Computational Models of Constraint Propagation

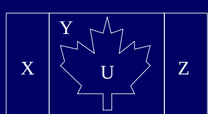
Alexander V. Konstantinou
Columbia University

Candidacy Exam
December 9th, 1999

Presentation Overview

- Introduction
- Constraint Propagation (5)
- Constraint Logic Programming (3)
- Algorithms (6)
  – Interval Propagation (2)
- Systems (6)
- Future Work

Constraint Example



- Color flag (red, white)
- Maple leaf is red
- Neighbors have different colors

Variables : X, Y, U, Z

Domains : $D_X = D_Y = D_U = D_Z$ = { white, red }

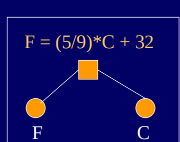
Constraints : (1) U = red (3) X ≠ Y
(2) Y ≠ U (4) U ≠ Z

Formal CSP Definition

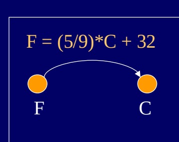
- *Constraint* is a relation over some domain *D*
- *Constraint graph G* = <*C, V, D*>
- *Valuation* θ function(v ∈ *V*) ⟶ elements of *D*
- *Solution S* is set of all valuations satisfying all *C*

E.g., *G*= < { c*1.8 = f - 32 } , {c, f}, *R* >
θ = { c, f ! 0.0, 32.0 }
S = { {c, f ! 0.0, 32.0 }
{c, f ! -40.0, -40.0 } … }

Constraint Graph Representation



N-ary constraints



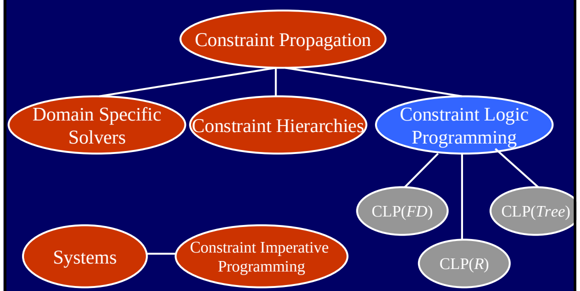
Binary constraints

- How are constraints evaluated ?
- F = (5/9)*C + 32 methods : (multi-way constraint)
  – F := (5/9)*C + 32
  – C := (9/5)*(F - 32)

Constraint Satisfaction

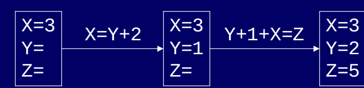
- Generate & Test (NP)
- Local Propagation (P)
  – No cycles (simultaneous equations)
  – No partial information constraints (greater-than)
- Search (NP)
  – E.g., Backpropagation + local propagation
- Domain-specific algorithms (P/NP)
  – E.g., Gaussian elimination (integers)

Constraint Research



Local Propagation

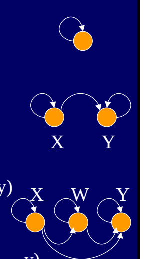
- Data-flow phases :
  - Determine variable value using constraint
  - Use value in another constraint, determine new variable value
- Handles non-numeric constraints
- Does not handle:
  - Cycles (simultaneous equations) — Is not complete
  - Partial information constraints (greater-than)



Backtracking + Propagation

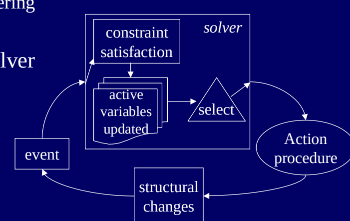
[Mackworth 1977, Mohr 1986]

- Node V_i is ***node consistent*** iff
  - $\forall x \in D_i, C_i(x)$
- Arc(i,j) is ***arc consistent*** iff
  - $\forall x \in D_i, C_i(x) \; \exists y \in d_j, C_j(y) \wedge C_{ij}(x, y)$
- Path $(i_0, i_1, \ldots i_m)$ is ***path consistent*** iff
  - $\forall x \in D_{i0}, y \in D_{im}, C_{i0}(x) \wedge C_{im}(y) \wedge C_{io\ im}(x,y)$
    $\exists z_1 \in D_{i1}, \ldots z_{m-1} \in D_{im-1}$:
    (i) $C_{i1}(z1) \wedge \ldots \wedge C_{im-1}(z_{m-1})$
    (ii) $C_{io\ i1}(x, z_1) \wedge C_{i1\ i2}(z_1, z_2) \wedge \ldots \wedge C_{im-1\ im}(z_{m-1}, y)$



Spreadsheet Model [Zanden92]

- Active-value-spreadsheet model
  - allow side-effects during constraint solving
  - solver decides ordering
  - cycle handling
- Procedures help solver
  - gain in efficiency
  - increase program complexity



Constraint Hierarchies

- Overconstrained/underconstrained problems
- Which variables to alter to satisfy multi-way constraints ?
  - E.g., change IP host address, or renumber whole network ?
- Constraint Hierarchies :
  - Labeled constraints (strength [0 … m])
  - Comparators (**locally-better**/globally-better)
  - Weights
  - Annotations (read/write only)

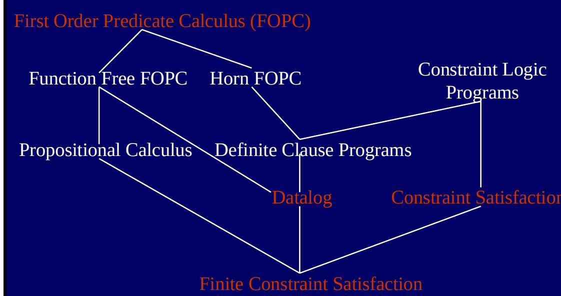
Constraint Logic Programming

Constraint Logic Programming
[Cohen90]

- The equality "1 + X = 3" fails in Prolog
  - Symbol '+'considered unevaluated and unification fails
- Workarounds exist (use successor, or "is")
- Solution: replace unification by constraints
- ...

```
solve([], C, C)
solve(Goal|Restgoal, Previous_C, New_C) :-
    solve(Goal, Previous_C, Temp_C),
    solve(Restgoal, Temp_C, New_C).
solve(Goal, Previous_C, New_C),
    clause(Goal, Body, Current_C),
    merge_constraints(Previous_C, Current_C, Temp_C),
    solve(Boddy, Temp_C, New_C).
```


Expressive Power [Mackworth92]



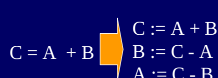
Algorithms

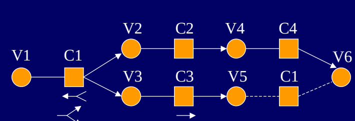
Incremental Local Propagation
(DeltaBlue) [Gagnet92]

- Local propagation
- No cycles
- One-way constraints
- Incremental
- Handles constraint hierarchies
- Maintains solution graph
- Separates planning from evaluation
- O(M*N) | N constraints, M max methods/constr.
- Implemented in various systems

Multi-way Propagation (SkyBlue)
[Sannella, 1994]

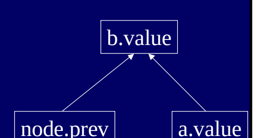
- General solver (methods)
- Incremental
- Cycle-aware
- Selects method, constructs directed method graph





Pointer Variables [Zanden, 1994]

- `node.value >= node.prev.value`
- Incremental algorithms (lazy/eager)
- Dependency graph based nulification/reevaluation
- Timestamps to support changing references **during** constraint evaluation
- One-way constraints
- Handles cycles
- O(|affected|)



Inequality Constraints (Indigo)

[Borning, 1996]

C = A + B
10 <= A <= 20
30 <= B <= 40

→

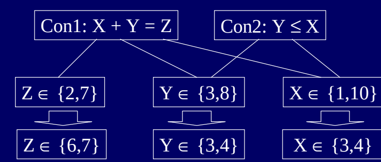
A.tighten(C.bounds - B.bounds)
B.tighten(C.bounds - A.bounds)
C.tighten(A.bounds - B.bounds)

→

A [10, 20]
B [30, 40]
C [40, 60]

- Acyclic graph
- Initially [-∞, +∞]
- Problem: division by zero
- Issue: single vs. multiple intervals
- Process strongest to weakest
- O(n*m) | n: variables, m: constraints
- One var. tightening per constraint (acyclic)

Interval Propagation [Davis87, Hyvönen92]



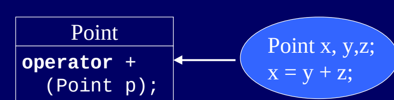
- Label refinement (Waltz)
- Deductively sound
- Finite set O(a*e) | a : domain size, e : constraints
- Label languages, constraint languages,

Systems & Constraint Imperative Programming

Constraints + Object-Orientation

[Wilk91, Benson92, Lopez94]

- Goals :
  - Preserve flexibility of modern OO languages
  - Constraints on object methods
  - Maintain imperative OO style
  - Solve useful collections of constraints
  - Use refinement method (v.s. perturbation)



Constraints + Object-Orientation

[Avesani90, Wilk91, Benson92, Lopez94]

- Integration Options:
  - Local propagation (known issues)
  - Constraints on primitive leaves [Avesani90]
  - New constraint solvers (per domain)
  - Graph rewriting [Wilk91]
  - Constraint constructors [Benson92]
  - Other
    - E.g., local propagation + iterative relaxation

Kaleidoscope'91 [Benson92]

- New OO language (multiple dispatching)
- Types as constraints
- Time & Assignment
  - Pellucid values (keeps current, previous)
  - Assignment: once constraint + weak stay
- Constraint constructors
  - Dispatch on each operator (no side effects)

solve type constraints → execute constraint constructors → solve primitive constraints

Future Work

Constraints & Network Mgmt

- Object-relationship configuration model
- Under-constrained system
- Policy directed change propagation
- Domains : integers, strings, relations
- Constraints : equality, interval, set membership, ...
- Expressing constraints & propagation policies
  - Graphical language ?