Preprocessing Boolean Formulae for BDDs in a Probabilistic Context

Theofrastos Mantadelis[1], Ricardo Rocha[2], Angelika Kimmig[1] and Gerda Janssens[1]

[1] Departement Computerwetenschappen, K.U. Leuven
Celestijnenlaan 200A - bus 2402, B-3001 Heverlee, Belgium
{Theofrastos.Mantadelis,Angelika.Kimmig,Gerda.Janssens}@cs.kuleuven.be
[2] CRACS & INESC-Porto LA, Faculty of Sciences, University of Porto
Rua do Campo Alegre, 1021/1055, 4169-007 Porto, Portugal
ricroc@dcc.fc.up.pt

1 Complexity of Preprocessing

This section analyzes the worst case complexity of our preprocessing algorithms. We use N to denote the number of proofs or conjunctions and M to denote the number of probabilistic facts or Boolean variables. By our experience, $N >> M$ usually holds for typical ProbLog programs.

1.1 Naive Method

The naive approach writes N conjunctions, each with a maximum of M Boolean variables, and one disjunction of length N. Thus, its worst case complexity is $O(N \cdot M + N) = O(N \cdot M)$.

1.2 Decomposition Method

The decomposition method repeatedly scans a DNF to assign conjunctions to subformulae which are then in turn decomposed further. If decomposing on the first variable of the current formula, the decomposition variable is determined in constant time. In the worst case, the N conjunctions of the initial DNF are of length M each. Thus, the first decomposition step needs to scan the entire formula, corresponding to a cost of $N \cdot M$. This then results in three recursive calls for each part, respectively, with N_1, N_2 and N_3 conjunctions subject to the constraint $N = N_1 + N_2 + N_3$, with maximal length $M - 1$ for each conjunction. Decomposing the i^{th} part in turn requires scanning with cost $N_i \cdot (M - 1)$, i.e., the total cost of the second level is $N_1 \cdot (M - 1) + N_2 \cdot (M - 1) + N_3 \cdot (M - 1) = (N_1 + N_2 + N_3) \cdot (M - 1) = N \cdot (M - 1)$. Again, it then results in three recursive calls for each part, with N_{i1}, N_{i2} and N_{i3} conjunctions respectively, with maximal length $M - 2$ for each conjunction, and subject to the constraint $N_i = N_{i1} + N_{i2} + N_{i3}$. In general, due to the fixed number of conjunctions to be assigned to subformulae, the total cost of the j-th decomposition level is

$N \cdot (M+1-j)$. As the number of such levels is bound by the number of variables M that can be used for decomposition, the overall cost is $\sum_{j=1}^{M} N \cdot (M+1-j)$ and thus $O(N \cdot M^2)$.

1.3 Recursive Node Merging

Recursive node merging complexity strongly depends on the amount of prefix sharing in the initial trie. In the following analysis, we will consider our new approach using the depth breadth trie.

In the first iteration of the algorithm, it is possible that no conjunctions (depth reductions) are found, in which case the size of the trie is not reduced. However, in later iterations, depth reduction will always reduce the depth of the trie by at least one. Likewise, breadth reduction will always reduce the number of leaves by at least one, i.e., after $O(min(M, N))$ iterations, either the depth or the breadth of the trie will be reduced to one and the algorithm will terminate with the next iteration. In each iteration, both depth and breadth reductions will touch each leaf, leading to a complexity of $O(N)$ for each iteration.

Viewed over all iterations, depth reduction will scan and reduce each branch of the trie once, starting from the leaves, and process each node once. The complexity of depth reduction is thus $O(\#nodes) = O(N \cdot M)$.

Obtaining a reference for each depth and breadth reduction corresponds to a check/insert operation on the depth breadth trie, which is linear in the number of nodes the reduction term contains, plus the number of sibling nodes visited. Note however that searching through a chain of sibling nodes that represent alternative paths in the trie could be too expensive (another factor of N) if we have a large number of nodes. To avoid this problem, a threshold value (8 in our implementation) controls whether to dynamically index the sibling nodes through a hash table, hence providing direct node access and optimizing search. Further hash collisions are reduced by dynamically expanding the hash tables. As each node in the proof trie participates in one reduction, for the total execution, the cost of obtaining references is thus bound by $O(8 \cdot \#nodes) = O(N \cdot M)$.

Therefore, the overall complexity of recursive node merging using a depth breadth trie is $O(min(M, N)) \cdot O(N) + O(\#nodes) + O(\#nodes) = O(N \cdot M)$. While the constant factor is higher than for the naive method, in practice, this is typically outweighted by the fact that prefix sharing causes the number of nodes in the proof trie to be far less than $N \cdot M$.