CSP Solving by Constraint Joining

Shant Karakashian* José Luis Ambite**
Berthe Y. Choueiry*

*Constraint Systems Laboratory
University of Nebraska-Lincoln

**Information Sciences Institute
University of Southern California

Email: shantk@cse.unl.edu

Working Note 2-2008

December 10, 2008

1 Experimental set up

Experiments were conducted on:

- CSPs with 30 variables, domain size 8, density varying from 30 to 70 ($n = 30$, a=8, d=30% to 70%).
- CSPs with 20 variables, domain size 8, density varying from 20 to 70 ($n = 20$, a=8, d=20% to 70%).
- For all instances, tightness is adjusted to place the instance *at the phase transition*.
- For each instance, we ran the below algorithms with the joining algorithm being with sort-merge then hash-join. For each algorithm we display the numerical results as listed below:
  1. Forward Checking (FC): finding one solution (1 sol) and all solutions (All sol).
  2. Bushy join (Bsh): CPU time to complete the join (up), the number of tuples in the result (#Sol).
  3. Left-deep join (LftD): CPU time to complete the join (up), the number of tuples in the result (#Sol).

4. Bushy join with projection (Bsh Prj): CPU time to go up (up), the number of tuples in the partial result (#Sol), the total CPU time to compute the join (up+down), which includes the time to go up and to reconstruct the result.
5. Left-deep join with projection (LftD Prj): CPU time to go up (up), the number of tuples in the partial result (#Sol), the total CPU time to compute the join (up+down), which includes the time to go up and to reconstruct the result.
6. Bushy join with projection and F1 filtering (Bsh Prj F1): The numerical values are as the previous. The filtering F1 operates by keeping up to date the domains of the variables (i.e., columns) and filtering any two relations accordingly before joining them.
7. Left-deep join with projection and F1 filtering (LftD Prj F1): same as previous but for left-deep join.
8. Bushy join with projection and F2 filtering (Bsh Prj F2): The numerical values are as the previous. The filtering F2 operates by filtering all 'future' relations to remove tuples that are inconsistent with the current 'joins', but does not filter the current join.
9. Left-deep join with projection and F2 filtering (LftD Prj F2): same as previous but for left-deep join.
10. Bushy join with projection and F3 filtering (Bsh Prj F3): The numerical values are as the previous. The filtering F3 operates by filtering all 'future' relations *and* the current joins to remove tuples that are inconsistent with the current 'joins', doing only one loop.

In general, filtering is executed by joining the relations using hashing and projecting the resulting join on the respective scopes

- Instances for each problem size were separated in solvable and unsolvable instances, resulting in 4 files as Excel spreadsheets. Each Excel file has two sheets, one giving the results with sort-merge and the second with hash-join.
- When an execution could not finish, it is indicated with '-' in the result table. This situation may occur for any of two reasons: the relation size or the execution time exceeds a given threshold.

2 Observations

We make the following observations.

2.1 General comments

- *Hash-join versus sort-merge:* Hash-join-based algorithms are systematically less good than sort-merge-based algorithms, except when joining

two very large tables. We restrict our comments below on the sort-merge join algorithm.

- To examine the results, we recommend to study the experiments for
  - for n=30 variables, at 10% density.
  - For all other experiments, examine only n=20 variables.
- For all join-based algorithms using projection, the resulting join has to reconstructed after projection. Thus, we distinguish two phases going-up and going-down. The going-up phase uses a greedy by sophisticated heuristic for choosing the next relation to join with. This phase is thus fairly optimized. The going-down phase, however, follows the order chosen by the previous phase and does not exploit the greedy heuristic. For this reason, the reconstruction phase is much slower than the going-up phase. It is far from being optimized and should not been used in the analysis.

2.2 The champion is FC

- For all algorithm instances and for CSP instances, FC is significantly faster then any other algorithm tested.
- Only bushy-type algorithms become competitive with FC for very sparse problems (i.e., density is 10%). Indeed, bushy-type algorithms with projection become comparable to FC. Further, with filtering, they become comparable with FC.
- On unsolvable instances, the performance of FC is not matched, even remotely, by any algorithm.
- As density increases, the performance of all join-based techniques deteriorates orders of magnitude with respect to that of FC. Very quickly, they are not able to terminate, which affects more quickly bushy than left-deep strategies. (Interpretation: The size of the tables in bushy grow prohibitively large and bushy has the challenge of joining many large tables whereas left-deep is dealing with only one large table.)

2.3 Analyzing join-based algorithms

- Bushy exhibits a search-like performance: it does more steps, but is less costly in terms of CPU cycles.
- Left-deep is like computing higher levels of consistency. It does fewer steps, but costs more time.
- Bushy plans tend to improve as we add projection, filtering, and more aggressive filtering.
- The advantage of filtering is not visible for left-deep plans. More aggressive filtering increases the cost. Only project improves the left-deep plans.

2.4 Bushy versus left-deep

- On average, bushy plans demonstrate better performance than left-deep plans. Exceptions exist, but are outliers.
- Bushy plans seems quicker than left-deep plans, but they seem to be more costly in terms of space. So, in terms of time, bushy plans are better than left-deep plans; in terms of space, the inverse seems to hold.
- Bushy plans benefit from filtering better than left-deep plans.

  Increasingly powerful filtering dramatically benefit Bushy while they slow left deep. Interpretation: filtering in bushy is not costly because the number of relations to filter against is smaller than in left-deep, to the point that left-deep performance is hindered by the filtering.

  Lesson: if you are doing bushy, you should definitely filter as aggressively as you can, regardless of the density. If you are using left-deep, filtering is useful for low density problems. As density grows, filtering hinders performance.
- On unsolvable instances: Left-deep plans discover unsolvability earlier than bushy plans. (Interpretation: left-plans achieve higher consistency earlier than Bushy plans.) Best case of left-deep is better than best case of bushy.