PrefixCCFWC: performance comparison

Victor Lecomte

September 10, 2015

Abstract

For the PrefixCC problem, whose statement is described in the main technical report, we implemented several approaches with different complexities and pruning levels. In order to decide which ones to keep, we performed several benchmarks, which we present here with results, comments and conclusions.

Contents

1 Approaches

Four approaches were considered for the performance testing:

— *Multiple GCCFWCs*: our baseline, which consists in adding as many regular `GCCFWC` constraints as there are prefixes constrained.

This approach is expected to be slow because of the redundancy of the reversible structures and simply the large number of constraints to be queued at each node.

— *PrefixCC Segments*: the main approach, described in the accompanying technical report. First performs some deduction and filtering on the bounds and then prunes according to those but in an efficient non-redundant way, by working segment-by-segment instead of prefix-by-prefix.

This approach is expected to stay quick even for a large number of variables.

— *PrefixCC Fenwick*: another efficient approach, that makes the same types of deductions as the previous ones but also takes into account the current state of the variables. According to that, it keeps with a Fenwick tree the dynamic list of the best-known bounds for each prefix and prunes values every time the lower and upper bounds become equal in an interval. In that way it is, for each value, arc-consistent on the subset of bounds relative to the value.

This approach is expected to have a better pruning.

— *PrefixCC Embedded GCCFWCs*: same as the segments one except that once the bounds are deducted and filtered it keeps a separate count for each constraint which is a bit more redundant (but not as much as the baseline).

This approach is expected to be quick for small numbers of variables.

2 Tests performed

Two kinds of tests were performed:

— test cases where the lower bounds and upper bounds were very weak so that pruning matters only very little and raw propagation speed is shown;

— harder test cases where solutions are harder to find and backtracks take up most of the time, in order to isolate the solutions with the best pruning.

This section describes the methodology for generating those test cases. We will first touch the common aspects of the benchmarks (2.1) and then explain the differences between them (2.2 and 2.3).

2.1 Common aspects

For every test, we picked a certain number of variables `nVariables`. We used 50, 100 and 200 variables to make our tests. Each variable was given a random domain among three possible values.

We then added `nVariables` lower bounds and `nVariables` upper bounds by choosing a random prefix and a random value to constrain. The actual bounds depended on the benchmark. Given that there are three values, this makes for one bound for each value every three prefixes in average. Such a quantity of bounds was necessary to pick up with clarity the complexities of the approaches relative to the number of bounds.

After that we started a binary static search with the table of variables shuffled in the same way for every approach. The shuffling avoided stumbling upon a special case with a better complexity for some approaches. The search stopped after finding a set number of solutions.

For every test, the approaches were run in a random order to remove undesirable biases due to JVM warmup.

2.2 Weak constraints

In this benchmark, in order to measure the raw propagation speed of the approaches, we made the solutions very easy to find. To do that, we gave the bounds very conservative values.

We simply assigned the lower bounds to 1/7 of the size of the prefix and the upper bounds to 6/7 of the size of the prefix. Since there are only three values those bounds are nearly always respected and we could compare the approaches as they went through similar search trees.

2.3 Harder cases

In this benchmark, we wanted to give cases where finding solutions is more tricky so that the solutions with the best pruning have an advantage. But there still had to be some solutions so that we can time the approaches correctly.

To do that, we took a random solution with respect to the domains of the variables and we based the bounds on the effective occurrences of the values in that solution. We allowed lower bounds

to vary randomly between 75% and 100% of the number of occurrences in the model solution, and upper bounds to vary randomly between 100% and 125% of the number of occurrences in the model solution. That way we had strong bounds but still at least one solution (and probably many more).

We also tried narrower intervals for the lower bounds and upper bounds but all solutions except the one with the best pruning ended up timing out, which was comforting but did not give interesting data.

3 Commented results

4 Conclusion