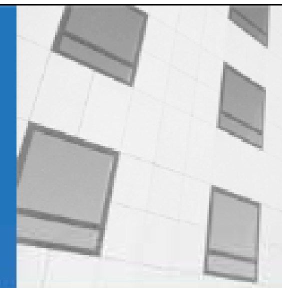


Progress in Linear and Integer Programming and Emergence of Constraint Programming

Dr. Irvin Lustig

Manager, Technical Services

Optimization Evangelist

ILOG Direct

Agenda

Outline

- Mathematical Programming
  - Improvements in Performance
- Constraint Programming
  - A Quick Tutorial
- Constraint Programming Successes

2

Mathematical Programming

Some material courtesy of Bob Bixby

3

Mathematical Programs

Linear Programming

Minimize $c^T x$ ← Objective Function

Subject to $Ax = b$ ← Constraints

$l \le x \le u$

Lower Bounds → l; Upper Bounds → u; Decision Variables → x

4

Mathematical Programs

Linear Programming

Minimize $c^T x$

Subject to $Ax = b$ (LP)

$l \leq x \leq u$

```
Maximize
      x1 + 2 x2 + 3 x3
Subject To
    - x1 +   x2 + x3 ≤ 20
      x1 - 3 x2 + x3 ≤ 30

      0 ≤ x1 ≤ 40
      x2, x3 ≥ 0
```

5

Mathematical Programs

Linear Programming

- **George Dantzig, 1947**
  - Introduces LP and recognized it as more than a conceptual tool: Computing answer important.
  - Invented "primal" simplex algorithm.
  - First LP solved: Laderman, 9 cons., 77 vars., 120 MAN-DAYS.
- **What is the single most important event in LP since Dantzig?**
  - We have (since ~1990) 3 algorithms to solve LPs
    - Primal Simplex Algorithm (Dantzig, 1947)
    - Dual Simplex Algorithm (Lemke, 1954)
    - Barrier Algorithm (Karmarkar, 1984 and others)

6

Linear Programming

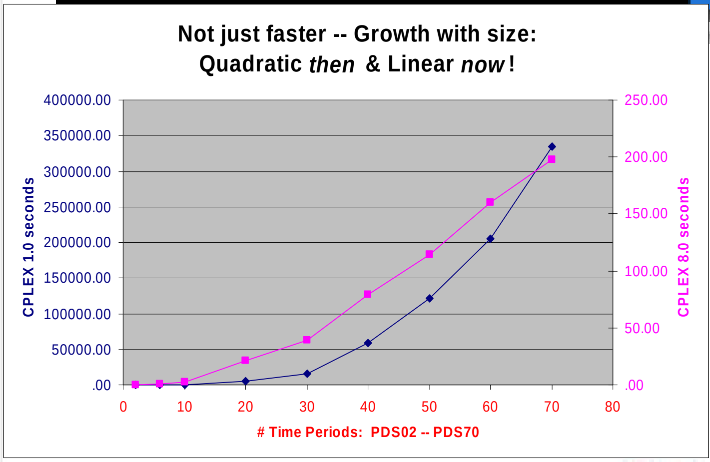
PDS Models

"Patient Distribution System": Carolan, Hill, Kennington, Niemi, Wichmann, *An empirical evaluation of the KORBX algorithms for military airlift applications*, Operations Research 38 (1990), pp. 240-248

MODEL	ROWS	CPLEX1.0 1988	CPLEX5.0 1997	CPLEX8.0 2002	SPEEDUP 1.0→8.0
pds02	2953	0.4	0.1	0.1	4.0
pds06	9881	26.4	2.4	0.9	29.3
pds10	16558	208.9	13.0	2.6	80.3
pds20	33874	5268.8	232.6	20.9	247.3
pds30	49944	15891.9	1154.9	39.1	406.4
pds40	66844	58920.3	2816.8	79.3	743.0
pds50	83060	122195.9	8510.9	114.6	1066.3
pds60	99431	205798.3	7442.6	160.5	1282.2
pds70	114944	335292.1	21120.4	197.8	1695.1
		Primal Simplex	Dual Simplex	Dual Simplex	

7

Linear Programming



8

Linear Programming

BIG TEST: The testing methodology

- **Not possible for one test to cover 10+ years: Combined several tests.**
- **The biggest single test:**
  - Assembled 680 real LPs (up to 7 million consts.)
  - Test runs: Using a time limit (4 days per LP), two chosen methods would be compared as follows:
    - Run method 1: Generate 680 solve times
    - Run method 2: Generate 680 solve times
    - Compute 680 ratios and form GEOMETRIC MEAN (not arithmetic mean!)

The same methodology was applied throughout.

9

Linear Programming

Progress: 1988 – Present

- Algorithms
  - Best simplex **960x**
  - Best simplex + barrier **2360x**
- Machines
  - Simplex algorithms **800x**
  - Barrier algorithms **13000x**

10

Linear Programming

Algorithm comparison: Extracted from the previous results …

- Dual simplex vs. primal: Dual 2x faster
- Best simplex vs. barrier: About even
- Best of three vs. primal: Best 7.5x faster

11

Mixed Integer Programming

Mixed Integer Programming

Minimize $c^T x$

Subject to Ax = b (MIP)

$$l \leq x \leq u$$

Some x are integer

```
Maximize  x1 + 2 x2 + 3 x3 + x4
Subject To
   - x1 +   x2 + x3 + 10 x4  ≤ 20
     x1 - 3 x2 + x3          ≤ 30
            x2       - 3.5 x4 = 0

     0 ≤ x1 ≤ 40   x2, x3 ≥ 0
     2 ≤ x4 ≤ 3
     x4  integer
```

12

Mixed Integer Programming

Computational History: 1950 –1998

- **1954 Dantzig, Fulkerson, S. Johnson: 42 city TSP**
  - Solved to optimality using cutting planes and LP
- **1957 Gomory**
  - Cutting plane algorithm: A complete solution
- **1960 Land, Doig, 1965 Dakin**
  - Branch-and-bound (B&B)
- **1971 MPSX/370, Benichou et al.**
- **1972 UMPIRE, Forrest, Hirst, Tomlin (Beale)**
- **1972 – 1998 Good B&B remained the state-of-the-art in commercial codes, in spite of**
  - 1973 Padberg
  - 1974 Balas (disjunctive programming)
  - 1983 Crowder, Johnson, Padberg: PIPX, pure 0/1 MIP
  - 1987 Van Roy and Wolsey: MPSARX, mixed 0/1 MIP
  - Grötschel, Padberg, Rinaldi …TSP (120, 666, 2392 city models solved)

13

Mixed Integer Programming

1998… A new generation of MIP codes

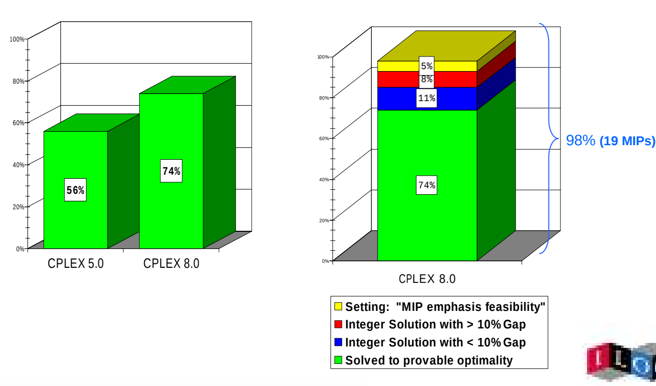
- **Linear programming**
  - Stable, robust dual simplex
- **Variable/node selection**
  - Influenced by traveling salesman problem
- **Primal heuristics**
  - 8 different tried at root
  - Retried based upon success
- **Node presolve**
  - Fast, incremental bound strengthening (very similar to Constraint Programming)
- **Presolve – numerous small ideas**
  - Probing in constraints:
  - $\sum xj \le (\sum uj)\ y,\ \ y = 0/1$
  - $\rightarrow xj \le ujy$ (for all j)
- **Cutting planes**
  - **Gomory**, knapsack covers, flow covers, mix-integer rounding, cliques, GUB covers, implied bounds, path cuts, disjunctive cuts
  - Various extensions
    - Aggregation

14

Mixed Integer Programming

Computational Results I: 964 models (30 hour time limit)

Solving to Optimality **Finding Feasible Solutions**



15

Mixed Integer Programming

Computational Results II: 651 models (all solvable to optimality)

- **Ran for 30 hours using defaults**
- **Relative speedups:**
  - All models (651): **12x**
  - CPLEX 5.0 > 1 second (447): **41x**
  - CPLEX 5.0 > 10 seconds (362): **87x**
  - CPLEX 5.0 > 100 seconds (281): **171x**

16

Mathematical Programming

Summary of Progress

- **Through a combination of advances in algorithms and computing machines, combined with developments in data availability and modern modeling languages, what is possible today could only have been dreamed of even 10 years ago.**
- **The result is that whole new application domains have been enabled**
  - Larger, more accurate models and multiple scenarios
  - Tactical and day-of-operations are possible, not just planning
  - Disparate components of the extended enterprise can now be "optimized" in concert.

17

Constraint Programming

18

Mathematical Programming

Problem Definition

- **Minimize (or maximize) an *Objective Function***
- **Subject to *Constraints***
- **Over a set of values of *Decision Variables***

- **Usual Requirements**
  - Objective function and constraints have closed mathematical forms (linear, quadratic, nonlinear, etc.)
  - Decision variables are real or integer-valued
    - Each variable takes values over an interval

19

Mathematical Programming

Problem Types

- Linear Program
- (Mixed) Integer Program
- Quadratic Program
- Nonlinear Program
- ...

A program is a *problem*

20

Constraint Programming

Computer Programming

- Knuth, 1968, The Art of Computer Programming
  - "An expression of a computational method in a computer language is called a program."
- Programming Paradigms
  - Procedural Programming
  - Object-oriented Programming
  - Functional Programming
  - Logic Programming
  - ....

21

Constraint Programming

Definition

- A computer programming methodology
- Solves
  - Constraint satisfaction problems
  - Combinatorial optimization problems
- Methodology
  - Represent a model of a problem in a computer programming language
  - Describe a search strategy for solving the problem

22

Constraint Programming

Constraint Satisfaction Problems

- **Find a *Feasible Solution***
- **Subject to *Constraints***
- **Over a set of values of *Decision Variables***

- **Usual Requirements**
  - Constraints are easy to evaluate
    - Closed mathematical forms or table lookups
  - Decision variables are values over a discrete set

23

Constraint Programming

Combinatorial Optimization Problems

- **Minimize (or maximize) an *Objective Function***
- **Subject to *Constraints***
- **Over a set of values of *Decision Variables***

- **Usual Requirements**
  - Objective Function and Constraints are easy to evaluate
    - Closed mathematical forms or table lookups
  - Decision variables are values over a discrete set

24

Constraint Programming

What is a potential representation?

- **Let $x_1, x_2, \ldots, x_n$ be the *decision variables***
- **Each x_j (j = 1, 2, ..., n) has a domain D_j of allowable values**
  - Note that a domain may be finite or infinite
  - A domain may have "holes" (e.g., even numbers between 0 and 100)
  - The allowable values could be elements of a particular set
- **A *constraint* is a function *f***

  $f(x_1, x_2, \ldots, x_n) \in \{0, 1\}$
  - The function may just be a table of values!

25

Constraint Programming

Constraint Satisfaction Problem

- **A constraint satisfaction problem is**

  Find values of $x_1, x_2, \ldots, x_n$ such that

  $$x_j \in D_j \quad (j = 1, 2, \ldots, n)$$
  $$f_k(x_1, x_2, \ldots, x_n) = 1 \quad (k=1,\ldots,m) \qquad \text{(CSP)}$$

- **A *solution* of this problem is any set of values satisfying the above conditions**

26

Constraint Programming

Optimization Problem

- **Suppose you have an *objective function***

  $g(x_1, x_2, \ldots, x_n)$

  that you wish to minimize.

- **Optimization Problem is then**

  minimize $g(x_1, x_2, \ldots, x_n)$

  subject to

  $$x_j \in D_j \quad (j = 1, 2, \ldots, n)$$
  $$f_k(x_1, x_2, \ldots, x_n) = 1 \quad (k=1,\ldots,m)$$

27

Constraint Programming

Examples of Constraints

- **Logical constraints**
  - If **x** is equal to 4, then **y** is equal to 5
  - Either "Activity a" precedes "Activity B" OR "Activity B" precedes "Activity A"
- **Global constraints**
  - All of the values in the array **x** are different
  - Element `i` of the array `card` is the number of times that the `i`th element of the array `value` appears in the array `base`
- **Meta constraints**
  - The number of times that the array **x** has the value 5 is exactly 3
- **Element constraint**
  - The cost of assigning person `i` to job `j` is `cost[job[i]]`, when `job[i]` is `j`

28

Constraint Programming

Constraint Programming Provides:

- **A *modeling methodology* for stating decision variables, constraints, and objective functions**
- **A *programming language* for stating a *search algorithm* for finding values of the variables that satisfy the constraints and optimize the objective**
- **A *programming system* that includes**
  - Predefined constraints with powerful *filtering algorithms* for reducing the size of the search space
  - Functionality to allow definitions of new constraints and filtering algorithms

29

Constraint Programming

Examples of Constraints

- **Logical constraints**
  - `(x = 4) => (y = 5)`
  - `(a.end <= b.start) \/ (b.end <= a.start)`
- **Global constraints**
  - `alldifferent(x)`
  - `distribute(card,value,base)`
    - `card[i]` is the number of times `value[i]` appears in `base`
- **Meta constraints**
  - `sum (i in S) (x[i] < 5) = 3;`
- **Element constraint**
  - `z = y[x[i]]`

30

Example

Map Coloring Example

- **Have a list of countries**

```
enum Country {Belgium,Denmark,France,Germany,
              Netherlands,Luxembourg};
```

- **Have a set of colors to use on a map to color the countries**

```
enum Colors {blue,red,yellow,gray};
```

- **Want to decide how to assign the colors to the countries so that no two bordering countries have the same color**

```
var Colors color[Country];
```

The decision variables are values from a ***set***

31

Example

Constraint Programming Model

```
enum Country {Belgium,Denmark,France,Germany,
              Netherlands,Luxembourg};
enum Colors {blue,red,yellow,gray};

var Colors color[Country];

solve {

     color[France] <> color[Belgium];
     color[France] <> color[Luxembourg];
     color[France] <> color[Germany];
     color[Luxembourg] <> color[Germany];
     color[Luxembourg] <> color[Belgium];
     color[Belgium] <> color[Netherlands];
     color[Belgium] <> color[Germany];
     color[Germany] <> color[Netherlands];
     color[Germany] <> color[Denmark];
};
```

Data

Decision Variables

Find all Solutions

Constraints

32

Example

Constraint Satisfaction

```
enum Country {Belgium,Denmark,France,Germany,
              Netherlands,Luxembourg};
enum Colors {blue,red,yellow,gray};

var Colors color[Country];

solve {

     color[France] <> color[Belgium];
     color[France] <> color[Luxembourg];
     color[France] <> color[Germany];
     color[Luxembourg] <> color[Germany];
     color[Luxembourg] <> color[Belgium];
     color[Belgium] <> color[Netherlands];
     color[Belgium] <> color[Germany];
     color[Germany] <> color[Netherlands];
     color[Germany] <> color[Denmark];
};
```

33

Example

Constraint Satisfaction

```
enum Country {Belgium,Denmark,France,Germany,
              Netherlands,Luxembourg};
enum Colors {blue,red,yellow,gray};
var Colors color[Country];



solve {
     color[France] <> color[Belgium];
     color[France] <> color[Luxembourg];
     color[France] <> color[Germany];
     color[Luxembourg] <> color[Germany];
     color[Luxembourg] <> color[Belgium];
     color[Belgium] <> color[Netherlands];
     color[Belgium] <> color[Germany];
     color[Germany] <> color[Netherlands];
     color[Germany] <> color[Denmark];
};
```

34

Example

Optimization

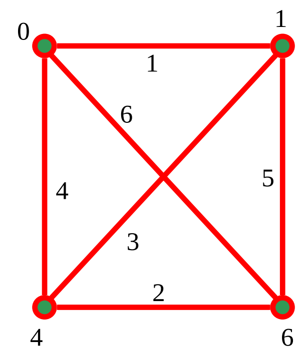
```
enum Country {Belgium,Denmark,France,Germany,
              Netherlands,Luxembourg};
enum Colors {blue,red,yellow,gray};
var Colors color[Country];
var int colorcount[Colors] in 0..card(Country);
maximize colorcount[yellow]
subject to {
     forall (i in Colors)
        colorcount[i] = sum(j in Country) (color[j] = i);
     color[France] <> color[Belgium];
     color[France] <> color[Luxembourg];
     color[France] <> color[Germany];
     color[Luxembourg] <> color[Germany];
     color[Luxembourg] <> color[Belgium];
     color[Belgium] <> color[Netherlands];
     color[Belgium] <> color[Germany];
     color[Germany] <> color[Netherlands];
     color[Germany] <> color[Denmark];
};
```

35

Search Example

Example: Graceful graphs

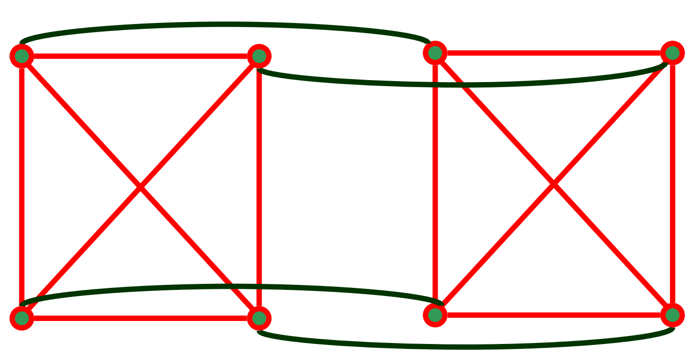
- **Given a graph G=(N,E)**
- **G is *graceful* if there is a labeling chosen from 0,1,2,…, |E| of the nodes and edges with the following properties:**
  - All node labels are unique
  - All edge labels are unique
  - If E=(a,b), then L(E)=Abs(L(a)-L(b))



36

Search Example

Is this graph graceful?



37

Search Example

Model for graceful

```
int numnodes = ...;
range Nodes 1..numnodes;
struct Edge {
   Nodes  i;
   Nodes  j;
};
{Edge} edges = ...;
int numedges = card(edges);
range Labels 0..numedges;
var Labels nl[Nodes];
var Labels el[edges];
solve {
   alldifferent (nl) ;
   alldifferent (el) ;
   forall (e in edges) {
      el[e] = abs(nl[e.i] - nl[e.j]);
      el[e] > 0;
   }
};
```

```
numnodes = 8;
edges = { <1,2>,  <5,6>,
          <1,3>,  <5,7>,
          <1,4>,  <5,8>,
          <2,3>,  <6,7>,
          <2,4>,  <6,8>,
          <3,4>,  <7,8>,
          <1,5>,
          <2,6>,
          <3,7>,
          <4,8> };
```

```
search {
   generate (nl);
   generate (el);
};
```

38

Search Example

What does `generate(nl)` do?

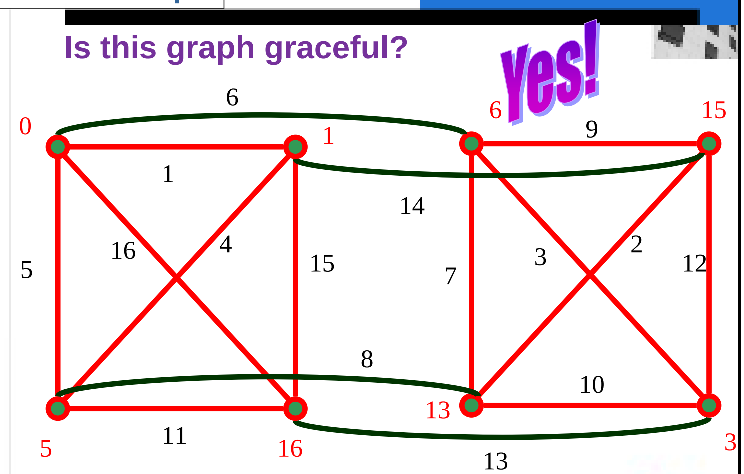
- **It generates all possible values for each element of the array**
- **Ordering of the variables**
  - Pick the variable with the smallest domain
- **Ordering of the values**
  - Try all values in the domain smallest to biggest

```
forall (i in Nodes : not bound(nl[i])
                  ordered by increasing dsize(nl[i])) {
   tryall (j in [dmin(nl[i])..dmax(nl[i])] :
               isInDomain(nl[i],j) )
       nl[i] = j onFailure nl[i] <> j;
   };
};
```

39

Search Example

Is this graph graceful?



40

Comparing CP and MP

Warehouse Assignment

- **Want to assign S stores to W warehouses. The problem is as follows:**
  - The cost of assigning store `s` to warehouse `w` is given by the array element **`supplyCost[s,w]`.**
  - Each warehouse `w` can have at most **`capacity[w]`** stores assigned to it.
  - There is a fixed cost **`fixed=30`** for opening up each warehouse.

41

Comparing CP and MP

Warehouse assignment: MIP

```
var int open[Warehouses] in 0..1;
var int supply[Stores,Warehouses] in 0..1;

minimize
      sum(w in Warehouses) fixed * open[w] +
      sum(w in Warehouses, s in Stores)
              supplyCost[s,w] * supply[s,w]
subject to {
      forall(s in Stores)
           sum(w in  Warehouses) supply[s,w] = 1;
      forall(w in Warehouses, s in Stores)
          supply[s,w] <= open[w];
      forall(w in Warehouses)
           sum(s in Stores) supply[s,w] <= capacity[w];
};
```

42

Comparing CP and MP

Warehouse assignment: CP

```
var int open[Warehouses] in 0..1;
var Warehouses supplier[Stores];
var int cost[Stores] in 0..maxCost;

minimize
      sum(s in Stores) cost[s] +
      sum(w in Warehouses) fixed * open[w]
subject to {
   forall(s in Stores)
      cost[s] = supplyCost[s,supplier[s]];
   forall(s in Stores )
      open[supplier[s]] = 1;
   forall(w in Warehouses)
      sum(s in Stores) (supplier[s] = w) <= capacity[w];
};

search {
   forall(s in Stores ordered by decreasing regretdmin(cost[s]))
      tryall(w in Warehouses ordered by increasing supplyCost[s,w])
         supplier[s] = w;
};
```

43

Comparing CP and MP

MIP versus CP formulation

- **Decision variables**
  - The constraint programming formulation has 2S+W decision variables.
  - The mixed integer formulation has SW+W decision variables.
  - The CP formulation has a decision variable over a finite set of values to represent the cost of shipping for store s.
  - The MIP formulation represents the cost of shipping for store s as an implied expression.

```
sum (w in Warehouses) supplyCost[s,w] * supply[s,w]
```

- **Expressions**
  - The CP formulation uses expressions of the form **`open[supplier[s]]`**, which uses a decision variable to index into another decision variable.
  - The CP formulation uses the expression **`(supplier[s] = w)`** that evaluates to a 0/1 value.

44

Comparing CP and MP

CP includes search!

```
search {
   forall(s in Stores ordered by decreasing regretdmin(cost[s]))
      tryall(w in Warehouses ordered by increasing supplyCost[s,w])
         supplier[s] = w;
};
```

- **`cost[s]` can only take on values from `supplyCost[s,w]` for the set of open warehouses `w`**
- **`regretdmin` = (second lowest value) - (lowest value)**
- **Pick the store with the largest regret, then pick the warehouse with the smallest cost**
- **Then open that warehouse**

45

Comparing CP and MP

But Which is BETTER????

- **It depends upon the data**
- **It depends on the search strategy**
- **It depends on the combinatorial nature of the problem**

- **For general applications, you need tools that allow you to try both methodologies!**

46

Comparing CP and MP

What is a solution?

- **Linear programs and integer programs always have objective functions**
- **A constraint satisfaction problem may simply be a feasibility problem**
  - It may have many possible solutions!
- **People in constraint programming say that they have a "solution" when people in mathematical programming would say they have a "feasible solution"**

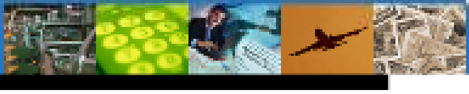
47

Comparing CP and MP

Vocabulary Differences

Mathematical Programming	Constraint Programming
Feasible Solution	Solution
Optimal Solution	Optimized Solution
Decision Variable	Constrained Variable
Fixed Variable	Bound Variable
Bound Strengthening	Domain Reduction (a superset)
Iterative Presolve	Constraint Propagation

48



Constraint Programming Successes

49

Optimization Successes

DaimlerChrysler

- Centralized Vehicle Scheduler: for vehicle production
- Results: Competitive advantage & savings
  - 10-20% improvement in purge rates
  - Increased production by 4,000 cars/year/plant
  - Estimated savings of $27 million annually

50

Optimization Successes

First Union

- Loan Arranger: Searches for loan that best meets each customer's requirements
- Results: Competitive advantage & savings
  - 4 x increase in monthly loan volume
  - 15% increase in average loan size
  - Reduced "time to funding" from 21 to 8 days
  - Reduced underwriting costs by 78%

51

Optimization Successes

SNCF Railways

- Rolling Stock Maintenance Operations
- Schedule Operations Efficiently
- Save 10% of 2,000 maintenance workers

52

Optimization Successes

Nissan (UK)

- **Challenge: Build 3rd car model with 2 existing production lines**
- **Results: Europe's already most efficient car production facility is even more productive**
  - No need to add any new production line and no significant investment needed
  - Production capacity increased by 30%
  - Schedule adherence rose from 3% to 90%

53

Constraint Programming

Applications

- **Scheduling**
- **Dispatching**
- **Configuration**
- **Enumeration**
- **Sequencing**

54

Summary

Conclusions

- **Optimization technologies have significantly improved over the past 15 years**
- **Multiple techniques**
  - Traditional Mathematical Programming
  - Newer Constraint Programming
- **An explosion of applications**

55