SIMPL: An Integrated Solver for Optimization Problems

Ionuț D. Aron, IBM T. J. Watson Research Center
John Hooker, Carnegie Mellon University
Tallys H. Yunes, University of Miami

March 2007

Outline

- **Intro and Motivation**
- **SIMPL: Our General Purpose System**
- **Examples**
  - Production Planning
  - Product Configuration
  - Machine Scheduling
- **Future Work**

Why Integrate?

- **User convenience:** several methods and their combinations available in one package.

- **Simpler models:** constraint-programming style modeling adapted to general optimization.
  - Model communicates problem structure to solver.
  - It uses *metaconstraints* = generalization of global constraints in constraint programming

- **Better performance:** combine the complementary strengths of different methods.
  - Solver chooses best mix of relaxation and propagation methods, based on metaconstraints.

SIMPL Objectives

- **High-level** modeling language
  - **Concise** and **easily understandable** models
  - **Natural** specification of integrated models
  - Allow the modeler to **reveal problem structure** to the solver
- **Micro-level** integration
  - Integrated methods are more effective when the underlying technologies interact at **micro level** throughout the search process.

Models vs. problems

- **Models should be distinguished from problem instances.**
  - MIPLIB is a collection of **models**, not problem instances.
    - Unfortunately, the problem original description is often unknown.
    - So it's hard to write a better model.
- **A model should reveal problem structure.**
  - Much as a **scientific model**.
  - MIPLIB instances are better seen as **formulations** than models.
    - They are not **explanatory**.

Algorithmic Idea behind SIMPL

- CP and IP are special cases of a general method, not separate methods to be combined
- Common solution strategy:

  Search – Infer – Relax
- Search = enumeration of problem restrictions

For example: Classical Solution Methods

- **Branch and Cut (IP)**
  - **Inference**: preprocessing at root node, cutting planes
  - **Relaxation**: dropping integrality constraints
  - **Search**: branching on integrality constraints
- **Typical CP solver**
  - **Inference**: domain reduction
  - **Relaxation**: collecting reduced domains into constraint store
  - **Search**: domain splitting
- **Benders decomposition**
  - **Inference**: Benders cuts
  - **Relaxation**: master problem
  - **Search**: creating sub-problems by fixing "hard" variables
- **(Meta) Heuristics**

Constraints, Constraints…

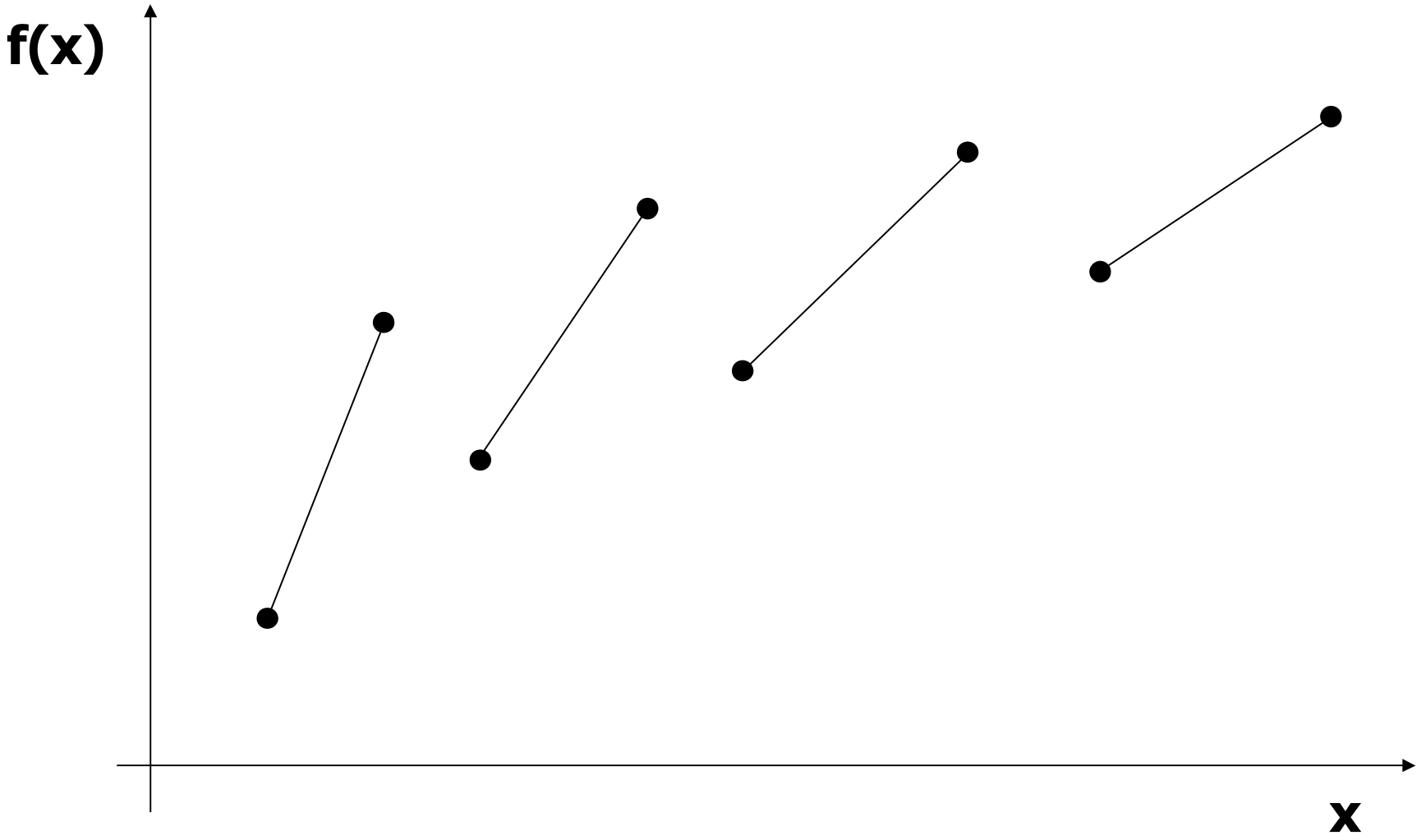
Constraint-oriented modeling and solution

- **Infer:** constraints **drive the inference**
  - Each constraint has a **filtering/inference module**
  - This module provides new constraints to **tighten the relaxation**
- **Relax:** constraints **determine the relaxations**
  - Each constraint has a **relaxation module**
  - This module **reformulates the constraint** and sends it to the appropriate relaxation (LP, CP, MIP etc)
- **Search:** constraints **direct the search**
  - Each constraint has a **branching module**
  - This module **creates new problem restrictions**

Example 1: Production Planning

- Manufacture several products to maximize net income
- Each product has several production modes (small scale, medium scale, etc.)
- Only certain ranges of production quantities are possible (gaps in the domain)
- Net income function f(x) is semi-continuous piecewise linear (non-convex and non-concave)

Production Planning: f(x)



Production Planning: MIP Model

- x_i = quantity of product i
- y_{ik} = 1 if product i is manufactured in mode k
- λ_{ik}, μ_{ik} = weights for mode k (convex comb.)

$$\max \sum_{ik} (\lambda_{ik} c_{ik} + \mu_{ik} d_{ik})$$

$$\sum_i x_i \le C$$

$$x_i = \sum_k (\lambda_{ik} L_{ik} + \mu_{ik} U_{ik}), \forall i$$

$$\sum_k (\lambda_{ik} + \mu_{ik}) = 1, \forall i$$

$$0 \le \lambda_{ik} \le y_{ik}, \forall i,k$$

$$0 \le \mu_{ik} \le y_{ik}, \forall i,k$$

$$\sum_k y_{ik} = 1, \forall i$$

$$y_{ik} \in \{0,1\}, \forall i,k$$

Production Planning: Integrated

- x_i = quantity of product i
- y_i = net income from product i

$$\max \sum_i y_i$$

$$\sum_i x_i \le C$$

$$\text{piecewise}(x_i, y_i, L_i, U_i, c_i, d_i), \forall i$$

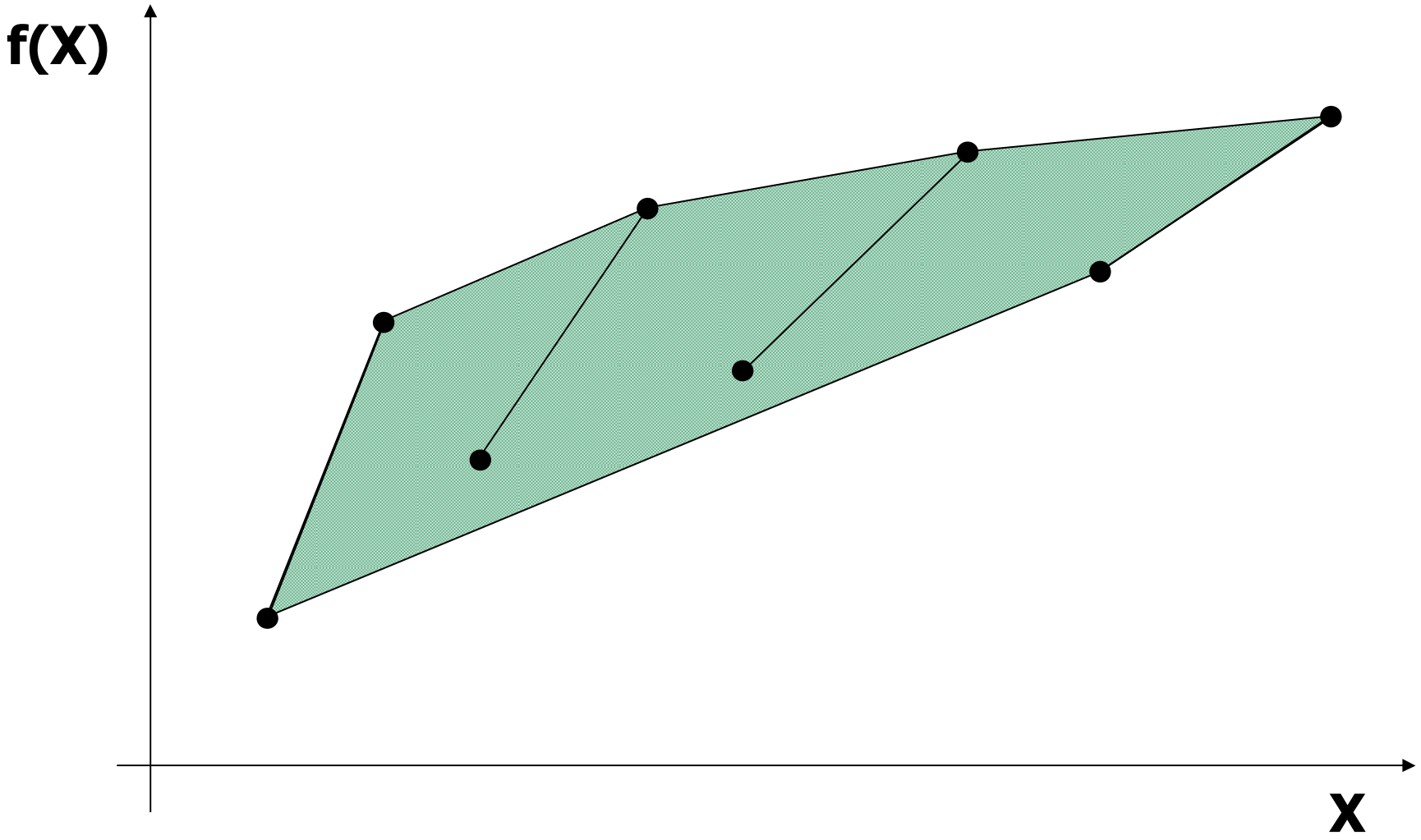
$$x_i \in D_i$$

Production Planning: Integrated

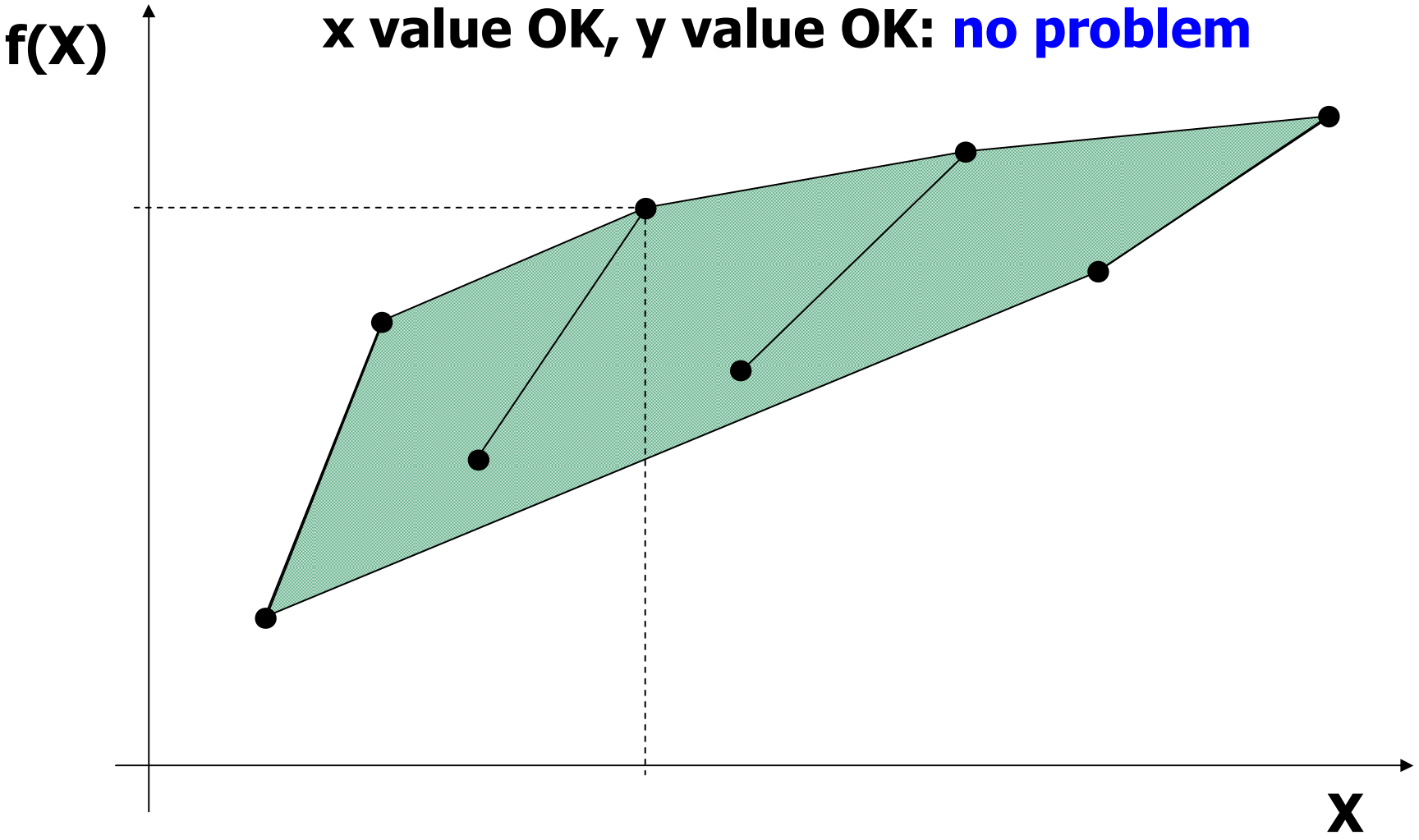
SIMPL Model

```
OBJECTIVE
  maximize sum i of u[i]
CONSTRAINTS
  capacity means {
    sum i of x[i] <= C
    relaxation = { lp, cp } }
  piecewisectr means {
    piecewise(x[i],u[i],L[i],U[i],c[i],d[i]) forall i
    relaxation = { lp, cp } }
SEARCH
  type = { bb:bestdive }
  branching = { piecewisectr:most }
```

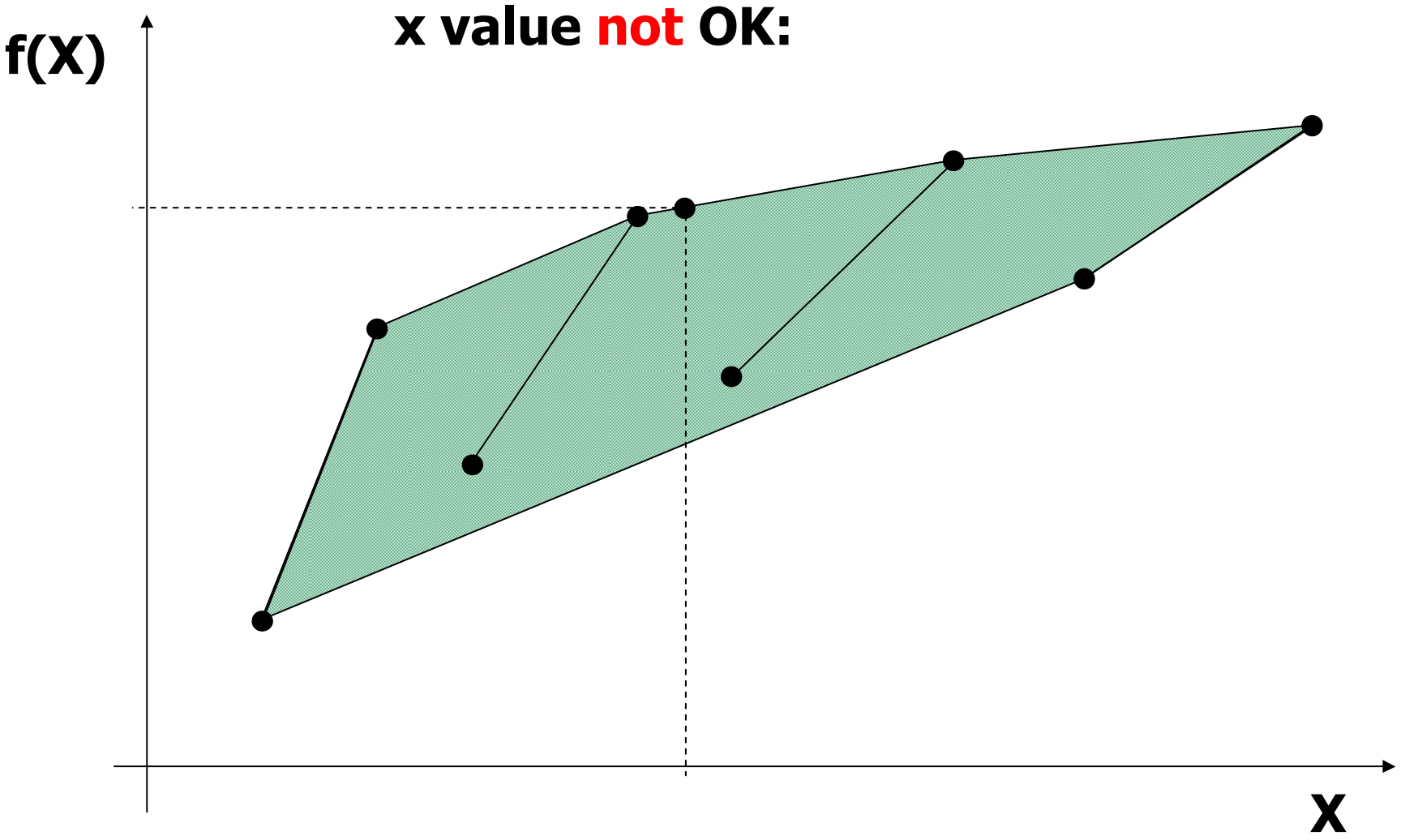
Relaxation of piecewise constraint



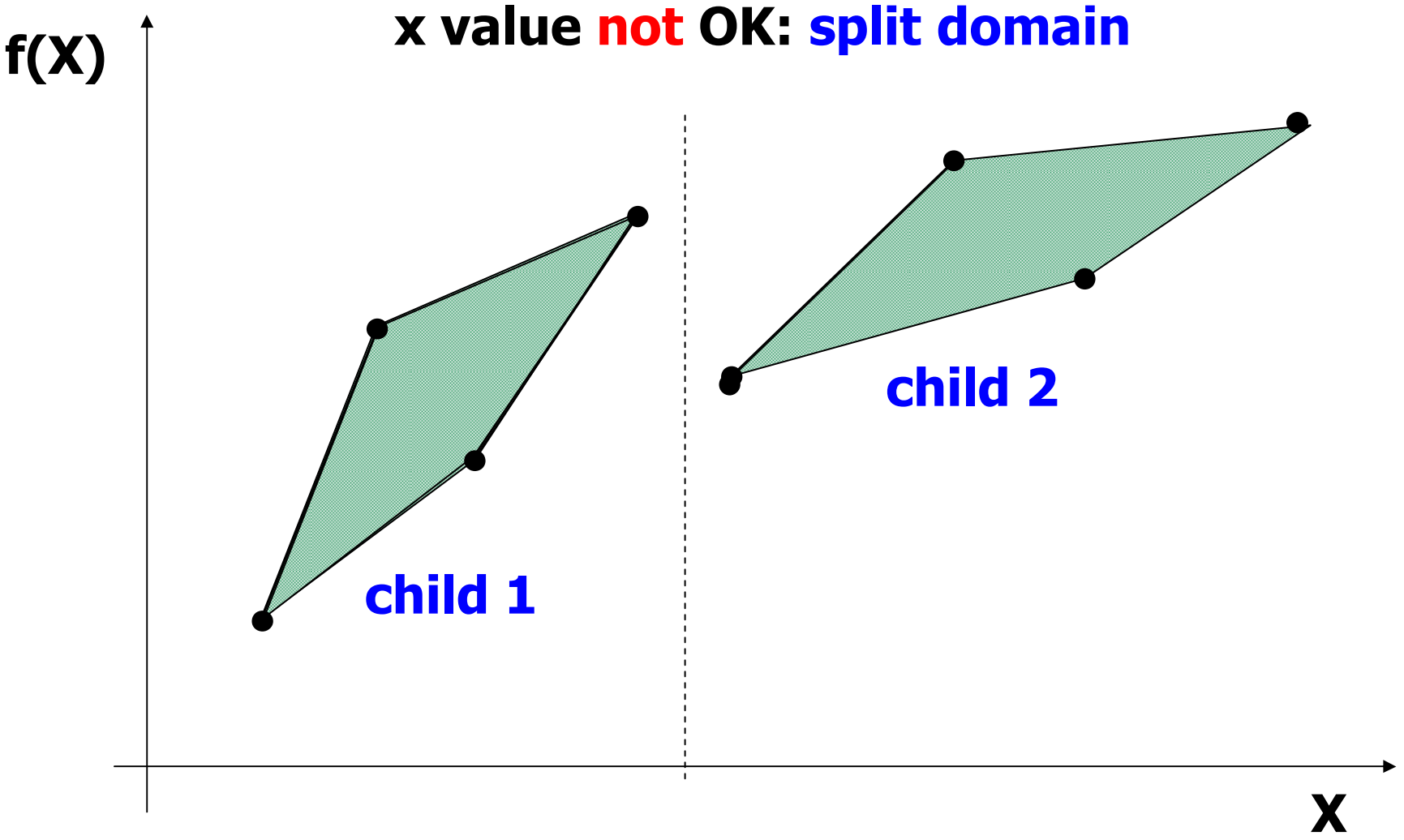
Branching on piecewise constraint



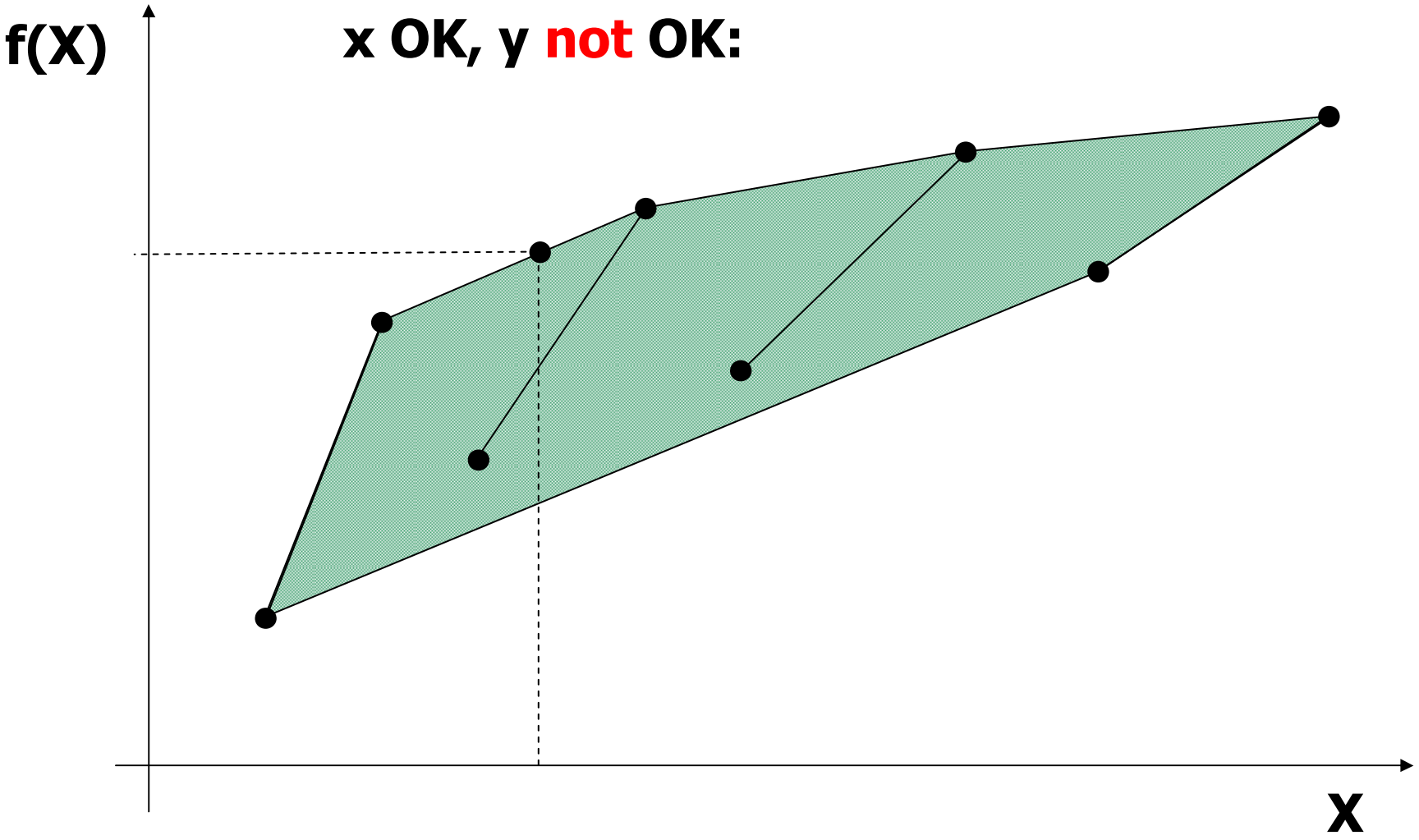
Branching on piecewise constraint



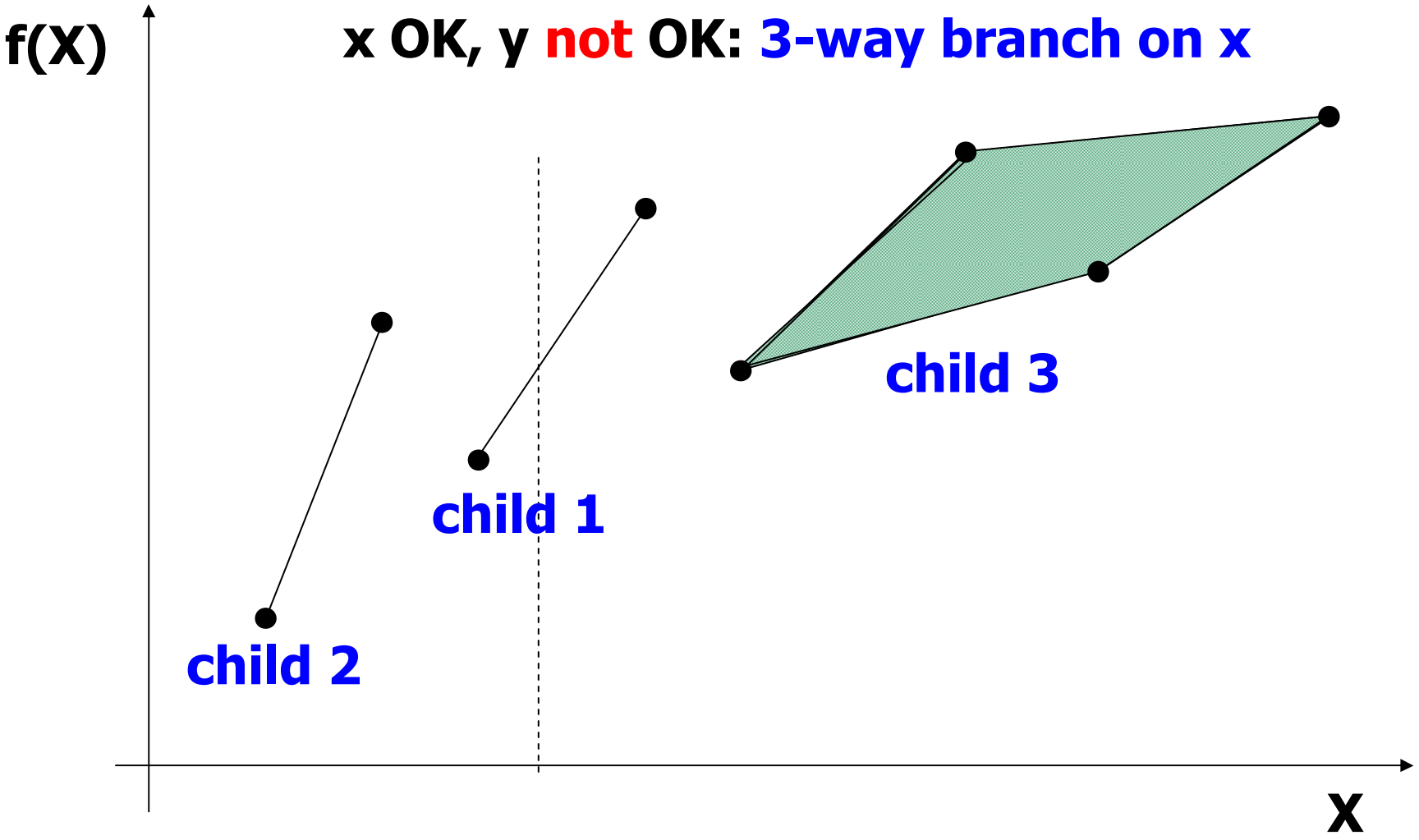
Branching on piecewise constraint



Branching on piecewise constraint



Branching on piecewise constraint



Computational Results

- **SIMPL**
  - LP solver = CPLEX 9.0.
  - CP solver = Eclipse 5.8.
- **MILP**
  - CPLEX 9.0
- **Hardware**
  - Pentium 4, 3.7 GHz, 4GB RAM.
  - Gentoo Linux kernel 2.6.12.5

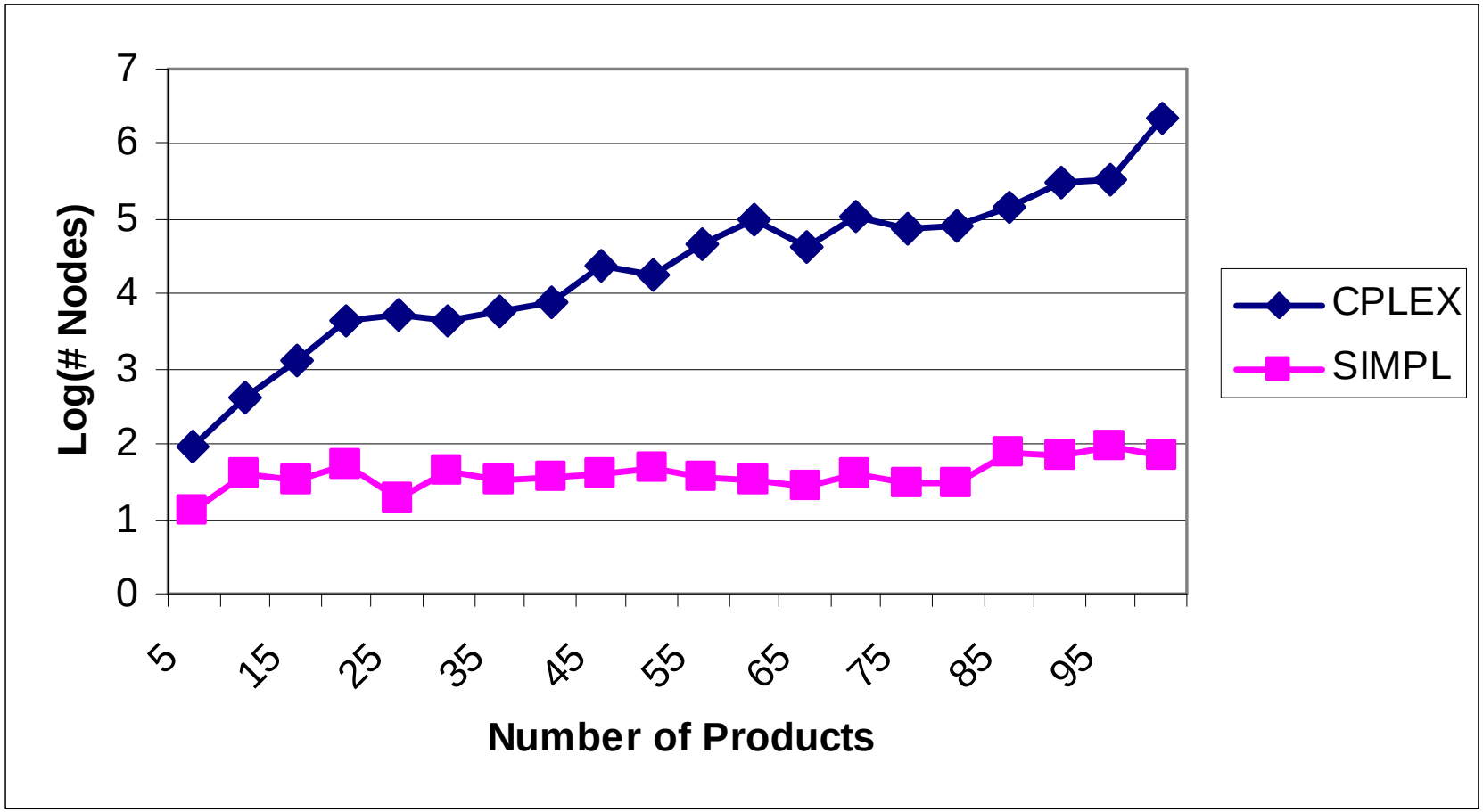
Computational Results

- **The aim is not to show that integrated methods can be faster.**
  - This is shown in a growing literature.
- **The aim is to show that SIMPL can achieve same or better speedups than hand-crafted integrated methods in the literature.**
  - With **no coding** and a **simple model** (simpler than MILP).

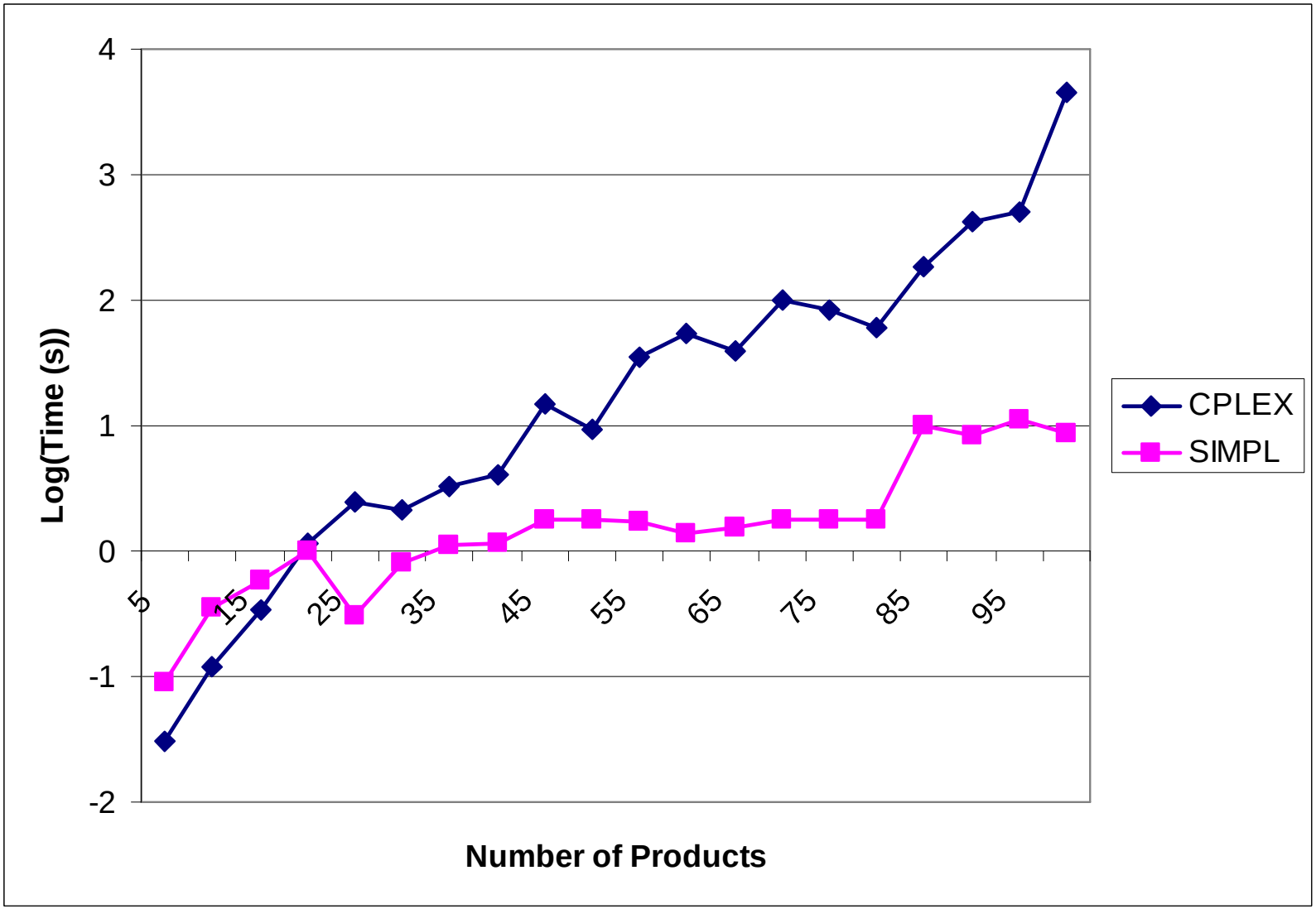
Computational Results: Search Nodes

All products have the same cost structure

Symmetry breaking: $x_i \leq x_{i+1}$



Computational Results: Time (s)



Computational results

- **These speedups are better than those reported in the literature.**
  - Refalo 1999
  - Ottosson, Thorsteinsson and Hooker 1999, 2002.

Example 2: Product Configuration

- A product (e.g. computer) is made up of several components (e.g. memory, cpu, etc.)
- Components come in different types

- Type j of component i consumes/produces A_{ijk} units of resource k
- There are lower and upper bounds on resource consumption/production

- c_k = unit cost of resource k
- Minimize total cost

Product Configuration: MIP Model

- v_k = total consumption/production of resource k
- x_{ij} = whether or not type j is chosen for component i
- q_{ij} = units of type j of component i

$$\min \sum_k c_k v_k$$

$$v_k = \sum_{i,j} A_{ijk} q_{ij}, \ \forall k$$

$$q_{ij} \le M_i x_{ij}, \ \forall i, j$$

$$\sum_j x_{ij} = 1, \ \forall i$$

$$L_k \le v_k \le U_k, \ \forall k \ \text{ and } \ x_{ij} \in \{0,1\}, \forall i, j$$

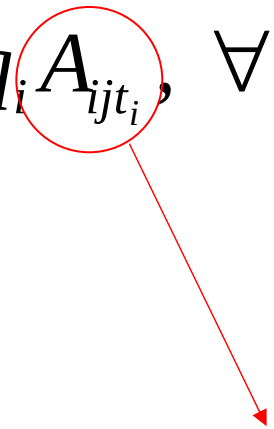
Product Configuration: Integrated

- v_k = total consumption/production of resource k
- q_i = quantity of component i
- t_i = type of component i

$$\min \sum_k c_k v_k$$

$$v_k = \sum_i q_i A_{ijt_i}, \quad \forall k$$

$$v_k \in D_k$$



Modeled internally with an element constraint

Product Configuration: Integrated

SIMPL Model

```
OBJECTIVE
  minimize sum j of c[j]*v[j]
CONSTRAINTS
  usage means {
    v[j] = sum i of q[i]*a[i][j][t[i]] forall j
    relaxation = { lp, cp }
    inference = { knapsack } }
  quantities means {
    q[1] >= 1 => q[2] = 0
    relaxation = { lp, cp } }
  types means {
    t[1] = 1 => t[2] in {1,2}
    t[3] = 1 => (t[4] in {1,3} and t[5] in {1,3,4,6} and t[6] = 3)
    relaxation = { lp, cp } }
SEARCH
  type = { bb:bestdive }
  branching = { quantities, t:most, q:least:triple, types:most }
  inference = { q:redcost }
```

Computational Results

	CPLEX		SIMPL	
Instance	Nodes	Time (s)	Nodes	Time (s)
1	1	0.06	61	6.64
2	1	0.08	34	3.08
3	184	0.79	164	20.42
4	1	0.04	27	2.53
5	723	4.21	30	2.91
6	1	0.05	30	1.99
7	111	0.59	32	2.97
8	20	0.19	29	2.94
9	20	0.17	18	0.97
10	1	0.03	32	2.60

Computational Results

- **First application of integrated method was much faster than CPLEX at that time.**
  - CPLEX 7.0 stopped after 100,000 nodes on 7 of the 10 instances.
  - It solved the remaining 3 with 77,000 nodes.
  - Thorsteinsson and Ottosson (2001) used the integrated method to solve these instances in about the same time as SIMPL.

Example 3: Machine Scheduling

- Given n tasks and m machines (disjunctive)
- It costs c_{ij} to process task i on machine j
- Processing time of task i on machine j is p_{ij}
- Task i has release date r_i and due date d_i
- **Goal:** schedule all tasks and minimize total cost

  - Use Hybrid IP/CP **Benders Decomposition** approach

Benders Decomposition

- **Master Problem**
  - Assign tasks to machines at minimum cost
  - Regardless of release dates and due dates
  - $x_{ij} = 1$ if task i assigned to machine j

- **Subproblems**
  - Jobs in I_j assigned to machine j.
  - Try to find feasible schedule (with given tasks)
  - If infeasible for mach j, generate the Benders cut:

$$\sum_{i \in I_j} x_{ij} \leq \left| I_j \right| - 1$$

Machine Scheduling in SIMPL

```
OBJECTIVE
   min sum i,j of c[i][j] * x[i][j];
CONSTRAINTS
  assign means {
    sum i of x[i][j] = 1 forall j;
    relaxation = { ip:master } }
  xy means {
    x[i][j] = 1 <=> y[j] = i forall i, j;
    relaxation = { cp } }
  tbounds means {
    r[j] <= t[j] forall j;
    t[j] <= d[j] - p[y[j]][j] forall j;
    relaxation = { ip:master, cp } }
  machinecap means {
    cumulative({ t[j], p[i][j], 1 } forall j |
        x[i][j] = 1, 1) forall i;
    relaxation = { cp:subproblem, ip:master }
    inference = { feasibility } }
SEARCH
  type = { benders }
```

Machine Scheduling

- **Results compared with:**
  - CPLEX 9.0 (ILOG Scheduler is generally slower).
  - Integrated CP/IP implementation of Jain and Grossmann 2001.

Computational Results: Time (s)

n, m	p_{ij}	Best Comm	J&G	SIMPL
3, 2	long			
	short			
7, 3	long	0.10	0.52	0.08
	short	0.27	0.02	0.00
12, 3	long			
	short			
15, 5	long	91.59	2.25	0.68
	short	5.58	0.04	0.04
20, 5	long			
	short			

Never more than 31 iterations and 60 cuts in Benders approach

Machine Scheduling in SIMPL

- **Extensions:**
  - **Cumulative** (resource-constrained) scheduling in subproblem.
  - **Other objective functions** use more interesting logic-based Benders cuts and relaxations.
    - **Makespan (JNH 2004)**
    - **Number of late jobs (JNH 2005)**
    - **Total tardiness (JNH 2005, 2007)**
  - **Equally good speedups** in most cases.

Future Work

- **Increase SIMPL's functionality: add new constraints, solvers, search mechanisms, etc.**
  - **Easy** to add new constraints.
- **Find valid propagation methods and relaxations for more constraints.**
- **Release a beta version.**
- **Allow users to contribute their own constraints.**
  - Create a **modeling "Wikipedia"?**