Progress in Linear and Integer Programming and Emergence of Constraint Programming

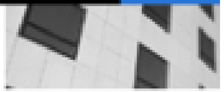
Dr. Irvin Lustig

Manager, Technical Services

Optimization Evangelist

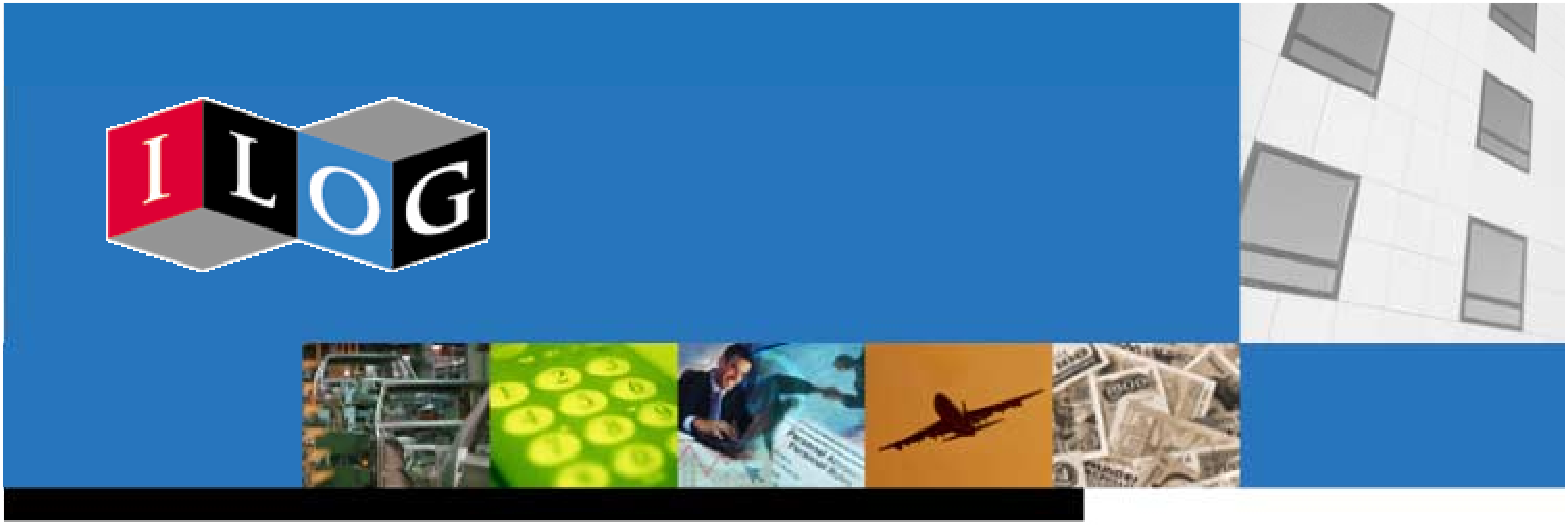
ILOG Direct

Agenda



Outline

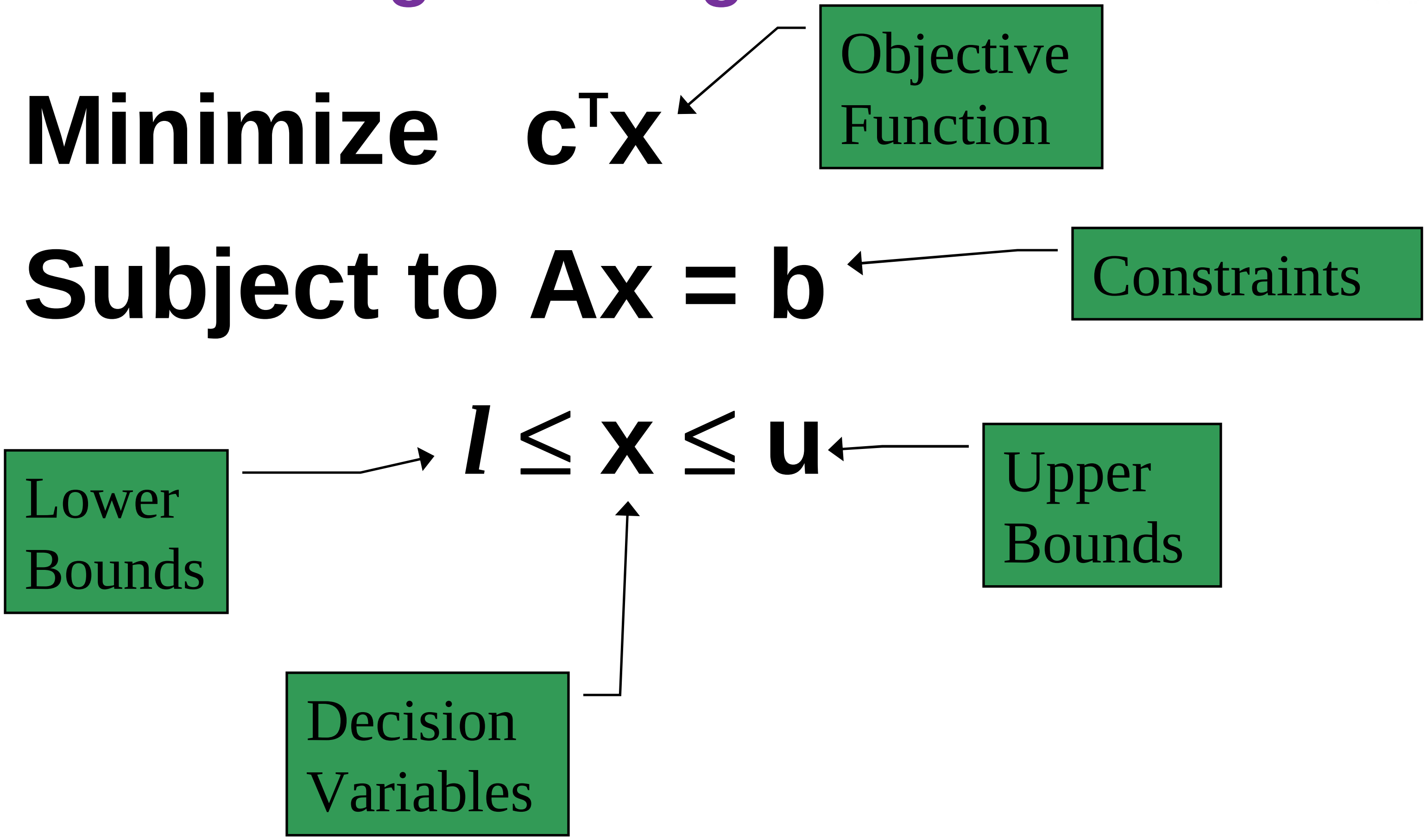
- □ **Mathematical Programming**
  - □ Improvements in Performance
- □ **Constraint Programming**
  - □ A Quick Tutorial
- □ **Constraint Programming Successes**



Mathematical Programming

Some material courtesy of Bob Bixby

Linear Programming



Minimize $c^T x$

Objective Function

Subject to $Ax = b$

Constraints

$$l \le x \le u$$

Lower Bounds

Upper Bounds

Decision Variables

Mathematical Programs

Linear Programming

$$\textbf{Minimize} \quad c^T x$$

$$\textbf{Subject to } Ax = b \qquad \text{(LP)}$$

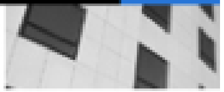
$$l \le x \le u$$

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

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

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)

Linear Programming

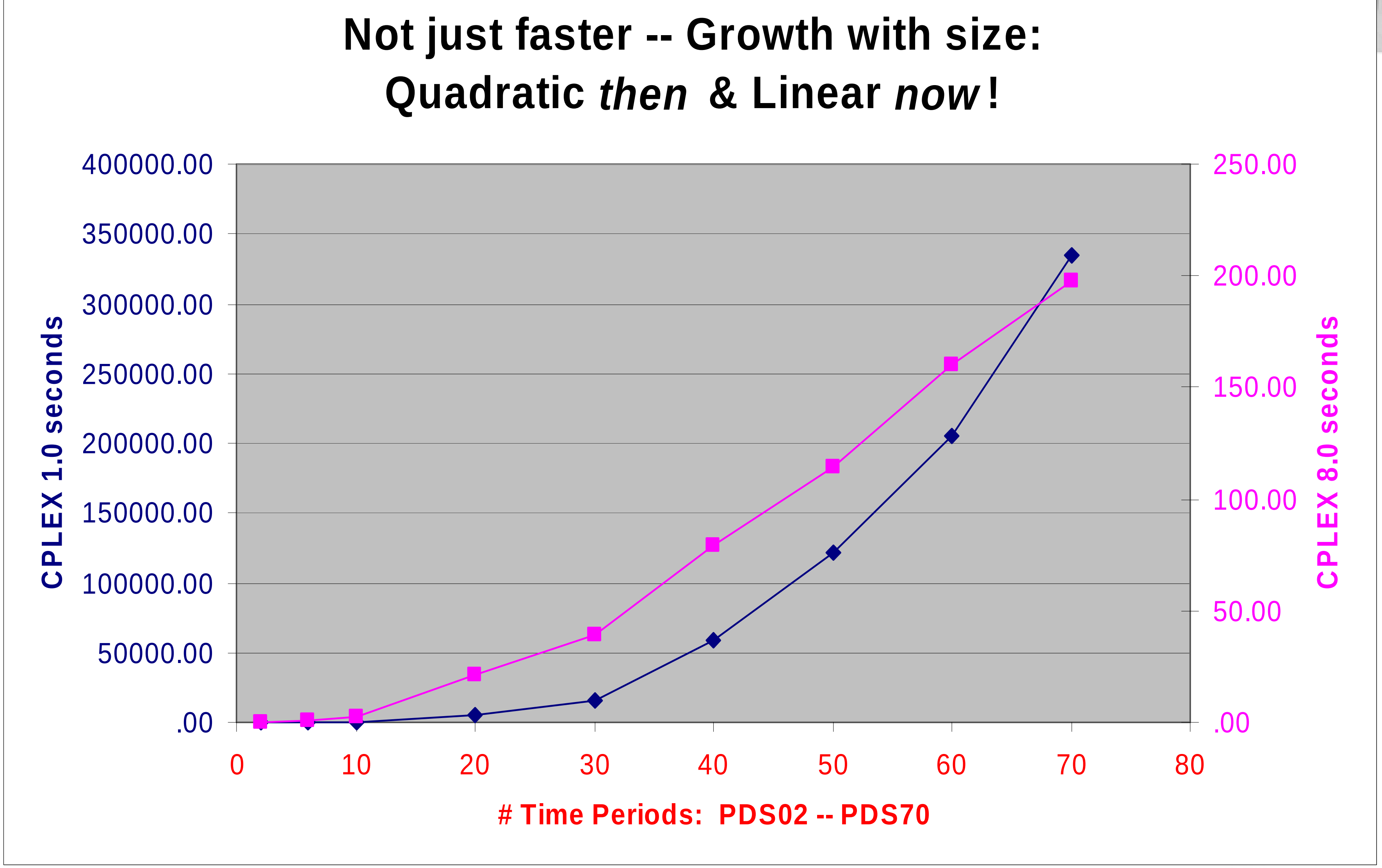


PDS Models

"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

		CPLEX1.0	CPLEX5.0	CPLEX8.0	SPEEDUP
MODEL	ROWS	1988	1997	2002	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	

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.

Progress: 1988 – Present

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

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

Mixed Integer Programming

Minimize $c^T x$

Subject to $Ax = b$ (MIP)

$l \le x \le u$

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
```

Computational History: 1950 –1998

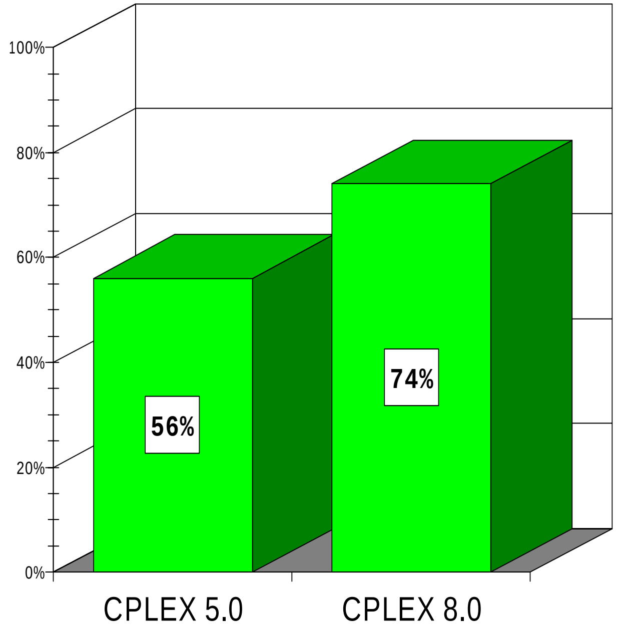
- **1954 Dantzig, Fulkerson, S. Johnson: 42 city TSP**
  - Solved to optimality using cutting planes and LP
- **1957 Gomory**
  - Cutting plane algorithm: A complete solution
- **1960 Land, Doig, 1965 Dakin**
  - Branch-and-bound (B&B)
- **1971 MPSX/370, Benichou et al.**
- **1972 UMPIRE, Forrest, Hirst, 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)

1998… A new generation of MIP codes

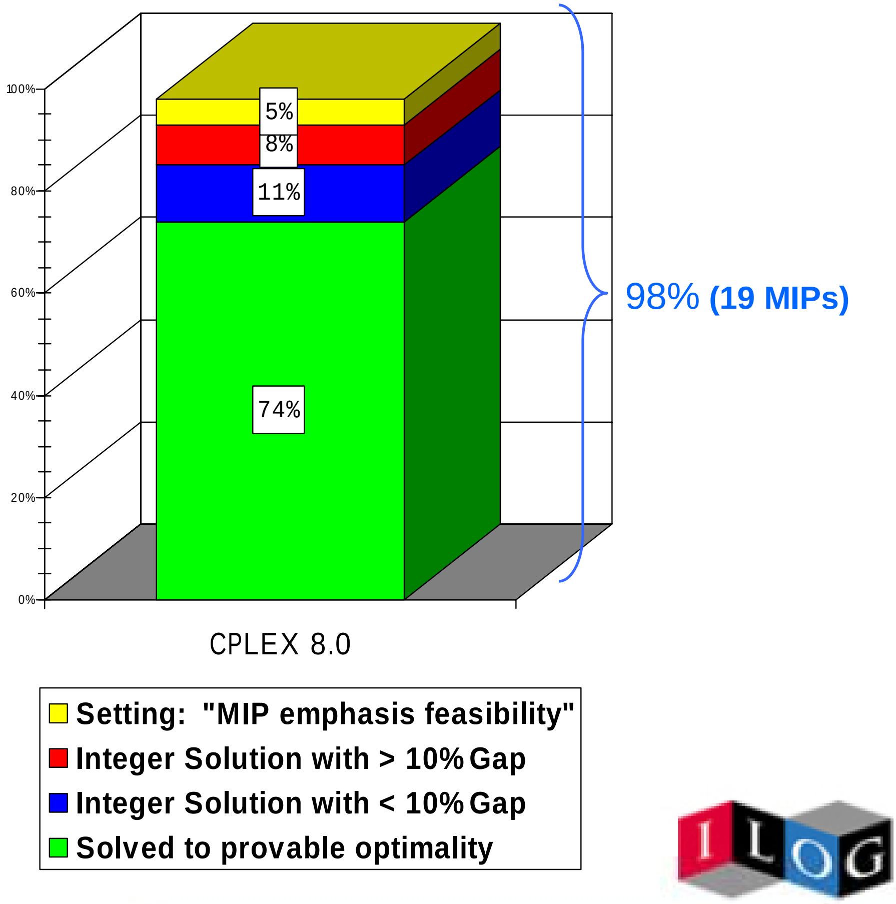
- **Linear programming**
  - Stable, robust dual simplex
- **Variable/node selection**
  - Influenced by traveling salesman problem
- **Primal heuristics**
  - 8 different tried at root
  - Retried based upon success
- **Node presolve**
  - Fast, incremental bound strengthening (very similar to Constraint Programming)
- **Presolve – numerous small ideas**
  - Probing in constraints:
  - $\sum xj \le (\sum uj)\, y, \quad y = 0/1$
  - ➔ $xj \le ujy$ (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

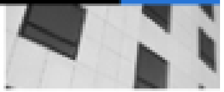
Computational Results I: 964 models (30 hour time limit)

Solving to Optimality



Finding Feasible Solutions

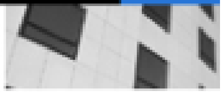




Computational Results II: 651 models (all solvable to optimality)

- **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**



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.

Constraint Programming

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

Mathematical Programming

Problem Types

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

A <span>**program**</span> **is a** ***problem***

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
  - ....

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

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

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

What is a potential representation?

- **Let x_1, x_2 ,..., 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***

  $$f\,(x_1, x_2 ,..., x_n) \in \{0, 1\}$$

  - The function may just be a table of values!

Constraint Satisfaction Problem

- **A constraint satisfaction problem is**

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

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

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

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

  $x_j \in D_j \quad (j = 1, 2, \ldots, n)$

  $f_k\ (x_1, x_2, \ldots, x_n) = 1 \quad (k=1,\ldots,m)$

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`**

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

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]]`

Map Coloring Example

DENMARK
NETHERLANDS
GERMANY
BELGIUM
LUX.
FRANCE

- ❑ **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***

Example

Constraint Programming Model

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

Data

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

Decision Variables

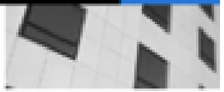
```
solve {
```

Find all Solutions

```
     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];
};
```

Constraints

Example



Constraint Satisfaction

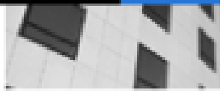
```
enum Country {Belgium,Denmark,France,Germany,
                  Netherlands,Luxembourg};
enum Colors {blue,red,yellow,gray};

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];
};
```

Example

Constraint Satisfaction

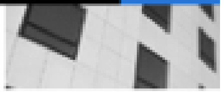
```
enum Country {Belgium,Denmark,France,Germany,
              Netherlands,Luxembourg};
enum Colors {blue,red,yellow,gray};
var Colors color[Country];


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];
};
```

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];
};
```



Problem Description

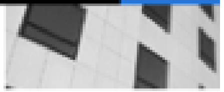
- **From Bradley, Hax, Magnanti, *Applied Mathematical Programming*, Chapter 9, Exercise 24**
  - Custom Pilot Chemical Company is a chemical manufacturer that produces batches of specialty chemicals to order. Principal equipment consists of eight interchangable reactor vessels, five interchangeable distillation columns, four large interchangeable centrifuges, and a network of switchable piping and storage tanks. Customer demand comes in the form of orders for batches of one or more specialty chemicals, normally to be delivered simultaneously for further use by the customer.
  - An order consists of a set of jobs. Each job has an optional precedence requirement, arrival week of the job, duration of the job in weeks, the week that the job is due, the number of reactors required, distillation columns required, and centrifuges required.
  - Find a schedule of the orders and jobs to minimize the completion time of all orders

Production Scheduling

Problem Data

						Resource requirements		
Order Number	*Job number*	*Precedence relations*	*Arrival Week*	*Duration in weeks*	*Week due*	*Reactors*	*Distillation columns*	*Centri-fuges*
AK14	1	None	15	4	22	5	3	2
	2	1	15	3	22	0	1	1
	3	None	15	3	22	2	0	2
AK15	1	None	16	3	23	1	1	1
	2	None	16	2	23	2	0	0
	3	1	16	2	23	2	2	0
AK16	1	None	17	5	23	2	1	1
	2	None	17	1	23	1	3	0

Production Scheduling



Data input

```
struct JobIndex {
   string ordernumber;
   int jobnum;
};
struct JobInfo {
   int jobprec;
   int arrival;
   int duration;
   int weekdue;
   int reactors;
   int columns;
   int centrifuges;
};
struct JobData {
   JobIndex ind;
   JobInfo  info;
};

setof(JobData) jobs = ...;
```

```
jobs =
{
< < "AK14", 1 >, < 0, 15, 4, 22, 5, 3, 2 > >,
< < "AK14", 2 >, < 1, 15, 3, 22, 0, 1, 1 > >,
< < "AK14", 3 >, < 0, 15, 3, 22, 2, 0, 2 > >,
< < "AK15", 1 >, < 0, 16, 3, 23, 1, 1, 1 > >,
< < "AK15", 2 >, < 0, 16, 2, 23, 2, 0, 0 > >,
< < "AK15", 3 >, < 1, 16, 2, 23, 2, 2, 0 > >,
< < "AK16", 1 >, < 0, 17, 5, 23, 2, 1, 1 > >,
< < "AK16", 2 >, < 0, 17, 1, 23, 1, 3, 0 > >
};
```

Data organization

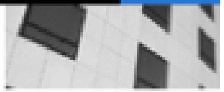
```
setof(JobIndex) joblist = { i | <i,j> in jobs };

assert ( card(joblist) = card(jobs) );

JobInfo datarray[joblist];
initialize {
   forall (j in jobs)
      datarray[j.ind] = j.info;
};
```

```
datarray[<"AK14", 1>] = < 0, 15, 4, 22, 5, 3, 2 >
datarray[<"AK14", 2>] = < 1, 15, 3, 22, 0, 1, 1 >
datarray[<"AK14", 3>] = < 0, 15, 3, 22, 2, 0, 2 >
datarray[<"AK15", 1>] = < 0, 16, 3, 23, 1, 1, 1 >
datarray[<"AK15", 2>] = < 0, 16, 2, 23, 2, 0, 0 >
datarray[<"AK15", 3>] = < 1, 16, 2, 23, 2, 2, 0 >
datarray[<"AK16", 1>] = < 0, 17, 5, 23, 2, 1, 1 >
datarray[<"AK16", 2>] = < 0, 17, 1, 23, 1, 3, 0 >
```

```
int reactors = ...;
int columns = ...;
int centrifuges = ...;
```

```
reactors = 8;
columns = 5;
centrifuges = 4;
```

Production Scheduling

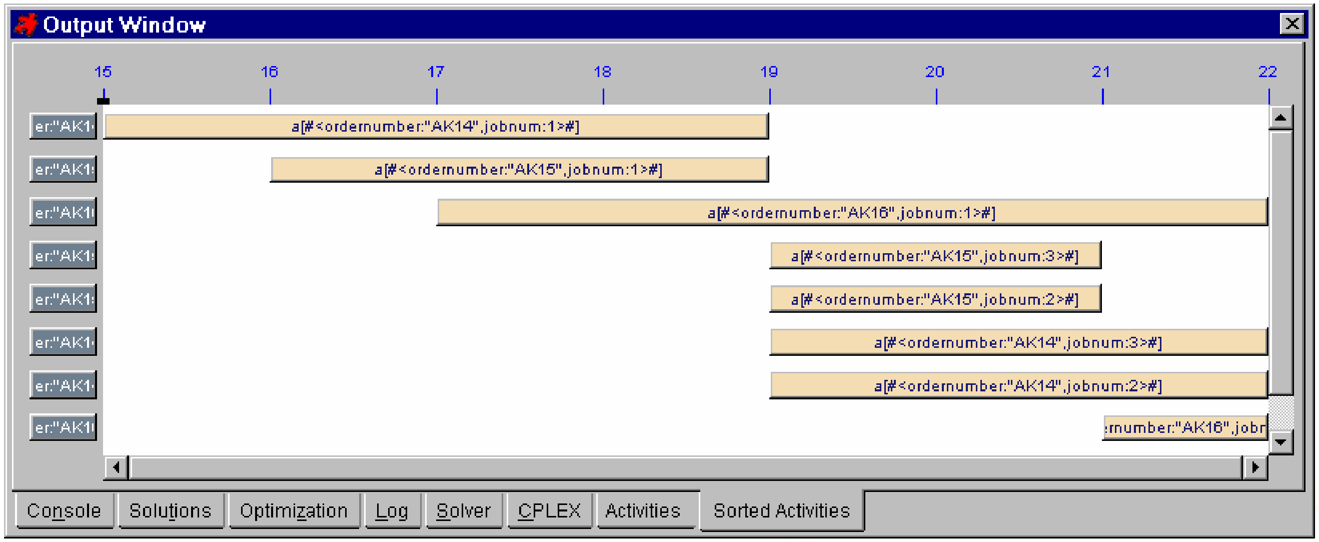


Model

```
scheduleOrigin = min(j in jobs) j.info.arrival;
scheduleHorizon = max(j in jobs) j.info.weekdue;

Activity makespan(0);
Activity a[j in joblist](datarray[j].duration);
DiscreteResource Reactors(reactors);
DiscreteResource Columns(columns);
DiscreteResource Centrifuges(centrifuges);

minimize makespan.end
subject to
{
   forall (j in joblist) {
      a[j] precedes makespan;
      if (datarray[j].jobprec > 0) then
         a[<j.ordernumber,datarray[j].jobprec>] precedes a[j]
      endif;
      a[j] requires(datarray[j].reactors) Reactors;
      a[j] requires(datarray[j].columns) Columns;
      a[j] requires(datarray[j].centrifuges) Centrifuges;
      a[j].start >= datarray[j].arrival;
      a[j].end <= datarray[j].weekdue;
   };
};
```

Solution for activities

```
Optimal Solution with Objective Value: 22
makespan = [22 -- 0 --> 22]
a[#<ordernumber:"AK14",jobnum:1>#] = [15 -- 4 --> 19]
a[#<ordernumber:"AK14",jobnum:2>#] = [19 -- 3 --> 22]
a[#<ordernumber:"AK14",jobnum:3>#] = [19 -- 3 --> 22]
a[#<ordernumber:"AK15",jobnum:1>#] = [16 -- 3 --> 19]
a[#<ordernumber:"AK15",jobnum:2>#] = [19 -- 2 --> 21]
a[#<ordernumber:"AK15",jobnum:3>#] = [19 -- 2 --> 21]
a[#<ordernumber:"AK16",jobnum:1>#] = [17 -- 5 --> 22]
a[#<ordernumber:"AK16",jobnum:2>#] = [21 -- 1 --> 22]
```

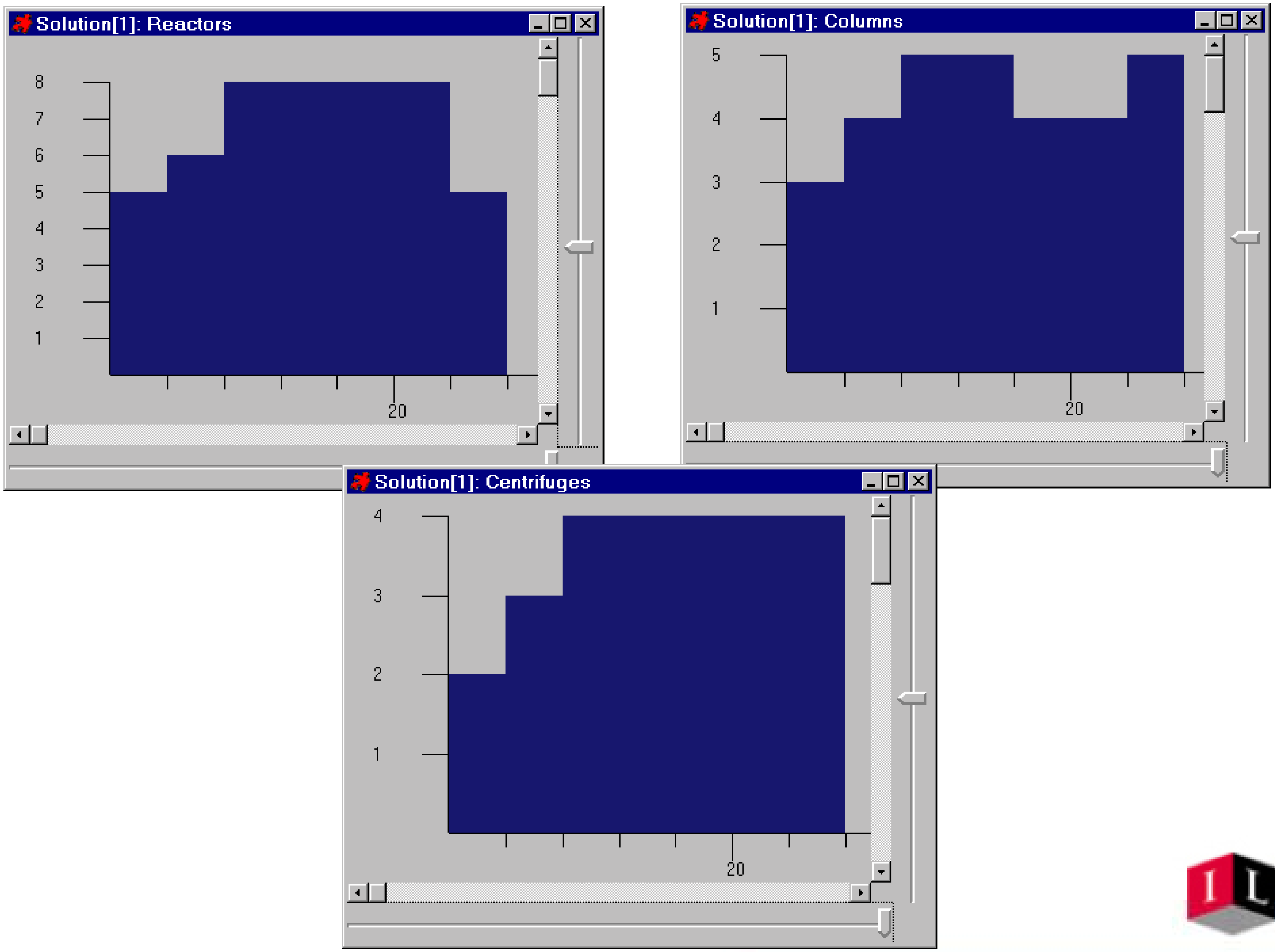

Output Window
15
16
17
18
19
20
21
22
er:"AK1
er:"AK1
er:"AK1
er:"AK1
er:"AK1
er:"AK1
er:"AK1
er:"AK1
a[#<ordernumber:"AK14",jobnum:1>#]
a[#<ordernumber:"AK15",jobnum:1>#]
a[#<ordernumber:"AK16",jobnum:1>#]
a[#<ordernumber:"AK15",jobnum:3>#]
a[#<ordernumber:"AK15",jobnum:2>#]
a[#<ordernumber:"AK14",jobnum:3>#]
a[#<ordernumber:"AK14",jobnum:2>#]
rnumber:"AK16",jobr
Console
Solutions
Optimization
Log
Solver
CPLEX
Activities
Sorted Activities

Production Scheduling

Resource allocation (text)

```
Reactors = Discrete Resource
  required by a[#<ordernumber:"AK16",jobnum:2>#] over [21,22]  in capacity 1
  required by a[#<ordernumber:"AK16",jobnum:1>#] over [17,22]  in capacity 2
  required by a[#<ordernumber:"AK15",jobnum:3>#] over [19,21]  in capacity 2
  required by a[#<ordernumber:"AK15",jobnum:2>#] over [19,21]  in capacity 2
  required by a[#<ordernumber:"AK15",jobnum:1>#] over [16,19]  in capacity 1
  required by a[#<ordernumber:"AK14",jobnum:3>#] over [19,22]  in capacity 2
  required by a[#<ordernumber:"AK14",jobnum:1>#] over [15,19]  in capacity 5

Columns = Discrete Resource
  required by a[#<ordernumber:"AK16",jobnum:2>#] over [21,22]  in capacity 3
  required by a[#<ordernumber:"AK16",jobnum:1>#] over [17,22]  in capacity 1
  required by a[#<ordernumber:"AK15",jobnum:3>#] over [19,21]  in capacity 2
  required by a[#<ordernumber:"AK15",jobnum:1>#] over [16,19]  in capacity 1
  required by a[#<ordernumber:"AK14",jobnum:2>#] over [19,22]  in capacity 1
  required by a[#<ordernumber:"AK14",jobnum:1>#] over [15,19]  in capacity 3

Centrifuges = Discrete Resource
  required by a[#<ordernumber:"AK16",jobnum:1>#] over [17,22]  in capacity 1
  required by a[#<ordernumber:"AK15",jobnum:1>#] over [16,19]  in capacity 1
  required by a[#<ordernumber:"AK14",jobnum:3>#] over [19,22]  in capacity 2
  required by a[#<ordernumber:"AK14",jobnum:2>#] over [19,22]  in capacity 1
  required by a[#<ordernumber:"AK14",jobnum:1>#] over [15,19]  in capacity 2
```

Resource allocation (graphs)



Comparing CP and MP

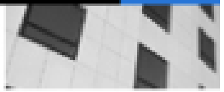
Which is BETTER????

- 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!

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"**

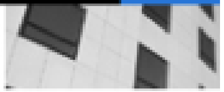


Vocabulary Differences

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

Constraint Programming Successes

Optimization Successes

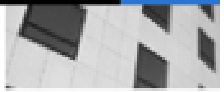


DaimlerChrysler

- ❑ **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

DAIMLERCHRYSLER

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%

Optimization Successes

SNCF Railways

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

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%

Applications

- Scheduling
- Dispatching
- Configuration
- Enumeration
- Sequencing

Conclusions

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