Adaptive Parameterized Consistency for Non-Binary CSPs by Counting Supports

R.J.Woodward[1,2], A.Schneider[1], B.Y.Choueiry[1], and C.Bessiere[2]
[1]Constraint Systems Laboratory • University of Nebraska-Lincoln • USA
[2]LIRMM-CNRS • University of Montpellier • France

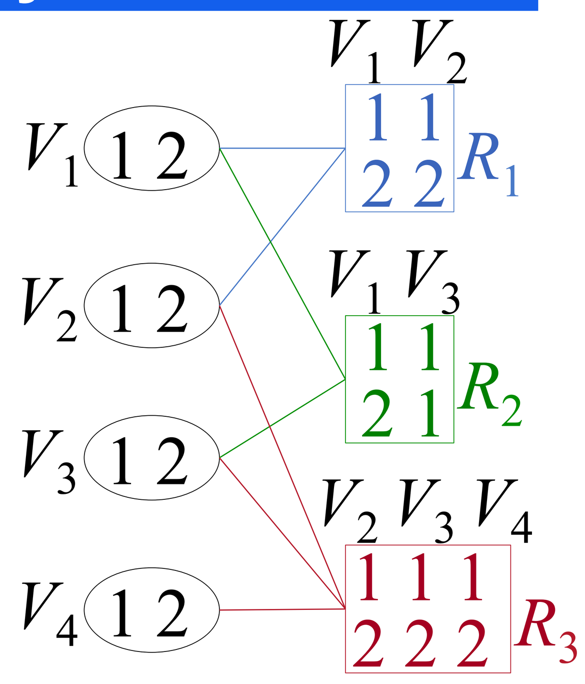
1. Contributions

1. Extend p-stability for AC to GAC (to operate on non-binary CSPs)
2. Provide a more precise mechanism for computing it
3. Algorithm for enforcing *apc*-LC

2. Local Consistency

Generalized Arc Consistency (GAC) ensures any value in the domain of any variable in the scope of every relation can be extended to a tuple satisfying the relation. E.g., filters value 2 from V_3.

STR ensures GAC, filtering both domains and relations. E.g., filters value 2 from V_3, tuple ⟨2,1⟩ from R_2.



***m*-wise Consistency, R(*,*m*)C** ensures every tuple in every relation can be extended to every combination of $m-1$ other relations. R(*,3)C filters tuple ⟨2,2⟩ from R_1, ⟨2,1⟩ from R_2 and ⟨2,2,2⟩ from R_3. Projecting relations onto domains filters 2 from every variable.

Pairwise Consistency ≡ R(*,2)C.

3. Parameterized Adaptive Consistency

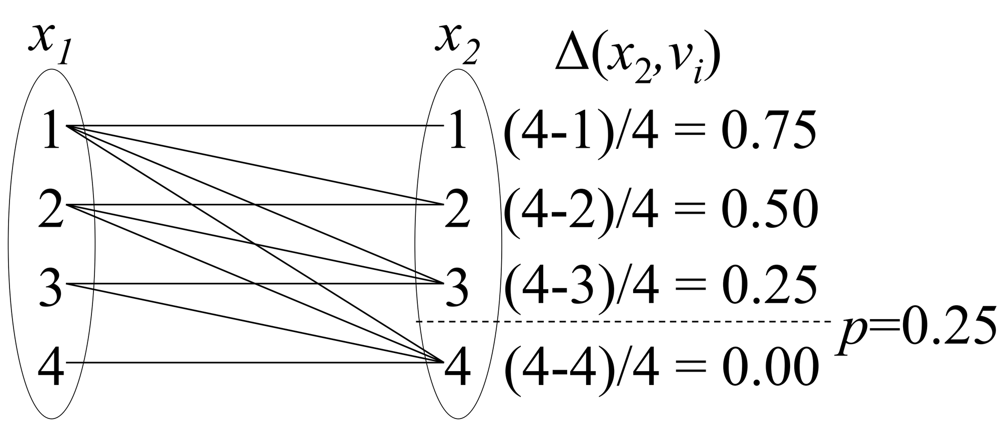
p-stability for AC [Balafrej+ CP2013]

Relative position of v_i in $dom^o(x_i)$:

$$\Delta(x_i, v_i) = \frac{|dom^o(x_i)| - rank(v_i, dom^o(x_i))}{|dom^o(x_i)|}$$

For parameter p, $\langle x_i, v_i\rangle$ is ***p*-stable for AC** for c_{ij} on x_i,x_j if it has AC-support $\langle x_j, v_j\rangle$ with $\Delta(x_j, v_j) \geq p$.

⟨x_1,1⟩ is 0.25-stable for AC
⟨x_1,2⟩ is 0.25-stable for AC
⟨x_1,3⟩ is 0.25-stable for AC
⟨x_1,4⟩ not 0.25-stable for AC



Issue: Position is not a precise measure of support.

p-stability for GAC

$\langle x_i, v_i\rangle$ is ***p*-stable for GAC** if for every constraint c_j on x_i

$$\frac{|\sigma_{x_i=v_i}(R_j)|}{|R_j^o|} \geq p$$

Where R_j^o is the original, unfiltered relation.

	x_1	x_2
0	1	1
1	1	2
2	1	3
3	1	4
4	2	2
5	2	3
6	2	4
7	3	3
8	3	4
9	4	4

gacSupports$[R_j](\langle x_i,v_i\rangle)$: data structure to store $\sigma_{x_i=v_i}(R_j)$.

gacSupports$[R_j](\langle x_1,1\rangle)$={0,1,2,3} ⟨$x_1$,1⟩ is 0.25-stable for GAC ($\frac{4}{10} \geq 0.25$)
gacSupports$[R_j](\langle x_1,2\rangle)$={4,5,6} ⟨$x_1$,2⟩ is 0.25-stable for GAC ($\frac{3}{10} \geq 0.25$)
gacSupports$[R_j](\langle x_1,3\rangle)$={7,8} ⟨$x_1$,3⟩ not 0.25-stable for GAC ($\frac{2}{10} < 0.25$)
gacSupports$[R_j](\langle x_1,4\rangle)$={9} ⟨$x_1$,4⟩ not 0.25-stable for GAC ($\frac{1}{10} < 0.25$)

Parameterized Adaptive Consistency

- LC: A local consistency stronger than (G)AC
- pc-LC: $\forall\langle x_i,v_i\rangle$ and c_j on x_i, $\langle x_i,v_i\rangle$ is
  - *p*-stable for (G)AC on c_j or
  - LC on c_j
- *apc*-LC: adaptive version of *pc*-LC that uses the weight of a constraint, $w(c_j)$, like *dom/wdeg*

$$p(c_j) = \frac{w(c_j) - \min_{c_k \in \mathcal{C}}(w(c_k))}{\max_{c_k \in \mathcal{C}}(w(c_k)) - \min_{c_k \in \mathcal{C}}(w(c_k)) + 1}$$

4. Algorithm & Empirical Evaluations

- Generate the *gacSupports* at pre-processing
- $\forall c_j$, $\forall x_i$ in c_j, $\forall v_i$ in $dom(x_i)$
  - Enforce LC when $\langle x_i,v_i\rangle$ is not p-stable for GAC
  - Enforce STR when $\langle x_i,v_i\rangle$ has no support

Algorithm 1: Living-STR(c_i): set of variables

Input: c_j: a constraint of $\mathcal{P}$
Output: Set of variables in $scope(c_j)$ whose domains have been modified

```
X_modified ← ∅
foreach x_i ∈ scope(c_j) | x_i ∉ past(P) do
    foreach v_i ∈ dom(x_i) do
        if |gacSupports[R_j](⟨x_i,v_i⟩)| ≠ 0 and |gacSupports[R_j](⟨x_i,v_i⟩)| / |R_j^o| ≱ p(c_j) then
            Apply-LC(R_j, gacSupports[R_j](⟨x_i,v_i⟩))
        if |gacSupports[R_j](⟨x_i,v_i⟩)| = 0 then
            foreach c_k ∈ cons(x_i) do
                delTuples(c_k, gacSupports[R_k](⟨x_i,v_i⟩), |past(P)|)
            dom(x_i) ← dom(x_i) \ {v_i}
            if dom(x_i) = ∅ then throw INCONSISTENCY
            X_modified ← X_modified ∪ {x_i}
return X_modified
```

Experimental Results

apc-R(*,2)C can solve more instances and quicker than both STR and R(*,2)C.

Total number of instance: 623	STR	R(*,2)C	*apc*-R(*,2)C
Number of instances solved by	504	550	**552**
# instances solved only by	10	5	0
# instances solved by STR, but missed by	0	18	11
# instances solved by R(*,2)C, but missed by	64	0	6
# instances solved by *apc*-R(*,2)C, but missed by	59	8	0
Average CPU time (sec.) over 458 instances	328.41	378.12	**313.31**
Median CPU time (sec.) over 458 instances	7.23	17.35	**7.21**

Benchmark results where *apc*-R(*,2)C performs the best, is competitive, and is the worst. Also included, a benchmark that could not be solved by STR.

benchmark	# Instances	Completed				Average CPU Time (sec)			Median CPU Time (sec)		
		STR	R(*,2)C	*apc*-R(*,2)C	All	STR	R(*,2)C	*apc*-R(*,2)C	STR	R(*,2)C	*apc*-R(*,2)C
a) *apc*-R(*,2)C is the best											
aim-50	24	24	24	24	24	**0.04**	0.07	**0.04**	0.02	0.04	**0.03**
allIntervalSeries	25	22	22	22	22	7.09	141.85	**6.00**	0.13	0.31	0.12
jnhSat	16	16	16	16	16	13.07	357.66	**11.74**	8.15	142.24	7.21
modifiedRenault	50	50	50	50	50	6.39	11.17	**6.29**	7.24	8.79	6.98
rand-3-20-20	50	31	43	41	31	1,666.10	939.88	**932.77**	1,211.50	822.54	811.74
b) *apc*-R(*,2)C is competitive											
aim-100	24	24	24	24	24	0.38	**0.26**	0.41	0.18	0.25	**0.16**
aim-200	24	22	24	24	22	414.48	**6.52**	286.27	2.39	1.37	**2.60**
jnhUnsat	34	34	34	34	34	**13.61**	294.77	13.95	10.74	153.50	**9.78**
lexVg	63	63	63	63	63	**69.81**	341.87	338.74	0.50	1.38	0.89
pret	8	4	4	4	4	**117.89**	347.03	136.04	115.81	354.82	145.70
rand-3-20-20-fcd	50	39	48	47	39	928.06	**546.84**	615.23	501.30	422.24	464.00
rand-8-20-5	20	9	20	20	9	2,564.94	**355.57**	372.76	1,987.35	314.26	**261.68**
rand-10-20-10	20	12	12	12	12	6.72	**1.67**	2.76	6.40	1.66	2.75
ssa	8	6	5	6	5	**64.60**	100.64	69.59	1.51	1.60	1.58
TSP-25	15	13	10	13	10	**232.38**	1,072.72	743.33	69.00	211.41	131.69
ukVg	65	37	31	34	31	**166.82**	796.90	421.35	36.29	54.65	**30.39**
varDimacs	9	6	6	6	6	**89.23**	587.55	319.20	1.56	6.43	2.94
wordsVg	65	65	58	58	58	**119.76**	532.05	400.22	0.39	0.95	0.59
c) *apc*-R(*,2)C is the worst											
dubois	13	7	8	6	6	1,000.54	**451.91**	1,456.01	552.13	255.25	779.57
TSP-20	15	15	15	15	15	**101.20**	318.37	335.13	23.32	61.55	**46.34**
d) Not solved by STR											
dag-rand	25	0	25	25	0	-	**123.70**	149.64	-	124.47	151.33

Average number of calls to STR and R(*,2)C in *apc*-R(*,2)C

- On allInternvalSeries, no R(*,2)C calls because solved at pre-processing (no weight set).
- Sometimes, there are more STR calls, others, more R(*,2)C calls. Thus, *apc*-R(*,2)C finds a good mix between STR and R(*,2)C for each instance. Success!

benchmark	STR Calls	R(*,2)C Calls	benchmark	STR Calls	R(*,2)C Calls
a) *apc*-R(*,2)C is the best			b) *apc*-R(*,2)C is competitive		
aim-50	456,823	39,491	aim-100	7,731,585	894,353
allIntervalSeries	38,281,694	0	aim-200	1,160,334,482	163,177,907
jnhSat	22,119,135	599,080	jnhUnsat	51,688,166	1,918,781
modifiedRenault	4,618,778	601,641	lexVg	564,010,457	2,180,503,026
rand-3-20-20	489,441,126	3,480,216,943	pret	422,987,946	13,973,748
			rand-3-20-20-fcd	455,664,100	2,956,467,994
c) *apc*-R(*,2)C is the worst			rand-8-20-5	77,470,561	184,764,543
dubois	3,343,830,604	4,668,288	rand-10-20-10	72,608	3,972
TSP-20	622,949,698	991,590,957	ssa	156,631,370	11,689,961
			TSP-25	2,903,953,315	3,947,391,769
			ukVg	341,565,892	1,002,334,753
d) Not solved by STR			varDimacs	720,843,958	84,123,204
dag-rand	359,248	21,870	wordsVg	514,840,737	2,052,367,934

Constraint Systems Laboratory
University of Nebraska Lincoln

Experiments were conducted on the equipment of the Holland Computing Center at the University of Nebraska-Lincoln. R.J. Woodward was supported by a National Science Foundation (NSF) Graduate Research Fellowship Grant No.1041000 and Chateaubriand Fellowship. This research is supported by NSF Grant No. RI-111795 and EU project ICON (FP7-284715).

Paper: CP 2014. September 2nd, 2014