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
A **Constraint Satisfaction Problem** (CSP) is a tuple \((X, D, C)\), where:

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

```plaintext
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
```

The Revise function implements a constraint propagation algorithm for domain reduction.

- E.g. an arc consistency algorithm or a propagator for a global constraint
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 were 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.

Not so in random problems where they are uniformly distributed.
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
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.

- 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.

Notice the difference in the random one!
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.
**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
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
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 rvice(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(\(l\)): 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 \(l\). 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 of redundant revisions after the last fruitful revision. If $l$ is set to 0 we get the fully automated H2.
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 $H_{124}$ applies $S$ on a constraint whenever the condition specified by either H1, H2, or H4 holds.
- Heuristic $H_{24}$ 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

<table>
<thead>
<tr>
<th>instance</th>
<th>AC</th>
<th>maxRPC</th>
<th>H1</th>
<th>H2</th>
<th>H3</th>
<th>H4</th>
<th>H14</th>
<th>H124</th>
</tr>
</thead>
<tbody>
<tr>
<td>s11</td>
<td>6.9</td>
<td>24.2</td>
<td>3.7</td>
<td>6.7</td>
<td>5.5</td>
<td>6.0</td>
<td>4.9</td>
<td>4.9</td>
</tr>
<tr>
<td>s11-f9</td>
<td>108184</td>
<td>37663</td>
<td>2197</td>
<td>26759</td>
<td>1938</td>
<td>3849</td>
<td>2462</td>
<td>2467</td>
</tr>
<tr>
<td>s11-f10</td>
<td>30.2</td>
<td>93.8</td>
<td>11.6</td>
<td>18.8</td>
<td>10.2</td>
<td>13.9</td>
<td>11.4</td>
<td>11.3</td>
</tr>
<tr>
<td>s11-f12</td>
<td>19.7</td>
<td>101.7</td>
<td>8.6</td>
<td>14.5</td>
<td>9.4</td>
<td>14.7</td>
<td>11.0</td>
<td>10.6</td>
</tr>
<tr>
<td>s01-f25</td>
<td>11998</td>
<td>5262</td>
<td>3114</td>
<td>10802</td>
<td>2938</td>
<td>12961</td>
<td>4367</td>
<td>4922</td>
</tr>
<tr>
<td>s03-f11</td>
<td>26.4</td>
<td>24.7</td>
<td>5.6</td>
<td>16.0</td>
<td>5.5</td>
<td>15.2</td>
<td>9.3</td>
<td>10.3</td>
</tr>
<tr>
<td>g08-f10</td>
<td>11948</td>
<td>6342</td>
<td>6650</td>
<td>6423</td>
<td>9540</td>
<td>4863</td>
<td>4474</td>
<td>4119</td>
</tr>
<tr>
<td>g08-f11</td>
<td>35.9</td>
<td>18.7</td>
<td>4.3</td>
<td>4.5</td>
<td>4.8</td>
<td>3.6</td>
<td>3.6</td>
<td>3.6</td>
</tr>
<tr>
<td>g14-f27</td>
<td>11602</td>
<td>926</td>
<td>10759</td>
<td>2237</td>
<td>9698</td>
<td>2877</td>
<td>2750</td>
<td>2750</td>
</tr>
</tbody>
</table>
Experimental Results

- From various structured problems

- In some of these problems maxRPC is much more efficient than AC
  - The heuristics, except $H_4$, can offer further improvement

- Overall, the disjunctive combinations are more robust than single heuristics

<table>
<thead>
<tr>
<th>instance</th>
<th>AC</th>
<th>maxRPC</th>
<th>$H_2$</th>
<th>$H_4$</th>
<th>$H_{54}$</th>
<th>$H_{124}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>queen8-8-8</td>
<td>t</td>
<td>n</td>
<td>&gt;1h</td>
<td>1458</td>
<td>2807</td>
<td>5863</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3.15</td>
<td>2.9</td>
<td>5.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&gt;1h</td>
<td></td>
<td>2.7</td>
</tr>
<tr>
<td>games120-9</td>
<td>3208852</td>
<td>1392922</td>
<td>5511126</td>
<td>2265133</td>
<td>1604133</td>
<td>1452449</td>
</tr>
<tr>
<td></td>
<td>403.7</td>
<td>432.3</td>
<td>834.3</td>
<td>293.7</td>
<td>216.1</td>
<td>195.9</td>
</tr>
<tr>
<td>driverlogw-08</td>
<td>3814</td>
<td>785</td>
<td>1003</td>
<td>3417</td>
<td>855</td>
<td>903</td>
</tr>
<tr>
<td></td>
<td>13.2</td>
<td>25.5</td>
<td>6.9</td>
<td>9.2</td>
<td>6.1</td>
<td>6.2</td>
</tr>
<tr>
<td>driverlogw-09</td>
<td>14786</td>
<td>8342</td>
<td>10802</td>
<td>10627</td>
<td>8859</td>
<td>8895</td>
</tr>
<tr>
<td></td>
<td>239.2</td>
<td>265.8</td>
<td>152.9</td>
<td>167.1</td>
<td>137.8</td>
<td>141.2</td>
</tr>
<tr>
<td>qcp-15-120-0</td>
<td>108336</td>
<td>21926</td>
<td>35394</td>
<td>101901</td>
<td>29990</td>
<td>27167</td>
</tr>
<tr>
<td></td>
<td>98.4</td>
<td>43.3</td>
<td>39.9</td>
<td>83.9</td>
<td>33.4</td>
<td>28.3</td>
</tr>
<tr>
<td>qcp-15-120-5</td>
<td>387742</td>
<td>80424</td>
<td>84193</td>
<td>370461</td>
<td>81269</td>
<td>112290</td>
</tr>
<tr>
<td></td>
<td>422.0</td>
<td>201.0</td>
<td>118.2</td>
<td>369.4</td>
<td>117.7</td>
<td>147.0</td>
</tr>
<tr>
<td>qcp-15-120-10</td>
<td>1136801</td>
<td>52112</td>
<td>58325</td>
<td>152497</td>
<td>76399</td>
<td>68046</td>
</tr>
<tr>
<td></td>
<td>1178.0</td>
<td>113.6</td>
<td>65.1</td>
<td>145.1</td>
<td>88.6</td>
<td>71.2</td>
</tr>
<tr>
<td>qwh-20-166-0</td>
<td>104288</td>
<td>20236</td>
<td>15550</td>
<td>62993</td>
<td>15591</td>
<td>24725</td>
</tr>
<tr>
<td></td>
<td>269.1</td>
<td>86.9</td>
<td>42.3</td>
<td>140.0</td>
<td>46.0</td>
<td>78.2</td>
</tr>
<tr>
<td>qwh-20-166-1</td>
<td>132842</td>
<td>22688</td>
<td>29681</td>
<td>66775</td>
<td>25147</td>
<td>39435</td>
</tr>
<tr>
<td></td>
<td>355.4</td>
<td>111.4</td>
<td>88.2</td>
<td>151.1</td>
<td>78.5</td>
<td>116.7</td>
</tr>
</tbody>
</table>
Experimental Results

- And some unstructured random problems...

- 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 list
    - but 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!