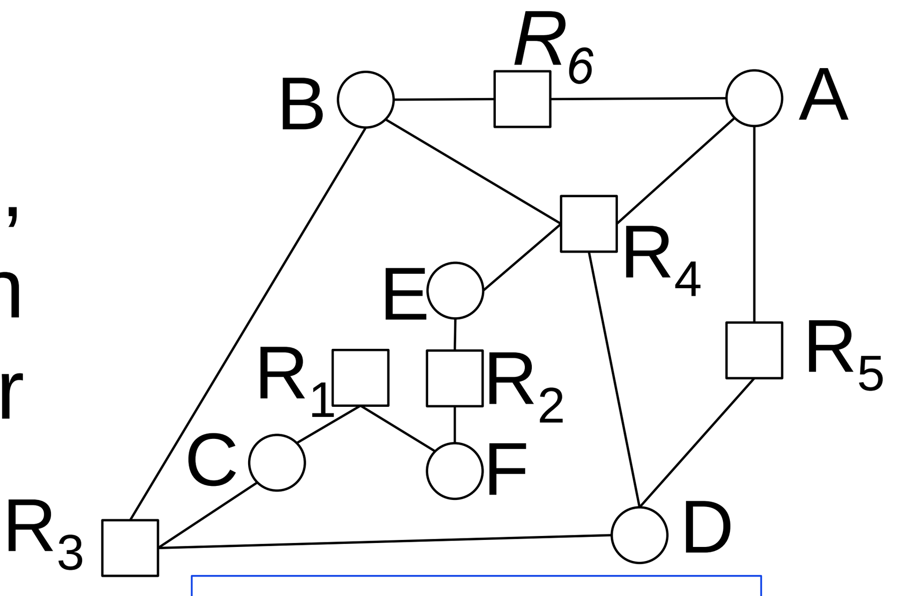
Contributions

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

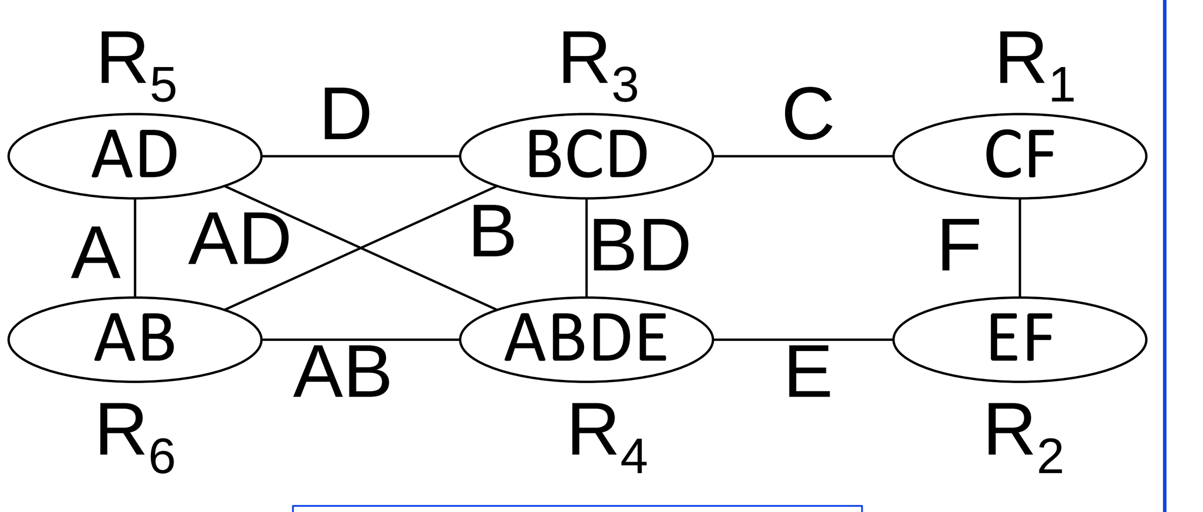
Definition

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** $\{R_1,R_2,R_3,\ldots\}$ 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**.



Hypergraph

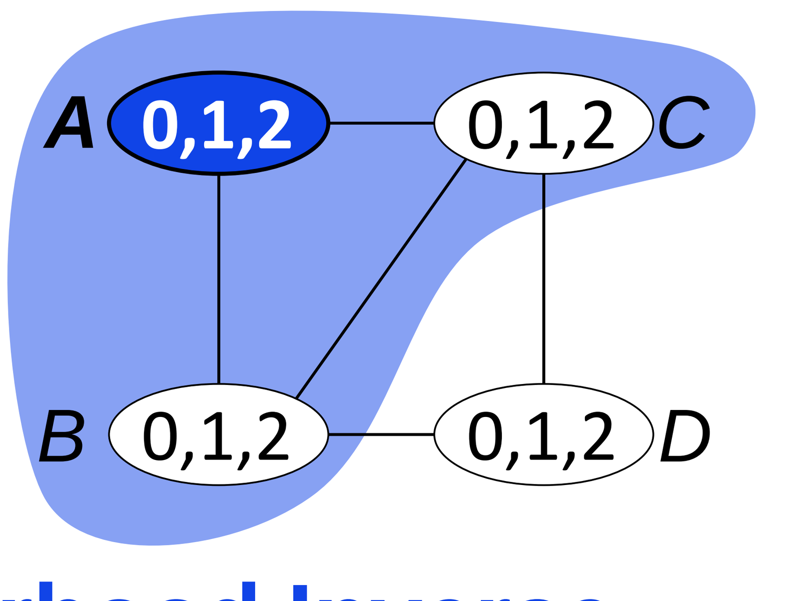


Dual Graph

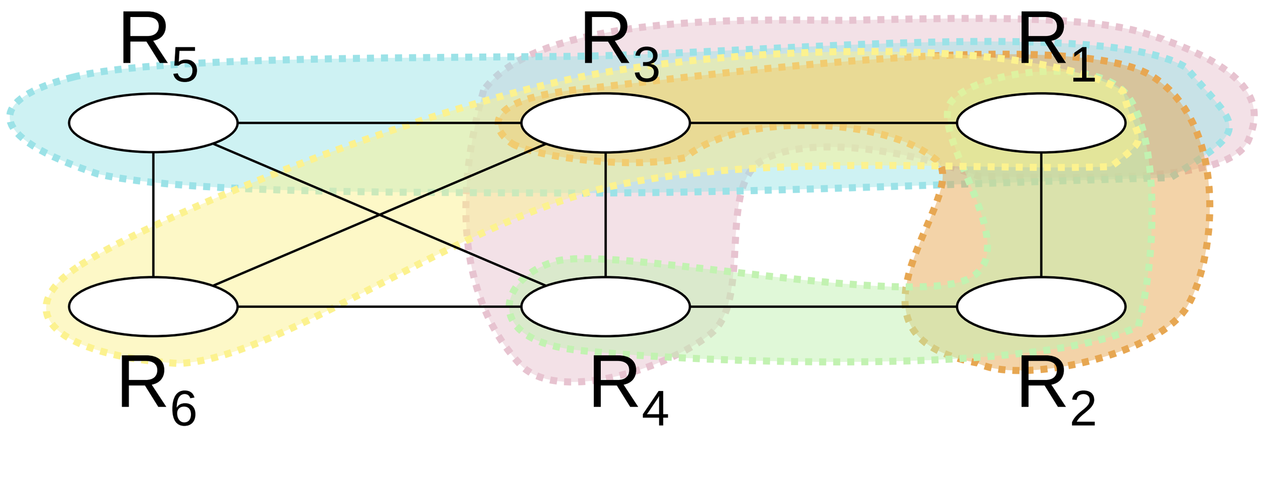
Local Consistency

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].



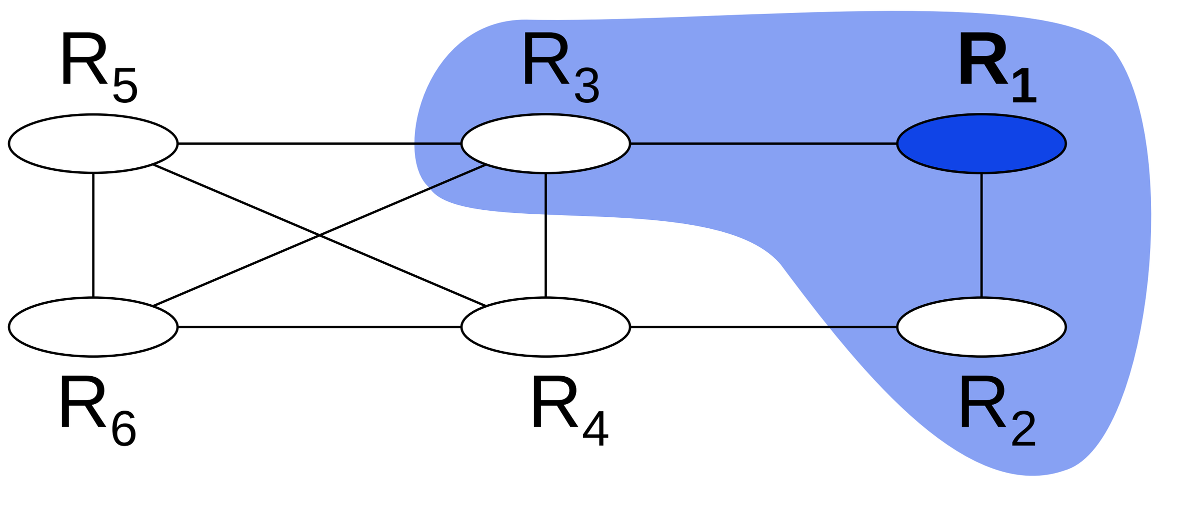
R(*,*m*) ensures that, in every given combination φ of m relations, every tuple τ_i in every relation R_i can be extended to a tuple τ_j in every relation $R_j \in \varphi \backslash \{R_i\}$ such that all those tuples form a consistent solution to the relations in φ [3]**.**



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

1. $\{R_1,R_2,R_3\}$
2. $\{R_1,R_2,R_4\}$
3. $\{R_1,R_3,R_4\}$
4. $\{R_1,R_3,R_5\}$
5. $\{R_1,R_3,R_6\}$
6. $\{R_2,R_3,R_4\}$
7. $\{R_2,R_4,R_5\}$
8. $\{R_2,R_4,R_6\}$
9. $\{R_3,R_4,R_5\}$
10. $\{R_3,R_4,R_6\}$
11. $\{R_3,R_5,R_6\}$
12. $\{R_4,R_5,R_6\}$

Relational Neighborhood Inverse Consistency (RNIC) ensures that every tuple τ_i in every relation R_i can be extended to a tuple τ_j in each $R_j \in$ Neigh(R_i) such that together all those tuples are consistent with all the relations in Neigh(R_i) [4].

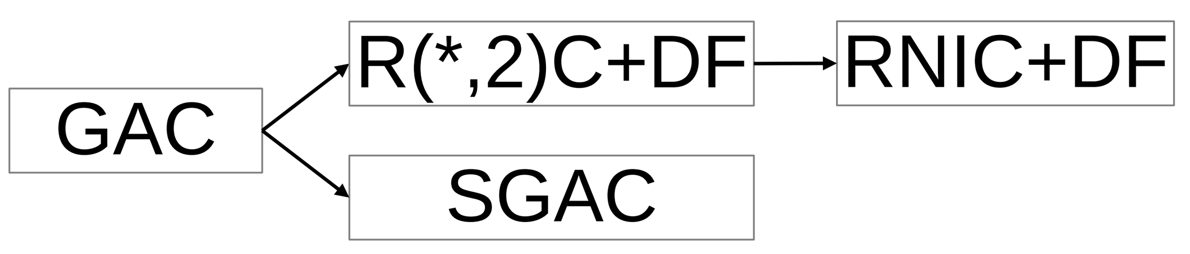


- Number of subproblems=number of constraints=e
- Size of subproblems varies, |Neigh(R_i)|+1
- Six induced subproblems
  - Neigh(R_1) = $\{R_2,R_3\}$
  - Neigh(R_2) = $\{R_1,R_4\}$
  - Neigh(R_3) = $\{R_1,R_4,R_5,R_6\}$
  - Neigh(R_4) = $\{R_2,R_3,R_5,R_6\}$
  - Neigh(R_5) = $\{R_3,R_4,R_6\}$
  - Neigh(R_6) = $\{R_3, R_4,R_5\}$

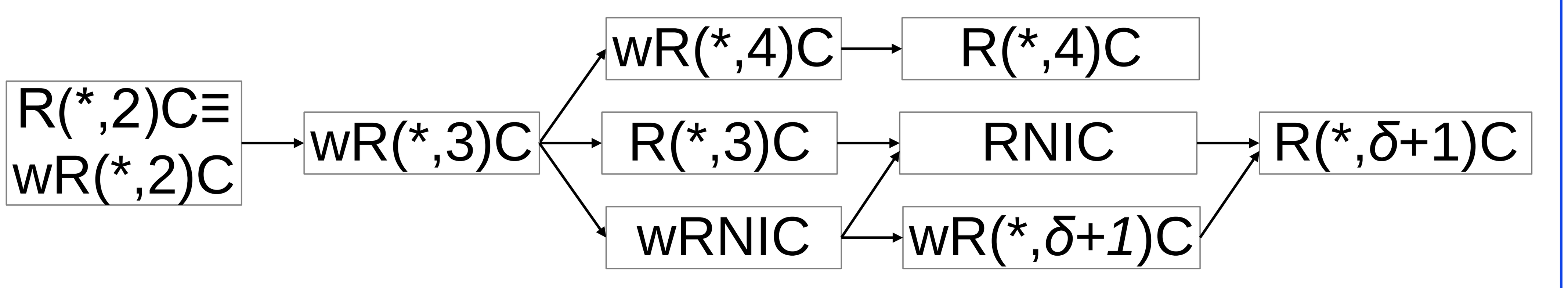
August 10th, 2011

Characterizing RNIC

At the Domains Level



At the Relations Level

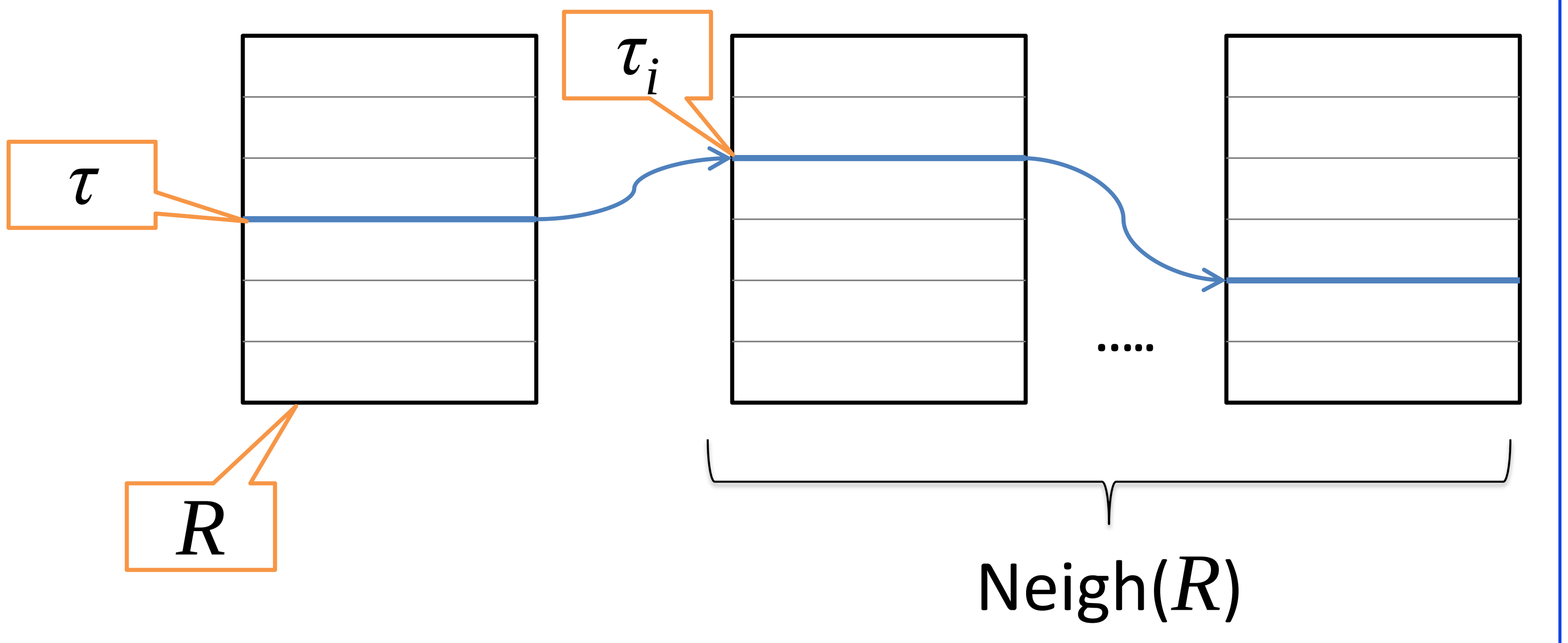


Algorithm for Enforcing RNIC

Propagation Algorithm

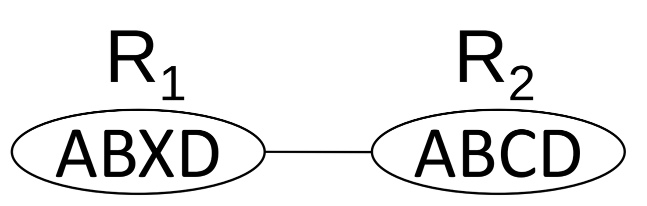
- A queue Q of relations to update
- For each relation R, a queue of tuples $Q_t(R)$ whose supports must be verified
- Algorithm iterates over every R in Q and applies SEARCHSUPPORT to every τ in $Q_t(R)$
- SEARCHSUPPORT runs over Neigh(R)

SEARCHSUPPORT



Implementation

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

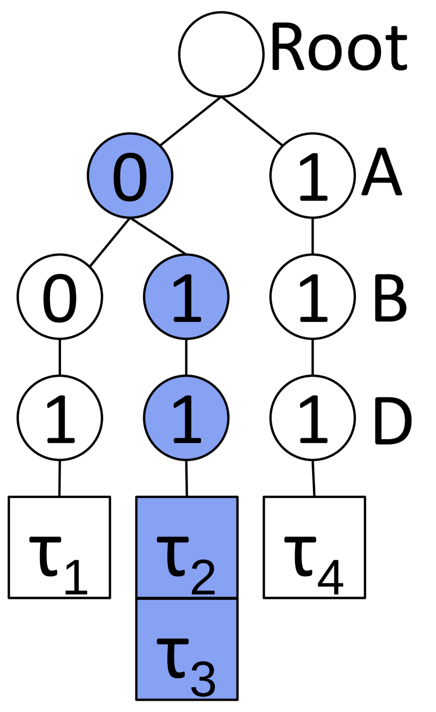


Given R_1's tuple:

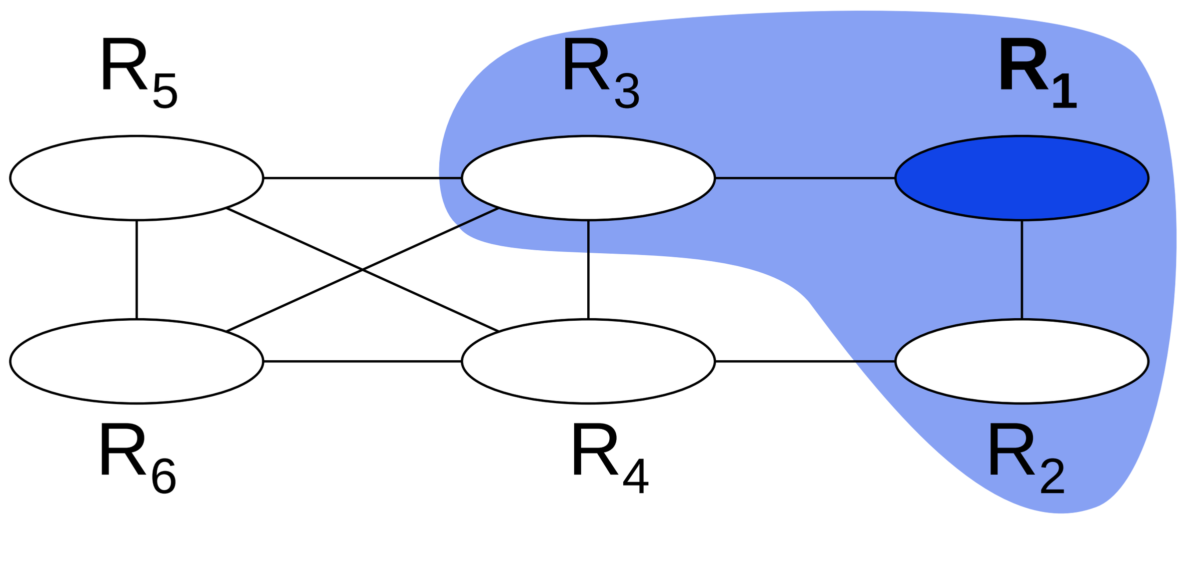
A	B	X	D
0	1	0	1

find its support in R_2

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



	A	B	D
τ_1	0	0	1
τ_2	0	1	1
τ_3	0	1	1
τ_4	1	1	1



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)∪{R_1}

Complexity

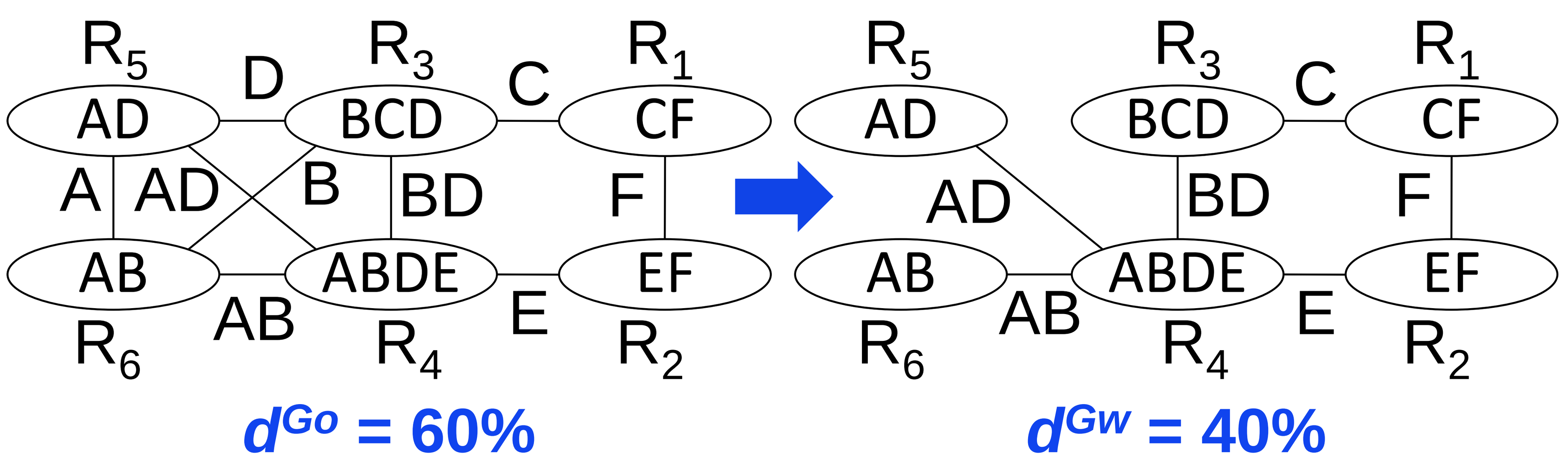
- **Time:** $O(t^{\delta+1}e\delta)$
  - Delete at most $O(te)$ tuples, enqueuing $O(\delta)$ relations
  - For each tuple, SEARCHSUPPORT executes search on a problem with δ variables of domain size t
- **Space:** $O(ket\delta)$
  - Storing $O(et\delta)$ supports, $O(ket\delta)$ Index-Trees

- d = maximum domain size
- k = maximum constraint arity
- e = number of relations
- δ = degree of the dual graph
- t = maximum number of tuples

Reformulating the Dual Graph

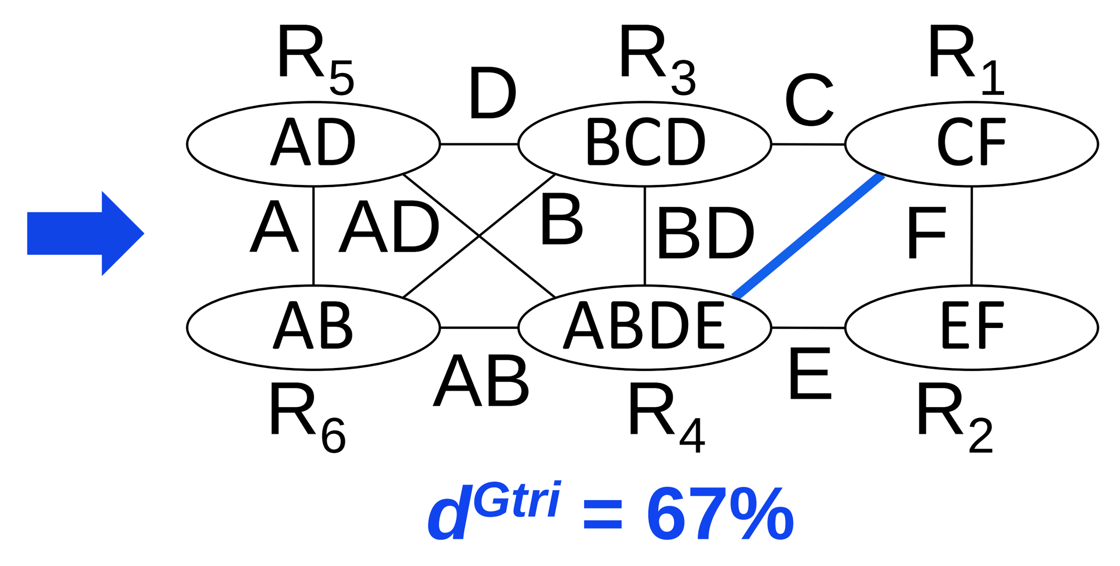
Removing Redundant Edges [2]

- Dense dual graphs → Neighborhoods are large → Cost of our algorithm increases
- Redundancy removal reduces cost



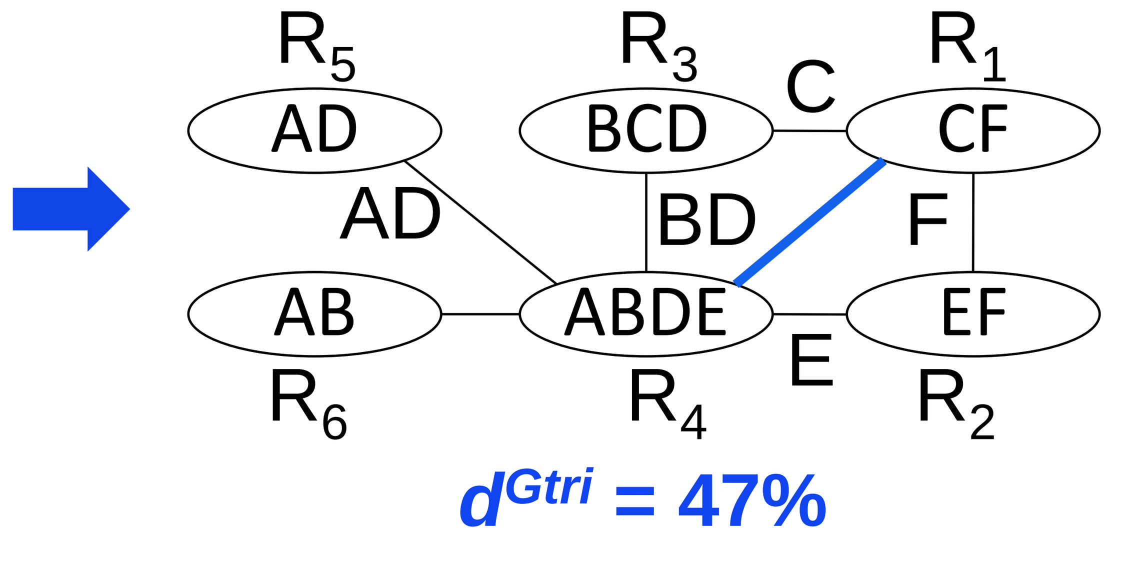
Triangulating the Dual Graph

- In cycles of length ≥ 4, propagation is poor, RNIC≡R(*,3)C
- Triangulation boosts propagation



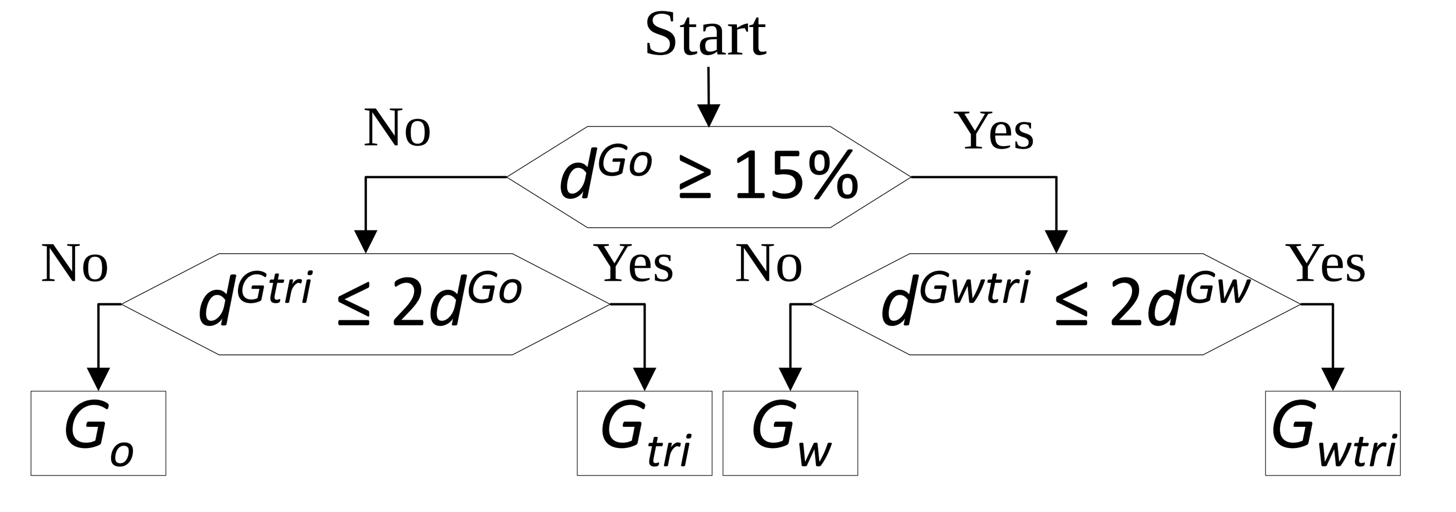
Triangulating a minimal dual graph

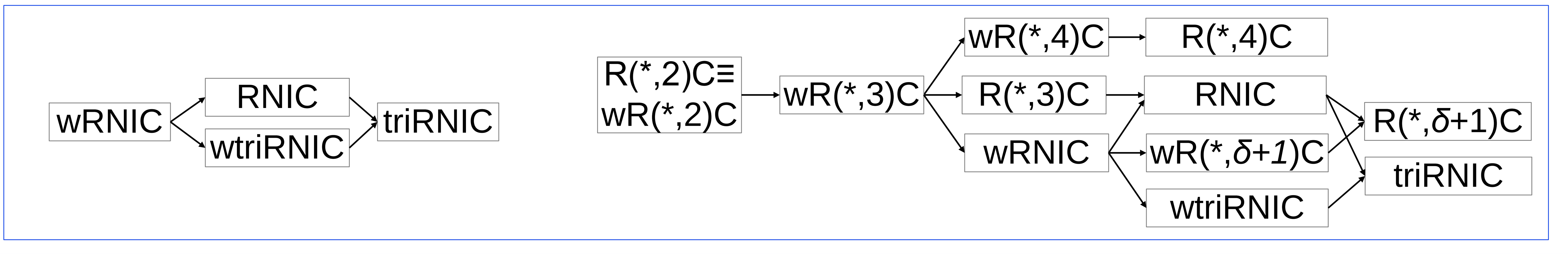
- The two operations do not 'clash'
- The solution set of the CSP is the same in all three reformulations
- In total, four types of dual graphs



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.

- **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
- **#C**: Number of instances completed
- **EquivCmp**: Equivalence classes by completion
- **#BT-free**: Number of instances solved BT-free. Reflects strength of a given consistency, regardless of implementation

Algorithm	CPU	#F	Rank	EquivCPU	#C	EquivCmp	#BT-free
169 instances: aim-100, aim-200, lexVg, modifiedRenault, ssa							
wR(*,2)C	944924	52	3	A	138	B	79
wR(*,3)C	925004	8	4	B	134	B	92
wR(*,4)C	1161261	2	5	B	132	B	108
GAC	1711511	83	7	C	119	C	33
RNIC	6161391	19	8	C	100	C	66
triRNIC	3017169	9	9	C	84	C	80
wRNIC	1184844	8	6	B	131	B	84
wtriRNIC	937904	3	2	B	144	B	129
selRNIC	751586	17	1	A	159	A	142