Chapter 5

Grammar based Recognition of Human Activities

5.1 Introduction

In Chapter 4, we presented TAG, a graph-based representation of human activities. The TAG representation keeps track of how qualitative spatial relations between extended objects change over time during an activity. The activities are then classified using an TAG-kernel based SVM classifier. In this chapter, we discuss a grammar-based generative learning mechanism to model the underlying structure of a class of activities represented as TAGs.

In literature, there are reports of work by researchers wherein grammars are used for encoding the recursive and hierarchical nature of human activities [52, 54]. However, another feature of human activities is that they may involve several concurrently occurring sub-events. For example, raising both hands would involve the simultaneous sub-events of raising the left hand and raising the right hand. To handle such complications, most grammar-based recognition of human activities reported in literature use complex parsing mechanisms [52] or complex grammar structures [54]. A *Temporal Activity Graph Grammar* is presented in this chapter that exploits the temporal sequencing and graph structure of TAGs to deal with the complexities of modelling human activities. Further, a Temporal Activity Graph Grammar induction algorithm is proposed; the induction algorithm can be used to learn the TAG grammar rules for an activity class given a set of positive examples of the class described as TAGs. The *Temporal Activity Graph Grammar* is a probabilistic context free graph grammar for TAGs.

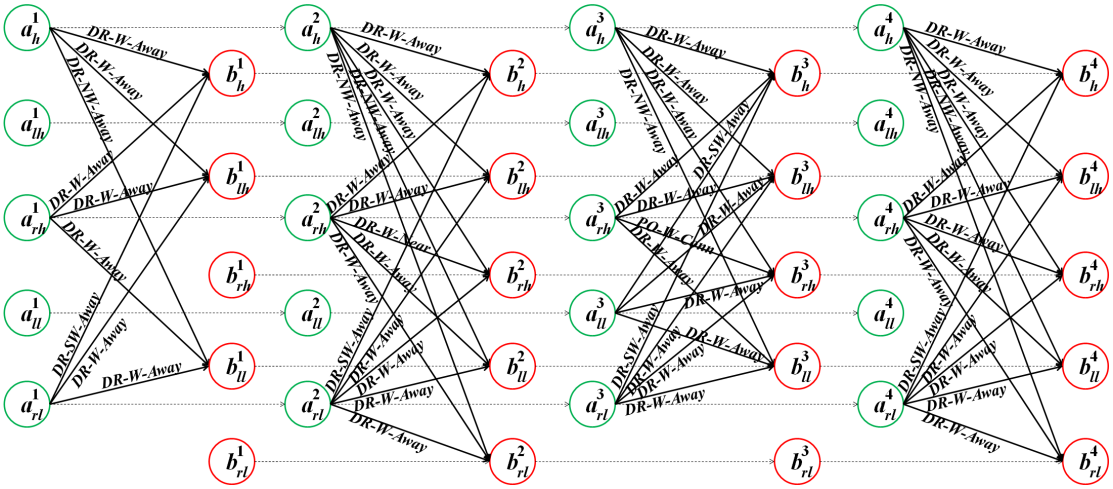
5.2 Temporal Activity Graph Grammar

In Chapter 4, TAGs were used for representation of human activities. An activity involving a pair of entities (humans or objects) is seen as a sequence of relations between them. The interacting entities are abstracted as extended objects based on the data available from the underlying tracking system. The TAG captures such an activity as a sequence of temporal snapshots. Each snapshot is represented as a subgraph that captures the spatial relational configuration of the extended objects at a specific time-point. The vertices in the subgraph correspond to components of the extended objects; vertices may be labeled depending on the availability of the skeletal information from the underlying tracking mechanism. The edges between vertices of such a subgraph are the spatial edges; spatial edges are labeled using the qualitative spatial relations between the corresponding components computed using Extended CORE9 (see Chapter 3). The sequence of subgraphs linked using temporal edges constitute a TAG.

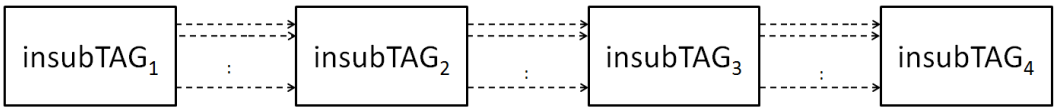
The TAGs can be easily seen as a temporal sequence of *instantaneous* TAG *subgraphs* as shown in Figure 5-1. An *instantaneous* TAG *subgraph* or *insubTAG* is a temporal snapshot of the TAG, such that it is a subgraph containing all vertices corresponding to a single time-point t (see Definition 4.2 in Chapter 4). Every TAG can be re-written as a *string* of insubTAGs assuming that there is a set of temporal edges connecting the vertices in insubTAG at time $t-1$ to the vertices in insubTAG at time t. Figure 5-1a is an instance of a TAG that represents an activity. This TAG can be seen as the sequence of insubTAGs as shown in Figure 5-1b; the dashed edges between the insubTAGs corresponds to the temporal edges in the corresponding TAG. In Figure 5-1b, insubTAG$_t$ correspond to a subgraph of the TAG in Figure 5-1a. The insubTAG$_t$ includes vertices a_x^t, b_y^t and all spatial edges between these vertices that are also in the TAG, such that $x, y \in \{h, lh, rh, ll, rl\}$.

The TAG grammar is a *probabilistic context free grammar* with elements of *node replacement graph grammar*. As discussed in Chapter 2 (see Section 2.8.1) a context free grammar (CFG) [98] consists of four elements - (a) a set of non-terminals, Δ, (b) a set of terminals, Σ, (c) a set of rules, R, and (d) a start non-terminal, S. A probabilistic context free grammar (CFG) is a CFG in which every rule, $A \rightarrow \alpha$, is associated with a probability of the rule being used to replace A in a derivation.

A node-replacement graph grammar, is a grammar defined over undirected graphs [126]. Similar to string grammars, it is a four-tuple of $\langle \Delta, \Sigma, R, S \rangle$. Δ, Σ,



(a)



(b)

Figure 5-1: (a) TAG representing an activity (b) Illustration of the TAG in Figure 5-1a as sequence of insubTAGs

R and S have the usual interpretations. However, terminals and non-terminals correspond to labels on graph vertices. Each rule is of the form $X \rightarrow D$ where X is a non-terminal and D is graph consisting of vertices labeled with terminals and non-terminals (see Section 2.8.3 of Chapter 2). Further a connection relation is associated with every production rule that defines how a daughter graph is embedded in the mother graph.

TAG grammar takes advantage of both string grammars and graph grammars. Due to the temporal sequential structure of TAGs, they are much similar to strings. This allows one to take advantage of the simplicity of string grammars. However, unlike a string, a TAG is a sequence of subgraphs, or more specifically, a sequence of insubTAGs. Therefore, a TAG is rewritten as a string of smaller graphs or insubTAGs. The terminals in the TAG grammar correspond to insubTAGs. The process of generating a strings using rules of a grammar is called a *derivation*. Within the TAG Grammar, the rules generate sequences of insubTAGs which are subgraphs of the TAG. The question arises of how these insubTAGs are *connected* to one another during a derivation. Therefore, it is essential for the TAG grammar to have *connection rules* or *embeddings*, similar to node replacement grammars (see Section 2.8.3 of Chapter 2).

In regular string grammars, a rule is of the form $A \rightarrow \alpha$ where α is a string

of terminals and non-terminals. In TAG grammars, α is restricted to either a terminal (i.e. a single insubTAG) or a sequence of non-terminals. Such a restriction ensures that that TAG grammar is always in a semi-Chomsky Normal Form[1]. It allows for several simplifications in the algorithm for induction of grammar rules (see Section 5.3).

In traditional node-replacement graph grammars, terminals and non-terminals are labels on the *nodes* or *vertices*. In such grammars, the right-hand side of rules define the vertices and edges of the generated graph. Whereas in a TAG, there are two types of edges: *spatial edges* and *temporal edges*. By defining the terminals to be insubTAGs, it is ensured that vertices and spatial edges of the generated TAG are defined by terminals. The temporal edges of the generated TAG are defined by the right-hand side of the rules and the associated *connection rules*. Furthermore, every rule has a probability associated with it similar to PCFGs. The formal definition of a TAG grammar is as follows.

Definition 5.1. *A **TAG grammar**, $\mathbb{G}$, is defined as a five-tuple $\langle \Delta, \Sigma, R, S, Pr \rangle$, where*

- Δ *is the set of non-terminals*
- Σ *is the set of terminals. Here, a terminal is an* insubTAG.
- R *is the set of rules such that each $R_i \in R$ can be of two types:*
  - *A terminal rule that is of the form $A \rightarrow a$ such that $A \in \Delta$ and $a \in \Sigma$*
  - *A production rule that is of the form $A \rightarrow \alpha$ such that $A \in \Delta$ and $\alpha \in \Delta^*$*

  Every rule in R is associated with a connection rule. A connection rule defines the set of temporal edges in the graph generated by some rule $R_i \in R$.
- S *is the start variable, $S \in \Delta$*
- $Pr : R \rightarrow [0, 1]$ *is function that gives the probability of applying rule $R_i : A \rightarrow \alpha$ to replace variable $A \in \Delta$ in a derivation,*

$$Pr(A \rightarrow \alpha \geq 0) \quad \textit{and} \quad \sum_{\alpha} Pr(A \rightarrow \alpha) = 1$$

[1] A rule in a grammar that is in Chomsky Normal Form is either of the form $A \rightarrow a$ or $A \rightarrow BC$. Here, $A, B, C \in \Delta$ and $a \in \Sigma$. The TAG grammar is said to be in a semi-Chomsky Normal Form because all rules are of the form $A \rightarrow a$ or $A \rightarrow A_1 A_2 ... A_n$, where $A, A_1, A_2 ... A_n \in \Delta$ and $a \in \Sigma$

The terms *terminal rule* and *production rule* are used to distinguish between rules in order to enable specialized procedures. Such distinction of the rules is not normally seen in literature of context free grammars or graph grammars. In this work, this distinction is necessary because in the grammar induction algorithm the two types rules are dealt using separate specialized procedures (see Section 5.3, Algorithm 2). It is to be noted that if R_t is the set of terminal rules and R_p is the set of production rules then,

$$R = R_t \cup R_p$$

From Definition 4.2, a vertex in a TAG is labelled as a_i^k if it corresponds to the i^{th} component of the extended objects, A, involved in the activity at time point k. Say a sequence of insubTAGs, $g^1 g^2 ... g^k g^{k+1} ...$, is generated by some TAG grammar. In this context, the nomenclature a_i^k can be extended to be understood as the i^{th} component of the extended object A in the k^{th} insubTAG of the sequence.

Connection Rules: In node replacement grammar, a connection rule defines how a subgraph is connected to another subgraph in a derivation. In a TAG grammar, the set of connection rules is based on the labels on the vertices obtained using the skeletal information. For this work, we assume that the skeletal information of the extended objects involved in an activity is known and the vertices in the TAG are labeled accordingly (see illustrative example in Section 4.3.5, Chapter 4). Further, a connection rule defines the temporal edges between a pair of consecutive insubTAGs in a sequence generated by the grammar. Such a connection rule, can be generalized using Equation 5.1 under the following assumptions.

1. The TAGs, and thereby the insubTAGs, involve two extended objects A and B such that all spatial edges are directed from components of A to components of B.

2. The skeletal information of the extended objects are known. Further, the i^{th} component of A correspond to the same skeletal part as the i^{th} component of B. For example: if the i^{th} component of A correspond to the left hand of A then the i^{th} component of B correspond to the left hand of B.

In this thesis, activities involving humans are dealt with. Further, part-based tracking of humans are available in literature [31] that recognize individual skeletal parts. Thus, within the scope of our discussion, the assumptions stated above are easily met. As such, the connection rule between any pair of insubTAGs $g^k g^{k+1}$ is

the following set of temporal edges,

$$\{(a_i^k, a_i^{k+1}), (b_i^k, b_i^{k+1}) | 1 \leq i < n\} \tag{5.1}$$

Here, n is the maximum of the number of components in any of the involved extended objects.

Lemma 5.1. *The TAG grammar is a probabilistic context free grammar.*

Proof. In a probabilistic context free grammar $\langle \Delta, \Sigma, R, S, Pr \rangle$, all rules in R are of the form $A \rightarrow \gamma$ where $A \in \Delta,\ \gamma \in (\Sigma \cup \Delta)^*$. Further, the function Pr assigns probability to the rules such that sum of the probabilities of all rules with some non-terminal A on the LHS is 1.

According to Definition 5.1, a TAG grammar has the same components as a probabilistic context free grammar, except that the all terminals in Σ are insub-TAGs. TAG grammar rules can be of two types: $A \rightarrow a$ and $A \rightarrow \alpha$ where $A \in \Delta,\ a \in \Sigma,\ \alpha \in \Delta^*$. However, $\Sigma \subset (\Sigma \cup \Delta)^*$ and $\Delta^* \subset (\Sigma \cup \Delta)^*$. That is, the RHS of the rules have the same structure as a probabilistic context free grammar. Further, the LHS of every rule is a single non-terminal $A \in \Delta$. This suggests that the TAG grammar has the *context-free* property, i.e. replacement of a non-terminal A in a derivation does not depend on the context in which A appears. Furthermore, the probability function defined for the TAG grammar uses the same definition as in a probabilistic context free grammar. Therefore, a TAG grammar is a probabilistic context free grammar. □

5.3 TAG Grammar Induction

The intuition behind defining a TAG grammar is to induce a grammar to model an activity class. Consequently, such a grammar can be used to classify an activity (represented as a TAG) to the activity class it belongs based on whether the corresponding grammar recognizes it. However, such an idea is feasible only if some grammar induction algorithm is available to learn the grammar rules from a set of examples. A TAG grammar induction algorithm is defined in this section.

While graph grammar induction algorithms have been proposed in literature, such algorithms are usually aimed at finding common substructures within general graphs [127]. In order to find common substructures within a set of graphs such algorithms have to rely on computationally expensive procedures such as graph isomorphism. The uniform substructures and temporal sequencing of TAGs make it possible to simplify the grammar induction algorithm.

The proposed algorithm does the following: Given a set, Γ, of positive examples of activities in the form of TAGs for some activity class C, a TAG grammar $\mathbb{G}_C = \langle \Delta, \Sigma, R, S, Pr \rangle$ is induced. To some extent, the algorithm for inducing $\mathbb{G}_C$ is inspired by probabilistic context free grammar induction algorithm for regular strings [128].

To begin with, each activity $X \in \Gamma$ represented as a TAG, is rewritten as a sequence of insubTAGs. Say $X = (g^1, g^2, ...g^n)$ where n is the number of time points in X. While generating the sequences, if an insubTAG, g^k, is encountered for the first time, create a new non-terminal T and a new terminal rule for the form

$$T \rightarrow g^k$$

The insubTAG sequence is then transformed to a sequence of non-terminals such that every insubTAG in the sequence is replaced by the corresponding non-terminal. After the TAG X is converted to a sequence of non-terminals $(T_1 T_2 ... T_n)$, create a new production rule of the form

$$P \rightarrow T_1 T_2 ... T_n$$

This is repeated for all $X \in \Gamma$, until all the TAGs are converted to a sequence of non-terminals and individual production rules are generated for each such sequence of non-terminals. This provides the base set of terminals (Σ), non-terminals (Δ), terminal rules (R_t) and production rules (R_p). The probability corresponding to every terminal rule and production rule is initially set to 1. This base TAG grammar is then made compact by finding repetitive patterns during interleaved *chunking* and *merging* operations. The TAG induction algorithm is presented in Algorithm 2.

The *chunking* and *merging* of terminal and production rules[2] is repeated in an interleaved fashion until the no more rules can be chunked or merged. The *chunking* and *merging* operations are discussed in detail in Sections 5.3.1, 5.3.2, and 5.3.3 respectively.

From Definition 5.1, terminal rules are of the form $A \rightarrow a$ where $A \in \Delta$ and $a \in \Sigma$ whereas production rules are of the form $A \rightarrow \alpha$ where $A \in \Delta$ and $\alpha \in \Delta^*$. The separation of the rule set into terminal rules and production rules is because the right hand side of terminal rules in a TAG grammar have a uniform graph structure. On the other hand the right hand side of production rules are a

[2]The set R is the combined set of terminal rules and production rules

Algorithm 2: *InduceTAGGrammar*(Γ), Algorithm to induce TAG Grammar $\mathbb{G}$ given a set Γ of n TAGs as positive examples

Input: Γ; `The set of` n `positive examples`
Output: $\mathbb{G} = \langle \Delta, \Sigma, R, S, Pr \rangle$; `Induced TAG Grammar`
begin
  Initialize $\Delta = \{S\}$, $\Sigma = \phi$, $R = \phi$
  forall $X = (g^1 g^2 ... g^n) \in \Gamma$ **do**
    Create a new non-terminal P
    $\Delta = \Delta \cup \{P\}$
    forall $1 <= i <= n$ **do**
      if $(T \rightarrow g^i) \notin R_t$ **then**
        $\Sigma \leftarrow \Sigma \cup \{g^i\}$; `Include the insubTAG` g^i `in` Σ
        $\Delta \leftarrow \Delta \cup \{T\}$; `Create a new non-terminal` T
        $R \leftarrow R \cup \{T \rightarrow g^i\}$; `Create a new terminal rule`
        $Pr(T \rightarrow g^i) \leftarrow 1$
      $T_i \leftarrow T$
    $R \leftarrow R \cup \{P \rightarrow T_1 T_2 ... T_n\}$; `Create a new production rule`
    $Pr(P \rightarrow T_1 T_2 ... T_n) = 1$; `Initialize probabilities to 1`
    $R \leftarrow R \cup \{S \rightarrow P\}$
    $Pr(S \rightarrow P) \leftarrow 1/n$; `Assuming equal distribution of priors`
  Merge terminal rules
  $\mathbb{G} \leftarrow$ *InterleavedChunkMergeProcedure*($\mathbb{G}$) ; `Apply Algorithm 3`
  return $\mathbb{G}$

sequence of non-terminals and have string-like structure. This allows us to define specialized procedures for handling merging of the graph based structures and takes advantage of the sequential structure of TAGs.

5.3.1 Merging Terminal Rules

In traditional graph-grammar induction algorithms, repeating patterns are usually found by checking for graph isomorphisms[129]. Isomorphic sub-graphs are replaced by variables to form new rules. This is simplified in TAG grammar induction because graph structures are restricted to the terminals. Further, insubTAGs that are terminals in a TAG grammar have a uniform structure with differences in the edge labels. This reduces the problem of finding graph isomorphisms to finding similarity in edge labels. For the scope of human activity recognition as discussed within this thesis, it is assumed that TAGs are used to describe an activity involving two entities; consequently insubTAGs can be seen as bipartite graphs.

A pair of terminal rules $T_p \rightarrow g$ and $T_q \rightarrow h$ can be merged based on certain conditions on the insubTAGs g and h. It is to be noted that g and h are

represented using a *reduced* adjacency matrix. Since insubTAGs are essentially bipartite graphs, a *reduced* adjacency matrix has vertices corresponding to one extended object along the rows and the other extended object along the columns. The element $g(i,j)$ corresponds to the label of the edge from i^{th} component of the first extended object to the j^{th} component of the second extended object. Further, The $NULL$ edge label appears whenever the edge label between the i^{th} component of the first extended object and the j^{th} component of the second extended object is not known or is missing. Edge labels are computed using Extended CORE9. However, edge labels may not be computed if one of the components is occluded or is missing from video at a particular instant of time (see illustrative example in Chapter 4, Section 4.3.5)

The pair of terminal rules $T_p \rightarrow g$ and $T_q \rightarrow h$ can be merged if any of the following two conditions hold.

Case 1: g and h differ by a single edge label i.e., $g(i,j) \neq h(i,j) \neq NULL$ and $g(i,j), h(i,j) \sqsubseteq \varepsilon$ and $\varepsilon \neq \top$ in the subsumption hierarchy (see Section 2.4.2 of Chapter 2). To handle this case, a new terminal insubTAG g' is created such that for the non-matching edge labels of g and h, $g'(i,j) = \varepsilon$. For all the matching edge labels of g and h, g' also has the same edge labels.

Case 2: g and h differ only by $NULL$ edge labels. That is,
$\forall i,j, (g(i,j) \neq h(i,j)) \Rightarrow (g(i,j) = NULL \vee h(i,j) = NULL)$
To handle this case, a new terminal insubTAG g' is created such that for the non-matching edge labels of g and h, $g'(i,j) = \varepsilon \vee NULL$. Here, $\varepsilon = g(i,j) \vee \varepsilon = h(i,j)$. For all the matching edge labels of g and h, g' also has the same edge labels.

If either of the two aforementioned conditions are met, a new non-terminal T_s is created and T_p and T_q are excluded from the set of non-terminals. Rules $T_p \rightarrow g$ and $T_q \rightarrow h$ are merged to form a new rule $T_s \rightarrow g'$ where g' is an insubTAG that can be said to *cover* both g and h. If $T_p \rightarrow g$ and $T_q \rightarrow h$ can be merged, then for some grammar $\mathbb{G} = \langle \Delta, \Sigma, R, S, Pr \rangle$ the following modifications are applied,

$$\Delta \leftarrow \Delta \cup \{T_s\} \setminus \{T_p, T_q\} \tag{5.2}$$

$$\Sigma \leftarrow \Sigma \cup \{g'\} \setminus \{g, h\} \tag{5.3}$$

$$R \leftarrow R \cup \{(T_s \rightarrow g')\} \setminus \{(T_p \rightarrow g), (T_q \rightarrow h)\} \tag{5.4}$$

$$Pr(T_s \rightarrow g') \leftarrow 1 \tag{5.5}$$

As a consequence of these changes, the non-terminals T_p and T_q should not appear anywhere in R. Therefore, all occurrences of T_p and T_q in any production rule of R are replaced by T_s. The merging of terminal rules is repeated until no more terminal rules can be merged.

5.3.2 Chunking Production Rules

In order to reduce the number of rules in the grammar, the production rules have to be merged. The merging process is effective when it is interleaved with a *chunking* process. At this stage of the TAG grammar induction, only production rules are considered. As discussed before, the right hand side of production rules can be easily treated as a *string* of non-terminals. This makes it possible to incorporate elements of string-grammar induction algorithms.

Chunking involves finding recurrent subsequences of non-terminals and abbreviating them using a single non-terminal as done for the string-grammar induction algorithm in [128]. The effect of the chunking process when interleaved with merging of production rules is two-fold:

1. Finding recurrent patterns within the grammar.

2. Reducing the size of the grammar and effectively expanding the scope of the grammar, i.e, the resulting grammar generates more than its predecessor.

In order to detect recurrent patterns of non-terminals, a bigram transition probability is computed for the non-terminals. For a grammar $\mathbb{G} = \langle \Delta, \Sigma, R, S, Pr \rangle$, the bigram transition probability matrix M is an $n \times n$ matrix such that $n = |\Delta|$. The matrix element $M[i, j]$ gives the transition probability of going from the i^{th} non-terminal, R_i, to the j^{th} non-terminal, R_j. After computing the bigram transition probability matrix M, find the index $[i, j]$ corresponding to the highest value in the matrix. If the subsequence R_iR_j also appears in at least half the total number of production rules then non-terminals R_i, R_j are *chunked* into a single non-terminal R_k. Consequently, all occurrences of R_iR_j in the set of production rules are replaced by R_k. A new production rule of the form $R_k \rightarrow R_iR_j$ is added to the set of rules. Further,

$$Pr(R_k \rightarrow R_iR_j) = 1 \tag{5.6}$$

The chunking process is repeated to find the bigram with the next highest value in M until no more chunking of non-terminals is possible using M. At this point

merging of production rules is done and the bigram-transition probability matrix is recomputed for the new set of non-terminals. Algorithm 3 gives details of the interleaved chunking and merging of production rules. The merging of production rules discussed in Section 5.3.3.

Special Case: Chunking R_iR_i

Chunking the bigram R_iR_i suggests a repetitive sequence of R_is. So, instead of simply abbreviating R_iR_i using a new non-terminal, we generate a recursive rule. As previously, a new non-terminal R_k is created and included in Δ. Two new rules $R_k \rightarrow R_i$ and $R_k \rightarrow R_iR_k$ are included in the rule set R. The probabilities of these new rules are set as follows:

$$Pr(R_k \rightarrow R_i) = 1 - M[i,i] \tag{5.7}$$

$$Pr(R_k \rightarrow R_iR_k) = M[i,i] \tag{5.8}$$

Consequently, any repeating sequence of R_is in the set of production rules are replaced by R_k. That is, all rules of the form $R_j \rightarrow \alpha R_iR_i...R_i\beta$ (where $\alpha, \beta \in \Delta^*$) are transformed to $R_j \rightarrow \alpha R_k\beta$.

Lemma 5.2. *Given a TAG grammar, chunking generates rules such that sum of probabilities of all rules with the same non-terminal on the LHS is 1.*

Proof. From Definition 5.1, in a TAG grammar the sum of probabilities of all rules with the same non-terminal on the LHS is 1. During the chunking of bigrams in a production rule $A \rightarrow \alpha$ does not change the number rules with A on the LHS. Therefore, the $Pr(A \rightarrow \alpha)$ does not change. Due to chunking of some bigram R_iR_j, a new rule $R_k \rightarrow R_iR_j$ is created. The new non-terminal R_k appears on the LHS of only one rule. From Equation 5.6, the sum of probabilities of all rules with R_k on the LHS is 1. Similarly, if a bigram R_iR_i is chunked, new rules $R_k \rightarrow R_i$ and $R_k \rightarrow R_iR_k$ are created. From Equations 5.7 and 5.8, the sum of probabilities of all rules with R_k on the LHS is 1. Thus, after chunking a TAG grammar the sum of probabilities of all rules with the same non-terminal on the LHS is 1. □

5.3.3 Merging Production Rules

The *chunking* of non-terminals ensures that repeating patterns are detected and compressed using new non-terminals. When the recurrent patterns are bigram

sequences of the same non-terminal, recursive rules are generated. Sometimes as a result of several chunking procedures, production rules with exactly similar non-terminal sequences on the right hand side may be generated. Further, production rules of the form $R_i \rightarrow R_j,\ (R_i, R_j \in R)$ may be generated; such rules are called *unit production rules*. Such *duplicate* rules and *unit* rules are eliminated by the merging process.

In certain cases, merging necessitates recomputation of the probability values of rules. Given a grammar $\mathbb{G} = \langle \Delta, \Sigma, R, S, Pr \rangle$ when dealing with unit productions rules of the form $R_i \rightarrow R_j$, the following cases may occur.

Case 1: If at most one of $S \rightarrow R_i$ or $S \rightarrow R_j$ is in R (S is the start non-terminal/variable) then, merging of R_i with R_j requires removing the rule $R_i \rightarrow R_j$ from R. This is followed by replacing all occurrences of R_i with R_j on the right hand side of all production rules.

Case 2: If both $S \rightarrow R_i$ and $S \rightarrow R_j$ are in R then the rules $R_i \rightarrow R_j$ and $S \rightarrow R_i$ are removed from R. From Definition 5.1, the sum of probabilities of all rules with any non-terminal V on the left hand side should always be 1. To ensure the same for the non-terminal S, the probability of the rule $S \rightarrow R_j$ changes as follows:

$$Pr(S \rightarrow R_j) = Pr(S \rightarrow R_i) + Pr(S \rightarrow R_j) \tag{5.9}$$

This modification of probability is also supported by the interpretation of probability values used herein. The probability of a rule $R_i \rightarrow R_j$ is based on the ratio of the number of TAGs that support the rule and the number of TAGs that use the non terminal R_i for generation.

It is worth noting that unit production rules of the form $S \rightarrow R_i$ (where S is the start non-terminal) are not merged.

As a consequence of *chunking* and *merging* of production rules, duplicate rules of the form $R_i \rightarrow \alpha,\ R_j \rightarrow \alpha$, such that $R_i = R_j$, may appear. To handle such a case, one of the two duplicates is removed. The probability of the remaining rule $R_i \rightarrow \alpha$ is modified as sum of the probabilities of the two duplicate rules. That is,

$$Pr(R_i \rightarrow \alpha) = Pr(R_i \rightarrow \alpha) + Pr(R_j \rightarrow \alpha) \tag{5.10}$$

Another type of duplicate that may appear is of the form, $R_i \rightarrow \alpha, R_j \rightarrow \alpha$ such that $R_i \neq R_j$ and $\nexists R_i \rightarrow \beta \in R,\ \nexists R_j \rightarrow \gamma,\ \alpha \neq \beta,\ \alpha \neq \gamma$ may appear. To handle

such duplicate rules, one of the duplicates is simply removed from R; no further action is required.

Lemma 5.3. *Given a TAG grammar, merging production rules modifies rules such that sum of probabilities of all rules with the same non-terminal on the LHS is 1.*

Proof. From Definition 5.1, in the initial TAG grammar, the sum of probabilities of all rules with the same non-terminal on the LHS is 1. If two rules $A \rightarrow \alpha$ and $A \rightarrow \beta$ are merged then one of the rules are deleted and, the probabilities of the rules may change according the Equations 5.9 and 5.10. Thus, ensuring that the sum of probabilities of all rules with A on the LHS remains 1. If two rules $A \rightarrow \alpha$ and $B \rightarrow \beta$ are merged then one of the rules are deleted. Say the rule $B \rightarrow \beta$ is deleted then the number of rules with A on the LHS remain unchanged. Therefore, the sum of probabilities of all rules with A on the LHS remains 1. □

Algorithm 3: *InterleavedChunkMergeProcedure*, Algorithm to chunk and merge rules of the grammar $\mathbb{G}$

Input: $\mathbb{G}$; `Input is the base grammar`
Output: $\mathbb{G}$; `Output is the compacted, modified grammar`
begin
  Initialize $ch = TRUE$
  while $ch \neq FALSE$ **do**
    Compute bigram transition probability matrix M
    while $\forall i, j, M[i, j] = 0$ **do**
      Find index $(maxi, maxj)$ with maximum value in M
      $M[maxi, maxj] \leftarrow 0$
      if R_iR_j *appears in more than* $currsize/2$ *rules in* R **then**
        Chunk R_iR_j
        $ch = TRUE$
      else
        $ch = FALSE$
    forall $(R_i \rightarrow \alpha) \in R$ *such that* $\alpha \in \Delta^*$ **do**
      Merge all Unit Rules of the form $R_i \rightarrow R_j,\ R_i \neq S$
      Merge all Duplicate Rules

5.3.4 Theoretical Analysis

The TAG grammar induction algorithm detailed in Algorithm 2 has three distinct phases:

(1) *generate a base set of grammar rules*,

(2) *merge terminal rules*, and

(3) *interleaved chunking and merging of production rules.*

Using this division of phases, it is proved in Theorem 5.1 that the TAG grammar induction algorithm detailed in Algorithm 2 generates a probabilistic context free grammar. Further, it is shown that the TAG grammar induction algorithm converges in a finite number of steps in Theorem 5.2.

Theorem 5.1. *The TAG grammar induction algorithm generates a probabilistic context free grammar.*

Proof. The TAG grammar induction algorithm detailed in Algorithm 2, generates a grammar $\langle \Delta, \Sigma, R, S, Pr \rangle$. The algorithm generates terminal rules of the form $A \rightarrow a$ and production rules of the form $A \rightarrow \alpha$ where $A \in \Delta,\ a \in \Sigma,\ \alpha \in \Delta^*$. From Lemma 5.1, grammars with these types of rules are context-free grammars.

Probabilistic context free grammar (PCFG) include a probability function that assigns probability to the rules. The probabilities are assigned such that sum of the probabilities of all rules with some non-terminal A on the LHS is 1. We show that, at every phase of Algorithm 2, the probability assigned to the rules maintain the same property as the probability function of PCFG.

In phase (1), each non-terminal except for the start variable S, appears on the LHS of exactly one rule. Assuming an equal distribution of priors and that there are n positive examples, we assign a probability for every rule with S on the LHS as follows:

$$Pr(S \rightarrow \alpha) = \frac{1}{n}$$

Since in this phase there are n such rules,

$$\sum_{\alpha} Pr(S \rightarrow \alpha) = n * \frac{1}{n} = 1$$

In this phase, all other rules where a non-terminal appears on the LHS of exactly one rule, $Pr(A \rightarrow \alpha) = 1$.

In phase (2), two rules $A \rightarrow a$ and $B \rightarrow b$ can be merged to form a single rule $C \rightarrow c$. The new non-terminal C appears in the LHS of only rule. Thus, $Pr(C \rightarrow c) = 1$.

In phase (3), from Lemmas 5.2 and 5.3, chunking and merging of production rules maintain the probabilistic property of TAG grammar. That is, the sum of probabilities of all rules with the same non-terminal on the LHS remain 1. □

Theorem 5.2. *The TAG grammar induction algorithm converges in a finite number of steps.*

Proof. For the algorithm to converge, the three individual phases should converge.

In phase (1), a set of base grammar rules is generated from the set of positive examples. This number of steps in this phase is linear in the number of positive examples.

In phase (2), the terminal rules are merged. This is done by a pairwise comparison of the rules until no more terminal rules can be merged. Let us assume that there are n terminal rules. In the worst case, at every step only two rules are merged to form a single rule; after n steps the number of terminal reduces to one. At this point, the procedure will have to stop because no more rules can be compared and subsequently merged.

In phase (3), the production rules are chunked and merged in an interleaved fashion as detailed in Algorithm 3.

- During chunking, bigrams that are repeatedly occurring on the RHS of the set of production rules are combined to form new non-terminals. Chunking a bigram creates a new production rule and replaces the bigram by a single non-terminal. Therefore, after a chunking operation the length of the RHS of the rule decreases by one and a new rule is generated. The new rule has the newly created non terminal on the LHS and the bigram consisting of two non terminals on the RHS. Let us assume that there are at least m non-terminals on the RHS of any production rule. In the worst case, chunking would reduce the length of the RHS of all the production rules to one. At this point there are no bigrams and consequently no more chunking is possible. Within a single interleaved iteration step, the chunking converges when there are no more bigrams to be chunked. Further, based on the same argument no more chunking is possible after a finite number of interleaved iteration steps.

- The merging of production rules eliminates the unit rules and duplicate rules generated during chunking. Elimination of a finite number of unit rules and a finite number of duplicate rules will need finite number of steps. Therefore, the merging process always terminates in a finite number of steps.

- Finally, the convergence of the interleaved chunking and merging depends on whether any new bigrams are chunked. If no bigrams are chunked, then no new unit rules or duplicate rules are generated; consequently no merging is necessary. In every interleaved iteration step, the number of bigrams that

can be chunked decreases. This is because when a bigram R_iR_j is chunked to form a new non-terminal R_k, then new bigrams are formed using R_k. However, the number of unique bigrams with R_k will be less then R_iR_j has appeared in any rule. Thus, in a finite number of interleaved iteration steps there will be a step when no chunking is done and thereby no merging is necessary. In other words, the third phase converges in a finite number of steps.

From the above arguments, Algorithm 2, converges in a finite number of steps. □

5.3.5 Parsing

To parse some activity A using a TAG Grammar $G = \langle \Delta, \Sigma, R, S, Pr \rangle$, the activity A is first converted to a *most probable string of terminals* belonging to the grammar. As discussed in Section 5.2, a terminal in a TAG Grammar is an *instantaneous* TAG *subgraph* or insubTAG. The *most probable string* is computed based on the *edge-label similarity* (see Equation 4.7 in Chapter 4). To convert an insubTAG of the activity sequence A to a terminal of G, we match it with all the available terminal insubTAGs of the G. If g^i is the ith insubTAG of A then the following two cases are possible, that are handled accordingly.

- If g^i exactly matches a single terminal of $t \in \Sigma$, then use t for g^i in the *most probable string*
- If g^i does not exactly match any terminal in Σ then use *edge-label similarity* to compute the average similarity between the corresponding edge-labels. Use the terminal which is the most similar.

After converting A to a most-probable sequence of terminals, a regular string parser can be used to parse the activity. We use an LR(0) parser for parsing [100]. The probability of the *most probable sequence* being generated by the grammar is updated whenever a *reduce* action (see Section 2.8.1 of Chapter 2) is applied. The sequence is classified as the activity class which has the highest probability of generating it.

Further, since the tested activities have never been seen before, in order to account for errors and variations from the training data we use an *error penalty*. During parsing of an activity, if a terminal is encountered that does not match any of the patterns described by the rules of the grammar the *error penalty* is applied to the probability computation and the non-matching terminal is ignored

from the activity sequence. The *error penalty*, ρ, is a value between 0 and 1 that is multiplied to the probability computation every time a non-matching terminal is encountered. The effect of the error penalty is two -fold:

(1) To allow unseen patterns to be parsed conditionally without rejecting it completely.

(2) To reduce the probability of the sequence to reflect that a pattern not covered by the grammar is encountered.

5.4 Experimental Evaluation

The TAG grammar is defined to model activities represented in the form of TAGs. The TAG grammar induction algorithm induces such a model grammar from a set of positive examples. To evaluate the TAG Grammar and the induction algorithm presented in this chapter we perform experiments on three datasets viz. Mind's Eye dataset, UT Interaction dataset [3], and SBU Kinect Interaction dataset [122]. The classification accuracies obtained from the experiments are compared with existing work reported in literature.

5.4.1 Experimental Setup

To perform a grammar-based recognition of human activities, we first obtain part-based tracking data of the activities in the video. The entities involved in the activity are abstracted as *extended objects*; based on the spatial relations computed using Extended CORE9 all activities are represented using TAGs (as described in Chapter 4). A detailed description of how a video is converted to TAG representation can be found in Appendix B

As shown in the block diagram, training and test sets are different. In the Learning phase the videos in the training set are converted to TAG. The TAG grammars are induced from these TAG representations using Algorithm 2 (Please refer to Section 5.3). The algorithm initializes a base set of rules for the grammar from the TAG represented example videos. The base rules go through several stages of modification because of merging terminal rules and an interleaved chunking merging procedure on the production rules. The interleaved chunking merging procedure is detailed in Algorithm 3. The details of the various stages of the induction algorithm - merging terminal rules, interleaved chunking merging of production - are detailed in Section 5.3.

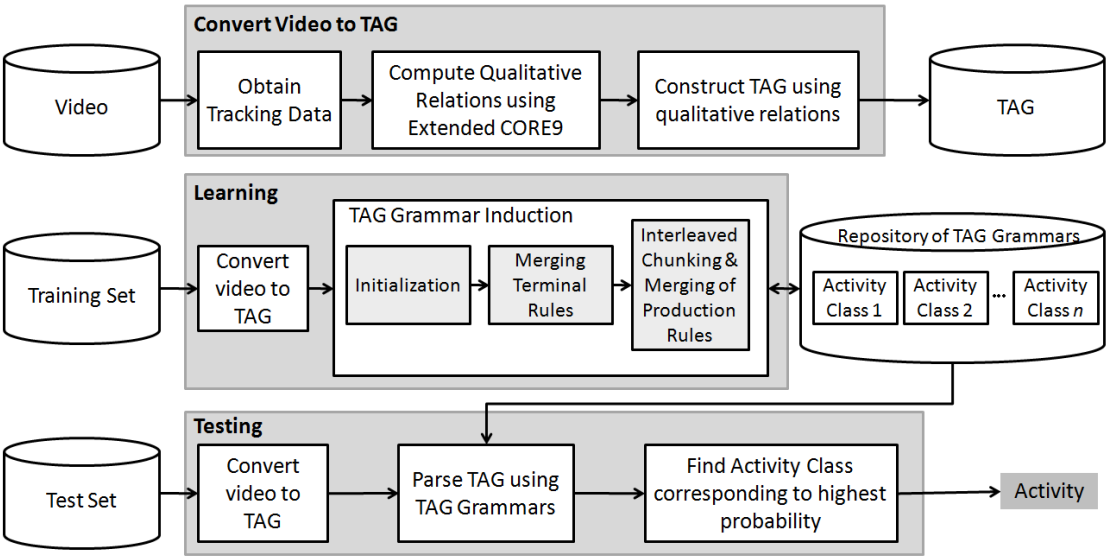


Figure 5-2: Block Diagram of the steps in Learning and Recognition of videos using TAG Grammars

The learning algorithm generates a probabilistic context free grammar using the examples from the training set. The learned grammar rules are then used to recognize activities in unseen test data. In the testing phase, the videos in the test set are converted to TAG and parsed using the learned TAG grammars.

For recognition using the learned TAG grammar, a parsing technique is discussed in Section 5.3.5. For parsing, an unseen video activity represented as a TAG is used as input. The TAG grammar is a probabilistic grammar and parsing using such a grammar generates a *probability* value. This value indicates the probability that the parsed TAG will be generated using a particular TAG grammar.

Individual TAG grammars are induced for every activity class using the training set. The TAGs in the test sets, representing activities, are parsed using all the generated grammars obtained from the training phase to classify the activity. An activity is classified as the activity class, C, if the corresponding TAG Grammar, $\mathbb{G}_C$, recognizes it. The TAG Grammar is a probabilistic grammar and the parsing result using such a grammar returns the probability of the activity being generated by the grammar (as discussed in Section 5.3.5). If there are more than one TAG Grammar that accepts a particular test activity, then it is categorized as the class corresponding to the grammar which has the highest probability of generating that activity.

5.4.2 Experimental Results

Experiments were performed using 110 videos from the Mind's Eye[3], 50 videos from the UT-Interaction [3] dataset and 282 videos from the SBU Kinect Interaction dataset [122]. For the Mind's Eye dataset we consider 11 activities - *approach, carry, catch, collide, drop, follow, hold, kick, pickup, push* and *throw* - and 10 videos for each activity. For the UT Interaction dataset we consider the five activities that involve at least two humans - *handshaking, hugging, kicking, punching* and *pushing*. The activities are represented as TAGs and TAG grammars are induced for each activity class.

The classification results using the induced grammars are obtained. Results of experiments conducted using an error penalty ($\rho = 0.5$) and without using any error penalty ($\rho = 0$) are reported herein. The *precision*, *recall*, and *f1-score* corresponding to each of the three datasets are obtained - the results for the Mind's Eye dataset are given in Table 5.1, the results for UT Interaction dataset are given in Table 5.2 and the results for the SBU Kinect Interaction dataset are given in Table 5.3. A comparison of the classification accuracies for each of the three datasets using TAG Grammar based recognition with TAG kernel SVM classification is given in Table 5.4. A comparison of the f1-scores for the three datasets using TAG grammar based recognition with TAG kernel SVM classification is shown in Figures 5-3, 5-4, and 5-5. Further, a comparison of the TAG grammar based recognition with existing work in literature is shown in Table 5.5.

Activity	$\rho = 0$			$\rho = 0.5$		
	P	R	F1	P	R	F1
Approach (10)	0.58	0.70	0.64	0.78	0.70	0.74
Carry (10)	0.70	0.70	0.70	1.00	0.80	0.89
Catch (10)	0.62	0.80	0.70	0.75	0.90	0.82
Collide (10)	1.00	0.60	0.75	1.00	0.80	0.89
Drop (10)	0.70	0.70	0.70	1.00	0.90	0.95
Follow (10)	1.00	0.70	0.82	1.00	0.60	0.75
Hold (10)	0.42	0.80	0.55	0.50	1.00	0.67
Kick (10)	0.64	0.90	0.75	0.90	0.90	0.90
Pickup (10)	1.00	0.40	0.57	1.00	0.70	0.82
Push (10)	1.00	0.80	0.89	0.64	0.70	0.67
Throw (10)	1.00	0.70	0.82	0.80	0.80	0.80

Table 5.1: Results for TAG Grammar based recognition on Mind's Eye dataset

[3]www.visint.org

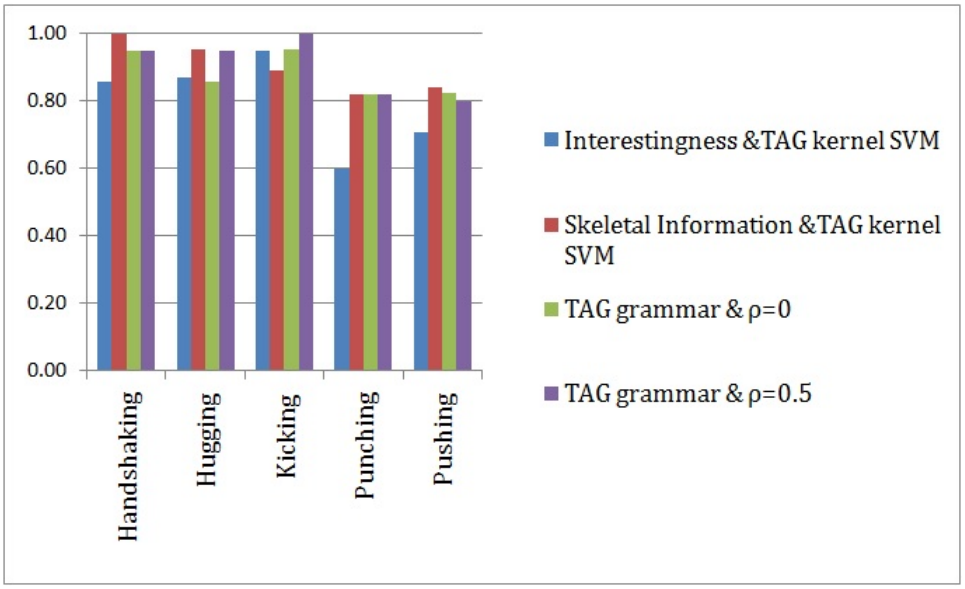


Figure 5-3: Comparison of F1-scores on UT Interaction Dataset

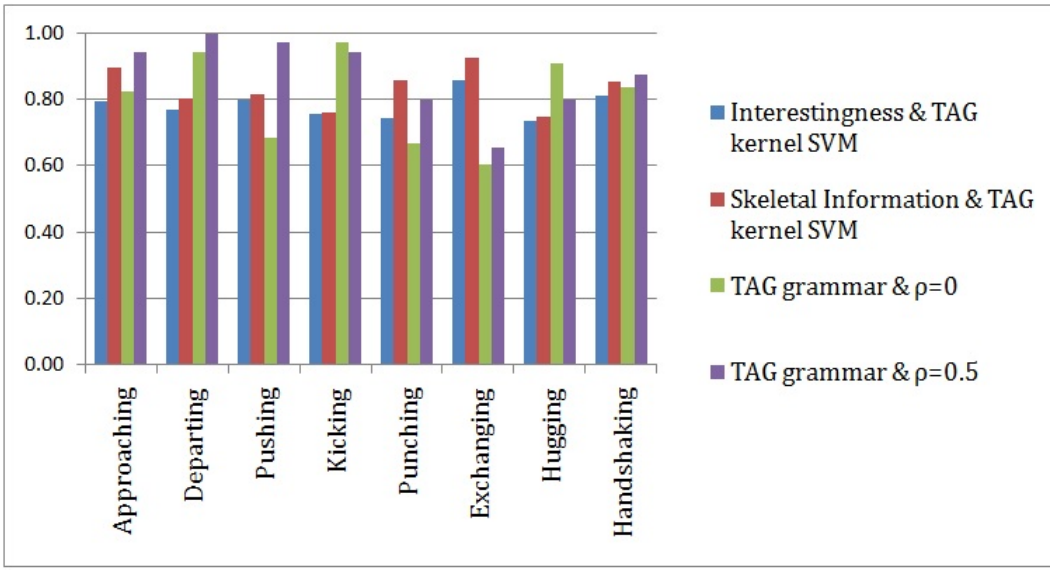


Figure 5-4: Comparison of F1-scores on SBU Kinect Interaction Dataset

Activity	$\rho = 0$			$\rho = 0.5$		
	P	R	F1	P	R	F1
Handshaking (10)	1.00	0.90	0.95	1.00	0.90	0.95
Hugging (10)	0.82	0.90	0.86	1.00	0.90	0.95
Kicking (10)	0.91	1.00	0.95	1.00	1.00	1.00
Punching (10)	0.75	0.90	0.82	0.75	0.90	0.82
Pushing (10)	1.00	0.70	0.82	0.80	0.80	0.80

Table 5.2: Results for TAG Grammar based Recognition on UT Interaction dataset

Activity	$\rho = 0$			$\rho = 0.5$		
	P	R	F1	P	R	F1
Approaching (42)	0.88	0.78	0.82	0.94	0.94	0.94
Departing (43)	1.00	0.89	0.94	1.00	1.00	1.00
Pushing (40)	0.71	0.67	0.69	1.00	0.94	0.97
Kicking (41)	1.00	0.94	0.97	1.00	0.89	0.94
Punching (18)	0.73	0.61	0.67	1.00	0.67	0.80
Exchanging(21)	0.46	0.89	0.60	0.49	1.00	0.65
Hugging(39)	1.00	0.83	0.91	1.00	0.67	0.80
Handshaking(38)	1.00	0.72	0.84	1.00	0.78	0.88

Table 5.3: Results for TAG Grammar based Recognition on SBU Kinect Interaction dataset

Method	UTI	ME	SBUKI
ExtCORE9 BoW + KNN	62%	58.18%	40.78%
ExtCORE9 BoW + SVM	74%	55.45%	47.16%
ExtCORE9 BoW + Naive Bayes	74%	55.45%	49.64%
ExtCORE9 BoW + Deep Learning	64%	57.27%	46.45%
TAG + Skeletal Information + TAG kernel SVM	90%	73.63%	81.91%
TAG + Interestingness + TAG kernel SVM	80%	65.45%	78.01%
TAG + TAG Grammar Recognition with $\rho = 0$	88%	70.91%	79.17%
TAG + TAG Grammar Recognition with $\rho = 0.5$	90%	80%	86.11%

Table 5.4: Comparison of classification accuracies on the UT Interaction (UTI) dataset, Mind's Eye (ME) dataset and SBU Kinect Interaction (SBUKI) dataset

5.4.3 Discussion

It is clear from the results reported in Tables 5.1, 5.2, and 5.3 that using an error penalty ($\rho = 0.5$) during parsing gives a better result then not using any error penalty ($\rho = 0$). This suggests that while the induced TAG grammar successfully covers most of the activity sequences under a particular class based on a limited training set, there are some test sequences which are not exactly recognized by the learned grammar rules. This is due to variations in the test sequences for the training data or errors and occlusions in the test data. This is easily corrected by introducing the concept of error penalty and allowing unseen sequences to be conditionally recognized by the grammar. This is found to be true for almost all

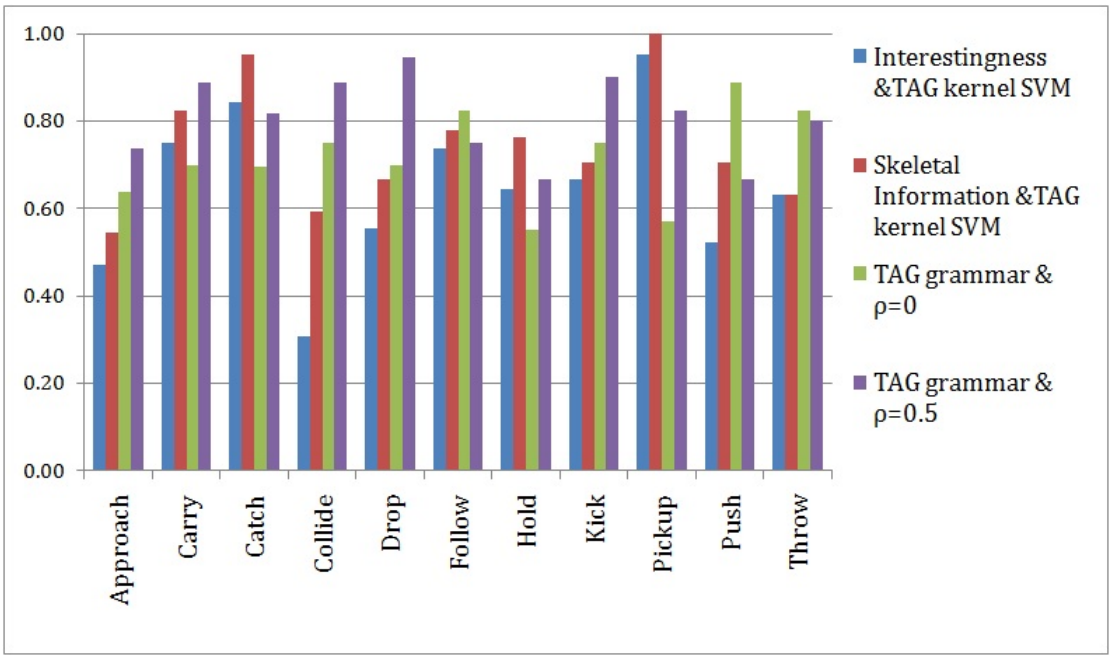


Figure 5-5: Comparison of F1-scores on Mind's Eye Dataset

Method	UTI	ME	SBUKI
TAG + TAG Grammar Recognition with $\rho = 0.5$	**90%**	**80%**	**86.11%**
TAG + TAG Grammar Recognition with $\rho = 0$	**88%**	**70.91%**	**79.17%**
TAG + Skeletal Information based Kernel	90%	73.63%	81.91%
TAG + Interestingness based Kernel	80%	65.45%	78.01%
ExtCORE9 BoW + Naive Bayes	74%	55.45%	49.64%
ExtCORE9 BoW + Deep Learning	64%	57.27%	46.45%
Angled CORE9 + LDA [41][a]	-	64.4%	-
BoW + SVM [118]	77%	-	-
BoP + SVM [118]	95%	-	-
Skeleton + Deep LSTM [58]	-	-	86.03%

Table 5.5: Comparison of classification accuracies with other approaches in literature

[a] In [41] only 5 activities are considered; in this thesis 11 activities of the dataset are considered.

the classes across the three datasets from the consistently higher f1-scores obtained for classification using $\rho = 0.5$.

It is further noted from Tables 5.1, 5.2, and 5.3, that when no error penalty is used, certain activity classes have high precision but comparatively low recall values. Activities *collide, pickup* from the Mind's Eye dataset, *pushing* from the UT Interaction dataset, and *punching, hugging, handshaking* from the SBU Kinect Interaction dataset belong to this category. These are seen to be the activities that have TAG grammar rules that are similar to other activity classes in the same dataset. For example, *collide* from the Mind's Eye dataset has a similar TAG grammar structure to *approach* or *catch*. This is further corroborated by the

higher recall and lower precision values of *approach* and *catch*. Similarly, *pickup* has a TAG grammar that is somewhat similar to that of *hold*. From the UT Interaction dataset *pushing* again has a similar TAG grammar to *punching*. From the SBU Kinect Interaction dataset *pushing* has a similar TAG grammar structure to *punching*; *hugging* is similar to *exchanging*; and *handshaking* is similar to *approaching*. Because of the structural similarities, any unseen sequence belonging to any of these classes are easily misclassified. By introducing the error-penalty such misclassifications are reduced.

From Figures 5-3, 5-4, and 5-5, it is seen that for most classes the TAG grammar based recognition has a higher f1-score than the TAG kernel based SVM classification discussed in Chapter 4. This is particularly true for the Mind's Eye dataset and the SBU Kinect Interaction dataset. However, there are exceptions, such as the *punching* and *exchanging* activities of SBU Kinect Interaction dataset and *catch, hold*, and *pickup* activity of the Mind's Eye dataset. Incidentally, these are also the same activities that are seen to have similar TAG grammars with other classes from the above discussion. For the UT-Interaction dataset, the results using the TAG kernel based SVM classification and the TAG grammar based recognition are mostly comparable.

A comparison of classification accuracies obtained using TAG grammar and existing literature is shown in Table 5.5. As discussed earlier in Chapter 4, Section 4.4.3, the results reported for Angled CORE9 [41] are for five activities while we use 11 activities of the Mind's Eye dataset. Further, an average MCC of 0.37 is reported in [124] for all activity classes in the Mind's Eye dataset. On the other hand TAG grammar has an average MCC of 0.79 for 11 activities of the Mind's Eye dataset. The TAG grammar also achieves classification accuracies that are comparable to the state-of-art results for the SBU Kinect Interaction dataset and UT Interaction dataset. However, BoP + SVM [118] outperforms our system performance. As discussed in Chapter 4 Section 4.4.3, this could be because the representation uses ST-phrases combine the spatio-temporal descriptions of related local features. On the other hand, the TAG representation provides only a temporal structure to the spatial relations between the extended objects.

It is worthwhile to note that the classification accuracies without using error-penalty based parsing is lower when compared to the TAG-kernel based SVM classification for all three datasets. This expected because it is traditionally accepted that discriminative classifiers such as SVMs provide a better classification of the data. However, generative learning mechanisms, such as induction of grammar rules, allow modelling the underlying structure of the classes.

5.5 Conclusion

In this chapter, we have presented TAG grammar to model activity classes wherein activities are represented as TAGs. Such a TAG grammar can be used to recognize and thereby classify activities using a simple parsing mechanism. An algorithm for automated induction of the rules of such a TAG grammar was presented. Further, a mechanism for parsing activities using such a grammar is discussed.

The TAG grammar presented in this chapter for modelling activities can be induced from a set of positive examples. One of the drawbacks of the proposed induction algorithm is that it is non-incremental. In the next chapter, an extended discussion of the drawbacks of the work presented in this thesis is given. In addition to the drawbacks, a detailed discussion of the novelty, possible extensions, and future work is presented.