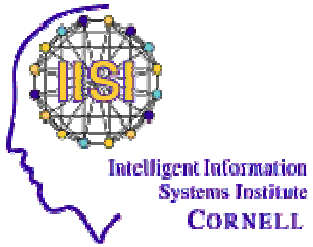
Soft Global Constraints

Constraint Programming Summer School on Global Constraints – June 18-23, 2006

Willem-Jan van Hoeve

IISI
Cornell University
U.S.A.
vanhoeve@cs.cornell.edu



© Copyright Willem-Jan van Hoeve, 2006

Outline

Cornell University

Lecture 1a:

- Constraint Programming in brief
  - basic concepts
  - domain filtering
- Soft Constraints
  - motivation
  - traditional approaches
  - cost-based approach
- Outlook

Lecture 1b:

- Soft alldifferent
  - two violation measures: variable-based and decomposition-based
  - filtering algorithm for variable-based violation measure using matchings
  - intermezzo: flow theory
  - alternative representation using flows
  - filtering algorithm for decomposition-based violation measure using flows

Outline

Cornell University

Lecture 2a:

- Soft global cardinality constraint
  - filtering algorithm for variable-based violation measure using flows
  - filtering algorithm for value-based violation measure using flows
  - alternative approach

Lecture 2b:

- Soft regular constraint
  - filtering algorithm for variable-based violation measure using shortest paths
- Conclusions and perspectives

Constraint Programming

Constraint Programming in brief

Cornell University

A Constraint Satisfaction Problem, or CSP, consists of

- a set of variables X,
- variable domains D(x) (for all $x \in X$),
- and a set of constraints on subsets of the variables

A solution to a CSP is:

assign to each variable a single element from its domain

such that all constraints are satisfied

Example:

variables	x_1, x_2, x_3
domains	$D(x_1) = \{1,2\}$, $D(x_2) = \{0,1,2,3\}$, $D(x_3) = \{2,3\}$
constraints	$x_1 > x_2$
	$x_1 + x_2 = x_3$
solution	$x_1 = 2, x_2 = 0, x_3 = 2$

Constraint Programming in brief

Cornell University

A Constraint Optimization Problem, or COP, consists of

- a set of variables X,
- variable domains D(x) (for all $x \in X$),
- a set of constraints on subsets of the variables,
- and an objective function $f(X) \rightarrow \mathbb{Q}$ to be optimized

A solution to a COP is:

assign to each variable a single element from its domain

such that all constraints are satisfied, and the objective function is optimal

Example:

variables/domains	$x_1 \in \{1,2\}$, $x_2 \in \{0,1,2,3\}$, $x_3 \in \{2,3\}$
constraints	$x_1 > x_2$
	$x_1 + x_2 = x_3$
objective function	maximize $x_2 + x_3$
solution	$x_1 = 2$, $x_2 = 1$, $x_3 = 3$

Constraint Programming in brief

Cornell University

A constraint programming solver must

- find a (optimal) solution to the CSP (COP)
- or prove that none exists

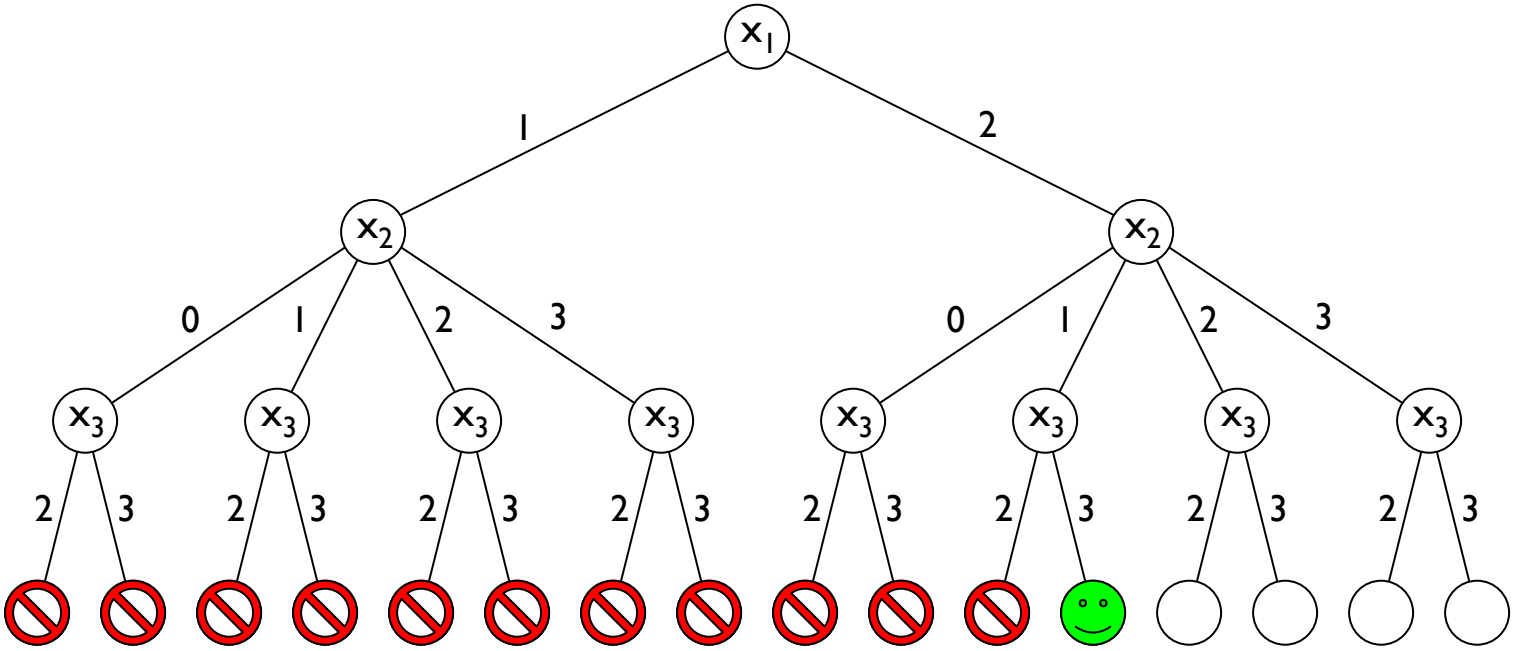
It does so by

- complete search
- domain filtering
- constraint propagation

Example:

variables/domains	$x_1 \in \{1,2\}$, $x_2 \in \{0,1,2,3\}$, $x_3 \in \{2,3\}$
constraints	$x_1 > x_2$
	$x_1 + x_2 = x_3$
	alldifferent(x_1,x_2,x_3)

complete search (no filtering or propagation)



Constraint Programming in brief

Cornell University

Example:

variables/domains $x_1 \in \{1,2\}$, $x_2 \in \{0,1,2,3\}$, $x_3 \in \{2,3\}$

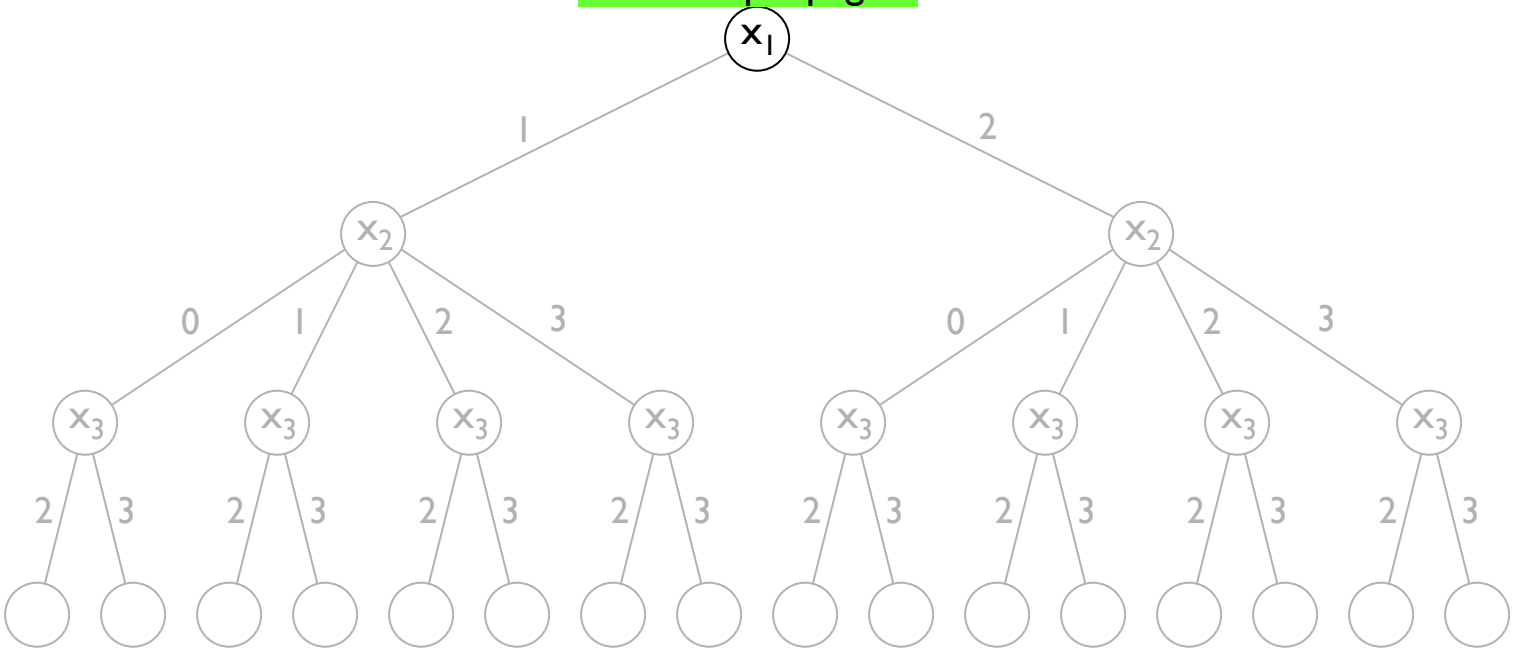
constraints $x_1 > x_2$

$x_1 + x_2 = x_3$

alldifferent(x_1,x_2,x_3)

complete search + **filtering** and **propagation**

filter and propagate



Example:

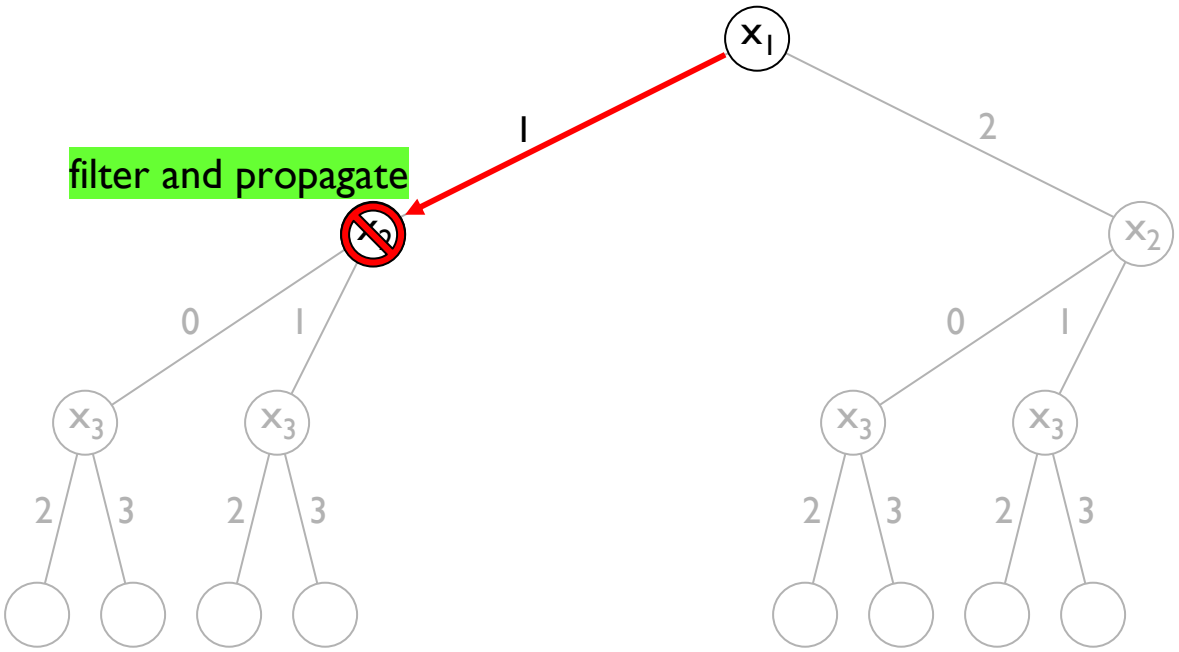
variables/domains $x_1 \in \{1,2\}, x_2 \in \{0,1\}, x_3 \in \{2,3\}$

constraints $x_1 > x_2$

$x_1 + x_2 = x_3$

alldifferent(x_1,x_2,x_3)

complete search + **filtering** and **propagation**



Example:

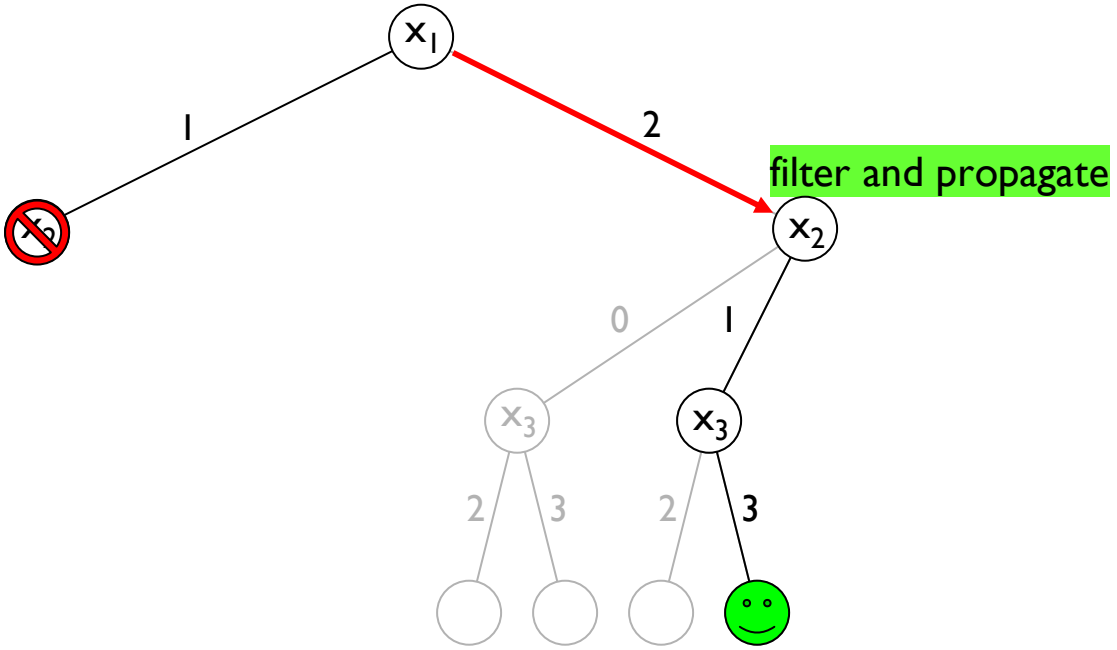
variables/domains $x_1 \in \{1,2\}, x_2 \in \{0,1\}, x_3 \in \{2,3\}$

constraints $x_1 > x_2$

$x_1 + x_2 = x_3$

alldifferent(x_1,x_2,x_3)

complete search + **filtering** and **propagation**



Constraint Programming in brief

Messages:

In order to speed up constraint programming solver

- always try to filter the domains using the constraints!
- more filtering is always better (given same time complexity)
- try to find efficient filtering algorithms (e.g., linear in size of constraint)
- if possible, make filtering algorithm incremental

Global constraints capture special structure of problem

- try to exploit structure for better/more filtering

Soft Constraints

Soft constraints

Cornell University

Over-constrained and preference-based problems:

- assign seats for over-booked airplane; no solution that carries all passengers
- create roster for nurses with conflicting preferences (e,g., they all want the same day off during soccer World Championship)
- factory wants to satisfy demands of all customers, but has limited resources
- ...

Estimate [anonymous]:

"At least 60% of all industrial problems are essentially over-constrained"

CP solver will report that no solution exists. How to find satisfying 'solution'?

- soften (some of) the constraints of the problem
- compute solution that minimizes conflicts or maximizes satisfaction

Soft constraints

Cornell University

A hard constraint must always be satisfied.

A soft constraint does not need to be satisfied, but we would like it to be.

Example:

Over-constrained CSP with two soft constraints:

$x_1 \in \{1,2\}$, $x_2 \in \{2,3\}$, $x_3 \in \{2,3\}$

(i) $x_1 > x_2$

(ii) $x_1 + x_2 = x_3$

$(x_1,x_2,x_3) = (1,2,2)$ both constraints violated

$(x_1,x_2,x_3) = (1,2,3)$ constraint (i) violated ← minimum violation or maximum satisfaction

$(x_1,x_2,x_3) = (1,3,2)$ both constraints violated

$(x_1,x_2,x_3) = (1,3,3)$ both constraints violated

$(x_1,x_2,x_3) = (2,2,2)$ both constraints violated

$(x_1,x_2,x_3) = (2,2,3)$ both constraints violated

$(x_1,x_2,x_3) = (2,3,2)$ both constraints violated

$(x_1,x_2,x_3) = (2,3,3)$ both constraints violated

Soft constraints

Cornell University

"Traditional" approaches to handle soft CSPs:

- Partial CSP [Freuder & Wallace, 1992]

  maximize number of satisfied constraints (previous example)

- Weighted CSP [Larossa, 2002]

  associate a weight to each constraint

  maximize weighted sum of satisfied constraints

- Possibilistic CSP [Schiex, 1992]

  associate weight to each constraint representing its importance

  hierarchical satisfaction of most important constraints, i.e. maximize smallest weight over all satisfied constraints

- Fuzzy CSP [Dubois et al., 1993] [Ruttkay, 1994]

  associate weight to each tuple of every constraint

  maximize smallest preference level (over all constraints)

These approaches can be modeled using valued CSPs [Schiex et al., 1995] or semi-rings [Bistarelli et al., 1997]

Drawbacks of traditional approaches:

- often coarse violation measure, especially for global/non-binary constraints

Example (cont'd):

$x_1 \in \{1,2\}$, $x_2 \in \{2,3\}$, $x_3 \in \{2,3\}$

(i) $x_1 > x_2$ new violation = deficit (i.e. gap between x_1 and x_2+1)

(ii) $x_1 + x_2 = x_3$ new violation = deficit (i.e. gap between x_1+x_2 and x_3)

evaluation using sum of violations:

$(x_1,x_2,x_3) = (1,2,2)$ violation is 2+1=3

$(x_1,x_2,x_3) = (1,2,3)$ violation is 2+0=2

$(x_1,x_2,x_3) = (1,3,2)$ violation is 3+2=5

$(x_1,x_2,x_3) = (1,3,3)$ violation is 3+1=4

$(x_1,x_2,x_3) = (2,2,2)$ violation is 1+2=3

$(x_1,x_2,x_3) = (2,2,3)$ violation is 1+1=2

$(x_1,x_2,x_3) = (2,3,2)$ violation is 2+3=5

$(x_1,x_2,x_3) = (2,3,3)$ violation is 2+2=4

minimum violation or maximum satisfaction

the new violation measures allow better evaluation of different tuples!

Drawbacks of traditional approaches:

- often coarse violation measure, especially for global/non-binary constraints
- little domain filtering & propagation

Example (cont'd):

$x_1 \in \{1,2\}$, $x_2 \in \{2,3\}$, $x_3 \in \{2,3\}$

(i) $x_1 > x_2$ violation = deficit (i.e. gap between x_1 and x_2+1)

(ii) $x_1 + x_2 = x_3$ violation = deficit (i.e. gap between x_1+x_2 and x_3)

$(x_1,x_2,x_3) = (1,2,2)$ violation is $2+1=3$

$(x_1,x_2,x_3) = (1,2,3)$ violation is $2+0=2$

$(x_1,x_2,x_3) = (1,3,2)$ violation is $3+2=5$

$(x_1,x_2,x_3) = (1,3,3)$ violation is $3+1=4$

$(x_1,x_2,x_3) = (2,2,2)$ violation is $1+2=3$

$(x_1,x_2,x_3) = (2,2,3)$ violation is $1+1=2$

$(x_1,x_2,x_3) = (2,3,2)$ violation is $2+3=5$

$(x_1,x_2,x_3) = (2,3,3)$ violation is $2+2=4$

suppose maximum violation is 3,
then value 3 in domain of x_2 has no support.
we want to remove this inconsistent value!

Soft constraints

Cornell University

Drawbacks of traditional approaches:

- often coarse violation measure, especially for global/non-binary constraints
- little domain filtering & propagation
- how can we apply this in traditional CP framework?
  - constraint programming solvers do not support soft constraints

Alternative approach [Baptiste et al., 1998], [Régin et al., 2000]:

- introduce a cost variable for each soft constraint
- this variable represents some violation measure of the constraint
- optimize aggregation of all cost variables (e.g., take their sum)
- use upper bound on cost variable to apply cost-based filtering

Note: In this way

- soft CSPs become COPs,
- soft global constraints become optimization constraints,
- we can apply classical constraint programming solvers.

Softening global constraints

Example:

soft CSP:

$x_1 \in \{0,1,2,3\}$, $x_2 \in \{0,1\}$

$x_1 \leq x_2$ (soft constraint)

i) introduce cost variable z for the soft constraint $x_1 \leq x_2$

ii) z represents the difference between x_1 and x_2 if it is violated (violation measure)

iii) minimize z

new COP:

$x_1 \in \{0,1,2,3\}$, $x_2 \in \{0,1\}$, $z \in \{0,1,2,3\}$

$((z = 0) \wedge (x_1 \leq x_2)) \vee ((z > 0) \wedge (z = x_1 - x_2))$

minimize z

cost-based domain filtering:

suppose $z \in \{0,1\}$, then $x_1 \leq 2$

Softening global constraints

Cornell University

Notation: $X = \{x_1, x_2, \ldots, x_n\}$, $D(X) = \cup_{x \in X} D(x)$

General recipe for global constraint C(X):

define violation measure μ: $D(x_1) \times D(x_2) \times \cdots \times D(x_n) \rightarrow \mathbb{Q}_+$

and cost variable z

then soften the constraint C as:

$soft\_C(X,z,\mu) := (\mu(X) \leq z)$

where $\mu=0$ iff C is satisfied and $\mu>0$ otherwise

Example (revisited):

$x_1 \in \{0,1,2,3\}$, $x_2 \in \{0,1\}$

constraint $c(x_1,x_2) = (x_1 \leq x_2)$ to be softened

add cost variable $z \in \{0,1,2,\ldots\}$

define violation measure $\mu(x_1,x_2) = \max\{x_1 - x_2, 0\}$

soften c as: $soft\_c(x_1,x_2,z,\mu) = (\mu(x_1,x_2) \leq z)$

Filtering soft global constraints

Basic cost-based filtering algorithm for soft_C(X,z,μ) = (μ(X) ≤ z) :

for all x∈ X and d∈ D(x) {
  compute minimum of μ with x=d
  if minimum > max(D(z)) remove d from D(x)
  if D(x) empty **return** inconsistent
}
update min(D(z)) ≥ min(all minima of μ)
if D(z) empty **return** inconsistent
else return consistent

Notes:

- similar to classical filtering for inequality constraint
- filtering establishes arc consistency[1] on soft_C, i.e., we remove all domain values that are not part of a solution to the constraint (this is the best possible)
- different violation measures define different soft constraints, and thus different filtering algorithms
- time complexity depends on computation of minimum of μ

[1] also referred to as hyper-arc consistency, generalized arc consistency, or domain consistency in the literature

Softening global constraints

Remark: soft constraint previously defined as

soft_C(X,z,μ) := (μ(X) ≤ z)

Why not

soft_C(X,z,μ) := (μ(X) **=** z) ?

Because then arc consistency becomes often NP-hard (for 'reasonable' violation measure μ), while inequality makes it often tractable

Alternative view:

- use equality in definition, but
- establish arc consistency w.r.t. X and 'bound consistency' w.r.t. z
- (effectively the same approach as with inequality)

Softening global constraints

Cornell University

Our goal is to design efficient filtering algorithms for soft global constraints, that establish arc consistency, using cost-based approach

Outlook:

constraint	violation measure	consistency check	arc consistency
hard alldifferent	-	O(m√n)	O(m)
soft alldifferent	variable-based	O(m√n)	O(m)
soft alldifferent	decomposition-based	O(mn)	O(m)
gcc	-	O(m√n)	O(m)
soft gcc	variable-based	O(m√n)	O(m)
soft gcc	value-based	O(m√n)	O(m)
regular	-	O(m)	O(m)
soft regular	variable-based	O(m)	O(m)

Here n is the number of variables, m is proportional to the sum of the domain sizes.

Exercises

Cornell University

Exercise 1: Consider the following over-constrained CSP

$x_1 \in \{1,2\}$, $x_2 \in \{2,3\}$, $x_3 \in \{2,3\}$, $x_4 \in \{1,2,3\}$

$x_1 \geq x_2$

$2x_1 + x_2 < x_3 + x_4$

$x_2 \neq x_3$

$x_3 \neq x_4$

a. compute a solution to the CSP using the Partial CSP approach

b. apply the cost-based approach to the CSP

- define a suitable violation measure for each constraint
- choose a suitable aggregation of cost variables
- compute a solution

Filtering algorithms for the soft alldifferent constraint

Softening alldifferent

Example:

$x_1 \in \{a,b\}$, $x_2 \in \{a,b\}$,
$x_3 \in \{a,b\}$, $x_4 \in \{b,c\}$,
alldifferent(x_1,x_2,x_3,x_4)

Over-constrained CSP; we need to soften the alldifferent constraint:

introduce cost variable z
define some violation measure, say μ
minimize z

Result:

$x_1 \in \{a,b\}$, $x_2 \in \{a,b\}$,
$x_3 \in \{a,b\}$, $x_4 \in \{b,c\}$,
$z \in \{0,1,2,3,....\}$
soft_alldifferent(x_1,x_2,x_3,x_4,z,μ)
minimize z

But what is a good violation measure?

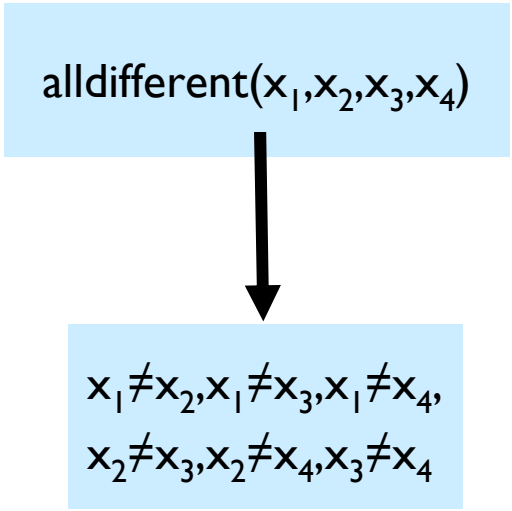
Violation measures

Cornell University

General violation measures:

- variable-based μ_{var}:
  - minimum number of variables that need to change their value in order to satisfy the constraint
- decomposition-based μ_{dec}:
  - minimum number of violated constraints in binary decomposition

[Petit et al., 2001]



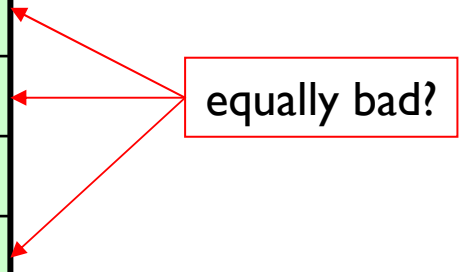
Violation measures

variable-based violation measure μ_{var}:

"minimum number of variables that need to change value in order to satisfy the constraint"

$x_1 \in \{a,b\}$, $x_2 \in \{a,b\}$,
$x_3 \in \{a,b\}$, $x_4 \in \{b,c\}$,
alldifferent(x_1,x_2,x_3,x_4)

x_1	x_2	x_3	x_4	μ_{var}
a	a	a	b	2
a	a	b	b	2
a	a	b	c	1
a	b	a	b	2
a	b	a	c	1
...	...	...	...	...



Violation measures

Cornell University

decomposition-based violation measure μ_{dec}:

"number of violated constraints in binary decomposition"

$x_1 \in \{a,b\}$, $x_2 \in \{a,b\}$,
$x_3 \in \{a,b\}$, $x_4 \in \{b,c\}$,
alldifferent(x_1,x_2,x_3,x_4) → $x_1 \neq x_2, x_1 \neq x_3, x_1 \neq x_4, x_2 \neq x_3, x_2 \neq x_4, x_3 \neq x_4$

x_1	x_2	x_3	x_4	μ_{var}	μ_{dec}
a	a	a	b	2	3
a	a	b	b	2	2
a	a	b	c	1	1
a	b	a	b	2	2
a	b	a	c	1	1
...	...	...	...	...	...

distinction

Filtering algorithm for soft alldifferent
using variable-based violation measure

Filtering for variable-based violation

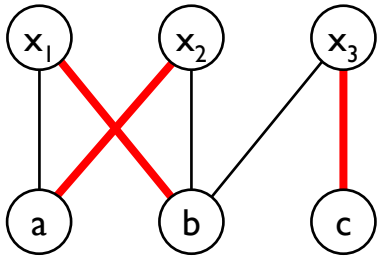
Theorem [Régin, 1994]:

solution to *hard* alldifferent ⇔ maximum matching in value graph

Example:

$x_1 \in \{a,b\}$, $x_2 \in \{a,b\}$, $x_3 \in \{b,c\}$

alldifferent(x_1,x_2,x_3)



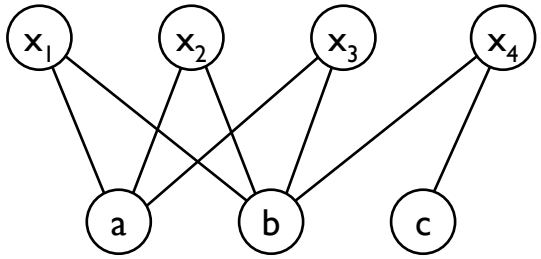
matching in graph:
subset of non-touching edges

Filtering for hard alldifferent: remove all edges (and corresponding domain values) that are not in any maximum matching

But what can we do if there is no solution?

Example:

$x_1 \in \{a,b\}$, $x_2 \in \{a,b\}$,
$x_3 \in \{a,b\}$, $x_4 \in \{b,c\}$,
alldifferent(x_1,x_2,x_3,x_4)



Filtering for variable-based violation

Fortunately, we can re-use these techniques

Theorem [Petit et al., 2001]: for alldifferent

minimum value of $\mu_{var}(x_1,x_2,...,x_n)$

=

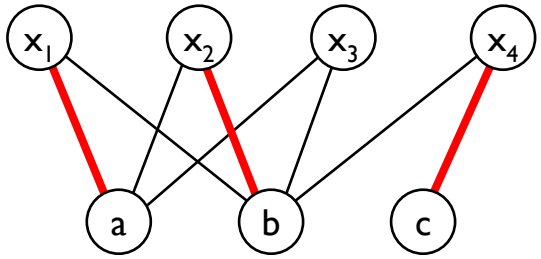
n - value of maximum matching in value graph

Example:

$x_1 \in \{a,b\}$, $x_2 \in \{a,b\}$,

$x_3 \in \{a,b\}$, $x_4 \in \{b,c\}$,

alldifferent(x_1,x_2,x_3,x_4)



n - value of maximum matching = 4 – 3 = 1,

minimum value for μ_{var} is 1 (see before)

Filtering for variable-based violation

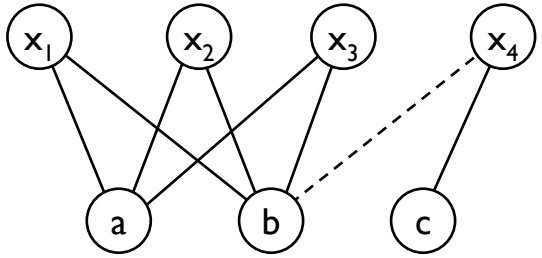
Filtering rules for $x_1,x_2,...,x_n$ in soft_alldifferent($x_1,x_2,...,x_n,z,\mu_{var}$):

Let M be maximum matching in value graph

- if (n - |M|) > max(D(z)) then constraint is inconsistent
- if (n - |M|) < max(D(z)) then all domain values are consistent
  (namely, if we change the value of one variable μ_{var} can increase at most one)
- if (n - |M|) = max(D(z)) then domain value $d \in D(x)$ is consistent iff
  edge (x,d) belongs to some maximum matching in value graph

Example:

$x_1 \in \{a,b\}$, $x_2 \in \{a,b\}$,
$x_3 \in \{a,b\}$, $x_4 \in \{b,c\}$, $z \in \{0,1\}$
soft_alldifferent($x_1,x_2,x_3,x_4,z,\mu_{var}$)
minimize z



maximum matching has size 3, hence minimum μ_{var}=4-3=1

this is equal to max(D(z)), so we need to check some edges for consistency:

value b in $D(x_4)$ is inconsistent because (x_4,b) not in maximum matching,
(maximum maching using (x_4,b) has size 2)

note that also value 0 in D(z) is inconsistent

Filtering for variable-based violation

To find consistent edges (similar to hard alldifferent):

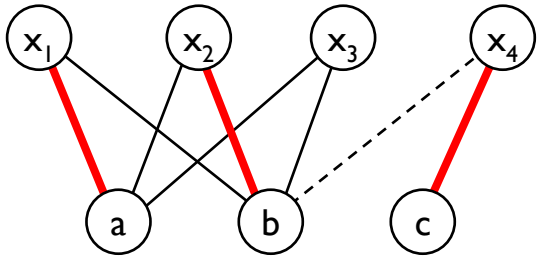
Theorem [Petersen, 1891]:

given maximum matching M, any edge e is in some maximum matching iff

- e in M, or
- e on even-length M-alternating path starting from M-free vertex, or
- e on even-length M-alternating circuit

Example:

$x_1 \in \{a,b\}$, $x_2 \in \{a,b\}$,
$x_3 \in \{a,b\}$, $x_4 \in \{b,c\}$, $z \in \{0,1\}$
soft_alldifferent(x_1,x_2,x_3,x_4,z,μ_{var})
minimize z



even-length M-alternating path starting from M-free vertex: $x_3 - b - x_2 - a - x_1$
even-length M-alternating circuit: $x_1 - a - x_2 - b - x_1$
again, edge (x_4,b) not in maximum matching

Filtering for variable-based violation

Cornell University

Algorithm to make soft_alldifferent(x_1,x_2,...,x_n,z,μ_{var}) arc consistent [Petit et al., 2001]:

compute maximum matching M in value graph ← O(m√n) [Hopcroft & Karp, 1973]
if (n - |M| > max(D(z)) **return** inconsistent
else if (n - |M| < max(D(z)) **return** consistent
else {
    remove all edges (and corresponding domain values) not in any maximum size matching ← O(m+n) [Tarjan, 1972]
    if some domain is empty **return** inconsistent
}
update min(D(z)) ≥ n - |M|
if D(z) is empty **return** inconsistent
else return consistent

Notation: m denotes #edges in graph

Notes:

- Checking consistency is done once (in O(m√n) time) while the filtering is done separately (in O(m) time).
- The algorithm for computing the maximum matching is incremental. When k variables have changed/lost their value (during search or propagation), we need at most O(km) steps to re-compute a maximum matching.

Exercise 2: Consider the following COP

$x_1 \in \{c,d\}$, $x_2 \in \{a,b,c\}$, $x_3 \in \{c,d\}$, $x_4 \in \{a,b,d\}$, $x_5 \in \{c,d\}$, $x_6 \in \{a,b,c\}$, $z \in \{0,1,2\}$

soft_alldifferent(x_1,x_2,x_3,x_4,x_5,x_6,z,μ_{var})

minimize z

a. compute $\mu_{var}(x_1,x_2,x_3,x_4,x_5,x_6)$ for various tuples

b. make the soft_alldifferent constraint arc consistent

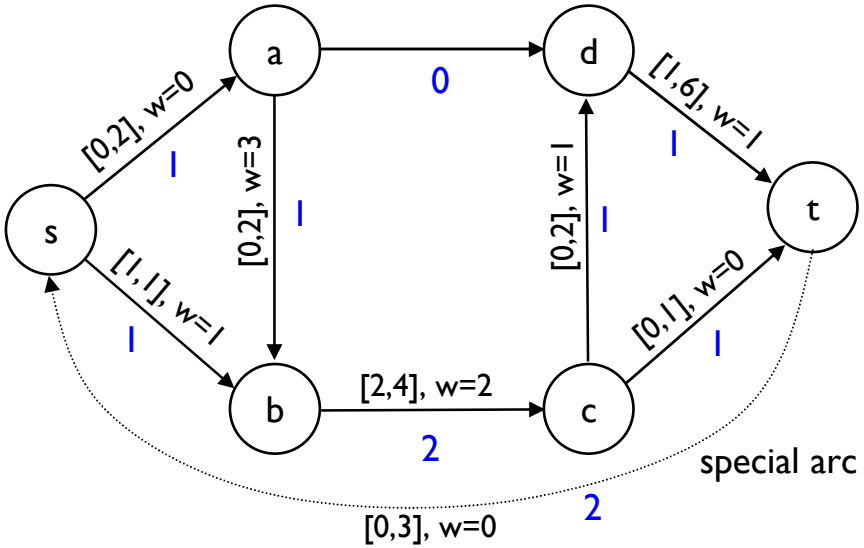
Intermezzo: Flow theory

Let G=(V,A) be a directed graph with vertex set V and arc set A. To each arc $a \in A$ we assign a capacity function [d(a),c(a)] and a weight function w(a).

Let $s,t \in V$. A function $f: A \to \mathbb{Q}$ is called an s-t flow (or a flow) if

- $f(a) \geq 0$ for all $a \in A$
- $\sum_{a \text{ enters } v} f(a) = \sum_{a \text{ leaves } v} f(a)$ for all $v \in V$ (flow conservation)
- $d(a) \leq f(a) \leq c(a)$ for all $a \in A$

Define the cost of a flow as $\sum_{a \in A} w(a)f(a)$. A minimum-cost flow is a flow with minimum cost.



flow (in blue) with cost 8

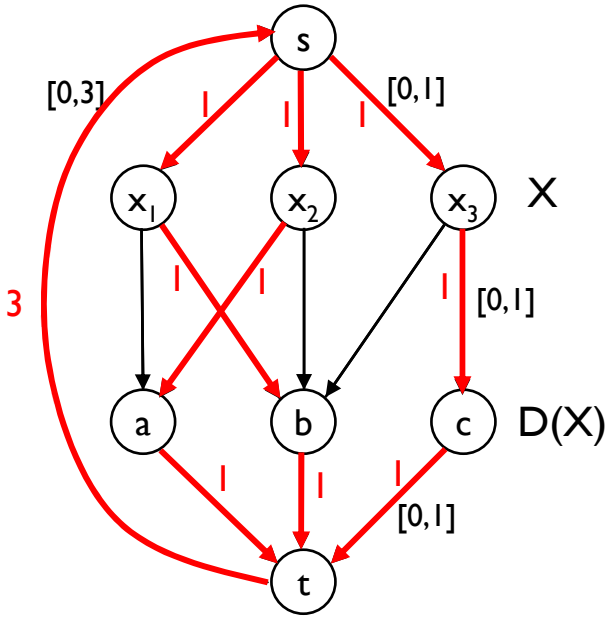
special arc to ensure flow conservation

Alternative graph representation

Fact: matching in bipartite graph ⇔ integer flow in directed bipartite graph

Step 1: direct edges from X to D(X)
Step 2: add a source s and sink t
Step 3: connect s to X, and D(X) to t
Step 4: add special arc (t,s)

all arcs have capacity [0,1] and weight 0
except arc (t,s) with capacity [0, min{|X|,|D(X)|}]



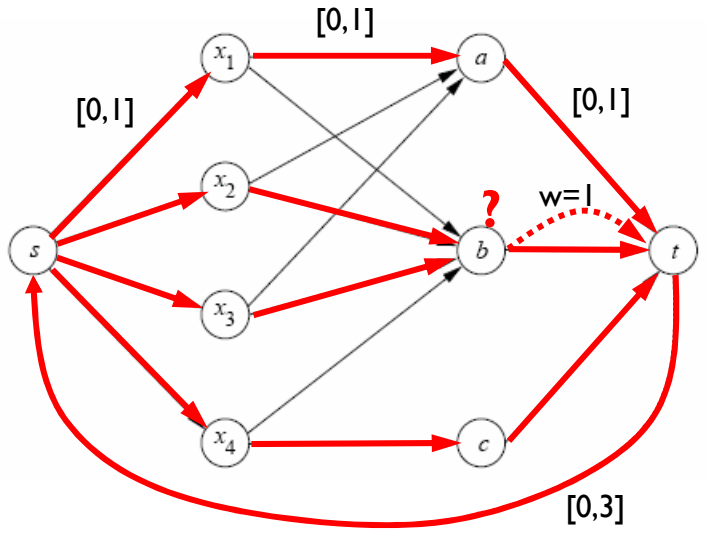
Proof: apply above construction
"⇒" define flow on arc in M to be 1, extend flow to s and t
"⇐" capacities ensure that all x in X and d in D(X) have at most 1 unit in-flow or out-flow
hence edges between X and D(X) with flow 1 define a matching

Alternative graph representation

Example:

$x_1 \in \{a,b\}$, $x_2 \in \{a,b\}$,
$x_3 \in \{a,b\}$, $x_4 \in \{b,c\}$,
alldifferent(x_1,x_2,x_3,x_4)

maximum flow (in red) corresponds to
x_1=a, x_2=b, x_4=c
but then x_3 has no value



Goal: transform graph for soft alldifferent so that
–all variables can be assigned a value
–flow captures variable-based violation measure

Solution: insert additional weighted arcs and compute weighted flow

In Example, flow has cost 1 (it uses one weighted arc) and μ_{var} is 1 in corresponding (minimum) solution

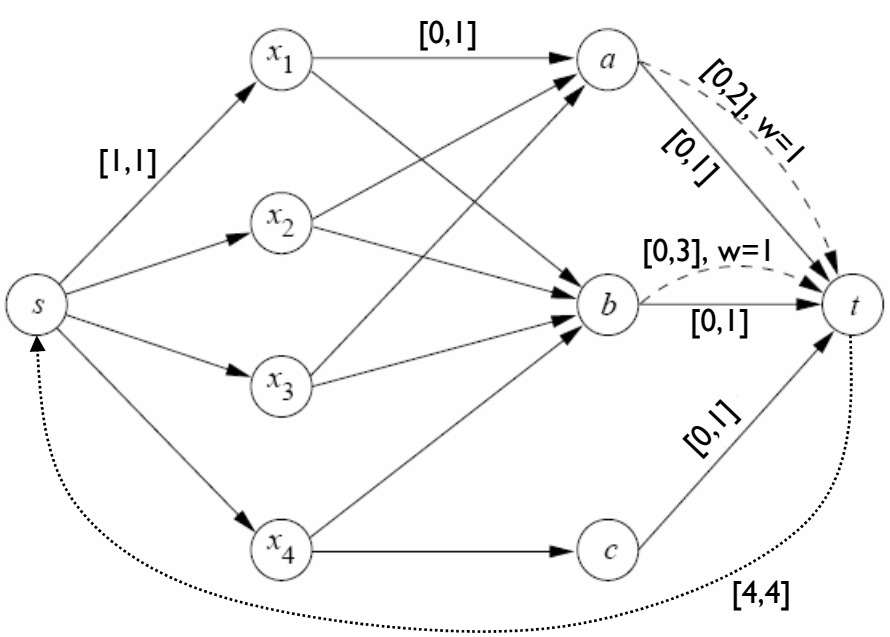
Alternative graph representation

More formally [v.H. et al., 2006]:

- for all d∈D(X) with k incoming arcs: add arc (d,t) with capacity [0,k-1] and weight 1
- for all x∈X: set capacity of arc (s,x) to [1,1], i.e. force x into solution
- minimum-cost flow ⇔ solution to soft_alldifferent minimizing μ_{var}

Example:

$x_1 \in \{a,b\}$, $x_2 \in \{a,b\}$,
$x_3 \in \{a,b\}$, $x_4 \in \{b,c\}$, $z \in \{0,1\}$
soft_alldifferent($x_1,x_2,x_3,x_4,z,\mu_{var}$)
minimize z



Note: minimum-cost flow can be computed in O(nm) time in this case, so previous approach is preferable (O(m√n) time to compute maximum matching)

Filtering algorithm for soft alldifferent
using decomposition-based violation measure

"number of violated constraints in binary decomposition"

Graph representation

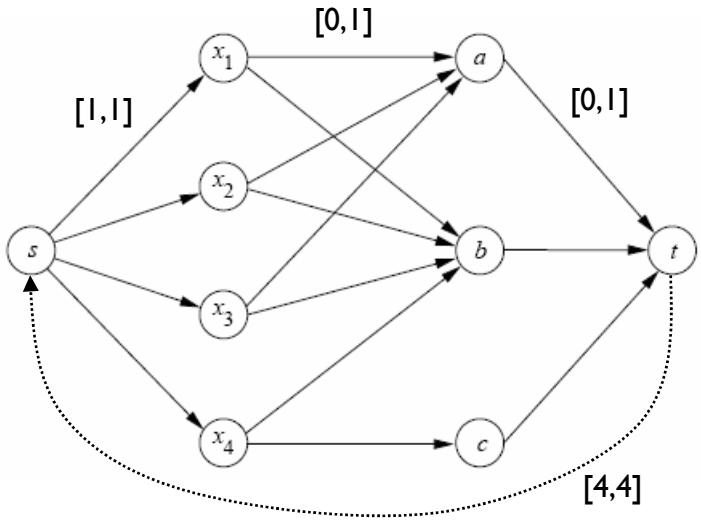
Cornell University

Again, start from basic network:

Example:

$x_1 \in \{a,b\}$, $x_2 \in \{a,b\}$,
$x_3 \in \{a,b\}$, $x_4 \in \{b,c\}$, $z \in \{0,1\}$
soft_alldifferent(x_1,x_2,x_3,x_4,z,μ_{dec})
minimize z

e.g. x_1=x_2=x_3=x_4=b should give violation cost 6



What can we do?

Options:

- manipulate capacities
- add/remove arcs
- add weights to some arcs

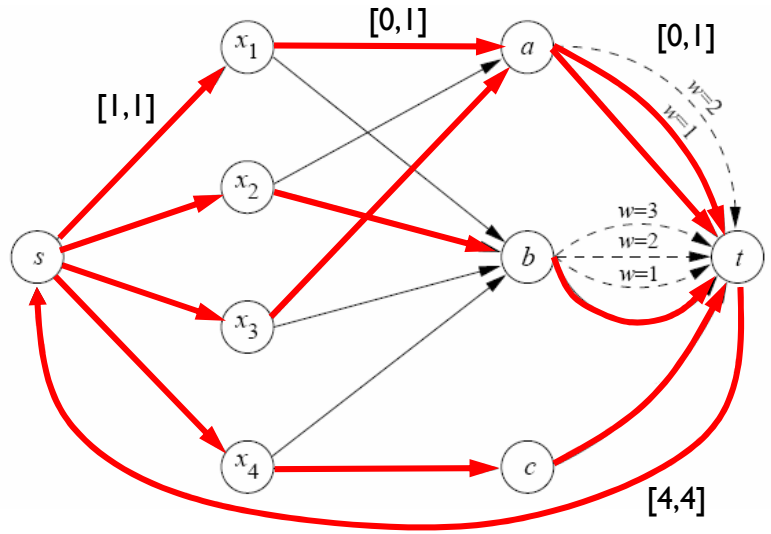
Graph representation

We can do the following [v.H., 2004]:

- for all $d \in D(X)$ with k incoming arcs: add arcs $(d,t)_1$, $(d,t)_2$,...,$(d,t)_{k-1}$, with capacity [0,1], and weight i for arc $(d,t)_i$ (i=1,...,k-1)
- for all $x \in X$: set capacity of arc (s,x) to [1,1], i.e. force x into solution
- minimum-cost flow $\Leftrightarrow$ solution to soft_alldifferent minimizing μ_{dec}

Example:

$x_1 \in \{a,b\}$, $x_2 \in \{a,b\}$,
$x_3 \in \{a,b\}$, $x_4 \in \{b,c\}$, $z \in \{0,1\}$
soft_alldifferent($x_1,x_2,x_3,x_4,z,\mu_{dec}$)
minimize z



Proof: every unit of flow entering $d \in D(X)$ leaves d (via arc from d to t) with lowest weight. k such units correspond to k variables having the same value. hence, the (k+1)-th unit violates k not-equal constraints, and contributes a cost of k (the weight of the arc it uses).

Filtering for decomposition-based violation

Algorithm to make soft_alldifferent(x_1,x_2,...,x_n,z,μ_{dec}) arc consistent:

for all arcs (x_i,d) {
    set capacity of (x_i,d) to [1,1] // force flow to represent x_i=d
    compute min-cost flow f
    if cost(f) > max(D(z)) remove d from D(x_i)
    if D(x_i) is empty **return** inconsistent
}
update min(D(z)) ≥ min(cost(f))
if D(z) is empty **return** inconsistent
else return consistent

Drawbacks:

- no separation between consistency check and filtering
- filtering relatively expensive: m flow computations each of O(mn) time complexity
- algorithm not incremental: start from scratch every time

Use flow theory to improve algorithm

Intermezzo: Flow theory (cont'd)

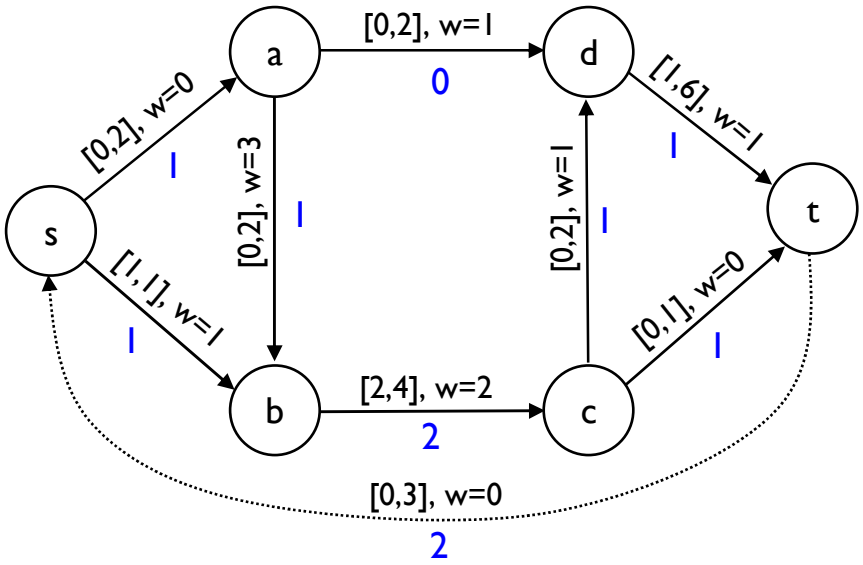
Given directed graph G=(V,A) and a flow f in G, the residual graph G_f is defined as (V,A_f) where for all $a \in A$,

$a \in A_f$ if $f(a) < c(a)$ with capacity $[\max\{d(a) - f(a), 0\}, c(a) - f(a)]$ and weight $w(a)$

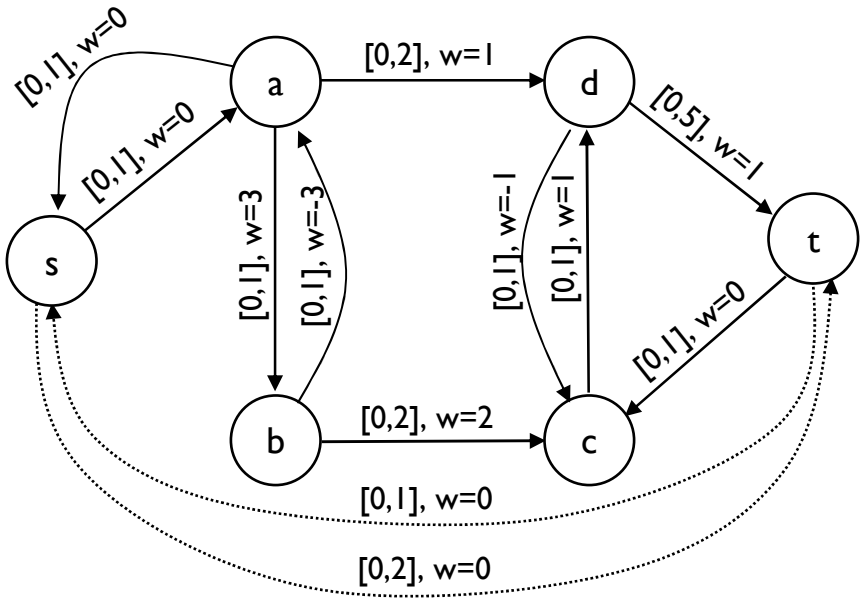
$a^{-1} \in A_f$ if $f(a) > d(a)$ with capacity $[0, f(a) - d(a)]$ and weight $-w(a)$

New capacities express how much more flow we can put on arc a or subtract from it (via a^{-1})

G and flow f



G_f



Filtering for decomposition-based violation

Theorem [e.g., Ahuja et al., 1993]:

f minimum-cost flow in G

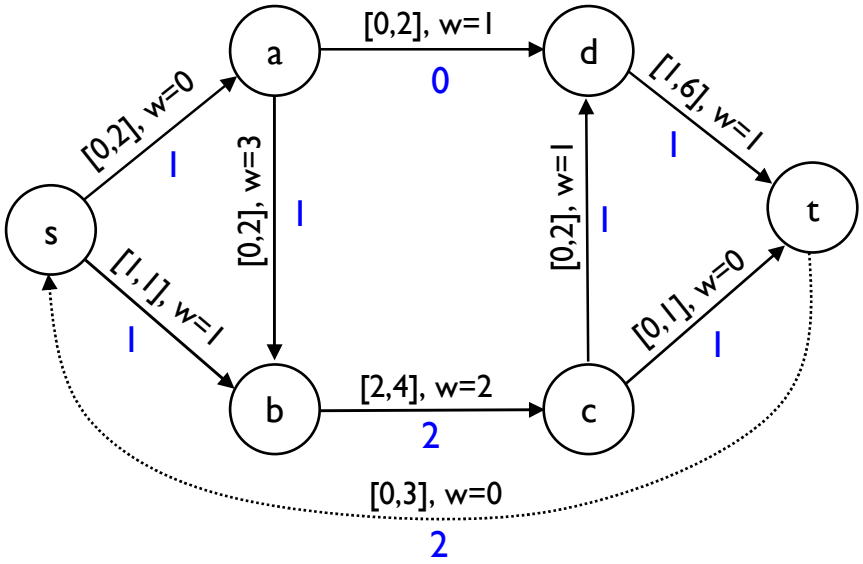
P shortest $d–x_i$ path in G_f

$\Leftrightarrow$

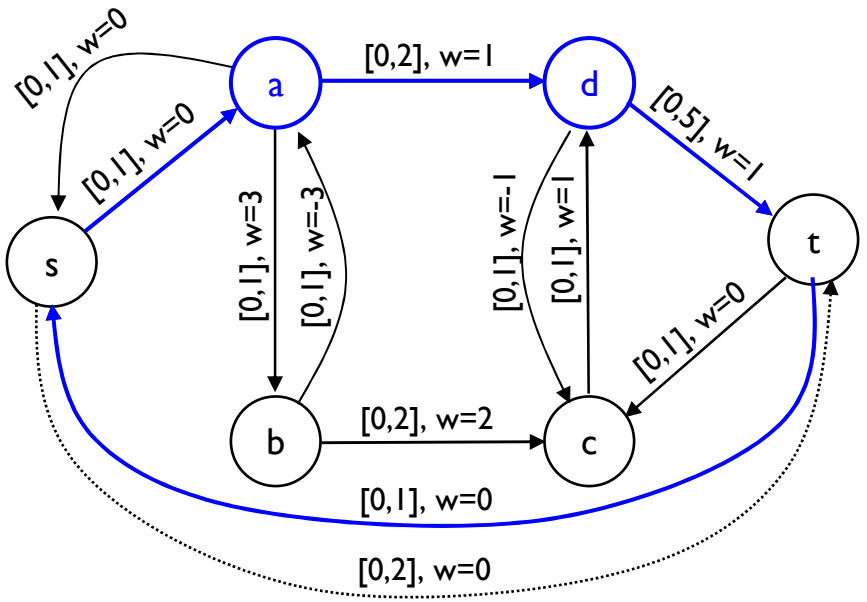
minimum-cost flow f' in G with $f'(x_i,d) = 1$ has

$cost(f') = cost(f) + cost(P)$

G and minimum-cost flow f



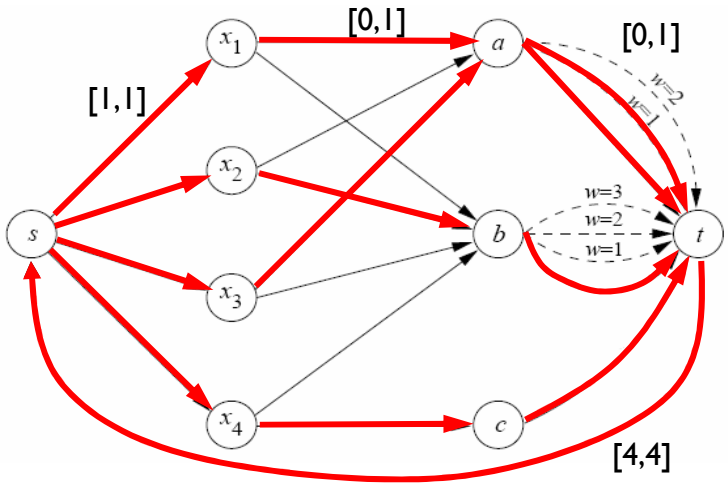
G_f



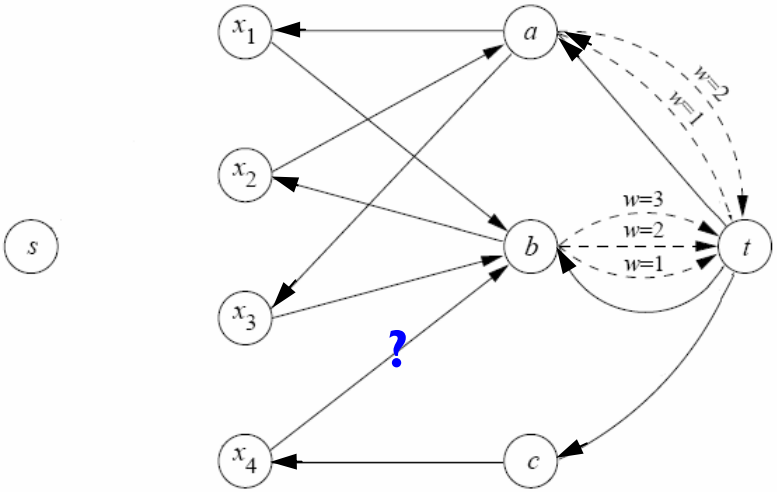
Filtering for decomposition-based violation

Apply shortest-path theorem to improve filtering algorithm [Régin, 1999 & 2002]:

compute minimum-cost flow f in G
if cost(f) > max(D(z)) return **inconsistent**
for all arcs (x_i,d) {
    compute minimum-cost d-x_i path P in G_f
    if cost(f) + cost(P) > max(D(z)) remove d from D(x_i)
    if D(x_i) is empty **return** inconsistent
}
update min(D(z)) ≥ cost(f)
else return consistent



G and minimum-cost flow f (in red)



G_f

Filtering for decomposition-based violation

Apply shortest-path theorem to improve filtering algorithm [Régin, 1999 & 2002]:

compute minimum-cost flow f in G
if cost(f) > max(D(z)) return **inconsistent**
for all arcs (x_i,d) {
    compute minimum-cost d-x_i path P in G_f
    if cost(f) + cost(P) > max(D(z)) remove d from D(x_i)
    if D(x_i) is empty **return** inconsistent
}
update min(D(z)) ≥ cost(f)
else return consistent

Improvements:

- consistency check and filtering are now separated
- filtering efficient: all shortest paths can be computed together in O(m) time [v.H.,2004]
- algorithm incremental: with k changes new flow can be computed in O(km) time

Exercise 3: Consider the following COP

$x_1 \in \{c,d\}$, $x_2 \in \{a,b,c\}$, $x_3 \in \{c,d\}$, $x_4 \in \{a,b,d\}$, $x_5 \in \{c,d\}$, $x_6 \in \{a,b,c\}$, $z \in \{0,1,2\}$

soft_alldifferent(x_1,x_2,x_3,x_4,x_5,x_6,z,μ_{dec})

minimize z

a. compute $\mu_{dec}(x_1,x_2,x_3,x_4,x_5,x_6)$ for various tuples

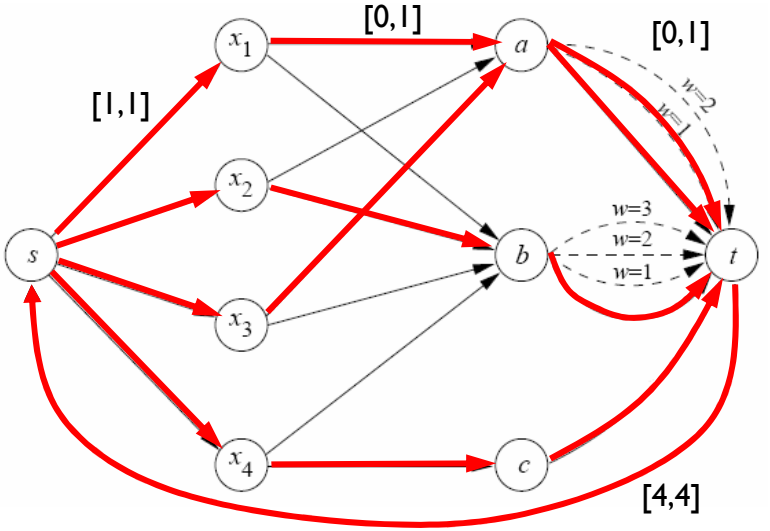
b. make the soft_alldifferent constraint arc consistent

Exercise 4 (more challenging): Find a polynomial algorithm that finds all shortest paths between values and variables in the residual graph (given minimum-cost flow) associated to decomposition-based soft_alldifferent.

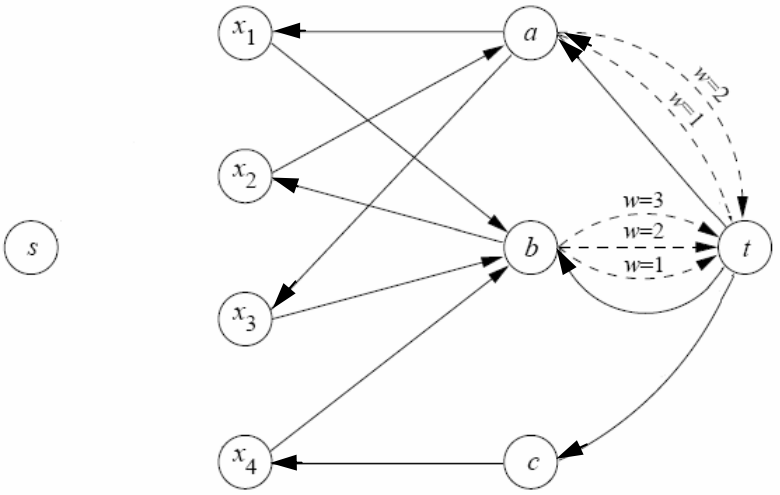
Hint 1: For some pairs the shortest path does not visit t, for others it does.

Hint 2: Strongly Connected Components can be computed in O(m+n) time [Tarjan, 1972].

G and minimum-cost flow f (in red)



G_f



Filtering algorithms for the soft global cardinality constraint

Global cardinality constraint

Cornell University

A global cardinality constraint is defined as

gcc(X, *l*, *u*)

where

X is set of variables $\{x_1,x_2,...,x_n\}$

l and *u* are arrays of (constant) integers, such that

domain value d is used between l_d and u_d times.

Example:

$x_1 \in \{0,1\}$, $x_2 \in \{0,1,2\}$, $x_3 \in \{1,2\}$, $x_4 \in \{1,2,3\}$

$l_0 = 0,\ l_1 = 1,\ l_2 = 0,\ l_3 = 1,$
$u_0 = 1, u_1 = 2, u_2 = 2, u_3 = 1,$
gcc($\{x_1,x_2,x_3,x_4\}$, *l*, *u*)

} rewrite gcc($\{x_1,x_2,x_3,x_4\}$, [0,1,0,1], [1,2,2,1])

A solution: $x_1=0$, $x_2=1$, $x_3=2$, $x_4=3$

Efficient arc consistency filtering algorithm for gcc in O(mn) time [Régin, 1996], recently improved to $O(m\sqrt{n})$ time [Quimper et al., 2004]

Special case: alldifferent = gcc with *l*=[0,0,...,0] and *u*=[1,1,...,1]

Softening global cardinality constraint

Example:

$x_1 \in \{1,2\}$, $x_2 \in \{1\}$, $x_3 \in \{1,2\}$, $x_4 \in \{1\}$

gcc($\{x_1,x_2,x_3,x_4\}$, [1,3], [2,5])

Over-constrained CSP; we need to soften the gcc:

introduce cost variable z

define some violation measure, say μ

minimize z

Result:

$x_1 \in \{1,2\}$, $x_2 \in \{1\}$, $x_3 \in \{1,2\}$, $x_4 \in \{1\}$

$z \in \{0,1,2,3,....\}$

soft_gcc($\{x_1,x_2,x_3,x_4\}$, [1,3], [2,5],z,μ)

minimize z

Again, what is a good violation measure?

Violation measures

Cornell University

General violation measures:

- variable-based μ_{var}:
  - minimum number of variables that need to change their value in order to satisfy the constraint
- decomposition-based μ_{dec}:
  - minimum number of violated constraints in binary decomposition

We can apply μ_{var}, but what is binary decomposition of gcc?

Example:

$x_1 \in \{1,2\}$, $x_2 \in \{1\}$, $x_3 \in \{1,2\}$, $x_4 \in \{1\}$
gcc($\{x_1,x_2,x_3,x_4\}$, [1,3], [2,5])

$\mu_{var}(1,1,1,1) = 3$ (e.g. change x_1, x_2 and x_3 to 2)
$\mu_{var}(2,1,2,1) = 1$ (e.g. change x_2 to 2)

Violation measures

Rewrite μ_{var} in terms of 'shortage' and 'excess' functions [v.H. et al., 2006].

Let $X = \{x_1, x_2, ..., x_n\}$ and $d \in D(X)$. Then

$$s(X,d) = \begin{cases} l_d - |\{x_i \mid x_i = d\}| & \text{if } |\{x_i \mid x_i = d\}| \leq l_d \\ 0 & \text{otherwise} \end{cases}$$

shortage of value d

$$e(X,d) = \begin{cases} |\{x_i \mid x_i = d\}| - u_d & \text{if } |\{x_i \mid x_i = d\}| \geq u_d \\ 0 & \text{otherwise} \end{cases}$$

excess of value d

We have

$$\mu_{\text{var}}(X) = \max\left(\sum_d s(X,d), \sum_d e(X,d) \right)$$

Example:

$x_1 \in \{1,2\}$, $x_2 \in \{1\}$, $x_3 \in \{1,2\}$, $x_4 \in \{1\}$

gcc($\{x_1,x_2,x_3,x_4\}$, [1,3], [2,5])

for X=(1,1,1,1): s(X,1)=0, s(X,2)=3, e(X,1)=2, e(X,2)=0, hence $\mu_{var}(1,1,1,1) = \max(3,2)=3$

for X=(2,1,2,1): s(X,1)=0, s(X,2)=1, e(X,1)=0, e(X,2)=0, hence $\mu_{var}(2,1,2,1) = \max(1,0)=1$

Violation measures

Cornell University

Example:

$x_1 \in \{1,2\}$, $x_2 \in \{1\}$, $x_3 \in \{1,2\}$, $x_4 \in \{1\}$

gcc($\{x_1,x_2,x_3,x_4\}$, [2,3], [3,5])

for X=(2,1,2,1): s(X,1)=0, s(X,2)=1, e(X,1)=0, e(X,2)=0, hence $\mu_{var}(2,1,2,1) = \max(1,0)=1$

but this is wrong! we can never assign 2+2=5 values to only 4 variables!

Problem: μ_{var} only applicable if $\sum_d l_d \leq |X| \leq \sum_d u_d$

Remedy: Define new 'value-based' violation measure $\mu_{val}(X) = \sum_d (s(X,d) + e(X,d))$

(compare: $\mu_{var}(X) = \max\left(\sum_d s(X,d), \sum_d e(X,d)\right)$)

Example:

$x_1 \in \{1,2\}$, $x_2 \in \{1\}$, $x_3 \in \{1,2\}$, $x_4 \in \{1\}$

gcc($\{x_1,x_2,x_3,x_4\}$, [2,3], [3,5])

for X=(1,1,1,1): s(X,1)=0, s(X,2)=3, e(X,1)=2, e(X,2)=0, hence $\mu_{val}(1,1,1,1) = 0+3+2+0=5$

for X=(2,1,2,1): s(X,1)=0, s(X,2)=1, e(X,1)=0, e(X,2)=0, hence $\mu_{val}(2,1,2,1) = 0+1+0+0=1$

Exercise 5: Consider the following COP

$x_1 \in \{3,4\}$, $x_2 \in \{2,3\}$, $x_3 \in \{3,4\}$, $x_4 \in \{1,2,4\}$, $x_5 \in \{3,4\}$, $x_6 \in \{1,2,3\}$, $z \in \{0,1,2\}$

soft_gcc(x_1,x_2,x_3,x_4,x_5,x_6,[3,1,0,0],[5,3,1,1],z,μ)

minimize z

compute $\mu_{var}(x_1,x_2,x_3,x_4,x_5,x_6)$ and $\mu_{val}(x_1,x_2,x_3,x_4,x_5,x_6)$ for various tuples

$$\mu_{\text{var}}(X) = \max\left(\sum_d s(X,d), \sum_d e(X,d)\right) \qquad \mu_{\text{val}}(X) = \sum_d (s(X,d) + e(X,d))$$

Solution:

x_1	x_2	x_3	x_4	x_5	x_6	s(X,1)	s(X,2)	s(X,3)	s(X,4)	e(X,1)	e(X,2)	e(X,3)	e(X,4)	μ_{var}	μ_{val}
3	2	3	1	3	1	1	0	0	0	0	0	2	0	2	3
4	3	4	4	4	3	3	1	0	0	0	0	1	3	4	8
3	2	4	1	3	1	1	0	0	0	0	0	1	0	1	2

Filtering algorithm for soft gcc
using variable-based violation measure

Filtering for variable-based violation

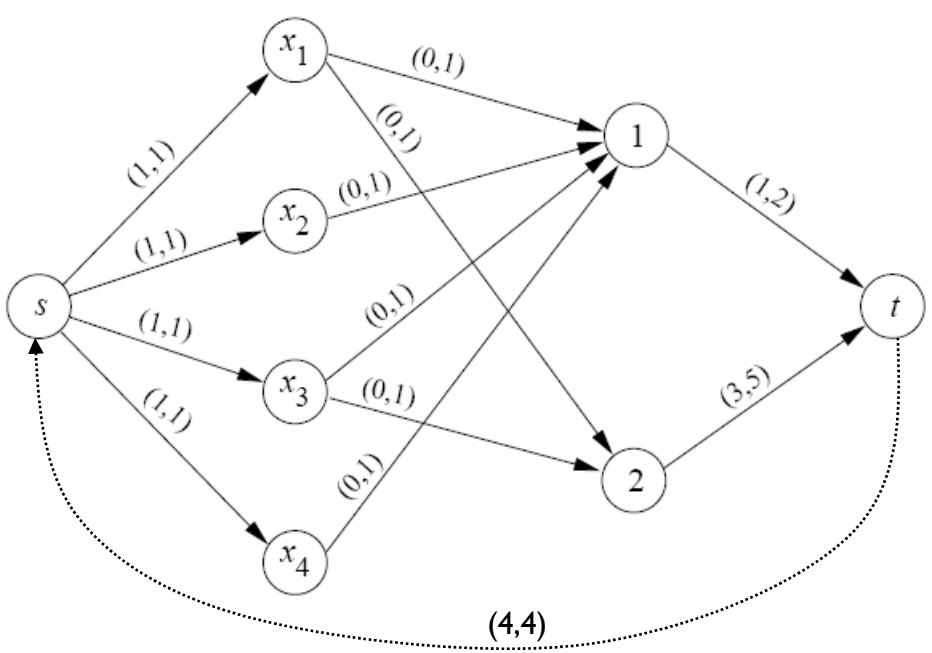
Theorem [Régin, 1996]:

solution to hard gcc ⇔ flow in particular graph

Example:

$x_1 \in \{1,2\}$, $x_2 \in \{1\}$, $x_3 \in \{1,2\}$, $x_4 \in \{1\}$

gcc($\{x_1,x_2,x_3,x_4\}$, [1,3], [2,5])



for each arc a, capacity indicated as (d(a), c(a))

over-constrained gcc:

no flow exists

what can we do?

insert weighted arcs?

Filtering for variable-based violation

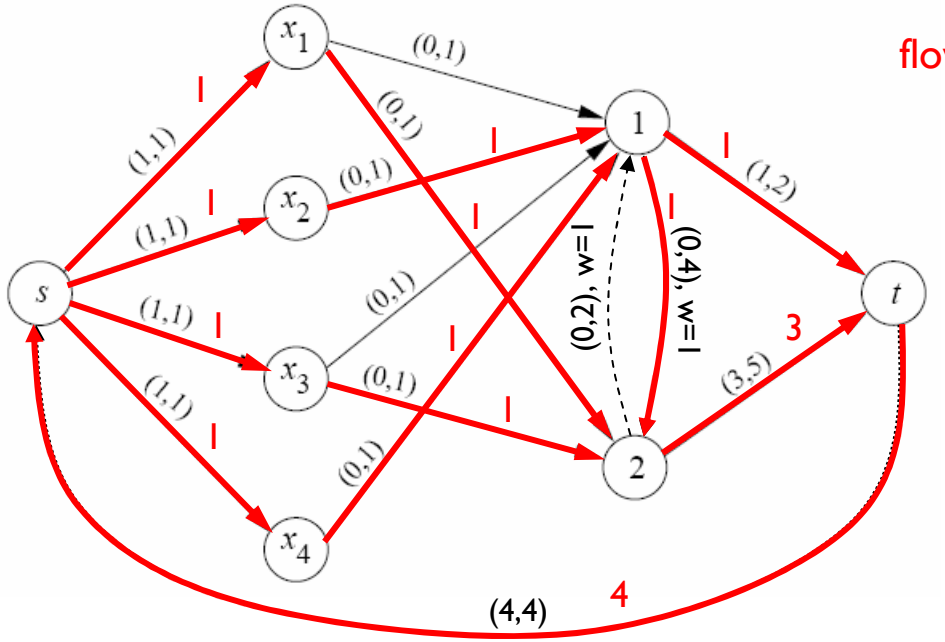
We can do the following [v.H. et al., 2006]:

- for all pairs u,v∈D(X): add arcs (u,v) with capacity [0,|X|], and weight 1
- minimum-cost flow ⇔ solution to soft_gcc minimizing μ_{var}

Example:

$x_1 \in \{1,2\}$, $x_2 \in \{1\}$, $x_3 \in \{1,2\}$, $x_4 \in \{1\}$

soft_gcc($\{x_1,x_2,x_3,x_4\}$, [1,3], [2,5], z, μ_{var})



flow with cost 1

Filtering for variable-based violation

Cornell University

Algorithm to make soft_gcc(X,*l*,*u*,z,μ_{var}) arc consistent:

for all arcs (x_i,d) {
    set capacity of (x_i,d) to [1,1] // force flow to represent x_i=d
    compute min-cost flow f
    if cost(f) > max(D(z)) remove d from D(x_i)
    if D(x_i) is empty **return** inconsistent
}
update min(D(z)) ≥ min(cost(f))
if D(z) is empty **return** inconsistent
else return consistent

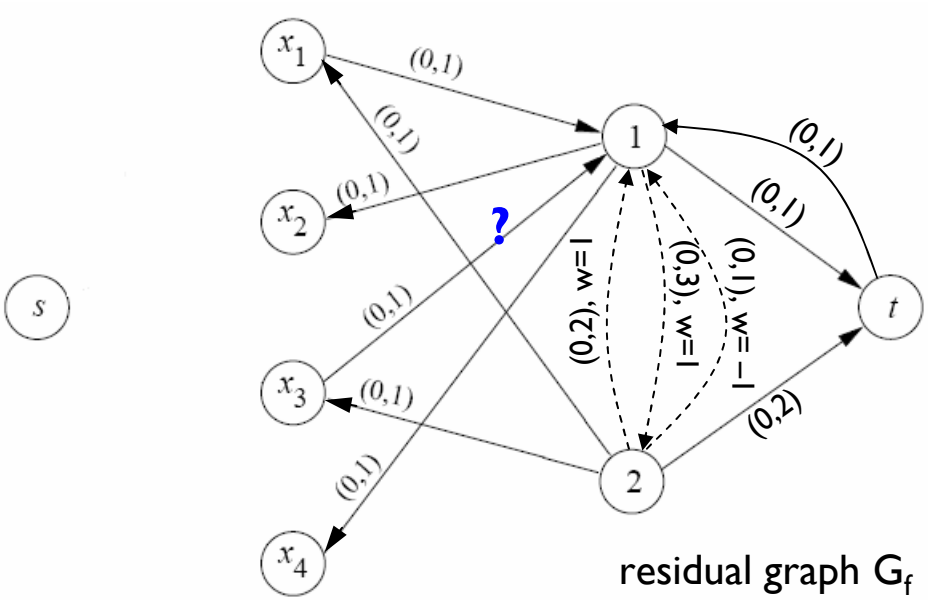
Drawbacks (again):

- no separation between consistency check and filtering
- filtering relatively expensive: m flow computations each of O(n(m+n log n)) time (each flow computes n shortest paths in O(m+n log n) time)
- algorithm not incremental: start from scratch every time

Filtering for variable-based violation

Again, apply shortest-path theorem to improve filtering algorithm:

compute minimum-cost flow f in G
if cost(f) > max(D(z)) return **inconsistent**
for all arcs (x_i,d) {
    compute minimum-cost d-x_i path P in G_f
    if cost(f) + cost(P) > max(D(z)) remove d from D(x_i)
    if D(x_i) is empty **return** inconsistent
}
update min(D(z)) ≥ cost(f)
return consistent



residual graph G_f

Filtering for variable-based violation

Again, apply shortest-path theorem to improve filtering algorithm:

compute minimum-cost flow f in G
if cost(f) > max(D(z)) return **inconsistent**
for all arcs (x_i,d) {
    compute minimum-cost d-x_i path P in G_f
    if cost(f) + cost(P) > max(D(z)) remove d from D(x_i)
    if D(x_i) is empty **return** inconsistent
}
update min(D(z)) ≥ cost(f)
return consistent

Improvements:

- consistency check and filtering are separated
- filtering more efficient: m-n shortest paths computations
- algorithm incremental: with k changes new flow can be computed with k shortest paths computations

Note: algorithm literally the same as for soft_alldifferent with μ_{dec}

Exercise 6: Consider the following COP

$x_1 \in \{3,4\}$, $x_2 \in \{2,3\}$, $x_3 \in \{3,4\}$, $x_4 \in \{1,2,4\}$, $x_5 \in \{3,4\}$, $x_6 \in \{1,2,3\}$, $z \in \{0,1\}$

soft_gcc(x_1,x_2,x_3,x_4,x_5,x_6,[3,1,0,0],[5,3,1,1],z,μ_{var})

minimize z

make the soft_gcc arc consistent

Solution: tuple with minimum violation is (3,2,4,1,3,1), with $\mu_{var} = 1$

(in that solution we can change value of x_5 from 3 to 1 to satisfy the constraint)

arc consistency (note that max(D(z)) = 1):

$x_1 \in \{3,4\}$, $x_2 \in \{2\}$, $x_3 \in \{3,4\}$, $x_4 \in \{1,2\}$, $x_5 \in \{3,4\}$, $x_6 \in \{1,2\}$, $z \in \{1\}$

Filtering algorithm for soft gcc
using value-based violation measure

Filtering for value-based violation

Cornell University

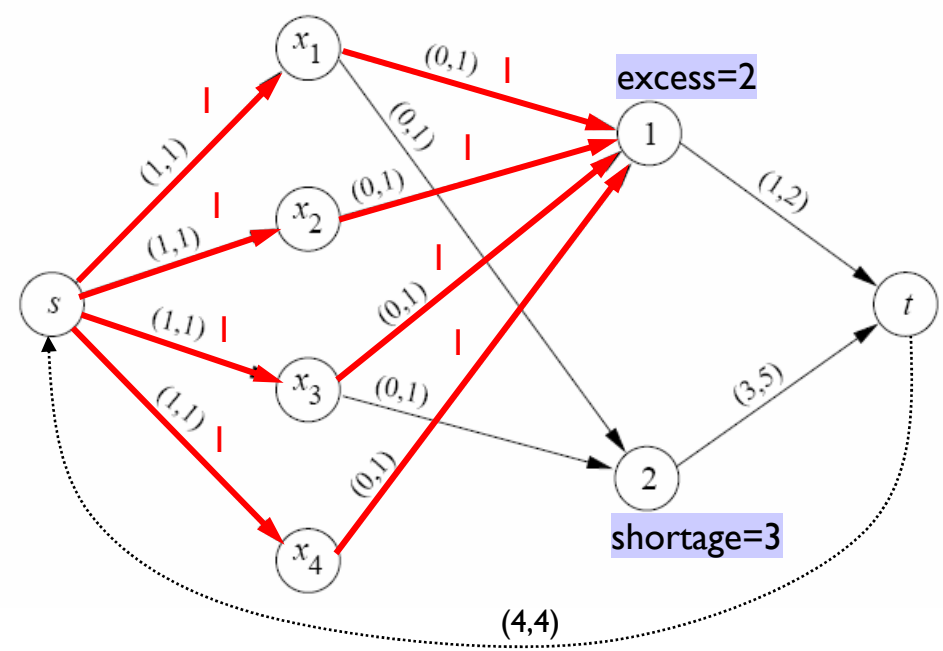
Again, start from basic network

Example:

$x_1 \in \{1,2\}$, $x_2 \in \{1\}$, $x_3 \in \{1,2\}$, $x_4 \in \{1\}$

gcc($\{x_1,x_2,x_3,x_4\}$, [1,3], [2,5])

e.g. when all variables are assigned to 1, μ_{val}=2+3=5



where to insert weighted arcs?

$$\mu_{\text{val}}(X) = \sum_d (s(X,d) + e(X,d))$$

Filtering for value-based violation

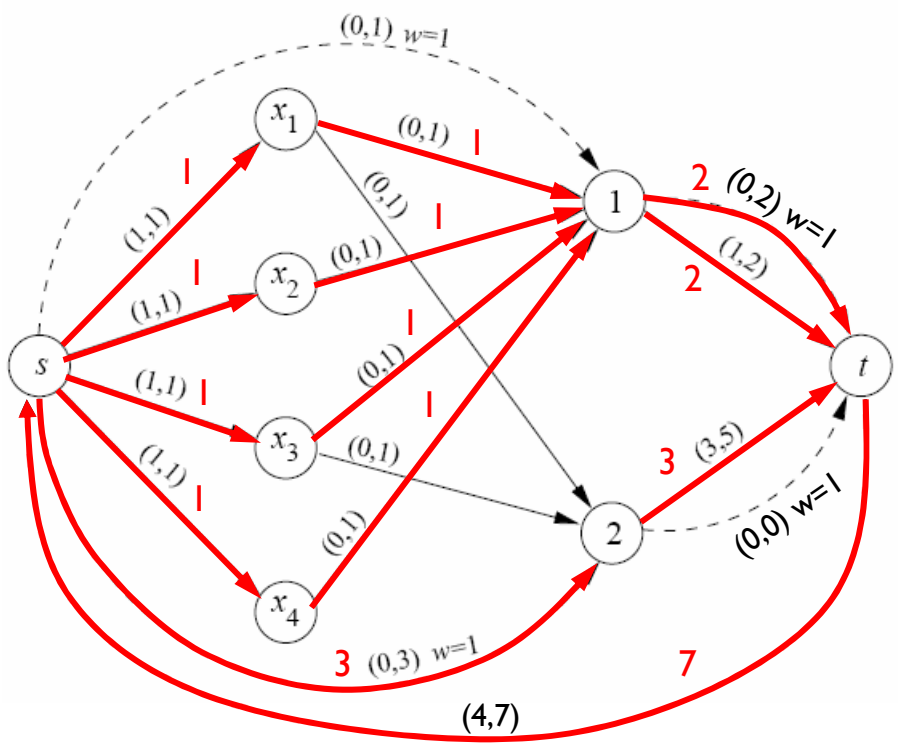
Cornell University

We can do the following [v.H. et al., 2006]:

- for all d∈ D(X): add arcs (s,d) with capacity $[0,l_d]$, and weight 1
- for all d∈ D(X): add arcs (d,t) with capacity $[0,\max\{|X|-u_d,0\}]$, and weight 1
- minimum-cost flow ⇔ solution to soft_gcc minimizing μ_{val}

Example: $x_1 \in \{1,2\}$, $x_2 \in \{1\}$, $x_3 \in \{1,2\}$, $x_4 \in \{1\}$

soft_gcc($\{x_1,x_2,x_3,x_4\}$, [1,3], [2,5], z, μ_{val})



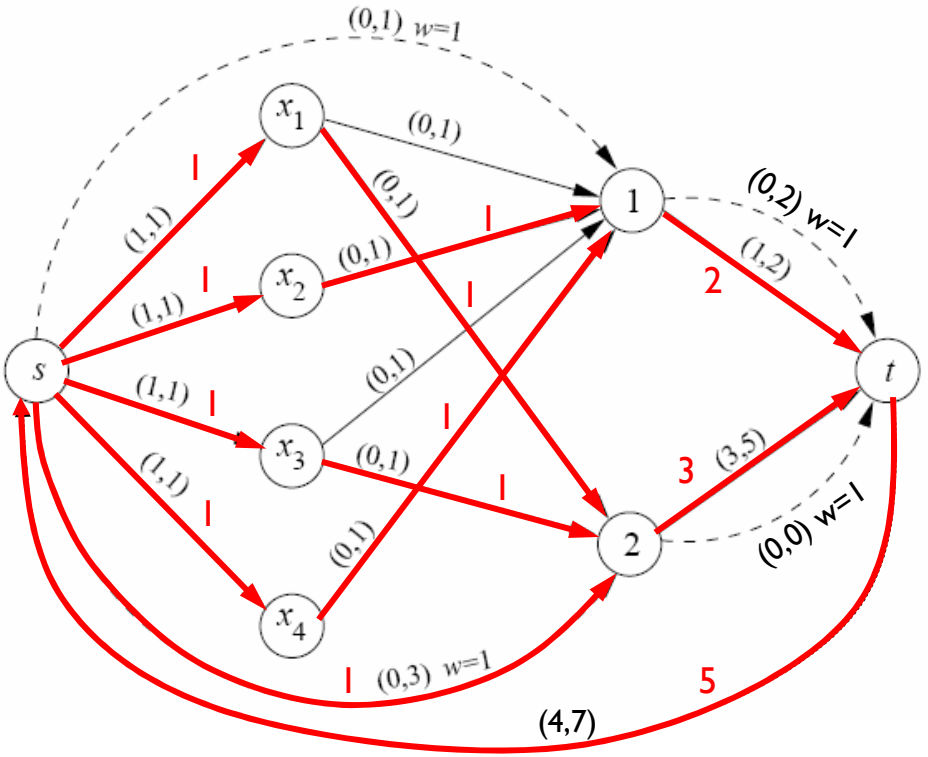
flow with cost 5

Filtering for value-based violation

Cornell University

Example: $x_1 \in \{1,2\}$, $x_2 \in \{1\}$, $x_3 \in \{1,2\}$, $x_4 \in \{1\}$

soft_gcc($\{x_1,x_2,x_3,x_4\}$, [1,3], [2,5], z, μ_{val})



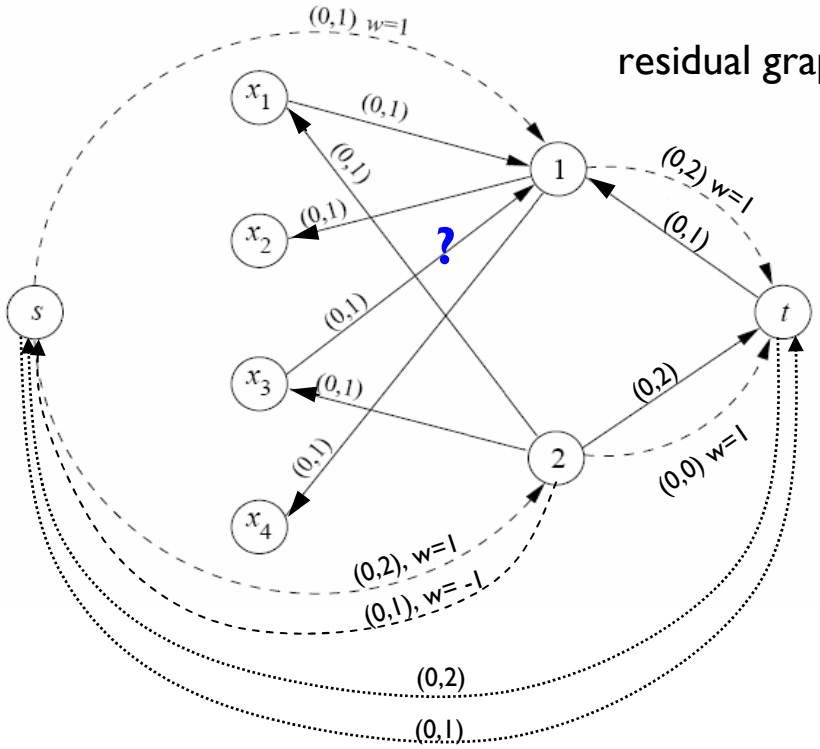
minimum-cost flow with cost 1

Filtering for value-based violation

Algorithm to make soft_gcc(X,*l*,*u*,z,μ_{val}) arc consistent (improved version):

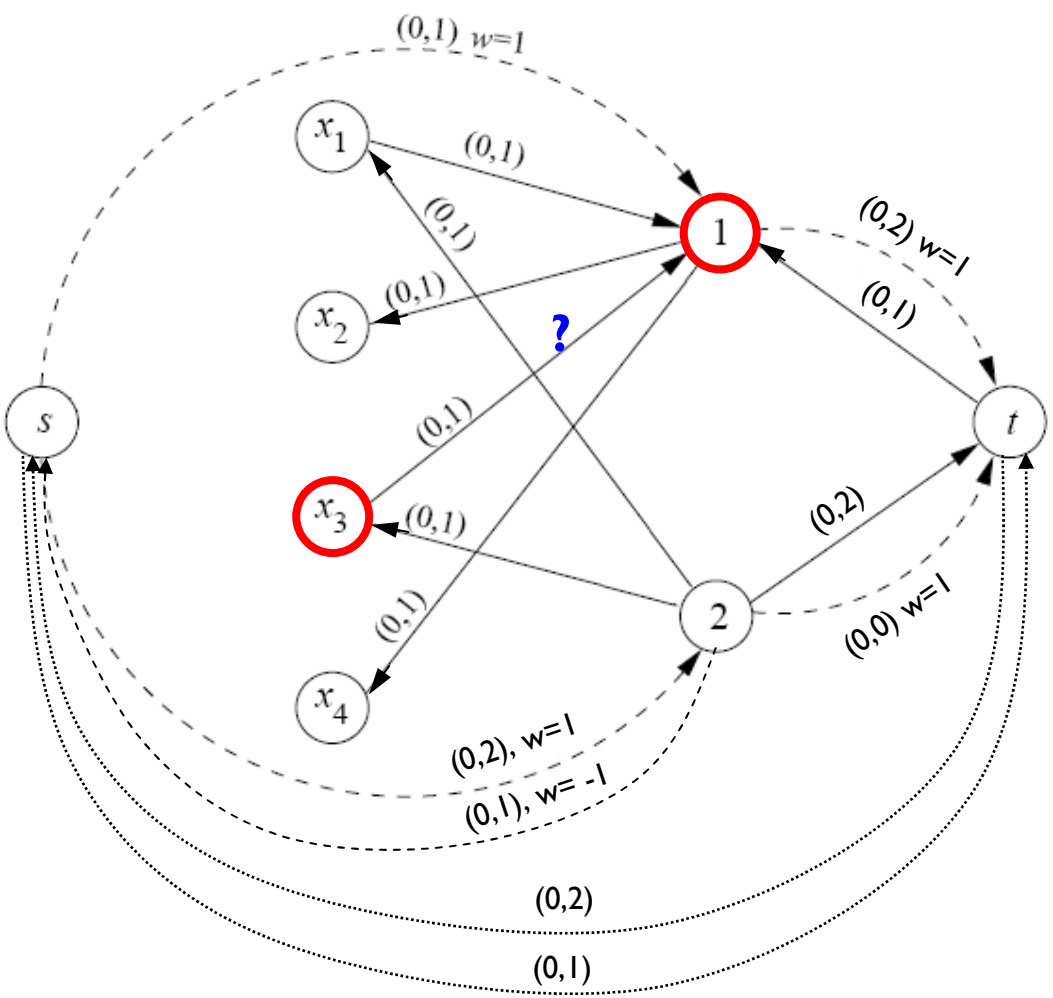
compute minimum-cost flow f in G
if cost(f) > max(D(z)) return **inconsistent**
for all arcs (x_i,d) {
    compute minimum-cost d-x_i path P in G_f
    if cost(f) + cost(P) > max(D(z)) remove d from D(x_i)
    if D(x_i) is empty **return** inconsistent
}
update min(D(z)) ≥ cost(f)
else return consistent

residual graph G_f



Note: Same algorithm as for soft_gcc with μ_{var}

Filtering for value-based violation



residual graph G_f

Exercise 7: Consider the following COP

$x_1 \in \{3,4\}$, $x_2 \in \{2,3\}$, $x_3 \in \{3,4\}$, $x_4 \in \{1,2,4\}$, $x_5 \in \{3,4\}$, $x_6 \in \{1,2,3\}$, $z \in \{0,1,2\}$

soft_gcc(x_1,x_2,x_3,x_4,x_5,x_6,[3,1,0,0],[5,3,1,1],z,μ_{val})

minimize z

make the soft_gcc arc consistent

Solution: tuple with minimum violation is (3,2,4,1,3,1), with $\mu_{val} = 2$

(in that solution we can change value of x_5 from 3 to 1 to satisfy the constraint)

arc consistency (note that max(D(z)) = 2):

$x_1 \in \{3,4\}$, $x_2 \in \{2\}$, $x_3 \in \{3,4\}$, $x_4 \in \{1,2\}$, $x_5 \in \{3,4\}$, $x_6 \in \{1,2\}$, $z \in \{2\}$

Final remarks

Previous algorithms establish arc consistency on soft_gcc in O(n(m + n log n)) time.

Recently, Zanarini et al. [2006] presented an alternative approach that improves the running time to $O(m\sqrt{n})$. Their approach extends the algorithm of Petit et al. [2001] for the variable-based soft alldifferent:

- apply result of Quimper et al. [2004] to compute two flows in the value graph:
  - one flows w.r.t. lower bounds *l*: f_l
  - one flow w.r.t. upper bounds *u*: f_u
- minimum shortage is
  - 0 if $\sum_d l_d \leq f_l$ i.e., all lower bounds are respected
  - $\sum_d l_d - f_l$ otherwise
- minimum excess is
  - 0 if $|X| \leq f_u$ i.e., all upper bounds are respected
  - $|X| - f_u$ otherwise
- filtering:
  - forcing x=d increases μ_{var} at most 1
  - forcing x=d increases μ_{val} at most 2 (hence bit more complex)

Filtering algorithm for the soft regular constraint

Regular constraint

Cornell University

A regular constraint is called in full a

"regular-language-membership-constraint" [Pesant, 2004]

A regular language can be represented by a deterministic finite automaton (DFA):

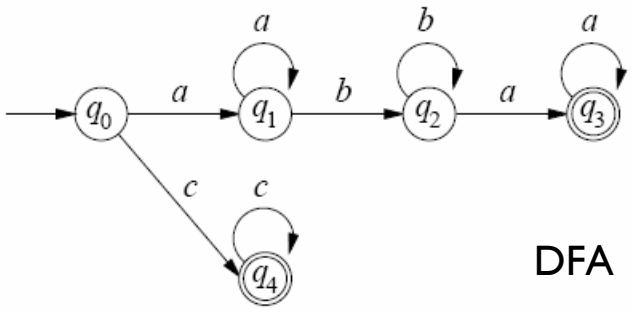
automaton accepts string ⇔ string belongs to regular language

Example:

q_0 is start state, q_3 and q_4 are end states

arrow is transition from one state to another

each transition has a label



DFA

automaton accepts string ⇔ string belongs to regular language

e.g. string aabbaa and ccc accepted, string caabbac not accepted

constraint regular(x_1,x_2,...,x_n,DFA): 'string' $x_1x_2\cdots x_n$ is accepted by DFA

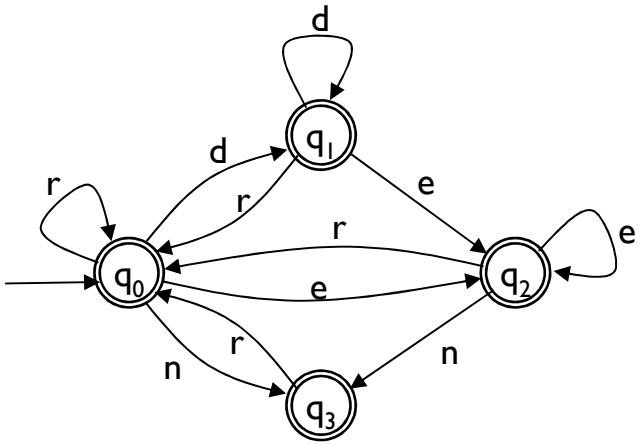
Regular constraint

Why is the regular constraint useful?

Consider the problem to roster nurses in a hospital

- each nurse works at most one shift a day
- each shift contains 8 consecutive hours
  - day shift: 8am-4pm
  - evening shift: 4pm-12am
  - night shift: 12am-8am
- after a night shift, nurse needs to take one day rest
- after an evening shift, nurse may not work a day shift

Feasible roster (7 days) for a nurse: day - day - evening - night - rest - day – day



DFA

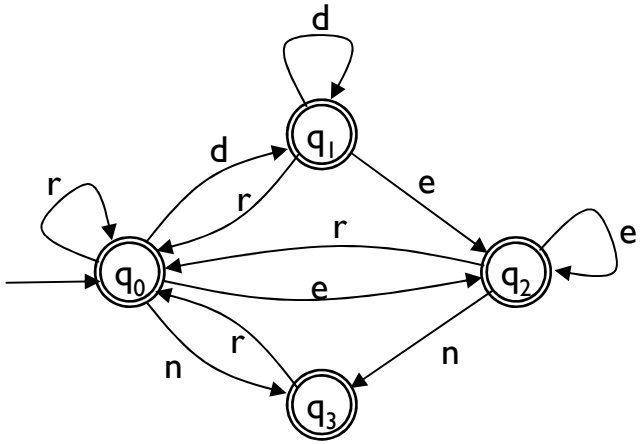
Regular constraint

Cornell University

for each nurse, introduce variables $X = \{x_1, x_2, ..., x_7\}$ representing shift on day 1,2,...7
with domains $D(x) = \{r,d,e,n\}$ for all $x \in X$

model the pattern using a regular constraint:

regular(X, DFA)



DFA

solution to this constraint $\Leftrightarrow$
string $x_1x_2\cdots x_7$ (roster) is accepted by DFA

Remark: also other constraints involved, for example global cardinality constraints to express minimum and maximum number of nurses during each shift

In short:

- regular constraint can be used to capture 'patterns' in a sequence of variables
- for example, we can represent stretch or period constraints with a regular constraint

Regular constraint

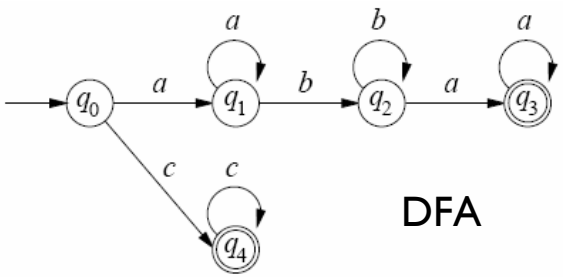
Cornell University

Theorem [Pesant, 2004]:

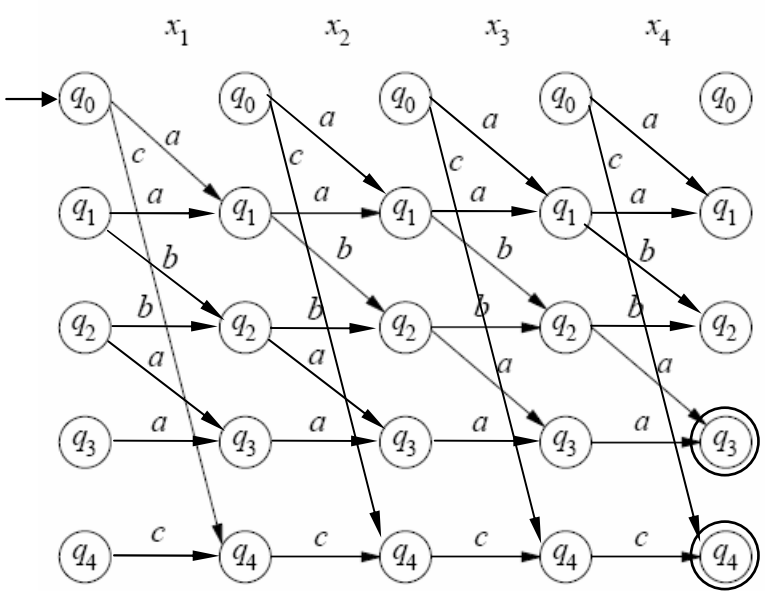
solution to regular ⇔ path from q_0 to 'end vertex' in unfolded graph

Example: $x_1 \in \{a,b,c\}$, $x_2 \in \{a,b,c\}$, $x_3 \in \{a,b,c\}$, $x_4 \in \{a,b,c\}$

regular(x_1,x_2,x_3,x_4,DFA)



unfolded graph

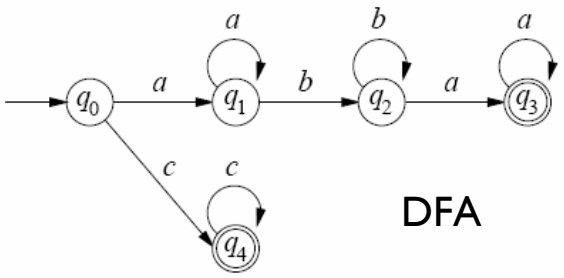


Filtering: remove all arcs whose label is not supported by domain value and vice versa

remove all arcs not on path from q_0 to 'end vertex' and update domains

(linear time in size of graph)

Regular constraint



Example: over-constrained CSP

$x_1 \in \{a,c\}$, $x_2 \in \{a,b,c\}$, $x_3 \in \{a,b,c\}$, $x_4 \in \{b,d\}$

regular(x_1,x_2,x_3,x_4,DFA)

Soften the regular constraint, as usual:

$x_1 \in \{a,c\}$, $x_2 \in \{a,b,c\}$, $x_3 \in \{a,b,c\}$, $x_4 \in \{b,d\}$, $z \in \{0,1,2,...\}$

soft_regular(x_1,x_2,x_3,x_4,DFA,z,μ)

minimize z

Again, what violation measure μ can we apply?

Exercise 8: compute μ_{var}(a,a,a,b) and μ_{var}(c,b,a,d) for the above regular constraint

Solution:

μ_{var}(a,a,a,b) = 2

μ_{var}(c,b,a,d) = 2

Filtering algorithm for soft regular constraint
using variable-based violation measure

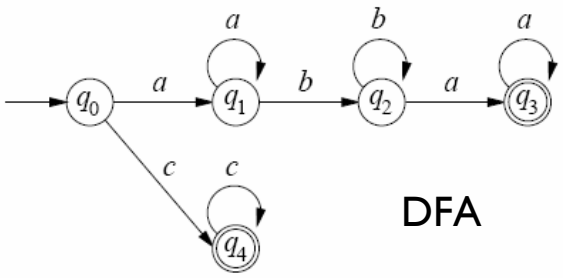
Filtering for variable-based violation

Cornell University

Example:

$x_1 \in \{a,b,c\}$, $x_2 \in \{a,b,c\}$, $x_3 \in \{a,b,c\}$, $x_4 \in \{b,d\}$

regular(x_1,x_2,x_3,x_4,DFA)



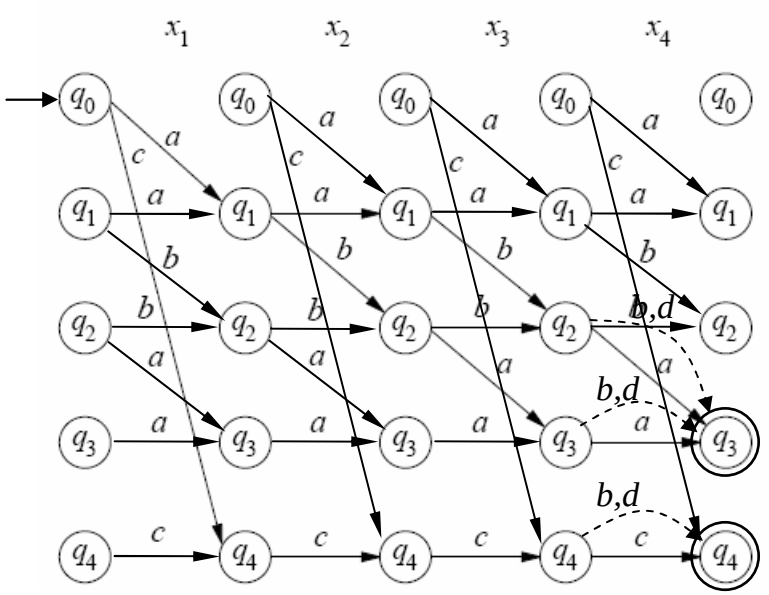
DFA

Can we add weighted arcs to the unfolded graph?

- e.g., we want to use value b or d for x_4
- we must start from q_0
- we must end in end vertex q_3 or q_4
- we want to capture μ_{var}:
  - allowed path must have cost of μ_{var}

For clarity, remove clearly infeasible arcs without touching the domains

Can we insert weighted arcs now?



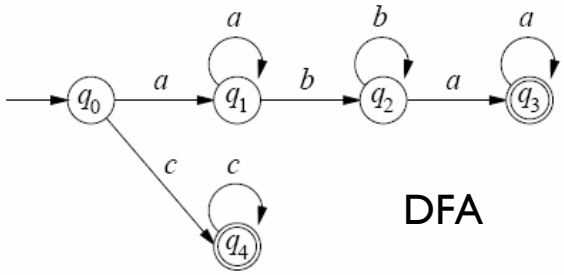
Filtering for variable-based violation

More formally [v.H. et al., 2006]:

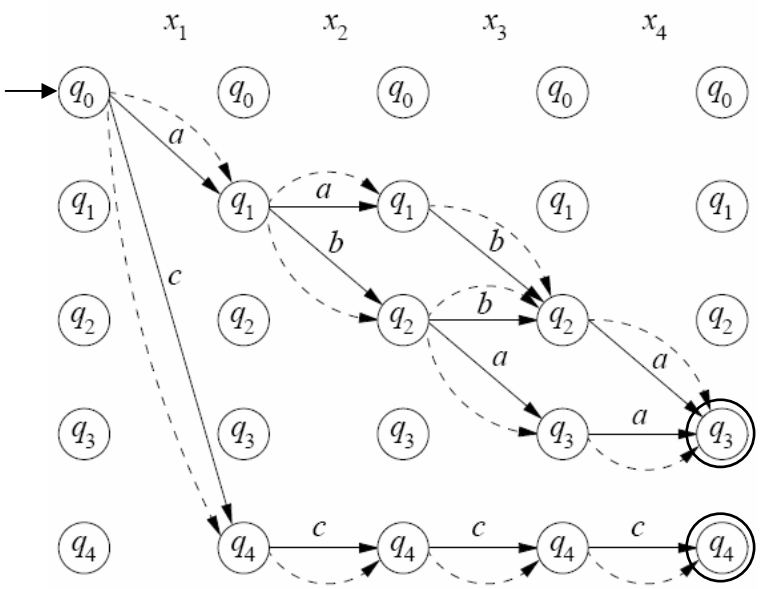
- for every arc in the unfolded graph, add parallel arc with 'void' label and weight 1
- minimum-cost path from q_0 to end vertex $\Leftrightarrow$ solution to soft_regular minimizing μ_{var}

Example:

$x_1 \in D(x_1)$, $x_2 \in D(x_2)$, $x_3 \in D(x_3)$, $x_4 \in D(x_4)$, $z \in \{0,1,2,...\}$,
soft_regular(x_1,x_2,x_3,x_4,DFA,z,μ_{var})
minimize z



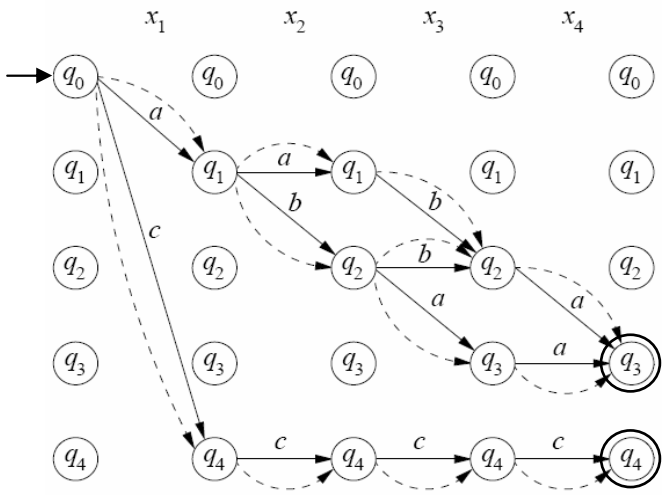
filtered unfolded graph



Filtering for variable-based violation

Algorithm to make soft_regular(X,DFA,z,μ_{var}) arc consistent:

compute shortest path from q_0 to all other vertices
compute shortest path from end vertices to all other vertices (by reversing the arcs)
update min(D(z)) ≥ cost(q_0-end path)
if D(z) is empty **return** inconsistent
else for all x∈X {
  for all arcs (u,v) in layer corresponding to x {
    remove arc if cost(q_0-u path) + cost(u-end path) + cost(u,v) > max(D(z))
  }
  update D(x)
  if D(x) is empty **return** inconsistent
}
return consistent



Filtering for variable-based violation

Algorithm to make soft_regular(X,DFA,z,μ_{var}) arc consistent:

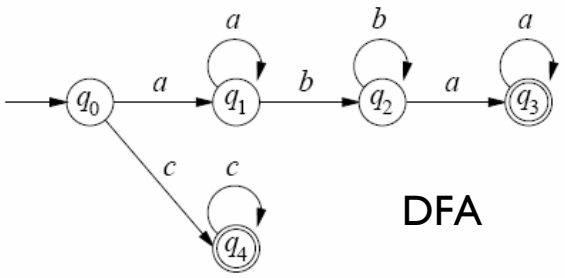
compute shortest path from q_0 to all other vertices
compute shortest path from end vertices to all other vertices (by reversing the arcs)
update min(D(z)) ≥ cost(q_0-end path)
if D(z) is empty **return** inconsistent
else for all x∈X {
  for all arcs (u,v) in layer corresponding to x {
    remove arc if cost(q_0-u path) + cost(u-end path) + cost(u,v) > max(D(z))
  }
  update D(x)
  if D(x) is empty **return** inconsistent
}
return consistent

Notes:

- shortest paths can be computed together in 2*O(m) time (unfolded graph is acyclic)
- total time complexity O(m), where m is number of arcs
- same time complexity as hard regular constraint

Exercise 8: Consider the following COP

$x_1 \in \{a,b,c,d\}$, $x_2 \in \{a,b,c,d\}$, $x_3 \in \{a,b,c\}$, $x_4 \in \{b,d\}$, $z \in \{0,1\}$
soft_regular(x_1,x_2,x_3,x_4,DFA,z,μ_{var})
minimize z



make the soft_regular constraint arc consistent

Solution: tuple with minimum violation is (a,b,a,b), with $\mu_{var} = 1$
(in that solution we can change value of x_4 from b to a to satisfy the constraint)

arc consistency (note that max(D(z)) = 1):

$x_1 \in \{a,c\}$, $x_2 \in \{a,b,c\}$, $x_3 \in \{a,b,c\}$, $x_4 \in \{b,d\}$, $z \in \{1\}$

Conclusions and perspectives

Conclusions

Cornell University

We have presented efficient filtering algorithms for several soft global constraints:

constraint	violation measure	consistency check	arc consistency
hard alldifferent	-	O(m√n)	O(m)
soft alldifferent	variable-based	O(m√n)	O(m)
soft alldifferent	decomposition-based	O(mn)	O(m)
gcc	-	O(m√n)	O(m)
soft gcc	variable-based	O(m√n)	O(m)
soft gcc	value-based	O(m√n)	O(m)
regular	-	O(m)	O(m)
soft regular	variable-based	O(m)	O(m)

Messages:

- soft global constraints are very useful for filtering purposes!
- soft versions practically as efficient to compute as hard version!

Perspectives

Cornell University

- There are many more global constraints, often not flow-based. How to make them soft?

  Examples: circuit, sequence, element, cumulative,...

- Clearly, some violation measures are more refined than others, for example variable-based versus decomposition-based. Are there more suitable violation measures? Can we identify useful relationships between them?

- How to aggregate several soft global constraints?

  soft_alldifferent(X_1,z_1,μ_{var})
  soft_alldifferent(X_2,z_2,μ_{dec})
  soft_gcc(X_3,l,u,z_3,μ_{val})
  soft_regular(X_4,DFA,z_4,μ_{var})
  minimize f(z_1,z_2,z_3,z_4) sum? weighted sum? bound some of the cost variables?

References

R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network Flows.* Prentice Hall, 1993.

P. Baptiste, C. Le Pape, and L. Péridy. Global Constraints for Partial CSPs: A Case-Study of Resource and Due Date Constraints. In M.J. Maher and J.-F. Puget, editors, *Proceedings of the Fourth International Conference on Principles and Practice of Constraint Programming (CP98)*, volume 1520 of *LNCS*, pages 87–101. Springer, 1998.

S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based Constraint Satisfaction and Optimization. *Journal of the ACM*, 44(2):201–236, 1997.

D. Dubois, H. Fargier, and H. Prade. The calculus of fuzzy restrictions as a basis for flexible constraint satisfaction. In *Proceedings of the Second IEEE International Conference on Fuzzy Systems*, volume 2, pages 1131–1136, 1993.

E.C. Freuder and R.J. Wallace. Partial constraint satisfaction. *Artificial Intelligence*, 58(1-3):21–70, 1992.

W.J. van Hoeve. A Hyper-Arc Consistency Algorithm for the Soft Alldifferent Constraint. In M. Wallace, editor, *Proceedings of the Tenth International Conference on Principles and Practice of Constraint Programming (CP 2004)*, volume 3258 of *LNCS*, pages 679–689. Springer, 2004.

W.-J. van Hoeve, G. Pesant, and L.-M. Rousseau. On Global Warming: Flow-Based Soft Global Constraints. *Journal of Heuristics*, 2006. To appear.

J.E. Hopcroft and R.M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973.

References

Cornell University

J. Larrosa. Node and Arc Consistency in Weighted CSP. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence*, pages 48–53. AAAI Press, 2002.

G. Pesant. A Regular Language Membership Constraint for Finite Sequences of Variables. In *Proceedings of the Tenth International Conference on Principles and Practice of Constraint Programming (CP 2004)*, volume 3258 of *LNCS*. Springer, 2004.

J. Petersen. Die Theorie der regulären graphs. *Acta Mathematica*, 15:193–220, 1891.

T. Petit, J.-C. Régin, and C. Bessière. Specific Filtering Algorithms for Over-Constrained Problems. In T. Walsh, editor, *Proceedings of the Seventh International Conference on Principles and Practice of Constraint Programming (CP 2001)*, volume 2239 of *LNCS*, pages 451–463. Springer, 2001.

C.-G. Quimper, A. López-Ortiz, P. van Beek and A. Golynski. Improved Algorithms for the Global Cardinality Constraint. In M. Wallace, editor, *Proceedings of the Tenth International Conference on Principles and Practice of Constraint Programming (CP 2004)*, volume 3258 of *LNCS*, pages 542–556. Springer, 2004.

J.-C. Régin. A Filtering Algorithm for Constraints of Difference in CSPs. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI)*, volume 1, pages 362–367. AAAI Press, 1994.

J.-C. Régin. Generalized Arc Consistency for Global Cardinality Constraint. In *Proceedings of AAAI/IAAI*, volume 1, pages 209–215. AAAI Press/The MIT Press, 1996.

References

Cornell University

J.-C. Régin. Arc Consistency for Global Cardinality Constraints with Costs. In J. Jaffar, editor, *Proceedings of the Fifth International Conference on Principles and Practice of Constraint Programming (CP'99)*, volume 1713 of *LNCS*, pages 390–404. Springer, 1999.

J.-C. Régin, T. Petit, C. Bessière, and J.-F. Puget. An Original Constraint Based Approach for Solving over Constrained Problems. In R. Dechter, editor, *Proceedings of the Sixth International Conference on Principles and Practice of Constraint Programming (CP 2000)*, volume 1894 of *LNCS*, pages 543–548. Springer, 2000.

J.-C. Régin. Cost-Based Arc Consistency for Global Cardinality Constraints. *Constraints*, 7:387–405, 2002.

Z. Ruttkay. Fuzzy constraint satisfaction. In *Proceedings of the First IEEE Conference on Evolutionary Computing*, pages 542–547, 1994.

T. Schiex. Possibilistic Constraint Satisfaction Problems or "How to handle soft constraints ?". In *Proceedings of the 8th Annual Conference on Uncertainty in Artificial Intelligence*, pages 268–275. Morgan Kaufmann, 1992.

T. Schiex, H. Fargier, and G. Verfaillie. Valued Constraint Satisfaction Problems: Hard and Easy Problems. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 631–639. Morgan Kaufmann, 1995.

R. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1:146–160, 1972.

A. Zanarini, M. Milano, and G. Pesant. Improved Algorithm for the Soft Global Cardinality Constraint. In J.C. Beck and B.M. Smith, editors, *Proceedings of CPAIOR 2006*, volume 39990 of *LNCS*, pages 288–299. Springer, 2006.