Chapter 4

TREE AND REASON MAINTENANCE

4.1 Introduction

The significance of prime implicants and the need for efficient technique for computing prime implicants have been discussed in the previous chapters. The algorithm PIAP as proposed in Chapter 3 employs a divide-and-conquer technique which naturally gives rise to a tree structure for representation. In this chapter, this concept is extended further to utilize the tree structure not only for computation of prime implicants but also to have full-fledged RMS. It is shown here that a binary tree is suitable to represent prepositional formula in CNF and also to represent the prime paths as a result of knowledge compilation. The implementation of PIAP makes use of the proposed tree representation.

In the later part of this chapter, the implementation details of the tree structure and the PIAP algorithm are discussed. The computational experiments carried out for PIAP are also reported in this chapter. It is shown here that the computing time as well as the number of subsumptions required by PIAP is much less than that of Socher's algorithm. The experimental results substantiate the theoretical arguments given in Chapter 3. Based on the discussions in Chapter 3 as well as the experiments reported in this chapter, it can be seen that the proposed algorithm (PIAP) is better than the Socher's algorithm for computation of prime implicants. Moreover, it is shown that the structure used for the algorithm can be extended to have a full-fledged RMS. Updating the data base of the RMS as and when the reasoner transmits clause(s) is very
important. Different methods to accomplish this are discussed in Section 4.6. Due to the
dynamic nature of the RMS, updating the prime paths incrementally is essential. The
same structure is suitable for incremental computation of prime paths as well as other
reasoning paradigms such as hypothetical reasoning.

4.2 Tree Representation of a Formula

A binary tree representation of a formula suitable for satisfiability problem is discussed
in [Horowitz 83, Frost 86]. In this representation, each node denotes an operator V, A
or, \( \rightarrow \) and the leaf nodes represent literals. The formula is obtained by traversing the
tree. The value of the formula represented by the tree can be computed by assigning true
or false to the variables present in the formula. To evaluate a formula, traverse the tree
in postorder evaluating subtrees until the entire expression is reduced to a single value.
However, in the present context, keeping in view the algorithm PIAP, this representation
scheme is not suitable. In the present work, a new tree representation scheme is developed
to represent a prepositional formula. Such a scheme naturally evolves from the algorithm
PIAP.

Any set of clauses representing a formula \( \mathcal{F} \) can be divided, with respect to a literal \( r \),
into two subsets of clauses, namely, a set of clauses containing \( r \) and another set of clauses
void of \( r \). If \( \mathcal{F} = \{D_1, D_2, \ldots, D_m\} \), let \( \mathcal{F}_r = \{D - \{r\} \mid r \in D, D \in \mathcal{F}\} \). Obviously,
\( \mathcal{F}_r \) is the set of clauses obtained by uniformly deleting the literal \( r \) from the clauses in \( \mathcal{F} \)
containing \( r \). Thus \( \mathcal{F}_r \) represents a subformula of \( \mathcal{F} \). Though \( \mathcal{F}_r \) does not contain any
clause as it is in \( \mathcal{F} \), we consider \( \mathcal{F}_r \) a subset of \( \mathcal{F} \). Similarly, \( \mathcal{F}_r = \{D \mid r \notin D, D \in \mathcal{F}\} \)
is another subset of \( \mathcal{F} \) which does not contain \( r \). Thus \( \mathcal{F}_r \) and \( \mathcal{F}_r \) are two subsets of \( \mathcal{F} \).
with empty intersection.

Each of these two sets of clauses can be further divided in a similar manner. The
literal with respect to which these two sets are partitioned need not be the same. That is, the set \( F_r \) can be divided with respect to some literal (say, \( r' \)) whereas \( F'_r \) can be divided with respect to some other literal (say, \( r'' \)). This branching process can be continued till the formula gets reduced to a single literal.

Thus, any set \( F \) of clauses is divided into two subsets \( F_r \) and \( F'_r \) of clauses, with respect to some literal \( r \) and hence can be visualized as a binary tree. Any node of this binary tree corresponds to a subset of \( F \), the root obviously corresponds to \( F \) itself. The literal \( r \) with respect to which the set \( F \) is divided to obtain two subsets, is associated with the node representing \( F \). This literal associated with every node is called the label of the node. The label of the nodes at any level of the tree need not be the same. In the binary tree representation of the formula \( F \), if \( r \) is the label of the node corresponding to the formula \( F \), then its left child corresponds to the formula \( F'_r \) whereas the right child corresponds to \( F_r \). For example, let the formula be given as \( F = \{abcde, abedg, abcf, abedg, abceg, abdf, abf, acf, acgeg\} \). The root corresponds to the formula \( F \). The literal \( a \) is chosen and the formula is split with respect to \( a \), giving \( F_a = \{acfe, acgeg\} \) represented by the left child and \( F'_a = \{bcde, bdg, bcf, bcedf, bcdf, bcdg, bceg, ddf, bkf\} \) represented by the right child of the root. These two nodes are further split independently, and the complete binary tree is obtained. The binary tree representation of the above formula thus obtained is given in Figure 4.1. The label of each node is circled, and the formula corresponding to the node is on the right side. If the subset \( F'_a \) is to be split with respect to \( a \), then according to the partitioning method, the left child will be a NULL node. The NULL node is represented by \( D \) and \( \cdot \) represents an empty formula in the Figure 4.1.

The matrix representation of a formula has already been discussed in Chapter 2. Following the matrix representation scheme, any node of the binary tree corresponds to a submatrix \( M[S,T] \) of \( M[C,\phi] \), the root obviously corresponds to \( M[C,\phi] \) itself.
Since the rows of the matrix correspond to the literals in the formula, the binary matrix associated to a given node is partitioned with respect to the row corresponding to the literal \( r \). If \( M[S, T] \) corresponds to a formula \( M(\mathcal{F}) \) at a given node where the label is \( r \), then the matrices \( M[S - S_r, T U \{r\}] \) and \( M[S_r, T U \{r\}] \) correspond to \( M(\mathcal{F}_r) \) and \( M(\mathcal{F}_s) \), respectively. As discussed earlier, if \( r \) is the label of the node in the tree representing the formula \( M[S, T] \), then the submatrix \( M[S - S_r, T U \{r\}] \) is the left child of the node. Similarly, the submatrix \( [S_r, T U \{r\}] \) corresponds to the right child of the node. Both \( M[S - S_r, T U \{r\}] \) and \( M[S_r, T U \{r\}] \) have one row (the row corresponding to the chosen literal \( r \)) less than that of the matrix \( M[S, T] \).

**Figure 4.1**: Tree-representation of the formula \( \mathcal{F} \) for TERMS

\( \square \) represents NULL node and \( \bullet \) represents zero matrix (empty formula).

Labels are circled and clauses corresponding to the nodes are on the right side of the node.
The main aim of this representation scheme is to compute the prime paths of each 
subset of clauses so as to compute the prime paths of a formula. It is already mentioned 
that all the literals in a disjunctive clause are prime paths for the clause. Hence, the 
branching process is continued until the submatrix becomes a column matrix, or a row 
matrix or, a zero matrix. The column matrix corresponds to a unit clause whereas a row 
matrix corresponds to a literal. The zero matrix corresponds to an empty formula. It may 
be noted that this occurs if there is redundancy among the set of clauses corresponding 
to the node. If the left child of a node is zero (empty), then by Theorem 3.5.2, the node 
has an empty set of prime paths. If the right child of the node is zero, it indicates that 
there is redundancy in the set of clauses corresponding to the node, and has no affect 
on the set of prime paths of the node. In Socher's algorithm, the redundant clauses are 
removed after each iteration by removing the absorbed columns (Definition 2.3.1). In 
Socher's algorithm, in order to accomplish the absorption check, each of the columns 
has to be compared with the other columns in the matrix which is obviously expensive 
especially when the matrix is of large size. However, this absorption check is avoided 
here since the redundancy among clauses leads to a node representing a zero matrix. 

In this ramification process, the left child of the node corresponding to $M[S,T]$ is 
NULL if all the entries in the $r^{th}$ row of $M[S,T]$ are 1. The right child of a node is NULL 
only when the node corresponds to a conjunctive clause. The right child of a node can 
not be NULL in other cases because it means that the formula does not have any literal 
present. If the formula does not have any literal, then the matrix corresponding to it is 
zero. If no node in the tree is zero, then the number of leaf nodes is equal to the number 
of clauses in the formula. Having chosen binary tree as the representation scheme for a 
formula, the question is how to represent the binary tree? The implementation details 
of the construction of the binary tree and the algorithm \textbf{PIAP} are presented in the 
following section.
4.3 Implementation Details

There are different representations for a binary tree, for example, representation by sequential numbering, array representation, linked representation etc. [Horowitz 83, Tremblay 84]. The binary tree representing a formula is, in general, not balanced and hence the representation by sequential numbering and array representation are not suitable. Moreover, insertion or deletion of nodes from the tree requires more quantum of changes in the level number of nodes. These problems are easily overcome through the use of linked representation. Hence, the linked representation of the binary tree is adopted here.

4.3.1 Structure of a node in the tree

Any node of the binary tree for TERMS has the following nine fields which is pictorially depicted in Figure 4.4.

Field 1 Integer $c$ :- This field acts as a flag to denote whether the current node is a root node, intermediate node or, a leaf node. Separate integer $ids$ are given to represent the root (0), intermediate left child (-2), intermediate right child (+2), left leaf node (-1) and right leaf node (1). These $ids$ are used while computing the prime paths of the tree.

Field 2 Integer $l$ :- This integer gives the row number corresponding to the label $r$ of each node and the row number is computed as follows: The number of Is in each of the row of the matrix corresponding to the node is computed, and the row having the maximum number of Is is selected. If there is a tie between the rows, then arbitrarily one of these rows is chosen. However, it is to be noted that the choice of $r$ does not affect the efficiency of the algorithm PIAP significantly as in the case of Socher's algorithm. If the node is a leaf node, then some flag is given as the
label so as to identify it since the computation of prime paths for leaf nodes is very simple. Further, if the node represents a zero matrix, a different label is given so that the computation of paths is made easy.

Field 3 Integer nrow :- The number of rows in the submatrix corresponding to the node is stored in this field. This helps to check the stopping criteria and also to compute the number of rows of its children. The root at the 0th level has the maximum number (equal to the number of literals present in the formula) of rows and the number of rows in the submatrix decreases as the level of the tree increases. Though the number of rows and the level of the tree are related, the level of a node is not required for the computation of prime paths. Hence, only the number of rows in the submatrix is stored to check the stopping criteria.

Field 4 Rows:- There is a one to one correspondence between the rows in the matrix and the literals. This correspondence must be maintained throughout and hence, the row numbers of the original matrix which are in the submatrix corresponding to the node are stored as a list of integers. Though the number of rows are same at a given level, the literals present in the clauses corresponding to the node may be different and hence it is necessary to know the literals which appear in the submatrix corresponding to the node. This information is obtained by the list of row numbers stored in each node. It is a linked list with two fields; one an integer representing the row number and the other, a pointer to the next element in the list. The structure of the list is given in Figure 4.2. This also helps when the tree is updated when there are new literals in the clauses transmitted by the reasoner.
Field 5 Cols: This is the field giving the list of columns of the original matrix which are columns of the submatrix corresponding to the node. This list is similar to that of the list of rows. The correspondence between the clauses and the columns of the matrix is maintained throughout by storing the column numbers. This is required to rebuild the original matrix from the constructed tree. This field also helps in easy deletion or, addition of clauses to the original formula which is required in RMS.

Field 6 Leftc: Pointer to the left child of the node.

Field 7 Rightc: Pointer to the right child of the node.

Field 8 Par: Pointer to the parent of the node. This pointer makes it possible to traverse the tree which is needed in the computation of prime paths.

Field 9 Path: This field gives the prime paths in the matrix corresponding to the node. This field is NULL during the construction of tree. It is used while computing the prime paths at each node. Each of the paths is stored as a list of row numbers representing the literals present in the path. The set of paths at each node is stored as a list of these paths. This data structure is suitable due to the dynamic nature of the paths as well as the set of paths. The row numbers in the lists representing the paths are stored in ascending order so as to facilitate easy subsumption check. Moreover, this representation makes deletion and addition of literals from a path easier. Further, the deletion and addition of paths from and to the set of prime
paths are made easy because of the linked representation of the set of paths. The structure of the set of prime paths is shown in Figure 4.3.

![Figure 4.3: Structure of a set of paths](image)

4.3.2 Creation of binary tree

Based on the structure of a node in the binary tree for TERMS discussed above, the pseudo code for the construction of the binary tree is given here followed by the function which computes the left child of a node.

Procedure CREATE_TREE \( (M[C, \phi]) \)

Input: \( M[C, \phi] \), the \( m \times n \) matrix corresponding to the formula \( \mathcal{F} \)

Output: \( T(\mathcal{F}) \), the binary tree corresponding to the formula \( \mathcal{F} \)

Step 0: Initialize the root node

Step 1: \( \text{root} \rightarrow c = 0 \)

Step 2: \( \text{root} \rightarrow \text{nrow} = n \)

Step 3: \( \text{root} \rightarrow \text{Col} = \text{create list}(n) \)  \( * \) Creates the list of integers 0,1,\ldots,n − 1  

Step 4: \( \text{root} \rightarrow \text{Row} = \text{create list}(m) \)  \( * \) Creates the list of integers 0,1,\ldots,m − 1  

![Figure 4.4: Structure of a node in the binary tree for TERMS](image)
Step 5: root\rightarrow 1 = maxrow (root\rightarrow row, root\rightarrow col)
   
   \* Computes the row having the maximum number of Is *

Step 6: root\rightarrow Leftc = compute_left (root)  \* Computes the left child of the root *

Step 7: root\rightarrow Rightc = compute_right (root) \* Computes the right child of the root *

Step 8: root\rightarrow Par = NULL

Step 9: root\rightarrow Path = NULL

END

Procedure  Compute-left (NODE)  \* Computes the left child of a NODE *

   Input : The node NODE

   Output : The left child of the NODE

NODE\rightarrow Leftc--\rightarrow Cols = compute_left_col (NODE\rightarrow Par\rightarrow 1, NODE\rightarrow Par\rightarrow Cols)
   
   \* Computes the list of columns of the left child of the NODE *

If (NODE\rightarrow Leftc--\rightarrow Cols is NULL) then
   return NULL
else
   NODE\rightarrow Leftc--\rightarrow nrow=NODE\rightarrow nrow-1.
   NODE\rightarrow Leftc--\rightarrow Rows=compute_chid_row(NODE\rightarrow Par\rightarrow 1, NODE\rightarrow Par\rightarrow Rows)
   
   \* Computes the list of rows in the left child of the NODE *

If (NODE\rightarrow Leftc--\rightarrow Rows or NODE\rightarrow Leftc--\rightarrow Cols has single element) then
   NODE\rightarrow Leftc--\rightarrow c = -1  \* left leaf node *
   NODE\rightarrow Leftc--\rightarrow 1 = -1  \* flag for leaf node *
   NODE\rightarrow Leftc--\rightarrow Leftc = NULL
   NODE\rightarrow Leftc--\rightarrow Rightc = NULL
else
   NODE\rightarrow Leftc--\rightarrow c = -2  \* left intermediate node *
Chapter 4. TREE AND REASON MAINTENANCE

4.4 Path Computation Using Tree

The aim of this section is to recast the PIAP for implementation using tree-representation. At any node (with label r) which represents a subformula, the corresponding prime paths are computed by making use of the sets of prime paths of the child nodes. As already mentioned in Section 4.2, the left child and right child of a node corresponding to $M[S,T]$
Chapter 4 TREE AND REASON MAINTENANCE

represent $M[S - S_r, T U \{r\}]$ and $M[S - S_r, T U \{r\}]$, respectively. In the tree representation of paths, this node (corresponding to $[S, T]$) contains $P[S - S_r, T U \{r\}]$ in the field $Path$ of the tree. Similarly, field $Path$ in the right child of the node contains $P[S_r, T U \{r\}]$. The prime paths of the node can be computed by performing simple operations like concatenation and merging of the prime paths of the left child and the right child of the node. Step 3 of PIAP is accomplished by checking whether any path of the right child of a node subsumes any path of the left child of the node. In the similar way, Step 4 is implemented by checking whether any path of the left child of the node subsumes any path of the right child of the node. By concatenating the prime paths of the left child with the label of the node, the Step 5 can be implemented. The concatenation of prime paths of the left child with the paths of the right child gives the paths obtained by Step 6. Thus the prime paths of a formula can be computed with the help of the tree constructed for the formula. Moreover, the tree structure can also be used to represent the prime paths of the formula.

4.4.1 Working of the algorithm PIAP

In this subsection, the computation of prime paths using the tree representation is illustrated with the help of an example. The example is so chosen that possible adverse cases are highlighted. Let $F = \{ab\tilde{d}, ab\tilde{c}\tilde{d}f, ab\tilde{c}f, ab\tilde{c}d, ab\tilde{c}f, ab\tilde{c}d, ab\tilde{c}f, ab\tilde{c}e, ab\tilde{c}f, ab\tilde{c}e\}$ be the formula chosen. The nodes of the tree representing $F$ are sequentially numbered as seen in Figure 4.5.

The node 15 at level 4 has prime paths $\{f, g\}$. The node 14 is NULL and so is the set of prime paths. Since the left child of node 9 is NULL, by Theorem 3.5.1, the label itself is a prime path for the node; i.e. $\{\tilde{a}\}$ is a prime path of node 9. The other paths of this node are those paths in the right child of the node. Thus the prime paths for the node 9 are $\{\tilde{a}\}$ and $\{f, g\}$. Similar is the case of node 5 and node 2. Traversing up the
tree and computing the prime paths at each node the prime paths of node 2 are \( \{a\}, \{c\}, \{e\} \) and \( \{f, g\} \). The prime paths are given on the right side of each node in Figure 4.5.

We have already seen in Section 3.6.1 that a formula is consistent if it has nonempty prime paths. Here it can be seen that the set of prime paths at node 13 is empty, even though its child nodes have nonempty set of prime paths. For the node 13, \( r = \bar{a} \), \( P[S - S_{\bar{a}}; T \cup \{r\}] = \{d, e, /\} \) and \( P[S_{\bar{a}}; T \cup \{r\}] = \{f, g\} \). The path \( \{d, e, /\} \) of the left child neither subsumes nor is subsumed by \( \{f, g\} \) giving \( P1 = \phi \). By Step 5, \( d \) has to be concatenated with \( \{d, e, f\} \). The concatenation is not possible since fundamentality is violated and therefore \( P3 = \phi \). In Step 6, though \( \{d, e, /\} \) has to be concatenated with \( \{f, g\} \), it is not possible since the union of two paths is not fundamental and hence \( P1 \) remains to be \( \phi \). Thus there is no prime path for the node 13 at level 3. In fact, one can see that the set \( \{d, e, f\} \) of clauses corresponding to the node 13 is not consistent.

The node 7 is the case where the right child of the node is empty and hence the paths are obtained only by Step 5 (concatenating the prime paths of the left child with the label of node 7). These paths are \( P3 = \{\{c, d\}, \{c, f\}, \{c, g\}\} \). All the paths in \( P3 \) are identified to be prime and hence subsumption is not required. Thus \( \{c, d\}, \{c, f\}, \{c, g\} \) are the prime paths of node 7. The prime paths \( \{b, g\}, \{f, g\}, \{b, d\}, \{b, f\}, \{d, f\} \) of node 6 are computed from the prime paths of the nodes 10 and 11.

So far we have seen a few simple cases. The prime paths for the node 3 are computed from the prime paths of node 6 and node 7. There is no path satisfying the conditions in Step 3 and Step 4, and hence \( P1 = \phi \) and \( P2 = \phi \). By concatenating the label \( b \) with the prime paths of node 6, \( P3 = \{\{b, f, g\}, \{b, d, f\}\} \) is obtained. By Step 6, \( P4 \) is obtained as \( \{b, c, d, g\}, \{b, c, f, g\}, \{b, c, g\}, \{c, d, f, g\}, \{c, f, g\}, \{b, c, d\}, \) and \( \{c, d, f\} \) of which \( \{b, c, d, g\}, \{b, c, f, g\} \) and \( \{c, d, f, g\} \) are subsumed in Step 7. Thus, the prime paths of node 3 are \( \{b, f, g\}, \{b, d, f\}, \{b, c, g\}, \{c, d, f, g\}, \{c, f, g\}, \{b, c, d\} \) and \( \{c, d, f\} \).
Chapter 4. TREE AND REASON MAINTENANCE

Figure 4.5: Binary tree and prime paths for the formula
\[ \mathcal{F} = \{\text{abcde, abcd, abd, abf, abg, abc, acf, ade, afg, abc, acf, adeg}\} \]
Thus the prime paths of the left subtree and right subtree of the root node are obtained. The root node gives a case where $P_2 \neq \phi$. The prime path $\{f,g\}$ of the left subtree subsumes the prime paths $\{b,f,g\}$ and $\{c,f,g\}$ of the right subtree. Therefore, by Step 4, the paths $\{b,f,g\}$ and $\{c,f,g\}$ in $P_2$ are prime paths for the root node also. The set of prime paths obtained by Step 5 is $P_3 = \{\{a,c\}, \{a,e\}, \{a,f,g\}\}$. By Step 6, the concatenations of the prime paths of the left subtree with that of the right subtree are obtained. Finally, all the subsumed paths are deleted and the prime paths of the formula are $\{a,c\}, \{a,e\}, \{a,f,g\}, \{b,f,g\}, \{c,f,g\}, \{\bar{a},b,d,f\}, \{\bar{a},b,c,g\}, \{\bar{a},b,c,d\}, \{\bar{a},c,d,f\}, \{b,c,d,f\}, \{b,d,e,f\}, \{b,c,e,g\}, \{b,c,d,e\}, \{c,d,e,f\}$.

4.5 Experimental Results

Based on the foregoing discussions, experiments were carried out to compare the efficiency of the proposed algorithm with respect to Socher's algorithm. The algorithms were coded in C language and were run in the same computing platform\footnote{The computing platform is IBM PC 386 Compatible}. In order to have a uniform measure of performance, the literals were taken from a fixed set of size 20 (10 variables together with its negations form the set of literals). The problem size is identified as the number of clauses in the formula. The output size is the number of prime paths computed by the algorithm. The length of a path is the number of literals present in a path.

If there are $n$ variables, then the maximum possible length of a path is $n$. The problems are generated randomly from a large set of consistent clauses. In other words, initially a very large set of clauses which is consistent is identified, and then from this set, arbitrarily, certain number of clauses are picked up to define a formula. This method was adopted because the automatic generation of random problems following some probability
distribution as well as having a consistent set of clauses was found to be a complex task. The number of problems generated by the proposed method being very large, it automatically is devoid of the dements of deviating from random generation.

In order to study the performance of the algorithm, several experiments were carried out. It is observed that the behaviour of the algorithm depends on the number of subsumptions carried out, overall time taken by the algorithm, and the number of candidate paths subjected to subsumption operation. It has already been pointed out that subsumption is the crucial operation in any prime implicants computation algorithm. Hence, the number of subsumption checks gives a measure of the performance of the algorithm. In PIAP, certain paths are identified (namely, the set \( P_3 \)) which are not subjected to any subsumption operation. Moreover, the subsumption operation is performed at different levels of the tree and hence the length of the candidate paths for subsumption are expectedly different. Thus the paths having smaller number of literals naturally take less time for subsumption than that of paths having more number of literals. So it is necessary to know the candidate path for subsumption at different levels of the tree. In the third experiment, the study is made to find the number of candidate paths at the root level. This is compared with the candidate paths in Socher's algorithm. The experimental results are summarized in the form of tables and are also pictorially depicted in the form of graphs. The details of each experiment are described below.

### 4.5.1 Experiment 1

In this experiment, the aim is to illustrate that the algorithm PIAP requires less number of subsumptions as compared to Socher's algorithm. This is so because certain subsumptions are carried out at the level of intermediate nodes where the paths of submatrices are subsumed. In one sense, a path is subsumed at a much earlier stage where expectedly the number of literals in the path is less. Moreover, at any given node, certain paths
which never get subsumed are identified in advance and this saves the computational effort. It is to be noted that it is emphasized again and again in literature [de Kleer 94, Jackson 92, Kean 90] (and here too) that subsumption operation is the most crucial operation in prime implicants computation. Hence, reducing number of subsumptions naturally results in a better and more efficient algorithm. In the present experiment, the subsumption operation is carried out in the usual way of subset checking. The same function of subsumption operation is used for both the algorithms.

The experimental results are summarized in Table 4.1 to Table 4.7. The first column of any of Table 4.1 to Table 4.7 gives the problem size and the sample number. For example, 15-12 means, this is the 12th sample data for the problem having 15 clauses. The second column gives the number of subsumptions carried out by Socher's algorithm. The third column gives the total number of paths that are generated and subjected to subsumption operation in Socher's algorithm. The fourth column gives the total number of subset checking carried out at intermediate nodes by PIAP. i.e., at nodes other than the root of the tree. This figure depicts the number of subsumptions carried out over paths in the submatrices and not on paths having full length. So, normally the subsumption operations on these paths take less computational time (as there are less number of literals) than the subsumption operation at the root. The number of subsumptions carried out at the root level is given in column 5. The total number (column 4 + column 5) of subsumptions by the PIAP is given in the sixth column. The seventh column gives the number of paths subjected to subsumption operation at the root level. The number of prime paths obtained after subsumption is given in column 8. These are the prime paths void of the label of the root. It is already mentioned that certain paths are identified to be prime at an early stage, and are not subjected to subsumption operation. The number of such paths is given in column 9. The number of paths in column 7 and column 9 together gives the number of paths that are generated
by PIAP so as to compute the prime paths of the formula. This is given in column 10.
The total number of prime paths (column 8 + column 9) for the problem is given in the
last column of the tables (Table 4.1 to Table 4.7). In the case of Socher's algorithm, this
is the outcome of performing subsumption operation on the number of paths given in
column 3.

It is to be noted here that if the left child of the root node is NULL, then column 9
gives the number of paths obtained by appending the label of the root node to the set of
prime paths of the right child of the root. These paths are never subjected to subsumption
checks since they are already prime. There are certain samples (15-3, 15-5, 15-6, 15-10,
15-13, 25-1, 25-19, 30-6, 30-13, 301-8, 35-1, 35-2, 35-8, 40-1, 40-2, 40-4, 40-18, 45-3, 45-6)
where the entries in columns 5, 7 and 8 are zero. In all these cases, the left child of
the root node is NULL and so is its set of prime paths. If the left child is not NULL,
column 9 gives the prime paths obtained by concatenating the label of the root node
to the paths in the set of prime paths of the left child of the root node. There are few
cases where there was memory allocation problem for Socher's algorithm. Such cases are
denoted by * in the tables. Further, SOCH stands for Socher's algorithm in the tables
given in this chapter.
### Table 4.1: Table giving the number of subset checking and the number of paths using Socher's algorithm and PIAP

<table>
<thead>
<tr>
<th>Prob. Size &amp; Samp. No.</th>
<th>SOCH # of Subs. paths for subs.</th>
<th># of Subsumptions Till root</th>
<th>At root</th>
<th>Total</th>
<th># of Paths for subs. after subs. in ( P3 )</th>
<th>Total # of Paths</th>
<th>Total Prime Paths</th>
</tr>
</thead>
<tbody>
<tr>
<td>15-1</td>
<td>6914</td>
<td>163</td>
<td>767</td>
<td>1003</td>
<td>1770</td>
<td>30</td>
<td>22</td>
</tr>
<tr>
<td>15-2</td>
<td>5595</td>
<td>142</td>
<td>906</td>
<td>2553</td>
<td>3495</td>
<td>71</td>
<td>25</td>
</tr>
<tr>
<td>15-3</td>
<td>300</td>
<td>27</td>
<td>56</td>
<td>0</td>
<td>56</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>15-4</td>
<td>11146</td>
<td>271</td>
<td>596</td>
<td>2661</td>
<td>3257</td>
<td>67</td>
<td>39</td>
</tr>
<tr>
<td>15-5</td>
<td>1267</td>
<td>65</td>
<td>348</td>
<td>0</td>
<td>348</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>15-6</td>
<td>4122</td>
<td>115</td>
<td>1101</td>
<td>0</td>
<td>1101</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>15-7</td>
<td>1795</td>
<td>57</td>
<td>928</td>
<td>873</td>
<td>1801</td>
<td>26</td>
<td>26</td>
</tr>
<tr>
<td>15-8</td>
<td>7212</td>
<td>122</td>
<td>562</td>
<td>1394</td>
<td>1956</td>
<td>34</td>
<td>30</td>
</tr>
<tr>
<td>15-9</td>
<td>1086</td>
<td>60</td>
<td>164</td>
<td>289</td>
<td>453</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>15-10</td>
<td>1511</td>
<td>65</td>
<td>194</td>
<td>0</td>
<td>194</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>15-11</td>
<td>1719</td>
<td>67</td>
<td>240</td>
<td>472</td>
<td>712</td>
<td>28</td>
<td>14</td>
</tr>
<tr>
<td>15-12</td>
<td>2105</td>
<td>95</td>
<td>163</td>
<td>347</td>
<td>510</td>
<td>16</td>
<td>13</td>
</tr>
<tr>
<td>15-13</td>
<td>391</td>
<td>34</td>
<td>161</td>
<td>0</td>
<td>161</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>15-14</td>
<td>2885</td>
<td>103</td>
<td>555</td>
<td>504</td>
<td>1059</td>
<td>16</td>
<td>15</td>
</tr>
<tr>
<td>15-15</td>
<td>2693</td>
<td>114</td>
<td>578</td>
<td>402</td>
<td>980</td>
<td>13</td>
<td>11</td>
</tr>
<tr>
<td>15-16</td>
<td>938</td>
<td>61</td>
<td>264</td>
<td>195</td>
<td>459</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>15-17</td>
<td>835</td>
<td>52</td>
<td>62</td>
<td>150</td>
<td>212</td>
<td>10</td>
<td>9</td>
</tr>
<tr>
<td>15-18</td>
<td>2306</td>
<td>72</td>
<td>646</td>
<td>548</td>
<td>1194</td>
<td>22</td>
<td>11</td>
</tr>
<tr>
<td>15-19</td>
<td>7096</td>
<td>229</td>
<td>1292</td>
<td>2312</td>
<td>3604</td>
<td>55</td>
<td>34</td>
</tr>
<tr>
<td>15-20</td>
<td>2574</td>
<td>93</td>
<td>424</td>
<td>1149</td>
<td>1573</td>
<td>37</td>
<td>28</td>
</tr>
</tbody>
</table>
Table 4.2: Table giving the number of subset checking and the number of paths using Socher's algorithm and PIAP

<table>
<thead>
<tr>
<th>Prob. Size &amp; Samp. No.</th>
<th>SOCH</th>
<th>PIAP</th>
<th>Total Prime Paths</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Subs. # of paths for subs.</td>
<td># of Subsumptions Till root</td>
<td>At root</td>
</tr>
<tr>
<td>20-1</td>
<td>30647</td>
<td>276</td>
<td>5438</td>
</tr>
<tr>
<td>20-2</td>
<td>33358</td>
<td>291</td>
<td>7725</td>
</tr>
<tr>
<td>20-3</td>
<td>60243</td>
<td>649</td>
<td>27832</td>
</tr>
<tr>
<td>20-4</td>
<td>1764</td>
<td>66</td>
<td>5982</td>
</tr>
<tr>
<td>20-5</td>
<td>1069</td>
<td>80</td>
<td>15</td>
</tr>
<tr>
<td>20-6</td>
<td>*</td>
<td>*</td>
<td>20303</td>
</tr>
<tr>
<td>20-7</td>
<td>2948</td>
<td>181</td>
<td>1765</td>
</tr>
<tr>
<td>20-8</td>
<td>14174</td>
<td>333</td>
<td>4794</td>
</tr>
<tr>
<td>20-9</td>
<td>3265</td>
<td>136</td>
<td>3444</td>
</tr>
<tr>
<td>20-10</td>
<td>*</td>
<td>*</td>
<td>23129</td>
</tr>
<tr>
<td>20-11</td>
<td>4929</td>
<td>118</td>
<td>887</td>
</tr>
<tr>
<td>20-12</td>
<td>6665</td>
<td>162</td>
<td>1684</td>
</tr>
<tr>
<td>20-13</td>
<td>4157</td>
<td>98</td>
<td>829</td>
</tr>
<tr>
<td>20-14</td>
<td>5620</td>
<td>167</td>
<td>397</td>
</tr>
<tr>
<td>20-15</td>
<td>1412</td>
<td>55</td>
<td>231</td>
</tr>
<tr>
<td>20-16</td>
<td>9507</td>
<td>234</td>
<td>899</td>
</tr>
<tr>
<td>20-17</td>
<td>1206</td>
<td>65</td>
<td>62</td>
</tr>
<tr>
<td>20-18</td>
<td>3257</td>
<td>88</td>
<td>642</td>
</tr>
<tr>
<td>20-19</td>
<td>2304</td>
<td>70</td>
<td>275</td>
</tr>
<tr>
<td>20-20</td>
<td>8706</td>
<td>187</td>
<td>1501</td>
</tr>
</tbody>
</table>
### Table 4.3: Table giving the number of subset checking and the number of paths using Socher's algorithm and PIAP

<table>
<thead>
<tr>
<th>Prob. Size &amp; Samp. No.</th>
<th>SOCH</th>
<th>PIAP</th>
<th>Total # of Prime Paths</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Subs.</td>
<td>paths for subs.</td>
<td># of Subsumptions</td>
</tr>
<tr>
<td></td>
<td>Till root</td>
<td>At root</td>
<td>Total</td>
</tr>
<tr>
<td>25-1</td>
<td>27</td>
<td>6</td>
<td>46</td>
</tr>
<tr>
<td>25-2</td>
<td>432</td>
<td>43</td>
<td>341</td>
</tr>
<tr>
<td>25-3</td>
<td>1500</td>
<td>88</td>
<td>521</td>
</tr>
<tr>
<td>25-4</td>
<td>17535</td>
<td>455</td>
<td>2846</td>
</tr>
<tr>
<td>25-5</td>
<td>4733</td>
<td>172</td>
<td>666</td>
</tr>
<tr>
<td>25-6</td>
<td>5805</td>
<td>190</td>
<td>386</td>
</tr>
<tr>
<td>25-7</td>
<td>3150</td>
<td>138</td>
<td>310</td>
</tr>
<tr>
<td>25-8</td>
<td>1102</td>
<td>88</td>
<td>150</td>
</tr>
<tr>
<td>25-9</td>
<td>1408</td>
<td>87</td>
<td>490</td>
</tr>
<tr>
<td>25-10</td>
<td>2010</td>
<td>94</td>
<td>491</td>
</tr>
<tr>
<td>25-11</td>
<td>13400</td>
<td>276</td>
<td>1582</td>
</tr>
<tr>
<td>25-12</td>
<td>64126</td>
<td>468</td>
<td>23129</td>
</tr>
<tr>
<td>25-13</td>
<td>3065</td>
<td>111</td>
<td>197</td>
</tr>
<tr>
<td>25-14</td>
<td>3065</td>
<td>149</td>
<td>88</td>
</tr>
<tr>
<td>25-15</td>
<td>11931</td>
<td>326</td>
<td>797</td>
</tr>
<tr>
<td>25-16</td>
<td>11031</td>
<td>477</td>
<td>902</td>
</tr>
<tr>
<td>25-17</td>
<td>7328</td>
<td>197</td>
<td>736</td>
</tr>
<tr>
<td>25-18</td>
<td>10276</td>
<td>288</td>
<td>915</td>
</tr>
<tr>
<td>25-19</td>
<td>17621</td>
<td>301</td>
<td>4355</td>
</tr>
<tr>
<td>25-20</td>
<td>117395</td>
<td>955</td>
<td>2396</td>
</tr>
</tbody>
</table>
Table 4.4: Table giving the number of subset checking and the number of paths using Socher's algorithm and PIAP

<table>
<thead>
<tr>
<th>Prob.</th>
<th>SOCH</th>
<th>PIAP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Subs.</td>
<td># of</td>
</tr>
<tr>
<td></td>
<td>paths for subs.</td>
<td># of</td>
</tr>
<tr>
<td>Size &amp; Samp. No.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>30-1</td>
<td>24066</td>
<td>450</td>
</tr>
<tr>
<td>30-2</td>
<td>13441</td>
<td>324</td>
</tr>
<tr>
<td>30-3</td>
<td>30453</td>
<td>511</td>
</tr>
<tr>
<td>30-4</td>
<td>7645</td>
<td>238</td>
</tr>
<tr>
<td>30-5</td>
<td>21869</td>
<td>518</td>
</tr>
<tr>
<td>30-6</td>
<td>13899</td>
<td>325</td>
</tr>
<tr>
<td>30-7</td>
<td>59776</td>
<td>487</td>
</tr>
<tr>
<td>30-8</td>
<td>4116</td>
<td>117</td>
</tr>
<tr>
<td>30-9</td>
<td>4069</td>
<td>191</td>
</tr>
<tr>
<td>30-10</td>
<td>8498</td>
<td>206</td>
</tr>
<tr>
<td>30-11</td>
<td>7482</td>
<td>248</td>
</tr>
<tr>
<td>30-12</td>
<td>9937</td>
<td>226</td>
</tr>
<tr>
<td>30-13</td>
<td>28</td>
<td>7</td>
</tr>
<tr>
<td>30-14</td>
<td>680</td>
<td>46</td>
</tr>
<tr>
<td>30-15</td>
<td>2371</td>
<td>104</td>
</tr>
<tr>
<td>30-16</td>
<td>6013</td>
<td>180</td>
</tr>
<tr>
<td>30-17</td>
<td>13666</td>
<td>367</td>
</tr>
<tr>
<td>30-18</td>
<td>29901</td>
<td>500</td>
</tr>
<tr>
<td>30-19</td>
<td>13324</td>
<td>258</td>
</tr>
<tr>
<td>30-20</td>
<td>15837</td>
<td>399</td>
</tr>
</tbody>
</table>
Table 4.5: Table giving the number of subset checking and the number of paths using Socher's algorithm and PIAP

<table>
<thead>
<tr>
<th>Prob.</th>
<th>SOCH</th>
<th>PIAP</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size &amp; Samp. No.</td>
<td># of Subsets</td>
<td># of Paths</td>
<td>Till root</td>
</tr>
<tr>
<td>35-1</td>
<td>1735</td>
<td>125</td>
<td>249</td>
</tr>
<tr>
<td>35-2</td>
<td>3101</td>
<td>220</td>
<td>343</td>
</tr>
<tr>
<td>35-3</td>
<td>6645</td>
<td>241</td>
<td>658</td>
</tr>
<tr>
<td>35-4</td>
<td>13399</td>
<td>420</td>
<td>781</td>
</tr>
<tr>
<td>35-5</td>
<td>8487</td>
<td>226</td>
<td>729</td>
</tr>
<tr>
<td>35-6</td>
<td>3536</td>
<td>99</td>
<td>560</td>
</tr>
<tr>
<td>35-7</td>
<td>8082</td>
<td>209</td>
<td>1471</td>
</tr>
<tr>
<td>35-8</td>
<td>11588</td>
<td>488</td>
<td>688</td>
</tr>
<tr>
<td>35-9</td>
<td>9319</td>
<td>222</td>
<td>805</td>
</tr>
<tr>
<td>35-10</td>
<td>8583</td>
<td>249</td>
<td>792</td>
</tr>
<tr>
<td>35-11</td>
<td>17219</td>
<td>405</td>
<td>3582</td>
</tr>
<tr>
<td>35-12</td>
<td>*</td>
<td>*</td>
<td>2145</td>
</tr>
<tr>
<td>35-13</td>
<td>17865</td>
<td>348</td>
<td>2326</td>
</tr>
<tr>
<td>35-14</td>
<td>6449</td>
<td>210</td>
<td>1192</td>
</tr>
<tr>
<td>35-15</td>
<td>2171</td>
<td>123</td>
<td>186</td>
</tr>
<tr>
<td>35-16</td>
<td>2550</td>
<td>129</td>
<td>420</td>
</tr>
<tr>
<td>35-17</td>
<td>7328</td>
<td>197</td>
<td>550</td>
</tr>
<tr>
<td>35-18</td>
<td>20035</td>
<td>425</td>
<td>2795</td>
</tr>
<tr>
<td>35-19</td>
<td>12737</td>
<td>246</td>
<td>1250</td>
</tr>
<tr>
<td>35-20</td>
<td>10783</td>
<td>360</td>
<td>883</td>
</tr>
</tbody>
</table>
Table 4.6: Table giving the number of subset checking and the number of paths using Socher's algorithm and PIAP

<table>
<thead>
<tr>
<th>Prob. Size &amp; Samp. No.</th>
<th>SOCH</th>
<th>PIAP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Subs.</td>
<td># of paths for subs.</td>
</tr>
<tr>
<td>40-1</td>
<td>26867</td>
<td>583</td>
</tr>
<tr>
<td>40-2</td>
<td>6066</td>
<td>225</td>
</tr>
<tr>
<td>40-3</td>
<td>35782</td>
<td>668</td>
</tr>
<tr>
<td>40-4</td>
<td>25408</td>
<td>447</td>
</tr>
<tr>
<td>40-5</td>
<td>25020</td>
<td>385</td>
</tr>
<tr>
<td>40-6</td>
<td>6529</td>
<td>216</td>
</tr>
<tr>
<td>40-7</td>
<td>5510</td>
<td>269</td>
</tr>
<tr>
<td>40-8</td>
<td>10091</td>
<td>348</td>
</tr>
<tr>
<td>40-9</td>
<td>20270</td>
<td>412</td>
</tr>
<tr>
<td>40-10</td>
<td>26210</td>
<td>665</td>
</tr>
<tr>
<td>40-11</td>
<td>16982</td>
<td>419</td>
</tr>
<tr>
<td>40-12</td>
<td>11309</td>
<td>279</td>
</tr>
<tr>
<td>40-13</td>
<td>11319</td>
<td>306</td>
</tr>
<tr>
<td>40-14</td>
<td>14061</td>
<td>321</td>
</tr>
<tr>
<td>40-15</td>
<td>4997</td>
<td>229</td>
</tr>
<tr>
<td>40-16</td>
<td>17551</td>
<td>342</td>
</tr>
<tr>
<td>40-17</td>
<td>16663</td>
<td>436</td>
</tr>
<tr>
<td>40-18</td>
<td>1671</td>
<td>179</td>
</tr>
<tr>
<td>40-19</td>
<td>6217</td>
<td>191</td>
</tr>
<tr>
<td>40-20</td>
<td>20949</td>
<td>557</td>
</tr>
</tbody>
</table>
### Table 4.7: Table giving the number of subset checking and the number of paths using Socher's algorithm and PIAP

<table>
<thead>
<tr>
<th>Prob.</th>
<th>SOCH</th>
<th>PIAP</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Size</td>
<td># of</td>
<td># of</td>
<td></td>
</tr>
<tr>
<td>&amp; Samp.</td>
<td>Subs.</td>
<td>Subsumptions</td>
<td>Paths</td>
</tr>
<tr>
<td>No.</td>
<td>Checks</td>
<td>Till root</td>
<td>At root</td>
</tr>
<tr>
<td>45-1</td>
<td>31064</td>
<td>3235</td>
<td>2338</td>
</tr>
<tr>
<td>45-2</td>
<td>9600</td>
<td>1249</td>
<td>729</td>
</tr>
<tr>
<td>45-3</td>
<td>32788</td>
<td>3615</td>
<td>0</td>
</tr>
<tr>
<td>45-4</td>
<td>10382</td>
<td>1005</td>
<td>263</td>
</tr>
<tr>
<td>45-5</td>
<td>15661</td>
<td>1545</td>
<td>193</td>
</tr>
<tr>
<td>45-6</td>
<td>12159</td>
<td>989</td>
<td>0</td>
</tr>
<tr>
<td>45-7</td>
<td>32939</td>
<td>1620</td>
<td>1556</td>
</tr>
<tr>
<td>45-8</td>
<td>28817</td>
<td>1922</td>
<td>1697</td>
</tr>
<tr>
<td>45-9</td>
<td>18245</td>
<td>1123</td>
<td>1062</td>
</tr>
<tr>
<td>50-1</td>
<td>8552</td>
<td>823</td>
<td>351</td>
</tr>
<tr>
<td>50-2</td>
<td>15889</td>
<td>1389</td>
<td>932</td>
</tr>
<tr>
<td>50-3</td>
<td>3459</td>
<td>291</td>
<td>20</td>
</tr>
<tr>
<td>50-4</td>
<td>20616</td>
<td>1198</td>
<td>1297</td>
</tr>
<tr>
<td>50-5</td>
<td>*</td>
<td>*</td>
<td>5183</td>
</tr>
<tr>
<td>50-6</td>
<td>15117</td>
<td>527</td>
<td>345</td>
</tr>
<tr>
<td>50-7</td>
<td>*</td>
<td>*</td>
<td>6125</td>
</tr>
</tbody>
</table>
From these tables, it is clear that the number (column 5) of subsumption operations carried out by PIAP at the root level is much less compared to the number (column 2) of subsumption operations carried out by Socher's algorithm. In fact, the total number (column 6) of subsumptions required by PIAP is less compared to Socher's algorithm except for few stray cases, (15-7, 20-4, 20-9, 25-1, 25-2, 30-13). However, for these samples also, the number of subsumptions required at the root level is less compared to Socher's algorithm. The number of subsumptions are more in these cases due to the following reason. In order to obtain the prime paths at any node, subsumption check is performed between the set of prime paths of its children so as to obtain the paths which are prime for the parent node (Step 3 and Step 4 of PIAP). If there is no path in the set of prime paths of children which hold the property of being prime at the parent node, then these subsumptions are of no use. If most of the nodes in the tree are of this type, then the number of subsumptions required by PIAP can be more compared to Socher's algorithm. However, these subsumptions are performed at the intermediate levels where the length of paths are less and hence take less execution time, ex. 15-7, 20-9, 25-1, 25-2, and 30-13. Problem 20-4 and 25-12 are the only two cases among the samples tested, where the number of subsumptions as well as the execution time is more for PIAP. However, it can be seen that the number of paths generated by PIAP for all samples (including 20-4 and 25-12) is less than paths generated by Socher's algorithm.

The average number of subsumptions required by both the algorithms for samples of fixed size is calculated. In order to find the significance of the algorithm PIAP over Socher's algorithm, paired t-test has been performed and the results obtained are given in Table 4.8. Apart from the average number of subsumptions required by both the algorithms (column 2 and column 5), the standard deviation (SD) and coefficient of variation (CV) of the samples for both the algorithms are also given. The degrees of freedom (DF), computed t value (paired t), and the significant level (≈ probability p),
are given respectively in the last three columns of Table 4.8. \( p < .0005 \) means that the computed \( t \)-value is greater than the tabled \( t \) value [Rao 75] of \( t \)-distribution at the 0.05% level of significance. In most of the cases \( p < .0005 \) in Table 4.8 and this substantiates that the algorithm PIAP perform significantly better than Socher's algorithm.

The behaviour of the algorithm not only depends on the input size but also on the output size - in fact more on the output size than the input size. In order to study the behaviour of the algorithms based on the number of prime paths, the samples were grouped according to the output size. The average number of subsumptions required for each of the groups is calculated and the same is given in Table 4.9. The results obtained by the paired \( t \)-test performed to study the performance of both the algorithms are quite significant and substantiates positively that PIAP is significantly better than Socher's
algorithm.

The first order regression curve for the data in Table 4.9 is obtained\(^2\) and is given in Figure 4.6. It is evident from the Figure 4.6 that the performance of PIAP is better than Socher's algorithm.

Table 4.9: Table giving the \# of Prime Paths and the average
\# of subsumptions with SD and CV, by Socher's algorithm and PIAP

<table>
<thead>
<tr>
<th># of PIs</th>
<th>SOCH</th>
<th>PIAP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mean</td>
<td>SD</td>
</tr>
<tr>
<td>10–15</td>
<td>1534</td>
<td>1219</td>
</tr>
<tr>
<td>15–20</td>
<td>2954</td>
<td>3170</td>
</tr>
<tr>
<td>20–25</td>
<td>4494</td>
<td>3314</td>
</tr>
<tr>
<td>25–30</td>
<td>7086</td>
<td>3980</td>
</tr>
<tr>
<td>30–35</td>
<td>7348</td>
<td>3941</td>
</tr>
<tr>
<td>35–40</td>
<td>13977</td>
<td>6699</td>
</tr>
<tr>
<td>40–45</td>
<td>15030</td>
<td>7751</td>
</tr>
<tr>
<td>45–50</td>
<td>19030</td>
<td>7938</td>
</tr>
</tbody>
</table>

\(^2\)Package used for this purpose is Regression Analysis for Time Series (RATS) Ver. 2.0.
Figure 4.6: The graph depicting the # of subsumptions required by Socher's algorithm and PIAP.
4.5.2 Experiment 2

It is to illustrate that the execution time taken by PIAP is much less compared to Socher's algorithm. The time taken for the same samples considered in Experiment 1 are given in Table 4.10 to Table 4.16. From these tables it is clear that the execution time of PIAP is less than that of Socher's algorithm. The samples 20-4 and 25-12 are two cases where the execution time for PIAP is more than that of Socher's algorithm.

The average execution time for the samples of same input size is computed. As in the case of Experiment 1, paired t-test is performed and the results obtained are in given in Table 4.17. The values in the last column of the Table 4.17 gives the level of significance (≈ probability p). It can be seen that the value of p is < .0005 for all of the cases, supporting the claim that PIAP is efficient than Socher's algorithm.

It is already mentioned that the performance of algorithms depend on the size of the output also. Therefore, the average execution time taken by the algorithms depend on the number of prime paths. Hence, the average execution time required by both the algorithms for groups formed based on the output size are also calculated, and the paired t-test is performed for the data thus obtained. The results are given in Table 4.18. From the values of p in the last column of Table 4.18, it can be seen that the execution time taken by PIAP is less compared to that of Socher's algorithm.

The first order regression curve for the data in Table 4.18 is obtained and is given in Figure 4.7. The graph substantiate that PIAP is significantly efficient than Socher's algorithm. Thus, the execution time for PIAP proposed in Chapter 3 is less than that of Socher's algorithm.
### Table 4.10: Table giving the execution time of Socher's algorithm and PIAP

<table>
<thead>
<tr>
<th>Prob.No.</th>
<th>PIs</th>
<th>SOCH</th>
<th>PIAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>15–1</td>
<td>38</td>
<td>0.879121</td>
<td>0.219780</td>
</tr>
<tr>
<td>15–2</td>
<td>49</td>
<td>0.879121</td>
<td>0.494505</td>
</tr>
<tr>
<td>15–3</td>
<td>8</td>
<td>0.219780</td>
<td>0.054945</td>
</tr>
<tr>
<td>15–4</td>
<td>46</td>
<td>1.263736</td>
<td>0.274725</td>
</tr>
<tr>
<td>15–5</td>
<td>17</td>
<td>0.329670</td>
<td>0.054945</td>
</tr>
<tr>
<td>15–6</td>
<td>31</td>
<td>0.439560</td>
<td>0.109890</td>
</tr>
<tr>
<td>15–7</td>
<td>32</td>
<td>0.329670</td>
<td>0.164835</td>
</tr>
<tr>
<td>15–8</td>
<td>42</td>
<td>0.659341</td>
<td>0.219780</td>
</tr>
<tr>
<td>15–9</td>
<td>20</td>
<td>0.274725</td>
<td>0.109890</td>
</tr>
<tr>
<td>15–10</td>
<td>17</td>
<td>0.329670</td>
<td>0.054945</td>
</tr>
<tr>
<td>15–11</td>
<td>25</td>
<td>0.329670</td>
<td>0.109890</td>
</tr>
<tr>
<td>15–12</td>
<td>22</td>
<td>0.384615</td>
<td>0.054945</td>
</tr>
<tr>
<td>15–13</td>
<td>11</td>
<td>0.274725</td>
<td>0.054945</td>
</tr>
<tr>
<td>15–14</td>
<td>28</td>
<td>0.494505</td>
<td>0.164835</td>
</tr>
<tr>
<td>15–15</td>
<td>26</td>
<td>0.494505</td>
<td>0.109890</td>
</tr>
<tr>
<td>15–16</td>
<td>20</td>
<td>0.384615</td>
<td>0.054945</td>
</tr>
<tr>
<td>15–17</td>
<td>14</td>
<td>0.384615</td>
<td>0.054945</td>
</tr>
<tr>
<td>15–18</td>
<td>35</td>
<td>0.384615</td>
<td>0.219780</td>
</tr>
<tr>
<td>15–19</td>
<td>41</td>
<td>1.043956</td>
<td>0.384615</td>
</tr>
<tr>
<td>15–20</td>
<td>34</td>
<td>0.549451</td>
<td>0.164835</td>
</tr>
</tbody>
</table>

Average Time: 0.516733 0.1565392
Table 4.11: Table giving the execution time of Socher’s algorithm and PIAP

<table>
<thead>
<tr>
<th>Prob. No.</th>
<th>PIs</th>
<th>SOCH</th>
<th>PIAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>20-1</td>
<td>105</td>
<td>2.032967</td>
<td>1.483516</td>
</tr>
<tr>
<td>20-2</td>
<td>115</td>
<td>2.087912</td>
<td>1.538462</td>
</tr>
<tr>
<td>20-3</td>
<td>104</td>
<td>4.890110</td>
<td>5.000000</td>
</tr>
<tr>
<td>20-4</td>
<td>26</td>
<td>.769231</td>
<td>.879121</td>
</tr>
<tr>
<td>20-5</td>
<td>17</td>
<td>.769231</td>
<td>.384615</td>
</tr>
<tr>
<td>20-6</td>
<td>226</td>
<td>*</td>
<td>5.659341</td>
</tr>
<tr>
<td>20-7</td>
<td>22</td>
<td>.989011</td>
<td>.494505</td>
</tr>
<tr>
<td>20-8</td>
<td>40</td>
<td>2.252747</td>
<td>1.153846</td>
</tr>
<tr>
<td>20-9</td>
<td>23</td>
<td>.879121</td>
<td>.604396</td>
</tr>
<tr>
<td>20-10</td>
<td>243</td>
<td>*</td>
<td>5.604396</td>
</tr>
<tr>
<td>20-11</td>
<td>44</td>
<td>.604396</td>
<td>.219780</td>
</tr>
<tr>
<td>20-12</td>
<td>38</td>
<td>.879121</td>
<td>.274725</td>
</tr>
<tr>
<td>20-13</td>
<td>39</td>
<td>.604396</td>
<td>.164835</td>
</tr>
<tr>
<td>20-14</td>
<td>35</td>
<td>.769231</td>
<td>.164835</td>
</tr>
<tr>
<td>20-15</td>
<td>20</td>
<td>.439560</td>
<td>.054945</td>
</tr>
<tr>
<td>20-16</td>
<td>36</td>
<td>1.263736</td>
<td>.219780</td>
</tr>
<tr>
<td>20-17</td>
<td>16</td>
<td>.434505</td>
<td>.054945</td>
</tr>
<tr>
<td>20-18</td>
<td>31</td>
<td>.494505</td>
<td>.164835</td>
</tr>
<tr>
<td>20-19</td>
<td>27</td>
<td>.439560</td>
<td>.109890</td>
</tr>
<tr>
<td>20-20</td>
<td>45</td>
<td>.989011</td>
<td>.384615</td>
</tr>
<tr>
<td>** Average Time**</td>
<td></td>
<td>1.199353</td>
<td>.741758</td>
</tr>
</tbody>
</table>
Table 4.12: Table giving the execution time of Socher's algorithm and PIAP

<table>
<thead>
<tr>
<th>Prob. No.</th>
<th>PIs</th>
<th>Execution time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>SOCH</td>
</tr>
<tr>
<td>25-1</td>
<td>5</td>
<td>.219870</td>
</tr>
<tr>
<td>25-2</td>
<td>11</td>
<td>.549451</td>
</tr>
<tr>
<td>25-3</td>
<td>18</td>
<td>.659341</td>
</tr>
<tr>
<td>25-4</td>
<td>42</td>
<td>2.692308</td>
</tr>
<tr>
<td>25-5</td>
<td>36</td>
<td>1.153846</td>
</tr>
<tr>
<td>25-6</td>
<td>28</td>
<td>1.263726</td>
</tr>
<tr>
<td>25-7</td>
<td>22</td>
<td>.989011</td>
</tr>
<tr>
<td>25-8</td>
<td>16</td>
<td>.714286</td>
</tr>
<tr>
<td>25-9</td>
<td>19</td>
<td>.769231</td>
</tr>
<tr>
<td>25-10</td>
<td>18</td>
<td>.659341</td>
</tr>
<tr>
<td>25-11</td>
<td>36</td>
<td>1.263736</td>
</tr>
<tr>
<td>25-12</td>
<td>23</td>
<td>3.681319</td>
</tr>
<tr>
<td>25-13</td>
<td>21</td>
<td>.769231</td>
</tr>
<tr>
<td>25-14</td>
<td>16</td>
<td>.824176</td>
</tr>
<tr>
<td>25-15</td>
<td>31</td>
<td>2.252747</td>
</tr>
<tr>
<td>25-16</td>
<td>30</td>
<td>2.472527</td>
</tr>
<tr>
<td>25-17</td>
<td>34</td>
<td>.934066</td>
</tr>
<tr>
<td>25-18</td>
<td>37</td>
<td>1.538462</td>
</tr>
<tr>
<td>25-19</td>
<td>50</td>
<td>1.813187</td>
</tr>
<tr>
<td>25-20</td>
<td>116</td>
<td>6.043956</td>
</tr>
</tbody>
</table>

Average Time: 1.563191 | .491758
Table 4.13: Table giving the execution time of Socher’s algorithm and PIAP

<table>
<thead>
<tr>
<th>Prob. No.</th>
<th>Pl</th>
<th>SOCH</th>
<th>PIAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>30-1</td>
<td>57</td>
<td>2.582418</td>
<td>.714286</td>
</tr>
<tr>
<td>30-2</td>
<td>41</td>
<td>1.813187</td>
<td>.384615</td>
</tr>
<tr>
<td>30-3</td>
<td>48</td>
<td>2.692308</td>
<td>.329670</td>
</tr>
<tr>
<td>30-4</td>
<td>38</td>
<td>1.263736</td>
<td>.219780</td>
</tr>
<tr>
<td>30-5</td>
<td>47</td>
<td>3.076923</td>
<td>.329670</td>
</tr>
<tr>
<td>30-6</td>
<td>36</td>
<td>1.758242</td>
<td>.219780</td>
</tr>
<tr>
<td>30-7</td>
<td>95</td>
<td>3.571429</td>
<td>1.153846</td>
</tr>
<tr>
<td>30-8</td>
<td>34</td>
<td>.769231</td>
<td>.164835</td>
</tr>
<tr>
<td>30-9</td>
<td>26</td>
<td>1.923077</td>
<td>.219780</td>
</tr>
<tr>
<td>30-10</td>
<td>33</td>
<td>1.428571</td>
<td>.109890</td>
</tr>
<tr>
<td>30-11</td>
<td>26</td>
<td>1.593407</td>
<td>.164835</td>
</tr>
<tr>
<td>30-12</td>
<td>43</td>
<td>1.758242</td>
<td>.384615</td>
</tr>
<tr>
<td>30-13</td>
<td>5</td>
<td>.549451</td>
<td>.054945</td>
</tr>
<tr>
<td>30-14</td>
<td>14</td>
<td>.494505</td>
<td>.054945</td>
</tr>
<tr>
<td>30-15</td>
<td>24</td>
<td>.879121</td>
<td>.219780</td>
</tr>
<tr>
<td>30-16</td>
<td>28</td>
<td>1.263736</td>
<td>.109890</td>
</tr>
<tr>
<td>30-17</td>
<td>40</td>
<td>1.978022</td>
<td>.274725</td>
</tr>
<tr>
<td>30-18</td>
<td>60</td>
<td>2.967033</td>
<td>.439560</td>
</tr>
<tr>
<td>30-19</td>
<td>46</td>
<td>1.428571</td>
<td>.439560</td>
</tr>
<tr>
<td>30-20</td>
<td>40</td>
<td>2.362637</td>
<td>.329670</td>
</tr>
<tr>
<td>Average Time</td>
<td></td>
<td>1.806479</td>
<td>.315954</td>
</tr>
</tbody>
</table>
Table 4.14: Table giving the execution time of Socher's algorithm and PIAP

<table>
<thead>
<tr>
<th>Prob. No.</th>
<th>PIs</th>
<th>SOCH</th>
<th>PIAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>35-1</td>
<td>10</td>
<td>.879121</td>
<td>.054945</td>
</tr>
<tr>
<td>35-2</td>
<td>14</td>
<td>1.813181</td>
<td>.109890</td>
</tr>
<tr>
<td>35-3</td>
<td>31</td>
<td>1.758242</td>
<td>.219780</td>
</tr>
<tr>
<td>35-4</td>
<td>34</td>
<td>2.582418</td>
<td>.274725</td>
</tr>
<tr>
<td>35-5</td>
<td>33</td>
<td>1.318681</td>
<td>.219780</td>
</tr>
<tr>
<td>35-6</td>
<td>30</td>
<td>.879121</td>
<td>.109890</td>
</tr>
<tr>
<td>35-7</td>
<td>33</td>
<td>1.538462</td>
<td>.384615</td>
</tr>
<tr>
<td>35-8</td>
<td>16</td>
<td>2.582418</td>
<td>.109890</td>
</tr>
<tr>
<td>35-9</td>
<td>33</td>
<td>1.758242</td>
<td>.219780</td>
</tr>
<tr>
<td>35-10</td>
<td>33</td>
<td>1.923077</td>
<td>.164835</td>
</tr>
<tr>
<td>35-11</td>
<td>38</td>
<td>2.582418</td>
<td>.659341</td>
</tr>
<tr>
<td>35-12</td>
<td>48</td>
<td>*</td>
<td>.439560</td>
</tr>
<tr>
<td>35-13</td>
<td>42</td>
<td>2.252747</td>
<td>.384615</td>
</tr>
<tr>
<td>35-14</td>
<td>32</td>
<td>1.373626</td>
<td>.274725</td>
</tr>
<tr>
<td>35-15</td>
<td>15</td>
<td>.769231</td>
<td>.054945</td>
</tr>
<tr>
<td>35-16</td>
<td>21</td>
<td>1.208791</td>
<td>.109890</td>
</tr>
<tr>
<td>35-17</td>
<td>19</td>
<td>.934066</td>
<td>.164845</td>
</tr>
<tr>
<td>35-18</td>
<td>49</td>
<td>3.791209</td>
<td>.549451</td>
</tr>
<tr>
<td>35-19</td>
<td>42</td>
<td>1.978022</td>
<td>.219780</td>
</tr>
<tr>
<td>35-20</td>
<td>29</td>
<td>2.637363</td>
<td>.329670</td>
</tr>
</tbody>
</table>

Average Time | 1.818970 | .2429153
Table 4.15: Table giving the execution time of Socher's algorithm and PIAP

<table>
<thead>
<tr>
<th>Size 40</th>
<th>Execution time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SOCH</td>
</tr>
<tr>
<td>Prob. No.</td>
<td>PIs</td>
</tr>
<tr>
<td>40-1</td>
<td>33</td>
</tr>
<tr>
<td>40-2</td>
<td>18</td>
</tr>
<tr>
<td>40-3</td>
<td>53</td>
</tr>
<tr>
<td>40-4</td>
<td>44</td>
</tr>
<tr>
<td>40-5</td>
<td>50</td>
</tr>
<tr>
<td>40-6</td>
<td>27</td>
</tr>
<tr>
<td>40-7</td>
<td>24</td>
</tr>
<tr>
<td>40-8</td>
<td>23</td>
</tr>
<tr>
<td>40-9</td>
<td>45</td>
</tr>
<tr>
<td>40-10</td>
<td>47</td>
</tr>
<tr>
<td>40-11</td>
<td>37</td>
</tr>
<tr>
<td>40-12</td>
<td>35</td>
</tr>
<tr>
<td>40-13</td>
<td>29</td>
</tr>
<tr>
<td>40-14</td>
<td>44</td>
</tr>
<tr>
<td>40-15</td>
<td>24</td>
</tr>
<tr>
<td>40-16</td>
<td>46</td>
</tr>
<tr>
<td>40-17</td>
<td>34</td>
</tr>
<tr>
<td>40-18</td>
<td>8</td>
</tr>
<tr>
<td>40-19</td>
<td>34</td>
</tr>
<tr>
<td>40-20</td>
<td>38</td>
</tr>
<tr>
<td>Average Time</td>
<td>2.573654</td>
</tr>
</tbody>
</table>
Table 4.16: Table giving the execution time of Socher’s algorithm and PIAP

<table>
<thead>
<tr>
<th>Prob. No.</th>
<th>PIs</th>
<th>SOCH</th>
<th>PIAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>45-1</td>
<td>56</td>
<td>3.406593</td>
<td>.549451</td>
</tr>
<tr>
<td>45-2</td>
<td>28</td>
<td>2.142857</td>
<td>.439560</td>
</tr>
<tr>
<td>45-3</td>
<td>37</td>
<td>4.725275</td>
<td>.439560</td>
</tr>
<tr>
<td>45-4</td>
<td>30</td>
<td>2.692308</td>
<td>.274725</td>
</tr>
<tr>
<td>45-5</td>
<td>28</td>
<td>4.175824</td>
<td>.274725</td>
</tr>
<tr>
<td>45-6</td>
<td>22</td>
<td>4.230769</td>
<td>.219780</td>
</tr>
<tr>
<td>45-7</td>
<td>43</td>
<td>4.450549</td>
<td>.329670</td>
</tr>
<tr>
<td>45-8</td>
<td>.51</td>
<td>3.846154</td>
<td>.329670</td>
</tr>
<tr>
<td>45-9</td>
<td>45</td>
<td>2.857143</td>
<td>.329670</td>
</tr>
<tr>
<td><strong>Average Time</strong></td>
<td></td>
<td><strong>3.614164</strong></td>
<td><strong>.354090</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Prob. No.</th>
<th>PIs</th>
<th>SOCH</th>
<th>PIAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>50-1</td>
<td>27</td>
<td>4.175824</td>
<td>.274725</td>
</tr>
<tr>
<td>50-2</td>
<td>38</td>
<td>3.516484</td>
<td>.384515</td>
</tr>
<tr>
<td>50-3</td>
<td>11</td>
<td>2.637363</td>
<td>.109890</td>
</tr>
<tr>
<td>50-4</td>
<td>39</td>
<td>3.131868</td>
<td>.384614</td>
</tr>
<tr>
<td>50-5</td>
<td>52</td>
<td>*</td>
<td>.989011</td>
</tr>
<tr>
<td>50-6</td>
<td>23</td>
<td>3.241758</td>
<td>.109870</td>
</tr>
<tr>
<td>50-7</td>
<td>76</td>
<td>*</td>
<td>.879121</td>
</tr>
<tr>
<td><strong>Average Time</strong></td>
<td></td>
<td><strong>3.340660</strong></td>
<td><strong>.252727</strong></td>
</tr>
</tbody>
</table>
Table 4.17: Table giving the # of input clauses and the average execution time with SD and CV, by Socher’s algorithm and PIAP

<table>
<thead>
<tr>
<th>#of PIs</th>
<th>SOCH Mean</th>
<th>SOCH SD</th>
<th>SOCH CV</th>
<th>PIAP Mean</th>
<th>PIAP SD</th>
<th>PIAP CV</th>
<th>DF</th>
<th>paired t</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>.517</td>
<td>.285</td>
<td>.551</td>
<td>.157</td>
<td>.120</td>
<td>.766</td>
<td>19</td>
<td>8.089</td>
<td>&lt; .0005</td>
</tr>
<tr>
<td>20</td>
<td>1.199</td>
<td>1.085</td>
<td>.905</td>
<td>.742</td>
<td>1.162</td>
<td>1.566</td>
<td>17</td>
<td>7.308</td>
<td>&lt; .0005</td>
</tr>
<tr>
<td>25</td>
<td>1.563</td>
<td>1.362</td>
<td>.872</td>
<td>.492</td>
<td>1.099</td>
<td>2.235</td>
<td>19</td>
<td>4.107</td>
<td>&lt; .0005</td>
</tr>
<tr>
<td>30</td>
<td>1.806</td>
<td>.857</td>
<td>.474</td>
<td>.316</td>
<td>.252</td>
<td>.798</td>
<td>19</td>
<td>9.850</td>
<td>&lt; .0005</td>
</tr>
<tr>
<td>35</td>
<td>1.819</td>
<td>.785</td>
<td>.432</td>
<td>.243</td>
<td>.162</td>
<td>.667</td>
<td>18</td>
<td>10.475</td>
<td>&lt; .0005</td>
</tr>
<tr>
<td>40</td>
<td>2.574</td>
<td>1.036</td>
<td>.403</td>
<td>.305</td>
<td>.157</td>
<td>.515</td>
<td>19</td>
<td>10.815</td>
<td>&lt; .0005</td>
</tr>
<tr>
<td>45</td>
<td>3.614</td>
<td>.888</td>
<td>.246</td>
<td>.354</td>
<td>.103</td>
<td>.291</td>
<td>8</td>
<td>11.919</td>
<td>&lt; .0005</td>
</tr>
</tbody>
</table>

Table 4.18: Table giving the # of Prime Paths and the average execution time with SD and CV, by Socher’s algorithm and PIAP

<table>
<thead>
<tr>
<th>#of PIs</th>
<th>SOCH Mean</th>
<th>SOCH SD</th>
<th>SOCH CV</th>
<th>PIAP Mean</th>
<th>PIAP SD</th>
<th>PIAP CV</th>
<th>DF</th>
<th>paired t</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>10-15</td>
<td>.989</td>
<td>.890</td>
<td>.900</td>
<td>.0785</td>
<td>.0294</td>
<td>.374</td>
<td>6</td>
<td>2.785</td>
<td>&lt; .025</td>
</tr>
<tr>
<td>15-20</td>
<td>.780</td>
<td>.599</td>
<td>.767</td>
<td>.114</td>
<td>.089</td>
<td>.764</td>
<td>14</td>
<td>4.401</td>
<td>&lt; .0005</td>
</tr>
<tr>
<td>20-25</td>
<td>1.365</td>
<td>1.069</td>
<td>.782</td>
<td>.215</td>
<td>.166</td>
<td>.775</td>
<td>11</td>
<td>3.913</td>
<td>&lt; .005</td>
</tr>
<tr>
<td>25-30</td>
<td>1.878</td>
<td>1.186</td>
<td>.631</td>
<td>.252</td>
<td>.185</td>
<td>.735</td>
<td>16</td>
<td>6.107</td>
<td>&lt; .0005</td>
</tr>
<tr>
<td>35-40</td>
<td>2.112</td>
<td>1.167</td>
<td>.552</td>
<td>.363</td>
<td>.230</td>
<td>.633</td>
<td>17</td>
<td>6.764</td>
<td>&lt; .0005</td>
</tr>
<tr>
<td>40-45</td>
<td>2.030</td>
<td>1.969</td>
<td>.477</td>
<td>.349</td>
<td>.111</td>
<td>.318</td>
<td>13</td>
<td>6.884</td>
<td>&lt; .0005</td>
</tr>
<tr>
<td>45-50</td>
<td>2.394</td>
<td>1.086</td>
<td>.453</td>
<td>.429</td>
<td>.132</td>
<td>.306</td>
<td>9</td>
<td>6.193</td>
<td>&lt; .0005</td>
</tr>
</tbody>
</table>
Figure 4.7: The graph depicting the execution time taken by Socher's algorithm and PIAP
4.5.3 Experiment 3

While the execution time for the algorithms is machine dependent, the number of paths generated is independent and hence a study on the number of paths generated by both the algorithms for each of the data is also carried out. As already discussed in Chapter 3, and substantiated here by Experiment 1, the number of subsumptions required by PIAP is much less compared to that required in Socher's algorithm. This is so because the number of paths generated by PIAP itself is much less. The comparison of column 3 and column 10 of Table 4.1 to Table 4.7 establishes that the number of paths generated by PIAP is much less compared to that generated by Socher's algorithm. This is because the generation of paths which may be subsumed at a later stage is avoided in PIAP. Hence the paths which are not prime in a bigger set (lower level of the tree) are few in number.

In order to check the efficiency of PIAP, the paired t test is carried out. The results obtained based on the input size and the output size are given in Table 4.19 and Table 4.20, respectively. It is observed (last columns of Table 4.19 and Table 4.20) that the algorithm PIAP is significantly better than Socher's algorithm since $p < .0005$ in most of the cases. The first order regression for the data in Table 4.20 is obtained and is depicted in Figure 4.8. The Table 4.20 and the Figure 4.8 substantiate the claim that PIAP is efficient than Socher's algorithm.

Thus, all the three experiments substantiate the theoretical arguments provided in Chapter 3 regarding the efficiency of PIAP over the Socher's algorithm.
Table 4.19: Table giving the # of input clauses and the average # of paths with SD and CV, by Socher's algorithm and PIAP

<table>
<thead>
<tr>
<th>#of PIs</th>
<th>SOCH</th>
<th>PIAP</th>
<th>DF</th>
<th>paired t</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mean</td>
<td>SD</td>
<td>CV</td>
<td>Mean</td>
<td>SD</td>
</tr>
<tr>
<td>15</td>
<td>100</td>
<td>62</td>
<td>.618</td>
<td>35</td>
<td>22</td>
</tr>
<tr>
<td>20</td>
<td>178</td>
<td>144</td>
<td>.810</td>
<td>54</td>
<td>43</td>
</tr>
<tr>
<td>25</td>
<td>245</td>
<td>219</td>
<td>.890</td>
<td>47</td>
<td>55</td>
</tr>
<tr>
<td>30</td>
<td>285</td>
<td>157</td>
<td>.551</td>
<td>46</td>
<td>24</td>
</tr>
<tr>
<td>35</td>
<td>260</td>
<td>115</td>
<td>.440</td>
<td>33</td>
<td>13</td>
</tr>
<tr>
<td>40</td>
<td>374</td>
<td>150</td>
<td>.401</td>
<td>37</td>
<td>13</td>
</tr>
<tr>
<td>45</td>
<td>515</td>
<td>126</td>
<td>.245</td>
<td>39</td>
<td>11</td>
</tr>
<tr>
<td>50</td>
<td>387</td>
<td>75</td>
<td>.194</td>
<td>35</td>
<td>17</td>
</tr>
</tbody>
</table>

Table 4.20: Table giving the # of prime paths and the average # of paths with SD and CV, by Socher's algorithm and PIAP

<table>
<thead>
<tr>
<th># of PIs</th>
<th>SOCH</th>
<th>PIAP</th>
<th>DF</th>
<th>paired t</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mean</td>
<td>SD</td>
<td>CV</td>
<td>Mean</td>
<td>SD</td>
</tr>
<tr>
<td>10-15</td>
<td>108</td>
<td>92.734</td>
<td>.862</td>
<td>14</td>
<td>2.326</td>
</tr>
<tr>
<td>15-20</td>
<td>129</td>
<td>115.998</td>
<td>.901</td>
<td>19</td>
<td>2.154</td>
</tr>
<tr>
<td>20-25</td>
<td>194</td>
<td>143.829</td>
<td>.742</td>
<td>27</td>
<td>6.171</td>
</tr>
<tr>
<td>25-30</td>
<td>255</td>
<td>156.314</td>
<td>.614</td>
<td>32</td>
<td>4.671</td>
</tr>
<tr>
<td>30-35</td>
<td>213</td>
<td>124.645</td>
<td>.584</td>
<td>37</td>
<td>5.284</td>
</tr>
<tr>
<td>35-40</td>
<td>336</td>
<td>137.803</td>
<td>.411</td>
<td>44</td>
<td>7.499</td>
</tr>
<tr>
<td>40-45</td>
<td>321</td>
<td>145.631</td>
<td>.454</td>
<td>48</td>
<td>5.919</td>
</tr>
<tr>
<td>45-50</td>
<td>386</td>
<td>161.547</td>
<td>.418</td>
<td>62</td>
<td>14.774</td>
</tr>
</tbody>
</table>
4.6 Editing of Tree

When the reasoner transmits the set $\mathcal{H}$ of clauses to the RMS there is the need to update the RMS database. In order to accomplish this, the binary tree corresponding to the clauses that are already transmitted to the RMS has to be modified. Addition and deletion of clauses and the changes thereof in the binary tree are being discussed in this section. Updating RMS database means that the prime paths for the updated database has to be obtained. The incremental computation of prime paths using the tree structure is also discussed in this section.
4.6.1 Addition of a clause

The need for updating a formula, and the incremental algorithm to update the compiled knowledge base has been discussed in earlier sections. It has already been discussed that the existing algorithms [Kean 90, Jackson 92, Socher 91] does not incorporate efficiently the set \( H \) of clauses transmitted by the reasoner. If \( T(\mathcal{F}) \) denotes the binary tree corresponding to the formula \( \mathcal{F} \), then the problem is to obtain \( T(\mathcal{F} \land H) \) making use of \( T(\mathcal{F}) \). The different ways of obtaining \( T(\mathcal{F} \land H) \) are discussed here.

Method 1:-

In this method, the set \( H \) is partitioned into \( H_r \) and \( H'_r \) with respect to the label \( r \) of the root node of the tree \( T(\mathcal{F}) \). The left subtree \( T(\mathcal{F}') \) of the root node is updated with those clauses \( (H'_r) \) in \( H \) which do not contain \( r \). The right subtree \( T(\mathcal{F}_r) \) of the root node is updated with the subset \( (H_r) \) of \( H \) obtained by removing the literal \( r \) from the clauses in \( H \) containing \( r \). This process is continued until leaf node is reached. If the leaf node of the tree \( T(\mathcal{F}) \) is to be updated with a nonempty subset of \( H \), then the node is no more a leaf node, and hence the corresponding formula is split until further partitioning is not possible. The pseudo-code to update a tree \( T(\mathcal{F}) \) and an example are given below.

\[
\text{Procedure } \text{Update1-Tree}(T(\mathcal{F}), H) \\
\text{Initialize root of } T(\mathcal{F} \land H) \text{ as root of } T(\mathcal{F}). \\
r = \text{the label of the root of } T(\mathcal{F}). \\
\text{Compute } H'_r = \{ h \in H \setminus r \notin h \} \\
\text{Compute } H_r = \{ h - r \setminus b \in H, r \notin h \} \\
\text{If } T(\mathcal{F}') \text{ is NULL, then} \\
\hspace{1cm} T(\mathcal{F}' \land H'_r) = \text{CREATE TREE }(H'_r). \\
\text{else}
\]
Chapter 4. THEE AND REASON MAINTENANCE

Update1_Tree \((T(F_r), H_r)\)

If \(T(F_r)\) is NULL, then
\[ T(F_r H_r) = \text{CREATE_TREE} (H_r). \]

else
\[ \text{Update1_Tree} (T(F_r), W_r) \]

END

Example 4.6.1:-

Let \(H = \{acef, adebf, \text{\textbar}acef, \text{\textbar}aceg\} \) be the formula to be appended to the formula \(F = \{abcde, abcdfg, \text{\textbar}ace/, abcdf, abcdg\} \). The set \(H\) is incorporated at the root of the tree \(T(F)\). This is depicted in Figure 4.9.

The clauses in the box has to be appended to the node to which the arrow points. The literal \(a\) is the label of the root of the tree \(T(F)\) and this literal is present in some clauses of \(H\). Hence \(H_a = \{bce, adebf, abedf\}\) is to be appended to the right subtree of node 1 (root node) in Figure 4.10. The remaining clauses, \(H_a' = \{\text{\textbar}acef, \text{\textbar}aceg\} \) are to be appended to the left subtree (node 2) of node 1 (Figure 4.10). The node 2 of \(T(F)\) is NULL and hence, the tree for the formula \(H_a'\) is created. One of the trees thus obtained is given in Figure 4.11. In order to update the node 3 of \(T(F)\), the right subtree of node 3 has to be updated with \(\{cef\}\) and the left subtree is updated with \(\{dfg, bf\}\). This process is continued and one of the complete binary tree thus constructed for \(T(F \land H)\) is the same as the tree already given in Figure 4.1. It is to be noted that in this method, if \(H\) has any literal which is not present in \(F\), then these literals become the labels of nodes only when a leaf node or NULL node of \(T(F)\) is reached.

A simple case of the addition of the formula is when the formula has to be updated by a single clause \(h\). For updating the binary tree, if the label of the root is present in \(h\) then the right child is updated by the clause \(h \rightarrow \{\text{\textbar}label\} \). On the other hand, if the label

\[ \text{There are different trees depending on the choice of the label} \]
of the node is not present in $h$ then the left child is updated by the clause $h$.

Figure 4.9: Incorporation of $\mathcal{H}$ to the root of $T(F)$

Figure 4.10: Incorporation of $\mathcal{H}_a$ and $\mathcal{H}_b'$ to the right child and left child of the root node of $T(F)$. 
Figure 4.11: Binary tree after incorporation of $\mathcal{H}_a$ to $T(F_a)$

Method 2:-

The main aim of tree-representation of $T(F \land \mathcal{H})$ is to facilitate computation of prime implicants incrementally as and when the reasoner transmits clauses. The Method 1 used to construct $T(F \land \mathcal{H})$ is not suitable for the incremental computation of prime paths for the simple reason that in this method, the prime paths already available for $T(F)$ are not made use of. Moreover, Method 1 of appending $\mathcal{H}$ to $F$ depends on $T(F)$ and requires updating of most of the nodes in it. Another method is proposed here to construct $T(F \land \mathcal{H})$ keeping $T(F)$ intact. In this method, the binary tree $T(\mathcal{H})$ for the formula $\mathcal{H}$ is constructed independent of $T(F)$. The binary tree for $T(F \land \mathcal{H})$ is created with $T(\mathcal{H})$ as the right subtree and $T(F)$ as the left subtree with the root node being dummy, if $F$ and $\mathcal{H}$ are defined over the same set of literals. If there is any literal in $\mathcal{H}$ foreign to $\mathcal{L}$, the set of literals, then the root of $T(F \land \mathcal{H})$ is not dummy. If the root node is dummy, a dummy literal (say, $x$) is considered the label of the root node of $T(F \land \mathcal{H})$. In this case, it is not necessary to consider the tree corresponding to the additional formula as the...
right subtree of the new root node created. However, if the root node is not dummy, any of the literals foreign to $\xi$ is considered the label of the root node of $T(\mathcal{F} \land \mathcal{H})$. In this case, the right subtree of $T(\mathcal{F} \land \mathcal{H})$ must necessarily be $T(\mathcal{H})$. The binary tree obtained by this method for the Example 4.6.1 is given in Figure 4.12, where the right subtree represents the formula $\mathcal{H}$. The particular case where $\mathcal{H}$ is a single disjunctive clause is so simple; the new clause becomes the right child leaf node of the dummy node.

Figure 4.12: Tree-representation of formula $\mathcal{F} \land \mathcal{F}$ by Method 2

$\mathcal{F} = \{abcde, abcdfg, abf, abcdf, abcdg\}$

$\mathcal{H} = \{abcef, adfg, abf, \overline{a}ce, \overline{a}ceg\}$
Method 3:-

This method is the combination of Method 1 and Method 2. The tree $T(\mathcal{H})$ is constructed separately as in Method 2 whereas the literals chosen for partitioning are the same as those in $T(\mathcal{F})$. i.e., if $r$ is the label of the root of $T(\mathcal{F})$, then the same $r$ is the label of the root node of $T(\mathcal{H})$. In this process, at any level of the tree the corresponding nodes of both trees will have the same label. The tree thus constructed for the Example 4.6.1 is given in Figure 4.13. It can be seen that the labels of the corresponding nodes of the trees are the same.

$\mathcal{F} = \{abde, abdfg, abef, abcdg\}$.

$\mathcal{H} = \{abce, adfg, abf, ââef, âceg\}$
The advantages of Method 3 over Method 1 and Method 2 are explained below. In Figure 4.13, let \( A \) and \( B \) are the left and right child, respectively of the root node \( N \) with label \( a \) representing \( \mathcal{F} \). Similarly, let \( A' \) and \( B' \) be the left" and right child of the root node \( N' \) representing \( \mathcal{H} \). Since the label of the nodes \( N \) and \( N' \) are the same, it can be visualized as branching from a single node (say, \( NN' \)). i.e., the nodes \( A \) and \( A' \) can be visualized as the left child of a single node \( NN' \). Similarly, \( B \) and \( B' \) is the right child of the node \( NN' \). Further, \( A \) and \( A' \) itself can be visualized as the left and right child of a dummy node (Figure 4.14). Similarly the nodes \( B \) and \( B' \) form the left and right child of another dummy node. In this visualization, the root node label of \( T(\mathcal{F} A \mathcal{H}) \) is the same as that of \( T(\mathcal{F}) \) and \( T(\mathcal{H}) \). This visualization helps in the incremental computation of prime paths which is discussed in Section 4.6.3.

![Figure 4.14: Visualization of nodes A, A' (the left child nodes) and B, B' (the right child nodes) of two nodes with same label at same level of tree as the left child and right child, respectively of a dummy node](image-url)

**Figure 4.14:** Visualization of nodes \( A, A' \) (the left child nodes) and \( B, B' \) (the right child nodes) of two nodes with same label at same level of tree as the left child and right child, respectively of a dummy node.
4.6.2 **Deletion of a clause**

Different methods of updating a tree has been discussed in the above section. Deletion of a clause is equally important as the reasoner might wish to transmit a clause temporarily and later delete it. Deleting a clause from the binary tree corresponding to a formula is the issue in this subsection.

If a set of clauses has to be deleted from a formula, choose the corresponding columns in the binary matrix representation of the formula. If these columns have 1 in the row (label) associated to the node representing the formula, then delete the chosen columns from the right child of the node; otherwise, delete the chosen columns from the left child of the node.

The deletion of clauses is easy if Method 2 or Method 3 is followed for updating a tree. If the reasoner transmits a clause temporarily and later wish to delete it, it is only required to delete the right subtree (since this corresponds to the formula temporarily added) of the root of the binary tree.

4.6.3 **Incremental computation of prime paths**

It is already mentioned in Section 3.4.4 that the algorithm PIAP is well suited for computing the prime implicants incrementally. We have also seen that when the reasoner transmits a set $ft$ of clauses, there are different ways of obtaining the binary tree representation of the formula $\mathcal{F} \land \mathcal{H}$. The computation of prime paths in each of these cases is discussed here.

**Case 1:** Tree is *updated using Method 1:*- In this case the prime paths for the new tree representing $\mathcal{F} \land \mathcal{H}$ is obtained exactly following the steps in PIAP. This method is definitely not efficient as it does not make use of the prime paths already computed for a portion of the submatrix representing each of the nodes.
Case 2: \( \mathcal{H} \) is defined over \( \mathcal{L} \) and \( T(\mathcal{H}) \) is constructed by Method 2: - In this case the binary tree for \( \mathcal{F} \) is kept intact and so is its set of prime paths. The prime paths for the formula \( \mathcal{H} \) are computed independently using PIAP. Since the root of the new tree constructed has dummy label the prime paths for the formula \( \mathcal{F} \) A \( \mathcal{H} \) is computed following PIAP except Step 4. Moreover, since the label of the root is dummy, it is immaterial whether \( T(\mathcal{F}) \) is the left or right child.

Case 3: \( \mathcal{H} \) is defined over \( \mathcal{L} \) and \( T(\mathcal{H}) \) is constructed by Method 3. It is already discussed that in this case, the two trees \( T(\mathcal{F}) \) and \( T(\mathcal{H}) \) can be merged together. This visualization makes it easy to compute the prime paths of \( \mathcal{F} \) A \( \mathcal{H} \) from the prime paths of \( \mathcal{F} \) and \( \mathcal{H} \), where \( \mathcal{F} \) and \( \mathcal{H} \) are the subformulae of \( \mathcal{F} \) and \( \mathcal{H} \), respectively. Thus, the prime paths of any node in the binary tree can be computed incrementally, making use of the prime paths already computed for the corresponding node in the original formula. Each of the intermediate node can be considered a tree with a dummy root node (as in the case of Method 2) and hence the prime paths of each of the intermediate node can be computed incrementally using the method adopted for Case 2.

Case 4: \( \mathcal{H} \) has some literal (say, \( r \)) foreign to \( \mathcal{C} \) and \( T(\mathcal{H}) \) is constructed by Method 2 or Method 3: - In this case also the binary tree for \( \mathcal{F} \) is kept intact and so is its set of prime paths and the prime paths for the formula \( \mathcal{H} \) are computed independently using PIAP. The root of the tree \( T(\mathcal{F} \) A \( \mathcal{H} \) has the literal \( r \) (any one, in the case of more than one) as the label and hence the prime paths for the formula \( \mathcal{F} \) A \( \mathcal{H} \) are computed exactly following PIAP with \( T(\mathcal{F}) \) as the left child and \( T(\mathcal{H}) \) as the right child. Further, Case 2 or Case 3 is applicable if \( T(\mathcal{H}) \) is constructed using Method 2 or Method 3, respectively.
4.7 Conclusion

In this chapter, a new binary tree representation of a formula is introduced. This representation naturally evolved from the partitioning scheme involved in PIAP proposed in Chapter 3. The basic structure of a node in the tree, the creation of the tree, and the implementation details of PIAP are also discussed in this chapter. Different experiments were carried out to compare the performance of PIAP and Socher's algorithm. The experimental results reported in this chapter substantiate positively that the algorithm PIAP is very efficient compared to Socher's algorithm. Apart from being efficient in sequential computation of prime implicants, PIAP has many other advantages. The binary tree representation is well suited for representing the updated knowledge as well as for compiling the knowledge incrementally. Different methods to accomplish this are also discussed in this chapter. The compiling method is global in the sense that all the clauses in the knowledge-base are treated collectively, and the newly-added information which can be a set of clauses, is also treated collectively for incremental computation. Further, unlike the earlier algorithms which are inherently sequential, the proposed algorithm is naturally parallelizable. The parallel algorithm is described in Chapter 5. Moreover, the tree structure is utilized to have a full-fledged RMS.

In a nut-shell form, the advantages of PIAP are:

- Number of subsumptions, execution time and the number of candidate paths for subsumption are less compared to Socher's algorithm.
- Well suited for knowledge compilation in global as well as incremental mode.
- The algorithm is naturally parallelizable.