[]).
diffs(L,[H|T]):-
        diff_pairs(L,[H|T],Diffs),
        alldifferent(Diffs),
        diffs(L,T).

diff_pairs(_,[],[]).
diff_pairs([X|X1],[Y|Y1],[D|D1]):-
        X #= Y+D,
        diff_pairs(X1,Y1,D1).
```

Cork Constraint Computation Centre

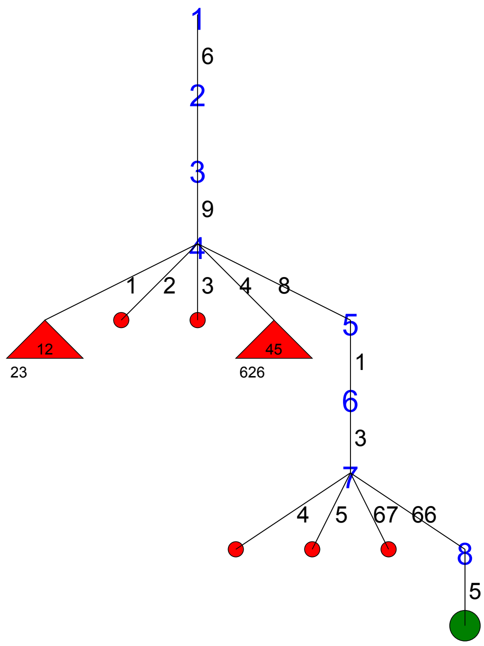
Basic Model



Other Problem Sizes

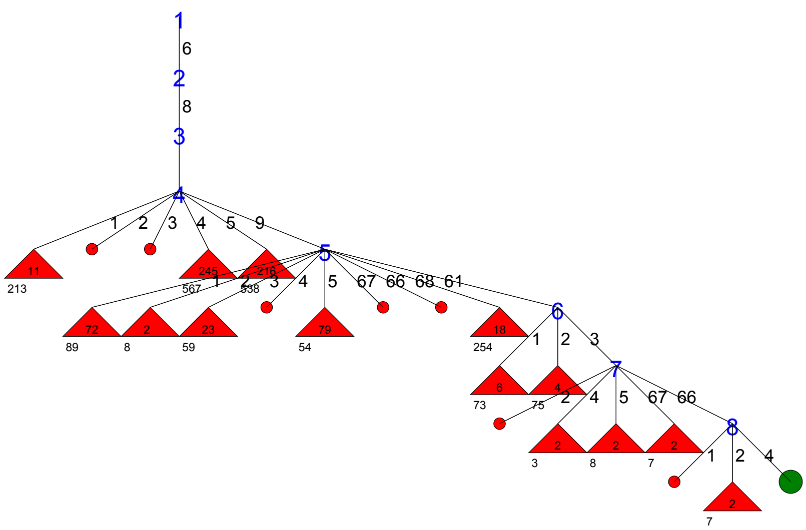
	Basic Model	
Size	Backtrack	Time
10	4	0.00
11	118	0.08
12	50	0.05
13	335	0.36
14	5008	6.23
15	47332	68.92
16	157773	271.22
17	1641685	3278.19
18	115745	283.97

Search tree (Size 12)

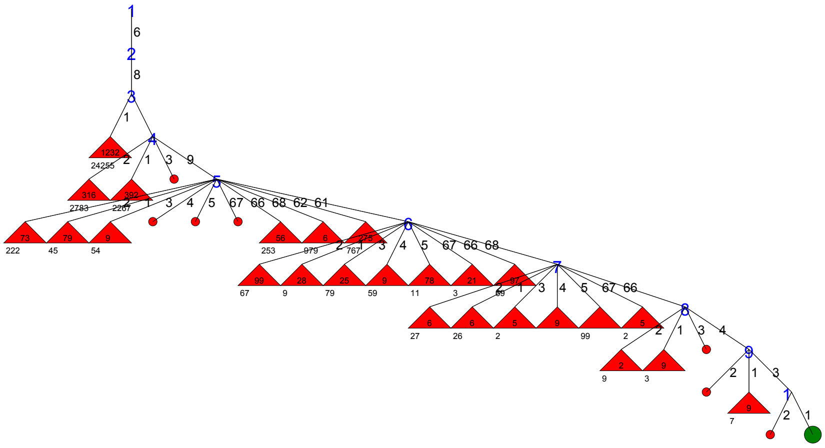


Problem
Program
Search
Improvements

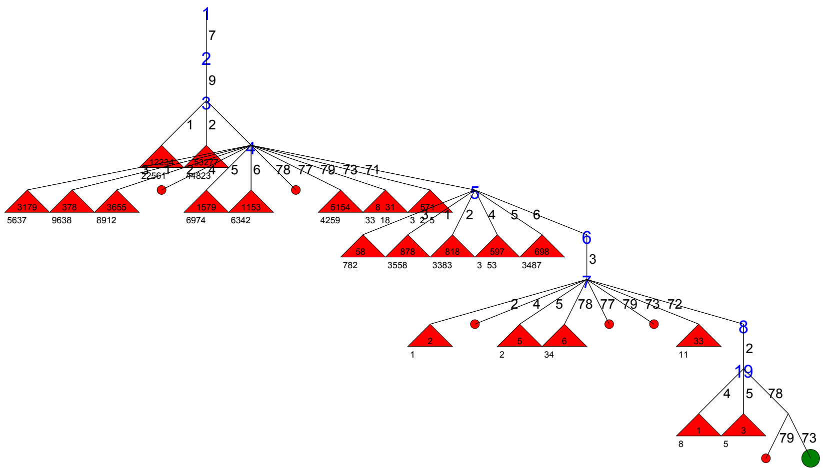
Search tree (Size 13)



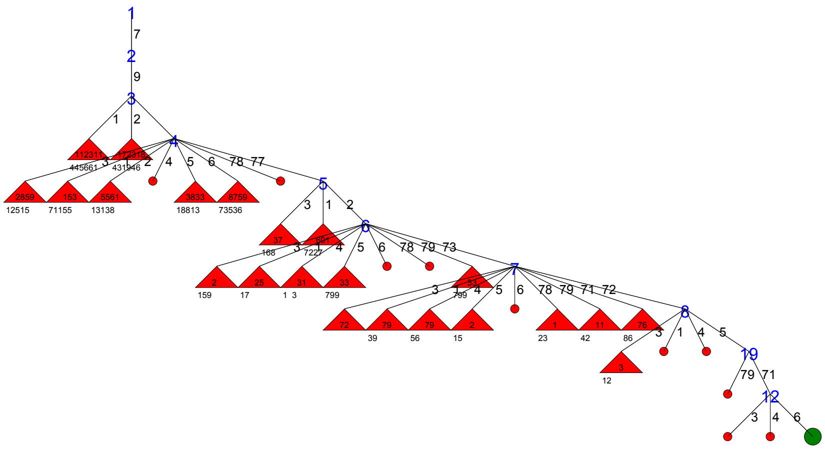
Search tree (Size 14)



Search tree (Size 15)



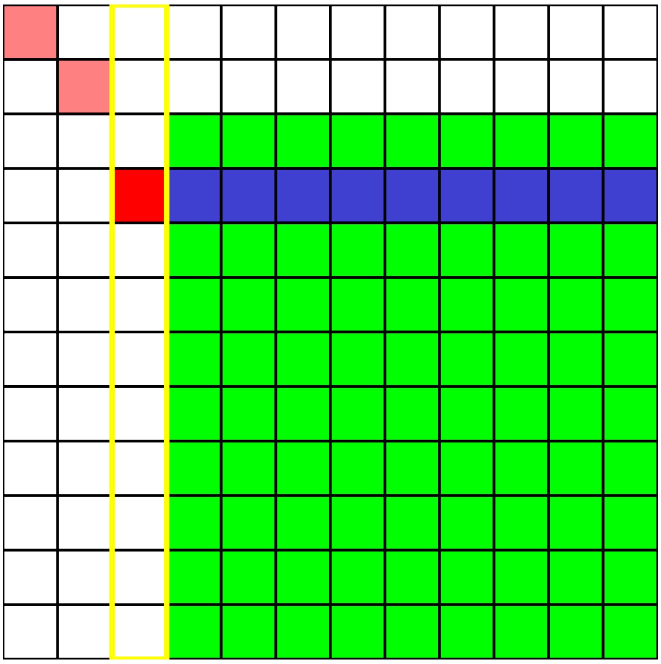
Search tree (Size 16)



Observation

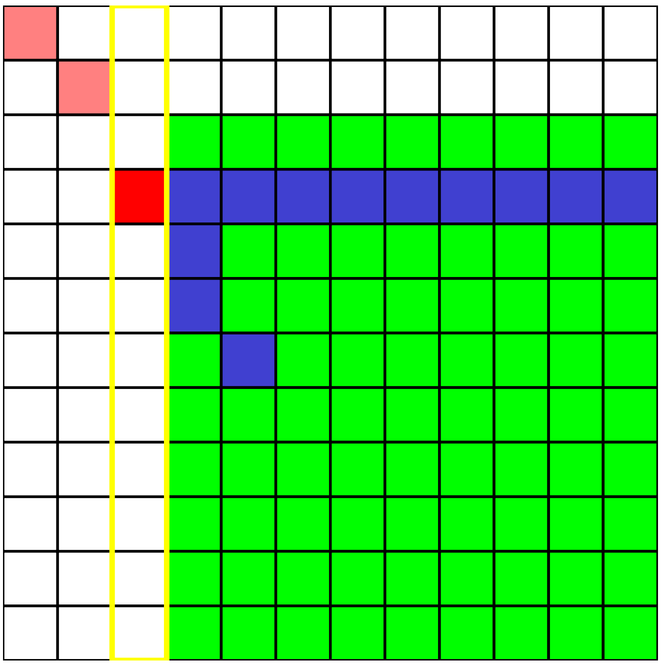
- Problem becomes harder with increasing size
- Failures occur from level 3 down
- Deep backtracking required to undo wrong choices
- Value selection not working, have to explore all choices
- Increase not uniform

Missing Propagation



The model is doing this

Missing Propagation



It could be doing that!

Changed Differences

```
diffs(_,[]).
diffs(L,[H|T]):-
        diff_pairs(L,[H|T],Diffs,Triples),
        impose_triples(Triples,[]),
        alldifferent(Diffs),
        diffs(L,T).

diff_pairs(_,[],[],[]).
diff_pairs([X|X1],[Y|Y1],[D|D1],[t(X,Y,D)|T]):-
        X #= Y+D,
        diff_pairs(X1,Y1,D1,T).
```


Changed Differences

```
impose_triples([],_).
impose_triples([t(X,Y,D)|R],Others):-
        suspend(impose_triple(D,R),4,D->inst),
        suspend(impose_triple(D,Others),4,D->inst),
        impose_triples(R,[t(X,Y,D)|Others]).
```

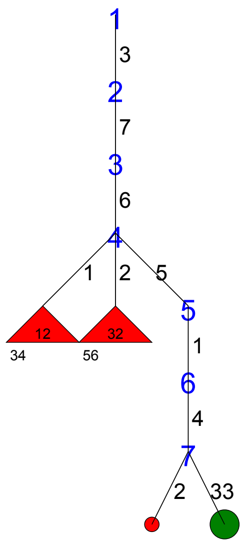
Changed Differences

```
impose_triple(_D,[]).
impose_triple(D,[t(X,Y,_)|R]):-
        (var(X) ->
            suspend(impose_one_triple12(D,X,Y),
                    4,X->inst)
        ;
            impose_one_triple12(D,X,Y)
        ),
%       ...
        impose_triple(D,R).

impose_one_triple12(D,X,Y):-
        V is X-D,
        Y #\= V.
```


Further Model Improvements

- DC consistent `alldifferent` between variables
- (DC consistent `alldifferent` between differences)
- DC difference constraint

Improved Model

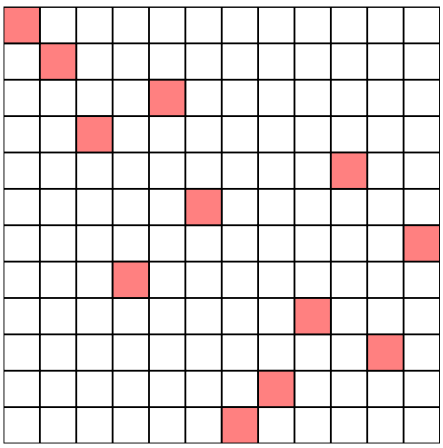


Problem
Program
Search
Improvements

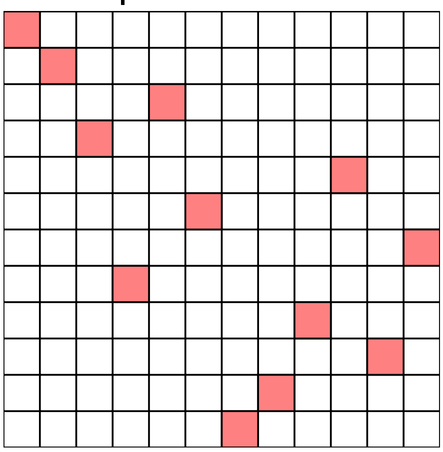
Adding Constraints
Change of Search Strategy

Comparison (Solutions)

Initial Model

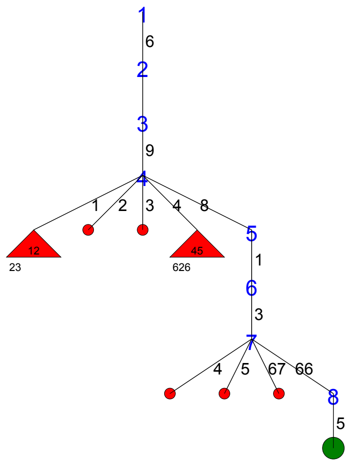


Improved Model

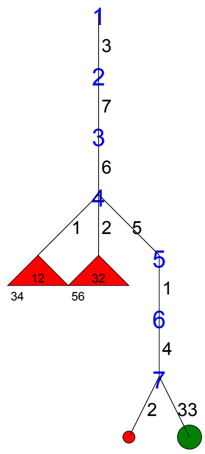


Comparison (Search Trees)

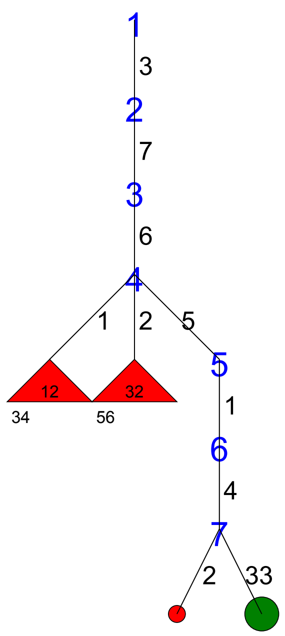
Initial Model



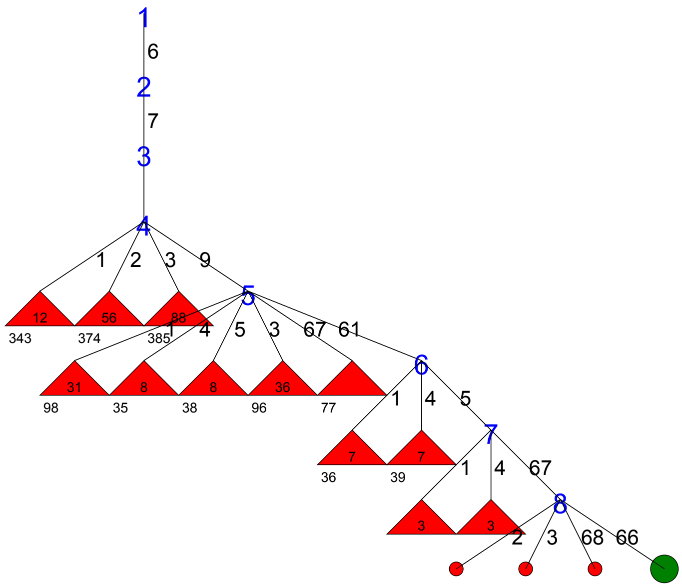
Improved Model



Search tree (Size 12)



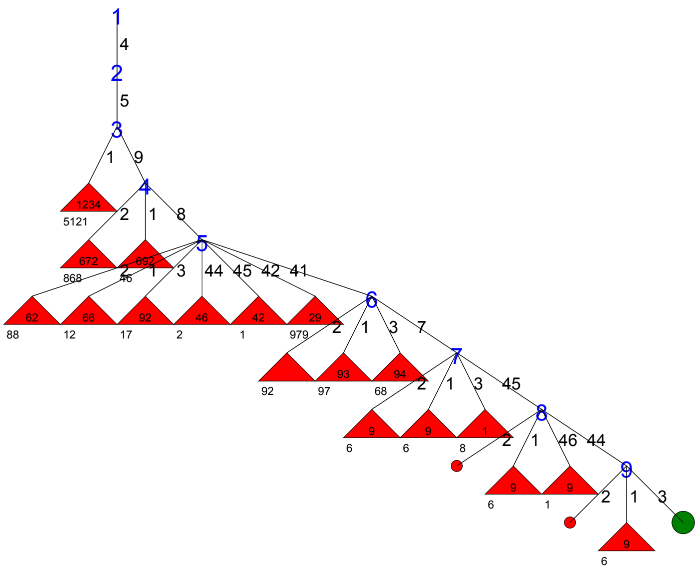
Search tree (Size 13)



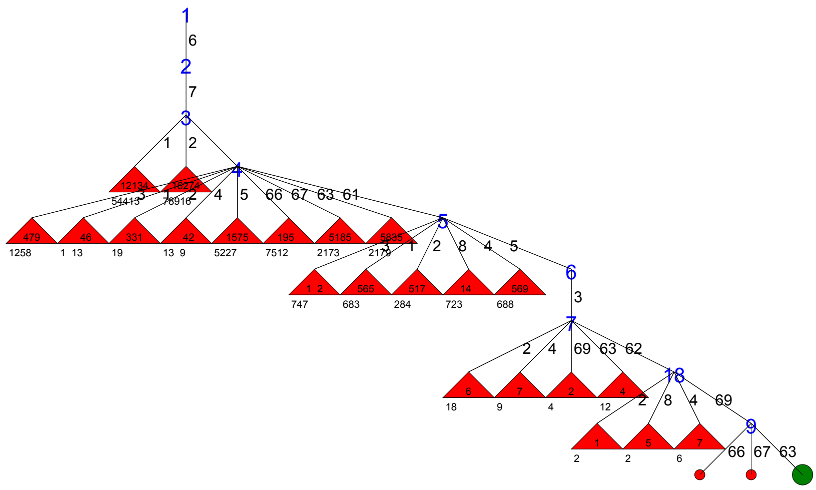
Problem
Program
Search
Improvements

Adding Constraints
Change of Search Strategy

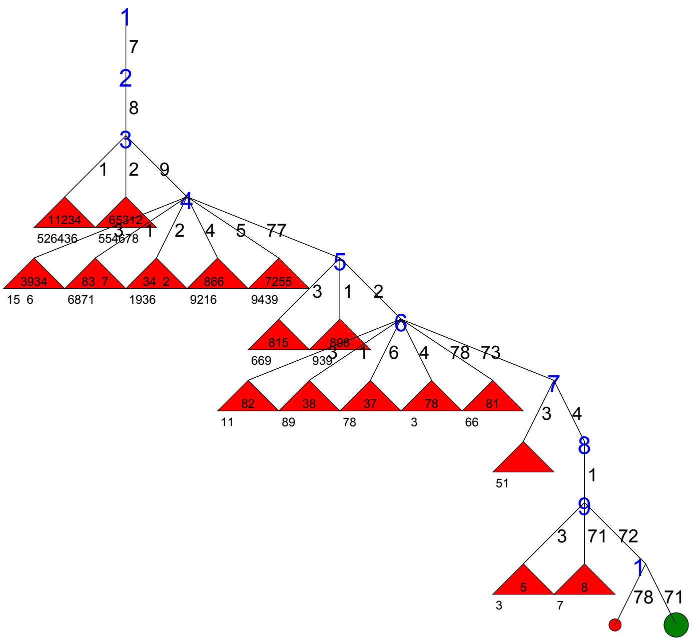
Search tree (Size 14)



Search tree (Size 15)

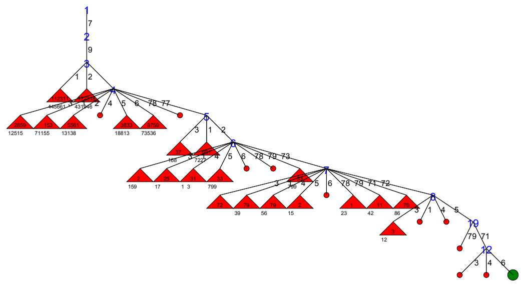


Search tree (Size 16)

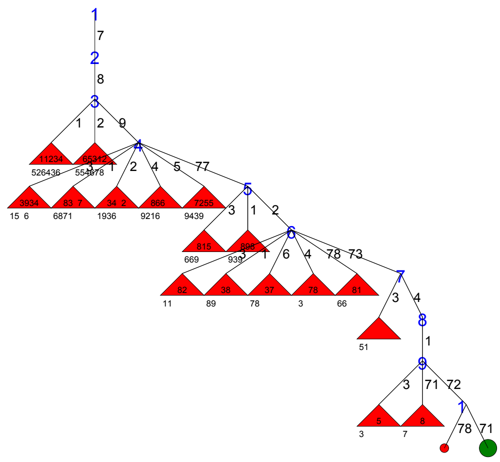


Comparison (Search Tree, size 16)

Initial Model



Improved Model



Other Problem Sizes

	Basic Model		Improved Model	
Size	Backtrack	Time	Backtrack	Time
10	4	0.00	4	0.16
11	118	0.08	77	1.44
12	50	0.05	31	0.94
13	335	0.36	216	6.22
14	5008	6.23	2875	95.94
15	47332	68.92	25820	1046.75
16	157773	271.22	84161	4099.52
17	1641685	3278.19	825590	49371.02
18	115745	283.97	55102	4530.83

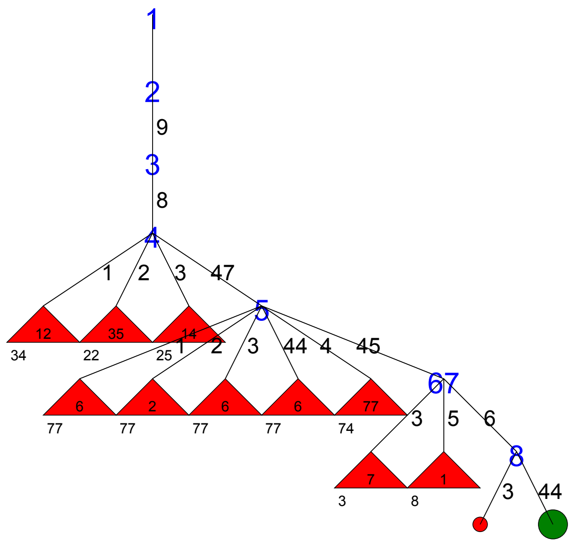
Observation

- Changes reduce backtracks by 50%
- But, run times explode
- Being clever does not always pay
- Or, perhaps, we did not make the right improvements?

Change of Search Strategy

- Idea: Make more difficult choices first
- Reorder variables to start from middle
- Assign values starting in middle

Labeling From Middle



Other Problem Sizes

	Improved Model		Improved Model, Middle	
Size	Backtrack	Time	Backtrack	Time
10	4	0.16	1	0.01
11	77	1.44	13	0.03
12	31	0.94	72	0.26
13	216	6.22	513	1.81
14	2875	95.94	589	2.37
15	25820	1046.75	7840	34.30
16	84161	4099.52	13158	63.91
17	825590	49371.02	56390	298.16
18	55102	4530.83	19750	115.64

Observation

- Big improvement in backtracks and time
- Not for all problem sizes
- Question: Do we need improvement of model for this to work?
- Experiment: Run changes search routine on basic model

Labeling Basic Model from Middle

	Basic Model		Basic Model, Middle	
Size	Backtrack	Time	Backtrack	Time
10	4	0.00	1	0.00
11	118	0.08	17	0.01
12	50	0.05	97	0.09
13	335	0.36	644	0.74
14	5008	6.23	746	1.03
15	47332	68.92	10041	16.03
16	157773	271.22	17005	31.12
17	1641685	3278.19	73080	152.72
18	115745	283.97	28837	60.97

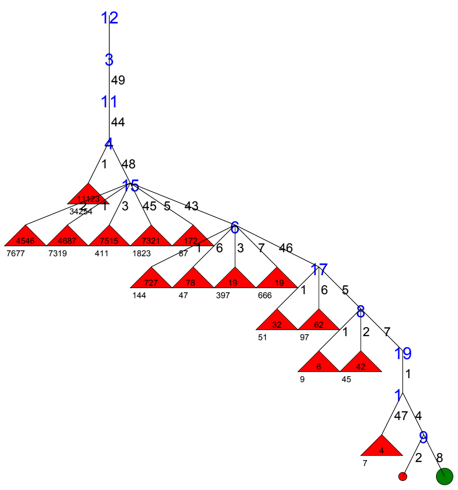
Comparison: Model Impact

	Basic Model,Middle		Improved Model, Middle	
Size	Backtrack	Time	Backtrack	Time
10	1	0.00	1	0.01
11	17	0.01	13	0.03
12	97	0.09	72	0.26
13	644	0.74	513	1.81
14	746	1.03	589	2.37
15	10041	16.03	7840	34.30
16	17005	31.12	13158	63.91
17	73080	152.72	56390	298.16
18	28837	60.97	19750	115.64
19	1187618	3174.72	1044751	4474.56

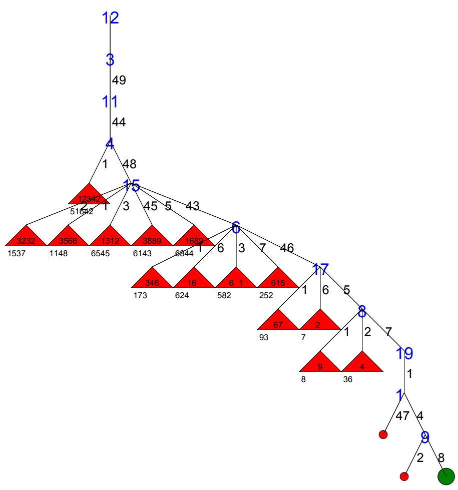
Cork Constraint Computation Centre

Comparison (Search Tree, size 18)

Initial Model



Improved Model



Cork Constraint Computation Centre

Observation

- Search strategy does not depend on model
- Variable selection is the same!
- Basic model is about two times faster
- About 50% more backtrack steps
- Again, sometimes reasoning does not pay!
- Better search strategy pays off dramatically

A Different Model

- Model shown is not the only way to express problem

0/1 Models

- SAT (Minisat)
- Pseudo Boolean (Minisat+)
- MIP (Coin-OR)

0/1 Models: Variables

- X_{iv}: Variable i takes value v
- D_{ijv}: Difference between variables i and j is v

MIP Model: Constraints

- alldifferent between variables
  - $\sum_i X_{iv} = 1$
  - $\sum_v X_{iv} = 1$
- alldifferent between differences
  - $\sum_v D_{ijv} = 1$
  - $\sum_{i-j=c} D_{ijv} \leq 1$
- link between variables and differences
  - $D_{ijv} = \sum_{v1=v2+v} X_{iv_1} X_{jv_2}$

Chapter 12: Systematic Development

Helmut Simonis

Cork Constraint Computation Centre
Computer Science Department
University College Cork
Ireland

ECLiPSe ELearning Overview

Licence

Outline

1. Introduction
2. Application Structure
3. Documentation
4. Data Representation
5. Programming Concepts
6. Style Guide

Overview

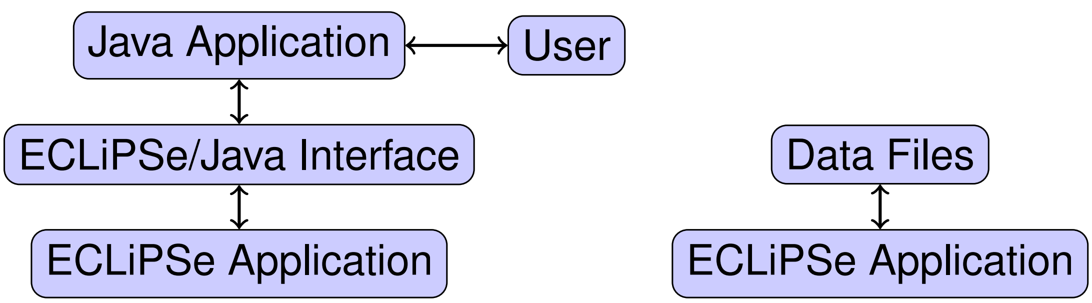
- How to develop large applications in ECLiPSe
- Software development issues for Prolog
- This is essential for large applications
  - But it may show benefits already for small programs
- This is not about problem solving, but the *boring bits* of application development

Disclaimer

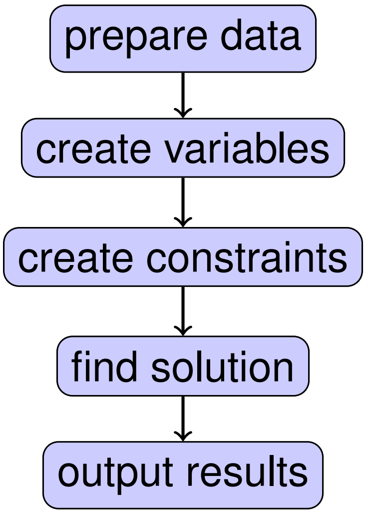
- This is not *holy writ*
  - But it works!
- This is a team issue
  - People working together must agree
  - Come up with a local style guide
- Consistency is not optional
  - Every shortcut must be paid for later on
- This is an appetizer only
  - The real story is in the tutorial Developing Applications with ECLiPSe (part of the ECLiPSe documentation)

Application Structure

Full Application | Batch Application



LSCO Structure



Top-Down Design

- Design queries
- UML static class diagram (structure definitions)
- API document/test cases
- Top-level structure
- Data flow analysis
- Allocate functionality to modules
- Syntactic test cases
- Module expansion
  - Using programming concepts where possible
  - Incremental changes

Modules

- Grouping of predicates which are related
- Typically in a single file
- Defined external interfaces
  - Which predicates are exported
  - Mode declaration for arguments
  - Intended types for arguments
  - Documentation
- Helps avoid Spaghetti structure of program

Creating Documentation

- Your program can be documented in the same way as ECLiPSe library predicates
- Comment directives in source code
- Tools to extract comments and produce HTML documentation with hyper-links
- Quality depends on effort put into comments
- Every module interface should be documented

Example

```
:- comment(prepare_data/4,[
    summary:"creates the data structures
for the flow analysis",
    amode:prepare_data(+,+,+,-),
    args:[
"Dir":"directory for report output",
"Type":"the type of report to be generated",
"Summary":"a summary term",
"Nodes":"a nodes data structure"],
    desc:html("
This routine creates the data
structures for the flow analysis.
...
```



```
    see_also:[hop/3]
```


External Data Representation

Property	Argument	Data File	Term File	Facts	EXDR
Multiple runs	++	+	+	-	+
Debugging	-	+	+	++	-
Test generation effort	-	+	+	+	-
Java I/O Effort	-	+	-	-	+
ECLiPSe I/O Effort	++	+	++	++	++
Memory	++	-	-	--	-
Develoment Effort	+	-	+	+	-

Internal Data Representation

- Named structures
  - Define & document properly
- Lists
  - Do not use for fixed number of elements
- Hash tables, e.g. lib(hash)
  - Efficient
  - Extensible
  - Multiple keys possible
- Vectors & arrays
  - Requires that keys are integers (tuples)
- Multi-representation
  - Depending on key use one of multiple representations

Internal Representation Comparison

	Named Structures	Lists	Hash Tables	Vectors Arrays	Multi-representation
hold disparate data	++	–	–	–	–
access specific info	+	–	+	+	+
add new entries	–	+	++	–	–
do loops	+	++	-	++	++
sort entries	–	++	-	-	++
index calculations	-	–	–	++	+

Getting it to work

- Early testing `lib(test_util)`
  - Define what a piece of code should do by example
  - May help to define behaviour
- Stubs
- Line coverage `lib(coverage)`
  - Check that tests cover code base
- Heeding warnings of compiler, `lib(lint)`
  - Eliminate all causes of warnings
  - Singleton warnings typically hide more serious problems
- Small, incremental changes
  - Matter of style
  - Works for most people

Programming Concepts

- Many programming tasks are similar
  - Finding the right information
  - Putting things together in the right sequence
- We don't need the fastest program, but the easiest to maintain
  - Squeezing the last 10% improvement normally does not pay
- Avoid unnecessary inefficiency
  - `lib(profile), lib(port_profiler)`

List of concepts

- Alternatives
- Iteration (list, terms, arrays)
- Transformation
- Filtering
- Combine
- Minimum/Best and rest
- Sum
- Merge
- Group
- Lookup
- Cartesian
- Ordered pairs

Example: Cartesian

```
:-mode cartesian(+,+,-).
cartesian(L,K,Res):-
        (foreach(X,L),
         fromto([],In,Out,Res),
         param(K) do
             (foreach(Y,K),
              fromto(In,In1,[pair(X,Y)|In1],Out),
              param(X) do
                  true
             )
        ).
```

Input/Output

- Section on DCG use
  - Grammars for parsing and generating text formats
- XML parser in ECLiPSe
  - `lib(xml)`
- EXDR format to avoid quoting/escaping problems
- Tip:
  - Generate hyper-linked HTML/SVG output to present data/results as development aid

If it doesn't work

- Understand what happens
  - Which program point should be reached with which information?
  - Why do we not reach this point?
  - Which data is wrong/missing?
- Do not trace through program!
- Debugging is like solving puzzles
  - Pick up clues
  - Deduce what is going on
  - Do not simulate program behaviour!

Correctness and Performance

- Testing
- Profiling
- Code Reviews
  - Makes sure things are up to a certain standard
  - Don't expect reviewer to find bugs
- Things to watch out for
  - Unwanted choice points
  - Open streams
  - Modified global state
  - Delayed goals

Did I mention testing?

- Single most important/neglected activity
- Re-test directly after every change
  - Identifies faulty modification
  - Avoids lengthy debugging session after making 100s of changes
- Independent verification
  - Check results by hand (?)
  - By other program (??)
  - Use constraint solver as checker

Style Guide

- Rules that should be satisfied by finished program
- Things may be relaxed during prototyping
- Often, choice among valid alternatives is made arbitrarily, so that a consistent way is defined
- If you don't like it, change it!
  - But: better a bad rule than no rule at all!

Style Guide Examples

- There is one directory containing all code and its documentation (using sub-directories).
- Filenames are of form `[a-z][a-z_]+` with extension `.ecl` .
- One file per module, one module per file.
- Each module is documented with comment directives.
- ...
- Don't use `','/2` to make tuples.
- Don't use lists to make tuples.
- Avoid `append/3` where possible, use accumulators instead.

Layout rules

- How to format ECLiPSe programs
- Pretty-printer format
- Eases
  - Exchange of programs
  - Code reviews
  - Bug fixes
  - Avoids extra reformatting work

Core Predicates List

- Alphabetical predicate index lists 2940 entries
  - You can't possibly learn all of them
  - Do you really want to know what `set_typed_pool_constraints/3` does?
- List of Prolog predicates you need to know
  - 69 entries, more manageable
- Ignores all solver libraries
- If you don't know what an entry does, find out about it
  - what does `write_exdr/2` do?
- If you use something not on the list, start to wonder...

Other Sources

- Developing Applications with ECLiPSe
  - H. Simonis
  - `http://www.eclipse-clp.org`
- Constraint Logic Programming Using ECLiPSe
  - K. Apt, M. Wallace
  - Cambridge University Press
- The Craft of Prolog
  - R.O'Keefe, MIT Press

Cork Constraint Computation Centre

Conclusions

- Large scale applications can be built with ECLiPSe
- Software engineering is not that different for Prolog
- Many tasks are similar regardless of solver used
- Correctness of program is useful even for research work

Cork Constraint Computation Centre

Chapter 13: Visualization Techniques

Helmut Simonis

Cork Constraint Computation Centre
Computer Science Department
University College Cork
Ireland

ECLiPSe ELearning Overview

Licence

Outline

What we want to introduce

- How to visualize constraint programs
- Variable visualizers
- Understanding search trees
- Constraint visualizers
- Complex visualizations

Chapter 14: Finite Set and Continuous Variables - SONET Design Problem

Helmut Simonis

Cork Constraint Computation Centre
Computer Science Department
University College Cork
Ireland

ECLiPSe ELearning Overview

Licence

Outline

What we want to introduce

- Finite set variables
- Continuous domains
- Optimization from below
- Advanced symmetry breaking
- SONET design problem without inter-ring flows

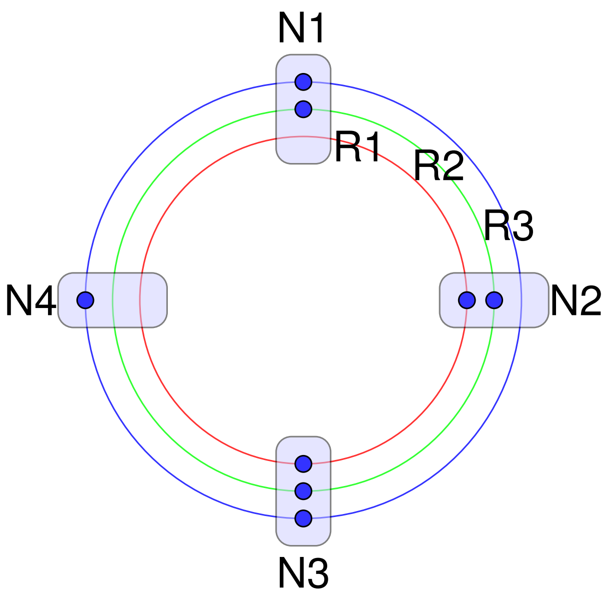
Problem Definition

SONET Design Problem

We want to design a network with multiple SONET rings, minimizing ADM equipment. Traffic can only be transported between nodes connected to the same ring, not between rings. Traffic demands between nodes are given. Decide which nodes to place on which ring(s), respecting a maximal number of ADM per ring, and capacity limits on ring traffic. If two nodes are connected on more than one ring, the traffic between them can be split arbitrarily between the rings. The objective is to minimize the overall number of ADM.

Cork Constraint Computation Centre

Example



3 rings, 4 nodes, 8 ADMEvery node connected to at least one ringOn every ring are at least two nodes$N1$ connected to $R2$ and $R3$$N4$ and $N2$ can't talk to each otherTraffic between $N1$

Cork Constraint Computation Centre

$R1$ or $R2$, or both

Data

- Demands $d \in D$ between nodes f_d and t_d of size s_d
- Rings R, total of $|R| = r$ rings
- Each ring has capacity c
- Nodes N

Model

- Primary model integer 0/1 variables x_{ik}
  - Node i has a connection to ring k
  - A node can be connected to more than one ring
- Continuous [0..1] variables f_{dk}
  - Which fraction of total traffic of demand d is transported on ring k
  - A demand can use a ring only if both end-points are connected to it

Constraints

$$\min \sum_{i \in N} \sum_{k \in R} x_{ik}$$

s.t.

$$
\begin{aligned}
\sum_{i \in N} x_{ik} &\leq r && (1)\\
\sum_{k \in R} f_{dk} &= 1 && (2)\\
\sum_{d \in D} s_d * f_{dk} &\leq c && (3)\\
f_{dk} &\leq x_{f_d k} && (4)\\
f_{dk} &\leq x_{t_d k} && (5)
\end{aligned}
$$

Dual Models

- Introducing finite set variables
- Range over sets of integers, not just integers
- Most useful when we don't know the number of items involved
- Here: for each node, the rings on which it is placed
- Could be one, could be two, or more
- Hard to express with finite domain variables alone

Dual Model 1

- Finite set variables N_i
  - Which rings node i is connected to
- Cardinality finite domain variables n_i
  - $|N_i| = n_i$

Dual Model 2

- Finite set variables R_k
  - Which nodes ring k is connected to
- Cardinality finite domain variables r_k
  - $|R_k| = r_k$

Channeling between models

- Use the zero/one model as common ground
- $x_{ik} = 1 \Leftrightarrow k \in N_i$
- $x_{ik} = 1 \Leftrightarrow i \in R_k$

Constraints in dual models

- For every demand, source and sink must be on (at least one) shared ring
  - $\forall d \in D: \quad |N_{f_d} \cap N_{t_d}| \geq 1$
- Every node must be on a ring
  - $n_i \geq 1$
- A ring can not have a single node connected to it
  - $r_k \neq 1$

Assignment Strategy

- Cost based decomposition
- Assign total cost first
- Then assign n_i variables
- Finally, assign x_{ik} variables
- If required, fix flow f_{dk} variables
- Might leave flows as bound-consistent continuous domains

Optimization from below

- Optimization handled by assigning cost first
- Enumerate values increasing from lower bound
- First feasible solution is optimal
- Depends on proving infeasibility rapidly
- Does not provide sub-optimal initial solutions

Redundant Constraints

- Deduce bounds in n_i variables
  - Helps with finding n_i assignment which can be extended
- Symmetry Breaking

Symmetries

- Typically no symmetries between demands
- Full permutation symmetry on rings
- Gives $r!$ permutations
- These must be handled somehow
- Further symmetries if capacity seen as discrete channels

Symmetry Breaking Choices

- As part of assignment routine
  - SBDS (symmetry breaking during search)
  - Define all symmetries as parameter
  - Search routine eliminates symmetric sub-trees
- By stating ordering constraints
  - As shown in the BIBD example
  - Ordering constraints not always compatible with search heuristic
  - Particular problem of dynamic variable ordering

Defining finite set variables

- Library `ic_sets`
- Domain definition `X :: Low..High`
  - *Low*, *High* sets of integer values, e.g. `[1,3,4]`
- or `intsets(L,N,Min,Max)`
  - `L` is a list of `N` set variables
  - each containing all values between `Min` and `Max`

Using finite set variables

- Set Expressions: `A /\ B`, `A \/ B`
- Cardinality constraint: `#(Set,Size)`
  - *Size* integer or finite domain variable
- `membership_booleans(Set,Booleans)`
  - Channeling between set and 0/1 integer variables

Using continuous variables

- Library `ic` handles both
  - Finite domain variables
  - Continuous variables
- Use floats as domain bounds, e.g. `X :: 0.0 .. 1.0`
- Use `$=` etc for constraints instead of `#=`
- Bounds reasoning similar to finite case
- But must deal with safe rounding
- Not all constraints deal with continuous variables

Ambiguous Import

- Multiple solvers define predicates like `::`
- If we load multiple solvers in the same module, we have to tell ECLiPSe which one to use
- Compiler does not deduce this from context!
- So
  - `ic:(X :: 1..3)`
  - `ic_sets:(X :: [] .. [1,2,3])`
- Otherwise, we get loads of error messages
- Happens whenever two modules export same predicate

Top-level predicate

```
:-module(sonet).
:-export(top/0).
:-lib(ic),lib(ic_global),lib(ic_sets).

top:-
    problem(NrNodes,NrRings,Demands,
            MaxRingSize,ChannelSize),
    length(Demands,NrDemands),
    ...
```

Matrix of x_{ik} integer variables

```
...
dim(Matrix,[NrNodes,NrRings]),
ic:(Matrix[1..NrNodes,1..NrRings] :: 0..1),
...
```


Node and ring set variables

```
...
dim(Nodes,[NrNodes]),
intsets(Nodes[1..NrNodes],NrNodes,1,NrRings),
dim(NodeSizes,[NrNodes]),
ic:(NodeSizes[1..NrNodes] :: 1..NrRings),
dim(Rings,[NrRings]),
intsets(Rings[1..NrRings],NrRings,1,NrNodes),
dim(RingSizes,[NrRings]),
ic:(RingSizes[1..NrRings] :: 0..MaxRingSize),
...
```

Channeling node set variables

```
...
(for(I,1,NrNodes),
 param(Matrix,Nodes,NodeSizes,NrRings) do
    subscript(Nodes,[I],Node),
    subscript(NodeSizes,[I],NodeSize),
    #(Node,NodeSize),
    membership_booleans(Node,
                        Matrix[I,1..NrRings])
),
...
```


Channeling ring set variables

```
...
(for(J,1,NrRings),
 param(Matrix,Rings,RingSizes,NrNodes) do
    subscript(Rings,[J],Ring),
    subscript(RingSizes,[J],RingSize),
    RingSize #\= 1,
    #(Ring,RingSize),
    membership_booleans(Ring,
                        Matrix[1..NrNodes,J])
),
...
```

Demand ends must be (on atleast one) same ring

```
...
(foreach(demand(I,J,_Size),Demands),
 param(Nodes,NrRings) do
     subscript(Nodes,[I],NI),
     subscript(Nodes,[J],NJ),
     ic:(NonZero :: 1..NrRings),
     #(NI /\ NJ,NonZero)
),
...
```


Flow Variables

```
...
dim(Flow,[NrDemands,NrRings]),
ic:(Flow[1..NrDemands,1..NrRings]::0.0 .. 1.0),
(for(I,1,NrDemands),
 param(Flow,NrRings) do
     (for(J,1,NrRings),
      fromto(0.0,A,A+F,Term),
      param(Flow,I) do
     subscript(Flow,[I,J],F)
     ),
     eval(Term) $= 1.0
),
...
```

Ring Capacity Constraints

```
...
(for(I,1,NrRings),
 param(Flow,Demands,ChannelSize) do
     (foreach(demand(_,_,Size),Demands),
      count(J,1,_),
      fromto(0.0,A,A+Size*F,Term),
      param(Flow,I) do
     subscript(Flow,[J,I],F)
     ),
     eval(Term) $=< ChannelSize
),
...
```


Linking x_{ik} and f_{dk} variables

```
...
(foreach(demand(From,To,_),Demands),
 count(I,1,_),
 param(Flow,Matrix,NrRings) do
     (for(K,1,NrRings),
      param(I,From,To,Flow,Matrix) do
     subscript(Flow,[I,K],F),
     subscript(Matrix,[From,K],X1),
     subscript(Matrix,[To,K],X2),
     F $=< X1,
     F $=< X2
     )
),
...
```

Setting up degrees

```
...
dim(Degrees,[NrNodes]),
(for(I,1,NrNodes),
 param(Degrees) do
    subscript(Degrees,[I],Degree),
    neighbors(I,Neighbors),
    length(Neighbors,Degree)
),
...
```


Defining cost and assigning values

```
...
sumlist(NodeSizesList,Cost),
assign(Cost,Handle,NrNodes,Degrees,
       NodeSizes,Matrix).
```

Assignment Routines

```
assign(Cost,Handle,NrNodes,Degrees,
       NodeSizes,Matrix):-
    indomain(Cost),
    order_sizes(NrNodes,Degrees,NodeSizes,
                OrderedSizes),
    search(OrderedSizes,1,input_order,indomain,
           complete,[]),
    order_vars(Degrees,NodeSizes,Matrix,
               VarAssign),
    search(VarAssign,0,input_order,indomain_max,
           complete,[]).
```


Order ring size variables by increasing degree

```
order_sizes(NrNodes,Degrees,NodeSizes,
            OrderedSizes):-
    (for(I,1,NrNodes),
     foreach(t(X,D),Terms),
     param(Degrees,NodeSizes) do
        subscript(Degrees,[I],D),
        subscript(NodeSizes,[I],X)
    ),
    sort(2,=<,Terms,OrderedSizes).
```

Ordering decision variables

```
order_vars(Degrees,NodeSizes,Matrix,VarAssign):-
    dim(Matrix,[NrNodes,NrRings]),
    (for(I,1,NrNodes),
     foreach(t(Size,Y,I),Terms),
     param(Degrees,NodeSizes) do
        subscript(NodeSizes,[I],Size),
        subscript(Degrees,[I],Degree),
        Y is -Degree
    ),
    sort(0,=<,Terms,Sorted),
    ...
```


Problem
Program
Search
Conclusions

Ordering decision variables

```
    ...
    (foreach(t(_,_,I),Sorted),
     fromto(VarAssign,A1,A,[]),
     param(NrRings,Matrix) do
        (for(J,1,NrRings),
         fromto(A1,[X|AA],AA,A),
         param(I,Matrix) do
            subscript(Matrix,[I,J],X)
        )
    ).
```

Data (13 nodes, 7 rings, 24 demands)

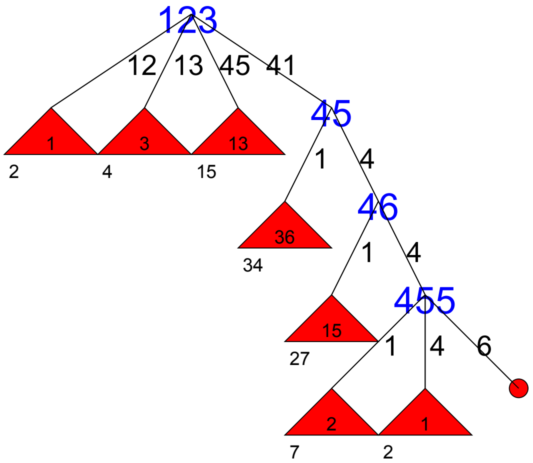
```
problem(13,7,
     [demand(1,9,8),demand(1,11,2),demand(2,3,25),
      demand(2,5,5),demand(2,9,2),demand(2,10,3),
      demand(2,13,4),demand(3,10,2),demand(4,5,4),
      demand(4,8,1),demand(4,11,5),demand(4,12,2),
      demand(5,6,5),demand(5,7,4),demand(7,9,5),
      demand(7,10,2),demand(7,12,6),demand(8,10,1),
      demand(8,12,4),demand(8,13,1),demand(9,12,5),
      demand(10,13,9),demand(11,13,3),
      demand(12,13,2)
     ],5,40).
```

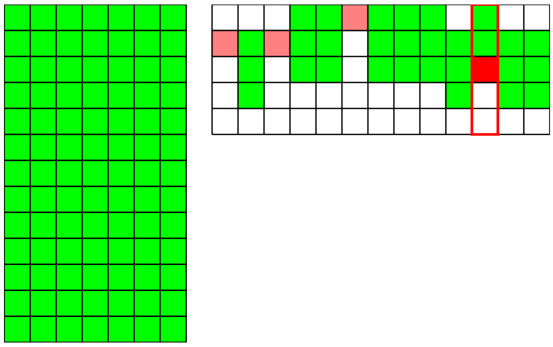

Problem
Program
Search
Conclusions

Neighbors of a node

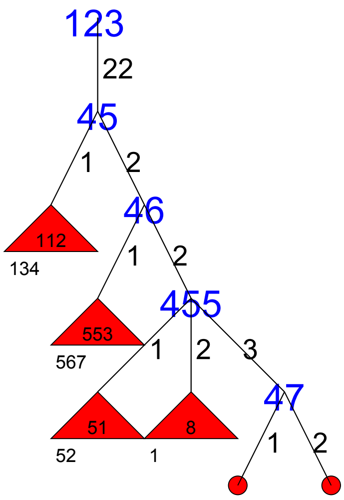
```
neighbors(N,List):-
    problem(_,_,Demands,_,_),
    (foreach(demand(I,J,_),Demands),
     fromto([],A,A1,List),
     param(N) do
        (N = I ->
            A1 = [J|A]
        ; N = J ->
            A1 = [I|A]
        ;
            A1 = A
        )
    ).
```

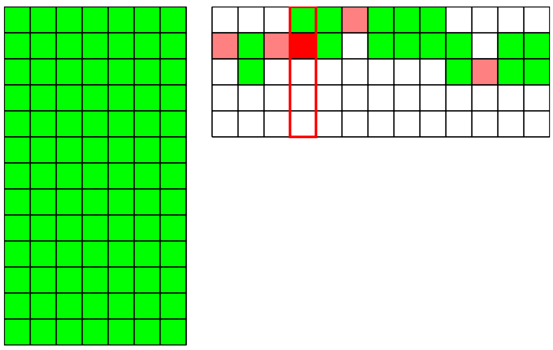
Search at Cost 18-21



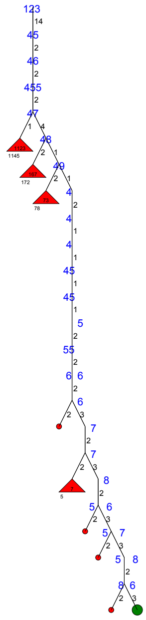


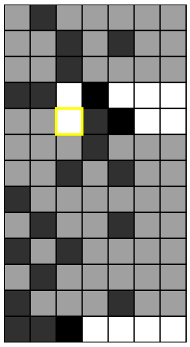
Search at Cost 22

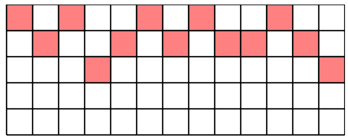




Search at Cost 23







Conclusions

- Introduced finite set and continuous domain solvers
- Finite set variables useful when values are sets of integers
- Useful when number of items assigned are unknown
- Can be linked with finite domains (cardinality) and 0/1 index variables

Continuous domain variables

- Allow to reason about non-integral values
- Bound propagation similar to bound propagation over integers
- Difficult to enumerate values
- Assignment by domain splitting

SONET Problem

- Example of optical network problems
- Competitive solution by combination of techniques
- Channeling, redundant constraints, symmetry breaking
- Decomposition by branching on objective value

Chapter 15: Network Applications

Helmut Simonis

Cork Constraint Computation Centre
Computer Science Department
University College Cork
Ireland

ECLiPSe ELearning Overview

Licence

Outline

Common Theme

- How can we get better performance out of a given network?
- Make network transparent
  - Users should not need to know about details
  - Service maintained even if failures occur
- Restricted by accepted techniques available in hardware
  - Interoperability between multi-vendor equipment
  - Very conversative deployment strategies

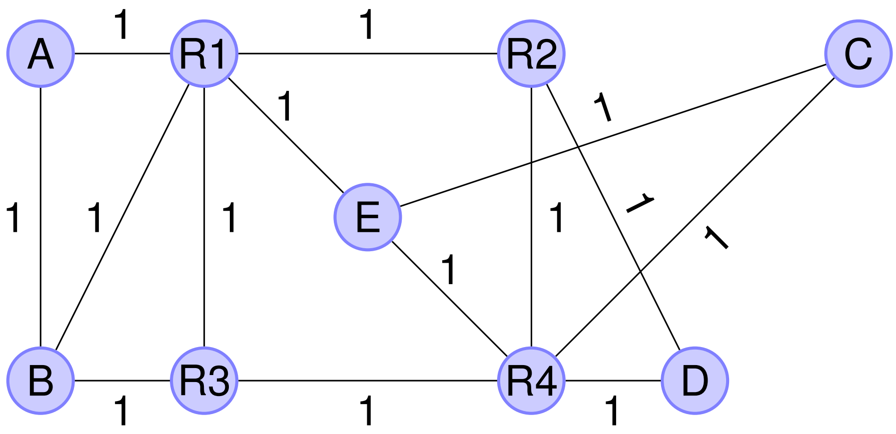
Reminder: IP Networks

- Packet forwarding
- Connection-less
- Destination based routing
  - Distributed routing algorithm based on shortest path algorithm
  - Routing metric determines preferred path
- Best effort
  - Packets are dropped when there is too much traffic on interface
  - Guaranteed delivery handled at other layers (TCP/applications)

Disclaimer

- Flexible border between CP and OR
- CP is ...
  - what CP people do.
  - what is published in CP conferences.
  - what uses CP languages.
- Does not mean that other approaches are less valid!

Example Network (Uniform metric 1, Capacity 100)

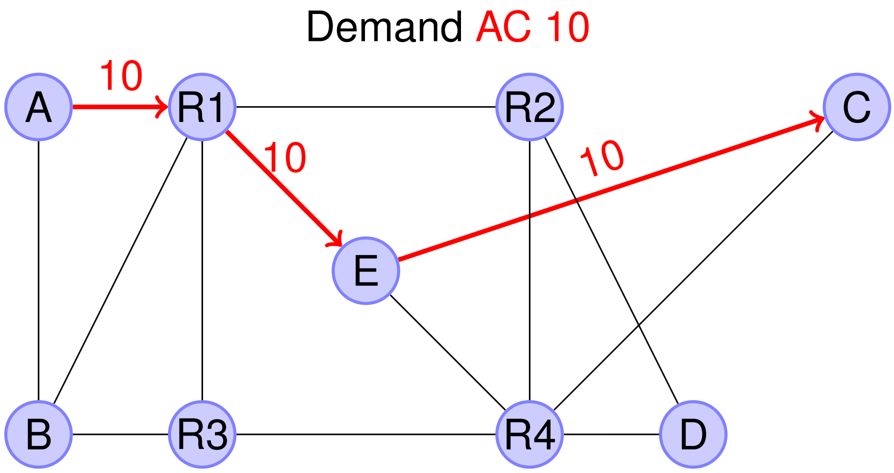


Example Traffic Matrix

Only partially filled in for example

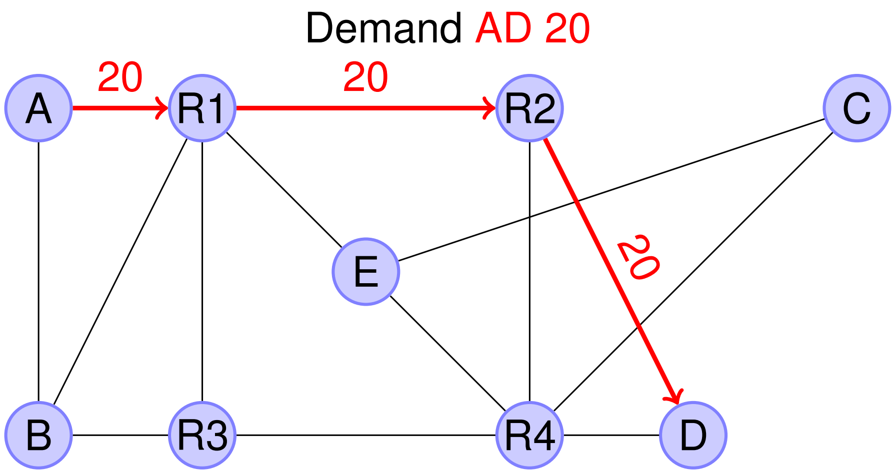
	A	B	C	D	E
A	0	0	10	20	20
B	0	0	10	20	20
C	0	0	0	0	0
D	0	0	0	0	0
E	0	0	0	0	0

Using Routing

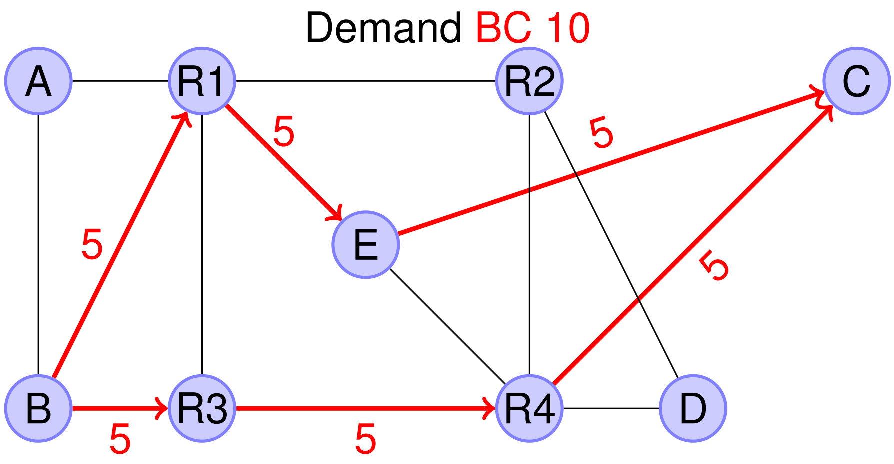


Traffic Placement
Capacity Management
Other Problems
Link Based Model
Path-Based Model
Node-Based Model
Commercial Solution
Multiple Paths

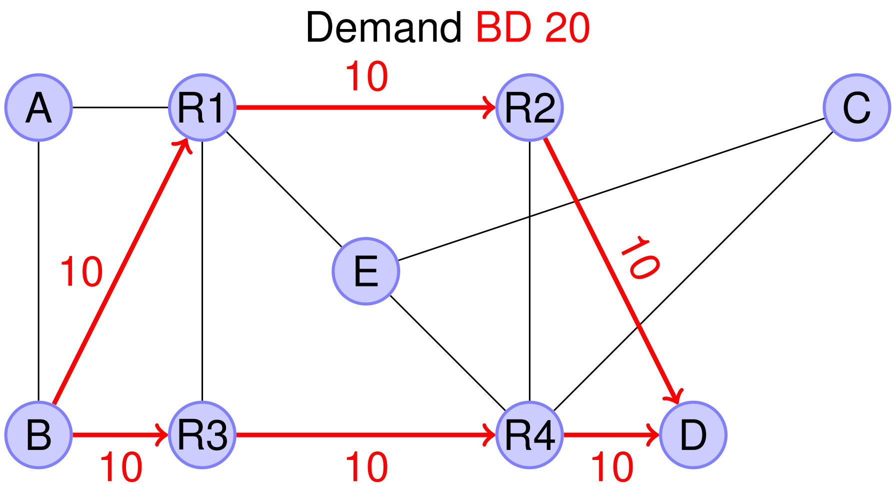
Using Routing



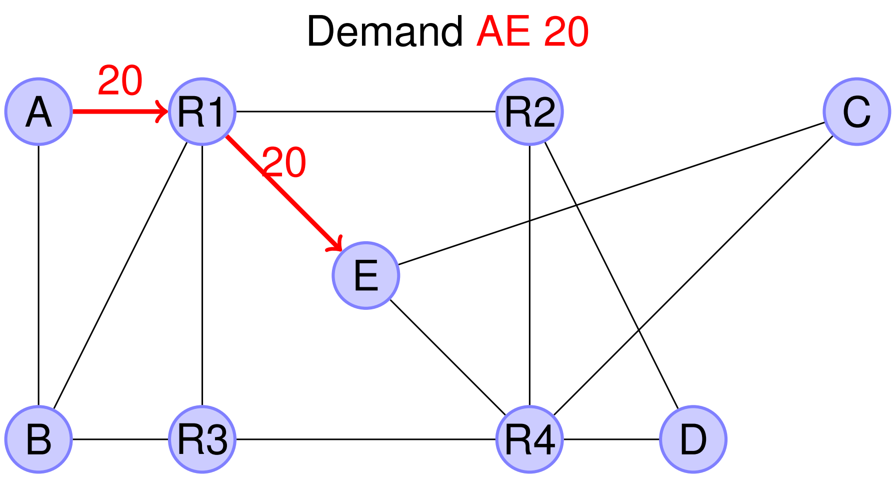
Using Routing



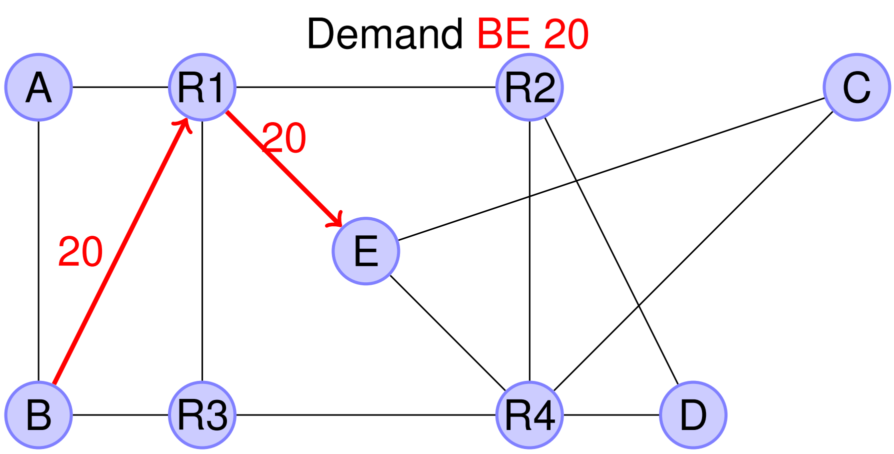
Using Routing



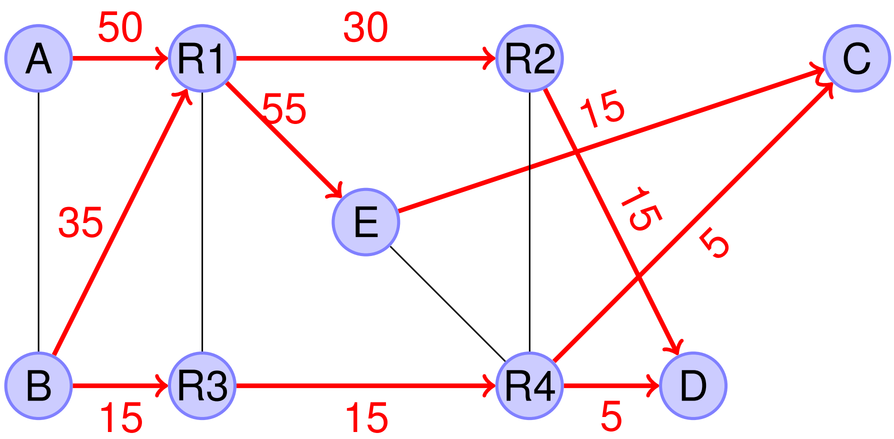
Using Routing



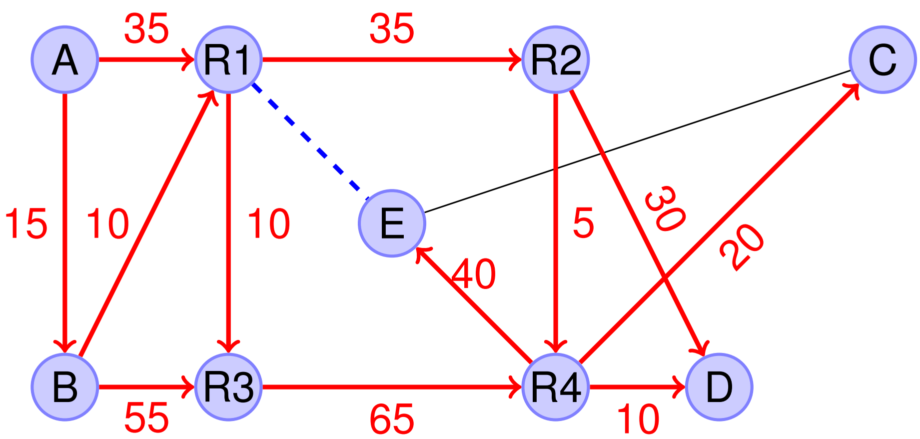
Using Routing



Resulting Network Load



Considering failure of R1-E



Can we do better?

- Choose single, explicit path for each demand
- Requires hardware support in routers (MPLS-TE)
- Baseline: CSPF, greedy heuristic

Cork Constraint Computation Centre

Why not just use Multi-Commodity Flow Problem Solution?

- Can not use arbitrary, fractional flows in hardware
- MILP does not scale too well

Cork Constraint Computation Centre

Modelling Alternatives

- Link based Model
- Path based Model
- Node based Model

Variants

- Demand Acceptance
  - Choose which demands to select fitting into available capacity
- Traffic Placement
  - All demands must be placed

Intuition

- Decide if demand d is run over link e
- Select which demands run over link e (Knapsack)
- Demand d must run from source to sink (Path)
- Sum of delay on path should be limited (QoS)

Link Based Model

$$\min_{\{X_{de}\}} \max_{e \in \mathbf{E}} \frac{1}{\mathtt{cap}(e)} \sum_{d \in \mathbf{D}} \mathtt{bw}(d) X_{de} \qquad \mathtt{or} \qquad \min_{\{X_{de}\}} \sum_{e \in \mathbf{E}, d \in \mathbf{D}} \mathtt{bw}(d) X_{de}$$

st.

$$\forall d \in \mathbf{D}, \forall n \in \mathbf{N}: \quad \sum_{e \in \mathbf{OUT}(n)} X_{de} - \sum_{e \in \mathbf{IN}(n)} X_{de} = \begin{cases} -1 & n = \mathtt{dest}(d) \\ 1 & n = \mathtt{orig}(d) \\ 0 & \text{otherwise} \end{cases}$$

$$\forall e \in \mathbf{E}: \quad \sum_{d \in \mathbf{D}} \mathtt{bw}(d) X_{de} \leq \mathtt{cap}(e)$$

$$\forall d \in \mathbf{D}: \quad \sum_{e \in \mathbf{E}} \mathtt{del}(e) X_{de} \leq \mathtt{req}(d)$$

$$X_{de} \in \{0, 1\}$$

Solution Methods

- Lagrangian Relaxation
  - Path decomposition
  - Knapsack decomposition
- Probe Backtracking

Lagrangian Relaxation - Path decomposition

[Ouaja&Richards2003]

- Dualize capacity constraints
- Starting with CSPF initial solution
- Finite domain solver for path constraints
- Added capacity constraints from st-cuts
- At each step solve shortest path problems

Lagrangian Relaxation - Knapsack decomposition

[Ouaja&Richards2005]

- Dualize path constraints
- At each step solve knapsack problems
- Reduced cost based filtering

Probe Backtracking

[Liatsos et al 2003]

- Start with (infeasible) CSPF heuristic
- Consider capacity violation
  - Resolve by forcing one demand off/on link
  - Find new path respecting path and added constraints with ILP
- Repeat until no more violations, feasible solution
- Optimality proof when exhausted search space
  - Search space often very small

Intuition

- Choose one of the possible paths for demand d
- This paths competes with paths of other demands for bandwidth
- Usually too many paths to generate a priori, but most are useless

Path-Based Model

$$\max_{\{Z_d, Y_{id}\}} \sum_{d \in \mathbf{D}} \mathtt{val}(d) Z_d$$

st.

$$\forall d \in \mathbf{D} : \sum_{1 \le i \le \mathtt{path}(d)} Y_{id} = Z_d$$

$$\forall e \in \mathbf{E} : \sum_{d \in \mathbf{D}} \mathtt{bw}(d) \sum_{1 \le i \le \mathtt{path}(d)} h^e_{id} Y_{id} \le \mathtt{cap}(e)$$

$$Z_d \in \{0, 1\}$$

$$Y_{id} \in \{0, 1\}$$

Solution Methods

- Blocking Islands
- Local Search/ FD Hybrid
- (Column Generation)

Blocking Islands

[Frei&Faltings1999]

- Feasible solution only
- CSP with variables ranging over paths for demands
- No explicit domain representation
- Possible to perform forward checking by updating blocking island structure

Local Search/FD Hybrid

[Lever2004]

- Start with (feasible) CSPF heuristic
- Add more demands one by one
  - Use repair to solve capacity violations
- Use FD model to check necessary conditions
  - Determine bottlenecks by st-cuts
  - Force paths on/off links
- Define neighborhood by rerouting demands currently over violations

Node Based Model: Intuition

- For each demand, decide for each router where to go next
  - Many routers not used
- Treat link capacity with cumulative/diffn constraints
- Pure FD model, no global cost view

Cisco ISC-TEM

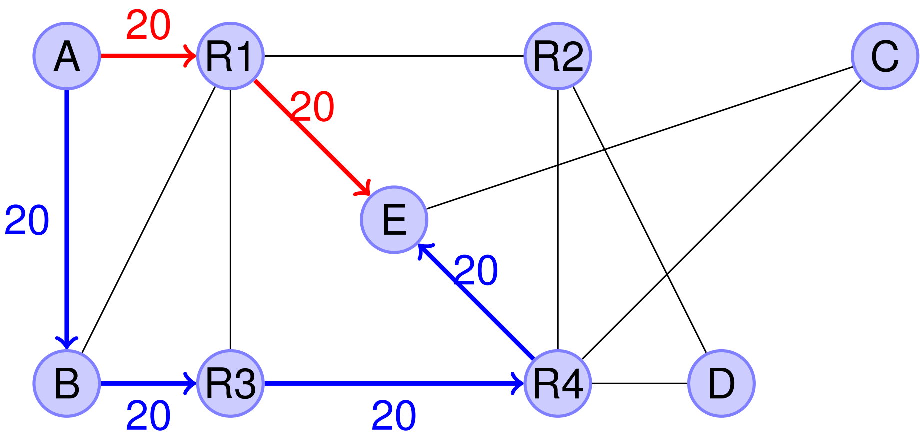
- Path placement algorithm developed for Cisco by PTL and IC-Parc (2002-2004)
- Internal competitive selection of approaches
- Strong emphasis on stability
- Written in ECLiPSe
- PTL bought by Cisco in 2004
- Part of team moved to Boston

Problem

- What happens if element on selected path fails?
- Choose second path which is link (element) disjoint
- State bandwidth constraints for each considered failure case
- Problem: Very large number of capacity constraints

Example

Primary/Secondary path for demand AE



Which bandwidth to count?

Failed Element	No Failure	A-R1	R1-E	All Others
Capacity for Path	Primary	Secondary	Secondary	Primary

Multiple Path Model

$$\max_{\{Z_d, X_{de}, W_{de}\}} \sum_{d \in \mathbf{D}} \mathtt{val}(d) Z_d$$

$$\forall d \in \mathbf{D}, \forall n \in \mathbf{N}: \sum_{e \in \mathbf{OUT}(n)} X_{de} - \sum_{e \in \mathbf{IN}(n)} X_{de} = \begin{cases} -Z_d & n = \mathtt{dest}(d) \\ Z_d & n = \mathtt{orig}(d) \\ 0 & \text{otherwise} \end{cases}$$

$$\forall e \in \mathbf{E}: \sum_{d \in \mathbf{D}} \mathtt{bw}(d) * X_{de} \le \mathtt{cap}(e)$$

$$\forall d \in \mathbf{D}, \forall n \in \mathbf{N}: \sum_{e \in \mathbf{OUT}(n)} W_{de} - \sum_{e \in \mathbf{IN}(n)} W_{de} = \begin{cases} -Z_d & n = \mathtt{dest}(d) \\ Z_d & n = \mathtt{orig}(d) \\ 0 & \text{otherwise} \end{cases}$$

$$\forall e \in \mathbf{E}, \forall e' \in \mathbf{E} \setminus e: \sum_{d \in \mathbf{D}} \mathtt{bw}(d) * (X_{de} - X_{de'} * X_{de} + X_{de'} * W_{de}) \le \mathtt{cap}(e)$$

$$\forall d \in \mathbf{D}, \forall e \in \mathbf{E}: \quad X_{de} + W_{de} \le 1$$

$$Z_d \in \{0,1\}, X_{de} \in \{0,1\}, W_{de} \in \{0,1\}$$

Solution Method

- Benders Decomposition [Xia&Simonis2005]
- Use MILP for standard demand acceptance problem
- Find two link disjoint paths for each demand
- Sub-problems consist of capacity constraints for failure cases
- Benders cuts are just no-good cuts for secondary violations

The Problem

- How to provide cost effective, high quality services running an IP network?
- Easy to build high quality network by massive over-provisioning
- Easy to build consumer grade network disregarding Quality of Service (QoS)
- Very hard to right-size a network, providing just enough capacity

The Approach

- Bandwidth on Demand
  - Create temporary bandwidth channels for high-value traffic
  - Avoid disturbing existing traffic
- Resilience Analysis
  - Find out how much capacity is required for current traffic
  - Provide enough capacity to survive element failures without service disruption

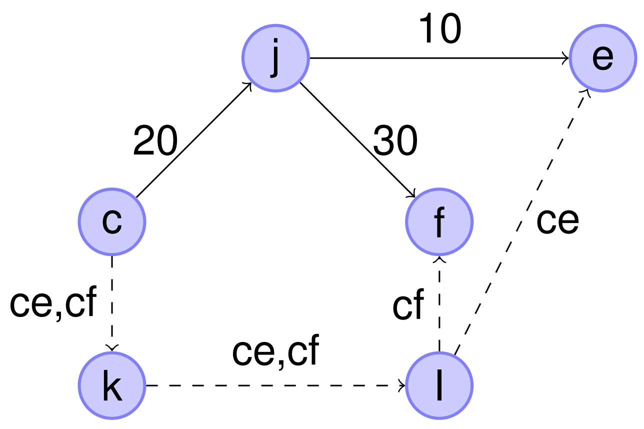
Background

- Failures of network should not affect services running on network
- Not cost effective to protect connections in hardware
- Response time is critical
    - Interruption > 50ms not acceptable for telephony
    - Reconvergence of IGP 1 sec (good setup)
    - Secondary tunnels rely on signalling of failure (too slow)
    - Live/Live connections too expensive

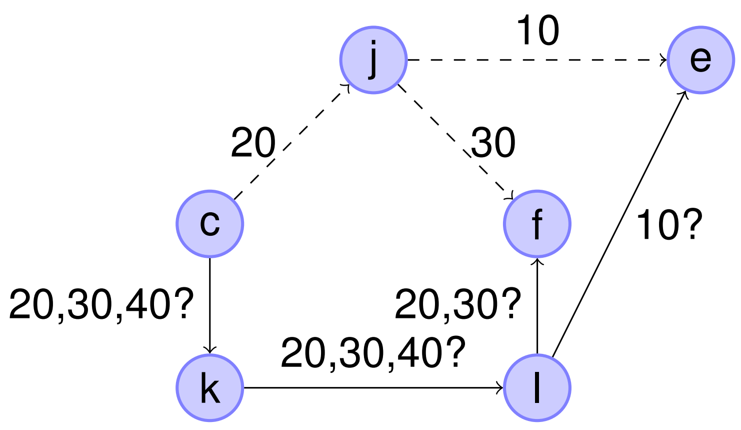
Approach

- Fast Re-route
    - If element fails, use detour around failure
    - Local repair, not global reaction
    - Pre-compute possible reactions, allows offline optimization
- Link protection rather easy
- Node protection quite difficult

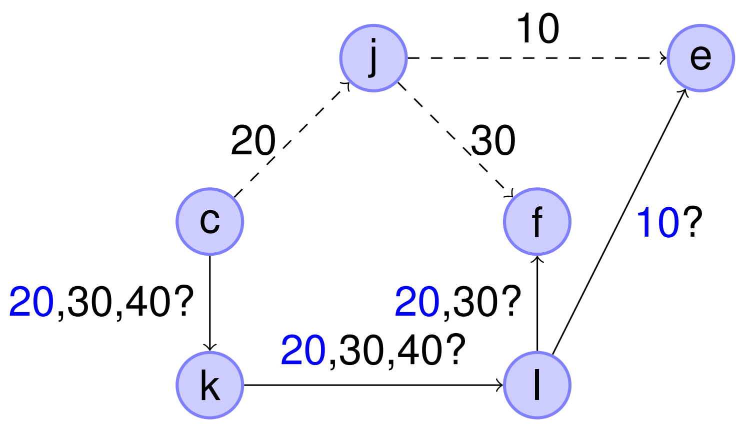
Example Problem



Node j Failure



Node j Failure (Result)



Bandwidth Protection Model

$$\min_{\{X_{fe}\}} \sum_{f \in \mathbf{F}} \sum_{e \in \mathbf{E}} X_{fe}$$

$$st. \begin{cases} \forall f \in \mathbf{F}: & \begin{cases} \forall n \in \mathbf{N} \setminus \{\mathtt{orig}(f), \mathtt{dest}(f)\}: & \sum_{e \in \mathbf{IN}(n)} X_{fe} = \sum_{e \in \mathbf{OUT}(n)} X_{fe} \\ n = \mathtt{orig}(f): & \sum_{e \in \mathbf{OUT}(n)} X_{fe} = 1 \\ n = \mathtt{dest}(f): & \sum_{e \in \mathbf{IN}(n)} X_{fe} = 1 \end{cases} \\ \forall e \in \mathbf{E}: & \mathtt{cap}(e) \geq \begin{cases} \max_{\{Q_{fe}\}} \sum_{f \in \mathbf{F}} X_{fe} Q_{fe} \\ st. \begin{cases} \forall o \in \mathtt{orig}(\mathbf{F}): & \mathtt{ocap}(o) \geq \sum_{f:\mathtt{orig}(f)=o} Q_{fe} \\ \forall d \in \mathtt{dest}(\mathbf{F}): & \mathtt{dcap}(o) \geq \sum_{f:\mathtt{dest}(f)=d} Q_{fe} \end{cases} \end{cases} \\ & X_{fe} \in \{0, 1\} \\ & \mathtt{quan}(f) \geq Q_{fe} \geq 0 \end{cases}$$

Solution Techniques

[Xia, Eremin & Wallace 2004]

- MILP
  - Use of Karusch-Kahn-Tucker condition
  - Removal of nested optimization
  - Large set of new variables
  - Not scalable
- Problem Decomposition
  - Integer Multi-Commodity Flow Problem
  - Capacity Optimization
- Improved MILP out-performs decomposition [Xia 2005]

Cisco Tunnel Builder Pro

- Algorithm/Implementation built by PTL/IC-Parc for Cisco
- Not based on published techniques above
- In period 2000-2003
- Written in ECLiPSe
- Embedded in Java GUI
- Now subsumed by ISC-TEM

Planning Ahead

- Consider demands with fixed start and end times
- Demands overlapping in time compete for bandwidth
- Demands arrive in batches, not always in temporal sequence
- Problem called Bandwidth on Demand (BoD)

Model: BoD

$$\max_{\{Z_d, X_{de}\}} \sum_{d \in \mathbf{D}} \mathtt{val}(d) Z_d$$

st.

$$\mathbf{T} = \{\mathtt{start}(d) | d \in \mathbf{D}\}$$

$$\forall d \in \mathbf{D}, \forall n \in \mathbf{N}: \sum_{e \in \mathbf{OUT}(n)} X_{de} - \sum_{e \in \mathbf{IN}(n)} X_{de} = \begin{cases} -Z_d & n = \mathtt{dest}(d) \\ Z_d & n = \mathtt{orig}(d) \\ 0 & \text{otherwise} \end{cases}$$

$$\forall t \in \mathbf{T}, \forall e \in \mathbf{E}: \sum_{\substack{d \in \mathbf{D} \\ \mathtt{start}(d) \leq t \\ t < \mathtt{end}(d)}} \mathtt{bw}(d) X_{de} \leq \mathtt{cap}(e)$$

$$Z_d \in \{0, 1\}$$

$$X_{de} \in \{0, 1\}$$

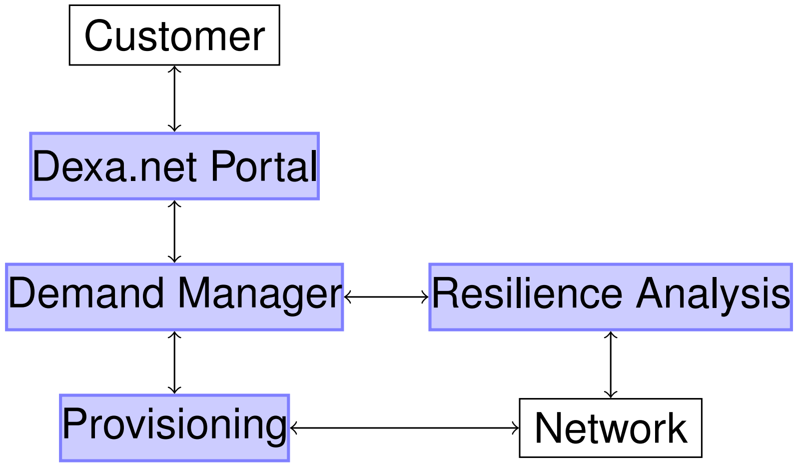
Solution Methods

- France Telecom for ATM network [Lauvergne et al 2002, Loudni et al 2003]
- Schlumberger Dexa.net (PTL, IC-Parc)

Schlumberger Dexa.net

- Small, but global MPLS TE+diffserv network
- Oil field services
- (Very) High value traffic
  - Well logging
  - Video conferencing
- Bandwidth demand known well in advance, fixed period
- Low latency, low jitter required

Architecture



Workflow

- Customer requests capacity for time slot via Web-interface
- Demand Manager determines if request can be satisfied
  - Based on free capacity predicted by Resilience Analysis
  - Taking other, accepted BoD requests into account
- Email back to customer
- At requested time, DM triggers provisioning tool to
  - Set up tunnel
  - Change admission control
- At end of period, DM pulls down tunnel

How much free capacity do we have in network?

- Easy for normal network state (OSS tools)
- Challenge: How much is required for possible failure scenarios?
- Consider single link, switch, router, PoP failures
- Classical solution
  - Get Traffic Matrix
  - Run scenarios through simulator

How to get a Traffic Matrix?

- Many algorithms assume given traffic matrix
- Traffic flow information is not collected in the routers
- Only link traffic is readily available
- Demand pattern changes over time, often quite dramatically
- Measuring traffic flows with probes is very costly

From a network consultant:

> *We have been working on extracting a TM for this network for 15 months, and we still don't have a clue if we've got it right.*

Idea

- Use the observed traffic to deduce traffic flows
- Network Tomography [Vardi1996]
  - All flows routed over a link cause the observed traffic
  - Must correct for observation errors
  - Highly dependent on accurate routing model
- Gravity Model [Medina et al 2002]
  - Ignore core of network
  - Assume that flows are proportional to product of ingress/egress size
- Results are very hard to validate/falsify

Model: Traffic Flow Analysis

$$\forall i, j \in \mathbf{N}: \quad \min_{\{F_{ij}\}} / \max_{\{F_{ij}\}} \quad F_{ij}$$

st.

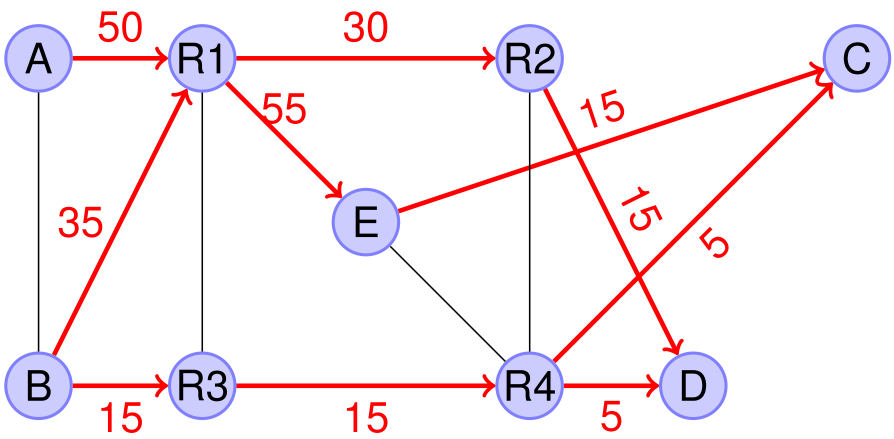
$$\forall e \in \mathbf{E}: \quad \sum_{i,j \in \mathbf{N}} r_{ij}^{e} F_{ij} = \mathtt{traf}(e)$$

$$\forall i \in \mathbf{N}: \quad \sum_{j \in \mathbf{N}} F_{ij} = \mathtt{ext}^{in}(i)$$

$$\forall j \in \mathbf{N}: \quad \sum_{i \in \mathbf{N}} F_{ij} = \mathtt{ext}^{out}(j)$$

$$F_{ij} \geq 0$$

Start with Link Traffic



Traffic Placement
Capacity Management
Other Problems
Bandwidth Protection
Bandwidth on Demand
Resilience Analysis

Setup Model to Find Flows

```
[AC,AD,BC,BD,AE,BE] :: 0.0 .. 1.0Inf,
AC + AD + AE $= 50, % A R1
0.5*BC + 0.5*BD + BE $= 35, % B R1
0.5*BC + 0.5*BD $= 15, % B R3
AD + 0.5*BD $= 30, % R1 R2
AC + 0.5*BC + AE +BE $= 55, % R1 E
AD + 0.5*BD $= 30, % R2 D
0.5*BC + 0.5*BD $= 15, % R3 R4
AC + 0.5*BC $= 15, % E C
0.5*BC $= 5, % R4 C
0.5*BD $= 10, % R4 D
```

Solve for Different Flows

```
min(AC,MinAC),max(AC,MaxAC),
min(AD,MinAD),max(AD,MaxAD),
min(BC,MinBC),max(BC,MaxBC),
min(BD,MinBD),max(BD,MaxBD),
min(AE,MinAE),max(AE,MaxAE),
min(BE,MinBE),max(BE,MaxBE),
...
```


Results of Analysis

	C	D	E
A	10	20	20
B	10	20	20

Problem solved, no?

Benchmark Problems

Network	Routers	PoPs	Lines	Lines/router
dexa	51	24	59	1.15
as1221	108	57	153	1.41
as1239	315	44	972	3.08
as1755	87	23	161	1.85
as3257	161	49	328	2.03
as3967	79	22	147	1.86
as6461	141	22	374	2.65

TFA Result for Benchmarks

Network	$\frac{Low}{Simul}$ (%)	$\frac{High}{Simul}$ (%)	Obj	Time (sec)
dexa	0	2310.65	1190	11
as1221	0.09	8398.64	11556	1318
as1239	n/a	n/a	n/a	n/a
as1755	0.15	6255.31	7482	699
as3257	0.04	12260.03	25760	12389
as3967	0.1	5387.10	6162	500
as6461	0.28	8688.39	19740	8676

Reduce Problem Size

- Pop Level Analysis
- Only consider flows between PoPs, not routers
- Local area connections typically not bottlenecks
- Modelling routing can be tricky

PoP Level Results

Network	$\frac{Low}{Simul}$ (%)	$\frac{High}{Simul}$ (%)	Obj	Time (sec)
dexa	0	1068.37	557	5
as1221	0.24	2964.93	3205	424
as1239	0.63	1401.72	1931	101359
as1755	0.66	1263.28	526	103
as3257	0.30	2028.73	2378	2052
as3967	0.1	1209.37	483	90
as6461	1.47	951.41	481	768

Increase Accuracy

- LSP Counters
  - In MPLS networks only, provide improved resolution
  - Implementation buggy, not all counters can be used
- Netflow
  - Collect end-to-end flow information in router
  - Impact on router (memory)
  - Impact on network (data aggregation)

Cork Constraint Computation Centre

TFA with LSP Counters

Network	$\frac{Low}{Simul}$ (%)	$\frac{High}{Simul}$ (%)	Obj	Time (sec)
dexa	30.35	249.71	1190	7
as1221	9.94	685.37	11556	885
as1239	10.74	1151.03	98910	72461
as1755	25.29	269.30	7482	397
as3257	23.77	425.67	25760	5121
as3967	24.47	300.17	6162	275
as6461	19.43	477.44	19740	2683

Cork Constraint Computation Centre

PoP TFA with LSP Counters

Network	$\frac{Low}{Simul}$ (%)	$\frac{High}{Simul}$ (%)	Obj	Time (sec)
dexa	60.62	145.85	557	3
as1221	28.49	499.16	3205	271
as1239	33.36	211.84	1931	2569
as1755	50.33	169.37	526	46
as3257	36.82	249.16	2378	640
as3967	40.72	182.97	483	36
as6461	34.05	210.93	481	136

What now?

- Choose some particular solution?
- Which one? How to validate assumptions?
- Massively under-constrained problem
  - $|N|^2$ variables
  - $|E| + 2|N|$ constraints
  - $2|N|^2$ queries
- Ill-conditioned even after error correction
- Aggregation helps
  - We are usually not interested in individual flows
  - We want to use the TM to investigate something else

Resilience Analysis

- How much capacity is needed to survive all reasonable failures?
- Use normal state as starting point
- Consider routing in each failure case
- Aggregate flows in rerouted network
- Calculate bounds on traffic in failure case

Model: Resilience Analysis

$$\forall e \in \mathbf{E}: \quad \min_{\{F_{ij}\}} / \max_{\{F_{ij}\}} \sum_{i,j \in \mathbf{N}} \overline{r}_{ij}^{e} F_{ij}$$

st.

$$\forall e \in \mathbf{E}: \quad \sum_{i,j \in \mathbf{N}} r_{ij}^{e} F_{ij} = \mathtt{traf}(e)$$

$$\forall i \in \mathbf{N}: \quad \sum_{j \in \mathbf{N}} F_{ij} = \mathtt{ext}^{in}(i)$$

$$\forall j \in \mathbf{N}: \quad \sum_{i \in \mathbf{N}} F_{ij} = \mathtt{ext}^{out}(j)$$

$$F_{ij} \geq 0$$

Resilience Analysis

Network	$\frac{Low}{Simul}$ (%)	$\frac{High}{Simul}$ (%)	Obj	Time (sec)	Cases
dexa	68.91	108.25	3503	57	59
as1221	85.75	102.60	14191	2869	153
as1239	*92.53*	*102.64*	*4499*	*44205*	*10*
as1755	92.82	105.39	8409	1815	161
as3257	93.69	103.15	31093	39934	328
as3967	91.60	108.79	9090	1635	141
as6461	96.51	103.44	24808	20840	374

Results over 100 runs

Network	lower bound/simul		upper bound/ simul	
	average	stdev	average	stdev
dexa	91.50	0.14	108.28	0.16
as1755	88.65	0.11	106.08	0.056
as3967	94.08	0.073	106.88	0.091
as1221	87.34	0.10	102.05	0.025

Results with LSP counters

Network	$\frac{Low}{Simul}$ (%)	$\frac{High}{Simul}$ (%)	Obj	Time	Cases
dexa	97.76	101.33	3503	36	59
as1221	98.15	100.69	14191	1840	153
as1239	*99.37*	*100.38*	*4499*	*3974*	*10*
as1755	99.28	100.66	8409	964	161
as3257	99.41	100.44	31093	13381	328
as3967	98.88	101.00	9090	819	147
as6461	99.44	100.52	24808	8006	374

Results over 100 runs (with LSP Counters)

Network	lower bound/simul		upper bound/ simul	
	average	stdev	average	stdev
dexa	99.60	0.029	100.33	0.025
as1755	99.31	0.016	100.63	0.015
as3967	99.41	0.014	100.61	0.014
as1221	98.10	0.025	100.57	0.010

Perspectives

- High polynomial complexity
- Possible to reduce number of queries
  - Small differences between failure cases
  - Many queries are identical or dominated
- Possible to reduce size of problem dramatically
- Integrate multiple measurements in one model
- Which other problems can we solve without explicit TM?

Problem

- Which links should be used to build network structure?
- Link speed is related to cost
- Model simple generalization of path finding
- Assumptions about routing in target network?

Model

$$\min_{\{X_{de}, W_{ie}\}} \sum_{e \in \mathbf{E}} \sum_{1 \le i \le \mathtt{alt}(e)} \mathtt{cost}(i,e) W_{ie}$$

$$\forall d \in \mathbf{D}, \forall n \in \mathbf{N}: \quad \sum_{e \in \mathbf{OUT}(n)} X_{de} - \sum_{e \in \mathbf{IN}(n)} X_{de} = \begin{cases} -1 & n = \mathtt{dest}(d) \\ 1 & n = \mathtt{orig}(d) \\ 0 & \text{otherwise} \end{cases}$$

$$\forall e \in \mathbf{E}: \quad \sum_{d \in \mathbf{D}} \mathtt{bw}(d) X_{de} \le \sum_{1 \le i \le \mathtt{alt}(e)} \mathtt{cap}(i,e) W_{ie}$$

$$\forall e \in \mathbf{E}: \quad \sum_{1 \le i \le \mathtt{alt}(e)} W_{ie} = 1$$

$$W_{ie} \in \{0,1\}$$

$$X_{de} \in \{0,1\}$$

Issues

- Real-life problem not easily modelled
- Possible choices/costs not easily obtained (outside US)
- Choices often are inter-related
- Package deals by providers
- Some regions don’t allow any flexibility at all

Problem

- How to set weights in IGP to avoid bottlenecks?
- Easy to beat default values
- Single/equal cost paths required/allowed/forbidden?

Model

$$\min_{\{Y_{id}, W_e\}} \max_{e \in \mathbf{E}} \frac{1}{\mathtt{cap}(e)} \sum_{d \in \mathbf{D}} \mathtt{bw}(d) \sum_{1 \leq i \leq \mathtt{path}(d)} h^e_{id} Y_{id}$$

st.

$$\forall d \in \mathbf{D}: \quad \sum_{1 \leq i \leq \mathtt{path}(d)} Y_{id} = 1$$

$$\forall d \in \mathbf{D}, 1 \leq i \leq \mathtt{path}(d): \quad P_{id} = \sum_{e \in \mathbf{E}} h^e_{id} W_e$$

$$\forall d \in \mathbf{D}, 1 \leq i, j \leq \mathtt{path}(d): \quad P_{id} = P_{jd} \implies Y_{id} = Y_{jd} = 0$$

$$\forall d \in \mathbf{D}, 1 \leq i, j \leq \mathtt{path}(d): \quad P_{id} < P_{jd} \implies Y_{jd} = 0$$

$$Y_{id} \in \{0, 1\}$$

$$\text{integer} \quad W_e \geq 1$$

$$P_{id} \geq 0$$

Solution Methods

- Methods tested at IC-Parc
  - Branch and price
  - Tabu search
  - Set constraints
- Very hard to compete with (guided) local search

Further Reading

H. Simonis. Constraint Applications in Networks. Chaper 25 in F. Rossi, P van Beek and T. Walsh: Handbook of Constraint Programming. Elsevier, 2006.

Summary

- Network problems can be solved competitively by constraint techniques.
- Hybrid methods required, simple FD models usually don't work.
- Constraint based tools commercial reality.
- Open Problems
  - How to make this easier to develop?
  - How to make this more stable to solve?

Chapter 16: More Global Constraints (Car Sequencing)

Helmut Simonis

Cork Constraint Computation Centre
Computer Science Department
University College Cork
Ireland

ECLiPSe ELearning Overview

Licence

Outline

Cork Constraint Computation Centre

What we want to introduce

- Car Sequencing Problem
- `gcc` Global cardinality constraint
- `sequence` constraint
- Search based auxiliary variables

Cork Constraint Computation Centre

Problem Definition

Car Sequencing

We have to schedule a number of cars for production on an assembly line. Each car is of a certain type, and we know how many cars of each type we have to produce. Car types differ in the options they require, i.e. sun-roof, air conditioning. For each option, we have capacity limits on the assembly line, expressed as k cars out of n consecutive cars on the line may have some option. Find an assignment which produces the correct number of cars of each type, while satisfying the capacity constraints.

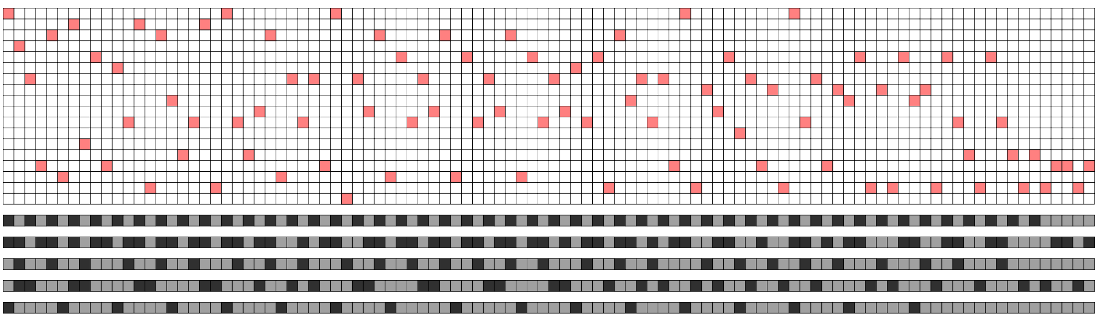
Example (DSV88)

- 100 cars
- 18 types
- 5 options
  - Option 1: 1 out of 2
  - Option 2: 2 out of 3
  - Option 3: 1 out of 3
  - Option 4: 2 out of 5
  - Option 5: 1 out of 5

Car Types

Type	Cars Required	Option 1	2	3	4	5
1	5	1	1	0	0	1
2	3	1	1	0	1	0
3	7	1	1	1	0	0
4	1	0	1	1	1	0
5	10	1	1	0	0	0
6	2	1	0	0	0	1
7	11	1	0	0	1	0
8	5	1	0	1	0	0
9	4	0	1	0	0	1
10	6	0	1	0	1	0
11	12	0	1	1	0	0
12	1	0	0	1	0	1
13	1	0	0	1	1	0
14	5	1	0	0	0	0
15	9	0	1	0	0	0
16	5	0	0	0	0	1
17	12	0	0	0	1	0
18	1	0	0	1	0	0

Solution



Modelling Alternatives

- Assign start time (sequence number) to each car
  - 100 variables, each with 100 values
  - Handling of car types implicit
  - Symmetry breaking for cars of same type (inequalities)?
  - Capacity constraints?
- Assign car type to each slot on assembly line
  - 100 variables, 18 values
  - How to control number of cars of each type?
  - How to express capacity constraints?

Cork Constraint Computation Centre

Model

- 100 Variables ranging over car types
- `gcc` constraint to control number of items with same type
- 5×100 0/1 variables indicating use of option for each slot
- `element` constraints to map car types to options used
- `sequence` constraints to enforce limits on each option

Cork Constraint Computation Centre

`gcc(Pattern, Variables)`

- `gcc` *Global Cardinality Constraint*
- `Pattern` is list of terms `gcc(Low, High, Value)`
- The overall number of variables taking value `Value` is between `Low` and `High`
- Generalization of `alldifferent`
- Domain consistent version in ECLiPSe

`gcc` Example

```
X1 :: [2,4], X2 :: [1,3,4], X3 :: [1,2,3,4],
X4 :: [3,4,5], X5 :: [3,4,5],
gcc([gcc(1,1,1),gcc(2,3,2),gcc(1,3,3),
     gcc(0,4,4),gcc(1,3,5)],
    [X1,X2,X3,X4,X5]),
```

X1 = ?, X2 = ?, X3 = ?, X4 = ?, X5 = ?

gcc Reasoning

X1 :: [2,4], X2 :: [1,3,4], X3 :: [1,2,3,4],
X4 :: [3,4,5], X5 :: [3,4,5],
gcc([gcc(1,1,1),gcc(2,3,2),gcc(1,3,3),
gcc(0,4,4),gcc(1,3,5)],
[X1,X2,X3,X4,X5]),

X1 = ?2, X2 = ?, X3 = ?2, X4 = ?, X5 = ?

gcc Next Step

X1 :: [2,~~4~~], X2 :: [1,3,4], X3 :: [~~1~~,2,~~3~~,~~4~~],
X4 :: [3,4,5], X5 :: [3,4,5],
gcc([gcc(1,1,1),gcc(2,3,2),gcc(1,3,3),
gcc(0,4,4),gcc(1,3,5)],
[X1,X2,X3,X4,X5]),

X1 = 2, X2 = ?1, X3 = 2, X4 = ?, X5 = ?

gcc Continued

X1 :: [2,~~4~~], X2 :: [1,~~3~~,~~4~~], X3 :: [~~1~~,2,~~3~~,~~4~~],
X4 :: [3,4~~4~~,5], X5 :: [3,4~~4~~,5],
gcc([gcc(1,1,1),gcc(2,3,2),gcc(1,3,3),
gcc(0,4,4),gcc(1,3,5)],
[X1,X2,X3,X4,X5]),

X1 = 2, X2 = 1, X3 = 2, X4 = ?, X5 = ?

Cork Constraint Computation Centre

gcc Made Domain Consistent

X1 :: [2,~~4~~], X2 :: [1,~~3~~,~~4~~], X3 :: [~~1~~,2,~~3~~,~~4~~],
X4 :: [3,~~4~~,5], X5 :: [3,~~4~~,5],
gcc([gcc(1,1,1),gcc(2,3,2),gcc(1,3,3),
gcc(0,4,4),gcc(1,3,5)],
[X1,X2,X3,X4,X5]),

X1 = 2, X2 = 1, X3 = 2, X4 $\in \{3, 5\}$, X5 $\in \{3, 5\}$

Cork Constraint Computation Centre

How does the constraint solver do that?

Explained in optional material at end

Domain Consistent `gcc`

`element(X,List,Y)`

- `List` is a list of integers
- The X^{th} element of `List` is Y
- The index starts from 1
- Typical Uses:
  - Projection
  - Cost

Element Examples

`Prime` is 1 iff $X \in 1..10$ is a prime number

```
X :: 1..10,
element(X,[1,1,1,0,1,0,1,0,0,0],Prime),
```

`Cost` is the cost corresponding to the assignment of Y

```
Y :: 1..10,
element(Y,[5,3,34,0,3,1,12,12,1,3],Cost)
```


`sequence_total(Min,Max,Low,High,K,Vars)`

- Variables `Vars` have 0/1 domain
- Between `Min` and `Max` variables have value 1
- For every sub-sequence of length K, between `Low` and `High` variables have value 1

`sequence_total` Example

```
[X1,X2,X3,X4,X5,X6,X7,X8,X9,X10] :: 0..1,
sequence_total(2,3,1,2,3,
               [X1,X2,X3,X4,X5,X6,X7,X8,X9,X10]),
```

X1 = 0, X4 = 0, X7 = 0, X10 = 0

Example, cont'd

$$x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}$$

$$x_1, \overbrace{x_2, x_3, x_4}^{1..2}, \overbrace{x_5, x_6, x_7}^{1..2}, \overbrace{x_8, x_9, x_{10}}^{1..2}$$

$$x_1, \overbrace{\overbrace{x_2, x_3, x_4}^{1..2}, \overbrace{x_5, x_6, x_7}^{1..2}, \overbrace{x_8, x_9, x_{10}}^{1..2}}^{3..6}$$

$$\underbrace{x_1, \overbrace{\overbrace{x_2, x_3, x_4}^{1..2}, \overbrace{x_5, x_6, x_7}^{1..2}, \overbrace{x_8, x_9, x_{10}}^{1..2}}^{3..6}}_{2..3}$$

$$\underbrace{x_1 = 0, \overbrace{\overbrace{x_2, x_3, x_4}^{1..2}, \overbrace{x_5, x_6, x_7}^{1..2}, \overbrace{x_8, x_9, x_{10}}^{1..2}}^{3..6}}_{2..3}$$

Mathematical Equivalent

$$\text{Vars} = [x_1, x_2, \ldots x_N]$$

$$\text{Min} \leq \sum_{1 \leq i \leq N} x_i \leq \text{Max}$$

$$1 \leq s \leq N - k + 1: \quad \text{Low} \leq \sum_{s \leq j \leq s+k-1} x_j \leq \text{High}$$

Mathematical Equivalent

- Pruning very different when using finite domain inequalities
- Currently no domain consistent implementation of `sequence_total`
- Weaker version `sequence` (no global counters) domain consistent
- Currently using decomposition:
  - `sequence_total` = `sequence` + `gcc` + more

Main Program

```
:-module(car).
:-export(top/0).
:-lib(ic).
:-lib(ic_global_gac).

top:-
    problem(Problem),
    model(Problem,L),
    writeln(L).
```


Structure Definitions

```
:-local struct(problem(cars,
                      models,
                      required,
                      using_options,
                      value_order)).
:-local struct(option(k,
                     n,
                     index_set,
                     total_use)).
```

Model (Part 1)

```
model(problem{cars:NrCars,
              models:NrModels,
              required:Required,
              using_options:List,
              value_order:Ordered},L):-
    length(L,NrCars),
    L :: 1..NrModels,
    (foreach(Cnt,Required),
     count(J,1,_),
     foreach(gcc(Cnt,Cnt,J),Card) do
        true
    ),
    gcc(Card,L),
    ...
```


Model (Part 2)

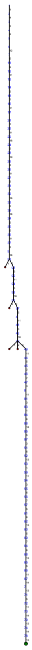
```
    ...
    (foreach(option{k:K,
                    n:N,
                    index_set:IndexSet,
                    total_use:Total},List),
     param(L,NrCars) do
        (foreach(X,L),
         foreach(B,Binary),
         param(IndexSet) do
            element(X,IndexSet,B)
        ),
        sequence_total(Total,Total,0,K,N,Binary)
    ),
    search(L,0,input_order,ordered(Ordered),
```

Data

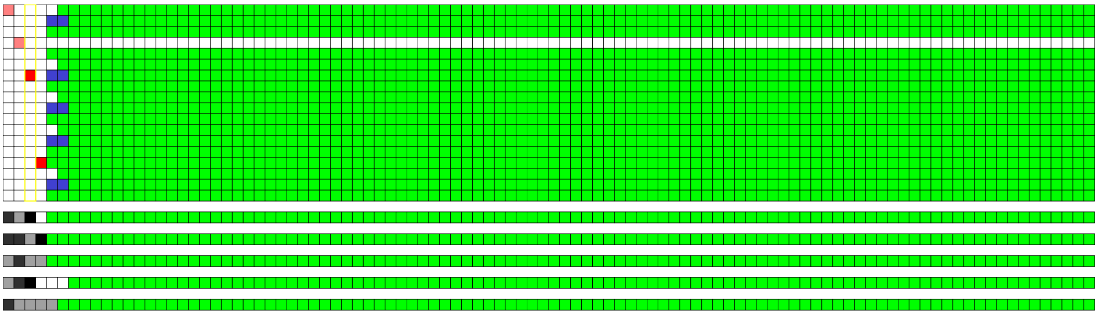
```
problem(100,18,
  [5,3,7,1,10,2,11,5,4,6,12,1,1,5,9,5,12,1],
  [option(1,2,[1,2,3,5,6,7,8,14],
     [1,1,1,0,1,1,1,1,0,0,0,0,0,1,0,0,0,0],48),
   option(2,3,[1,2,3,4,5,9,10,11,15],
     [1,1,1,1,1,0,0,0,1,1,1,0,0,0,1,0,0,0],57),
   option(1,3,[3,4,8,11,12,13,18],
     [0,0,1,1,0,0,0,1,0,0,1,1,1,0,0,0,0,1],28),
   option(2,5,[2,4,7,10,13,17],
     [0,1,0,1,0,0,1,0,0,1,0,0,1,0,0,0,1,0],34),
   option(1,5,[1,6,9,12,16],
     [1,0,0,0,0,1,0,0,1,0,0,1,0,0,0,1,0,0],17)],
  [1,3,2,4,6,8,7,12,13,5,9,11,10,14,16,18,17,15]).
```


Data Generation

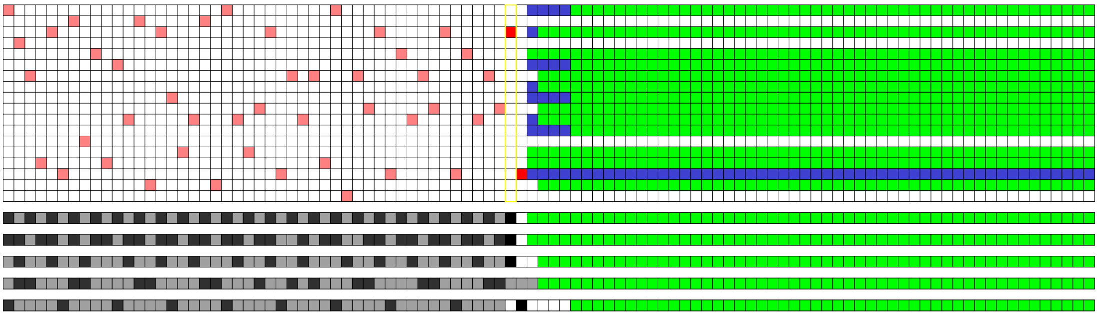
- Data not really stored as facts
- Generated from text data files in different format
- Benchmark set from CSPLIB (`http://www.csplib.org`)



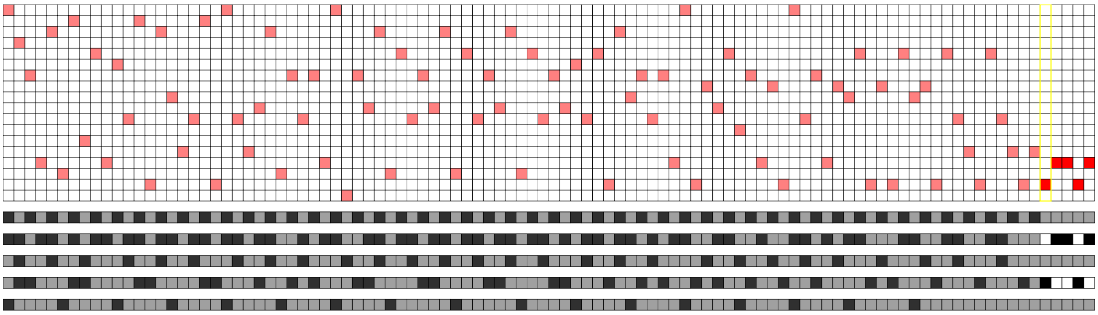
Assignment Step 4



Assignment Step 40



Assignment Step 83



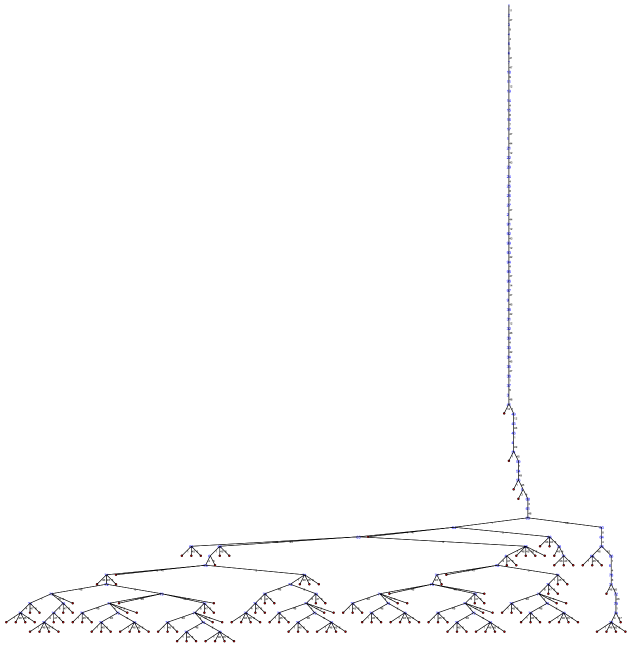
Another Example (PR97)

- 100 cars
- 22 types
- 5 options
  - Option 1: 1 out of 2
  - Option 2: 2 out of 3
  - Option 3: 1 out of 3
  - Option 4: 2 out of 5
  - Option 5: 1 out of 5

Second Example: Car Types

Type	Cars Required	Option 1	2	3	4	5
1	6	1	0	0	1	0
2	10	1	1	1	0	0
3	2	1	1	0	0	1
4	2	0	1	1	0	0
5	8	0	0	0	1	0
6	15	0	1	0	0	0
7	1	0	1	1	1	0
8	5	0	0	1	1	0
9	2	1	0	1	1	0
10	3	0	0	1	0	0
11	2	1	0	1	0	0
12	1	1	1	1	0	1
13	8	0	1	0	1	0
14	3	1	0	0	1	1
15	10	1	0	0	0	0
16	4	0	1	0	0	1
17	4	0	0	0	0	1
18	2	1	0	0	0	1
19	4	1	1	0	0	0
20	6	1	1	0	1	0
21	1	1	0	1	0	1
22	1	1	1	1	1	1

Search (Stopped After 1000 Nodes)



Observation

- This does not look good
- Typical `thrashing` behaviour
- We made a wrong choice at some point
- ... but did not detect it
- Many additional choices are made before failure is detected
- We have to explore the complete tree under the wrong choice
- This is far too expensive

Change of Search Strategy

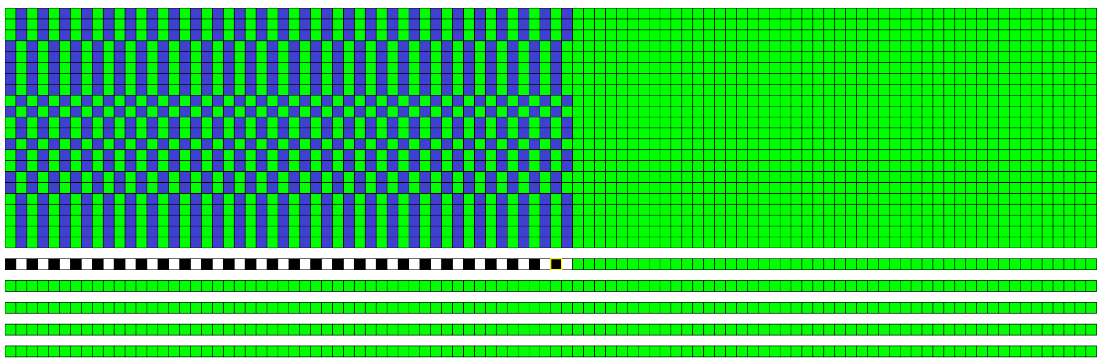
- Do not label car slot variables
- Decide instead if slot should use an option or not
- This restricts the car models which can be placed in this slot
- Start with the most restricted option
- When all options are assigned, the car type is fixed
- Potential problem: We now have 500 instead of 100 decision variables
- Naive searchspace $2^{500} = 3.2e150$ instead of $22^{100} = 1.7e134$

Second Modification

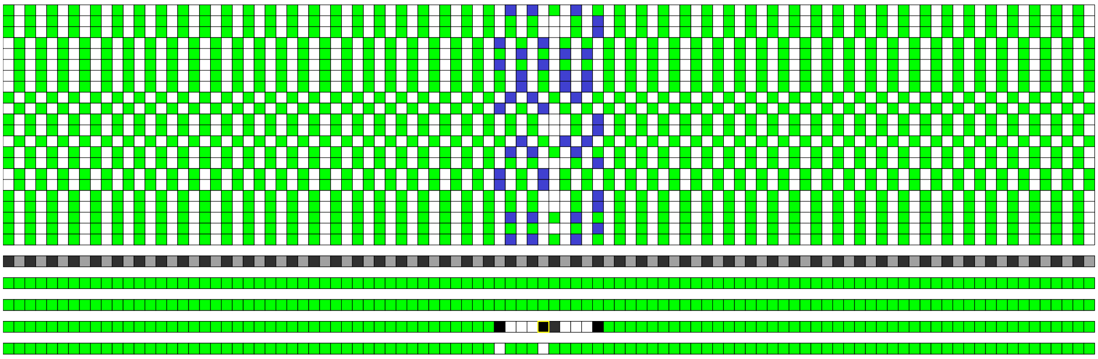
- Instead of assigning values left to right
- Start assigning in middle of board
- And alternate around middle until you reach edges
- Idea: Slots at edges are less constrained, i.e. easier to assign
- Save those slots until the end
- We already encountered this idea for the N-Queens problem

Modified Search

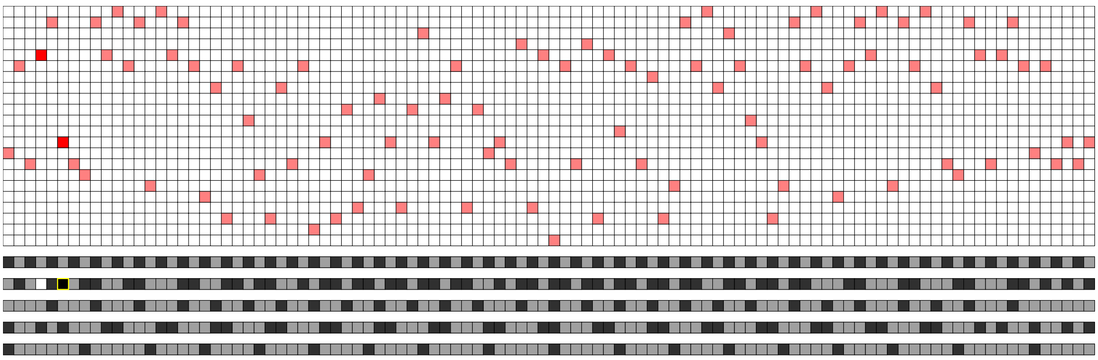
Assignment Step 2



Assignment Step 28



Assignment Step 119



Observations

- Important to start in middle
- Making hard choices first
- Concentrate on difficult to satisfy sub-problem
- Number of choices is much smaller than number of variables
- Some assignments lead to a lot of propagation

Conclusions

- Introduced two new global constraints, `gcc` and `sequence`
- Used `element` for projection
- Search on auxiliary variables can work well
- Raw search space measures are unreliable
- Modelling idea
  - Decide what to make in a given time slot
  - ... and not when to schedule some given activity

Making gcc Domain Consistent

```
X1 :: [2,4], X2 :: [1,3,4], X3 :: [1,2,3,4],
X4 :: [3,4,5], X5 :: [3,4,5],
gcc([gcc(1,1,1),gcc(2,3,2),gcc(1,3,3),
     gcc(0,4,4),gcc(1,3,5)],
    [X1,X2,X3,X4,X5]),
```

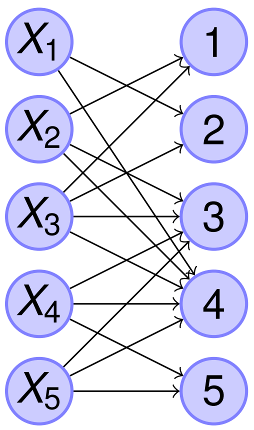
Cork Constraint Computation Centre

Method: Max Flow Model

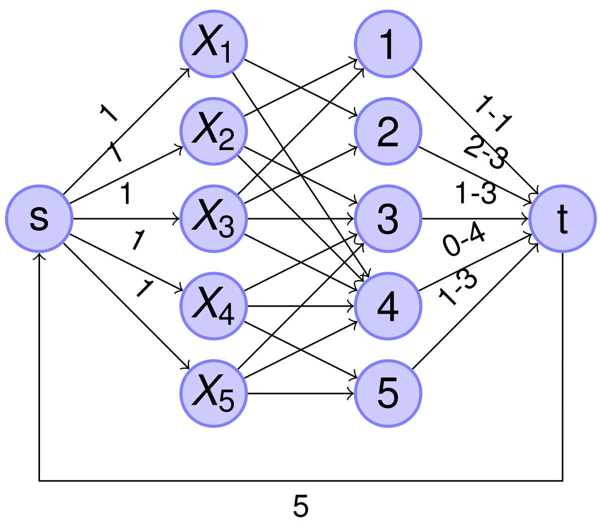
- Express constraint as max-flow problem
- Any flow solution corresponds to a valid assignment
- Only work with one flow solution
- Obtain all others by considering
  - *residual graph* and
  - *strongly connected components*
- Classical method, faster methods exist
- Use of max flow based propagators for many constraints

Cork Constraint Computation Centre

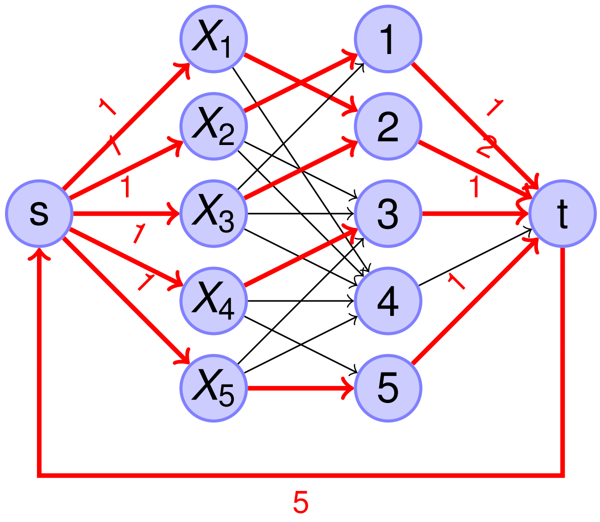
Start with Value Graph



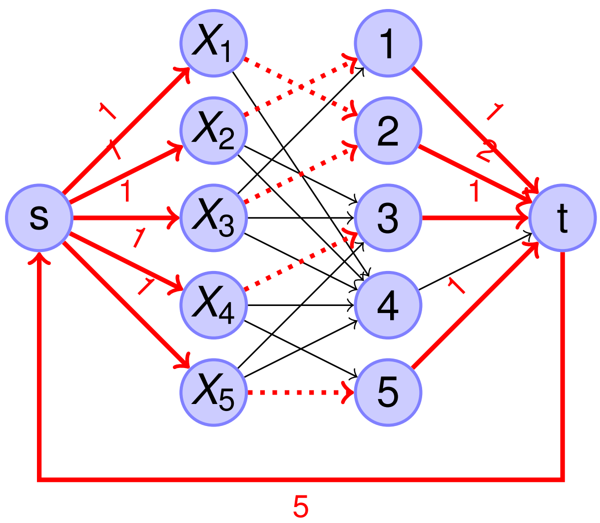
Convert to Flow Problem



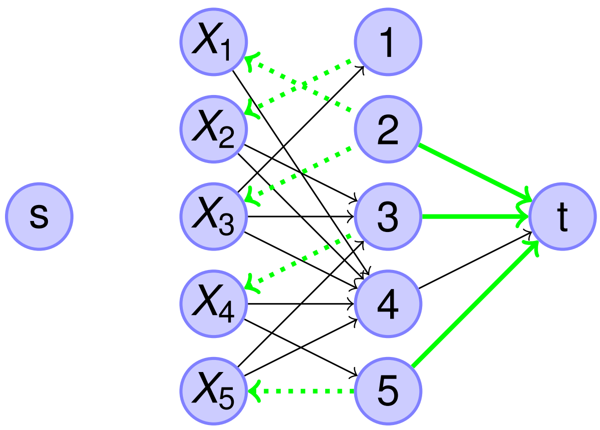
Find Maximal Flow



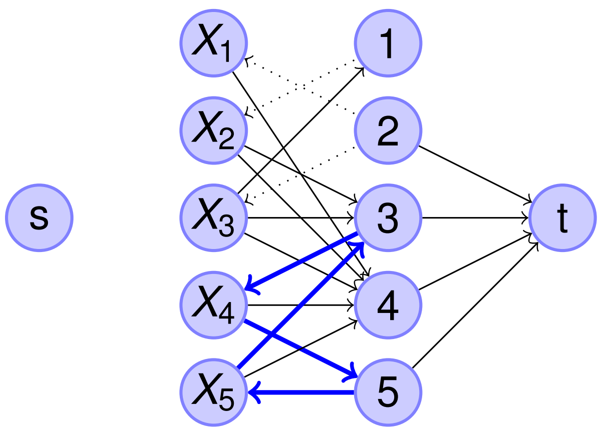
Mark Value Edges in Flow



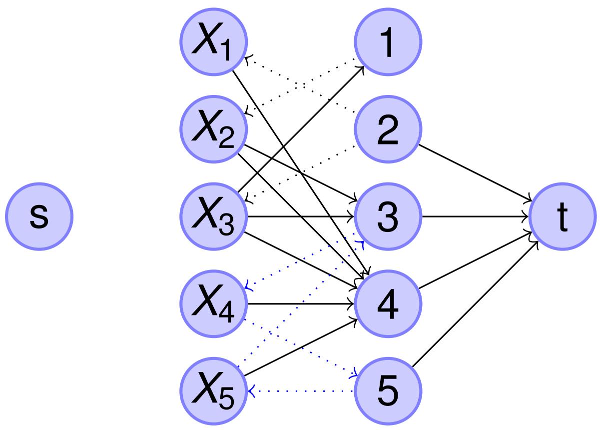
Residual Graph



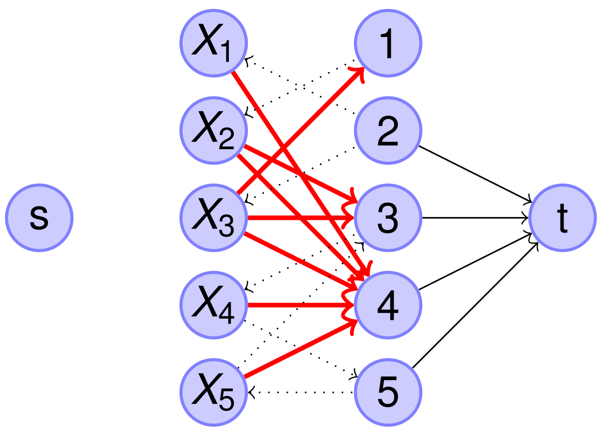
Find Strongly Connected Components



Mark Edges



Remove Unmarked Edges



Constraint is Domain Consistent