Constraint Programming

With .NET NSolver Engine

Dr. Andy Chun, Hon Wai
Department of Computer Science
Y6305 x7194
andy.chun@cityu.edu.hk

Agenda

- Concepts Behind CP
- Case Studies
- Technical Details

What is Constraint Programming?

- The ability to write programs in terms of variables and constraints
  - X, Y are unknowns
  - Y is 2 times X (X is half of Y)

- And… an algorithm that automatically solves these problems in a very efficient manner

What is Constraint Programming?

- "Constraint programming represents one of the closest approaches computer science has yet made to the Holy grail of programming: the user states the problem, the computer solves it"

EC Freuder
April 1997

Why is CP Important?

- CP can be used to solve a wide variety of very difficult real world problems:
  - Scheduling problems
  - Allocation problems
  - Timetabling problems
  - Optimization problems
- CP can solve difficult problems because of highly efficient built-in search algorithms that make use of constraint propagation

What is CSP?

- Typically, problems solved by CP are represented as Constraint Satisfaction Problems (CSP)

- Short/simple explanation of CSP:
  - Problems involving the assignment of values to variables subject to a set of constraints

A Formal Definition

- A CSP consists of
  - a finite set of n variables (or tasks) V1, ... , Vn,
  - a set of domains D1, ... , Dn, and
  - a set of constraint relations C1, ... , Cm
- Each Di defines a finite set of values (or labels or solutions) that variable Vi may be assigned

A Formal Definition

- A constraint Cj specifies the consistent or inconsistent choices among variables and is defined as a subset of the Cartesian product:
  - $Cj \subseteq D1 \times D2 \times \ldots \times Dn$
- The goal of the CSP is to find one tuple from D1 x ... x Dn such that n assignments of values to variables satisfy all constraints simultaneously

What is Missing in .NET?

- For example, let's try to solve this IQ quiz:
  - We are in a farm. In the field we see 20 heads and 56 legs. We know there are only rabbits and pheasants in the field. How many rabbits and pheasants are there?

C# Solution:

```
using System;
public class Rabbit {
  public int rabbit = 0;
  public int pheasant = 0;
  public void Run() {
    for (rabbit = 0; rabbit<=20; rabbit++)
      for (pheasant = 0; pheasant<=20; pheasant++)
        if (rabbit+pheasant==20
            && rabbit*4+pheasant*2==56)
          Console.WriteLine(
            "Rabbit["+rabbit+"] Pheasant["+pheasant+"]");
  }
  public static void Main() {(new Rabbit()).Run();}
}
```

C#

Problems

- No notion of variables/unknowns
- No notion of what values are legal or not legal for each variable (the domain)
- No notion of relationship/constraint between variables
- Everything is hardcoded and not scalable to more complex problems
- Brute force exhaustive search

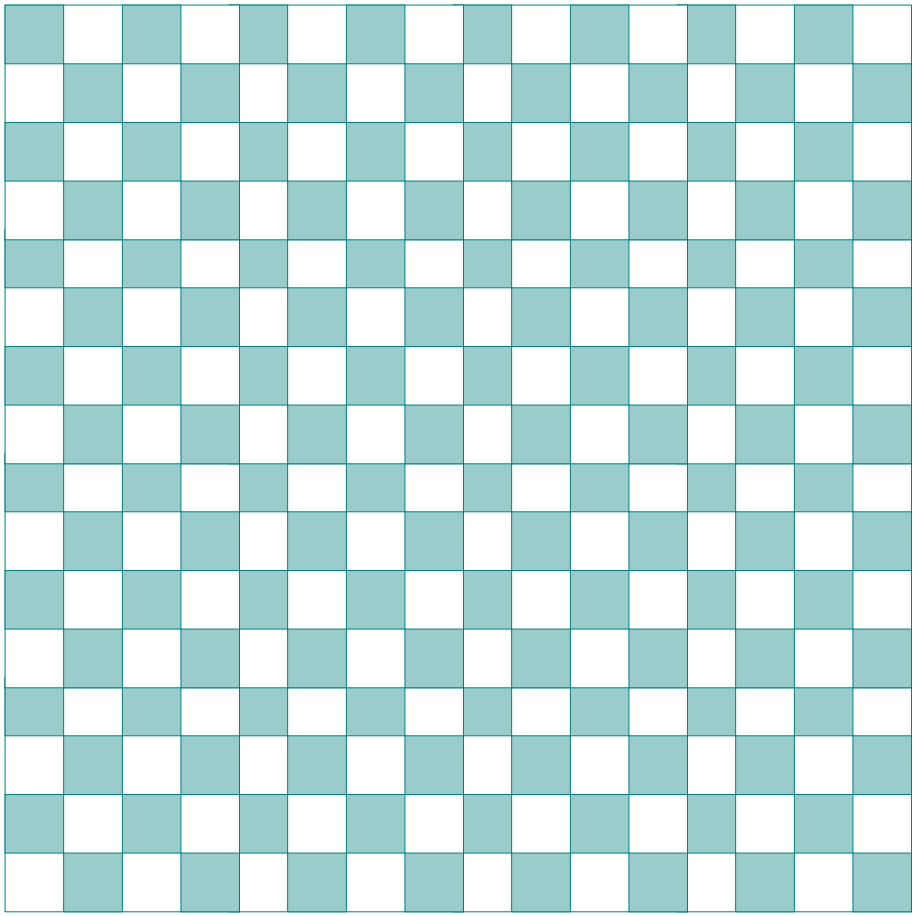
NSolver Solution:

```
using Plantation.Solver;
using System;
public class Rabbit : Solver {
  public Var rabbits, pheasants;
  public void Run() {
    rabbits = var(0, 20, "Rabbits");
    pheasants = var(0, 20, "Pheasants");
    Post(rabbits.Sum(pheasants).Eq(20));
    Post(rabbits.Prod(4).Sum(pheasants.Prod(2)).Eq(56));
    Console.WriteLine(rabbits+" "+pheasants);
  }
  public static void Main() {(new Rabbit()).Run();}
}
```

C#

N Queens Problem

- If n = 100
- Imagine writing 100 nested for loops!
- NSolver takes 0.15 sec
- Brute force – several days
- Search space 100^{100}



Basic Concepts

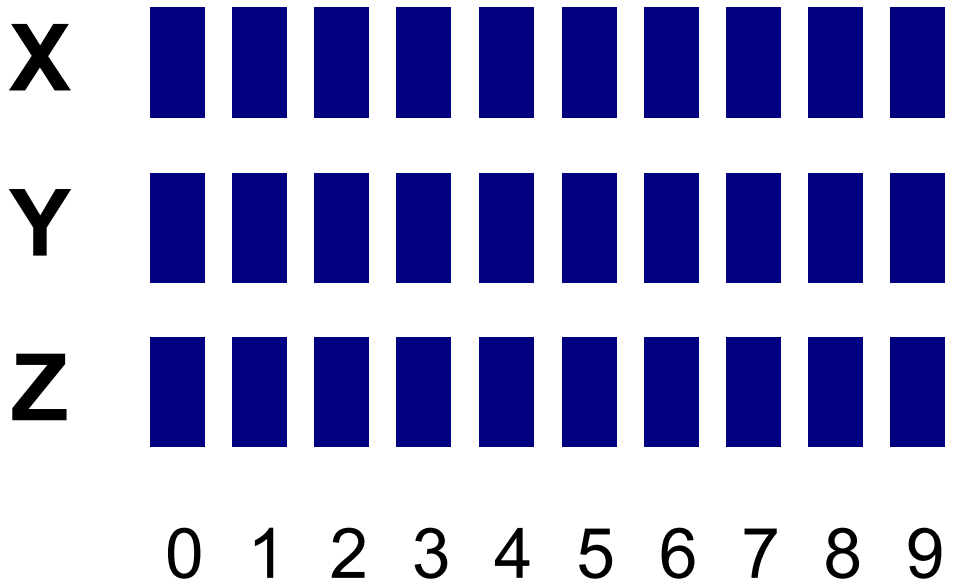
- There are only 3 key concepts to learn in CP:
  - Variables
  - Constraints
  - Search

Search

- Why Need Search?
  - constraints might not be enough to find a solution, or
  - there may be more than one solution
- Brute force search might take "forever"
- CP search algorithms makes use of many different features to improve efficiency, for example:
  - Constraint propagation
  - Backtracking

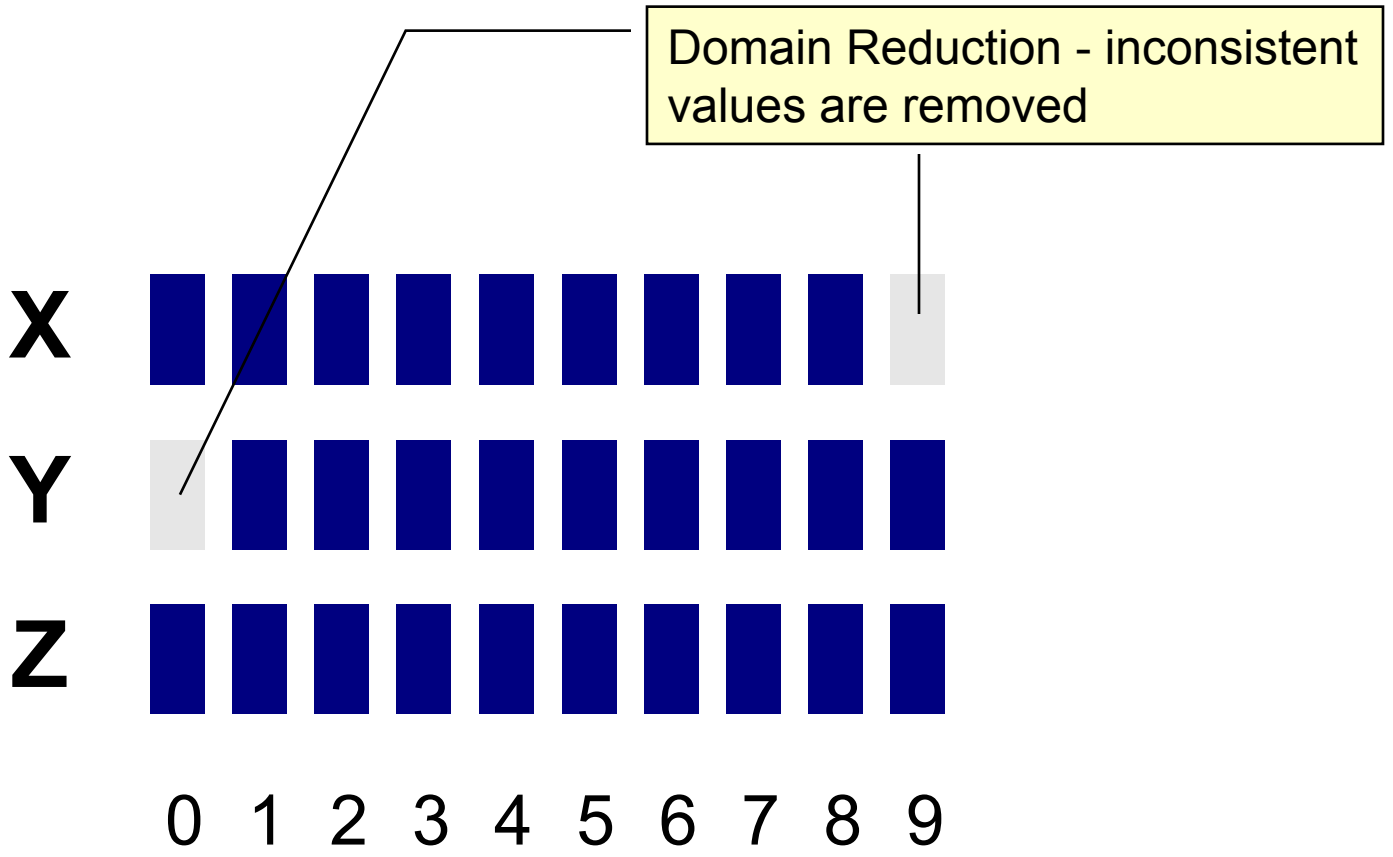
Constraint Propagation

- Three constrained integer variables x, y and z:
  - x is an integer in [0 .. 9]
  - y is an integer in [0 .. 9]
  - z is an integer in [0 .. 9]



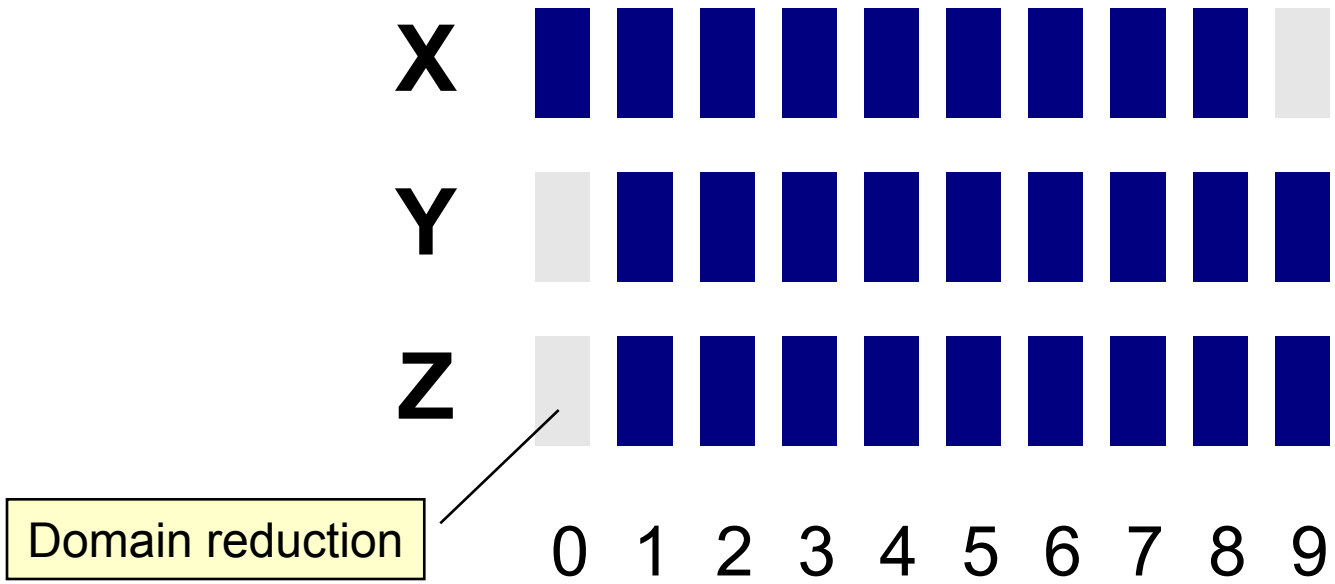
Constraint Propagation

- Post the constraint $x < y$.



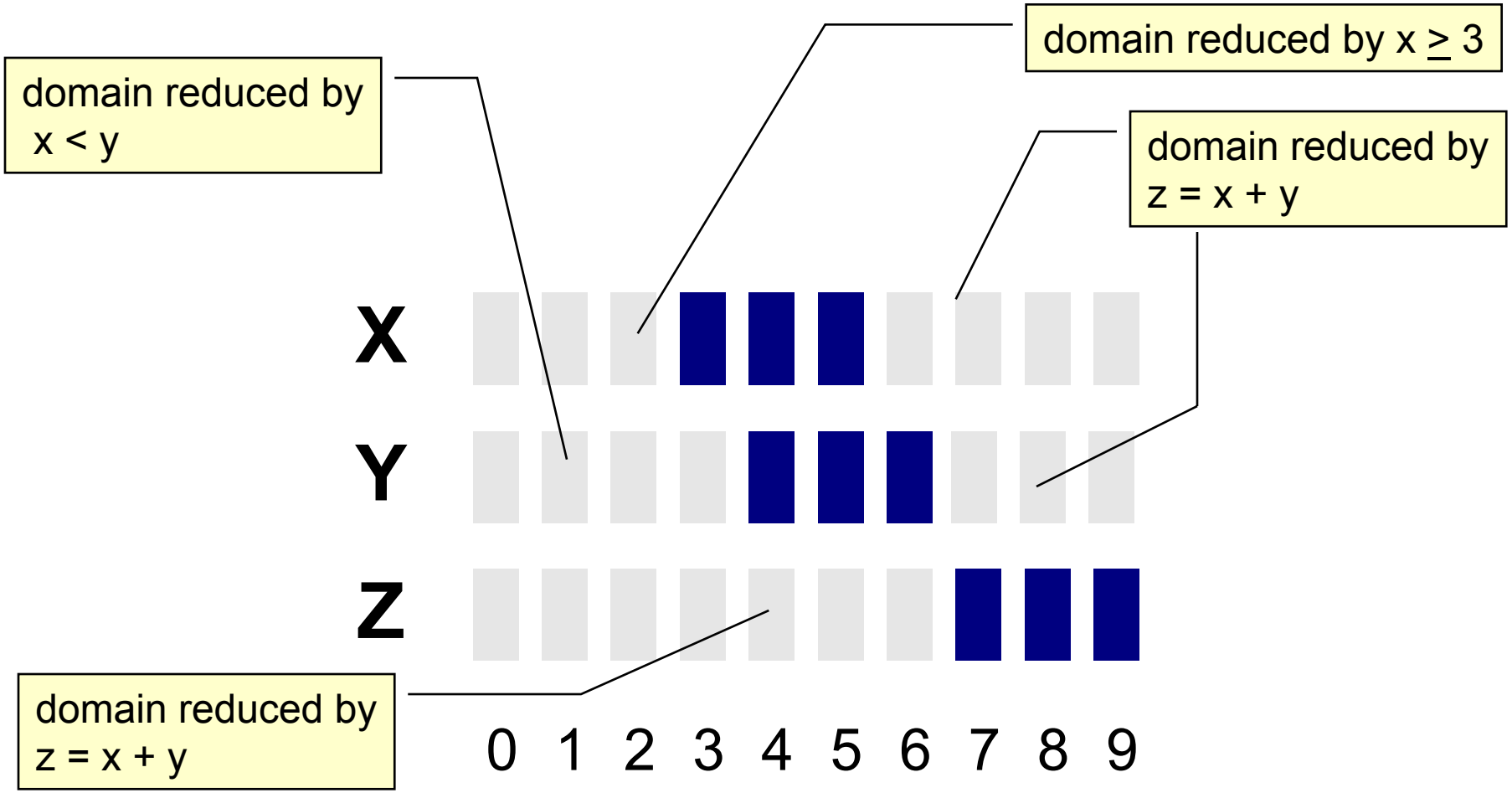
Constraint Propagation

- Post the constraint $z = x + y$.



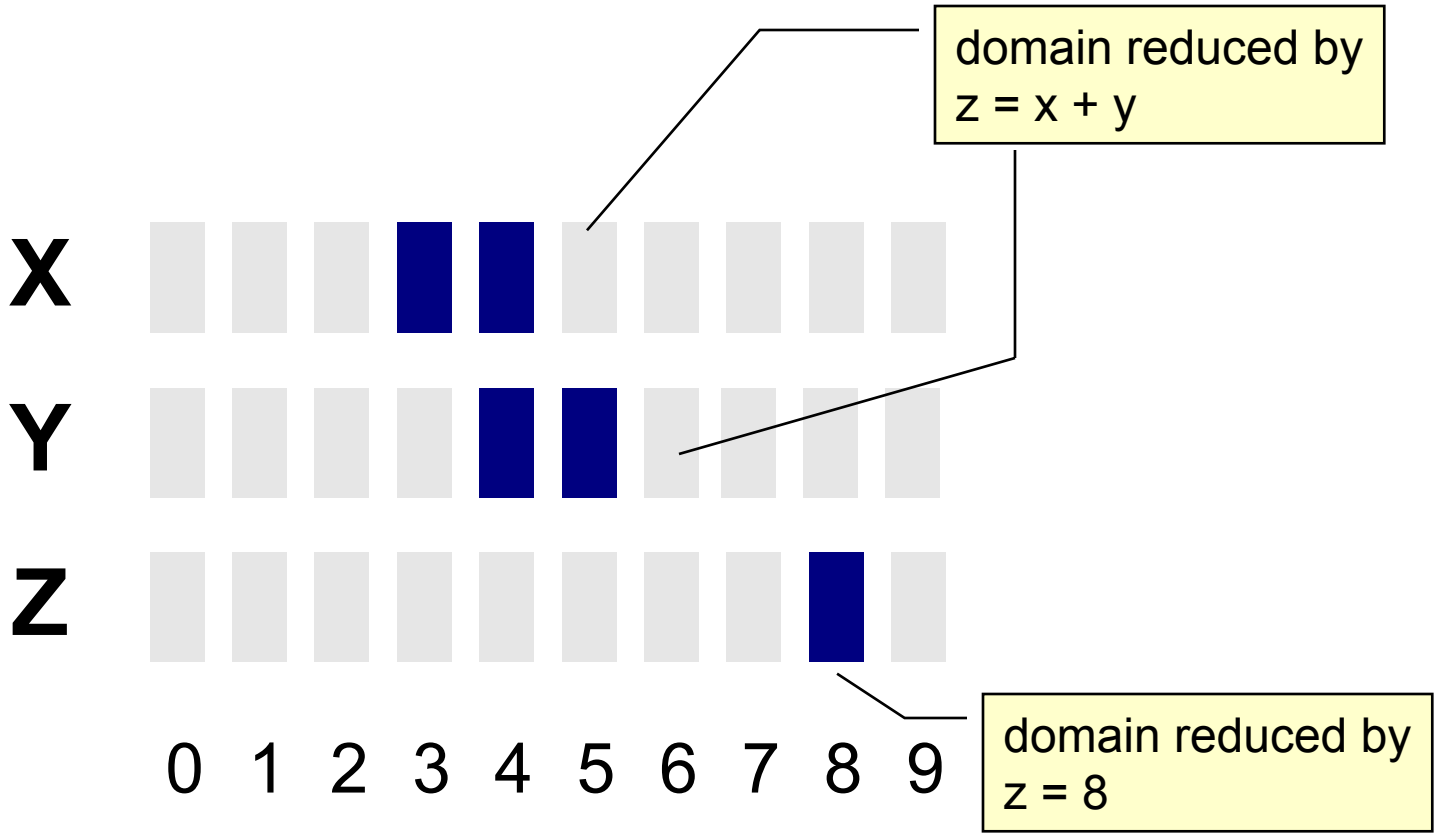
Constraint Propagation

- Post the constraint $x > 3$.



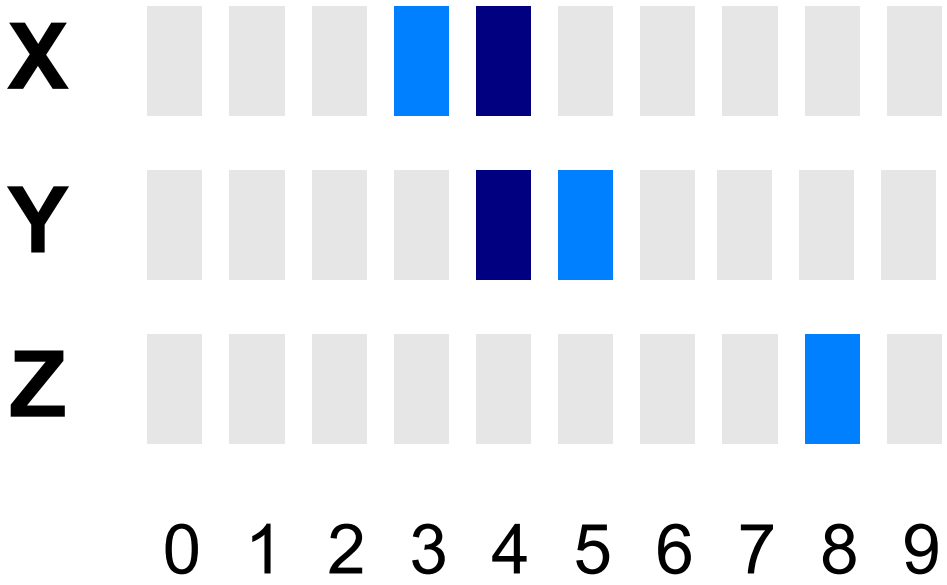
Constraint Propagation

- Post the constraint $z = 8$.



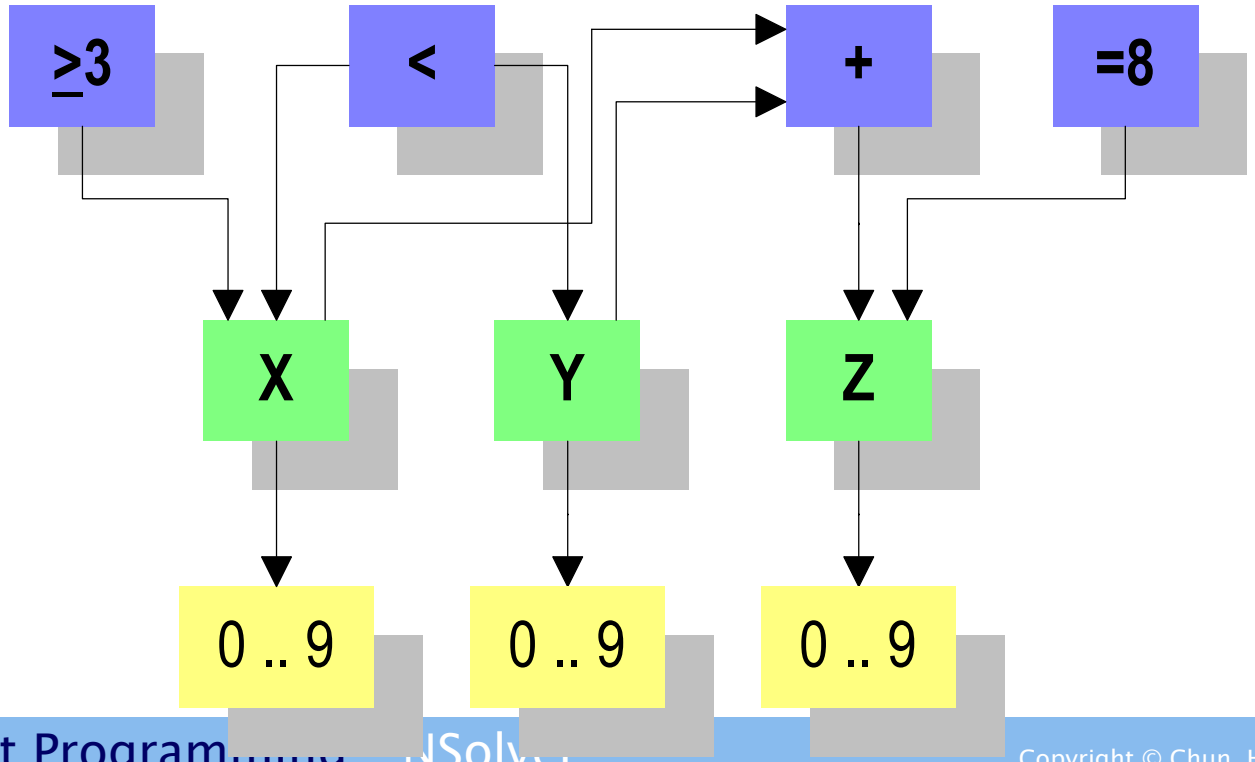
Constraint Propagation

- Search for solutions:
  - x = 3, y = 5, z = 8

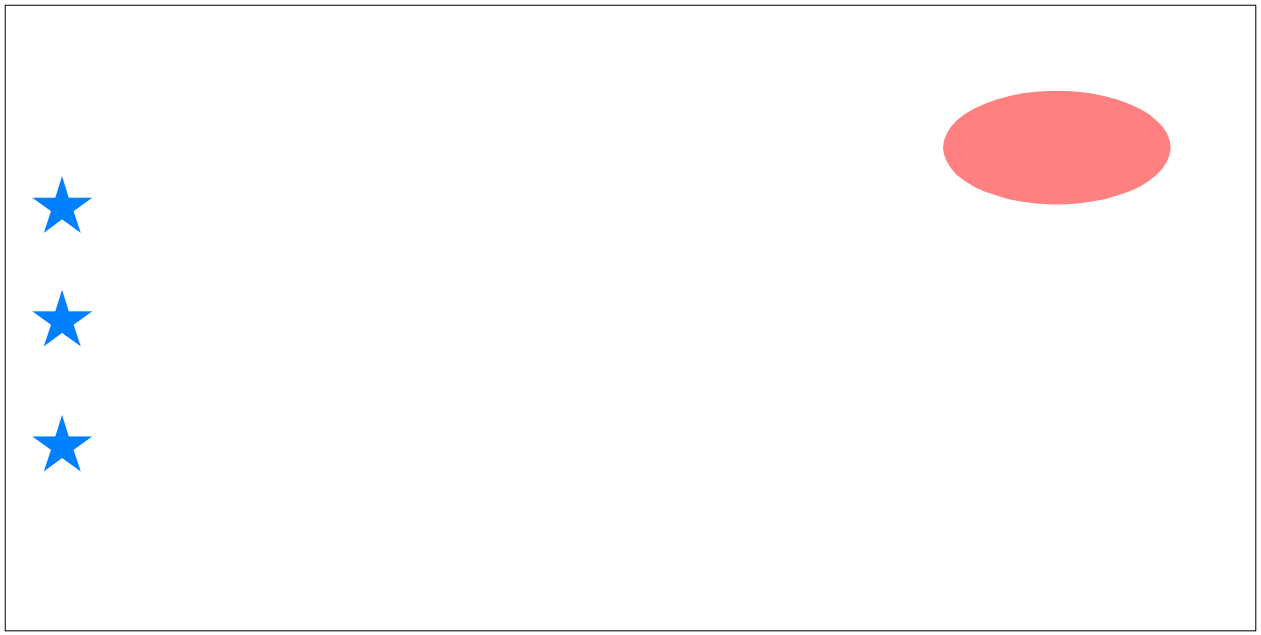


Constraint Propagation

- Three variables x, y, and z
- Domain between 0 and 9
- Constraints: $x < y$, $z = x + y$, $x > 3$, and $z = 8$

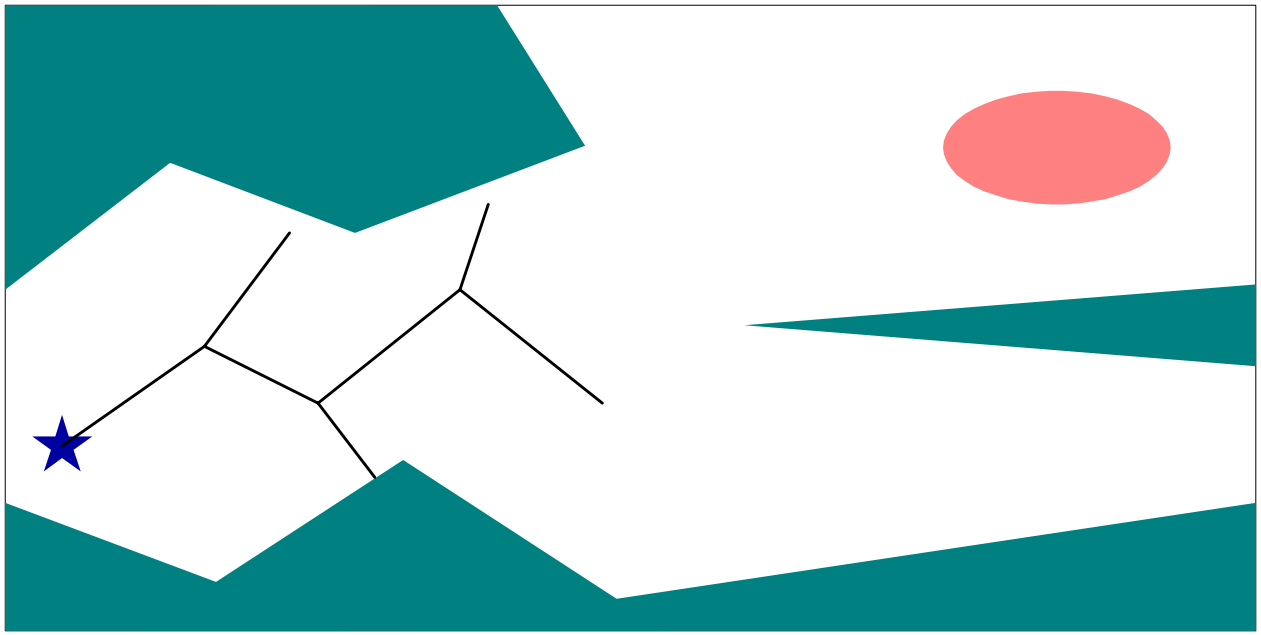


Constraint Propagation



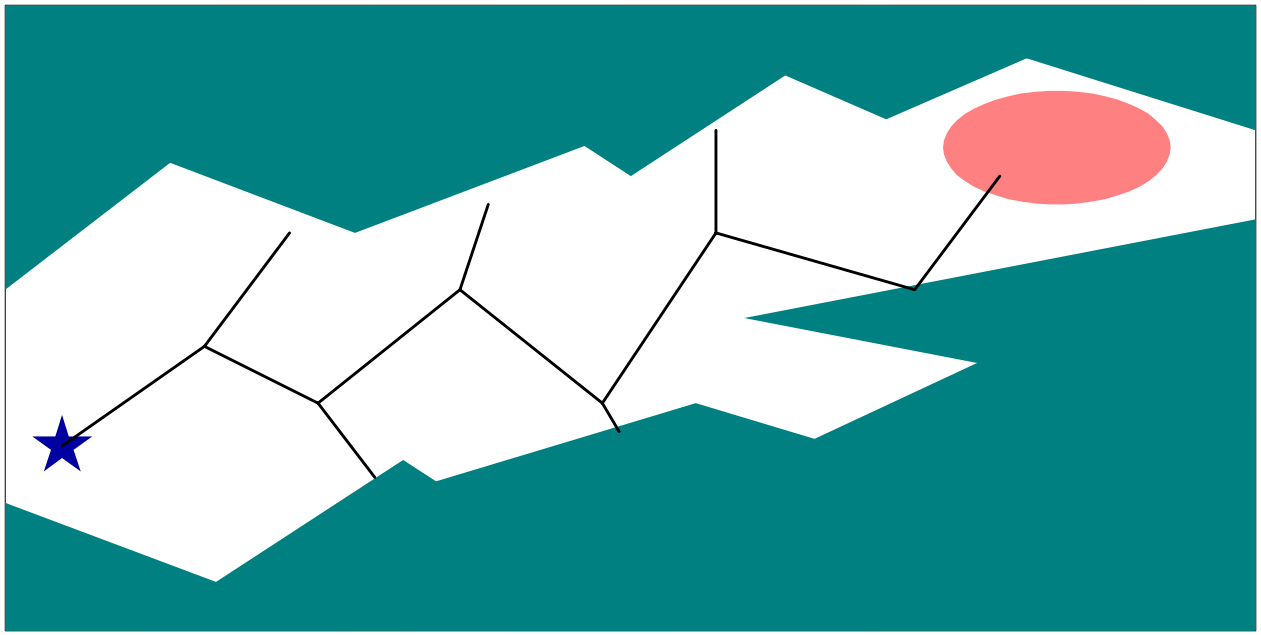
Domain Reduction

Domain Reduction

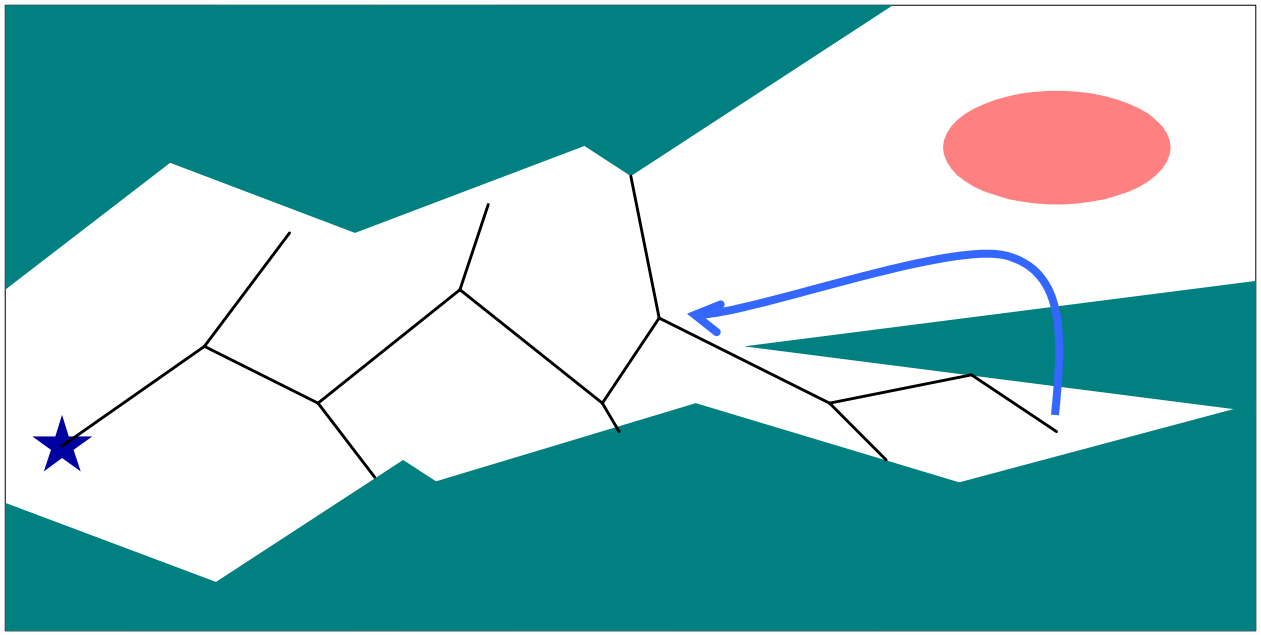


Domain Reduction

Solution !

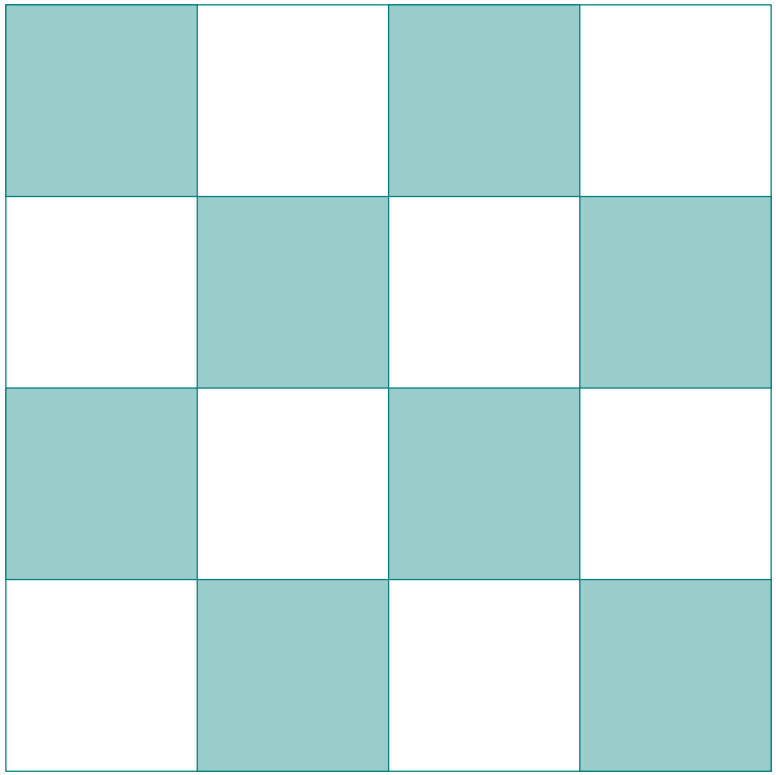


Backtracking



Backtracking

- N-Queens Problem

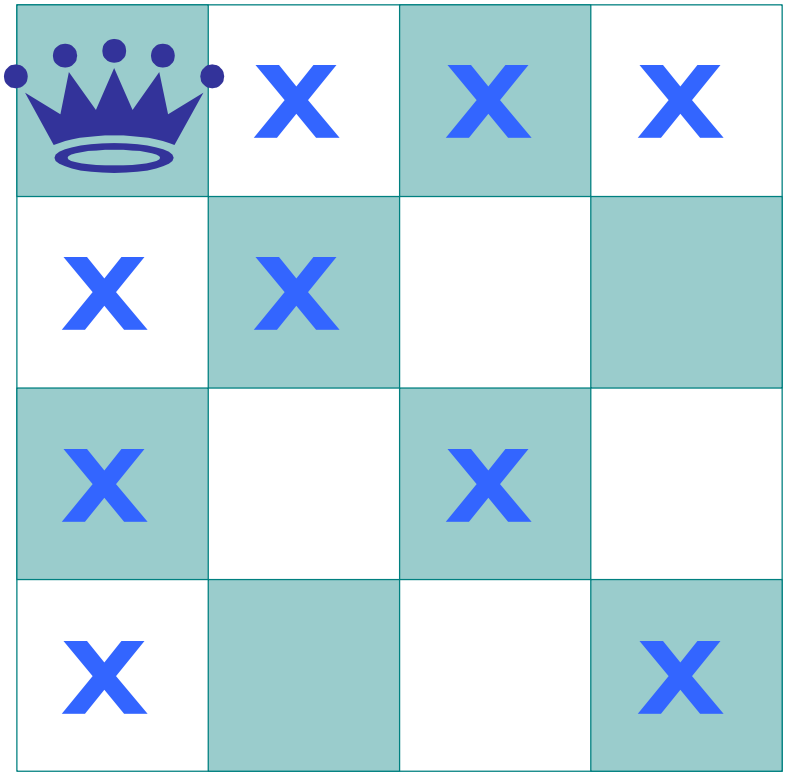


The N-Queen Problem

- Place n queens on an n x n chessboard such that no queen can take another
- Variables
  - One queen per row
  - One variable per row, representing the column where the queen is located:
  - V1, V2, V3, V4
- Domain
  - The columns: {a b c d}
- Constraints
  - Different columns
  - Different diagonals

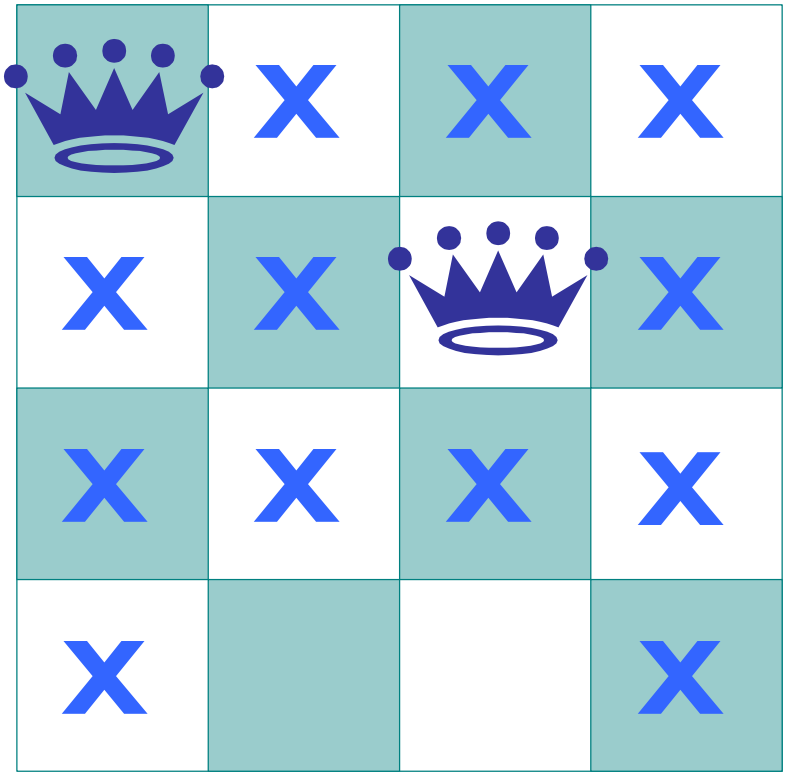
N-Queens

- Assign value and propagate constraints
  - domains are reduced



Backtrack if Fail

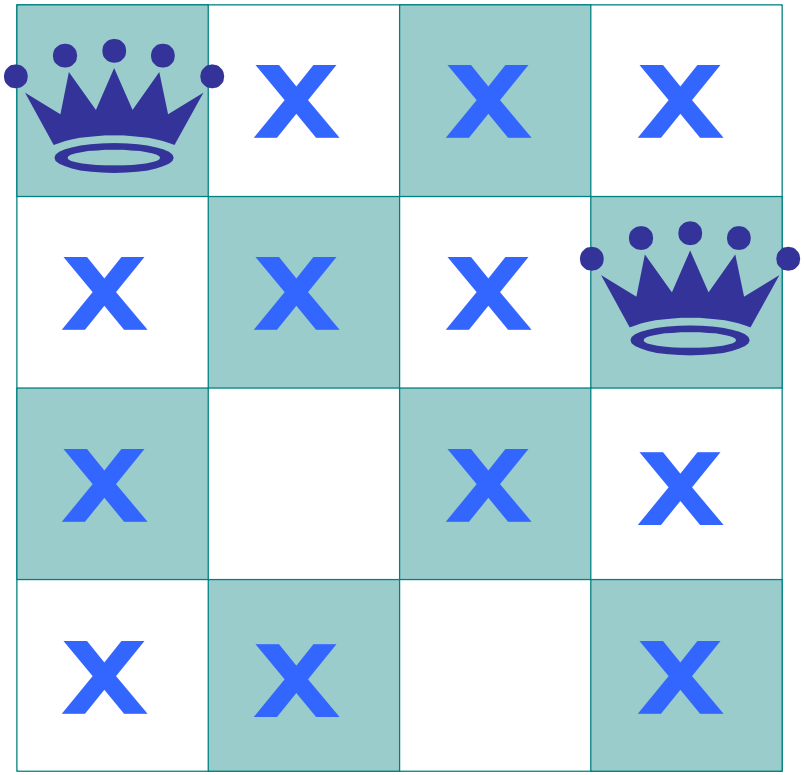
- No solution – trigger backtracking



→No Solution - Backtrack

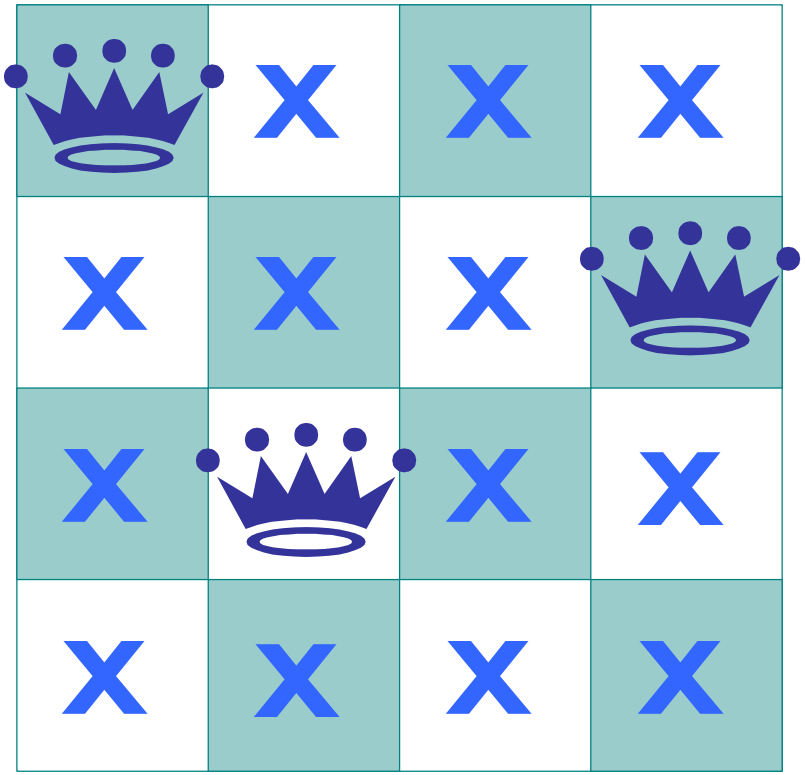
Backtracking

- Undo and continue from previous choice point



Backtracking

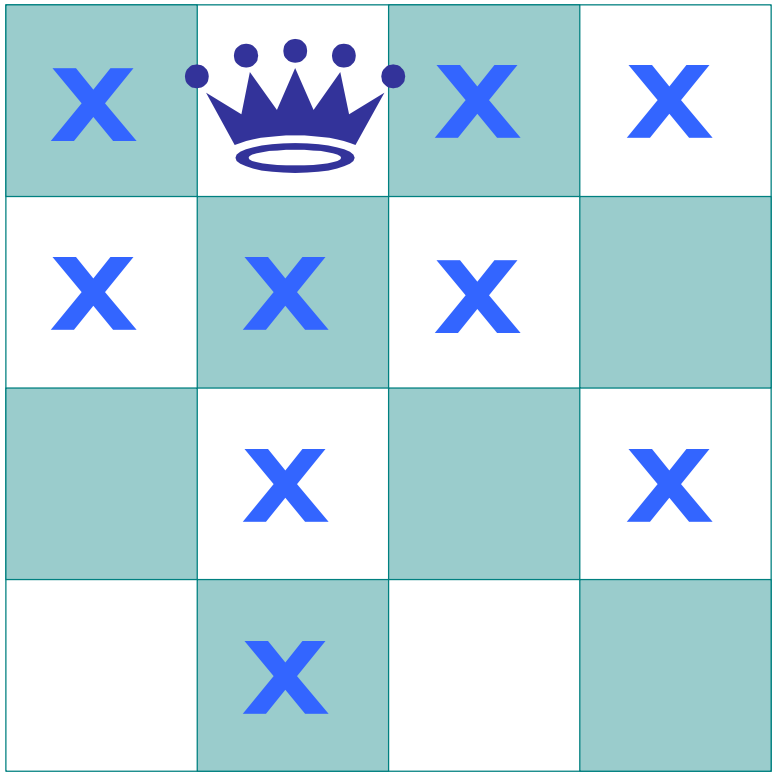
- No solution!



→No Solution - Backtrack

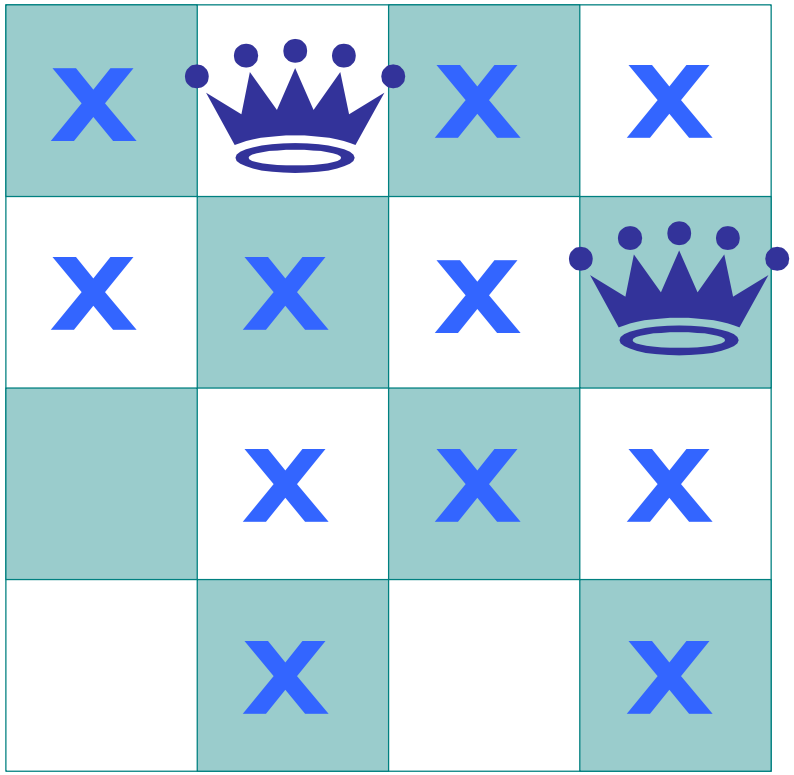
Backtracking

- Backtrack to previous choice point.

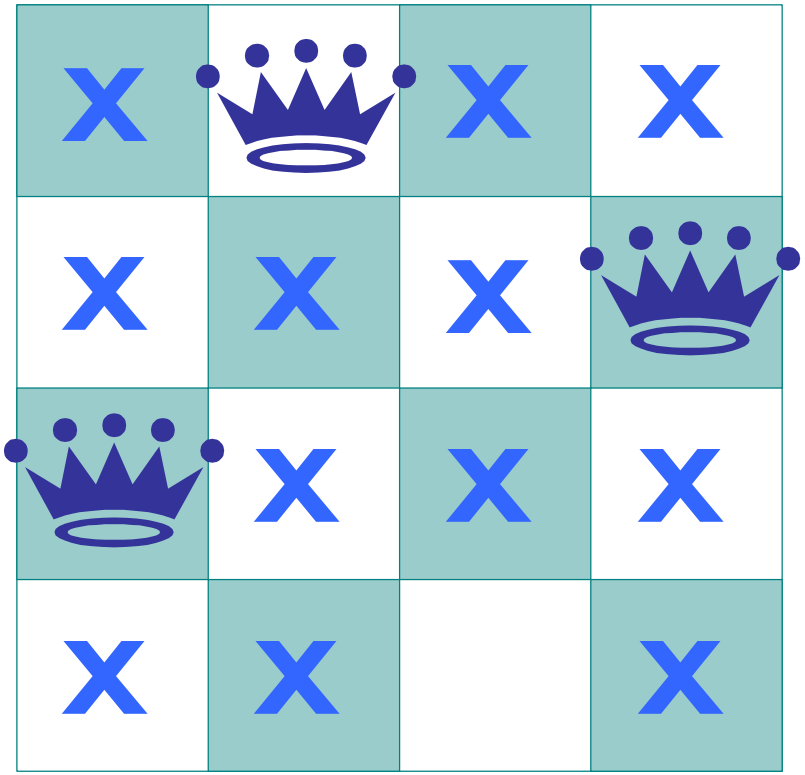


Backtracking

- Continue with the search

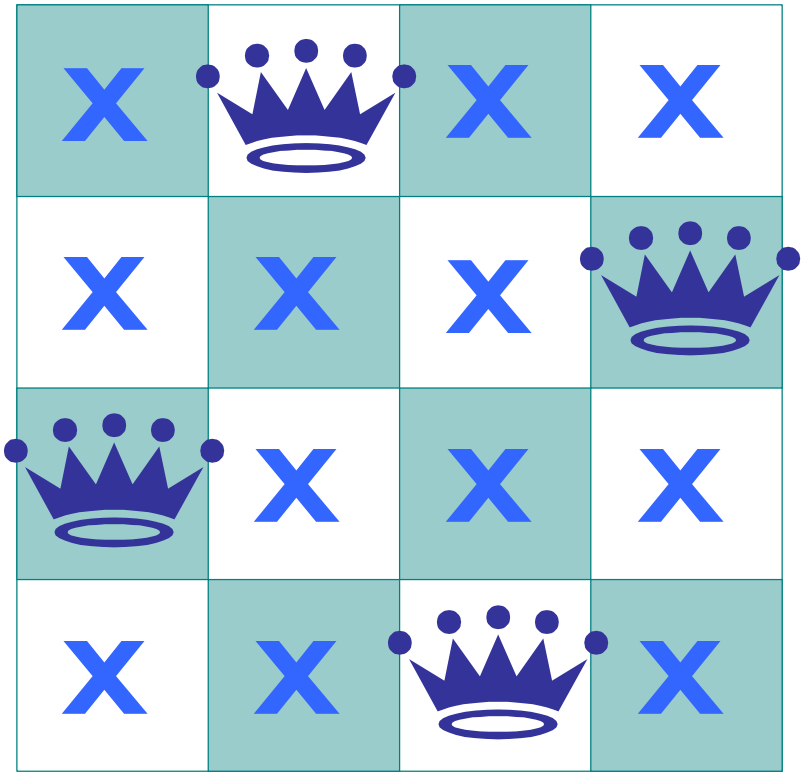


Backtracking



Backtracking

- Solution found!



Advantages of CP

- Declarative programming
  - separate problem solving from problem specification (or model)
  - e.g. can load program as XML data
- A highly efficient search for problems with large search space
  - due to constraint propagation and domain reduction

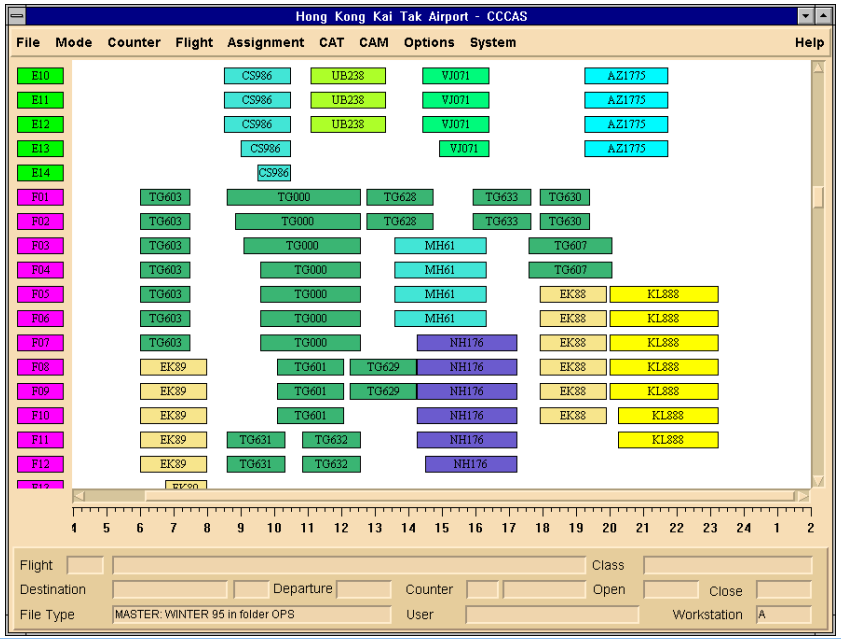
Agenda

- Concepts Behind CP
- Case Studies
- Technical Details

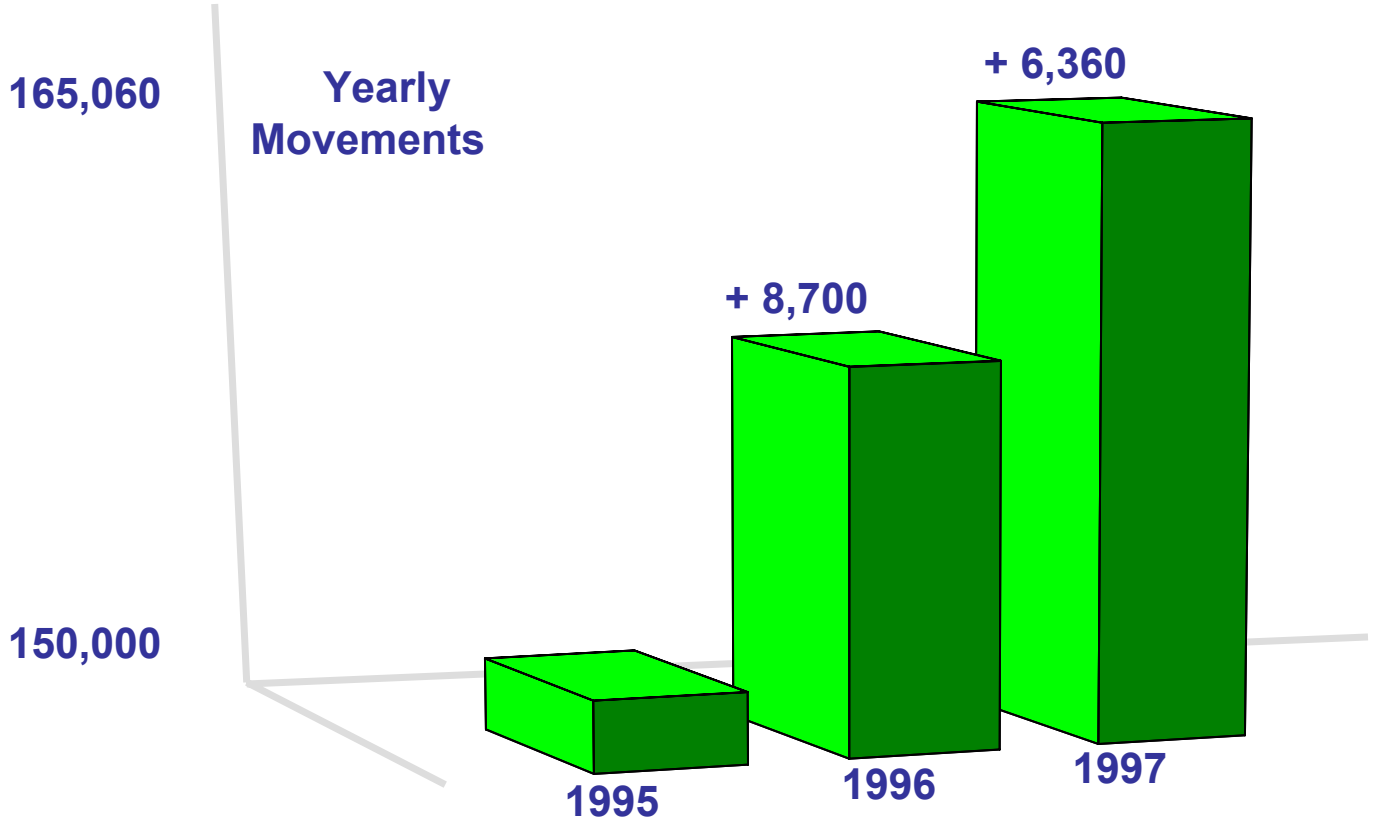
CP Case Studies

Civil Aviation Department

- Computerized check-in counter allocation
- Combine simulation with constraint-based scheduling
- Increase traffic while maintaining service levels

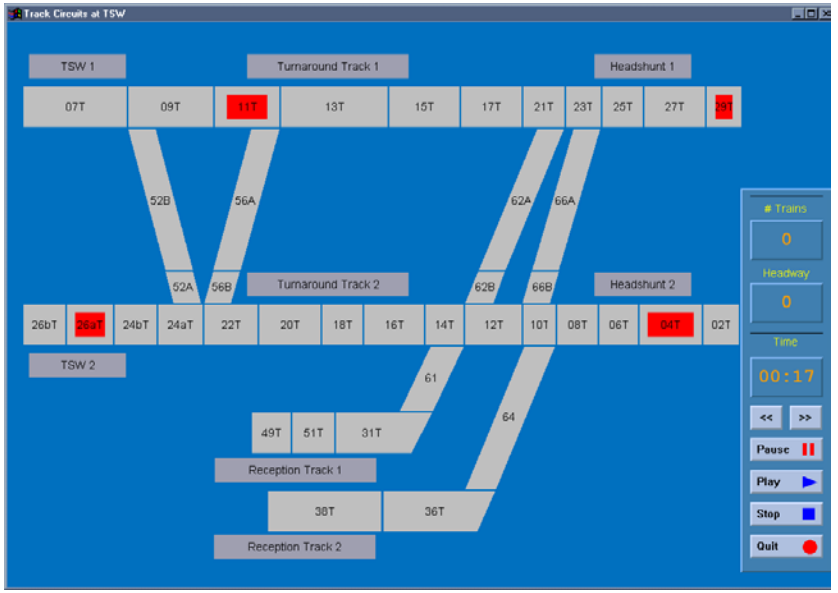


Improved Throughput



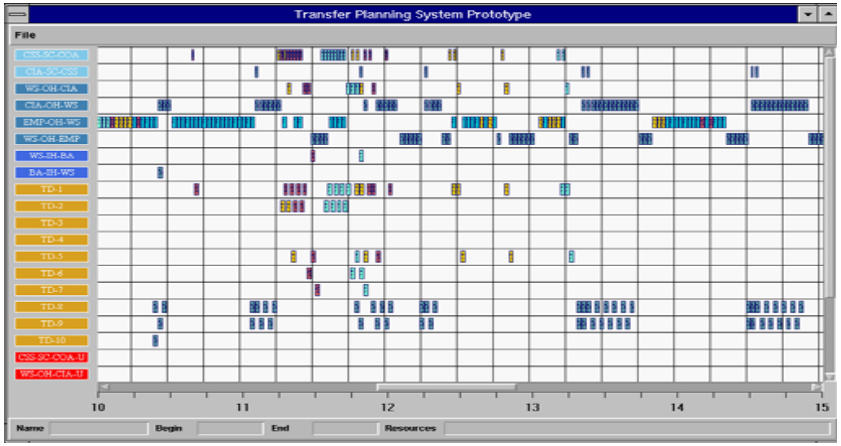
Mass Transit Railway

- Schedule dispatch of trains at TSW line
- Complexity too difficult for humans
- Increased number of trains during rush hours



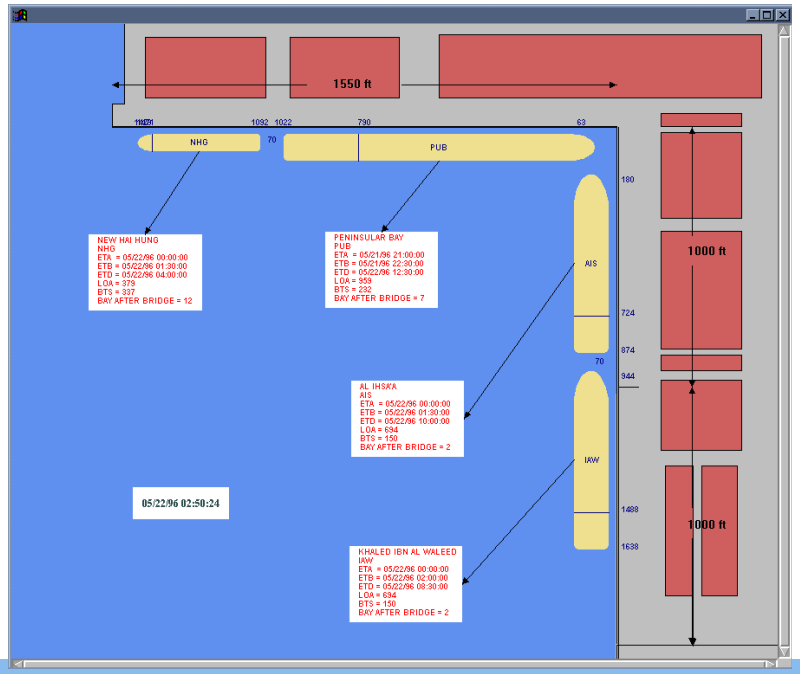
HK Air Cargo Terminals

- Allocation of equipment
- Schedule ULD movement
- Allocation of work space
- Rostering of workers
- Increased efficient and flexibility to cope with dynamic changes

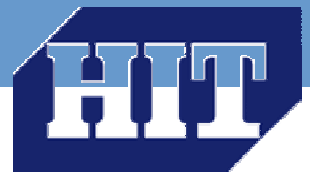


Modern Terminals Ltd

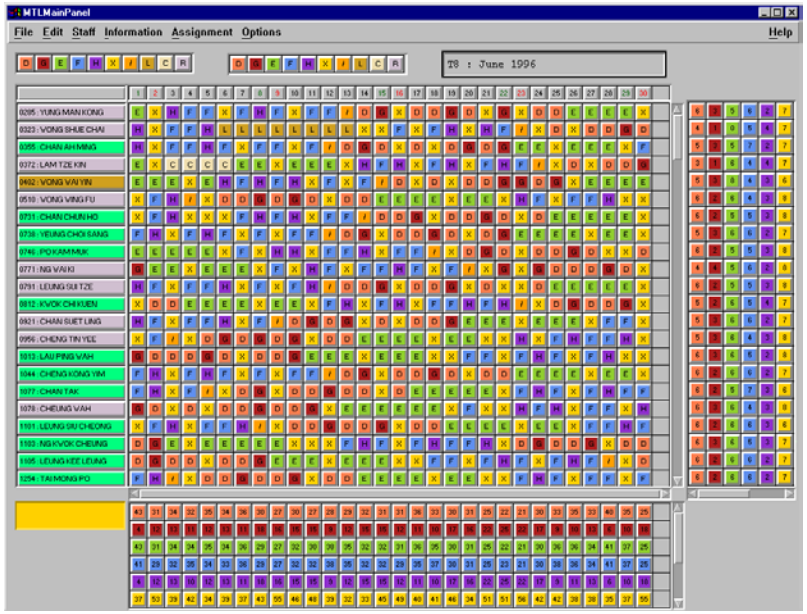
- Rostering for equipment operators
- Container vessel berth allocation
- Increase number of vessel berthing
- Increased efficiency in manpower usage



HK Int'l Terminals

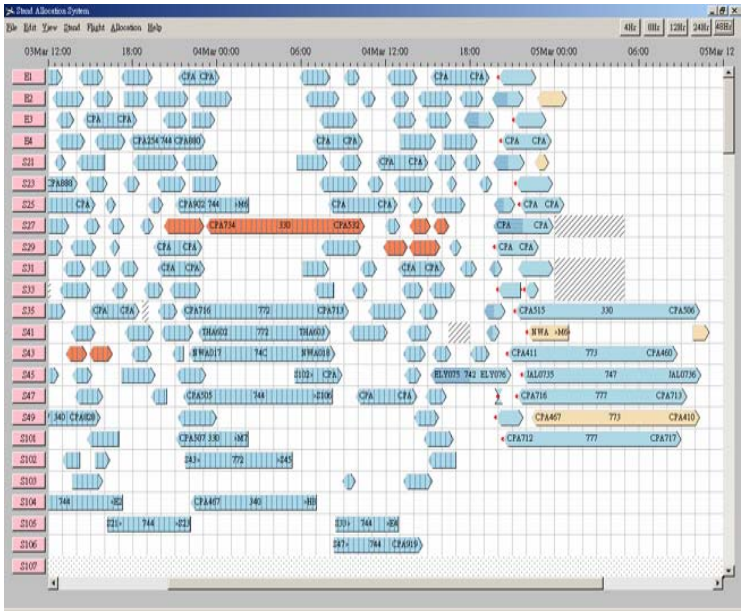


- Yearly workforce forecasting
- Monthly roster of equipment operators
- Support flexible shifts
- Increased overall efficiency of container yard



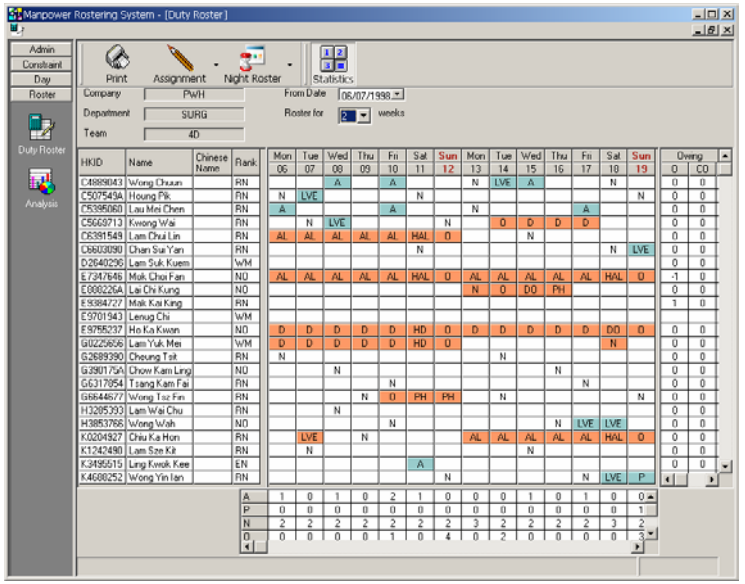
Airport Authority HK

- Planning and real-time management of the Chek Lap Kok airport apron
- 24x7 real-time stand allocation
- Multi-user system used in control tower
- 1999 AAAI Award Winner



HK Hospital Authority

- Staff rostering for public hospital wards
- Satisfy all HA and Labor regulations
- Ensure fairness
- Cope with changes in demand and availability
- 2000 AAAI Award Winner



Agenda

- Concepts Behind CP
- Case Studies
- Technical Details

Technology Spikes

- Search Mechanism
- Constraint Network
- Constraint Propagation
- Reversible Assignments

Spike #1 – Search Mechanism

- Ability to store search tree as it expands
  - Solution: use .NET stacks
- Ability to store all choice points – branches in search tree – without expanding the tree!
  - Solution: use recursive statements
- Ability to keep snapshots for backtracking
  - Solution: simply keep track of stack size

Store Expanding Tree

- Use .NET Stack

```
internal sealed class State : ICloneable {
  internal Stack orStack = new Stack();
  ...
}
```

C#

Store Choice Pts

- Use recursion
- Pseudo code for choice points

```
Goal Instantiate(Var x) {
  int y = next value();
  return (x=y or (x!=y and
                  Instantiate(x)));
```

Keep Snapshot for Backtracking

- Instead of cloning everything, just keep a pointer to stack:
  - orStackSize = state.orStack.Count;

C#

Technology Spikes

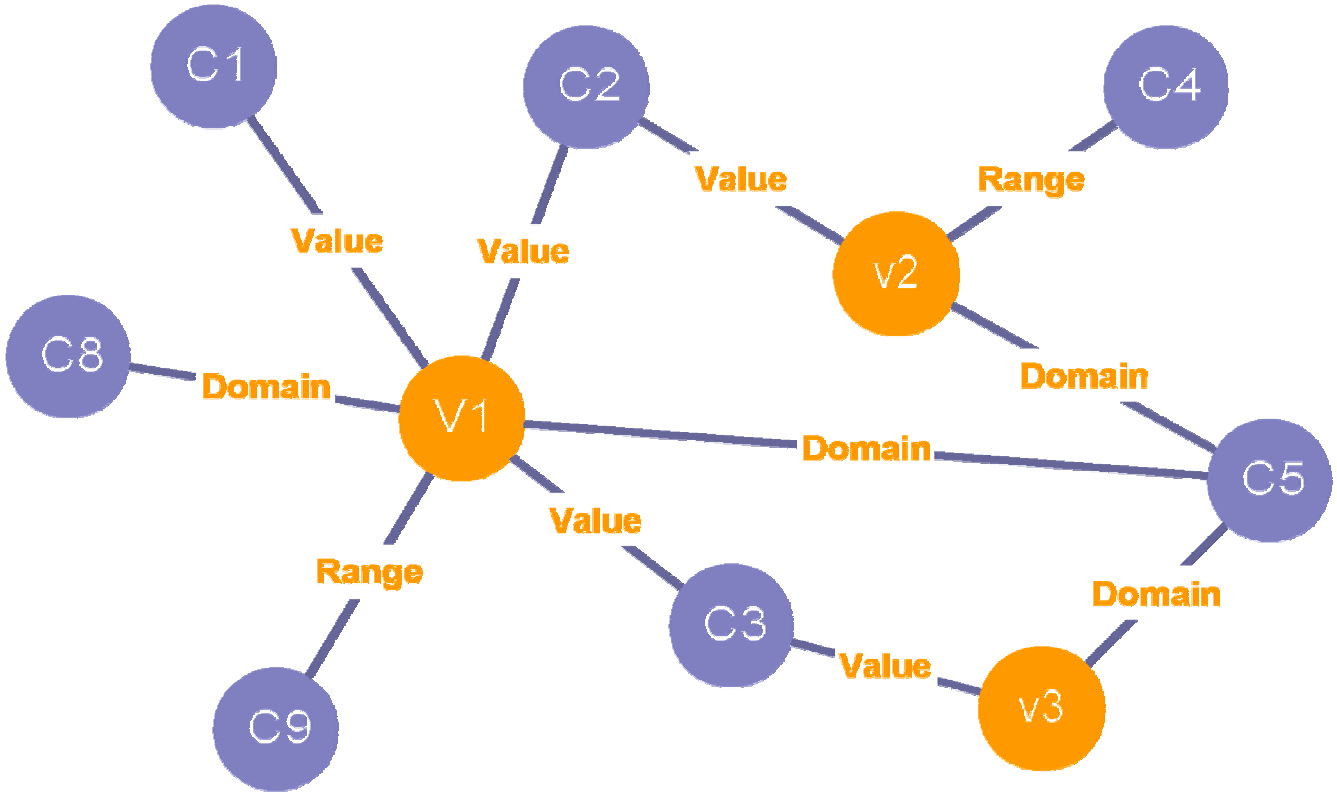
- Search Mechanism
- Constraint Network
- Constraint Propagation
- Reversible Assignments

Spike #2 – Constraint Network

- Ability to find all “applicable” constraints without any search!
  - Solution: store constraints as a network

A Constraint Network

- Each variable knows which constraints are related to it and the event that triggers the constraint!



Storing Network in Variables

- Partial code for constraint network:

```
public class Var : IEnumerable {
  internal ArrayList valueAction = null;
  internal ArrayList rangeAction = null;
  internal ArrayList domainAction = null;
  ...
}
```

C#

Technology Spikes

- Search Mechanism
- Constraint Network
- Constraint Propagation
- Reversible Assignments

Spike #3 – Constraint Propagation

- Ability to automatically propagate changes whenever “anything” happens
  - Solution: use .NET event handling

Using .NET Events for Propagation

- Partial code for event handling:

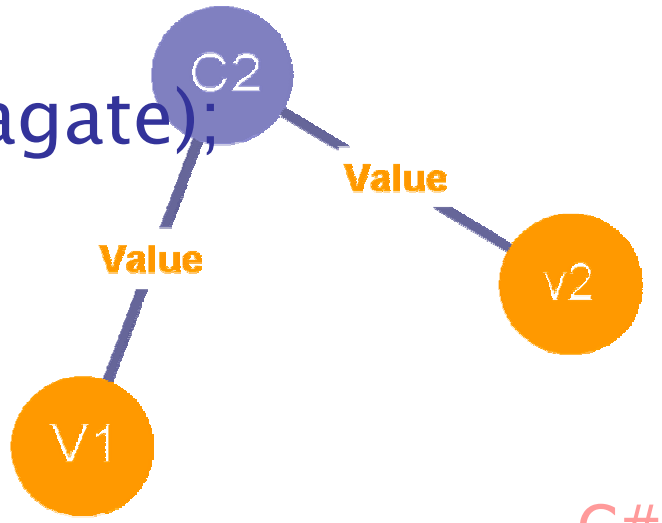
```
public class Var : IEnumerable {
  public event EventHandler ValueAction;
  public event EventHandler RangeAction;
  public event EventHandler DomainAction;
  public void OnValueChanged() {
    if (ValueAction!=null)
      ValueAction(this, EventArgs.Empty);
  }
  ...
}
```

C#

Hooking It Up

- Hook up .NET event handling:

```
class MyCnst : Constraint {
  public override void Post() {
    v1.ValueAction +=
      new EventHandler(Propagate);
    v2.ValueAction +=
      new EventHandler(Propagate);
  }
  ...
}
```



C#

Technology Spikes

- Search Mechanism
- Constraint Network
- Constraint Propagation
- Reversible Assignments

Spike #3 – Reversible Assignments

- Ability to undo mistakes!
  - Solution: use .NET stacks

Wouldn't It Be Nice, If...?

```
int a = 5;
a = 10; //oops!
undo(a);
```

Example of Requirement

```
using System;
public class Test {
  public static void Main() {
    rint a = 5;
    a.Value = 10;
    int time = a.TimeStamp;
    a.Value = 20;
    a.Value = 30;
    a.Value = 40;
    Console.WriteLine(a);
    Console.WriteLine(a.Undo());
    Console.WriteLine(a.Undo(time));
  }
}
```

- > Revert
- 40
- 30
- 10

C#

What Do I Need to Implement?

```
using System;
public class Test {
  public static void Main() {
    rint a = 5;
    a.Value = 10;
    int time = a.TimeStamp;
    a.Value = 20;
    a.Value = 30;
    a.Value = 40;
    Console.WriteLine(a);
    Console.WriteLine(a.Undo());
    Console.WriteLine(a.Undo(time));
  }
```

define Value property

implicit cast

define Timestamp property

stack to recall values

define Undo

define ToString

C#

Putting It Altogether

- NSolver
  - A powerful CP engine in 120K!
- Next steps:
  - Add Log4Net
  - Add NUnit
  - Allow XML input
  - Add OCL parser
  - Extend with Rules

Questions?