2005

Neighborhood Interchangeability and Dynamic Bundling for Non-binary CSPs

Anagh Lal
University of Nebraska-Lincoln, alal@cse.unl.edu

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

Eugene C. Freuder
University College Cork, e.freuder@cs.ucc.ie

Neighborhood Interchangeability and Dynamic Bundling for Non-binary CSPs

Anagh Lal and Berthe Y. Choueiry
Constraint Systems Laboratory • University of Nebraska-Lincoln

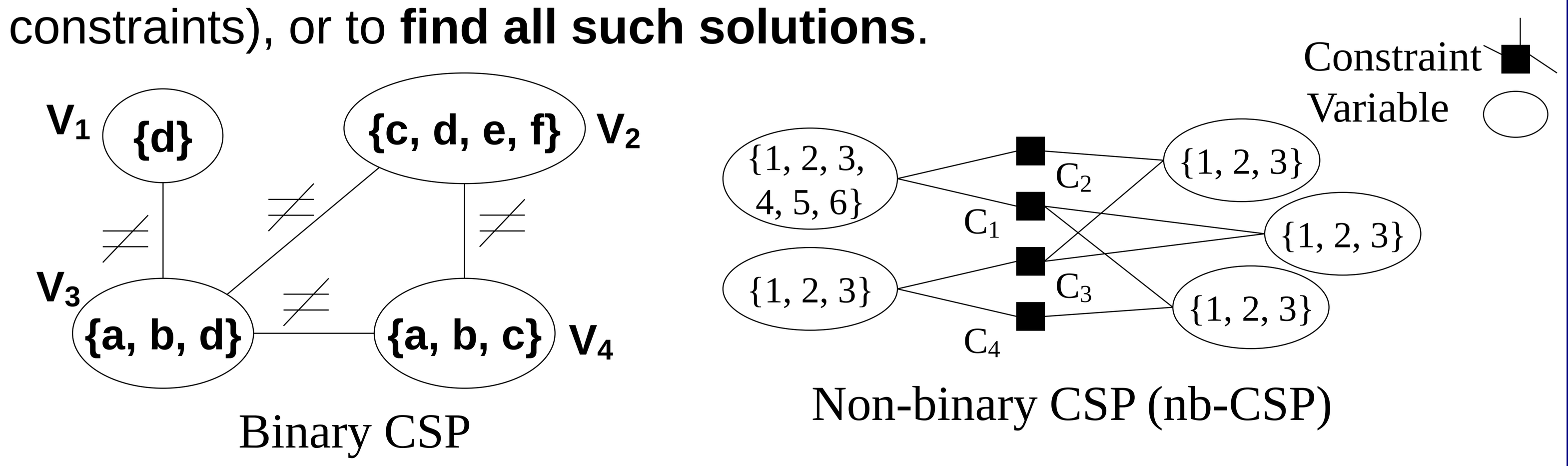
Eugene C. Freuder
Constraint Computation Center • University College Cork

Contributions

1. **Interchangeability**: An algorithm for computing interchangeability in non-binary CSPs.
2. **Dynamic bundling**: Integration of the above with backtrack search for solving non-binary CSPs.
3. **Experiments** demonstrating the benefits of dynamic bundling
   - Finding multiple, robust solutions.
   - Decreasing computational cost of search.

Constraint Satisfaction Problems

A Constraint Satisfaction Problem (CSP) is a combinatorial decision problem defined by a set of **variables**, a set of domain **values** for these variables, and a set of **constraints** restricting the allowable combinations of values for variables, where 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**.

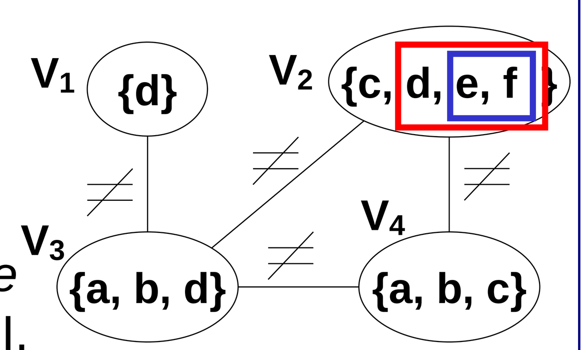


Interchangeability & Bundling

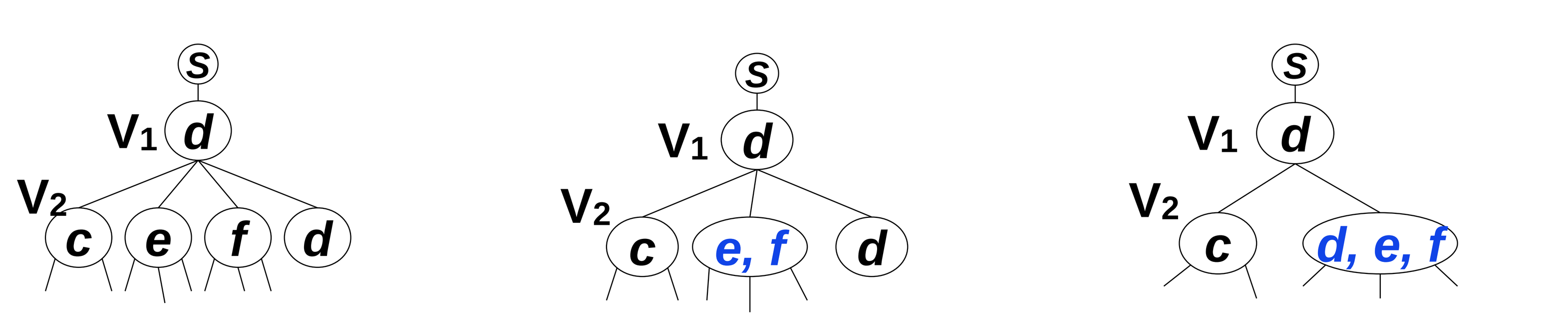
Interchangeability identifies values equivalent in all solutions of a CSP [2].

Full Interchangeability (FI): *d, e,* and *f* can be swapped for V_2 in any solution.

Neighborhood Interchangeability (NI): finds *e* and *f* but misses *d*. It efficiently approximates FI.



Bundling = Search + neighborhood interchangeability [3]
Dynamic bundling = Search + dynamic NI [1]



Search without bundling

Static bundling [3]: Interchangeability sets are computed *before* search

Dynamic bundling [1]: Interchangeability sets are updated *during* search

Interchangeability in non-binary CSPs

We show how to compute NI for non-binary constraints by:

1. Building the non-binary discrimination tree, nb-DT(V, C), a data-structure that determines the NI sets of a variable V given a constraint C defined on V.
2. Intersecting the NI sets from the nb-DTs of a set of constraints, which yields the domain partition of the variable V given the constraints.

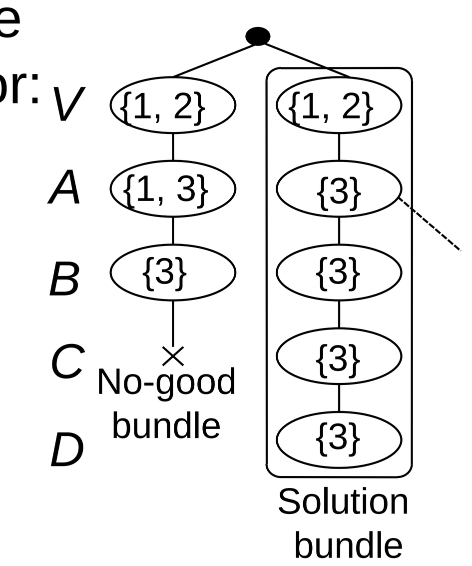
Dynamic Bundling for Non-Binary CSPs

Dynamic bundling (DynBndl) was thought to be an overkill. We show DynBndl is worthwhile for:

- Finding all solutions: theoretically best
- Finding first solutions: empirical evidence

Because DynBndl:

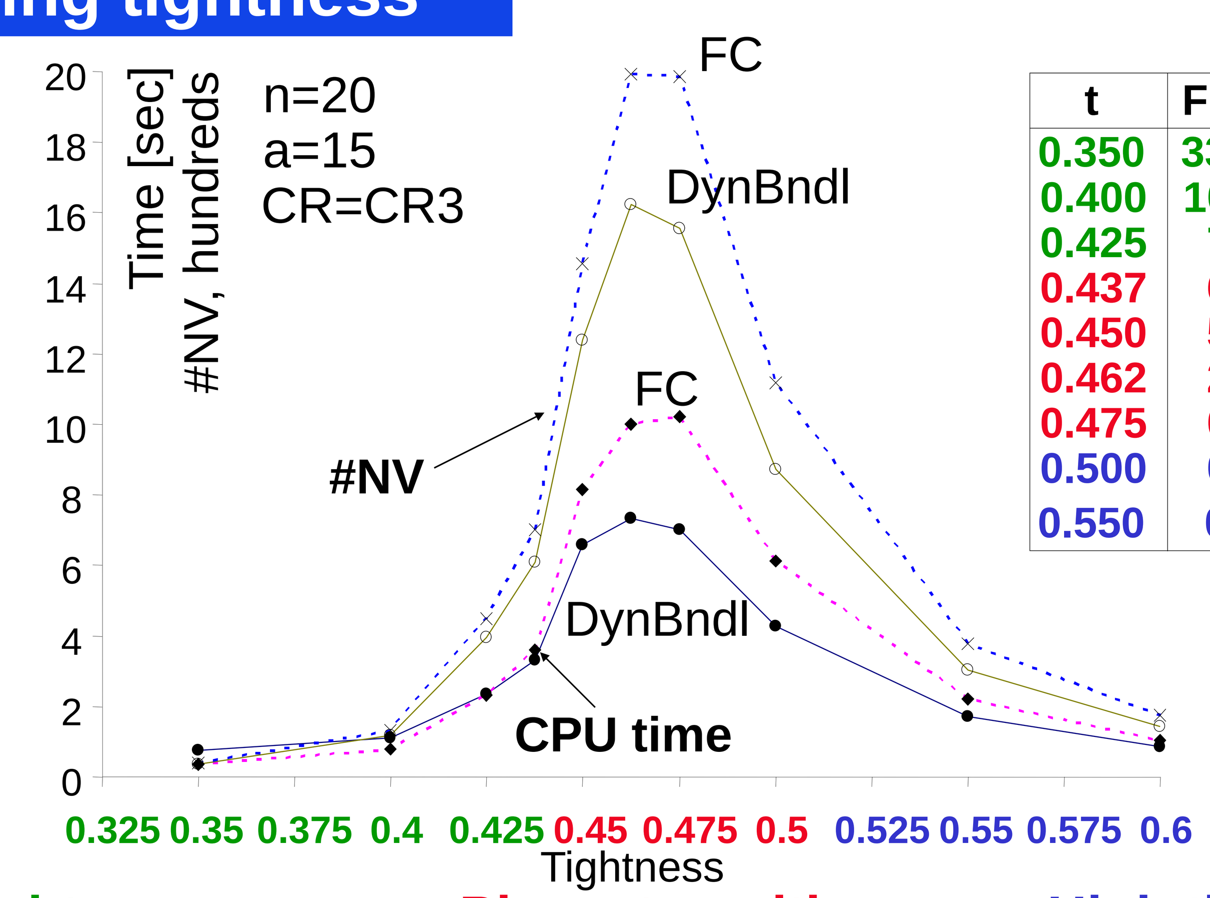
- Bundles solutions
- Bundles no-goods (i.e., bundles of inconsistent partial solutions).



Experiments: FC versus DynBndl

- Tested random CSPs, Model B, 1000 instances per sample
- **Criteria**: FBS (First Bundle Size), CPU time, number of nodes visited (NV), and number of constraint checks.
- **Statistical tools**: ANOVA and t-distribution for confidence intervals.

Varying tightness



t	FBS
0.350	33.44
0.400	10.91
0.425	7.13
0.437	6.38
0.450	5.62
0.462	2.37
0.475	0.66
0.500	0.03
0.550	0.00

Low tightness

- Large FBS: 33 at t = 0.35 (2254 in Dataset #13).
- Small bundling overhead.

Phase transition

- Multiple solutions exist.
- Maximum no-good bundling yields max savings in CPU time, NV, & CC.

High tightness

- Problems mostly unsolvable.
- Minimal bundling overhead.

Increasing domain size

Increasing a, we note in the phase-transition area:

- FBS increases (more chances for symmetry).
- CPU time decreases (better no-good bundling).

The benefits of DynBndl increase with increasing domain size: Use DynBndl in database applications where large domains are typical.

Constraint Ratio	p_2	c_3	c_4
CR1	0.25	3	2
CR2	0.25	6	5
CR3	0.40	3	2
CR4	0.40	6	5

CR	FBS		CPU improvement %	
	a=10	a=15	a=10	a=15
CR1	5.5	11.9	33.3	34.3
CR2	5.0	5.5	28.6	33.0
CR3	3.6	5.0	29.8	31.7
CR4	1.2	1.4	28.4	31.6

Dynamic Bundling in Databases [4]

R1		
A	B	C
1	12	23
1	13	23
1	14	23
2	10	25
3	16	30
3	16	24
4	10	25
5	12	23
5	13	23
5	14	23
6	13	27
6	14	27
7	14	28
7	19	20

R2		
A	B	C
1	12	23
1	13	23
1	14	23
1	15	23
2	10	25
3	17	20
3	18	22
4	10	25
5	12	23
5	13	23
5	14	23
5	15	23
6	13	27
6	14	27
8	14	28

Task: Join query

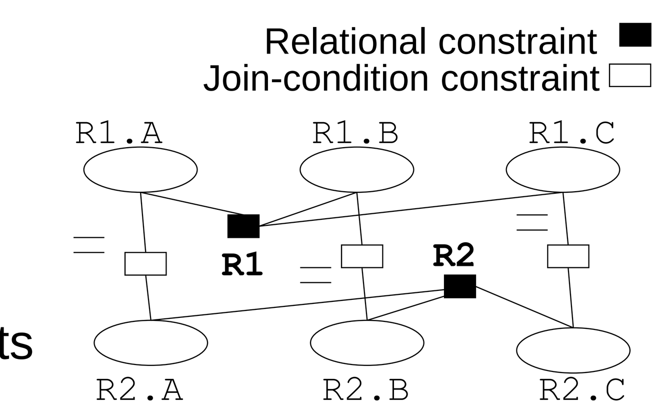
```
SELECT R1.A,R1.B,R1.C
FROM   R1,R2
WHERE  R1.A=R2.A
AND    R1.B=R2.B
AND    R1.C=R2.C
```

R1 ⋈ R2 (Compacted)		
A	B	C
{1, 5}	{12, 13, 14}	{23}
{2, 4}	{10}	{25}
{6}	{13, 14}	{27}

Result: 10 tuples in 3 nested tuples

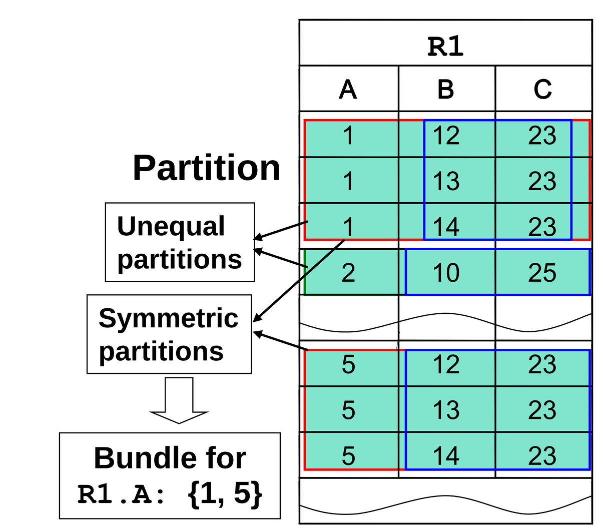
Modeling the join query as a CSP

Attributes → variables
Attribute values → domains
Relations → relational constraints
Join conditions → join-condition constraints



Sorting-based bundling algorithm

- partitions domains in a memory efficient manner.
- fits into the iterator model of databases and produces one bundle at a time.



Sort-merge join algorithm based on dynamic bundling

- Reduces number of tuples compared in the main memory.
- Is memory efficient and produces compacted results, saving
  - I/O for the next operator and
  - disk space (and network bandwidth in distributed databases).

Experiments

- Compaction rate achieved in a real-world problem: 2.26.
- Compaction rate achieved on a random data-set: 1.48 (10'000 tuples; memory size: 4'000 tuples; page size 200 tuples).

Future Research Directions

- Use new join algorithm when materializing join queries.
- Exploit bundled results in data-analysis/data-mining packages.
- Assist in query size estimation
- Improve accuracy of sampling operators

References

1. Choueiry, B.Y., Davis, A.M.: Dynamic bundling: Less Effort for More Solutions. SARA 02.
2. Freuder, E.C.: Eliminating Interchangeable Values in Constraint Satisfaction Problems. AAAI 91.
3. Haselböck, A: Exploiting Interchangeabilities in Constraint Satisfaction Problems. IJCAI 93.
4. Lal, A., Choueiry, B.Y.: Constraint Processing Techniques for Improving Join Computation. CDB 04.

University of Nebraska Lincoln

This research was supported by a Maude Hammond Fling Faculty Research Fellowship, National Science Foundation CAREER Award #0133568, and Science Foundation Ireland Grant 00/PI.I/C075.
Experiments were conducted utilizing the Research Computing Facility of the University of Nebraska-Lincoln.