Integrating CP and IP

J. N. Hooker

Carnegie Mellon University

Conference on Constraint Programming and Integer Programming
Dagstuhl, January 2000

Premise

- To a large extent, CP and IP are special cases of a single approach to problem solving.
  - They view it from different angles and emphasize different things.
  - Both exploit a fundamental search/inference duality.
  - They are to a large extent complementary.
  - Proper integration allows us to realize the full potential of both fields.

Objective

- Provide a rationale for integration of CP and IP, rather than
  - survey computational results (they are out there), or
  - propose a specific scheme for integration (although I can’t resist giving you a peek at one).

Characteristics of IP

- Relies on *continuous relaxation* (linear, Lagrangean, etc.; occasional discrete relaxation in special cases)
- Uses advanced methods for *strengthening relaxations* (cutting planes, reformulation).
- Gives much attention to highly-structured *problems*.
- Prefers *declarative* models.
- Writes models in a *restricted* language to make continuous relaxation easy.

• Historically, inference (in the form of cutting planes) is done primarily for the purpose of strengthening relaxations.

  • Moved away from implicit enumeration of the early days, perhaps due to the type of problems solved.

  • A key advance: cutting planes and separation algorithms (beginning with Chvátal-Gomory cuts).

Characteristics of CP

- Relies on *inference* (usually domain reduction, constraint propagation)
- Uses advanced methods for *processing global constraints* (domain reduction).
- Gives much attention to highly-structured *subsets of constraints* (represented by global constraints).
- Prefers to embed constraints in a *programming language*.
- Writes models in a *flexible* language that allows definition of new global constraints.

- The historical project of integrating constraints into a programming language permeates thinking in the field today.
    - It was much of the motivation for logic programming and constraint logic programming.
    - A key advance: incorporation of constraints into logic programming by interpreting unification as constraint solving.

Strengths of IP

- Good at proving optimality (due to good relaxations).
- Can deal with constraints that contain many variables (and are therefore hard to propagate--like pushing a string).
    - The objective function often contains many variables. (CP has more success with bottleneck objectives than cost objectives.) The presence of the objective function influenced the development of IP.
    - Much of CP is inspired by problems with binary constraints (2 variables per constraint), where restricting the domain of one variable tends to have a big effect on the domain of the other.

Strengths of CP

- Uses a flexible modeling language that allows the user to exploit structure.
- Gets better when constraints are added.
  - Addition of constraints in IP can make known cuts ineffective. They cut away regions that are no longer feasible in the continuous relaxation. ("Lifting" cuts can sometimes address this.)
  - IP never developed a theory (parallel to consistency) that explains how constraints can reduce a search apart from their role in strengthening relaxations.

Consistency

- A constraint set is *consistent* when any partial assignment that violates no constraints is consistent with some feasible solution.
  - That is, one can extend any partial assignment $(y_1,\ldots,y_k)=(v_1,\ldots,v_k)$ that violates no constraints to a feasible complete assignment $(y_1,\ldots,y_n)=(v_1,\ldots,v_n)$.
  - (Here, a partial assignment can violate a constraint only if assigns values to all the variables in the constraint.)
- A consistent constraint set is transparent in the sense that all implications of the constraints are already in the set.
- A branching algorithm can solve a consistent problem without backtracking (if it is feasible). It is roughly analogous to convex hull description of an IP problem.

Consistency

- A constraint set is *k-consistent* if any partial assignment to *k* variables that violates no constraints can be extended to an assignment to *k*+1 variables that violates no constraints.
    - This is related to backtracking and search order.
- It is *arc consistent* (generally defined for binary constraints) iff it is 2-consistent.
- It has *hyperarc consistency* (generalized arc consistency) if any value in a variable's domain is consistent with some feasible solution. (Reduces to arc consistency in binary problems.)
    - Ultimate goal of domain reduction.

Synergies between IP and CP

- **Exploiting problem structure:** Global constraints provide a practical means to take advantage of specialized algorithms (IP needs this).
- **Relaxation technology:** IP can supply continuous relaxations and "back propagation" for CP global constraints (CP needs this).
- **Inference technology:** CP can supply inference methods to reduce the search, e.g. domain reduction (IP's cutting planes strengthen the relaxation).

Synergies between IP and CP

- **Exploiting problem structure:** Global constraints provide a practical means to take advantage of specialized algorithms (IP needs this).
- **Relaxation technology:** IP can supply continuous relaxations and "back propagation" for CP global constraints (CP needs this).
- **Inference technology:** CP can supply inference methods to reduce the search, e.g. domain reduction (IP's cutting planes strengthen the relaxation).

Exploiting problem structure

• CP generally processes constraints *locally* or one at a time (results are “propagated” through the constraint store -- more on this later).

• A *global constraint* represents a specially-structured *set* of constraints (all-different, circuit, element, cumulative, etc.).

• By processing a global constraint, one exploits the *global* structure of the set of constraints it represents.

• Every global constraint “brings with it” a set of procedures (an idea suggested by the practice of modeling in a programming language).

Exploiting problem structure

• One can also associate *relaxations* and "*back propagation*" methods with global constraints. (More on this later.)

• This provides a principle for moving research output into commercial code.

  • Domain reduction methods are normally put to use right away, although as a result many are proprietary (cf. biotechnology).

  • Cutting plane methods tend to appear in the open literature, but many are not used in commercial codes (they are designed for special problems rather than special constraints).

Exploiting problem structure

- One can associate specialized cutting planes, etc., with global constraints representing constraint sets for which the cutting planes are designed.
  - For example, the global constraint circuit($y_1,\ldots,y_n$) requires that $y_1,\ldots,y_n$ represent a hamiltonian cycle on a graph, where y_j = vertex that follows vertex j. It could invoke a continuous relaxation that contains some TSP (separating) cuts.
  - This can put to use the large body of results in polyhedral analysis that are now employed only in specialized codes.

Exploiting problem structure

This approach creates a growing vocabulary of global constraints. This can get out of hand, but consider:

- The modeling language can exploit the expertise of the user
    - A domain expert thoroughly understands the structure of the problem in the real world (as opposed to the structure of its mathematical representation), and global constraints can capture this structure.

Exploiting problem structure

Example: cumulative($(t_1,\ldots,t_n)$,$(d_1,\ldots,d_n)$,$(r_1,\ldots,r_n)$,L)

t_i = start time of job i
d_i = duration
r_i = rate of resource consumption
L = limit on total rate of resource consumption at any time

Use cumulative($(t_1,\ldots,t_n)$,$(d_1,\ldots,d_n)$,$(1,\ldots,1)$,m) for m-machine scheduling.

Exploiting problem structure

- The user need only have advanced knowledge of the vocabulary that applies to his/her own application domain.
- A powerful feature of ordinary language is that we identify useful concepts that abbreviate clusters of more elementary concepts.
    - This requires learning more words but is inseparable from the task of learning how to do the task at hand.
    - Perhaps it is the same with modeling. The atomistic approach of IP misses this opportunity.

Synergies between IP and CP

• **Exploiting problem structure:** Global constraints provide a practical means to take advantage of specialized algorithms (IP needs this).

• **Relaxation technology:** IP can supply continuous relaxations and "back propagation" for CP global constraints (CP needs this).

• (IP's cutting planes strengthen the relaxation). **Inference technology:** CP can supply inference methods to reduce the search, e.g. domain reduction.

Relaxations (IP $\rightarrow$ CP)

- It is useful to work with an example: relaxation of the *element* constraint, which is important because it implements variable indices.

- *Variable indices* are a key modeling device for CP. A few models will illustrate their use.

Variable indices

Traveling salesman problem

$$\begin{aligned} \min \quad & \sum_j c_{y_j y_{j+1}} \\ \text{s.t.} \quad & \text{all-different}\{y_1, \ldots y_n\} \\ & y_j \in \{1, \ldots, n\} \end{aligned}$$

or...

$$\begin{aligned} \min \quad & \sum_j c_{j y_j} \\ \text{s.t.} \quad & \text{circuit}\{y_1, \ldots y_n\} \\ & y_j \in \{1, \ldots, n\} \end{aligned}$$

Variable indices

Quadratic assignment problem

$$\begin{aligned} \min \quad & \sum_{i,j} v_{ij} c_{y_i y_j} \\ \text{s.t.} \quad & \text{all-different}\{y_1, \ldots y_n\} \\ & y_j \in \{1, \ldots, n\} \end{aligned}$$

y_i = site assigned to facility i

v_{ij} = traffic between facility i and j

c_{kl} = distance between location k and l

Variable Indices

Assignment problem with two linked formulations

min some objective

s.t. constraints on x_i's

constraints on y_j's

$x_{y_j} = j$, all j

x_i = employee assigned time slot i
y_j = time slot assigned employee j

Variable Indices

- The linkage of two models improves constraint propagation.
- Here a variable (rather than a constant) has a variable subscript.

Element constraint

The constraint $c_y \leq 5$ can be implemented:

$$z \leq 5$$
$$\text{element}(y, (c_1, \ldots, c_n), z)$$

The constraint $x_y \leq 5$ can be implemented:

$$z \leq 5$$
$$\text{element}(y, (x_1, \ldots, x_n), z)$$

(this is a slightly different constraint)

Element constraint

element can be processed with a discrete domain reduction algorithm that maintains hvperarc consistency.

The more interesting case is $\text{element}(y, (x_1, \ldots, x_n), z)$

$$D_z \leftarrow D_z \cap \bigcup_{j \in D_y} D_{x_j}$$

$$D_y \leftarrow D_y \cap \{ j \mid D_x \cap D_{x_j} \neq \emptyset \}$$

$$D_{x_j} \leftarrow \begin{cases} D_z & \text{if } D_y = \{j\} \\ D_{x_j} & \text{otherwise} \end{cases}$$

Example... element$(y,(x_1,x_2,x_3,x_4),z)$

The initial domains are:

$D_z = \{20,30,60,80,90\}$

$D_y = \{1,3,4\}$

$D_{x_1} = \{10,50\}$

$D_{x_2} = \{10,20\}$

$D_{x_3} = \{40,50,80,90\}$

$D_{x_4} = \{40,50,70\}$

The reduced domains are:

$D_z = \{80,90\}$

$D_y = \{3\}$

$D_{x_1} = \{10,50\}$

$D_{x_2} = \{10,20\}$

$D_{x_3} = \{80,90\}$

$D_{x_4} = \{40,50,70\}$

Continuous relaxation of element

$\text{element}(y, (c_1, \ldots, c_n), z)$ is trivial.

The convex hull relaxation is $\min_i \{c_i\} \leq z \leq \max_i \{c_i\}$

$\text{element}(y, (x_1, \ldots, x_n), z)$ has the following relaxation

$$\sum_{i \in D_y} \frac{x_i}{m_i} - \left(\sum_{i \in D_y} \frac{1}{m_i} \right) z \geq -k + 1$$

$$-\sum_{i \in D_y} \frac{x_i}{m_i} + \left(\sum_{i \in D_y} \frac{1}{m_i} \right) z \geq -k + 1$$

provided $0 \leq x_i \leq m_i$ (and where $k = |D_y|$).

If $0 \leq x_i \leq m$ for all i, then the *convex hull* relaxation of

$\text{element}(y, (x_1, \ldots, x_n), z)$ is

$$\sum_{j \in D_y} x_j - (k-1)m \leq z \leq \sum_{j \in D_y} x_j$$

plus bounds, where $k = |D_y|$.

Example...

$$\text{element}(y, (x_1, x_2), z)$$
$$0 \le x_1 \le 5$$
$$0 \le x_2 \le 5$$

The convex hull relaxation is:

$$x_1 + x_2 - 5 \le z \le x_1 + x_2$$
$$0 \le x_1 \le 5$$
$$0 \le x_2 \le 5$$

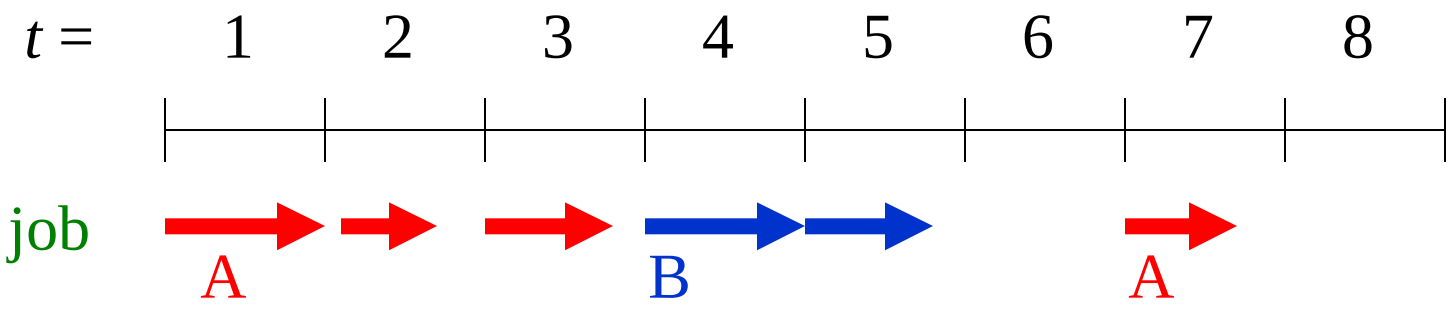
If $0 \le x_1 \le 4$ the above remains valid and we have

$$5x_1 + 4x_2 - 20 \le 9z \le 5x_1 + 4x_2 + 20$$

Discrete lot sizing example

- Manufacture at most one product each day.
- When manufacturing starts, it may continue several days. (R_i = minimum run length).
- Switching to another product incurs a cost.
- There is a certain demand for each product on each day.
- Products are stockpiled to meet demand between manufacturing runs.
- Minimize inventory cost + changeover cost.

Discrete lot sizing example



y_t = A A A B B 0 A 0

0 = dummy job

IP model

(from L. Wolsey)

$$\min \quad \sum_{t,i} \left(h_{it} s_{it} + \sum_{j \neq i} q_{ij} \delta_{ijt} \right)$$

$$\text{s.t.} \quad s_{i,t-1} + x_{it} = d_{it} + s_{it}, \quad \text{all } i,t$$

$$z_{it} \geq y_{it} - y_{i,t-1}, \quad \text{all } i,t$$

$$z_{it} \leq y_{it}, \quad \text{all } i,t$$

$$z_{it} \leq 1 - y_{i,t-1}, \quad \text{all } i,t$$

$$\delta_{ijt} \geq y_{i,t-1} + y_{jt} - 1, \quad \text{all } i,t$$

$$\delta_{ijt} \geq y_{i,t-1}, \quad \text{all } i,t$$

$$\delta_{ijt} \leq y_{jt}, \quad \text{all } i,t$$

$$z_{it} \leq y_{i,t+r-1}, \quad r = 1, \ldots, R_i, \text{ all } i,t$$

$$x_{it} \leq C y_{it}, \quad \text{all } i,t$$

$$\sum_{i} y_{it} = 1, \quad \text{all } t$$

$$y_{it}, z_{it}, \delta_{ijt} \in \{0,1\}$$

$$x_{it}, s_{it} \geq 0$$

The model

total inventory + changeover cost

stock level

changeover cost

daily production

inventory balance

$$\min \quad \sum_t (u_t + v_t)$$

$$\text{s.t.} \quad u_t \geq \sum_i h_i s_{it}, \ \text{all}\, t$$

$$v_t \geq q_{y_{t-1} y_t}, \ \text{all}\, t$$

$$s_{i,t-1} + x_{it} = d_{it} + s_{it}, \ \text{all}\, i,t$$

$$0 \leq x_{it} \leq C, \ s_{it} \geq 0, \ \text{all}\, i,t$$

$$(y_t \neq i) \rightarrow (x_{it} = 0), \ \text{all}\, i,t$$

$$(y_{t-1} \neq i = y_t) \rightarrow (y_{t+1} = \cdots = y_{t+R_i-1} = i), \ \text{all}\, i,t$$

Relaxation

Put into relaxation

Generate inequalities to put into relaxation

$$\min \quad \sum_t (u_t + v_t)$$

$$\text{s.t.} \quad u_t \geq \sum_i h_i s_{it}, \ \text{all } t$$

$$v_t \geq q_{y_{t-1} y_t}, \ \text{all } t$$

$$s_{i,t-1} + x_{it} = d_{it} + s_{it}, \ \text{all } i,t$$

$$0 \leq x_{it} \leq C, \ s_{it} \geq 0, \ \text{all } i,t$$

$$(y_t \neq i) \rightarrow (x_{it} = 0), \ \text{all } i,t$$

$$(y_{t-1} \neq i = y_t) \rightarrow (y_{t+1} = \cdots = y_{t+R_i-1} = i), \ \text{all } i,t$$

Apply constraint propagation to everything

To solve the example

- At each node of search tree:
  - Apply domain reduction and constraint propagation.
  - Generate and solve continuous relaxation to get bound.
- Characteristics:
  - Relaxation is somewhat weaker than in IP because logical constraints are not all relaxed.
  - But LP relaxations are much smaller--quadratic rather than cubic size.
  - Domain reduction helps prune tree.

Back propagation

• A global constraint can also be associated with a back propagation scheme, which reduces domains based on solution of the relaxation.

• A simple example is variable fixing using reduced costs, now used in both IP and CP.

  • For instance, one can give circuit($y_1,\ldots,y_n$) an assignment relaxation.

  • The value of the relaxation may be useless, but the reduced costs can allow one to exclude values of y_i.

  • Variables x_{ij} in assignment problem need not be defined. Just use appropriate data structure to solve problem.

A peek at a modeling framework for integration

Let every constraint i have *conditional form*

$$h_i(y) \rightarrow S_i(x)$$

Or be reducible to a set of conditionals.

$h_i(y)$ - *hard* constraint (belonging to NP); contains variables $y = (y_1,\ldots,y_m)$.

$S_i(x)$ - set of *easy* constraints (belonging to NP and co-NP?) that will go into relaxation; contains variables $x = (x_1,\ldots,x_n)$.

The objective function has the form $f(x) + g(y)$.

The basic search algorithm

- *Branch* on domains of variables y_j.
- Use *inference* to deduce, when possible, whether partially specified variables y_j satisfy constraints $h_i(y)$.
- *Solve* the relaxed problem of minimizing $f(x)$ with respect to x, subject to the $S_i(x)$'s that are enforced by true $h_i(y)$'s.
    - This relaxation can be strengthened, or augmented by other relaxations, if desired.
- Back-propagate from the solution of the relaxation.
- Search for value of y consistent with this solution.
- Continue in a branch-and-relax fashion.

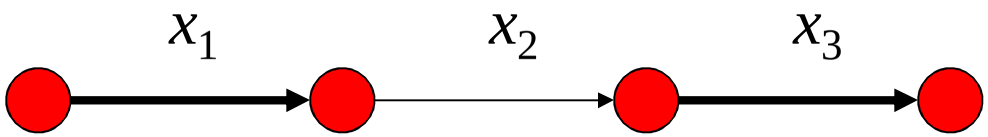
Synergies between IP and CP

- **Exploiting problem structure:** Global constraints provide a practical means to take advantage of specialized algorithms (IP needs this).
- **Relaxation technology:** IP can supply continuous relaxations and "back propagation" for CP global constraints (CP needs this).
- **Inference technology:** CP can supply inference methods to reduce the search, e.g. domain reduction (IP's cutting planes strengthen the relaxation).

Inference (CP $\rightarrow$ IP)

- Inference accelerates search by making the constraint set more nearly consistent--i.e., by making implications explicit.
    - The most popular approach is to reduce domains (aim for hyperarc consistency), which reduces branching.
- The research project of finding domain reduction algorithms is analogous to discovery of good cutting planes.
- The structural analysis associated with cutting plane theory (e.g., special subgraphs, etc.) may suggest constraints that are good in the sense that they move closer to consistency.
    - For example, alternating paths in matching correspond to derived constraints (which strictly dominate facet-defining cuts).

Inference (CP $\rightarrow$ IP)



Convex hull description of this matching problem:

$$x_1 + x_2 \leq 1$$
$$x_2 + x_3 \leq 1$$
$$x_1, x_2, x_3 \geq 0$$

Alternating path shown corresponds to $x_1 + 2x_2 + x_3 \leq 2$.

This strictly dominates both facets (in 0-1 sense).
It clearly dominates.
It also excludes (0,1,1), which satisfies 1st facet,
and (1,1,0), which satisfies 2nd facet.

Inference (CP $\rightarrow$ IP)

- Domain reduction can be regarded as constraint (cut) generation.
- To reduce the domain of y_j is to post an in-domain constraint $y_j \in D_j$.
- The generated in-domain constraints are normally regarded as forming a *constraint store*,which allows communication among constraints.
- But they can be regarded as forming a discrete *relaxation* (i.e., optimization problem solved to get a bound)
    - Each domain element belongs to some feasible solution, but simply picking an element from each domain may not yield a solution.

Inference (CP $\rightarrow$ IP)

- One might solve this relaxation to optimality (usually trivial) and generate "separating" in-domain constraints.
  - Solution of relaxation would guide domain reduction.
- One might also allow a broader class of easy constraints in the constraint store.
  - For example, constraints whose dependency graph has limited induced width, to be solved by nonserial dynamic programming.

A few things I didn’t tell you about

- Other forms of logical inference that can process discrete constraints in hybrid methods (e.g. resolution, useful in engineering design).
- Cuts for disjunctions of linear and nonlinear systems.
- Hybrid methods for mixed discrete/nonlinear problems (logic-based outer approximation, Benders decomposition for disjunctive programming).

A few things I didn’t tell you about

- Discrete relaxation duality as a method for obtaining bounds.
  - Lagrangean, surrogate and linear programming duality are classical special cases.
  - Parameterize relaxations and search the parameter space for a good relaxation.
  - Arguably, LP relaxation is “global” because it is in effect a relaxation dual problem.

A few things I didn't tell you about

- Logic-based Benders decomposition as a principle for integrating CP and IP (or optimization in general).
  - Computational results to date are encouraging.
  - Benders subproblem can be CP problem. Objective function appears in the master problem.
  - Dual of subproblem generalized to solution of "inference dual" whose solution is a proof (next slide).
  - Benders cut is a constraint that must be violated for proof to remain valid.

A few things I didn't tell you about

- Search/inference duality can be formalized to obtain "inference dual."
  - Leads to logic-based Benders decomposition and sensitivity analysis for discrete optimization.
- There are search strategies that generalize branching.
  - They provide more freedom to pursue promising directions but remain complete.
  - For example, dependency-directed backtracking, partial-order dynamic backtracking.
  - Can be characterized as constraint-based search using inference methods of various strengths (weakest in branching).