Towards an Optimal Approach to Soft Constraint Problems

Martine Ceberio and Vladik Kreinovich
Department of Computer Science
University of Texas at El Paso, 500 W. University
El Paso, TX 79968, USA, {mceberio,vladik}@cs.utep.edu

1. Design and Control Problems

- In many areas of science and engineering, we are interested in solving *design* and *control* problems.
- *In mathematical terms:* a design or a control can be usually represented by the values of the relevant numerical parameters $x = (x_1, \ldots, x_n)$.
- Usually, in these problems, the users describe several *constraints* that the desired design or control must satisfy.
- *Objective:* find a design (corr., a control) that satisfies all these constraints.

2. How to Describe Constraints?

- *Example:* an airplane design can be described in terms of:
  - the geometric parameters of the plane,
  - the thickness of the plates that form the airplane's skin,
  - the weight and power of the engine, etc.
- *Typical constraint:* a limitation on some characteristics $y = f(x_1, \ldots, x_n)$ of this design.
- *Examples*
  - the airplane's speed must exceed some y_0,
  - its fuel use must not exceed a certain amount,
  - the overall cost must be within given limits.
- So, constraints are of the type $f(x_1, \ldots, x_n) \le y_0$ or $f(x_1, \ldots, x_n) \ge y_0$ (or $f(x_1, \ldots, x_n) = y_0$).

3. Constraint Satisfaction vs. Constrained Optimization

- *Constraint satisfaction:* find a design that satisfies given constraints.
- *Problem:*
  - different designs that satisfy the given constraints;
  - we must select one of these designs.
- Users can often describe their preference in terms of an *objective function* $g(x_1, \ldots, x_n)$ (whose value should be made as large as possible).
- *Constrained optimization:* maximizing $g(x_1, \ldots, x_n)$ under the given constraints.
- *In general:* both problem are NP-hard.
- *In practice:* there are many efficient tools for solving them.

4. What Are Soft Constraints?

- *Problem:* sometimes, the users constraints are inconsistent.
- *Example:* design a plane that is:
  - as fast and as fuel-efficient as the existing Airbus or Boeing planes,
  - but with 0 noise level.
- *Reasons for inconsistency:*
  - some constraints are *absolute* (e.g., safety constraints),
  - others are *desires* – they can be dismissed if not possible.
- Such "not required" constraints are called *soft constraints*.
- *Comment:* soft constraints are an important research topic, with annual conferences.

5. Priority Approach to Soft Constraints: A Brief Description

- *Idea:*
  - when we cannot satisfy *all* the constraints,
  - we should satisfy as *many* constraints as possible.
- *Natural idea:*
  - ask the user to *prioritize* their constraints C_i, from the absolutely required to the less required:

$$C_1 \succ C_2 \succ \ldots \succ C_n;$$

  - find the largest possible value $k = k_{\text{opt}}$ for which all the constraints $C_1, C_2, \ldots, C_k$ are still consistent.

6. A Computational Question

- Constraint satisfaction tools can check consistency.
- *Possibility:* apply a tool to $\{C_1\}$, $\{C_1, C_2\}$, ..., until we get inconsistency.
- *Problem:* for large number of constraints ($1 \ll k_{\rm opt} \ll n$), we need too many iterations.
- *Alternative idea:* use iterative bisection:
  - at each stage, we have an interval $[k^-, k^+] \ni k_{\rm opt}$;
  - initially, $k^- = 0$ and $k^+ = n$;
  - at each stage, we check consistency for the midpoint

$$k_m \stackrel{\rm def}{=} \lfloor (k^- + k^+)/2 \rfloor,$$

    and replace the interval with a half-size one: $[k^-, k_m]$ or $[k_m, k^+]$.
- *Problem:* on some stages, too many ($n \gg k_{\rm opt}$) constraints, takes too long.
- *Question:* which method for finding $k_{\rm opt}$ is optimal?

7. Priority Approach to Soft Constraints: Toward Formalizing the Computational Question

- *Definition:* A *method* is a mapping that maps each pair (I, s), where:
  - I is an integer-valued interval $[k^-, k^+]$, where $0 \le k^- < k^+ \le n$ and $k^+ > k^- + 1$, and
  - s is a positive integer (= number of step)

  into an integer k_{next} from the open interval (k^-, k^+).
- *Meaning:* first k^- constraints are consistent, but first k^+ constraints are not.
- *Examples:*
  - sequential search: $k_{\text{next}} = k^- + 1$;
  - bisection: $k_{\text{next}} = \lfloor (k^- + k^+)/2 \rfloor$.

8. When Is a Method Optimal?

- *Known fact:* constraint satisfaction is NP-hard.
- *Meaning:* crudely speaking, computational complexity of a system of k constraints is $\sim 2^k$.
- More precisely: $t \sim p^k$ for $p \geq 2$.
- *Assumption:* $t = C \cdot p^k$.
- *Definition:* for a method M, $T_M(k)$ is defined as the worst-case overall time this method spends on checking when $k_{\text{opt}} = k$.
- *Ideal case:* we only check that k are consistent and $k+1$ are not, with time $p^k + p^{k+1}$.
- *Overhead:* $O_p(M) \stackrel{\text{def}}{=} \max_k \dfrac{T_M(k)}{p^k + p^{k+1}}$.
- *Objective:* we want to find a method M_{opt} with the smallest overhead, i.e., for which $O_p(M_{\text{opt}}) = \min_M O_p(M)$.

9. Main Result

- *Theorem.* For every $p \geq 2$, the sequential search method S is optimal.
- *Proof.* For sequential search S,

$$T_S(k) = p + p^2 + \ldots + p^{k+1} =$$

$$p^{k+1} \cdot (1 + p^{-1} + p^{-2} + \ldots + p^{-k}) <$$

$$p^{k+1} \cdot (1 + p^{-1} + p^{-2} + \ldots) = \frac{p^{k+1}}{(1 - p^{-1})}.$$

Since $p^k + p^{k+1} = p^{k+1} \cdot (1 + p^{-1})$, we conclude that

$$O_p(S) < \frac{1}{(1 - p^{-1}) \cdot (1 + p^{-1})}.$$

- In a method $M \neq S$, there exists an interval in which $k_{\text{next}} \geq k^- + 2$.
- So, if $k_{\text{opt}} = k^-$, we check both k and $\geq k + 2$; we also check $k + 1$ – to make sure that k is indeed the largest.

10. Proof (cont-d)

- Hence, $T_M(k) \geq p^k + p^{k+1} + p^{k+2}$, hence

$$O_p(M) \geq \frac{T_M(k)}{p^{k+1} \cdot (1 + p^{-1})} \geq \frac{p + 1 + p^{-1}}{1 + p^{-1}}.$$

- To complete our proof, we must show that

$$\frac{p + 1 + p^{-1}}{1 + p^{-1}} \geq \frac{1}{(1 - p^{-1}) \cdot (1 + p^{-1})} (\geq O_p(S)).$$

  Multiplying both sides of this inequality by the denominator of the RHS and by p^2, we get

$$p^3 - p^2 - 1 \geq 0.$$

- The equation $p^3 - p^2 - 1 = 0$ has a solution $p_0 \approx 1.47$.
- For $p \geq p_0$, its left-hand side is an increasing function – since its derivative is

$$3p^2 - 2p = (3p - 2) \cdot p > 0.$$

- Thus, $p^3 - p^2 - 1 > 0$ for all $p \geq p_0$ – in particular, for all $p \geq 2$. Q.E.D.

11. Acknowledgments

- This work was supported in part:
  - by NASA under cooperative agreement NCC5-209,
  - by NSF grants EAR-0112968, EAR-0225670, and EIA-0321328, and
  - by NIH grant 3T34GM008048-20S1.
- This research was partly done when Martine Ceberio was a Visiting Researcher at NII, Tokyo, Japan.
- The authors want to thank Seetharami R. Seelam for his help.

12. What If Constraints are not Prioritized?

- *Problem:* what if our constraints are not prioritized?
- *Discussion:* all the constraints are equally important.
- *Idea:* satisfy as many constraints as possible.
- *Result:* we will show that the corresponding problem if computationally difficult (NP-hard).
- *More detailed result:* the problem is NP-hard even when all the constraints are of the simplest possible type – linear equations.
- *Problem:* given a_{ij}, b_i, and $\varepsilon \in (0, 1)$, and constraints

$$\sum_{j=1}^{n} a_{ij} \cdot x_j = b_i, \quad i = 1, \ldots, N$$

check whether we can select a consistent set of $N \cdot (1 - \varepsilon)$ constraints.

13. Proof of NP-hardness

- *Idea:* reduce to a known NP-hard problem.
- *Subset sum:* given positive integers $s_1, \ldots, s_n$, and s, check whether $s = \sum_{i=1}^{n} x_i \cdot s_i$ for some $x_i \in \{0, 1\}$.
- *Reduction:* $N = n/\varepsilon$ constraints:
  - $2n$ constraints $x_1 = 0$, $x_1 = 1 \ldots$, $x_n = 0$, $x_n = 1$;
  - $N - 2n$ identical constraints $\sum s_i \cdot x_i = s$.
- Since $0 \neq 1$, at most $N - n$ are satisfied.
- If the subset problem has a solution, then:
  - all $N - 2n$ constraints $\sum s_i \cdot x_i = s$ are satisfied,
  - and for each i, $x_i = 0$ or $x_i = 1$,

  to the total of $N - n = N \cdot (1 - \varepsilon)$.
- If $N - n$ constraints are satisfied, then for every i, $x_i \in \{0, 1\}$ – a solution to the subset problem.

14. Constraint Propagation: Reminder

- Constraint propagation – traditional technique for solving constraint satisfaction problems.
- We start with the intervals $[\underline{x}_1, \overline{x}_1], \ldots, [\underline{x}_n, \overline{x}_n]$ containing the actual values of the unknowns $x_1, \ldots, x_n$.
- On each iteration:
  - select i and a constraint $f_j(x_1, \ldots, x_n) = 0$,
  - replace $[\underline{x}_i, \overline{x}_i]$ with new interval $\mathbf{x}_i^{(j)} = [\underline{x}_i^{(j)}, \overline{x}_i^{(j)}] \stackrel{\text{def}}{=}$

$$\{x_i : x_i \in [\underline{x}_i, \overline{x}_i] \,\&\, f_j(x_1, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots, x_n) = 0$$
$$\text{for some } x_k \in [\underline{x}_k, \overline{x}_k]\}.$$

- If the process stalls, we bisect the interval for one the variables into two and try to decrease both resulting half-boxes.
- *Problem:* cannot use it if not all constraints are valid.

15. New Idea

- On each iteration, we still select a variable x_i, but:
  - instead of selecting a *single* constraint,
  - we try *all* N constraints, and get N resulting intervals $[\underline{x}_i^{(j)}, \overline{x}_i^{(j)}]$.
- We know that $\geq N \cdot (1-\varepsilon)$ constraints are satisfied.
- Hence $x_i \leq \overline{x}_i^{(j)}$ for $\geq N \cdot (1-\varepsilon)$ different values j.
- Let us sort all N upper endpoints $\overline{x}_i^{(j)}$ $(1 \leq j \leq N)$ into an increasing sequence $u_1 \leq u_2 \leq \ldots \leq u_N$,
- Then we can guarantee that x_i is smaller than (or equal to) at least $N \cdot (1-\varepsilon)$ terms in this sequence.
- So, $x_i \leq u_{N \cdot \varepsilon}$.
- Similarly, if we sort the lower endpoints $\underline{x}_i^{(j)}$ into a decreasing sequence $l_1 \geq \ldots \geq l_n$, then $x_i \geq l_{N \cdot \varepsilon}$.

16. New Algorithm

- On each iteration, we:
  - we select a variable x_i;
  - for each of N constraints, we compute the corresponding interval $[\underline{x}_i^{(j)}, \overline{x}_i^{(j)}]$;
  - we sort all N upper endpoints $\overline{x}_i^{(j)}$ $(1 \le j \le N)$ into an increasing sequence $u_1 \le u_2 \le \ldots \le u_N$,
  - we sort all N lower endpoints $\underline{x}_i^{(j)}$ $(1 \le j \le N)$ into a decreasing sequence $l_1 \ge l_2 \ge \ldots \ge l_N$, and
  - we take $[l_{N\cdot\varepsilon}, u_{N\cdot\varepsilon}]$ as the new interval for x_i.
- If the process stalls, we bisect the interval and try to decrease both resulting half-boxes.
- *Comment:* producing $u_{N\cdot\varepsilon}$ can be done faster than sorting;