

# Heuristics for Dynamically Adapting Constraint Propagation in Constraint Programming



---

**Kostas Stergiou**

AI Lab

University of the Aegean

Greece

*CPAIOR'09 Workshop on Bound reduction techniques for CP and MINLP*



# Talk Outline

---

- **CP: some preliminaries**
- **Constraint activity during search**
- **Heuristics for dynamically adapting constraint propagation**
- **Experimental results**
- **Conclusions**



# Constraint Programming

---

- A **Constraint Satisfaction Problem (CSP)** is a tuple  $(X, D, C)$ , where:
  - **X** is a set of **n** variables  $\{x_1, x_2, \dots, x_n\}$
  - **D** is a set of domains  $\{D(x_1), D(x_2), \dots, D(x_n)\}$ 
    - *finite integer domains in our case*
  - **C** is a set of constraints  $\{c_1, c_2, \dots, c_m\}$  between variables of **X**
    - Each constraint  $c$  on variables  $x_i, \dots, x_j$  restricts the possible combinations of values that the variables can take
  - In a **constraint optimization problem** we also have an objective function



# Constraint Programming

- Typical constraint solvers are based on backtracking depth-first search coupled with constraint propagation
- After a branching decision is made (e.g. variable assignment) a list  $Q$  of constraints to be revised is formed

```
while  $Q$  is not empty  
  remove constraint  $c_i$  from  $Q$   
  for each variable  $x_j$  in  $c_i$   
     $\text{Revise}(x_j, c_i)$   
    if domain of  $x_j$  is empty then return FAIL  
    if domain of  $x_j$  is modified then put in  $Q$  each  $c_m$  that involves  $x_j$   
return 1
```

- The Revise function implements a constraint propagation algorithm for domain reduction
  - E.g. an arc consistency algorithm or a propagator for a global constraint



# Constraint Programming

---

- In many solvers constraints are propagated using the same propagation method throughout search
  - E.g. a MAC solver applies arc consistency on all constraints throughout search
- More sophisticated solvers may use an array of propagators for certain constraints
  - Usually such propagators apply GAC or lesser consistencies (e.g. bounds consistency)
    - For example, the constraint solver may be equipped with a gac and a bounds consistency algorithm for the common all-different constraint
  - But the choice of propagator is typically made statically prior to search
- In this presentation we are interested in problems where all constraints are binary



# Constraint Activity

## ■ Definitions:

- A constraint is *deletion-active* if it causes at least one domain reduction
- A constraint is *DWO-active* if it causes at least one domain wipeout

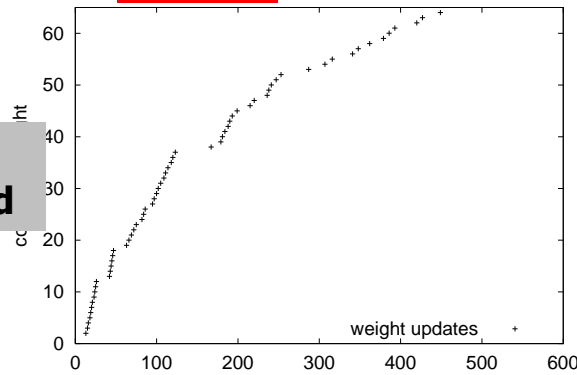
```
while  $Q$  is not empty
  remove constraint  $c_i$  from  $Q$ 
  for each variable  $x_j$  in  $c_i$ 
    Revise( $x_j$ ,  $c_i$ )
    if domain of  $x_j$  is empty then return FAIL
    if domain of  $x_j$  is modified then put in  $Q$  each  $c_m$  that involves  $x_j$ 
return 1
```

- It has been observed that not all constraints are deletion or/and DWO active during the run of a search algorithm
  - Particularly in structured problems only a few constraints are active in some cases
  - Many constraints are revised over and over with no fruitful result (redundant revisions)
- **Question:**
  - How is the activity of a constraint distributed during its revisions?

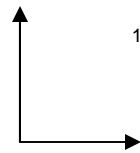
# Constraint Activity - DWOs

- Answer:** In structured problems DWOs are highly clustered

driver

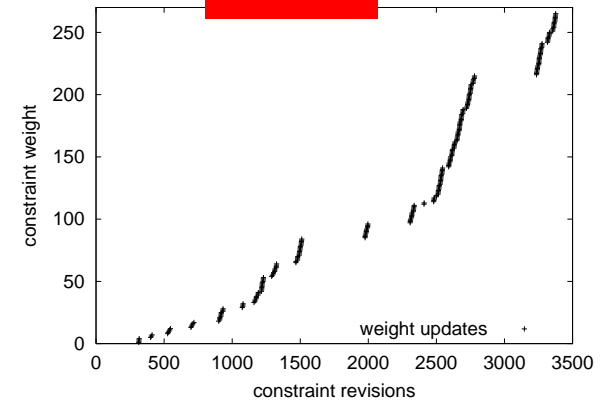


DWO caused

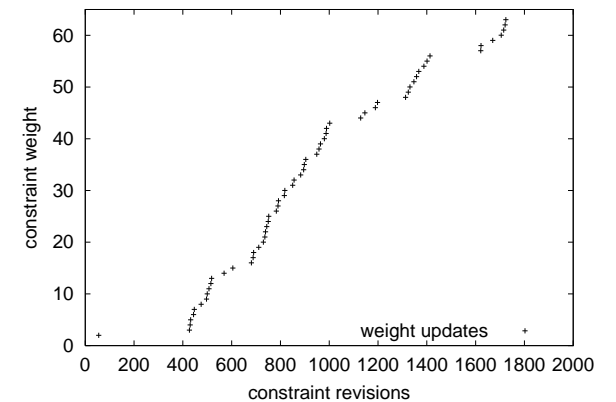
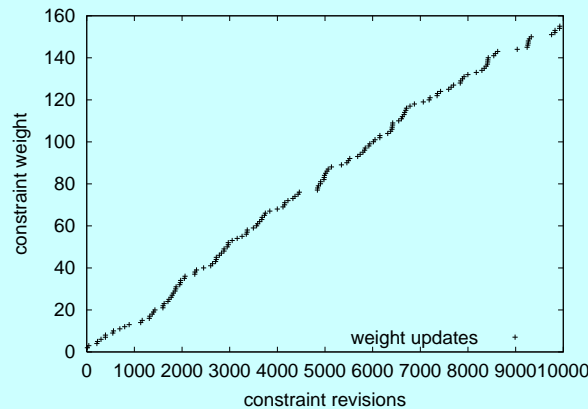


constraint revision

RLFAP



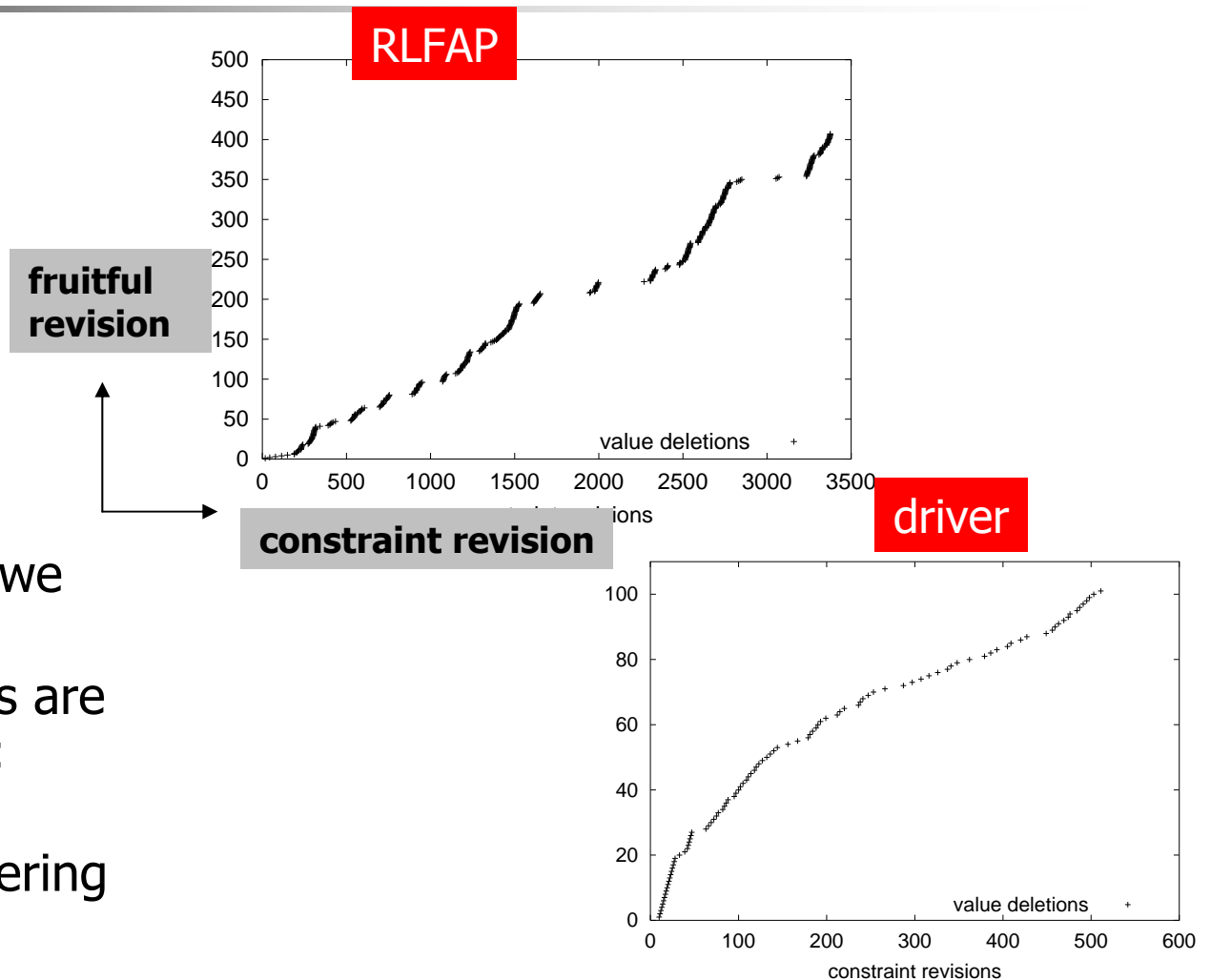
Not so in random problems where they are uniformly distributed



quasigroup completion

# Constraint Activity – Domain reductions

- **Answer:** In structured problems fruitful revisions (i.e. ones causing domain reductions) are also highly clustered
- Results given so far we obtained using **MAC**
  - i.e. all constraints are revised using arc consistency
- and the variable ordering heuristic dom/wdeg



# Constraint Activity - Results

- We run the *Expectation Maximization* (EM) clustering algorithm on some problems
  - We measure the “clustering” of propagation events (deletions and DWOs) for 20 sample constraints in each problem

DWO-active constraints /  
total constraints

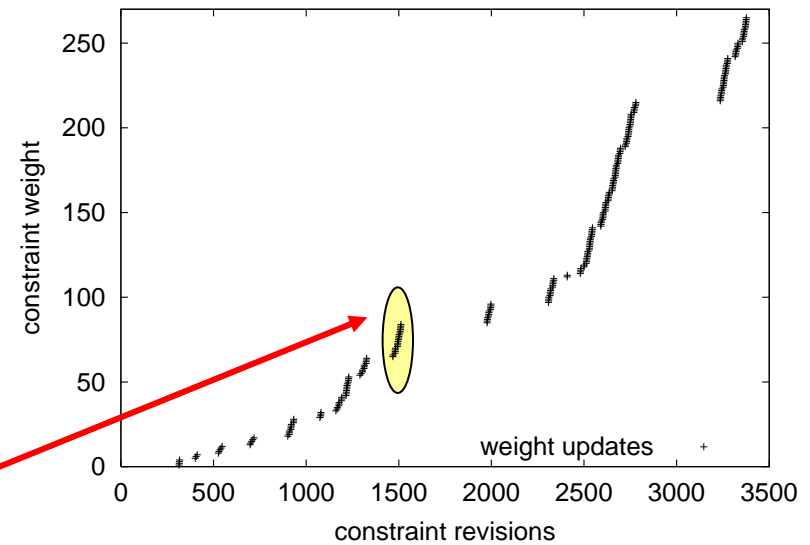
- The SD in a cluster gives the average distance from the cluster's centroid.
- The median SD is quite low in structured problems indicating that DWO-revisions are closely grouped together
- The mean is higher because of outliers

instance	#constraints	avg #clusters	avg size	mean SD	median SD
scen11	27/4103	6.66	10.82	41.09	16.12
driver-08c	87/9321	2.44	12.62	38.50	25.11
qcp15-120-0	554/3150	12.87	15.26	226.12	129.28
frb35-17-0	233/262	7.20	19.38	1856.70	1649.05

**Notice the difference  
in the random one!**

# Constraint Activity - Structure

- The “clustering” of domain reductions seems to depend on the structure of problems
  - It does not occur in unstructured random problems
  - This can be explained by the presence of hard sub-problems in structured CSPs
    - Once search reaches such a sub-problem it may cause repeated domain reductions
      - as the search algorithm assigns variables involved in the sub-problem and propagates these assignments





# Constraint Activity

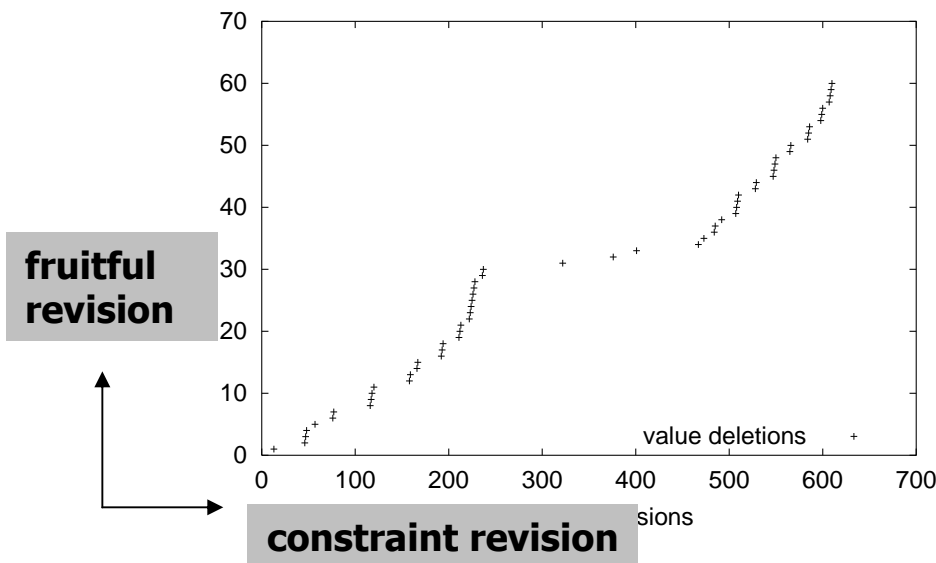
---

- **Question:** What is the dependence of the “clustering” phenomenon on the various features of a constraint solver?
  - Propagation method
  - Variable ordering heuristic
  - Restarts
  - Implementation of propagation list
- **Answer:** It seems to be independent!
  - The phenomenon persists when experimenting with different combinations of the above features

# Constraint Activity

- We tried using maxRPC instead of AC on all constraints
  - maxRPC is stronger (it achieves more domain reductions)
  - Again there appeared a clear clustering of domain reductions

- Similar when changing the variable ordering heuristic
  - dom, Brelaz, dom/deg, dom/fdeg
- and when adding restarts
- and when changing the implementation of the propagation list
  - constraint-oriented vs. variable-oriented
  - LIFO vs. FIFO





# From clustering of constraint activity to dynamically adapting propagation

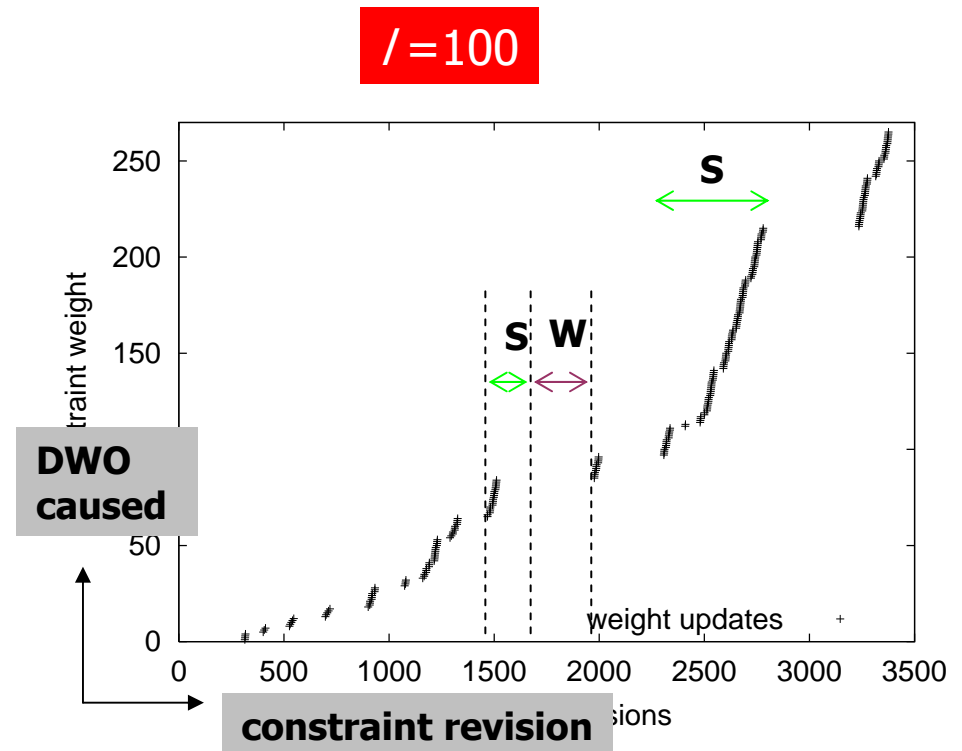
---

- We propose heuristics that exploit the clustering of propagation events to dynamically adapt the propagation method for individual constraints
- We only consider switching between a weak (**W**) - *and cheap* - and a strong (**S**) – *and expensive* – method
  - Case study with *AC* and *maxRPC*
- The heuristics monitor propagation events and switch between **W** and **S** on individual constraints when certain conditions are met
  - they are lightweight and very easy to implement!
  - Once procedure *revise(c)* is called the given heuristic determines whether *c* will be revised with **W** or **S**
- Heuristics can be fully automated (no user involvement) or semi-automated (user specifies a bound)

# Heuristics – H1

## H1( $\lambda$ ): *semi automated - DWO monitoring*

- H1 monitors and counts the revisions and DWOs of the constraints
- A constraint  $c$  is made  $S$  if the number of calls to  $Revise(c)$  since the last time it caused a DWO is less or equal to a (user defined) threshold  $\lambda$ . Otherwise, it is made  $W$ .

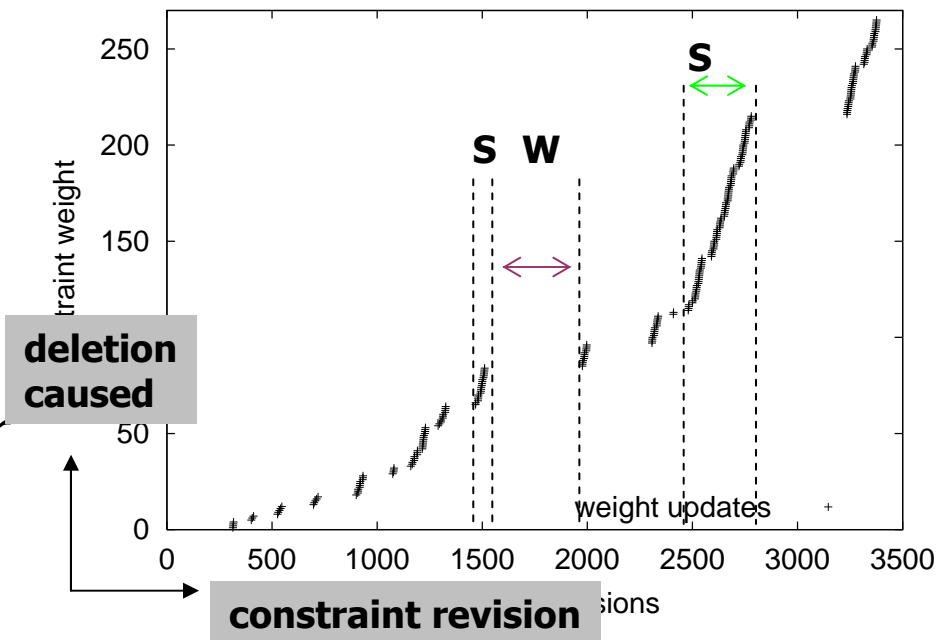


# Heuristics – H2

## H2: *fully or semi automated - deletion monitoring*

- H2 monitors revisions and value deletions.
- A constraint  $c$  is made  $S$  as long as the last call to  $Revise(c)$  deleted at least one value. Otherwise, it is made  $W$ .
  - H2 can be semi automated in a similar way to H1 by allowing for a number  $\ell$  of redundant revisions after the last fruitful revision. If  $\ell$  is set to 0 we get the fully automated H2

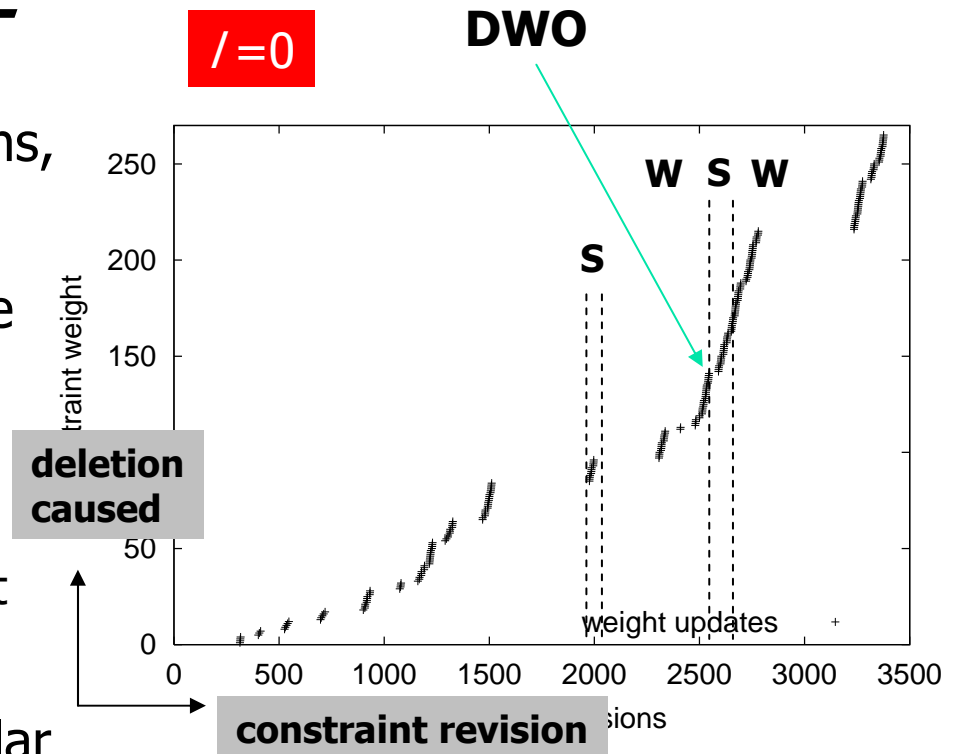
$\ell = 0$



# Heuristics – H3

## H3: *fully or semi automated - hybrid*

- H3 monitors revisions, value deletions, and DWOs.
- A constraint  $c$  is made  $S$  as long as revising it with  $S$  deletes at least one value. Otherwise, it is made  $W$ .
- Once the constraint causes a DWO, the monitoring of  $S$ 's effects starts again.
  - If this is not done then after the first DWO the constraint will be always propagated using  $W$ .
- H3 can be semi automated in a similar way to H1 and H2





# Heuristics – H4

---

- **H4: *fully or semi automated - deletion monitoring***
- H4 monitors value deletions.
- For any constraint  $c$ , H4 applies  $W$  until a value is deleted. In this case  $c$  is made  $S$ .
  - I.e.  $S$  is applied on the remaining available values in  $D(x)$ .
- H4 can be semi automated by insisting that  $S$  is applied only if a proportion  $p$  of  $x$ 's values have been deleted by  $W$  during the current revision of  $c$ .
  - With high values of  $p$   $S$  will be applied only when it is likely that it will cause a DWO.

# Disjunctive and Conjunctive Heuristic Combinations

- Importantly, the heuristics defined above can be combined either disjunctively or conjunctively in various ways
- For example, heuristic  $H_{124}^{\vee}$  applies **S** on a constraint whenever the condition specified by either H1, H2, or H4 holds.
- Heuristic  $H_{24}^{\wedge}$  applies **S** when both the conditions of H2 and H4 hold.
- We can choose a disjunctive or conjunctive combination depending on whether we want **S** applied to a greater or lesser extent respectively.



# Experiments

---

- Case study with *Arc Consistency* (W) and *max Restricted Path Consistency* (S)
  - maxRPC is strictly stronger than AC
  - We compare algorithms that apply AC and maxRPC to algorithms that apply single or combined heuristics to determine how constraints are propagated
  - Experiments on various structured and random problems
    - radio links frequency assignment (RLFAP), langford, black hole, driver, hanoi, quasigroup completion, quasigroup with holes, graph coloring, composed random, forced random, geometric random

# Experimental Results

- From structured RLFAPs

- The adaptive heuristics reduce node visits and run times in most cases

instance		AC	maxRPC	H <sub>1</sub>	H <sub>2</sub>	H <sub>3</sub>	H <sub>4</sub>	H <sub>14</sub> <sup>y</sup>	H <sub>124</sub> <sup>y</sup>
s11	n	2864	1334	1175	1842	1432	1678	1358	1360
	t	6.9	24.2	3.7	6.7	5.5	6.0	4.9	4.9
s11-f9	n	108184	37663	35102	47552	39312	53338	38202	37743
	t	539.6	3478.3	170.4	335.4	183.3	274.8	205.2	212.7
s11-f10	n	8576	2098	2197	2675	1938	3849	2462	2467
	t	30.2	93.8	11.6	18.8	10.2	13.9	11.4	11.3
s11-f12	n	6678	1923	1750	2804	1763	3095	1953	1921
	t	19.7	101.7	8.6	14.5	9.4	14.7	11.0	10.6
s02-f25	n	11998	5262	3114	10802	2938	12961	4367	4922
	t	9.3	65.1	5.6	16.0	5.5	15.2	9.3	10.3
s03-f11	n	8314	880	1047	4830	2762	4518	2068	1489
	t	26.4	24.7	5.6	20.2	11.8	17.2	12.5	9.5
g08-f10	n	11948	6342	6650	6423	9540	4863	4474	4119
	t	34.5	147.1	21.9	19.4	26.8	13.9	16.3	16.2
g08-f11	n	9996	629	753	960	748	713	608	619
	t	35.9	18.7	4.3	4.5	4.8	3.6	3.6	3.6
g14-f27	n	11602	926	10759	2237	9698	2877	2750	2750
	t	13.0	2.5	15.3	3.1	17.2	3.3	3.1	3.1

When maxRPC is better than AC

When maxRPC has no significant impact on node visits, the heuristics are better or competitive with AC

When maxRPC reduces nodes considerably but is slower than AC

# Experimental Results

- From various structured problems

instance		AC	maxRPC	H <sub>2</sub>	H <sub>4</sub>	H <sub>24</sub> <sup>Y</sup>	H <sub>124</sub> <sup>Y</sup>
queen8-8-8	n	-	1458	2807	-	5863	4244
	t	>1h	3.15	2.9	>1h	5.1	2.7
games120-9	n	3208852	1392922	5511126	2265133	1604133	1452449
	t	403.7	432.3	834.3	293.7	216.1	195.9
driverlogw-08	n	3814	785	1003	3417	855	903
	t	13.2	25.5	6.9	9.2	6.1	6.2
driverlogw-09	n	14786	8342	10802	10627	8859	8895
	t	239.2	265.8	152.9	167.1	137.8	141.2
qcp-15-120-0	n	108336	21926	35394	101901	29990	27167
	t	98.4	43.3	39.9	83.9	33.4	28.3
qcp-15-120-5	n	387742	80424	84193	370461	81269	112290
	t	422.0	201.0	118.2	369.4	117.7	147.0
qcp-15-120-10	n	1136801	52112	58325	152497	76399	68046
	t	1178.0	113.6	65.1	145.1	88.6	71.2
qwh-20-166-0	n	104288	20236	15550	62993	15591	24725
	t	269.1	86.9	42.3	140.0	46.0	78.2
qwh-20-166-1	n	132842	22688	29681	66775	25147	39435
	t	355.4	111.4	88.2	151.1	78.5	116.7

- In some of these problems maxRPC is much more efficient than AC
  - The heuristics, except H<sub>4</sub>, can offer further improvement
- Overall, the disjunctive combinations are more robust than single heuristics



# Experimental Results

---

- And some unstructured random problems...

instance		AC	maxRPC	H <sub>2</sub>	H <sub>4</sub>	H <sub>24</sub> <sup>v</sup>	H <sub>124</sub> <sup>v</sup>
frb35-17	n	23782	14920	15022	21182	15064	14642
	t	<b>13.5</b>	107.5	47.5	16.1	48.4	46.8
frb40-19	n	40058	20073	24446	32393	19722	22752
	t	<b>24.9</b>	151.6	76.8	27.9	63.4	76.1
geo50-20-75	n	227535	112785	148853	221211	142416	141726
	t	<b>218.9</b>	2089.4	765.7	247.1	748.3	750.1


- Here AC is clearly the winner
  - The only competitive heuristic is H4 which does not target clusters of propagation activity



# Conclusions & Future Work

---

- Propagation events (domain reductions and wipeouts) caused by individual constraints typically occur in clusters of close revisions
  - This phenomenon appears to be independent of important parameters of CP solvers
    - constraint propagation method, variable ordering heuristics, restarts, implementation of propagation listbut depends heavily on the presence of some structure in the problem
- We proposed heuristics for dynamically adapting the propagation of individual constraints
  - The heuristics monitor propagation events and change the propagation method when certain conditions are met
- Experimental results on structured problems are promising
- In the future:
  - non-binary consistencies and global constraints



*THANK  
YOU!*