Constraint Satisfaction Problems

Constraint Optimization

Bernhard Nebel, **Julien Hué**, and **Stefan Wölfl**

Albert-Ludwigs-Universität Freiburg

July 17, 2012

Constraint Satisfaction Problems

July 17, 2012 — Constraint Optimization

1 Motivation

Hard and Soft Constraint

Real-life problems often contain hard and soft constraints:

Hard constraints: must be satisfied;

Soft constraints: should be satisfied, but may be violated.

Example: In time-tabling problems,

- resource constraints such as “a teacher can teach only one class at a time” must be satisfied;
- a request such as “the schedule of teacher should be concentrated in two days” is simply a preference, but not essential for the solution.

What to do with soft constraints?

Constraint Optimization

Formalizing problems with soft and hard constraints leads to constraint networks augmented with a global cost function (also called criterion function or objective function), based on the satisfaction of soft constraints.

A constraint optimization problem (COP) is the problem of finding a variable assignment to all variables that satisfies all hard constraints and at the same time optimizes the global cost function.

Note: Every constraint satisfaction problem can be viewed as a constraint optimization problem – when not all constraints are satisfiable. Try to find an assignment that maximizes the number of satisfied constraints: MAX-CSP problem.

Example 1: Power Plant Maintenance

Given

1. a number of power generators,
2. preventive maintenance intervals,
3. time for maintenance,
4. accurate estimates for plant's power demands,

determine a maintenance schedule respecting (2) that minimizes operating and maintenance costs.

Example 2: Combinatorial Auctions

In combinatorial auctions, bidders can give bids for sets of items. The auctioneer than has to generate an optimal selection, e.g., one that maximizes revenue.

Definition

The combinatorial auction problem is specified as follows:

Given: A set of items $Q = \{q_1, \ldots, q_n\}$ and a set of bids $B = \{b_1, \ldots, b_m\}$ such that each bid is $b_i = (Q_i, r_i)$, where $Q_i \subseteq Q$ and r_i is a strictly positive real number.

Task: Find a subset of bids $B' \subseteq B$ such that any two bids in B' do not share an item maximizing $\sum_{(Q_i, r_i) \in B'} r_i$.

2 Cost Networks

From Constraint to Cost Networks

- We will extend constraint networks to cost networks.
- Hard constraint are modelled as ordinary constraints, we know already.
- Soft constraints are modelled by cost functions, which assign particular costs to variable assignments.
- The costs are aggregated by a global cost function

Global Cost Functions

A constraint optimization problem (COP) is a constraint network extended by a global cost function.

Definition

Given a set of variables $V = \{v_1, \ldots, v_n\}$, a set of real-valued functions $F_1, \ldots, F_l$ over scopes $S_1, \ldots, S_l$, $S_j \subseteq V$, and assignments a over V. The global cost function F is defined by

$$F(a) = \sum_{j=1}^{l} F_j(a),$$

where $F_j(a)$ means F_j applied to assignments in a restricted to the scope of F_j, i.e., $F_j(a) = F_j(\bar{a}[S_j])$.

Cost Networks

Constraint optimization problems can be viewed as defined over an extended constraint network called cost network.

Definition

A cost network is a 4-tuple $\mathcal{O} = \langle V, \mathrm{dom}, C_h, C_s \rangle$, where $\langle V, \mathrm{dom}, C_h \rangle$ is a constraint network (elements of C_h are called hard constraints), and $C_s = \{F_1, \ldots, F_l\}$ is a set of real-valued functions defined over scopes $S_1, \ldots, S_l$ (elements of C_s are called soft constraints).

Definition

A solution to a constraint optimization problem given by a cost network $\mathcal{O} = \langle V, \mathrm{dom}, C_h, C_s \rangle$, is an assignment a^* that maximizes (minimizes) $F(a)$ among all assignments a that satisfy $\langle V, \mathrm{dom}, C_h \rangle$.

3 Soft constraints

Example: Pacman

. . . you have the following needs. In a given number of steps, find the optimal route that:

- catches as many red dots as possible;
- then catches as many green dots as possible;
- then catches as many blue dots as possible.

Pacman Solution: find a proper valuation

In a given number of steps (lets say 100):

- red dots, then green dots, then blue dots.

The proper valuation would be:

Blue dots are the less valuable.
Blue dot = 1 point.

One green dot worth more than all blue dots. The worst case forces us to consider.
Green dot = 101 points.

One red dot worth more than all green and all blue dots.
Red dot = 10201 points.

Example: Pacman

Big drawbacks:

- need preprocessing to compute valuation that represents correctly the problem;
- quickly comes up with very big integers.

The sum as an aggregator to cost function is not adapted here.

Possibilistic logic

- Two measures of consistency: necessity and possibility defined on [0,1];
- Necessity measures how forced the beliefs are;
- Possibility measures how compatible with the bases the beliefs are.

$$\Pi(\emptyset) = 0 \qquad \Pi(\Omega) = 1 \qquad \Pi(A \vee B) = Max(\Pi(A), \Pi(B))$$

$\Pi(A) = 0$ means A is impossible.
$\Pi(A) = 1$ means A is possible (does not mean it is true).

$$N(A) = 1 - \Pi(\neg A) \qquad N(A \wedge B) = Min(N(A), N(B))$$

$N(A) = 0$ means A is not forced (does not mean it is wrong).
$N(A) = 1$ means A is true.

Possibilistic cost function

A constraint optimization problem (COP) is a constraint network extended by a possibilistic cost function.

Definition

Given a set of variables $V = \{v_1, \ldots, v_n\}$, a set of real-valued functions $F_1, \ldots, F_l$ over [0,1], and assignments a over V. The possibilistic cost function F is defined by

$$F(a) = \max_{j=1}^{l} F_j(a),$$

The aim here is to find a solution whose most important violated constraints has the lowest necessity degree.

A more general framework: Valued constraints

Definition

A valuation structure is a tuple $\langle E, \oplus, \preccurlyeq_v, \bot, \top \rangle$ such that:

- E is a set, whose elements are called valuations, totally ordered by $\preccurlyeq_v$ with a maximum ($\top$) and a minimum ($\bot$).
- $\oplus$ satisfies:
  - commutativity: $a \oplus b = b \oplus a$,
  - associativity: $a \oplus (b \oplus c) = (a \oplus b) \oplus c$,
  - monotonocity: $(a \preccurlyeq_v b) \rightarrow ((a \oplus c) \preccurlyeq_v (b \oplus c))$,
  - neutral element: $a \oplus \bot = a$
  - annihilator: $a \oplus \top = \top$

Relations between frameworks

Semiring	E	$\times_s$	$+_s$	$\succcurlyeq_s$	0	1
Classical	$\{t, f\}$	$\wedge$	$\vee$	$t \succcurlyeq_s f$	f	t
Fuzzy	$[0, 1]$	min	max	$\geq$	0	1
k-weighted	$\{0, \ldots, k\}$	$+^k$	min	$\leq$	k	0
Probabilistic	$[0, 1]$	xy	max	$\geq$	1	0
Valued	E	$\oplus$	min_v	$\preccurlyeq_v$	$\top$	$\bot$

Still a lot of adaptable real-life concept

- Partially pre-ordered preferences;
- Conditional preferences;
- Stratified Constraint Networks
- . . .

Example: Cost Network for Combinatorial Auction

For a combinatorial auction given by item set $Q = \{q_1, \ldots, q_n\}$ and bids $B = \{b_1, \ldots, b_m\}$ with $b_i = (Q_i, r_i)$ define a cost network as follows:

- Variables b_i with domain $\{0, 1\}$; 1 for selecting the bid, 0 otherwise;
- For each pair b_i, b_j such that $Q_i \cap Q_j \neq \emptyset$ a constraint R_{ij} prohibiting that b_i and b_j are assigned 1 simultaneously;
- Cost functions F_i with $F_i(a) = r_i$ if $a(b_i) = 1$, $F_i(a) = 0$ otherwise, for an assignment a.

Find a consistent assignment a to the b_i s that maximizes $F(a) = \sum_i F_i(a)$.

Note: cost network = constraint network, because all cost components are unary.

Example Auction

Consider the following auction:

$$
\begin{aligned}
b_1 &= \{1, 2, 3, 4\}, & r_1 &= 8, \\
b_2 &= \{2, 3, 6\}, & r_2 &= 6, \\
b_3 &= \{1, 4, 5\}, & r_3 &= 5, \\
b_4 &= \{2, 8\}, & r_4 &= 2, \\
b_5 &= \{5, 6\}, & r_5 &= 2.
\end{aligned}
$$

What is the optimal assignment?

Reduction of COP-Solving to CSP-Solving

We can always reduce COP-solving to solving a sequence of CSPs.

Given a COP $\mathcal{O}$ which we want to maximize. Consider a sequence of CSPs $\mathcal{C}_i$, s. t. each contains the constraint part of $\mathcal{O}$ and an additional constraint $\sum_j F_j(a) \geq c_i$, where $c_1 \leq \ldots \leq c_i \leq \ldots$.

Solve the CSPs with increasing cost bounds c_i until no solution can be found. Then the previous step is the optimal solution – provided the difference between the steps is not larger than the smallest difference between different values of the global cost function.

Example: Solving the Auction Problem

Assumption: Step size 1 and static variable ordering b_1, b_2, b_3, b_4, b_5.

For cost bounds from $c_1 = 0$ to $c_9 = 8$, $a(b_1) = 1$ and all others 0 is satisfying.

For cost bound $c_{10} = 9$ and $c_{11} = 10$, $a(b_1) = 1$ and $a(b_5) = 1$ (and all others 0) is satisfying.

For cost bound $c_{12} = 11$, $a(b_2) = 1$ and $a(b_3) = 1$ (and all others 0) is satisfying.

For cost bound $c_{13} = 12$, there is no satisfying assignment.

4 Branch and Bound

- Bounding function

Branch and Bound: First idea

When solving a COP using a sequence of CSPs, one could use all CSP techniques. However, instead of solving multiple CSPs, one may instead want to integrate the optimization process into the search process.

First idea:

1. Set bound $c = 0$.
2. Use any systematic search technique to find an assignment that satisfies the constraint part.
3. Remember solution in a and global cost in c if global cost $> c$.
4. Return a and c if no further solutions can be found, otherwise continue with next solution at (3).

Pruning

Of course, often it is possible to prune the search, even if no inconsistency has been detected yet.

Main idea behind depth-first branch-and-bound (BnB):
If the best solution so far is c, this is a lower bound for all other possible solutions. So, if a partial solution has led to costs of x for all cost components of fully instantiated variables and the best we can achieve for all other cost components is y with $x + y < c$, then we do not need to continue in this branch.

How can we find out what is the best we can achieve?

Bounding Evaluation Function

In the following, we will write $\vec{a_i}$ for partial instantiations of the first i variables, assuming a static variable ordering.

Definition

A bounding evaluation function for a maximizing (minimizing) constraint optimization problem is a function f over partial assignments such that $f(\vec{a_i}) \geq max_a F(a)$ ($f(\vec{a_i}) \leq min_a F(a)$) for all satisfying assignments a that extend $\vec{a_i}$.

Note:

- If $f(\vec{a_i}) < c$ for some already found solution c, then $\vec{a_i}$ cannot be extended to a maximal solution.
- f can also be used as a heuristic for choosing a value of the next variable!

Branch and Bound (BnB) Algorithm

BnB($\mathcal{O}, f$):

Input: cost network $\mathcal{O}$ and evaluation bounding function f
Output: an optimal assignment a' (possibly empty) and costs c'

```
∀i D'_i ← D_i, i ← 1, c' ← 0, a' ← ∅, a ← ∅
while (i ≠ 0)
    while (1 ≤ i ≤ n)
        remove (v_i ↦ _) from a // remove old assignment to v_i
        x ← SelectValue(i, c')
        if (x = null) D'_i ← D_i // no value for x_i: reset domain
            i ← i − 1 // backtrack
        else a ← a ∪ {v_i ↦ x}
            i ← i + 1 // step forward
    if (i = n + 1) // one solution found
        if (F(a) > c') // better solution
            a' ← a // remember best solution found so far
            c' ← F(a))
        i ← n // search for next solution
return(a', c')
```

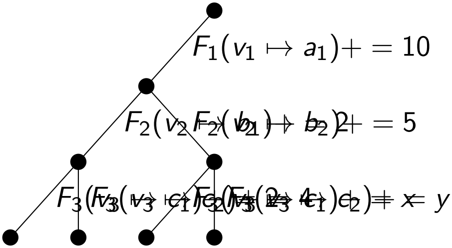
Branch and Bound Algorithm: SelectValue

SelectValue(i, c'):

while $(D'_i \neq \emptyset)$
    select $a_i^* \in D'_i$ such that
        $a_i^* = \text{pick one } \arg\max_{a_i \in D'_i} f(a \cup \{v_i \mapsto a_i\})$
    remove a_i^* from D'_i
    if $(a \cup \{v_i \mapsto a_i^*\})$ is consistent **and**
        $f(a \cup \{v_i \mapsto a_i^*\}) > c'$) **return**($a_i^*$)
return(*null*)

Bounding function - Introduction

Random "Minizing" problem



$$S = \{(v_1 \mapsto a_1, v_2 \mapsto b_1, v_3 \mapsto c_1) = 14\}$$

First-Choice Bounding Function

How to come up with a good bounding evaluation function?

In *Operation Research*, one often uses Linear Programming to come up with bounds for Integer Programming Problems.

Let us consider what we can achieve for all soft constraints in isolation subject to the partial assignment we have already. This function is called first-choice (fc) bounding function:

$$f_{fc}(\vec{a_i}) = \sum_{F_j \in C_s} \max_{a_{i+1},\ldots,a_n} F_j(\vec{a_i} \cup \{v_{i+1} \mapsto a_{i+1}, \ldots, v_n \mapsto a_n\})$$

How could one improve on that?

- Only allow locally consistent partial assignments.
- Do not consider all soft constraints in isolation, but combine them!

Example: Auction again

Let us consider BnB with the first-choice bounding function on our auction example:

1. $f_{fc}(\{b_1 \mapsto 1\}) = 8 + (6 + 5 + 2 + 2) = 23$
2. $f_{fc}(\{b_1 \mapsto 1, b_2 \mapsto 0\}) = 8 + (5 + 2 + 2) = 17$
3. $f_{fc}(\{b_1 \mapsto 1, b_2 \mapsto 0, b_3 \mapsto 0\}) = 8 + (2 + 2) = 12$
4. $f_{fc}(\{b_1 \mapsto 1, b_2 \mapsto 0, b_3 \mapsto 0, b_4 \mapsto 0\}) = 8 + (2) = 10$
5. ...

Russian Doll Search: Idea

One way to get more accurate bounding functions is to solve subproblems and store the optimal results, reusing them for larger problems.

Solve a sequence of n problems using BnB, where in the ith run the last i variables, i.e., v_{n-i+1} up to v_n, (and the relevant hard and soft constraints) are considered.

The results of the previous runs can be used:

1. as an initial lower bound,
2. in a heuristic for choosing values, and
3. to generate a more accurate bounding function.

Improving the Evaluation Function

- Solve n COPs $\mathcal{O}_i, (i = 1, \ldots, n)$ over the last i variables $v_{n-i+1}, \ldots, v_n$ using BnB and store maximal costs as c_i^*.
- In the $(n - i + 1)$th run, variables $v_i, \ldots, v_n$ are considered.
- Assume that the variables $v_i, \ldots, v_{i+j}$ are instantiated, denoted by the partial assignment $\vec{a_j^i}$, and that $C_{i,j}$ are all those soft constraints F such that their scopes have a non-empty intersection with $\{v_i, \ldots, v_{i+j}\}$.
- Then we use the optimal costs from the $n - i - j$th run to improve on the first-choice function:

$$f(\vec{a_j^i}) = c_{n-i-j}^* + \sum_{F \in C_{i,j}} \max_{a_{i+j+1}, \ldots, a_n} F(\{v_i \mapsto a_i, \ldots, v_n \mapsto a_n\}).$$

5 Bucket Elimination

General Idea

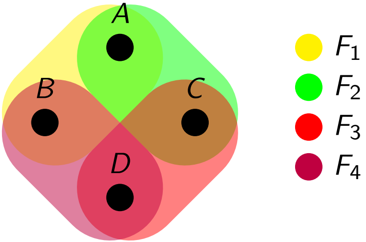
- Reformulation of adaptive consistency
- Process constraints to remove variables one by one
- Still exponential in the size of the constraints

Algorithm

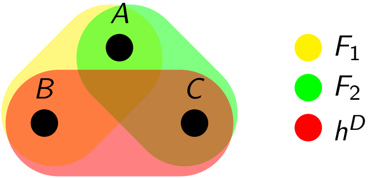
Algorithm 1 ELIM-OPT

Partition the constraints
for $p = n$ to 1 **do**
  $U_p = \cup_i S_i - x_p$
  $C^p = \pi_{U_p}(\bowtie_{i=1}^{t} C_i)$
  for Every tuple t over U_p **do**
    $h^p = \max_{\{a_p|(t,a_p) \text{ satisfies } C_i\}} \sum_{i=1}^{j} h_i(t, a_p)$
    Place h^p in latest bucket mentioning a variable in U_p
  end for
end for

Bucket Elimination: Example



A	B	$F_1(A,B)$	A	C	$F_2(A,C)$
1	1	2	2	2	3
2	4	0	1	3	0
2	–	0	1	–	0
–	4	0	–	3	0
–	–	0	–	–	0



C	D	$F_3(C,D)$	B	D	$F_4(B,D)$
3	3	4	4	4	4
2	4	0	1	3	0
2	–	0	1	–	0
–	4	0	–	3	0
–	–	0	–	–	0

C	D	$F_3(C,D)$	B	D	$F_4(B,D)$
3	3	4	4	4	4
2	4	0	1	3	0
2	–	0	1	–	0
–	4	0	–	3	0
–	–	0	–	–	0

$F_3(C,D) + F_4(B,D)$

Conclusion & Outlook

- Problems with hard and soft constraints lead to constraint optimization problems
- These are formalized using cost functions and cost networks
- They can be solved using a reduction to a sequence of CSP problems
- More efficiently, one can search for optimal solutions during the backtracking search
- Branch and Bound is the method of choice
- Its pruning power depends on the accuracy of the bounding evaluation function
- Russian doll search can boost its performance
- Further enhancements are possible using constraint inference techniques (such as bucket elimination).

Literature

Rina Dechter.
Constraint Processing,
Chapter 13, Morgan Kaufmann, 2003