A Portfolio Approach for Enforcing Minimality in a Tree Decomposition

Daniel J. Geschwender\textsuperscript{1,2}  R.J. Woodward\textsuperscript{1,2}  B.Y. Choueiry\textsuperscript{1,2}  S. D. Scott\textsuperscript{2}

\textsuperscript{1}Constraint Systems Laboratory
\textsuperscript{2}Department of Computer Science and Eng.
University of Nebraska-Lincoln

Acknowledgements
• Experiments conducted at UNL’s Holland Computing Center
• Geschwender supported by a NSF Graduate Research Fellowship Grant No. 1041000
• NSF Grants No. RI-111795 and RI-1619344
Daniel Geschwender

• 3rd year PhD student at University of Nebraska – Lincoln’s Constraint Systems Laboratory

• Studying high level relational consistencies and automated techniques for determining when to apply them

• Always ready to play a board game!
Claim: Cluster-level portfolio

We advocate the use of an algorithm portfolio for enforcing minimality on the clusters of a tree decomposition during lookahead in a backtrack search for solving CSPs
Outline

• Background
  – Minimality: property and algorithms (ALLSOL, PERTUPLE)
  – Minimality in a tree decomposition

• Processing clusters: FILTERCLUSTERS
  – GAC interleave
  – Cluster-level portfolio
  – Cluster-processing timeout

• Training the classifier
• Experiments
• Conclusion
Background: Minimality

- Global consistency property
- Every tuple in a relation can be extended to a full solution over the \( m \) relations

\[ \forall \text{tuple} \]

\[ \forall \text{relation} \]

\[ \forall m-1 \text{relations} \]
Background: **ALLSOL/PERTUPLE**

**ALLSOL**

- One search explores the entire search space
- Finds all solutions without storing them, keeps tuples that appear in at least one solution
- Better when there are many ‘almost’ solutions

[Karakashian, PhD 2013]
Background: **ALLSOL/PERTUPLE**

**PERTUPLE** [Karakashian, PhD 2013]

- For each tuple, finds one solution where it appears
- Many searches that stop after the first solution
- Better when many solutions are available
Background: Tree decomposition, minimality

- **Minimality on clusters** [Karakashian+ AAAI 2013]
  - Build a tree decomposition
  - Localize minimality to clusters
  - During search, after a variable instantiation
    - Enforce minimality on clusters
    - Propagate following tree structure

- **FILTERCLUSTERS** implements three improvements
  - GAC interleaving
  - Cluster-level portfolio
  - Cluster-processing timeout
FILTERCLUSURES: GAC interleaves

- It is often beneficial to run a lightweight algorithm (e.g., GAC) prior to running a more costly algorithm.
- We extend this idea and interleave a global GAC run between the processing of clusters.
FILTER CLUSTERS: Cluster-level portfolio

• Performance of ALLSOL and PERTUPLE vary
• Sometimes both algorithms are too costly
• Use algorithm portfolio on the cluster level
  – Different algorithms on different clusters
  – Different algorithms on the same cluster during propagation

‘AllSol’
‘PerTuple’
‘Neither’
FILTERCLUSERS: Cluster timeout

- Limits the time for processing a single cluster
- Allows recovery from a poor classification
- When interrupted, partial results of
  - PERTUPLE yield useful filtering
  - ALLSOL are useless
Classifier Training: Data

• 9362 individual clusters taken from 175 benchmarks

• For each cluster instance $i$, collected
  – The values of 73 classification features
  – The runtime of ALLSOL: $allSol(i)$
  – The runtime of PERTUPLE: $perTuple(i)$
Classifier Training: Labels

Runtime of All Instances

- **AllSol**
- **PerTuple**
- **Neither**

Start

- No: \( allSol(i) > 10 \text{ min} \) & \( perTuple(i) > 10 \text{ min} \)
- Yes

- No: \( allSol(i) > perTuple(i) \)
- Yes

- ‘AllSol’
- ‘PerTuple’
- ‘Neither’
Classifier Training: Weights

- Weight of a training instance $i$, $weight(i)$

$$weight(i) = \begin{cases} 
 w(\text{allSol}(i), \text{perTuple}(i)) & \text{label}(i) = \text{AllSol' || PerTuple'} \\
 20 & \text{label}(i) = \text{Neither'}
\end{cases}$$

$$w(a, p) = \left\lvert \log_{10} \left( \frac{a}{p} \right) \right\rvert \cdot \left\lvert \log_{10} (|a - p| + 0.01) \right\rvert$$

- Designed to emphasize instance with both a
  - large proportional difference $\frac{a}{p}$
  - large absolute difference $|a - p|$
Classifier Training: Features

• CSP parameters
  – #variables, #constraints, #values, #tuples
  – Constraint arity, constraint tightness
  – Relational linkage

• Graph parameters: on dual, primal, and incidence graph
  – Density
  – Degree
  – Eccentricity
  – Clustering coefficient

• Using several descriptive statistics
  – min, max, mean, coefficient of variation, entropy
Classifier Training: Decision tree

• We built a decision tree classifier using the J48 algorithm from the Weka machine learning software.

• Decision tree selected for:
  – Simplicity
  – Fast evaluation time
  – Only requires collection a subset of the features
Experiments: Set up

• Used 1055 instances from 42 benchmarks
• Backtrack search, dynamic $\text{dom/deg}$ ordering
• Intel Xeon E5-2650 v3 2.30GHz processors with 12 GB memory
• 2 hours total time out per instance
• Compared GAC and six strategies (variants of FILTERCLUSERS)
Experiments: Tested strategies

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>classifier</th>
<th>interleave GAC</th>
<th>timeout</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALLSol</td>
<td>Always select ‘AllSol’</td>
<td>false</td>
<td>∞</td>
</tr>
<tr>
<td>PERTUPLE</td>
<td>Always select ‘PerTuple’</td>
<td>false</td>
<td>∞</td>
</tr>
<tr>
<td>ALLSol+</td>
<td>Always select ‘AllSol’</td>
<td>true</td>
<td>1 (s)</td>
</tr>
<tr>
<td>PERTUPLE+</td>
<td>Always select ‘PerTuple’</td>
<td>true</td>
<td>1 (s)</td>
</tr>
<tr>
<td>RANDOM</td>
<td>Randomly select ‘AllSol’, ‘PerTuple’, or ‘Neither’</td>
<td>true</td>
<td>1 (s)</td>
</tr>
<tr>
<td>DEC_TREE</td>
<td>Decision tree selects ‘AllSol’, ‘PerTuple’, or ‘Neither’</td>
<td>true</td>
<td>1 (s)</td>
</tr>
</tbody>
</table>
## Experiments: Results

<table>
<thead>
<tr>
<th>Instances Completed</th>
<th>GAC</th>
<th>ALLSOL</th>
<th>PERTUPLE</th>
<th>ALLSOL+</th>
<th>PERTUPLE+</th>
<th>RANDOM</th>
<th>DEC TREE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>550</td>
<td>472</td>
<td>567</td>
<td>514</td>
<td>633</td>
<td>643</td>
<td>685</td>
</tr>
<tr>
<td>Average Time (s)</td>
<td>2,471</td>
<td>3,075</td>
<td>2,081</td>
<td>2,789</td>
<td>1,622</td>
<td>1,427</td>
<td>1,121</td>
</tr>
</tbody>
</table>
Conclusions

- A cluster-level portfolio, during lookahead
  - Is not only feasible, but also highly competitive
- Enforcing a timeout on consistency algorithms
  - Prevents getting stuck on one part of the problem
  - Does not affect soundness
- Future work
  - Dynamically determine timeout based on the anticipated amount of filtering
  - Heuristics for ordering the clusters
Thank you

Questions?
Algorithm 1: `FILTER_CLUSTERS(clusterOrder, classifier, interleaveGAC, timeout)`

**Input:** `clusterOrder, classifier, interleaveGAC, timeout`

**Output:** Entire problem is GAC with potentially minimal clusters

1. `didFiltering ← true`
2. `passDidFiltering ← true`
3. `consistent ← true`
4. `(consistent, didFiltering) ← GAC()`
5. **if** `consistent = false **then** return `false`
6. **while** `passDidFiltering` **do**
   7. `passDidFiltering ← false`
   8. **foreach** `cluster ∈ clusterOrder` **do**
      9. `algo ← CLASSIFY(cluster, classifier)`
      10. **if** `algo = ‘AllSol’` **then**
          11. `(consistent, didFiltering) ← ALLSOL(cluster, timeout)`
      12. **else if** `algo = ‘PerTuple’` **then**
          13. `(consistent, didFiltering) ← PERTUPLE(cluster, timeout)`
      14. **else** `didFiltering ← false`
      15. **if** `consistent = false` **then** return `false`
      16. **if** `didFiltering` **then** `passDidFiltering ← true`
6. **if** `interleaveGAC` **and** `didFiltering` **then**
   17. `(consistent, didFiltering) ← GAC()`
   18. **if** `consistent = false` **then** return `false`
20. `clusterOrder ← REVERSE(clusterOrder)`
21. **if** `interleaveGAC = false` **then**
   22. `(consistent, didFiltering) ← GAC()`
   23. **if** `consistent = false` **then** return `false`
24. return `true`
## Data Summary

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>GAC</th>
<th>AllSol</th>
<th>PerTuple</th>
<th>AllSol+</th>
<th>PerTuple+</th>
<th>Random</th>
<th>DecTree</th>
</tr>
</thead>
<tbody>
<tr>
<td>#Completed</td>
<td>770/1055</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average CPU time</td>
<td>2,471.6</td>
<td>3,075.3</td>
<td>2,081.9</td>
<td>2,789.4</td>
<td>1,622.7</td>
<td>1,427.4</td>
<td>1,121.3</td>
</tr>
<tr>
<td>Sum of CPU time</td>
<td>1,900,653.4</td>
<td>2,364,878.9</td>
<td>2,081.9</td>
<td>2,145,062.1</td>
<td>1,247,840.7</td>
<td>1,097,633.8</td>
<td>862,259.9</td>
</tr>
</tbody>
</table>

### Hybrid solvers are best

### No clear winner

### Basic solvers are best
**Classifier Training: Evaluation**

- Using 10-fold cross validation
- Using both weighted and un-weighted instances

<table>
<thead>
<tr>
<th></th>
<th>weighted</th>
<th>unweighted</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Accuracy</strong></td>
<td>90.8%</td>
<td>80.1%</td>
</tr>
<tr>
<td><strong>F-Measure</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>‘AllSol’</td>
<td>0.50</td>
<td>0.40</td>
</tr>
<tr>
<td>‘PerTuple’</td>
<td>0.89</td>
<td>0.85</td>
</tr>
<tr>
<td>‘Neither’</td>
<td>0.93</td>
<td>0.93</td>
</tr>
</tbody>
</table>
Repeat until quiescence

For cluster $C$ in $LIST$

Classify $C$

Process $C$ within time limit

Enforce GAC globally

Reverse $LIST$
Experiments: Tested strategies (2)

Runtime of All Instances

1 second cutoff per cluster
Experiments: Results (2)

Instance Completions by Runtime

Completed Instances vs. Runtime (sec)

- DECTree
- RANDOM
- PERTuple+
- GAC
- PERTuple
- ALLSol+
- ALLSol
Background: Tree decomposition, minimality

- Build a tree decomposition
- Localize the enforcement of minimality to the clusters
- Process clusters in \textsc{MaxClique}s order back and forth to quiescence