Logic, Optimization, and Constraint Programming

A Fruitful Collaboration

John Hooker
Carnegie Mellon University

Simons Institute, UC Berkeley
April 2023
There are deep connections between logic, optimization, and constraint programming (CP) – going back at least to George Boole.

This is a broad overview of these connections, as they developed over the 170-year period from Boole’s work to today’s research.

Collaboration among these fields could provide a fruitful trajectory for future research.
From Boole to Logic Programming & CP
George Boole advanced a project begun by Leibniz, although Boole (largely self-taught) was initially unaware of Leibniz’s work.

Leibniz believed that all of science can be formulated in a logical language (*characteristica universalis*) in which implications can be obtained by calculation (*calculus ratiocinator*), such as the calculus of infinitesimals.

Boole devised a language in which logical deductions can be *calculated*.
Boole’s work was largely forgotten for a century.

But it was studied by philosopher Charles Sanders Peirce in the late 19th century.

Boole introduced multi-place predicates, to which Pearce added logical quantifiers (“for all,” “for some”).

Gottlob Frege developed a fully formed first-order logic in the 1890s.
Löwenheim, Skolem, Herbrand and others developed systematic semantics for first-order logic. They proved fundamental theorems, including Herbrand’s compactness theorem.

There is an almost identical theorem in infinite-dimensional integer programming.
Herbrand’s theorem (compactness)

A formula of first order logic is unsatisfiable if and only if some finite set of ground instances of the formula is unsatisfiable.

Compactness theorem for integer programming

An infinite set of linear inequalities with integer variables is unsatisfiable if and only if some finite subset is unsatisfiable.

Proof?

The 2 theorems are structurally almost identical and have almost exactly the same proof.
From Boole to Logic Programming & CP

Logic programming arose from an effort to combine declarative and procedural modeling in quantified logic.

A logic program can be read as a declarative statement of the problem, as well as a procedure for obtaining the solution.

This later became a fundamental idea of constraint programming.
A key step in first order logic is **unification**, which finds variable substitutions that make two instantiations of a formula identical. This is essentially a **constraint solving** problem.

```
likes(Sue, X), likes(Y, Bob)
unified by setting X = Bob, Y = Sue
```

Logic programming was extended to **constraint logic programming** in **Prolog II**, which added disequations to the unification step. Other forms of constraint solving were added later.
Constraint programming “toolkits” retained constraint solving in a procedural/declarative framework, without requiring a strict logic programming formalism.

This led to CP-style modeling with finite domains and global constraints.

Constraint propagation allows efficient inference from constraint sets.

The constraint satisfaction literature studied consistency concepts and their connection with backtracking (more on this later).
From Boole to SAT
Much of Boole’s and Pearce’s work dealt with “Boolean algebra,” which is essentially propositional logic (“ground level” propositions).

The philosopher W. V. Quine proposed (1950s) a consensus method for simplifying propositional formulas that is a complete inference method for propositional logic.

When applied to CNF rather than DNF, the method is resolution.

\[
\begin{align*}
x_1 \lor x_2 \lor x_4 \\
x_1 \lor \neg x_4 \\
\hline
x_1 \lor x_2
\end{align*}
\]
The **Davis-Putnam algorithm**, devised to check validity in first-order logic, applies resolution to instantiated (ground level) propositions.

Resolution was later replaced with more efficient methods for checking satisfiability of CNF formulas, such as **branching** in the David-Putnam-Loveland-Logemann (DPLL) method.

These led to today’s highly efficient **SAT** methods, which use **watched literals**, **conflict clauses**, etc.
Conflict clauses lie at the heart of SAT algorithms. We will see later that they are actually Benders cuts.

Conflict clauses enable backjumping and reduce search.

Refutation using DPLL tree and conflict clauses:

Apply unit resolution at each node. Backtrack when unsatisfiable.

\[ (\overline{x_1} \lor x_3 \lor x_4) \land (\overline{x_2} \lor x_3 \lor x_4) \]
\[ (\overline{x_3} \lor \overline{x_1}) \land (\overline{x_3} \lor \overline{x_2}) \land (\overline{x_4} \lor \overline{x_1}) \land (\overline{x_4} \lor \overline{x_2}) \]
**Conflict clauses** lie at the heart of SAT algorithms. We will see later that they are actually **Benders cuts**.

We will derive this conflict clause:

\[
\begin{align*}
    x_1 &\lor x_5 \lor x_6 \\
    \overline{x}_6 &\lor x_1 \lor x_5 \\
    \overline{x}_5 &\lor x_2 \lor x_6 \\
    \overline{x}_5 &\lor \overline{x}_6 \lor x_2 \\
    (\overline{x}_1 &\lor x_3 \lor x_4) \land (\overline{x}_2 &\lor x_3 \lor x_4) \\
    (\overline{x}_3 &\lor \overline{x}_1) \land (\overline{x}_3 &\lor \overline{x}_2) \land (\overline{x}_4 &\lor \overline{x}_1) \land (\overline{x}_4 &\lor \overline{x}_2)
\end{align*}
\]

**Refutation using DPLL tree and conflict clauses:**

- Apply unit resolution at each node.
- Backtrack when unsatisfiable.

Conflict clauses enable **backjumping** and reduce search.
Conflict clause \( x_1 \lor x_5 \) is obtained from unit refutation by analyzing the implication graph at that node.

**Implication graph**

Darker circles indicate branching literals

**Conflict graph**

from implication graph

\( x_1 \lor x_5 \)

\[
\begin{align*}
(1) & \quad x_1 \lor x_5 \lor x_6 \\
(2) & \quad x_1 \lor x_5 \lor \neg x_6 \\
(3) & \quad x_2 \lor \neg x_5 \lor x_6 \\
(4) & \quad x_2 \lor \neg x_5 \lor \neg x_6 \\
(5) & \quad \neg x_1 \lor x_3 \lor x_4 \\
(6) & \quad \neg x_2 \lor x_3 \lor x_4 \\
(7) & \quad \neg x_3 \lor \neg x \\
(8) & \quad \neg x_4 \lor \neg x_1 \\
(9) & \quad \neg x_4 \lor \neg x_2 \\
\end{align*}
\]
From Boole to Decision Diagrams
C. S. Peirce applied Boolean methods to electrical switching circuits ... in 1886!

This work was again forgotten for decades.

**Claude Shannon** was required to take a philosophy course at the University of Michigan in the 1930s, which exposed him to Peirce’s work.

This gave him the idea for his famous master’s thesis at MIT (1937), in which he applied Boolean logic to electronic switching circuits.

This gave rise to the **computer age**.
Meanwhile, C. Y. Lee (1959) proposed binary-decision programs as a means of calculating the output of switching circuits.

S. B. Akers (1978) later represented these as binary decision diagrams.

Randy Bryant (1986) showed that ordered BDDs provide a unique minimal representation of a Boolean function.

This led to applications in logic circuits and product configuration.

Decision diagrams are now used for filtering and propagation in CP...
From Boole to Decision Diagrams

Propagation through domains. Let $x_1, x_2, x_3$ have domain \{1, 2, 3\}
\[ x_1 + 2x_2 + 3x_3 \leq 10 \] filters domains to $x_1, x_2 \in \{1, 2, 3\}, x_3 \in \{1, 2\}$
all-different$(x_1, x_2, x_3)$ no more filtering possible for propagated domains
From Boole to Decision Diagrams

Propagation through domains.

Let $x_1, x_2, x_3$ have domain $\{1, 2, 3\}$

\[ x_1 + 2x_2 + 3x_3 \leq 10 \quad \text{filters domains to } x_1, x_2 \in \{1, 2, 3\}, \ x_3 \in \{1, 2\} \]

all-different($x_1, x_2, x_3$) \quad \text{no more filtering possible for propagated domains}

Propagation through a relaxed decision diagram.

Relaxed diagram for $x_2 + 2x_2 + 3x_3 \leq 10$

Relaxed diagram after propagating all-different($x_1, x_2, x_3$)
From Boole to Decision Diagrams

Decision diagrams can perform all functions of an optimization solver.
From Boole to Probability and Belief Logics
LOGIC

Boole
Peirce
1st order logic
Herbrand's theorem
Logic programming
CLP

Probability logic
Prop. logic
Shannon & circuits
Davis-Putnam

Belief & default logics
Decision diagrams
Consistency
Constraint propagation

Constraint satisfaction & backtracking

OPTIMIZATION

Fourier-Motzkin
LP
IP
Resolution
Cutting planes
Network flows
Filtering
Edge finding
Consistency in IP

Inference dual
Benders decomp.
LBBD
Duality

Logic programming
CLP

CP in red
Boole considered probability logic to be his most important contribution. His major work was *An Investigation of the Laws of Thought on Which are Founded the Mathematical Theories of Logic and Probabilities* (1854).

Theodore Hailperin (1976) showed that Boole’s probability logic poses a linear programming problem.

Nils Nilsson (1986) proposed a very similar model for probability logic in AI.

This model is naturally solved by column generation, a widely used method in OR that generalizes Dantzig-Wolfe decomposition.
Example: What are the possible probabilities of statement C, given the following?

<table>
<thead>
<tr>
<th>Statement</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0.9</td>
</tr>
<tr>
<td>A → B</td>
<td>0.8</td>
</tr>
<tr>
<td>B → C</td>
<td>0.4</td>
</tr>
</tbody>
</table>

Solve the linear programming problems:

\[
\begin{align*}
\text{max/min} & \quad p_{001} + p_{011} + p_{101} + p_{111} \\
\text{subject to} & \\
p_{100} + p_{101} + p_{110} + p_{111} = 0.9 \\
p_{000} + p_{001} + p_{010} + p_{011} + p_{100} + p_{101} + p_{111} = 0.8 \\
p_{000} + \ldots + p_{111} = 1, \ p_{ijk} \geq 0
\end{align*}
\]

There are exponentially many variables, but LP column generation deals with this.

There are 8 possible outcomes:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>Prob.</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>(p_{000})</td>
</tr>
<tr>
<td>001</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>(p_{001})</td>
</tr>
<tr>
<td>010</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>(p_{010})</td>
</tr>
<tr>
<td>011</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>(p_{011})</td>
</tr>
<tr>
<td>100</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>(p_{100})</td>
</tr>
<tr>
<td>101</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>(p_{101})</td>
</tr>
<tr>
<td>110</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>(p_{110})</td>
</tr>
<tr>
<td>111</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>(p_{111})</td>
</tr>
</tbody>
</table>

The result is a range of probabilities for C: 0.1 to 0.4
Dempster-Shafer theory (belief logic) has a **linear programming** model similar to the one for Boole’s probability logic.

**Nonmonotonic logic** has a succinct **integer programming** model that arguably makes the concept clearer than a logical formulation.

Confidence factors in **rule-based systems** have a mixed integer/linear programming model.
From Fourier to Filtering
Fourier (1820s) developed a theory of **linear inequalities** and a method of solving them, later rediscovered by Motzkin (1936). The method is now called Fourier-Motzkin elimination.

Kantorovich (1939) formulated a linear optimization problem subject to inequality constraints – i.e., **linear programming**. Dantzig (1940s) independently proposed and solved the same model.
Fourier-Motzkin elimination can solve LP problems, but Dantzig’s simplex method is far more efficient and remains the method of choice for most applications today.

LP with integer variables, or integer programming, followed shortly thereafter...

...along with the study of combinatorial optimization in general, beginning with the traveling salesman problem.
Two major success stories for collaboration between CP and optimization:

1. **Network flow theory**, a special case of LP, has been widely applied to filtering methods in CP, beginning with the all-different constraint. LP duality plays a key role in this work.

2. **Edge-finding**, an algorithm for combinatorial scheduling, led to powerful domain reduction methods for scheduling problems in CP. Edge finding was originally published in the OR journal *Management Science* (Carlier and Pinson 1989), with most subsequent papers in the CP literature.
Example of network flows and filtering.

An all-different constraint has a solution if and only if there is a perfect matching:

\[
\text{alldiff}\left( x_1, x_2, x_3, x_4, x_5 \right)
\]

- \( x_1 \in \{1\} \)
- \( x_2 \in \{2,3,5\} \)
- \( x_1 \in \{1,2,3,5\} \)
- \( x_1 \in \{1,5\} \)
- \( x_1 \in \{1,3,4,5,6\} \)

Solution shown:
\[
(x_1, x_2, x_3, x_4, x_5) = (1,2,3,5,4)
\]
The matching problem can be viewed as a maximum flow problem on a network, which is a linear programming problem.

The dual solution of the problem indicates how to filter domains:

- \( x_1 \in \{1\} \)
- \( x_2 \in \{2,3\} \)
- \( x_3 \in \{2,3\} \)
- \( x_4 \in \{5\} \)
- \( x_5 \in \{4,6\} \)
If job 2 is **not before 3 and 5**, then there is not enough time in their time windows (7 hours) to run all 3 jobs (requiring 8 hours). So, time window for job 2 must be **reduced** to [0,2], which is infeasible.

Edge-finding check can run in **polynomial time**, and can be **generalized** to other scheduling problems.
From Fourier to Inference Duality
After a chance meeting on a rail platform near Princeton University, Dantzig and von Neumann combined ideas from LP and game theory to arrive at **LP duality**.

Duality has become a powerful idea in optimization, e.g. Lagrangian duality, **Dantzig-Wolfe decomposition** (column generation), and **Benders decomposition** (row generation).
All optimization duals are special cases of **inference duality**

**Primal problem:** Optimization

\[
\min \ f(x) \\
x \in S
\]

Find **best** feasible solution by searching over values of \(x\).

**Dual problem:** Inference

\[
\max \ v \\
x \in S \quad \text{such that} \quad f(x) \geq v \\
P \in \mathcal{P}
\]

Find a **proof** of optimal value by searching over proofs \(P\).

The **type** of dual depends on the **inference method** used. In **classical LP**, the proof is a tuple of **dual multipliers**. A **complete** inference method yields a **strong dual** (no duality gap).
<table>
<thead>
<tr>
<th>Type of Dual</th>
<th>Inference Method</th>
<th>Strong?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear programming</td>
<td>Nonnegative linear combination + material implication</td>
<td>Yes*</td>
</tr>
<tr>
<td>Lagrangian</td>
<td>Nonnegative linear combination + domination</td>
<td>No</td>
</tr>
<tr>
<td>Surrogate</td>
<td>Nonnegative linear combination + material implication</td>
<td>No</td>
</tr>
<tr>
<td>Subadditive</td>
<td>Cutting planes</td>
<td>Yes**</td>
</tr>
</tbody>
</table>

*Due to Farkas Lemma  
**Due to Chvátal’s theorem
Benders decomposition was designed for problems that become LPs after some variables are fixed.

The dual of the LP subproblem provides a Benders cut that excludes undesirable solutions.

Generalization to logic-based Benders cuts:

Using the inference dual, the subproblem can in principle be any optimization or constraint satisfaction problem.

So, a logical perspective leads to a substantial generalization with many new applications.
From Fourier to Inference Duality

Classical Benders decomposition

Solve the problem

\[
\min \ f(x) + cy \\
g(x) + Ay \geq b
\]

Master problem

\[
\begin{align*}
\min & \quad z \\
\text{s.t.} & \quad f(x) + u_k(b - g_k(x)), \\
& \quad \text{all } k \text{ (Benders cuts)}
\end{align*}
\]

Minimize cost \( z \) subject to bounds given by Benders cuts, obtained from values of \( x \) attempted in previous iterations \( k \).

Subproblem

\[
\begin{align*}
\min & \quad f(\bar{x}) + cy \\
\text{s.t.} & \quad Ay \geq b - g(\bar{x})
\end{align*}
\]

Obtain proof of optimality (solution \( u \) of LP dual). Use dual solution to obtain a Benders cut.

Repeat until the master problem and subproblem have the same optimal value.
Logic-based Benders decomposition (LBBD)

\[
\min \ f(x, y) \\
(x, y) \in S \\
x \in D
\]

Master problem

\[
\min \ z \\
z \geq v_k(x), \ all \ k \ (\text{Benders cuts})
\]

Minimize cost \( z \) subject to bounds given by Benders cuts, obtained from values of \( x \) attempted in previous iterations \( k \).

Subproblem

\[
\min \ f(\bar{x}, y) \\
(\bar{x}, y) \in S
\]

Obtain proof of optimality (solution of inference dual). Use \textit{same proof} to deduce cost bounds for other values of \( x \), yielding a Benders cut.

Repeat until the master problem and subproblem have the same optimal value.
From Fourier to Inference Duality

**LBBD** has been applied to a wide range of problems that simplify (perhaps by decoupling) when some variables are fixed.

It is a useful tool for **combining optimization and CP**.

Typically, an optimization method (such as MILP) solves the master problem and **CP solves the subproblem** (often a scheduling problem).

The **conflict clauses** that are central to **SAT solvers** are a special case of logic-based Benders cuts.

**SAT-modulo-theories** are also solved as a special case of LBBD.
From Fourier to Inference Duality

Conflict clauses as logic-based Benders cuts

- The **subproblem** is the problem at a node of the DPLL search tree.
- The **inference dual** is defined by **unit resolution**.
- The **dual solution** is a unit refutation, encoded in a **conflict graph**.

The conflict graph is a solution of the inference dual

The resulting **conflict clause** is a Benders cut

\[ x_1 \lor x_5 \]
From Fourier to Cutting Planes
Cutting planes, studied for over 60 years, are an essential component of integer programming solvers. They are closely related to resolution and Fourier-Motzkin elimination. They approximate the convex hull of integer solutions, so that the LP relaxation gives a tighter bound.

Ralph Gomory 1929-
When the logical clauses are written as inequalities (as suggested by Dantzig), resolution is Fourier-Motzkin elimination combined with rounding of fractions.

\[
\begin{align*}
x_1 + x_2 + x_4 & \geq 1 \\
x_1 - x_4 & \geq 0 \\
-x_2 & \geq 0
\end{align*}
\]

When rounding, we get

\[
x_1 + x_2 \geq \left\lfloor \frac{1}{2} \right\rfloor
\]
This means that a resolvent is a rank 1 Chvátal-Gomory cut.

\[
\begin{align*}
x_1 + x_2 + x_4 & \geq 1 \quad (1/2) \\
x_1 - x_4 & \geq 0 \quad (1/2) \\
x_2 & \geq 0 \quad (1/2)
\end{align*}
\]

\[
x_1 + x_2 \geq \left\lceil \frac{1}{2} \right\rceil
\]

The fundamental theorem of cutting planes (due to Chvátal) states that any valid cutting plane can be obtained from repeated generation of rank 1 Chvátal-Gomory cuts.

The proof of this theorem is based on the resolution algorithm!

Cutting planes lie at the heart of integer programming, and logic lies at the heart of cutting planes.
Consistency is a fundamental concept in CP.

It is not satisfiability or feasibility.

We can view a consistent constraint set as one in which any infeasible partial assignment is inconsistent with some constraint.

This avoids backtracking, because each node corresponds to a partial assignment defined by branches so far. We can detect whether deeper branching can find a feasible solution.

CP solvers try to achieve various kinds of partial consistency (e.g., domain consistency) to reduce backtracking.
Cutting planes are normally viewed as tightening an LP relaxation to obtain better bounds.

Separating cuts exclude fractional solutions.

But cutting planes also achieve (partial) consistency!

They exclude inconsistent partial assignments.

This helps to explain why they can reduce backtracking.
The constraint set

\[ 2x_1 + 4x_2 \leq 5 \]
\[ 7x_1 - 2x_2 \leq 6 \]
\[ x_1, x_2 \in \{0, 1\} \]

is not consistent

\[ x_1 = 1 \rightarrow \text{infeasible} \]
\[ x_2 = 1 \]
\[ x_1 = 0 \rightarrow \text{infeasible} \]

Backtracking can result even with forward checking.
The constraint set
\[ 2x_1 + 4x_2 \leq 5 \]
\[ 7x_1 - 2x_2 \leq 6 \]
\[ 2x_1 \leq 1 \]
\[ x_1, x_2 \in \{0, 1\} \]
is consistent.

Violates a constraint \( x_1 = 1 \) \( \rightarrow \) infeasible

\[ x_1 = 0 \]}

No backtracking with forward checking

Don’t take the \( x_1 = 1 \) branch
LP-consistency is a type of consistency that is relevant to IP:

An LP-consistent constraint set is one in which any infeasible partial assignment is infeasible in the LP relaxation.

This allows us to recognize inconsistent partial assignments by solving an LP.

Cutting planes can achieve partial LP consistency and thereby reduce backtracking.

Bounding and fractional solutions need not play a role.
From Fourier to Cutting Planes

The constraint set

\begin{align*}
2x_1 + 4x_2 &\leq 5 \\
7x_1 - 2x_2 &\leq 6 \\
x_1, x_2 &\in \{0, 1\}
\end{align*}

is not LP-consistent

Feasible in LP relaxation

\begin{align*}
x_1 &= 1 \\
x_2 &= 1 & \text{infeasible in IP} \\
x_1 &= 0 \\
x_2 &= 0 & \text{infeasible in IP}
\end{align*}

Backtracking can result even with forward checking.
The constraint set
\[2x_1 + 4x_2 \leq 5\]
\[7x_1 - 2x_2 \leq 6\]
\[x_1 + x_2 \leq 1\] 
with \(x_1, x_2 \in \{0, 1\}\) is LP-consistent.

Infeasible in LP relaxation

\[x_1 = 1\]
\[x_1 = 0\]

infeasible in IP

No backtracking with forward checking

Don’t take the \(x_1 = 1\) branch
Theorem: An IP constraint set is LP-consistent if and only if all implied logical clauses (written as inequalities) are rank 1 Chvátal-Gomory cuts.

This again links logic and cutting planes.
From Fourier to Cutting Planes

**Theorem:** An IP constraint set is LP-consistent **if and only if** all implied **logical clauses** (written as inequalities) are **rank 1 Chvátal-Gomory cuts**.

This again links **logic** and **cutting planes**.

Let **consistency cuts** be cutting planes that cut off inconsistent partial assignments.

We can achieve partial LP consistency with a restricted form of **RLT** (reformulation and linearization technique).

RLT-based consistency cuts can **reduce the search tree** substantially more than traditional separating RLT cuts, also with time savings.
Advances in the logic-optimization-CP interface

- Constraint logic programming, leading to CP
- Fundamental theorem of cutting planes in IP
- Conflict-directed clause generation in SAT
- Logic-based Benders decomposition
- Combinatorial optimization with decision diagrams
- IP models for first-order logic, nonmonotonic logic
- LP model with column generation for probability logic
- LP models for belief logics
- Flow-based filtering methods in CP
- CP-based solution of scheduling problems
- Reinterpretation of cutting planes as consistency maintenance
- More to come?
Thanks for your attention!