Preliminary Empirical Results on Anytime Propositional Reasoning (Abstract)*

Mukesh Dalal and **Li Yang**
Columbia University
Department of Computer Science
New York, NY 10027
Email: {dalal, lyang}@cs.columbia.edu.
Phone: (212) 939–{7114, 7116}

Abstract

An anytime family of propositional reasoners is a sequence $\vdash_0, \vdash_1, \ldots$ of inference relations such that each $\vdash_k$ is sound, tractable, and makes more inferences than $\vdash_{k-1}$, and each theory has a complete reasoner in the family. Anytime families are useful for resource-bounded reasoning in knowledge representation systems. We describe implementations of an anytime family $\{\vdash_k\}$ of clausal propositional reasoners using three different strategies. We present empirical results comparing the three strategies, the completeness of reasoning, the time for making inferences, and the space used for reasoning. Our results show that the reasoners with higher values of k infer significantly more formulas than reasoners with lower values of k, and the time for inferencing decreases significantly as k is increased from 0 to 2.

Introduction

Since deductive reasoning is intractable for propositional knowledge representation systems, several tractable approaches for making incomplete inferences have been proposed (Crawford 1992). The incompleteness of these reasoners make them unsuitable for several tasks where more inferences are needed (Doyle & Patil 1991). An attractive approach that is tractable as well as complete in the limit is based on the notion of *anytime reasoners* (Boddy & Dean 1988). They are complete reasoners that provide partial answers even if stopped prematurely; the degree of completeness of the answer improves with the time used in computing the answer. They are often used for providing a quick "first cut" to a problem, which can be later improved. In (Dalal 1996b; 1996a), we presented a family $\vdash_0, \vdash_1, \ldots$ of reasoners such that each $\vdash_k$ is sound and tractable, each $\vdash_{k+1}$ is at least as complete as $\vdash_k$, and each theory has a complete reasoner $\vdash_k$ for reasoning with it. Such a family is called an *anytime family* of reasoners, since given any reasoning task, one can presumably start with $\vdash_0$, and successively proceed to the next reasoner if more time is available.

* This work is partially supported by NSF Grant IRI-94-10117, NSF infrastructure grant CDA-9625374, and DARPA/ARL Contract DAAL01-94-K-0119.

In contrast to some other resource-bounded approaches (for example, see (Zilberstein 1993)) for dealing with limited computational resources, our anytime approach does not provide explicit quality measures with the answers, but instead provides semantic justifications (Dalal 1996c). In particular, each reasoner $\vdash_k$ is also characterized by a model-theoretic semantics. Since our family $\{\vdash_k\}$ is defined using boolean constraint propagation (BCP) (McAllester 1990), the semantics of each $\vdash_k$ is based on the semantics of BCP (Dalal 1996d).

In this document, we describe implementations of our anytime family $\{\vdash_k\}$ of reasoners using three different strategies. We present empirical results comparing the three strategies, the completeness of reasoning, the time for making inferences, and the space used for reasoning. We show that (1) one particular strategies consistently outperforms the others; (2) reasoners with higher values of k infer significantly more formulas than reasoners with lower values of k; (3) the time for inferencing decreases significantly as k is increased from 0 to 2; and (4) the space required grows with the increase in k.

Although BCP is an efficient linear-time method, it does not make all inferences that are logically entailed. For example, it cannot detect the inconsistency in the theory $\{(P \vee Q), (P \vee \neg Q), (\neg P \vee Q), (\neg P \vee \neg Q)\}$. We have shown that any theory can be transformed into a logically equivalent theory from which BCP can make all allowed inferences — such theories are called *vivid.* The term vivid is inspired by (Levesque 1986), where vivid theories are ones where an answer can be "read off" quickly. For a knowledge base that is accessed frequently, it might be useful to compile its theory into an equivalent vivid theory (Selman, Levesque, & Mitchell 1992; Cadoli 1996). This process of *vivification* was defined using a fixed-point construction based on inferences made by BCP. By suitably restricting this construction, we also defined notions of partial vivification, which provides yet another complete characterization of the anytime reasoners in the family $\{\vdash_k\}$. Our algorithms for anytime reasoning are based on partial vivification.

A Family of Anytime Reasoners

In this section, we review the definitions of the anytime family and vivification (Dalal 1996b; 1996c). For this, we restrict our attention to clausal propositional theories (Mendelson 1964), that is, a theory is a set of propositional clauses. The empty clause is denoted by $\mathbf{f}$.

Clausal boolean constraint propagation (BCP) is a variant of unit resolution. Given any theory Γ, BCP monotonically expands it by adding literals as follows: in each step, if any single clause in Γ and all the literals in Γ taken together logically entail any other literal (or $\mathbf{f}$), then this literal (or $\mathbf{f}$) is added to the theory Γ. This step is repeated until $\mathbf{f}$ is obtained or no new literal can be added to the theory. For example, starting with the theory $\{\neg P, P \vee \neg Q, P \vee Q \vee \neg R, P \vee Q \vee R\}$, BCP first obtains $\neg Q$ from $\neg P$ and $P \vee \neg Q$, then $\neg R$ from $\neg P$ and $\neg Q$ and $P \vee Q \vee \neg R$, and finally $\mathbf{f}$ from $\neg P$ and $\neg Q$ and $\neg R$ and $P \vee Q \vee R$.

The *complement*, $\sim(\psi)$, of a clause $\psi = \alpha_1 \vee \ldots \vee \ldots \alpha_n$ is defined to be the theory $\{\sim\alpha_1, \ldots, \sim\alpha_n\}$. A clause ψ is BCP-inferable (or inferable using BCP) from a theory Γ, denoted by $\Gamma \vdash_{\text{BCP}} \psi$, iff BCP obtains $\mathbf{f}$ from the theory $\Gamma \cup \{\sim(\psi)\}$. Although $\vdash_{\text{BCP}}$ is sound, it is complete only for restricted classes of theories, for example, Horn theories. A common source of incompleteness in $\vdash_{\text{BCP}}$ is its inability to use previously inferred clauses for inferring new clauses. For example, for the theory $\Gamma_0 = \{P \vee Q, P \vee \neg Q, \neg P \vee S \vee T, \neg P \vee S \vee \neg T\}$, both $\Gamma_0 \vdash_{\text{BCP}} P$ and $\Gamma_0 \cup \{P\} \vdash_{\text{BCP}} S$, but $\Gamma_0 \not\vdash_{\text{BCP}} S$. A theory Γ is called *vivid* if for any clause ψ: $\Gamma \models \psi$ iff $\Gamma \vdash_{\text{BCP}} \psi$.

If $\vdash_{\text{BCP}}$ is extended by allowing chaining on arbitrary formulas, the resulting entailment will be sound, complete, and intractable. So, we allow chaining on only a restricted set of formulas for defining the anytime family $\vdash$: For any natural number k, the consequence relation $\vdash_k$ is defined using the following two inference rules:

$$1.\ \frac{\Gamma \vdash_{\text{BCP}} \varphi}{\Gamma \vdash_k \varphi} \qquad 2.\ \frac{\Gamma \vdash_k \psi; \quad \Gamma, \psi \vdash_k \varphi}{\Gamma \vdash_k \varphi} \quad \text{for } |\psi| \leq k;$$

where Γ is any theory, ψ is any formula, and φ is any formula.

We now define the vivification process. For any theory Γ, we restrict our attention to only those clauses that are built from the atoms in Γ such that all literals in a clause have distinct atoms; these clauses are called *basic clauses*. A basic clause with at most k literals is called a k-clause. The *extended Herbrand base*, $E(\Gamma)$, of a theory Γ is the the set of all basic clauses, and the *k-extended Herbrand base*, $E(\Gamma, k)$, is the the set of all k-clauses in $E(\Gamma)$.

The operator $T_{\Gamma,k}$ on any set S of k-clauses produces the set of k-clauses that can be BCP-inferred from $\Gamma \cup S$, that is:

$$T_{\Gamma,k}(S) = \{\mu \in E(\Gamma, k) \mid \Gamma \cup S \vdash_{\text{BCP}} \mu\}.$$

Since $T_{\Gamma,k}$ is a monotonic operator over a finite lattice, it has a least fixpoint (Tarski 1955), which we denote by $\text{lfp}(T_{\Gamma,k})$. We will refer to $\text{lfp}(T_{\Gamma,k})$ as the kth fixpoint of Γ; k is said to be the *index* of this fixpoint. For example, if $\Gamma = \{(P \vee Q), (\neg P \vee Q), (P \vee \neg Q)\}$ then $\text{lfp}(T_{\Gamma,0}) = \emptyset$ and $\text{lfp}(T_{\Gamma,k}) = \{(P), (Q)\}$ for each $k > 0$. For any theory Γ and any number k, $\text{Viv}(\Gamma, k)$ is defined to be the theory $\Gamma \cup \text{lfp}(T_{\Gamma,k})$. Note that $\text{Viv}(\Gamma, k)$ augments the theory Γ, rather than replacing it, by the theory $\text{lfp}(T_{\Gamma,k})$, since this allows more clauses to be inferred from it using $\vdash_{\text{BCP}}$.

In (Dalal 1996c), we show that for any theory Γ, any clause ψ, and any number k: $\text{Viv}(\Gamma, k) \equiv \Gamma$ and $\text{Viv}(\Gamma, k) \vdash_{\text{BCP}} \psi$ iff $\Gamma \vdash_k \psi$. Thus, vivification can be used for anytime reasoning using the family $\{\vdash_k\}$ of reasoners.

Algorithms and Strategies

Algorithm `Vivify(k)` reads in a theory Γ and produces the theory $\text{Viv}(\Gamma, k)$ by possibly adding several basic clauses with at most k literals each.

Algorithm Vivify(k)
```
1. ReadTheory();
2. for size = 1 to k do
3.    Round(size);
4.    if (UNSAT) return(f);
5. return(t);
```
end (Vivify).

The exception UNSAT is trigerred anytime $\mathbf{f}$ is inferred from the theory. Procedure `Round(size)` iterates over basic clauses with at most `size` literals. It restarts enumerating clauses of size 1 onwards (`repeat` loop) as soon as any new clause is added to the theory. The intuition is that adding a new clause may cause the theory to entail new smaller size clauses, and adding smaller size clauses should speed up the vivification process. In Section 4, we provide empirical evidence that this strategy is faster than the one that does not reset the enumeration.

Procedure Round(max-size)
```
 1. start-size = max-size;
 2. repeat
 3.    for size = start-size to max-size do
 4.        start a new clause (pos=1);
 5.        while (pos > 0) do
 6.            pos = NextPos(pos, size);
 7.            if (UNSAT) return;
 8.        if (pos < 0) break {for};
 9.    if (pos = 0) return;
10.    if (pos < 0)
11.        start-size = 1;
```
end(Round).

A new clause in `Round(size)` is usually created from the current clause by adding a new literal at position `pos`.

Procedure `NextPos(pos, size)` generates the new literal by considering literals whose truth values are not yet determined. If such a literal is found, it tests whether the current clause can be BCP-inferred from the current theory (using `Ask-Lit(lit)`); if yes, the clause is added to the theory and the enumeration is reset by returning a negative flag. The next position is otherwise returned if the current clause hasn't exceeded the `size` bounds. If the current clause has `size` literals or no new literal can be found for the current position, it backtracks and removes the last literal that was added. If `pos` falls to 0, it means that no new clause of current `size` can be generated.

Among all possible choices at any position, the literal which occurs most often is chosen. We have tried another strategy (Dubois *et al.* 1995) that uses the weighted counts of both the literal and its negation. In Section 4, we provide empirical evidence that our strategy is faster than the one with weighted counts.

Procedure `Ask-Lit(lit)` determines whether `lit` is entailed using BCP from the current theory. It does this by adding the complement of `lit` as a unit clause to the current theory and invokes Procedure `BCP`. Any unit clause added to the theory is also added to the `Stack` of unprocessed unit clauses.

Procedure BCP

```
1. while Stack is not empty do
2.    lit = pop(Stack);
3.    for all cls in Head(lit) do
4.         ShortenHead(cls);
5.    for all cls in Tail(lit) do
6.         ShortenTail(cls);
```
end (BCP).

For each literal popped from the *Stack*, BCP (see (Zhang & Stickel 1996)) unit-resolves on the clauses having the negation of the literal either as the first (*head*) or the last atom (*tail*). Procedure `ShortenHead(cls)` examines the clause `cls` from its second literal to the last one. When encountering a literal set to TRUE, it returns because the clause is already satisfied. Literals set to FALSE are skipped because of unit-propagation. If all literals in the clauses are set to FALSE, the UNSAT flag is set because the clause can not be satisfied. Otherwise, the first unset literal becomes the new head of the clause. Whenever head of a clause meets its tail, the clause should be added again as unit clause by using Procedure `Add-Unit`. Procedure `ShortenTail` is identical to `Shortenhead`, except that it works from the second last literal back to the first one and determines the new tail of the clause.

Name	Theories	Atoms	Clauses	Sat	Unsat
aim1	8	100	160	4	4
aim2	4	200	320	0	4
jnh1	16	100	800	7	9
jnh2	18	100	850	2	16
jnh3	9	100	900	0	9
bf	2	1000	3500	0	2

Table 1: Groups of theories

Empirical Results

We have implemented our vivification algorithm using the three strategies described in Section 3. We ran the vivification algorithms and anytime reasoners based on them on several propositional theories taken from benchmarks archived at DIMACS web site (http://dimacs.rutgers.edu/). In this section, we present the results of these experiments.

Table 1 lists some information about the theories which are clustered into 6 named groups. It presents the number of theories, satisfiable theories, and unsatisfiable theories in each group. The number of atoms and number of clauses in each theory (identical within a group, except for the last where the numbers are approximate) are also provided.

Table 2 shows a comparison of the three strategies we have implemented so far on all the unsatisfiable theories. The algorithm `Vivify` in Section 3 describes the *Reset* strategy. The *No Reset* strategy does not reset the enumeration of clauses after adding a new clause, while the *Weighted Count* strategy uses a different strategy for selecting the next literal. The numbers in the table give the CPU times (in seconds) used for inferring that the theories are unsatisfiable (minimum, average, and maximum for each group). The data shows that *Weighted Count* is faster than *No Reset* in most cases, while the *Reset* shows tremendous improvements over the other two strategies in all cases, except for some *jnh* cases where the absolute time difference is slight.

The rest of the tables focus only on the satisfiable theories. Table 3 gives the fraction of number of basic clauses (with at most 3 literals) that can inferred using BCP from $\text{Viv}(\Gamma, \mathit{level})$ but not from Γ. For example, about 87% of all basic clauses were inferable from $\text{Viv}(\Gamma, 2)$ but not from an *aim1* theory Γ. The data shows that the completeness of reasoning with BCP increases with the level of vivification. Although there is a significant increase in completeness in going from Level 1 to Level 2, there is only a little increase in then going to Level 3.

Table 4 shows the time improvement in inferring basic clauses that are inferable using BCP from both Γ and $\text{Viv}(\Gamma, \mathit{level})$. In particular, it shows the improvement factor due to vivification. For example, it is about 15 times faster to infer clauses from $\text{Viv}(\Gamma, 2)$ than from an *aim1* theory Γ. The data shows that efficiency increases as we move from Level 0 (that is, Γ) to Level 2, and then often decreases to Level 3.

Name	No Reset			Weighted Count			Reset		
	Min	Aver	Max	Min	Aver	Max	Min	Aver	Max
aim1	1.120	6.105	14.350	2.330	4.362	9.170	1.120	2.010	2.850
aim2	5.800	990.807	3744.130	4.220	859.130	3340.120	1.950	53.807	198.500
jnh1	0.080	1.876	6.430	0.070	0.746	4.000	0.080	1.857	6.000
jnh2	0.120	115.284	1827.570	0.050	7.526	114.050	0.080	57.123	894.920
jnh3	0.030	0.903	5.280	0.010	1.064	8.320	0.020	0.915	5.460
bf	7967.330	11371.640	14775.950	2565.820	4288.545	6011.270	603.180	1167.430	1731.680

Table 2: Comparison of three strategies (CPU time in secs)

Name	Level 1			Level 2			Level 3		
	Min	Aver	Max	Min	Aver	Max	Min	Aver	Max
aim1	0.000000	0.000000	0.000000	0.870078	0.870325	0.870805	0.870078	0.870325	0.870805
jnh1	0.000000	0.073649	0.351922	0.327409	0.586764	0.793308	0.327632	0.586940	0.793308
jnh2	0.053111	0.313945	0.574778	0.453137	0.513958	0.574778	0.453150	0.513964	0.574778

Table 3: Fraction of clauses inferable after each level

Name	Level 1			Level 2			Level 3		
	Min	Aver	Max	Min	Aver	Max	Min	Aver	Max
aim1	0.743	1.066	1.457	5.647	15.190	34.250	4.576	14.090	34.250
jnh1	1.001	3.238	8.687	1.031	42.890	131.814	0.862	41.512	131.814
jnh2	8.735	94.233	179.730	11.741	99.330	186.920	11.334	92.204	173.074

Table 4: Time improvement (factor) in inference after each level

Name	Level 1			Level 2			Level 3		
	Min	Aver	Max	Min	Aver	Max	Min	Aver	Max
aim1	1.000	1.000	1.000	1.033	1.148	1.250	1.033	1.148	1.250
jnh1	1.000	1.001	1.006	1.041	1.123	1.353	1.044	1.218	1.589
jnh2	1.002	1.004	1.005	1.005	1.099	1.192	1.005	1.544	2.083

Table 5: Size of theory after each level (as fraction of initial size)

Table 5 shows the increase in size of the theory because of vivification. For example, Viv(Γ, 2) is about 1.148 times the size of an *aim1* theory Γ. The data shows that the sizes increase with increase in level, as expected.

Conclusions

We presented some empirical results demonstrating that vivification using BCP increases the completeness as well as efficiency of reasoning, at least until Level 2. We have used vivification for anytime temporal reasoning (Dalal & Feng 1996). Our current work involves removing redundant clauses to offset the increases in sizes of the theories.

References

Boddy, M., and Dean, T. 1988. Solving time dependent planning problems. Technical report, Dept. of Computer Science, Brown University.

Cadoli, M. 1996. Panel on knowledge compilation and approximations: terminology, questions, and references. In *Fourth International Symposium on Artificial Intelligence and Mathematics (AI/MATH-96)*, 183–186.

Crawford, J., ed. 1992. *Proceedings of the AAAI Workshop on Tractable Reasoning*. San Jose, California: American Association for Artificial Intelligence.

Dalal, M., and Feng, Y. 1996. Anytime temporal reasoning based on propositional satisfiability (extended abstract). In Freuder, E. C., ed., *Proceedings of Second International Conference on Principles and Practice of Constraint Programming (CP96)*, 535–536. Cambridge, Massachusetts: Springer.

Dalal, M. 1996a. Anytime clausal reasoning. Submitted to Annals of Mathematics and Artificial Intelligence.

Dalal, M. 1996b. Anytime families of tractable propositional reasoners. In *Fourth International Symposium on Artificial Intelligence and Mathematics (AI/MATH-96)*, 42–45.

Dalal, M. 1996c. Semantics of an anytime family of reasoners. In Wahlster, W., ed., *Proceedings Twelveth European Conference on Artificial Intelligence (ECAI 96)*, 360–364. Budapest, Hungary: John Wiley and Sons, Ltd.

Dalal, M. 1996d. Semantics of an efficient propositional reasoner: Preliminary report. In Stewman, J. H., ed., *Proceedings Ninth Florida AI Research Symposium (FLAIRS-96)*, 101–105.

Doyle, J., and Patil, R. 1991. Two theses of knowledge representation: language restrictions, taxanomic classification, and the utility of representation services. *Artificial Intelligence* 48(3):261–297.

Dubois, O.; Andre, P.; Boufkhad, Y.; and Carlier, J. 1995. SAT versus UNSAT. *DIMACS series in Discrete MAthematics and Theoretical Computer Science* 24.

Levesque, H. 1986. Making believers out of computers. *Artificial Intelligence* 30:81–108.

McAllester, D. 1990. Truth maintenance. In *Proceedings Eighth National Conference on Artificial Intelligence (AAAI-90)*, 1109–1116.

Mendelson, E. 1964. *Introduction to Mathematical Logic*. Princeton, N.J.: Van Nostrand.

Selman, B.; Levesque, H.; and Mitchell, D. 1992. A new method for solving hard satisfiability problems. In *Proceedings Tenth National Conference on Artificial Intelligence (AAAI-92)*, 440–446.

Tarski, A. 1955. A lattice-theoretical fixpoint theorem and its applications. *Pacific J. Math.* 5:285–309.

Zhang, H., and Stickel, M. E. 1996. An efficient algorithm for unit propagation. In *Fourth International Symposium on Artificial Intelligence and Mathematics (AI/MATH-96)*, 166–169.

Zilberstein, S. 1993. *Operational Rationality through Compilation of Anytime Algorithms*. Ph.D. Dissertation, University of California, Berkeley, California.