Adaptive Neighborhood Inverse Consistence as Lookahead for Non-Binary CSPs

Robert J. Woodward  
*University of Nebraska-Lincoln,* rwoodwar@cse.unl.edu

Shant K. Karakashian  
*University of Nebraska-Lincoln,* shantk@cse.unl.edu

Berthe Y. Choueiry  
*University of Nebraska-Lincoln,* choueiry@cse.unl.edu

Christian Bessiere  
*University of Montpellier, France,* bessiere@lirmm.fr

Follow this and additional works at: https://digitalcommons.unl.edu/cseconfwork

Part of the Computer Sciences Commons

https://digitalcommons.unl.edu/cseconfwork/219

This Article is brought to you for free and open access by the Computer Science and Engineering, Department of at DigitalCommons@University of Nebraska - Lincoln. It has been accepted for inclusion in CSE Conference and Workshop Papers by an authorized administrator of DigitalCommons@University of Nebraska - Lincoln.
Contributions
1. The property Relational Neighborhood Inverse Consistency (RNIC)
2. Characterization of RNIC in relation to previously known properties
3. An efficient algorithm for enforcing RNIC, bounded by degree of the dual graph
4. Three reformulations of the dual graph to address topological limitations of the dual graph
5. An adaptive, automatic selection policy for choosing the appropriate dual graph
6. Empirical evidence on difficult CSP benchmarks

Definition
A Constraint Satisfaction Problem (CSP) is a combinatorial decision problem defined by a set of variables \{A,B,C,...\}, a set of domain values for these variables, and a set of constraints \{R_1,R_2,R_3,...\} restricting the allowable combinations of values for variables.

The task is to find a solution (i.e., an assignment of a value to each variable satisfying all constraints), or to find all such solutions.

Local Consistency
Local consistency is at the heart of Constraint Processing. It guarantees that all values (or tuples) participate in at least one solution in a given combination of variables (or constraints).

Neighborhood Inverse Consistency (NIC) ensures that every value in the domain of a variable can be extended to a solution in the subproblem induced by the variable and its neighborhood [1].

Relational Neighborhood Inverse Consistency (RNIC) ensures that every tuple \( \tau_i \) in every relation \( R_i \) can be extended to a tuple \( \tau_j \) in every relation \( R_j \in \varphi \setminus \{ R_i \} \) such that all those tuples form a consistent solution to the relations in \( \varphi \) [3].

- Number of combinations = \( O(e^m) = e \)
- Size of each combination = \( m \)
- Twelve combinations for \( R^{(*)}3C \)

\[ \begin{align*}
1. \{ R_1, R_2, R_3 \} & \quad 7. \{ R_2, R_4, R_6 \} \\
2. \{ R_1, R_2, R_4 \} & \quad 8. \{ R_2, R_6 \} \\
3. \{ R_1, R_3, R_4 \} & \quad 9. \{ R_3, R_4, R_6 \} \\
4. \{ R_1, R_3, R_5 \} & \quad 10. \{ R_3, R_4, R_5 \} \\
5. \{ R_1, R_3, R_6 \} & \quad 11. \{ R_3, R_5, R_6 \} \\
6. \{ R_2, R_3, R_4 \} & \quad 12. \{ R_4, R_5, R_6 \}
\end{align*} \]

- Number of subproblems=number of constraints=\( e \)
- Size of subproblems varies, \(|\text{Neigh}(R_i)|+1\)
- Six induced subproblems
  - \( \text{Neigh}(R_i) = \{ R_2, R_3 \} \)
  - \( \text{Neigh}(R_2) = \{ R_1, R_4 \} \)
  - \( \text{Neigh}(R_3) = \{ R_1, R_4, R_5, R_6 \} \)
  - \( \text{Neigh}(R_4) = \{ R_3, R_5, R_6 \} \)
  - \( \text{Neigh}(R_5) = \{ R_3, R_4, R_6 \} \)
  - \( \text{Neigh}(R_6) = \{ R_3, R_4, R_5 \} \)
A queue $Q$ of relations to update
- For each relation $R$, a queue $Q_t(R)$ whose supports must be verified
- Algorithm iterates over every $R$ in $Q$ and applies SEARCHSUPPORT to every $\tau$ in $Q_t(R)$
- SEARCHSUPPORT runs over Neigh($R$)

**Characterizing RNIC**

At the Domains Level

\[
\begin{array}{c}
\text{GAC} \\
\text{SGAC}
\end{array}
\]

At the Relations Level

\[
\begin{array}{c}
R(*,2)C+DF \rightarrow \text{RNIC+DF} \\
R(*,3)C \rightarrow \text{RNIC} \\
wR(*,4)C \rightarrow R(*,4)C \\
wR(*,2)C \rightarrow wR(*,2)C
\end{array}
\]

Algorithm for Enforcing RNIC

**Propagation Algorithm**

- A queue $Q$ of relations to update
- For each relation $R$, a queue of tuples $Q_t(R)$ whose supports must be verified
- Algorithm iterates over every $R$ in $Q$ and applies SEARCHSUPPORT to every $\tau$ in $Q_t(R)$
- SEARCHSUPPORT runs over Neigh($R$)

**Implementation**

Index-Tree to quickly check the consistency of two tuples [3].

Given $R_1$'s tuple:

\[
\begin{array}{c}
A & B & X & D \\
0 & 1 & 0 & 1
\end{array}
\]

find its support in $R_2$

Index-Tree($R_2\{A,B,D\}$)

\[
\begin{array}{c}
\text{Root} \\
0 \\
1 & A \text{ B} \\
1 & B & D
\end{array}
\]

\[
\begin{array}{c}
\tau_1 \\
\tau_2 \text{ 1} \\
\tau_3 \text{ 1} \\
\tau_4 \text{ 1}
\end{array}
\]

Dynamically detect dangles, applying directional arc consistency to quickly detect inconsistency.

$R_2,R_3$ are dangles in the subproblem for $R_1$, induced by Neigh($R_1$)$\cup$($R_1$)

**Complexity**

- **Time:** $O(t^{\delta+1}e\delta)$
  - Delete at most $O(te)$ tuples, enqueueing $O(\delta)$ relations
  - For each tuple, SEARCHSUPPORT executes search on a problem with $\delta$ variables of domain size $t$
- **Space:** $O(ket\delta)$
  - Storing $O(et\delta)$ supports, $O(ket\delta)$ Index-Trees
  - $d$ = maximum domain size
  - $k$ = maximum constraint arity
  - $e$ = number of relations
  - $\delta$ = degree of the dual graph
  - $t$ = maximum number of tuples
Reformulating the Dual Graph

Removing Redundant Edges [2]
- Dense dual graphs → Neighborhoods are large → Cost of our algorithm increases
- Redundancy removal reduces cost

Triangulating the Dual Graph
- In cycles of length ≥ 4, propagation is poor, RNIC=R(*,3)C
- Triangulation boosts propagation

Triangulating a minimal dual graph
- The two operations do not ‘clash’
- The solution set of the CSP is the same in all three reformulations
- In total, four types of dual graphs

Selection Strategy
- If Density ≥ 15%, remove redundant edges
- If triangulation increases density no more than two fold, triangulate
- Each operation is executed at most once

Empirical Results
Statistical analysis on benchmark problems. Max of 90 minutes per instance, yielding censored data (data with values missing). Consistency properties used as full lookahead.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>CPU</th>
<th>#F</th>
<th>Rank</th>
<th>EquivCPU</th>
<th>#C</th>
<th>EquivCmp</th>
<th>#BT-free</th>
</tr>
</thead>
<tbody>
<tr>
<td>wR(*,2)C</td>
<td>944924</td>
<td>52</td>
<td>3</td>
<td>A</td>
<td>138</td>
<td>B</td>
<td>79</td>
</tr>
<tr>
<td>wR(*,3)C</td>
<td>925004</td>
<td>8</td>
<td>4</td>
<td>B</td>
<td>134</td>
<td>B</td>
<td>92</td>
</tr>
<tr>
<td>wR(*,4)C</td>
<td>1161261</td>
<td>2</td>
<td>5</td>
<td>B</td>
<td>132</td>
<td>B</td>
<td>108</td>
</tr>
<tr>
<td>GAC</td>
<td>1711511</td>
<td>83</td>
<td>7</td>
<td>C</td>
<td>119</td>
<td>C</td>
<td>33</td>
</tr>
<tr>
<td>RNIC</td>
<td>6161391</td>
<td>19</td>
<td>8</td>
<td>C</td>
<td>100</td>
<td>C</td>
<td>66</td>
</tr>
<tr>
<td>triRNIC</td>
<td>3017169</td>
<td>9</td>
<td>9</td>
<td>C</td>
<td>84</td>
<td>C</td>
<td>80</td>
</tr>
<tr>
<td>wRNIC</td>
<td>1184844</td>
<td>8</td>
<td>6</td>
<td>B</td>
<td>131</td>
<td>B</td>
<td>84</td>
</tr>
<tr>
<td>wtriRNIC</td>
<td>937904</td>
<td>3</td>
<td>2</td>
<td>B</td>
<td>144</td>
<td>B</td>
<td>129</td>
</tr>
<tr>
<td>selRNIC</td>
<td>751586</td>
<td>17</td>
<td>1</td>
<td>A</td>
<td>159</td>
<td>A</td>
<td>142</td>
</tr>
</tbody>
</table>

CPU: Censored data calculated mean
#F: Number of instances fastest
Rank: Censored data rank based on probability of survival data analysis
EquivCPU: Equivalence classes by CPU

August 10th, 2011
References


Acknowledgments

Robert Woodward was supported by a Goldwater Scholarship & an NSF Graduate Research Fellowship. This research is also supported by NSF Grant No. RI-111795.

Experiments were conducted utilizing the Holland Computing Center of the University of Nebraska.