Heuristic Search

In

Symbolic Probability Inference

by

Yonghong Pan

A research project submitted in partial
fulfillment of the degree of Master of Science

Major Professor: Bruce D'Ambrosio

Department of Computer Science
Oregon State University
Corvallis, OR 97331

November 18, 1991

ACKNOWLEDGMENTS

Thanks to my major professor, Bruce D'Ambrosio, for his support and guidance throughout this project. Thanks also to my other committee members, Professors Theodore G. Lewis and Prasad Tadepalli, for their assistance. Thanks to my friend Zhaoyu Li for discussion on this project. Thanks.

Table of Contents

0. Abstract

I present a new heuristic search approach to compute approximate answers for the probability query in belief nets. This approach can compute the 'best' bounds for a query in a period of any given time (if time permitted, it will get an exact value). It inherits the essence of Symbolic Probabilistic Inference (SPI), which is the factoring part of SPI, and searches the structure passed by SPI to find a approximate value. This paper also presents the theoretical background for this approach. Empirical results are presented for three heuristics of this approach and a best first search approach tested in a set of randomly generated belief nets and a net from the real world.

1. Introduction

The belief networks is a popular graphical approach for representing uncertain expert knowledge reasoning in coherent probabilistic form. Many algorithms have been developed for evaluating queries and performing probabilistic inference on belief networks. They can be classified into two groups: exact techniques and approximate techniques.

Two of the most well known exact techniques are Shachter's graph reduction algorithm [Shachter, 1986, 1989], and Pearl's message passing algorithm [Pearl,1988]. Both of these two approaches basically use a forward or data-directed control regime. A relatively new approach [D'Ambrosio, 1989], Symbolic Probabilistic Inference (SPI), however, uses a backward or goal-driven regime for probabilistic inference, and its performance is superior according to the empirical results of [Li, 1990], [D'Ambrosio & Li, 1991].

Exact diagnostic inference in general multiply connected networks has been shown to be NP-hard [Cooper 1987]. Therefore, there is considerable interest in the development of methods that provide greater efficiency at the cost of imprecision in the results. There have been two main directions in which researchers have sought efficient approximate algorithms. One approach involves random sampling of network instantiations or stochastic simulation [Henrion, 1988], [Chin &Cooper, 1987], [Pearl, 1987], [Shacher&Peot, 1989]. The other involves the search methods to find the most probable

hypotheses [Shimony & Charniak, 1990], and the bound on the exact posterior probabilities [Henrion, 1991]

In this paper, I present a search approach to compute approximate answers for the probability query in belief nets. In general, the answer to any query is the sum of a large number of terms, each of which in turn is the product of a set of probabilities provided in the original model. We believe that, under reasonable assumptions about the nature of the originally provided probabilities, a few large terms will dominate this sum, and the remaining terms will be very small and contribute little to the final answer. My algorithm employs the heuristic search method to search for the largest terms or mass terms thus far in the SPI internal computing structure, so that it can compute the lower and upper bound for the query. Additional search progressively narrows the bounds on the probabilities. Therefore, it can compute the 'best' bounds for the query in a period of any given time, finally, if time permitted, it can compute the exact value.

In the rest of this paper, section 2 presents some background knowledge about SPI. Section 3 presents some theoretical background and methodology based on it. Section 4 presents the empirical results. Section 5 discusses the future research. Finally, section 6 gives a summary.

2. Background

2.1. Belief Nets

A belief network consists of a directed acyclic graph G = (V,E), and a set of numeric probability distributions, where nodes in V represent variables, arcs in E and numeric probability distributions represent probabilistic relationships between the nodes (marginal for those nodes with no incoming arcs, otherwise conditioned on the nodes at the tails of the incoming arcs). Furthermore, for all $v \in V$, if c(v) is the set of all parents of v, and a(v) is the set of propositional variables in V excluding v and v's descendents, and W is any subset of a(v), then W and v are conditionally independent given c(v).

2.2. Symbolic Probabilistic Inference

Symbolic probabilistic inference (SPI) [D'Ambrosio 1989] is a goal-driven method which uses Bayes theorem directly for probabilistic inference in belief nets. Processing takes place in three phases: phase one forms an algebraic expression which corresponds to the answer to any query. Phase two constructs a factoring of the expression which permits efficient evaluation. Phase three finally evaluates the factored expression to determine the numeric answer. In the rest of this section, I will present some examples [D'Ambrosio 1989] to illustrate how the SPI works.

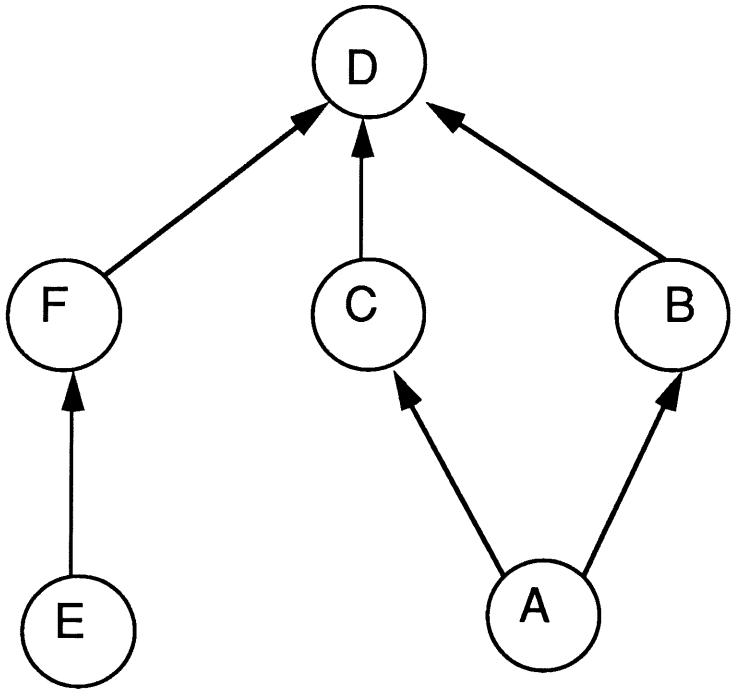


Figure 1: A Simple Belief Net

In a simple belief net shown in figure 1, using the chain rule of inference, we can compute the following symbolic expressions for the prior probability of each node:

$$P(A) = P(A)$$

$$P(B) = \Sigma_A \, P(B|A)P(A)$$

$$P(C) = \Sigma_A \, P(C|A)P(A)$$

$$P(D) = \Sigma_{ABCEF} P(D|BCF)P(F|E)P(E)P(C|A)P(B|A)P(A) \quad (1)$$
$$P(E) = P(E)$$
$$P(F) = \Sigma_E P(F|E)P(E)$$

Now we can compute the prior probabilities for any node (variable) in the belief net from the above expressions (we only discuss prior probabilities here for simplicity). This method extends quite simply to include queries about arbitrary joint distributions simply by evaluating the union of the symbolic expressions for the corresponding nodes. But, the problem is the efficiency. That is computational complexity. Efficient evaluation of the expression requires keeping the size of intermediate results small. The key to efficient evaluation lies in recognizing that summation over some dimensions can be done early in the computation, rather than at the end. For example, we can more efficiently evaluate P(D) as follows:

$$P(D) = \Sigma_{BC}(\Sigma_F(P(D|BCF)(\Sigma_E P(F|E)P(E))))(\Sigma_A P(B|A)(P(C|A)P(A))) \quad (2)$$

The ideal solution is to find an optimal factoring for the expression, but optimal factoring is a hard problem. The factoring problem SPI considered is the factoring, which 1). minimizes the number of floating point multiplications needed for evaluation, and 2). makes use of the intermediate results computed in previous queries.

SPI splits the factoring problem into two parts: a static part, in which a structure (partition tree) is established to guide query evaluation, and a dynamic part consisting of: first, a control heuristic that manages information flow between partitions; and second, a local-evaluation heuristic which generates efficient evaluation trees for distributions being combined within a partition. The partition tree, and the evaluation tree of the belief net in figure 1 are given in figure 2 and figure 3, respectively.

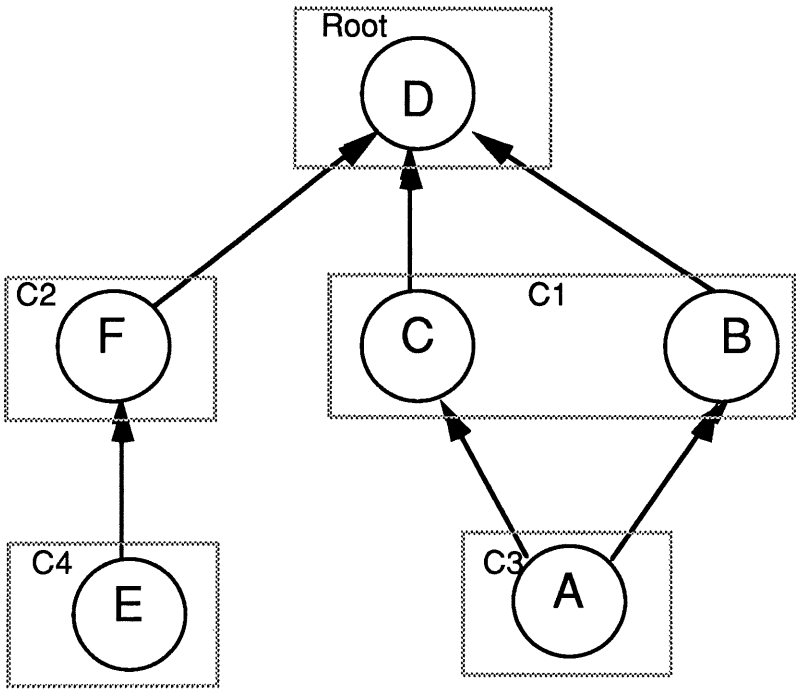


Figure 2: A Partition of the Sample Belief Net

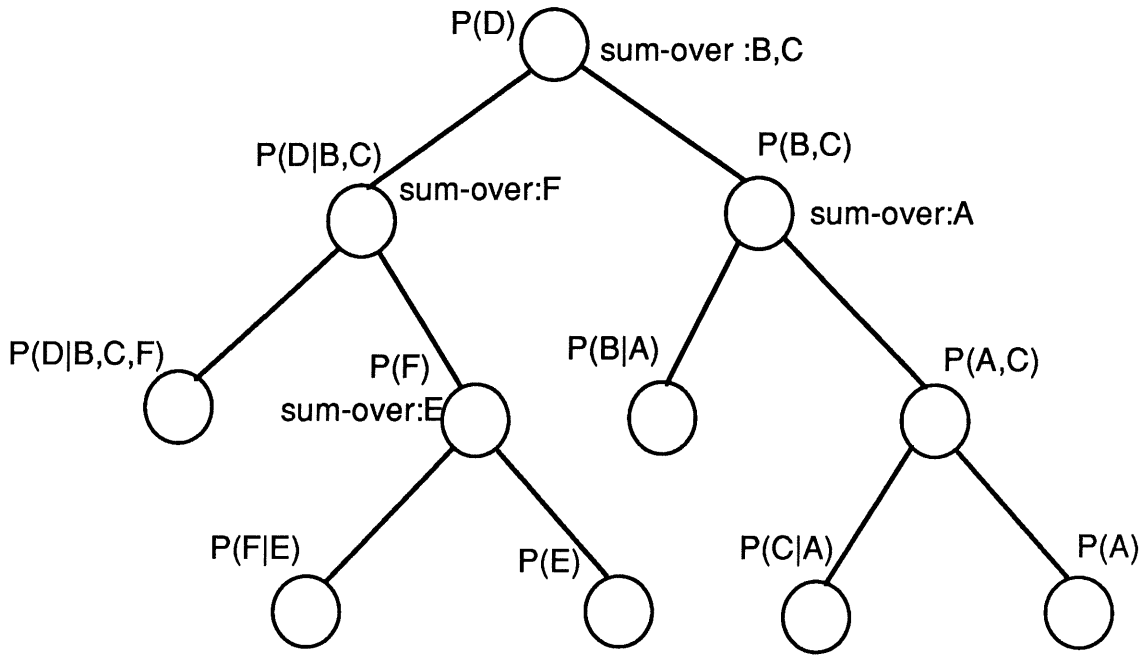


Figure 3: An Evaluation Tree for Query P(D)
Actually, this tree is equivalent to equation (2)

2.3. Search-Based Approximation in SPI

SPI is quite efficient, and its performance is the best according to the empirical result of [Li, 1990], [D'Ambrosio & Li, 1991], due to its approach to the factoring problem mentioned in the above section. However, since exact computation for probabilistic inference is a NP-hard problem [Cooper, 1987], we have to find an approximation approach for probabilistic inference for very large belief net.

Our approximate approach inherits the essence of SPI, which is the factoring part of SPI (the first two phases of SPI mentioned in section 2.1), and searches in the structure (a evaluation tree) passed by SPI to find an approximate value. In other words, to answer any query in a belief net, it first produces an evaluation tree for the query using the SPI techniques, then searches the evaluation tree to find the approximate value for the query. We will present our methodology in the next section.

3. Notation and Methodology

3.1. An Overview

In general, the answer to any query is the sum of a large number of terms, each of which in turn is the product of a set of probabilities provided in the original model. For example, in the sample belief net, a query P(D) is a sum of terms given by formula (1) or (2). Each term is the product of probabilities P(D|BCF), P(F|E), P(E), P(C|A), P(B|A), and P(A). We believe that, under reasonable assumptions about the nature of the original provided probabilities, a few large terms will dominate this sum, and the remaining terms will be very small and contribute little to the final answer. The problem of computing an approximate answer quickly, then, can be reduced to the problem of identifying these large terms. An obvious approximation method is to search those large terms among all the terms for the query.

3.2. Notation and Theorem

Suppose we have a belief net C = (V, E, P), where G = (V, E) is a directed acyclic graph, and P is the corresponding probability

distribution. For any $v \in V$, let $c(v)$, $V \supseteq c(v)$, be the set of all parents of v, then for each node v in the graph, we have a probability distribution $P(v|c(v))$, (for those root nodes, $c(v)$ is empty).

Theorem 1. For any belief net $C = (V, E, P)$, $P(V)$ is given by

$$P(V) = \prod_{v \in V \ \& \ P(c(v))>0} P(v|c(v))$$

where $\prod_{v \in V \ \& \ P(c(v))>0}$ means we are taking the product of all propositional variables in V for which $P(c(v))>0$ [Neapolitan, 1990].

In the rest of this paper, we will name this product as ***a term***, and $P(v|c(v))$ as ***an item***.

Theorem 2. For any subset V' of V,

$$P(V') = \sum_{v \in (V - V')} \prod_{P(c(v))>0} P(v|c(v))$$

Proof: From the property of marginal probability,

$$P(V') = \sum_{v \in (V - V')} P(V)$$

$$= \sum_{v \in (V - V')} \prod_{P(c(v))>0} P(v|c(v)) \qquad \{\text{theorem 1}\}$$

In other words, theorem 2 tells us that any query is a sum of the related terms.

Corollary 3. If we have a set of terms for P(V'), let say, $t_1, t_2, ..., t_n$, then

$$P(V') \geq \sum_{i=1}^{n} t_i$$

Proof: we know that $P(v|c(v) \geq 0$, thus, each term ≥ 0,

$$P(V') = \sum_{all} t_j$$

$$\geq \sum_{i=1}^{n} t_i$$

Theorem 4. If all the values V' can take are $x_1, x_2, ..., x_n$, and $P(V'= x_i) \geq L_i$, then

$$P(V' = x_i) \le 1 - \sum_{j \neq i} L_j$$

In other words, if we know all the lower bounds for P(V'), we know the upper bounds too.

Proof: from the definition of the probability

$$\sum_{i=1}^{n} P(V' = x_i) = 1$$

$$P(V' = x_i) = 1 - \sum_{j \neq i} P(V' = x_j)$$

$$\le 1 - \sum_{j \neq i} L_j$$

The above lower and upper bound theory is proved for the case without observed value (prior probability). In fact, it is also satisfied for the case with observed value (posterior probability).

Theorem 5. Suppose the observed variable set is Obs, the query variable set is Q, and we already know $P(Q, Obs) \ge L$, then L is also the lower bound of the posterior probability of Q.

Proof: from the definition of condition probability

$$\begin{aligned} P_{post}(Q) &= P(Q \mid Obs) \\ &= P(Q, Obs) / P(Obs) \\ &\ge P(Q, Obs) \qquad \{\text{because } P(Obs) \le 1\} \\ &\ge L \end{aligned}$$

Obviously, corollary 3 is also satisfied for the case with observed values.

From Corollary 3, Theorem 4, and Theorem 5, we know that, if we find some terms for P(V'), we get the lower bounds and upper bounds for P(V'). The more terms or the bigger term we find, the narrower bounds we will get. This is the basic theory our search algorithm bases on. The problem is how to search the bigger term efficiently? Unfortunately, searching for the biggest term is another NP-hard problem. SPI [D'Ambrosio & Shachter, 1990] gives more

theorems and heuristics about how to avoid the redundant terms, how to get better factoring, and how to cache the intermediate values. Our algorithm will do the search on the evaluation tree given by SPI, which is essence of the factoring problem SPI dealing with.

An evaluation tree is a representation of SPI factoring for a query. It is a binary tree having the following properties:

1. The leaf nodes are distributions of the original belief net. For example, in figure 3, P(A), P(C|A), P(B|A), P(E), P(F|E), and P(D|B,C,F) are all in leaf nodes of the tree.

2. The internal nodes are the probability distributions produced during the computation for the query (The root of the tree is the queried distribution). Each probability is the product of its two children's probabilities, summed over the sum-over variables in *sum-over* (*sum-over* is the variable set to be summed over). For example, in figure 3, P(A,C), P(B,C), P(F), P(D|B,C), and P(D) are all in the internal nodes of the tree (we name these intermediate probabilities as ***mass items***), and $P(D|B,C) = \Sigma_F P(D|B,C,F)\ P(F)$. In SPI, all these probabilities are exact values, but in our system, they may be the partial (accumulated) values. Corresponding to the notation after theorem 1, each *item* P(v|c(v)) is stored in the leaf node, a *term* is the product of all probabilities in the leaf nodes. We define a ***mass term*** as following: a mass term of a query or an evaluation tree is the product of items or mass items satisfying : i) each (mass) item's ancestor mass item (ancestor, descendant, brother, and father here are all in terms of the evaluation tree) is not in the same mass term, and ii) each (mass) item's brother (mass) item or brother's descendant item must be in the mass term. In other words, we can get a mass term by repeatedly substituting two brother (mass) items in a (mass) term by its father mass item. For example, in figure 3, we get a mass term P(D|B,C,F)P(F)P(B|A)P(C|A)P(A) by substituting P(F) for P(F|E) and P(E) from a term P(D|B,C,F)P(F|E)P(E)P(B|A)P(C|A)P(A). We can get another mass term P(D|B,C)P(B|A)P(C|A)P(A) by substituting P(D|B,C) for P(D|B,C,F) and P(F) from a mass term P(D|B,C,F)P(F)P(B|A)P(C|A)P(A).

Max Dim of an evaluation tree is the maximum dimension of probability distributions in the evaluation tree. For example, the Max Dim of the evaluation tree in figure 3 is 4.

3.3. Algorithms

The heart of the SPI heuristic search algorithm *Query*, showed in Figure 4, has four input parameters:
q: the current partial term value (a partial term is the partial product of a term), the first time called value is 1.
B: the query variable set and the corresponding query values.
H: the heuristic function.
T: the evaluation tree (in fact, it is a pointer to the tree).

Query returns the 'biggest' term or mass term for query B thus far, in terms of the heuristic function H, and the state of the evaluation tree T. It will return the next 'biggest' (mass) term next time *Query* is called.

Algorithm *Query* can be described as: At any node i of the evaluation tree, when a query arrives (together with a found partial term value), if the value for the query has already been cached at the node, we obtain a term by multiplying this cached value and the found partial term value. Otherwise, it will call *HSearch* from its both subtrees to get heuristic values. Then, *Instantiate* the whole subtree with a bigger heuristic value (hopefully, instantiating this subtree will get a bigger subterm. And instantiating the subtree will bind *sum-over* set a value not summed over), and *Query* (a recursive call) the other subtree. In other words, the sum-over order at the internal nodes of the evaluation tree is guided by the *HSearch* and the heuristic function *H*.

Query is implemented as a non-backtracking search. One reason is that any query is a sum of a large number of terms, each term we found is useful for the query, even though it is relatively small. Another reason of not a best first search is that best first search approach tends to be memory bound, especially for large belief nets. Finally, the time needed to find the biggest (next) term in the worst case is exponential to the number of nodes in belief net (Thus, in some case, search for the biggest term may even take longer time than the exact value). *Query* here is a compromise of the SPI structure (evaluation tree), time and space. The worst case time for finding a next 'biggest' term is guaranteed to be polynomial with respect to some heuristic functions.

```
algorithm Query(q, B, H, T)
begin
   if (there is a whole value for P(B) already cached at the root of T) then
      return (the cached value);
      {we find a term (q*the return value)}
   while(there is no a whole value for query B cached at the root of T)
   begin
        (v1, B1) = HSearch(B, H, leftSubTree(T));    {note 1}
        (v2, B2) = HSearch(B, H, rightSubTree(T));
        if (v1 > v2) then
          begin
             (q1, B12) = Instantiate((B ∪ B1), H, leftSubTree(T)); {note 2}
             q2 = Query(q1*q, (B ∪ B12), H, rightSubTree);
             Cache(q1*q2);
             q1 = Query(q2*q, B ∪ S), H, leftSubTree(T));{note 4}
             Cache(q1*q2);
          end
        else
          begin
             (q2, B21) = Instantiate((B ∪ B2), H, rightSubTree(T));
             q1 = Query(q2*q, (B ∪ B21), H, leftSubTree);
             Cache(q1*q2);
             q2 = Query(q1*q, B ∪ S), H, rightSubTree(T));{note 4}
             Cache(q1*q2);
          end  {end if, note 3}
   end; {end while, a whole value for query B cached at the root of T}
   P(B) = (the cached value);
end;
```

Figure 4. the recursive call procedure for each node in the evaluation tree.

Note 1: (v, B') = HSearch(B, H, T), will search in the tree T to find the 'biggest' value v, in terms of the heuristic function H and the query set B, where B' is the instantiated set (or called binding) for value v. *HSearch* algorithm is as following:

```
algorithm HSearch(B, H, T)
begin
   (v, B') = H(B, T); {note 1.1}
   if (v < 0) then   {note 1.2}
     begin
       (v1, B1) = HSearch(B, H, leftSubTree(T));
       (v2, B2) = HSearch(B, H, rightSubTree(T));
       if (v1> v2) then
         return (v1, B1)
       else
         return (v2, B2);
     end
   else
     return (v, B')
end;
```

Note 1.1: For simplicity, we suppose heuristic function H return a value v, and binding B. Actually, this is not true for all heuristics.

Note 1.2: (v < 0) means that the heuristic function H can't find a proper value.

Note 2: (v, B') = Instantiate(B, H, T), will instantiate the whole tree T, to get the 'biggest' term v and the corresponding binding B', with respect to the heuristic function H, the instantiated set B, and the space not yet instantiated. The *Instantiate* algorithm is as following:

```
algorithm Instantiate(B, H, T)
begin
   (v, B') = H(B, T);
   if (v < 0)    {heuristic H can't get a proper value}
      begin
        (v1, B1) = HSearch(B, H, leftSubTree(T));
        (v2, B2) = HSearch(B, H, rightSubTree(T));
        if (v1 > v2) then      {note 3}
          begin
             (q1, B12) = Instantiate((B ∪ B1), H, leftSubTree(T));
             (q2, B22) = Instantiate((B ∪ S), H, rightSubTree(T));{note 4}
             Cache(q1*q2);
          end
        else
          begin
             (q2, B22) = Instantiate((B ∪ B2), H, rightSubTree(T));
             (q1, B12) = Instantiate(B ∪ S), H, leftSubTree(T));{note 4}
             Cache(q1*q2);
          end
        return (the cached value and the corresponding binding);
      end
   else
      return (v, B');
end;
```

Note 3: This if-statement instantiates the subtree with bigger heuristic return value v first. It employs the heuristics of ``bigger item implying bigger term''.

Note 4: S is the instantiated sum-over variable set at root of the tree T. It is instantiated by its first instantiated subtree.

This algorithm is implemented in LISP as a generator. Every time it is called, it returns a 'biggest' term or a mass term thus far. In other words, it narrows the lower and upper bound for query B every time

it is called. This implementation is especially suitable for the real time decision making/diagnosis problem. In real time diagnosis decision making, it can be called repeatedly until the due time, or until it gets a satisfying bound (knowing an absolute bound is enough in some situation). Of course, it will give an exact answer if there is enough time.

3.4. Complexity

The most expensive routine in algorithm *Query* is *H* and *HSearch*. The number of times *Query* calls *HSearch* is O(Tsize) (Since whether *HSearch* calls itself or not depends on heuristic *H*, O(Tsize) here does not include *HSearch* calling itself, we will discuss more in section 3.5), where Tsize is the number of nodes of the evaluation tree, Tsize = 2n -1, and n is the number of nodes in the belief net. Thus, the number of times *Query* calling *HSearch* is O(n). Similarly, the number of times *Query* calls *H* is O(n) (not including *HSearch* calling *H*). What is the complexity of *H* and *HSearch*? They will be discussed below.

Our data structure is the evaluation tree for a query, and the information kept in each node of the partition tree. Therefore, the space complexity is the same as SPI.

3.5. Heuristics

The heuristic function in algorithm *Query* is a parameter so that we can easily try different heuristics, and see which heuristic is best for which kinds of belief nets.

3.5.1. H0: Naive Heuristic

H0 will guide *Query* to do summation over *sum-over* set at each internal node in a numeral enumeration order. In other words, once *H0* is called at node i, it will return the next value set in the enumeration order for *sum-over*. For example, at node i, *sum-over* is {a, b}, first time *H0* is called, it returns (0, 0), next time returns (0, 1), and so on (suppose the domain of a and b is [0, 1]).

Complexity of *H* and *HSearch* in this case is constant O(1), thus, the worst case time complexity of *Query* for a single term is O(n).

3.5.2. H1: Bigger Item Implying Bigger Term

This heuristic *H1* will guide HSearch(B, H, T) routine to find the biggest uninstantiated item in the tree T, and guide *Instantiate* to instantiate the biggest uninstantiated item. More exactly, the heuristic function will try to find the biggest uninstantiated whole probability value in the root of tree T. If it can find one, *HSearch* finishes. Otherwise, the *HSearch* will recursively call heuristic function *H1* for T's both subtrees, and return the bigger one (see *Note1* in section 3.3)

The worst case time complexity of the heuristic function is $O(D^{Dim})$, and the worst case time complexity for the *HSearch* is $O(nD^{Dim})$, where D is the domain size of all distribution variables (suppose they have the same domain sizes), Dim is the Max Dim, and n is the number of nodes in tree T. Therefore, the worst case time complexity of *Query* for a single term is $O(D^{Dim}nlogn)$.

3.5.3. H2: Randomly Instantiate One of the Subtrees

From the analysis in the above section 3.5.2, we know that the worst case time complexity of *HSearch*, for such a simple heuristic H1, is very high. And in the algorithm *Query* and *Instantiate*, *HSearch* is called recursively. As noted in *Note 3* of figure 4 and figure 5, calling *HSearch* and the if-statement there are driven by the heuristics of "bigger item implying bigger term". Is this heuristic already fixed in the algorithm *Query* and *Instantiate*? Not really. We can get rid of it by letting the *HSearch* do nothing but randomly return 0 or 1, so that algorithm *Query* and *Instantiate* can instantiate the subtree randomly, and do less *HSearch*. In this case, H2 is the same as H1 when it is called directly by *Instantiate*, and is a 0 and 1 random generator when it is called directly by *HSearch*.

The worst case time complexity of the heuristic function is the same $O(D^{Dim})$, the complexity of *HSearch* in this case is constant O(1), the complexity of *Query* calling *Instantiate* is O(n), thus, the worst case time complexity of *Query* for a single term is $O(nD^{Dim})$.

3.6. Example

Suppose that John is taking a class. The probability of John study hard is 0.9, the probability of John being clever is 0.7. Suppose further that whether a student can get an A in the class depends on

whether s/he studies hard, and is clever (the condition probability is given in figure5). Query : what is the probability of John getting an A for the class?

The corresponding belief net and the evaluation tree are shown in figure 5 and figure 6, respectively.

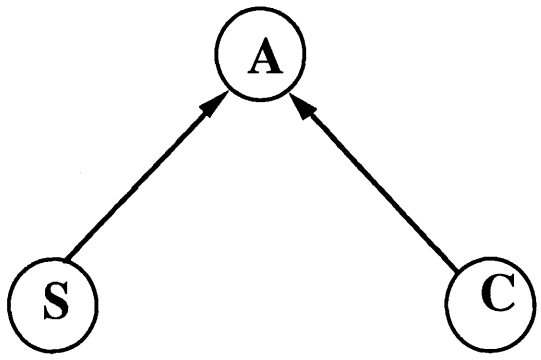


Figure 5: Belief Net of the example

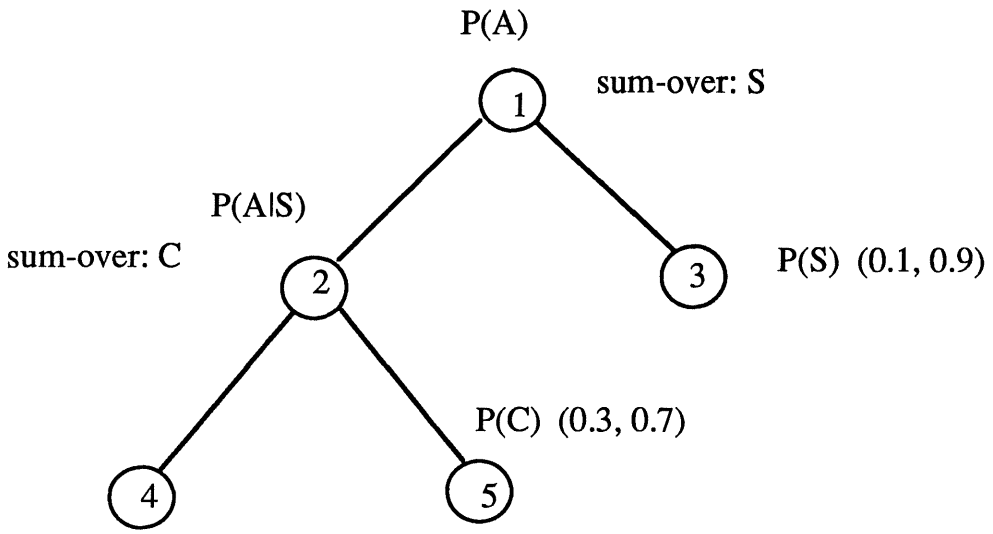


Figure 6: Evaluation tree of the example

3.6.1. *Query* Calling Scenario for *H 0*

```
Query(1, ((A 1)), H0, T1);
   no a whole value for P(A=1) cached at root of T1;
   (H 0) = HSearch(((A 1)), H0, T1);
   Instantiate(((A 1) (H 0)), H0, T2);
      (C 0) = HSearch(((A 1) (H 0)), H0, T2);
      (0.2 ((A 1)(H 0)(C 0))) = Instantiate(((A 1)(H 0)(C 0)), H0,T4);
      (0.3 ((C 0))) = instantiate(((A 1)(H 0)(C 0)), H0, T5);
      return (0.06 ((A 1)(H 0)(C 0)));
   (0.1 ((H 0))) = Query(0.06, ((A 1) (H 0)), H0, T3);
   return (0.006 ((A 1)(H 0)(C 0)));   {first term found}

   Query(0.1, ((A 1)(H 0)), H0, T2);
      no a whole value for P(A=1,H=0) cached at root of T2;
      (C 1) = HSearch(((A 1) (H 0)), H0, T2);
      (0.5 ((A 1)(H 0)(C 1))) = Instantiate(((A 1)(H 0)(C 1)), H0,T4);
      (0.7 ((C 1))) = Query(0.05, ((A 1)(H 0)(C 1)), H0, T5);
      return (0.035 ((A 1)(H 0)(C 1))); {second term found}

   still no a whole value for P(A=1) cached at root of T1;
   (H 1) = HSearch(((A 1)), H0, T1);
   Instantiate(((A 1) (H 1)), H0, T2);
      (C 0) = HSearch(((A 1) (H 1)), H0, T2);
      (0.5 ((A 1)(H 1)(C 0))) = Instantiate(((A 1)(H 1)(C 0)), H0,T4);
      (0.3 ((C 0))) = Instantiate(((A 1)(H 1)(C 0)), H0, T5);
      return (0.15 ((A 1)(H 1)(C 0)));
   (0.9 ((H 1)) = Query(0.15, ((A 1) (H 1)), H0, T3);
   return (0.135 ((A 1)(H 1)(C 0)));   {third term found}

   Query(0.9, ((A 1)(H 1)), H0, T2);
      no a whole value for P(A=1,H=1) cached at root of T2;
      (C 1) = HSearch(((A 1) (H 1)), H0, T2);
      (0.8 ((A 1)(H 0)(C 1))) = Instantiate(((A 1)(H 1)(C 1)), H0,T4);
      (0.7 ((C 1))) = Query(0.72, ((A 1)(H 1)(C 1)), H0, T5);
      return (0.504 ((A 1)(H 1)(C 1))); {fourth term found}

   finish;
```

3.6.2. *Query* Calling Scenario for *H 1*

Query(1, ((A 1)), H1, T_1);
no a whole value for P(A=1) cached at root of T_1;
HSearch(((A 1)), H1, T_2);
(0.8 ((A 1)(H 1)(C 1))) = HSearch(((A 1)), H1, T_4);
(0.7 ((C 1))) = HSearch(((A 1)), H1, T_5);
return (0.8 ((A 1)(H 1)(C 1))); {return the bigger one}
(0.9 ((H 1))) = HSearch(((A 1)), H1, T_3);
(0.9 ((H 1))) = Instantiate(((A 1) (H 1)), H1, T_3);
Query(0.9, ((A 1)(H 1)), H1, T_2);
no a whole value for P(A=1, H=1) cached at root of T_2;
(0.8 ((A 1)(H 1)(C 1)))= HSearch(((A 1) (H 1)), H1, T_4);
(0.7 ((C 1)))= HSearch(((A 1) (H 1)), H1, T_4);
(0.8 ((A 1)(H 1)(C 1))) = Instantiate(((A 1)(H 1)(C 1)), H1,T_4);
(0.7 ((C 1))) = Query(0.72, ((A 1)(H 1)(C 1)), H1, T_5);
return (0.504 ((A 1)(H 1)(C 1))); {first term found}

Query(0.9, ((A 1)(H 1)), H1, T_2);
no a whole value for P(A=1,H=1) cached at root of T_2;
HSearch(((A 1) (H 1)), H1, T_2);
(0.5 ((A 1)(H 1)(C 0)))=HSearch(((A 1)(H 1)), H1, T_4);
(0.3 ((C 0))) = HSearch(((A 1) (H 1)), H1, T_5);
return (0.5 ((A 1)(H 1)(C 0)))
(0.5 ((A 1)(H 1)(C 0))) = Instantiate(((A 1)(H 1)(C 1)), H1,T_4);
(0.3 ((C 0))) = Query(0.45, ((A 1)(H 1)(C 0)), H1, T_5);
return (0.135 ((A 1)(H 1)(C 0))); {second term found}

still no a whole value for P(A=1) cached at root of T_1;
HSearch(((A 1)), H1, T_2);
(0.5 ((A 1)(H 0)(C 1))) = HSearch(((A 1)), H1, T_4);
(0.7 ((C 1))) = HSearch(((A 1)), H1, T_5);
return (0.7 ((C 1)));
(0.1 ((H 0))) = HSearch(((A 1)), H1, T_3);
Instantiate(((A 1) (C 1)), H1, T_2);
(0.5 ((A 1)(H 0)(C 1))) = HSearch(((A 1)(C 1)), H1, T_4);
(0.7 ((C 1))) = HSearch(((A 1)(C 1)), H1, T_5);
(0.7 ((A 1)(C 1))) = Instantiate(((A 1)(C 1)), H1,T_5);
(0.5 ((A 1)(H 0)(C 1))) = Instantiate(((A 1)(C 1)), H1,T_4);
return (0.35 ((A 1)(H 0)(C 1)));
(0.1 ((H 0))) = Query(0.35, ((A 1) (H 0)), H1, T_3);
return (0.035 ((A 1)(H 0)(C 1))); {third term found}

Query(0.1, ((A 1)(H 0)), H1, T_2);
no a whole value for P(A=1,H=0) cached at root of T_2;
(0.2 ((A 1)(H 0)(C 0))) = HSearch(((A 1)(H 0)), H1, T_4);
(0.3 ((C 0))) = HSearch(((A 1)), H1, T_5);
(0.3 ((C 0))) = Instantiate(((C 0)), H1,T_4);
(0.2 ((C 1))) = Query(0.03, ((A 1)(H 0)(C 0)), H1, T_5);
return (0.006 ((A 1)(H 0(C 0))); {fourth term found}

finish;

3.6.3. *Query* Calling Scenario for *H 2*

Query(1, ((A 1)), H2, T_1);
no a whole value for P(A=1) cached at root of T_1;
Instantiate(((A 1)), H2, T_2);
(0.8 ((A 1)(H 1)(C 1)))=Instantiate(((A 1)), H2, T_4);
(0.7 ((C 1)))=Instantiate(((A 1)), H2, T_5);
return (0.56 ((A 1)(H1)(C 1)));
(0.9, ((H 1)))=Query(0.56, ((A 1)(H 1)(C 1)), H2, T_3);
return (0.504 ((A 1)(H 1)(C 1))); {first term found}

Query(0.9, ((A 1)(H 1)), H2, T_2);
no a whole value for P(A=1,H=1) cached at root of T_2;
(0.5 ((A 1)(H 1)(C 0))) = Instantiate(((A 1)(H 1)), H2,T_4);
(0.3 ((C 0))) = Query(0.45, ((A 1)(H 1)(C 0)), H2, T_5);
return (0.135 ((A 1)(H 1)(C 0))); {second term found}

still no a whole value for P(A=1) cached at root of T_1;
Instantiate(((A 1)), H2, T_2);
(0.5 ((A 1)(H 0)(C 1))) = Instantiate(((A 1)), H2,T_4);
(0.7 ((A 1)(C 1))) = Instantiate(((A 1)), H2,T_5);
return (0.35 ((A 1)(H 0)(C 1)));
(0.1 ((H 0))) = Query(0.35, ((A 1) (H 0)), H2, T_3);
return (0.035 ((A 1)(H 0)(C 1))); {third term found}

Query(0.1, ((A 1)(H 0)), H2, T_2);
no a whole value for P(A=1,H=0) cached at root of T_2;
(0.2 ((A 1)(H 0)(C 0))) = Instantiate(((A 1)(H 0)), H2, T_4);
(0.3 ((C 0))) = Query(0.02, ((A 1)(H 0)(C 0)), H2, T_5);
return (0.006 ((A 1)(H 0(C 0))); {fourth term found}

finish;

4. Empirical Results

The search heuristics described in the above section, and a best first search developed by D'Ambrosio [D'Ambrosio, 1991] are tested in nineteen belief nets for these experiments. One of those nets is the *intel* net currently used for circuit diagnosis. The others are generated by J. Suermondt's random net generator. For each net, I only choose those queries with large evaluation tree for these experiments, because we are only interested in the query with large size. Instead of comparing the run time among these algorithms, we compare the basic operations which dominate the algorithms, so that the performance comparison is independent of the implementation (Therefore, in this section, time means the number of times of the basic operation). This basic operation is the examination of the items (distribution values), which includes 1) the comparison: whether this item is a bigger one, and 2) validation: whether this item can be taken or not.

4.1. Data

Table 1: Experimental Network Characteristics

Net	Nodes	Arcs	Query	tree size	MAX DIM
1	23	28	var23	9	4
			var21	15	4
2	10	38	var10	19	9
			var9	17	8
3	11	26	var11	13	4
			var8	9	3
4	14	21	var14	11	5
			var12	19	5
			var10	9	5
5	10	15	var10	15	4
			var9	13	4
6	27	35	var27	11	5
			var26	11	3
			var24	13	4
7	12	26	var12	17	6
			var10	13	5
8	12	23	var10	13	5
			var9	13	6
9	14	15	var14	9	4
10	12	22	var12	15	5
11	20	42	var20	19	7
12	16	35	var16	11	4
			var13	13	6
13	12	20	var12	17	7
14	28	34	var28	15	4
			var27	23	5
15	25	30	var19	21	4
16	22	29	var22	17	4
			var20	31	5
17	27	41	var27	17	5
18	16	16	var16	13	5
intel	100	106	IHigh_225_fccd'sI	17	8
			Inom_224_cd'sI	61	8
			INormal_225m_fccd'sI	63	8
			7.5_<=_BVDP2_<=11.5	63	8
			ILow_225m_fccd'sI	17	8
			BVDN2>=15	63	8

Table 2: First term time (basic operations) and the mass value percentage to the exact value

Net	Query	First Term Ops				first Term mass %			
		A*	H0	H1	H2	A*	H0	H1	H2
1	var23	89	9	27	9	17.13	3.2	6.72	5.57
	var21	224	15	51	15	16.23	0.67	2.83	2.20
2	var10	4720	19	412	51	2.12	0.30	0.25	0.25
	var9	1730	17	255	17	2.95	0.38	0.54	0.34
3	var11	227	13	85	13	7.62	0.44	0.37	1.86
	var8	76	9	26	11	31.18	3.8	11.38	10.34
4	var14	168	11	52	11	6.57	2.57	2.75	2.75
	var12	372	19	72	21	6.67	0.04	2.84	0.45
	var10	64	9	25	9	22.05	3.78	20.89	11.7
5	var10	168	15	69	32	12.25	0.05	2.78	0.75
	var9	148	13	57	36	10.69	0.16	3.50	3.57
6	var27	251	11	33	12	17.82	1.65	8.36	4.16
	var26	107	11	35	16	21.83	4.09	3.81	2.94
	var24	120	13	51	17	10.76	1.17	2.22	1.41
7	var12	268	17	74	24	8.42	0.08	0.84	0.84
	var10	128	13	43	14	10.30	0.53	7.69	7.69
8	var10	268	13	55	13	5.88	0.63	2.93	1.36
	var9	818	13	52	21	7.12	1.82	3.75	2.77
9	var14	128	9	26	9	14.13	9.86	7.05	7.05
10	var12	361	15	69	18	13.05	0.87	2.31	0.25
11	var20	283	19	87	19	4.57	0.82	1.77	1.77
12	var16	133	11	32	17	17.12	5.53	3.84	8.50
	var13	178	13	68	20	16.06	1.86	2.24	2.24
13	var12	559	17	76	63	4.88	0.08	0.67	0.35
14	var28	185	15	50	15	14.22	0.28	4.38	2.23
	var27	470	23	106	24	4.25	0.28	0.49	0.07
15	var19	325	21	93	30	2.68	0.14	0.34	0.12
16	var22	497	17	54	19	11.09	0.16	1.71	0.77
	var20	932	31	136	52	0.82	0.003	0.01	0.02
17	var27	182	17	74	17	7.44	0.17	0.83	0.83
18	var16	120	13	52	14	6.37	2.80	2.77	1.53
intel	!High_225_fccd's!	*	17	61	17	*	0.78	2.96	2.96
	!nom_224_cd's!	*	61	276	69	*	1.19d-6	6.30d-8	1.17d-6
	!Normal_225m_fccd's!	*	63	316	290	*	5.96d-8	1.87d-7	1.16d-6
	7.5_<=_BVDP2_<=11.5	*	63	316	71	*	5.96d-8	3.91d-7	1.15d-6
	!Low_225m_fccd's!	*	17	61	17	*	0.82	3.27	3.27
	BVDN2>=15	*	63	316	71	*	5.96d-8	3.91d-7	1.15d-6

Note: The * in the table denotes an unknown value since the algorithm could not finish running those test cases in ten hours

Table 3: First term H mass value relative error to the A* mass value relative error = (mass(A*) - mass(H))/ mass(A*)

Net	Query	tree size	MAX DIM	mass relative error		
				H0	H1	H2
1	var23	9	4	0.81	0.61	0.67
	var21	15	4	0.96	0.83	0.86
2	var10	19	9	0.86	0.88	0.88
	var9	17	8	0.87	0.82	0.88
3	var11	13	4	0.84	0.95	0.75
	var8	9	3	0.88	0.63	0.67
4	var14	11	5	0.61	0.58	0.58
	var12	19	5	0.99	0.57	0.93
	var10	9	5	0.83	0.05	0.47
5	var10	15	4	0.996	0.77	0.94
	var9	13	4	0.99	0.67	0.67
6	var27	11	5	0.91	0.53	0.77
	var26	11	3	0.81	0.83	0.87
	var24	13	4	0.89	0.79	0.87
7	var12	17	6	0.99	0.90	0.90
	var10	13	5	0.95	0.25	0.25
8	var10	13	5	0.89	0.50	0.77
	var9	13	6	0.74	0.47	0.61
9	var14	9	4	0.30	0.5	0.5
10	var12	15	5	0.93	0.82	0.98
11	var20	19	7	0.82	0.61	0.61
12	var16	11	4	0.68	0.78	0.50
	var13	13	6	0.88	0.86	0.86
13	var12	17	7	0.98	0.86	0.93
14	var28	15	4	0.98	0.69	0.84
	var27	23	5	0.93	0.88	0.98
15	var19	21	4	0.95	0.87	0.96
16	var22	17	4	0.98	0.85	0.93
	var20	31	5	0.996	0.98	0.98
17	var27	17	5	0.98	0.89	0.89
18	var16	13	5	0.56	0.57	0.76

Table 4: mass values in the same amount of time of A* algorithm for the first and second terms

Net	Query	A*1st term ops	mass %				A*2nd term ops	mass %			
			A*	H0	H1	H2		A*	H0	H1	H2
1	var23	89	17.13	100.0	27.00	86.92	136	32.57	100.0	56.24	100.0
	var21	224	16.23	100.0	66.00	100.0	253	27.35	100.0	76.00	100.0
2	var10	4720	2.12	100.0	1.33	11.55	5829	2.82	100.0	2.09	13.50
	var9	1730	2.95	100.0	2.43	16.36	2106	4.29	100.0	2.75	20.60
3	var11	227	7.62	100.0	1.35	41.57	257	14.97	100.0	1.50	53.95
	var8	76	31.18	100.0	41.00	92.00	110	36.03	100.0	47.00	100.0
4	var14	168	6.57	100.0	11.60	76.00	217	30.54	100.0	13.50	95.40
	var12	372	6.67	100.0	12.12	34.70	381	10.10	100.0	12.17	38.40
	var10	64	22.05	40.00	56.00	21.00	85	31.62	100.0	62.00	42.60
5	var10	168	12.25	100.0	13.50	9.00	189	17.43	100.0	15.64	14.10
	var9	148	10.69	100.0	8.20	14.30	174	19.59	100.0	9.64	16.90
6	var27	251	17.82	100.0	61.30	93.00	283	25.79	100.0	61.90	96.50
	var26	107	21.83	100.0	79.11	100.0	181	44.67	100.0	100.0	100.0
	var24	120	10.76	100.0	22.00	70.00	227	17.93	100.0	69.27	100.0
7	var12	268	8.42	83.88	11.00	42.32	300	10.22	100.0	12.60	43.82
	var10	128	10.30	100.0	37.72	56.52	141	15.92	100.0	41.00	60.00
8	var10	268	5.88	89.00	6.50	38.56	353	8.89	100.0	10.00	50.00
	var9	818	7.12	100.0	52.90	100.0	1633	13.25	100.0	88.83	100.0
9	var14	128	14.13	100.0	65.00	100.0	171	21.36	100.0	100.0	100.0
10	var12	361	13.05	100.0	7.42	21.32	413	13.05	100.0	9.60	25.11
11	var20	283	4.57	82.25	19.76	35.20	412	7.92	100.0	34.11	47.00
12	var16	133	17.12	100.0	77.93	97.27	177	31.72	100.0	86.00	100.0
	var13	178	16.06	66.98	6.00	11.40	210	23.96	76.56	8.00	13.81
13	var12	559	4.88	72.62	14.80	11.50	645	7.11	88.23	16.21	13.51
14	var28	185	14.22	100.0	29.00	100.0	269	25.22	100.0	46.86	100.0
	var27	470	4.25	100.0	8.00	7.06	495	8.46	100.0	12.00	72.00
15	var19	325	2.68	100.0	11.34	100.0	357	4.92	100.0	13.10	100.0
16	var22	497	11.09	100.0	100.0	100.0	526	15.40	100.0	100.0	100.0
	var20	932	0.82	100.0	12.70	100.0	1073	1.19	100.0	16.98	100.0
17	var27	182	7.44	100.0	3.00	15.75	242	10.49	100.0	8.16	50.00
18	var16	120	6.37	100.0	22.85	48.60	129	8.87	100.0	29.00	49.00

Table 5: mass values in the same amount of time of H1 heuristic for the first and second terms

Net	Query	H1 1st temr ops	mass %			H1 2nd term ops	mass %		
			H0	H1	H2		H0	H1	H2
1	var23	27	41.67	6.72	15.00	44	66.87	12.48	25.54
	var21	51	17.34	2.83	10.64	69	48.53	5.03	12.30
2	var10	412	6.20	0.25	1.03	1023	21.48	0.29	2.40
	var9	255	6.81	0.54	1.62	539	28.41	0.73	4.70
3	var11	85	8.54	0.37	7.12	124	28.32	0.58	10.03
	var8	26	11.18	11.38	32.36	41	48.42	21.73	42.00
4	var14	52	29.84	2.75	17.32	68	62.63	4.53	24.79
	var12	72	1.18	2.84	2.04	107	13.09	3.57	4.92
	var10	25	10.70	20.89	19.22	40	34.07	39.02	19.30
5	var10	69	2.56	2.78	4.89	100	7.98	8.09	8.01
	var9	57	3.02	3.50	3.21	88	10.77	4.97	5.25
6	var27	33	12.46	8.36	28.94	41	24.56	12.51	30.00
	var26	35	58.41	3.81	29.93	41	90.00	6.10	42.99
	var24	51	24.70	2.22	21.06	62	40.41	5.89	24.25
7	var12	74	4.44	0.84	5.50	94	7.23	1.40	9.00
	var10	43	2.29	7.69	26.53	51	2.48	11.32	30.00
8	var10	55	7.09	2.93	6.14	113	27.72	3.47	17.64
	var9	52	9.76	3.75	7.46	64	12.45	7.24	8.26
9	var14	26	31.54	7.05	35.56	38	55.55	22.48	48.71
10	var12	69	8.81	2.31	3.66	83	17.03	2.88	4.12
11	var20	87	24.00	1.77	18.36	100	26.26	2.84	19.75
12	var16	32	34.34	3.84	30.92	38	43.87	5.52	38.51
	var13	68	16.80	2.24	5.00	89	19.98	4.03	7.50
13	var12	76	2.52	0.67	0.74	84	3.54	1.93	1.00
14	var28	50	19.07	4.38	13.89	58	44.81	5.94	14.83
	var27	106	3.62	0.49	0.26	121	11.00	0.59	0.32
15	var19	93	6.21	0.34	7.15	100	6.41	0.79	7.62
16	var22	54	8.84	1.71	1.46	63	10.45	2.93	2.60
	var20	136	3.46	0.01	0.04	157	3.79	0.11	0.07
17	var27	74	6.38	0.83	5.48	80	6.85	1.46	6.94
18	var16	52	41.33	2.77	22.83	58	47.37	3.66	25.00
intel	!High_225_fccd's!	61	2.72	2.96	2.96	205	6.72	3.70	4.00
	!nom_224_cd's!	276	4.71d-6	6.30d-8	1.17d-5	408	7.44d-6	1.10d-7	1.64d-5
	!Normal_225m_fccd's!	316	2.94d-7	1.87d-7	1.16d-6	506	4.28d-7	4.28d-7	1.05d-5
	7.5_<=_BVDP2_<=11.5	316	2.94d-7	3.91d-7	1.38d-5	506	6.85d-7	4.28d-7	2.07d-5
	!Low_225m_fccd's!	61	2.54	3.27	3.27	225	5.67	3.42	4.12
	BVDN2>=15	316	2.94d-7	3.91d-7	1.27d-7	506	6.85d-7	4.28d-7	1.95d-5

Table 6: mass values in the same amount of time of H2 heuristic for the first and second terms.

Net	Query	H2 1st term ops	mass %		H2 2nd term ops	mass %	
			H0	H2		H0	H2
1	var23	9	3.20	5.57	14	6.95	12.48
	var21	15	0.97	2.20	21	2.00	5.03
2	var10	51	0.63	0.25	108	1.29	0.67
	var9	17	0.38	0.34	33	0.71	0.74
3	var11	13	0.44	1.86	41	2.55	2.31
	var8	11	6.26	10.34	18	8.61	21.73
4	var14	11	2.57	2.75	17	6.04	4.53
	var12	21	0.04	0.45	33	0.40	1.25
	var10	9	3.78	11.7	15	6.70	19.22
5	var10	32	0.35	0.75	51	0.58	1.05
	var9	21	0.57	0.86	36	1.52	1.92
6	var27	12	1.65	4.16	20	8.72	9.62
	var26	16	11.84	2.94	21	14.02	8.62
	var24	17	1.72	1.41	25	3.18	3.22
7	var12	24	0.48	0.84	36	1.58	1.34
	var10	14	0.53	7.69	19	4.54	11.32
8	var10	13	0.62	1.36	28	2.59	1.97
	var9	21	2.79	2.77	32	3.92	5.33
9	var14	9	9.86	7.05	14	14.59	22.48
10	var12	18	1.08	0.25	26	1.99	1.32
11	var20	19	0.82	1.77	24	1.04	2.40
12	var16	17	24.45	8.50	22	35.34	13.46
	var13	20	7.92	2.24	38	13.30	4.03
13	var12	63	2.03	0.35	71	2.37	0.74
14	var28	15	0.28	2.23	19	2.34	8.20
	var27	24	0.001	0.07	36	0.02	0.09
15	var19	30	0.47	0.12	41	1.01	0.20
16	var22	19	0.16	0.77	26	0.32	0.77
	var20	52	0.01	0.015	62	0.017	0.017
17	var27	17	0.17	0.83	20	0.39	1.46
18	var16	14	2.80	1.53	19	3.70	1.93
intel	!High_225_fccd's!	17	0.87	2.96	152	5.52	3.70
	!nom_224_cd's!	69	1.19d-6	1.17d-6	88	1.60d-6	2.35d-6
	!Normal_225m_fccd's!	290	2.43d-7	1.16d-6	338	3.11d-7	2.32d-6
	7.5_<=_BVDP2_<=11.5	71	6.26d-8	1.15d-6	109	1.23d-7	3.45d-6
	!Low_225m_fccd's!	17	0.78	3.27	172	4.79	3.41
	BVDN2>=15	71	6.26d-8	1.15d-6	90	7.99d-8	2.30d-6

4.2 Discussion

Table 2 shows the time (basic operations) and the mass value percentage for finding the first term of the queries. The * in the table denotes an unknown value since the algorithm could not finish running that test cases in ten hours. Notice that

$mass(H1) \geq mass(H2) \geq mass(H0)$

in most cases (of course, mass(A*) is always the biggest), but

$time(A^*) \gg time(H1) > time(H2) \geq time(H0)$

is always true.

we draw six charts from table 2. They present how the mass values or time changes for different evaluation tree sizes when the Max Dim is fixed, or for different Max Dim when the evaluation tree size is fixed. These six charts are all put in the Appendix.

Chart1 and chart2 show that when the Max Dim is fixed, time(H0) is linear with the evaluation tree size, time(H2) is the same as or just a little bit longer than time(H0), time(H1) is almost linear with the size of the evaluation tree, and time(A*) increases with the tree size dramatically. These are consistent with the time complexity analysis in section 3, which are O(n), $O(D^{Dim}n\log n)$, and $O(nD^{Dim})$, for H0, H1, and H2, respectively.

Chart3 shows that when the evaluation tree size is fixed, time(H0) is a constant, time(H2) is the same as or just a little bit longer than time(H0), time(H1) increases with the Max Dim fast, but not to the extend of exponent. Time(A*) seems to increases with the Max Dim exponentially. These are also consistent with the time complexity analysis in section 3, but the results in table 2 and chart3 show that the first term time for H1, and H2 are much smaller than the worst case time in general.

Chart4, chart5, and chart6 show that the mass values become smaller, when the problem size increases. This is as expected: the larger the numbers of items smaller than one, the smaller the product of all these items. From these three charts, we can see that

the ratio of mass(H) to mass(A*) is independent of the problem size (that is the tree size and the Max Dim).

Table 3 shows clearly that the relative error

$$(\text{mass}(A^*) - \text{mass}(H)) / \text{mass}(A^*)$$

is independent of the problem size. This relative error depends on the distributions of the original belief net, and the heuristics.

As showed in table 2, for any net, the bigger the first term mass value, the larger the cost. What is the trade-off between the mass value and the time? Table 4 shows the mass values in the same amount of time of A* algorithm for the first and second terms. We see that,

$$\text{mass}(H0) > \text{mass}(H2) > \text{mass}(H1) > \text{mass}(A^*)$$

in most of the test cases. This implies that the search is too expensive. Actually, for the A* algorithm, searching for the largest term sometimes takes time longer than the time for exact value computation, and also tends to be memory bound (as those queries in table 2 for intel net).

Table 5 shows the mass values in the same amount of time of H1 heuristic for the first and second terms. We see that, H1 is still the worst, H2's performance is improved, but still not better than H0.

Table 6 shows the mass values in the same amount of time of H2 heuristic for the first and second terms. In these shorter amount of time, H2 performs better than H0.

5. Future Research

The heuristic function in the algorithm is implemented as an input parameter, we can try more heuristic functions for various belief networks to see which heuristic is good for which kinds of belief nets, and how the performance changes in terms of different net-scaling.

6. Summary

I have presented a new search approach to compute approximate answers for the probability query in belief nets. This approach can compute the 'best' bounds for a query in a period of any given time (if time permitted, it will get an exact value). Three heuristics for this search approach and a best first search approach were tested in a set of randomly generated belief nets and a net from the real world. I also discussed the trade-off among the heuristics and the best first search approaches. The heuristic search approximation has a better performance than the best first search in these test cases. Among the three heuristics, H0 and H2 are better than H1 for any given time limitation in most of the test cases. If the given time is very short, H2 is better than H0 in most cases, but, with time increasing, H0 looks better.

Since probabilistic inference in belief networks is computationally hard, we believe no one algorithm will be able to perform optimally in every situation (e.g. time, constraints, accuracy goals, network topology). Instead, specialized algorithms are needed to match situations, and intelligent meta-level control mechanisms are needed to match situations with algorithms.

7. Reference

[Chin, H. L. and G. F. Cooper, 1987]. Stochastic Simulation of Bayesian Belief Networks, in Proceedings of the Third Workshop on Uncertainty in Artificial Intelligence, Seattle, Washington, 106-113.

[Cooper, G. F. 1987] Probabilistic Inference Using Belief Networks is NP-Hard. Report KSL-87-27, Medical Computer Science Group, Stanford University.

[D'Ambrosio, B. 1989] Symbolic Probabilistic Inference in Belief Networks. OSU technical report. December 1989.

[D'Ambrosio, B. and Shachter, R. 1990] Symbolic Probabilistic Inference. Draft. Sept, 1990.

[D'Ambrosio, B. and Li, Z. 1991] Complexity of Probabilistic Inference in Belief Nets -- An Experimental Study.

[D'Ambrosio, B. 1991] Term Computation System. In preparation.

[Henrion, M. 1988] Propagations of uncertainty by probabilistic logic sampling in Bayes' networks. In Uncertain in Artificial Intelligence, Vol 2, J. Lemmer & L.N. Kanal (Eds.), North-Holland, Amsterdam. pp149-164.

[Henrion, M. 1991] Search-based methods to bound diagnostic probabilities in very large belief nets. 1991 Conference on Uncertainty and AI.

[Li, Zhaoyu 1990] Complexity of Probabilistic inference in Belief Nets - an experimental Study. MS thesis, OSU. November 1990.

[Neopolitan, R. 1990] Probabilistic Reasoning in Expert Systems: Theory and Algorithms. New York: Wiley, page 162, 1990.

[Pearl, J. 1987] Evidential Reasoning Using Stochastic Simulation of Causal Models. Artificial Intelligence, 33:131.

[Pearl, J. 1987] Addendum: Evidential Reasoning Using Stochastic Simulation of Causal Models. Artificial Intelligence, 32:245-257.

[Pearl, 1988] Judea Pearl. Probabilistic Reasoning in Intelligent Systems. Morgan Kaufmann, San Mateo, 1988.

[Shachter, 1986] R. D. Shachter. Evaluating Influence Diagrams. In Operations Researcch, 34(6): 871-882.

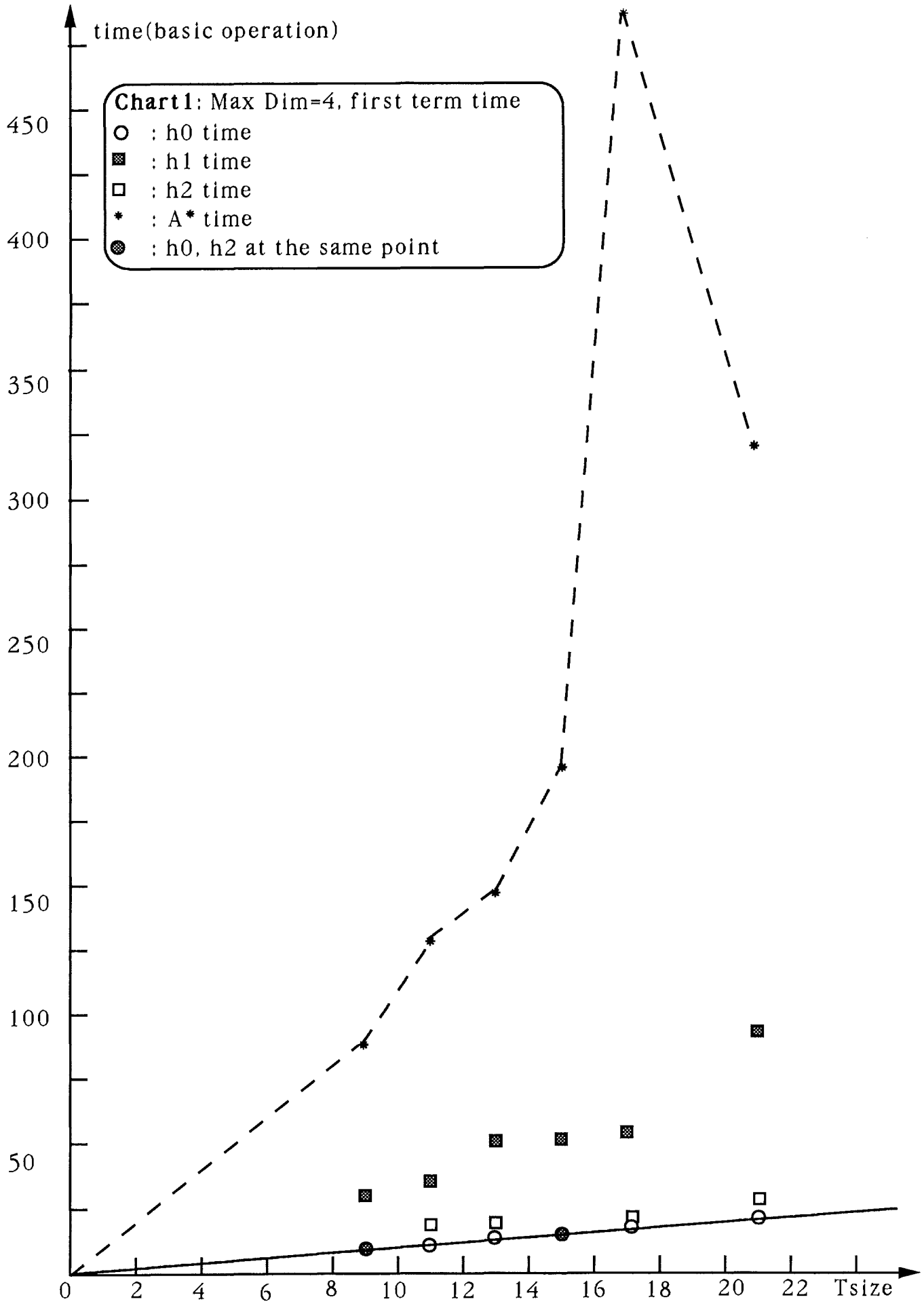
[Shachter, 1986] R. D. Shachter. Evidence Absorption and Propagation Through Evidence reversals. Proceedings of the Fifth Workshop on Uncertainty in AI, pages 303-310, 1989

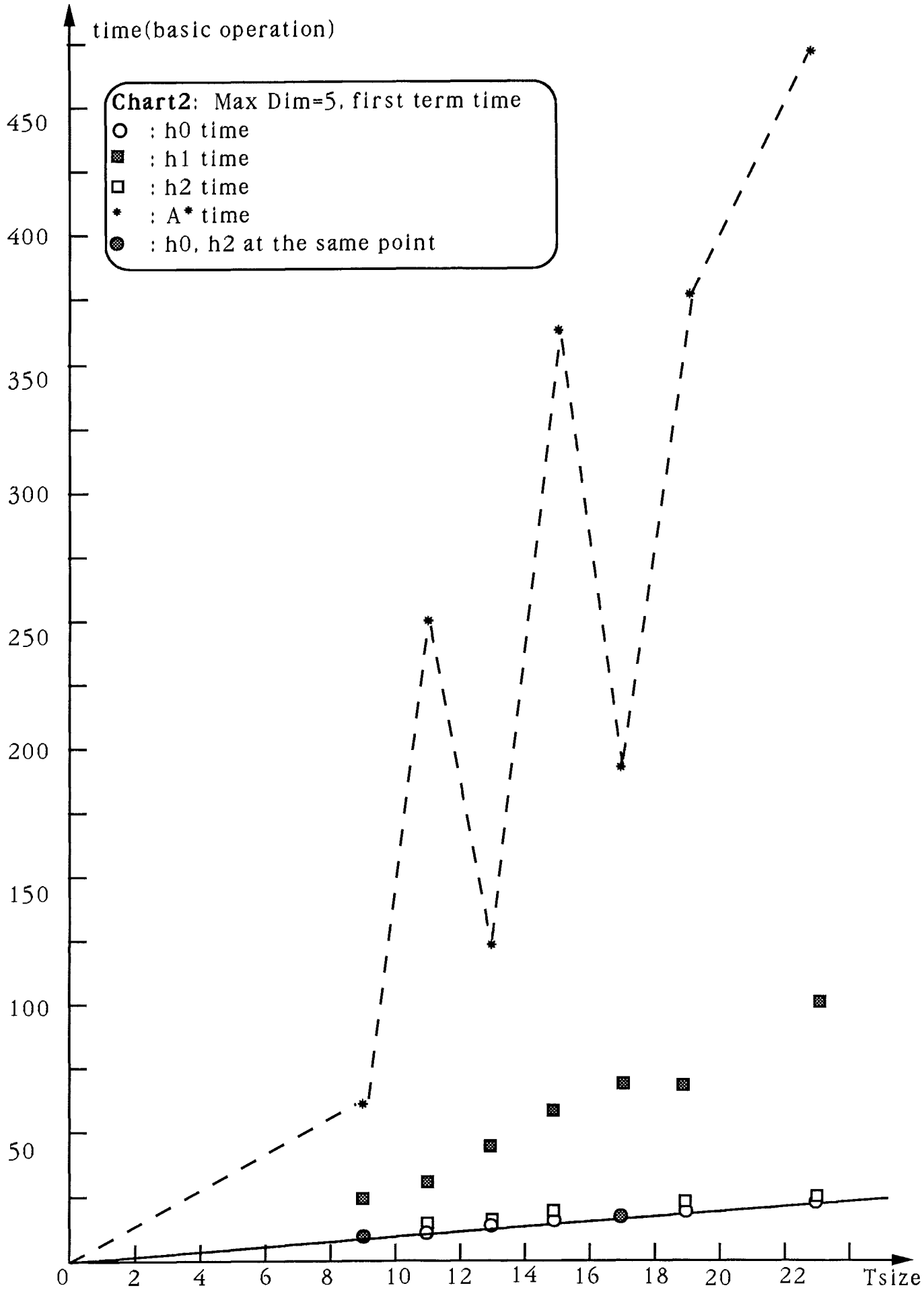
[Shacher R.D. and Peot. M., 1989]. Simulation Approached to Probabilistic Inference for General Probabilistic Inference on Belief Networks.

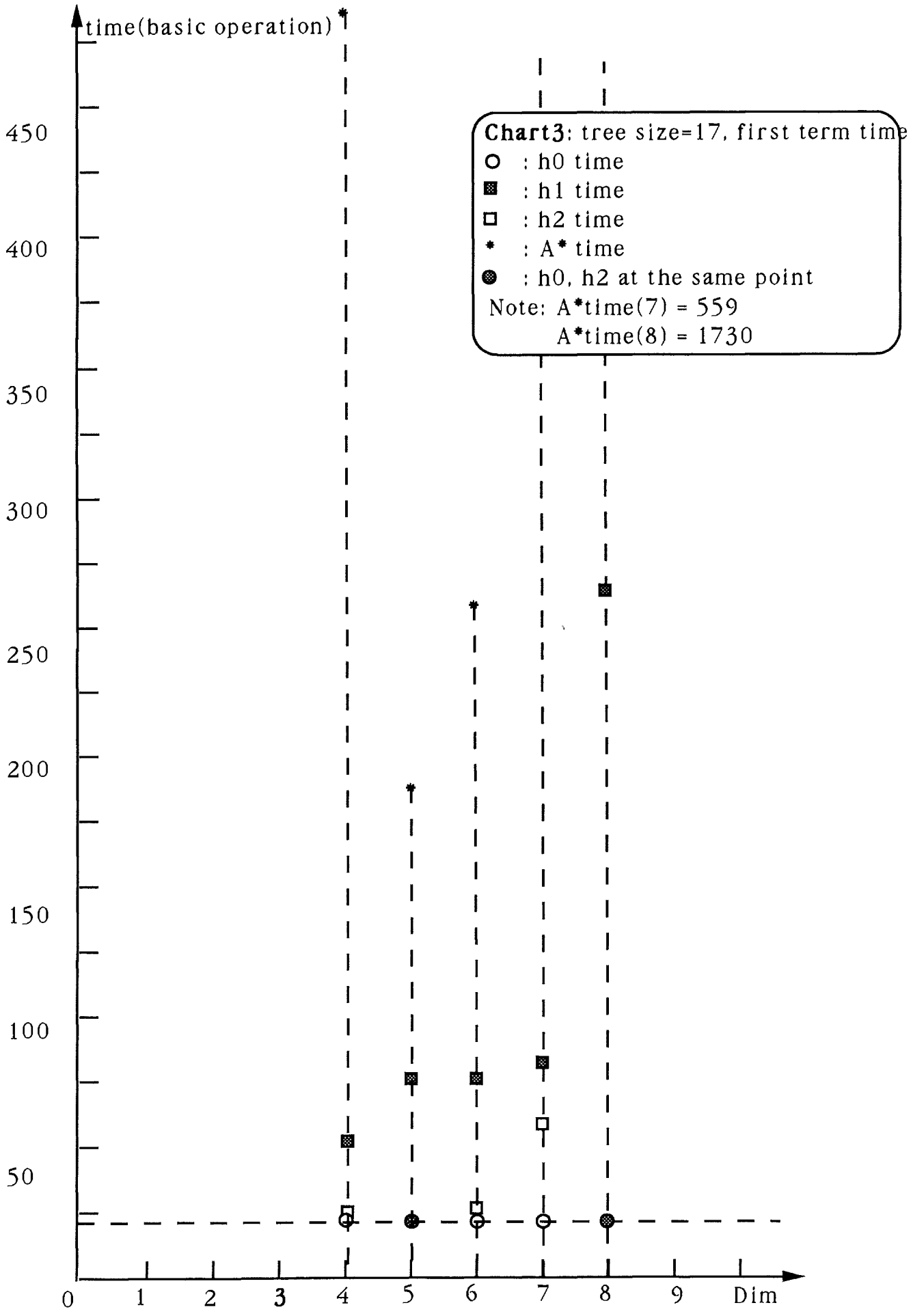
[Shimony, S. E. and Charniak, E. 1990] A new algorithm for finding MAP assignments to belief networks, in Proc of Sixth Conference on Uncertainty in AI, Cambridge, Ma., p98-103

8. Appendix

Chart1 to Chart6, drawn from the data in table2, are presented in the following pages.


time(basic operation)
Chart1: Max Dim=4, first term time
: h0 time
: h1 time
: h2 time
: A* time
: h0, h2 at the same point
450
400
350
300
250
200
150
100
50
0
2
4
6
8
10
12
14
16
18
20
22
Tsize


time(basic operation)
Chart2: Max Dim=5, first term time
: h0 time
: h1 time
: h2 time
: A* time
: h0, h2 at the same point
450
400
350
300
250
200
150
100
50
0
2
4
6
8
10
12
14
16
18
20
22
Tsize


time(basic operation)
450
400
350
300
250
200
150
100
50
0
1
2
3
4
5
6
7
8
9
Dim
Chart3: tree size=17, first term time
: h0 time
: h1 time
: h2 time
: A* time
: h0, h2 at the same point
Note: A*time(7) = 559
A*time(8) = 1730

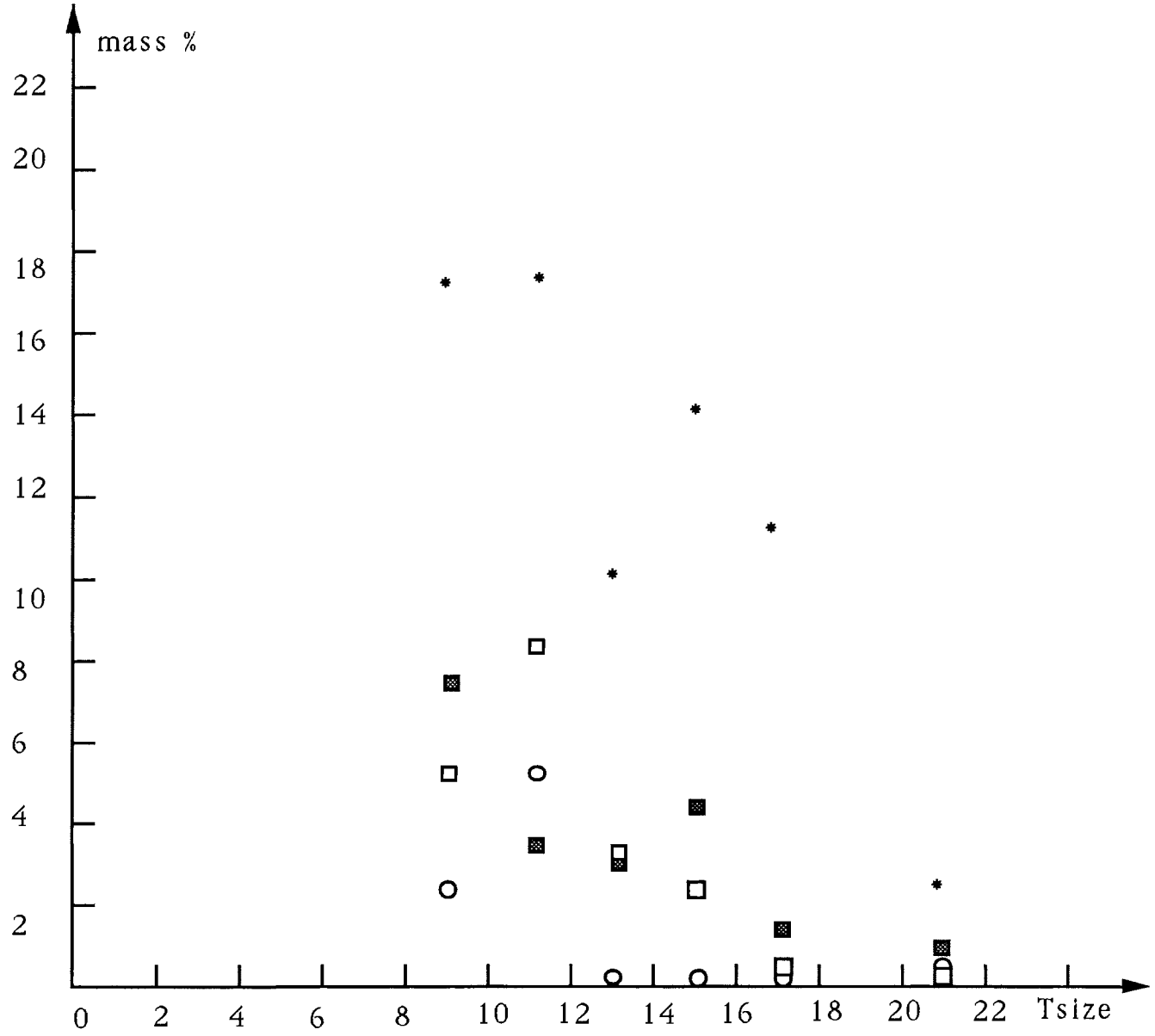


Chart4: Max Dim = 4, first term mass value
time: the number of basic operations
Tsize: the size of the evaluation tree

○ : h0 value
▩ : h1 value
□ : h2 value
* : A* value

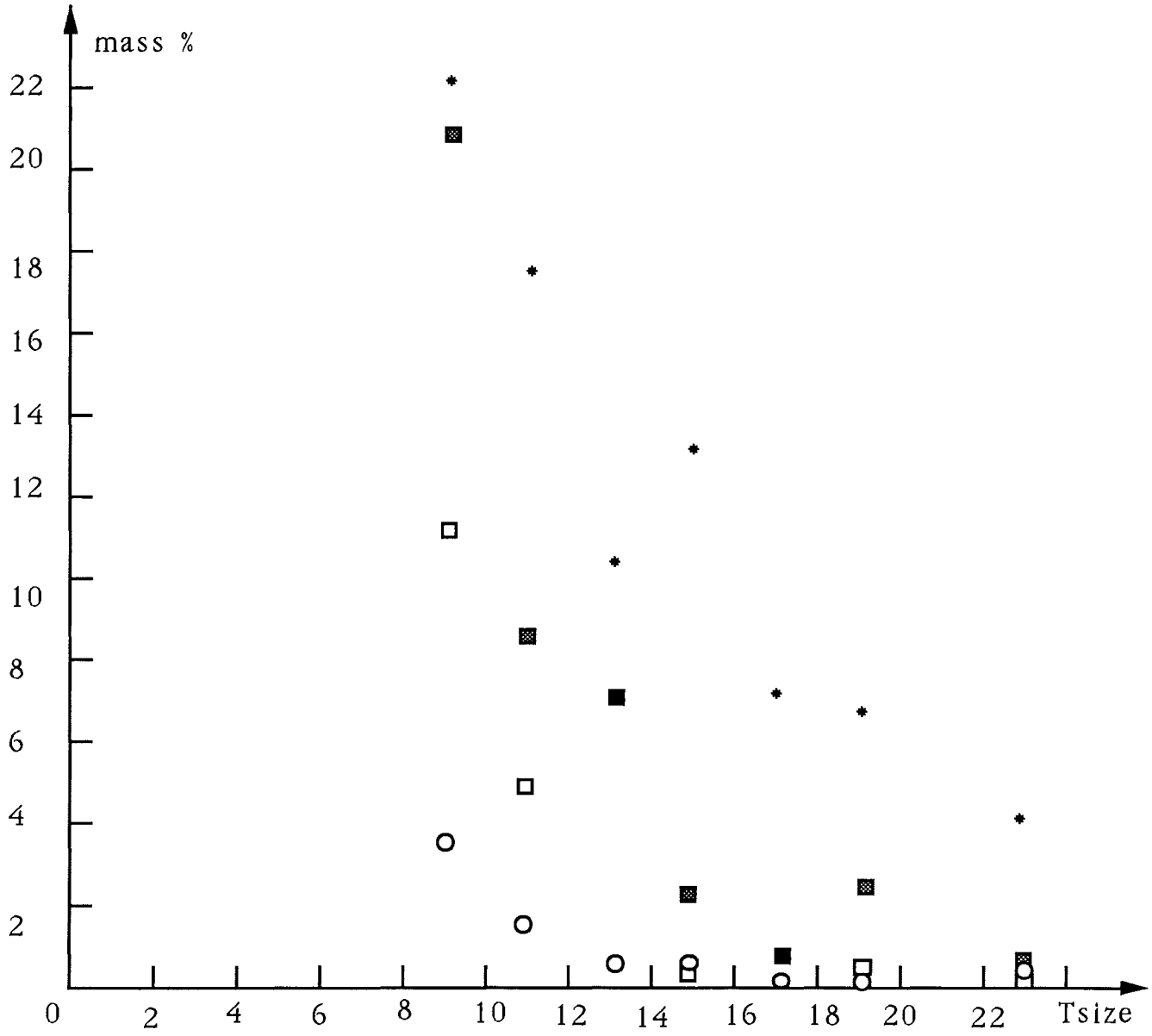


Chart5: Max Dim = 5, first term mass value
time: the number of basic operations
Tsize: the size of the evaluation tree

○ : h0 value
▩ : h1 value
□ : h2 value
■ : h1, h2 values at the same point
* : A* value

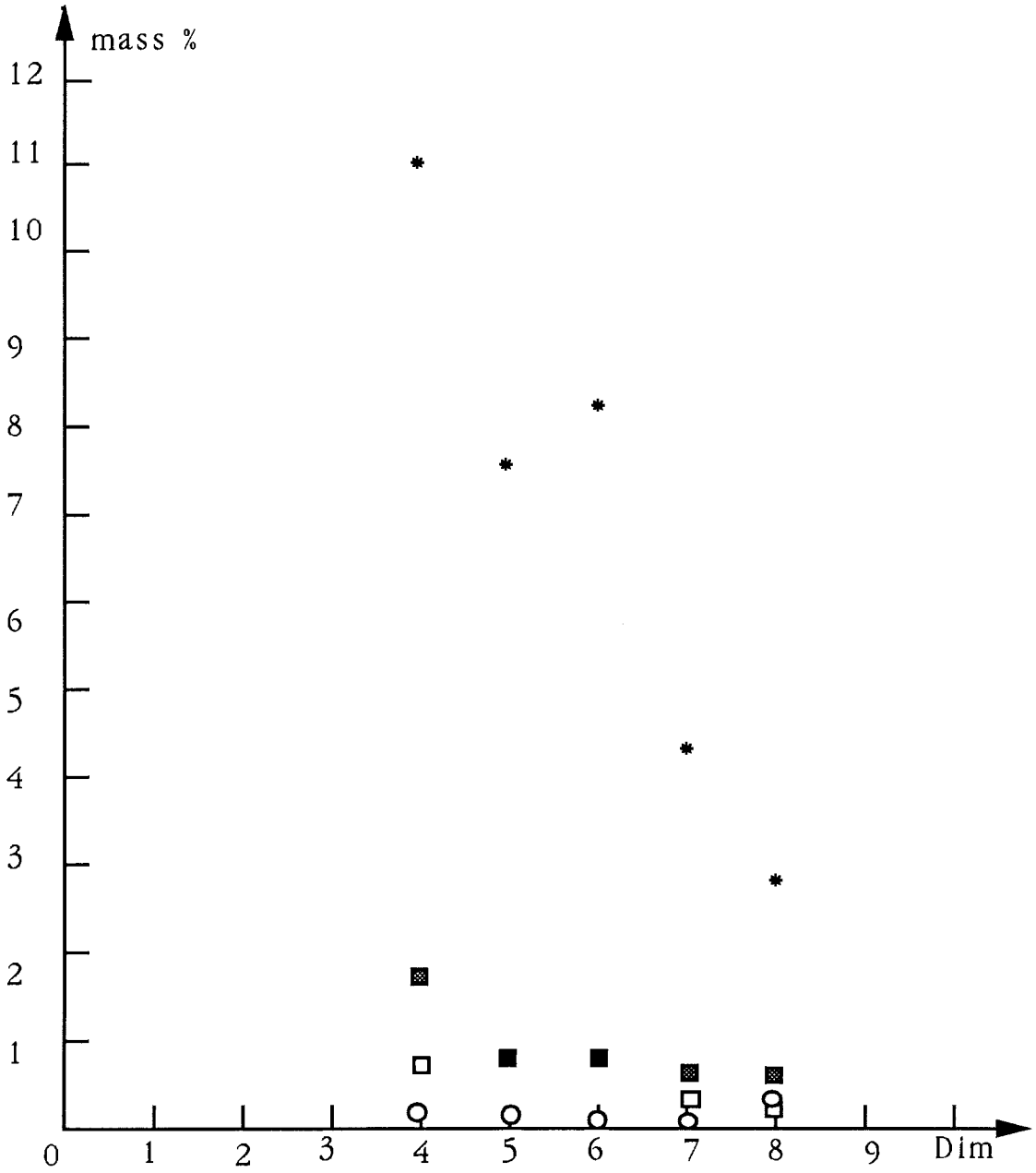


Chart6: tree size = 17, first term mass value
time: the number of basic operations
Tsize: the size of the evaluation tree

- o : h0 value
- ▩ : h1 value
- □ : h2 value
- ■ : h1, h2 values at the same point
- * : A* value