Measuring inference-engine run-time efficiencies between Truth Table enumeration, Back chaining & Forward chaining algorithms

Cormac Collins and Jacob Milligan

Computer Science - Introduction to Artificial Intelligence
Swinburne University of Technology

Contents

Introduction:

Following the creation of an ‘Inference-Engine’ we are interested in observing the expected time and performance differences between the 3 methods of inference applied to knowledge bases inferred from propositional logic. The methods include the following: Truth-Table (TT) algorithm / inference checking, Horn-Clause ‘Forward Chaining’ (FC) & Horn-Clause ‘Backward Chaining’ (BC).

Information is provided to the program through propositional logic statements i.e. $p => q; p$ this information can be parsed and implemented into a data structure appropriate for each method. More complex sentences are read in BNF (Backus-Naur Form), allowing accommodation for operator precedence and sentence hierarchy (Russell & Norvig, 2010).

Each algorithm operates on the logic of entailment α |= β , implying that a query or truth within a propositional statement is only solvable **if and only if** every model in which a is true, is also true in b (therefor a can be known). In simple terms, a proves the existence of b. Each algorithm within this inference engine can *entail* any proposition, however the algorithm may not be *complete* (doesn’t know the answer) if inadequate information is provided. Each method was implemented via the pseudo code guidelines presented by Russell & Norvig (2010).

Truth-Table:

The truth table in essence works by checking each and every possible combination of symbols capable, which are constructed as a truth table or truth table equivalent data structure. These models of the world are then tested against the ‘rules’ that are provided or ‘told’ to the agent in the form of propositional logic sentences. Our expectations are that run times will resemble $O(n^2)$, when constructing a knowledge base and answering a logical query. As we will discuss shortly, we expect this to be the slowest of each inference method. In fact this methods insufficiencies are outlined by the growth of the truth table by 2^n , n representing the number of unique symbols within a propositional statement. We aim to clearly show this insufficiency within our testing results.

Forward-Chaining:

Utilizing ‘Horn-Clause’ logical information, FC applies the laws of propositional logic to the sentences provided by finding the ‘known truths’ within the statement and proceeding to solve the ‘goal’ or the ‘ask’ query provided. Simply from this description the reader might consider this a far more efficient method for inference with an expected run time of $O(n)$ with a relatively low coefficient applied to the addition of more symbols. The FC method is defined as being *data-driven* inference Russell & Norvig (2010).

Backward-Chaining:

Also utilizing ‘Horn-Clause’ sentencing, BC essentially operates in reverse when compared to FC. By using the knowledge of the goal, the algorithm ‘works back’ through each symbol &/or sentence that is required to solve the goal. If solvable, the agent should eventually reach a sentence or symbol that it possesses information on, from here it can then proceed to solve each statement it has been saving - all the way to the goal. The key benefit expected for BC over FC, is the ability of the BC to avoid useless calculations and utilize the most important data to solving the problem. For example, if the Both FC and BC have several pieces of information available to solve a given problem, the BC is guaranteed to find the quickest steps to solving the goal, while the FC approach may first solve irrelevant statements before finding the correct path to the goal solution. BC is will also run at $O(n)$ but is expected to be very close to FC in most run-time respects. The BC method is referred to as *goal-driven* inference.

**It is important to note that a ‘horn-clause’ statement does not entail all propositional logic (such as negations), therefore test cases for these chaining methods must only contain ‘horn-clause’ logic, see Russell & Norvig (2010) for more information.*

Methodology:

Several different test cases were specifically chosen for the purpose of displaying the unique differences between each method. Below we will display and describe each test case.

Test cases

1.) T1 - *Traits being several types of propositions and significant symbol numbers*

$TELL$
$p2 => p3;\ p3 => p1;\ c => e;\ b\&e => f;\ f\&g => h;\ p1 => d;\ p1\&p3 => c;\ a;\ b;\ p2;$
ASK
d

2.) T2 - *Excess conjunctions - easy solvability for chaining*

$TELL$
$a \ \& \ b \ \& \ c \ \& \ d \ \& \ e \Rightarrow \ k1; \ a; \ b; \ c; \ d; \ e$
ASK
$k1$

3.) T3 - *High count truth knowledge*

$TELL$
$a \Rightarrow b; b \Rightarrow c; c \Rightarrow f; f \Rightarrow g; g \Rightarrow z; a; b; c; d; e; f;$
ASK
z

4.) T4 - *Long clause and high symbol count*

$TELL$
$a \Rightarrow b; \ b \Rightarrow c; \ c \Rightarrow d; \ d \Rightarrow e; \ e \Rightarrow f; f \Rightarrow g; \ g \Rightarrow h; \ h \Rightarrow i; \ i \Rightarrow j; j \Rightarrow k; \ k \Rightarrow l;$
ASK
l

5.) T5 - *Long clause and high symbol count - but early knowledge within clause chain 'g'*

$TELL$
$a \Rightarrow b; \ b \Rightarrow c; \ c \Rightarrow d; \ e \Rightarrow f; f \Rightarrow g; \ g \Rightarrow h; \ h \Rightarrow i; \ i \Rightarrow j; j \Rightarrow k; \ k \Rightarrow l; \ a; \ g$
ASK
l

6.) T6 - *Very Long clause and high symbol count*

$TELL$
$a \Rightarrow b; \ b \Rightarrow c; \ p2 \Rightarrow y; \ c \Rightarrow d; \ d \Rightarrow e; \ e \Rightarrow f; f \Rightarrow g; \ g \Rightarrow h; \ h \Rightarrow i; \ i \Rightarrow j;$
$j \Rightarrow l; \ a; \ p1; \ p2; \ p3; \ p4; \ p5; \ p6; \ p7;$
ASK
l

Testing procedure:

Each test was run 100 times for each test case, with run times (ms) and inferences solved (chaining only) being calculated for each of them. Testing was undertaken on an ubuntu operating system with an Intel(R) Core(TM) i7-4710MQ CPU @ 2.50GHz.

Results:

The results of testing showed an overall trend towards a much greater increased runtime for the Truth-Table algorithm (Figure 1), while similar times were displayed between the BC and FC algorithms with smaller discrete differences observable in Figure 4.

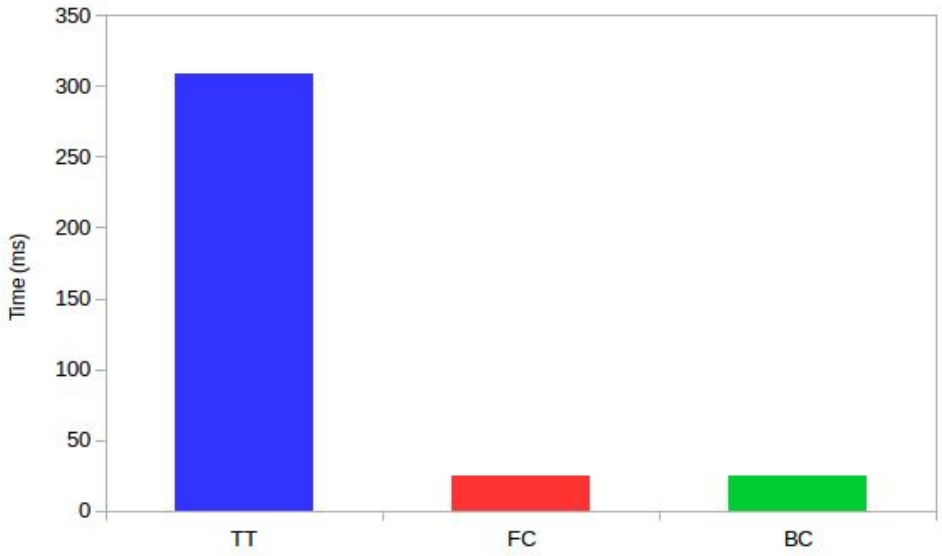

Figure 1: Average run-time for inference algorithms Truth-Table, Forward-Chaining and Backward-Chaining.

For average run-times we can observe a significantly larger value for the (308.4 ms) truth-table algorithm, while the FC & BC showed similar averages of 24.06 and 25.09 ms respectively (Figure 1). Figure 2 shows us the run-times relative to the unique symbol count for each test. There is the appearance of a trend for the TT algorithm as the symbol count moves between 7,11 and 12 symbol counts. The FC and BC algorithms display similar trends towards symbol count in our measured range.

Figure 2: Time vs Symbol Count for inference algorithms Truth-Table, Forward Chaining and Backward Chaining.

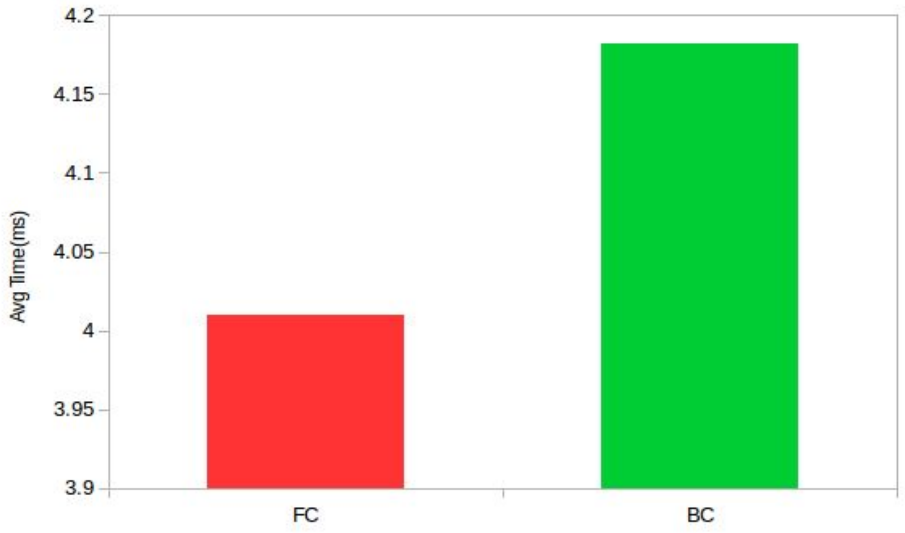

Figure 3: Average time (ms) for inference algorithms Forward Chaining & Backward Chaining.

Taking a closer look at the FC to BC comparison we can see an average difference of .17 ms with BC having a slightly larger run time of 4.18 ms compared to FC (4.01m) (Figure 3).

Figure 4: Time (ms) vs Inferences (sentences) for inference algorithms Forward Chaining and Backward Chaining for Test case T3 (High truth count).

For the test case T3 we have a closer look at the inference (sentences) difference between BC(3) and FC(8), these differences do not appear to accommodate differences in run-time.

Discussion:

It was hypothesized that we would see significant differences between the TT and chaining algorithms. We can confirm this when observing the results in Figure 1, with up to a 12.8 times greater run time on average than the chaining algorithms. We can easily attribute this to the aforementioned method used in TT, that is, an enumeration count of 2^n of n symbols. We can observe this theory further in figure 2, displaying a potential trend between symbol count and runtime for TT during what appeared to be 'mid-range' symbol counts relative to our testing (7-12). We can observe a drop off in this linear trend following the 12 count for symbols, this may be related to the particular case used for our 18 symbol test case T6.

Looking at the chaining algorithms, we are surprised by the lower average being displayed by FC (4.01ms) compared to BC (4.18ms) in figure 3, as the reverse outcomes were expected. However this difference is particularly negligible at these time differences so cannot be used to infer any meaning at this point.
It could be hypothesized that BC may be better suited when higher amounts of information (More atomically true symbols provided initially) are provided with a query i.e. $a;\ b;\ c;\ d;\ e;$. Conversely, FC may have difficulty choosing the best 'entry' point to solving the problem. In all cases BC's approach is always constant - it will always take only the steps required as opposed to the aforementioned risk placed upon FC. Despite this reasoning, BC appears to come at a higher cost as per the aforementioned difference in average run-times (Figure 3). Figure 4 does show evidence for the suggestion that BC can become more efficient when higher truth counts are provided, displayed by a lower inference count (3) compared to FC

(8) in T3 (Figure 4). However, the average run-times in this test case still showed almost indeterminant differences. Further testing of higher complexity / higher information counts is required. This theory is supported by Russell & Norvig (2010) as BC implements what is called *goal-directed reasoning* as opposed to the *data-driven reasoning* implemented by FC.

Importantly to note, is part of the implementation comparison between FC and BC. The FC method can work by solving a statement and discarding it immediately, storing only the remaining symbol truths in a lookup table. Comparatively, BC must store each unsolved premise as a 'goal' as it 'backtracks' from the initial goal state, while also storing the accumulated truth values in the KB. Understandably this may make for in general, more computation until BC has reached the point where it can solve for the goal.

Conclusion:

As per expected we have shown the inefficiency of the TT algorithm, however further testing is required to better display the differences between BC and FC. More complex sentences with higher truth counts may be a method for exposing these differences.

References:

Russell, S. J, Norvig, P. (2010) *Artificial Intelligence A Modern Approach*, Third Edition, Upper Saddle River. New Jersey: Pearson Education Inc.