Multimedia Systems June 2012, Volume 18, Issue 3, pp 263-281. The final publication is available at at Springer via http://dx.doi.org/10.1007/s00530-011-0255-8.

Spatio-Temporal Querying Recurrent Multimedia Databases Using a Semantic Sequence State Graph

M.M. NAIK, M. SIGDEL and R.S. AYGUN
Computer Science Department
University of Alabama in Huntsville, Huntsville, AL, USA.
E-mail: {mnaik,ms0023,raygun}@cs.uah.edu

Abstract

We present an indexing method for spatiotemporal data: semantic sequence state graph (S^3G). S^3G maintains objects with their locations as states and events as transitions. The spatial information is maintained in states while the semantic events result in temporal ordering between the states. If the objects visit the same locations repeatedly, we call such databases as recurrent databases. Our querying interface supports queries based on spatio-temporal logic (STL) that includes operators such as 'next' and 'eventually'. The interactive querying interface enables the user to build the query interactively and see the intermediate results of the query.

1. INTRODUCTION

With the growing interest in spatio-temporal data for more than a decade, the indexing and retrieval of spatio-temporal data has been a challenging research area. Multimedia data plays a key role in today's world including but not limited to education, advertisement, entertainment, communication, and information retrieval. Especially, videos have been the most intriguing media since videos have multimodal features along with spatio-temporal properties. Everyday many videos are uploaded on websites such as You Tube [YouTube 2011] and Google Video [Google Video 2011]. Various strategies have focused on modeling of different aspects of videos such as modeling fuzzy information [Aygun 2004] and spatio-temporal features of the objects in a video [Pissinou 2004; Li 1997].

The goal of our research is to model, store, query, and index the semantic contents of the videos. We classify spatio-temporal querying (STQ) into two based on how STQ can be executed:

- *Split STQ:* This query usually targets objects that satisfy some spatial constraints *within* a period of time or vice versa. The data satisfy the spatial constraints independent of historical data as long as the data belong to the given time domain. A sample query is as follows: "Give a list of regions, where the ball appears between 5^{th} and 10^{th} minutes" [Koprulu 2004].
- *Coupled STQ:* This query usually targets objects that satisfy some (sequential) spatial constraints *over* a period of time. Series of consecutive or nonconsecutive constraints are part of a query. A sample query is "return videos having a Delta plane take-off right before United plane take-off

but after Continental plane landing in Huntsville, Alabama." Complex spatio-temporal patterns (STP) [Hadjieleftheriou 2005] are an example of coupled STQ.

Without generalization, the spatio-temporal indexing methodologies can be divided into two: tree-based and sequence matching. Tree-based methods require that a hierarchical organization of data is possible. In most cases, this hierarchical organization is achieved with respect to spatial properties. Tree-based approaches are effective for split STQ. The coupled STQ with tree-based indexing can be achieved by linking subqueries of a STQ. However, tree-based indexing may lose its efficiency if sequence information is queried rather than time interval due to comparing the data for all time intervals. Sequence matching approaches can deal with temporal sequence matching. However, a proper indexing methodology is required before starting sequence matching otherwise the number of sequence matchings may be significantly high.

A good indexing structure should address the requirements of the data model as well as querying. We define a *recurrent database* as a database where the content or objects of interest with corresponding actions are repeated frequently. In other words, objects are associated with spatial locations, and a possible event (or an action) causes the change of these locations. If the domains of spatial locations, events, and actions that cause the change of locations are finite, the objects are likely to appear at the same locations (due to some actions) in the database. The camera is usually mounted on an almost static platform for capturing the environment. In this sense, almost all sports videos are a subset of recurrent databases. Other examples include news-anchor, distance-learning education, and surveillance videos. Our goal in this paper is to index such recurrent databases.

We believe that spatio-temporal content of video is important for querying. Rather than a semantic instantaneous event (e.g., "missed shot") in a clip, we are interested in a sequence of actions and objects with their locations. When we compare various spatio-temporal techniques, a general assumption on the database is the presence of a single timeline. However, it is likely that there may be more than one timeline. For example, each timeline maybe set for a different day. In such a case, the ordering of timelines may not be an important factor for STQ. The tree-based and sequence matching methods should adapt to the presence of multiple timelines.

Traditional indexing methods cannot achieve the goals of semantic databases that maintain high-level information since they are usually designed based on non-semantic properties. Most common indexing techniques use comparison operators to retrieve the requested data from the database. The semantic querying can only be achieved by having additional layer on top of traditional indexing methods.

Our Approach. In this paper, we propose an indexing and search method that directly targets coupled STQ in a recurrent database. In order to provide an efficient storage and retrieval of video data, a

semantic modeling and retrieval system, SMART [Jain 2008; Jain 2009] was proposed. In this paper, the SMART system is improved by providing a novel indexing method, named as S^3G: a Semantic Sequence State Graph. An early version of S^3G was introduced in [Naik 2008]. The major difference between this indexing method and the traditional ones is that in S^3G the links between states have semantics where states maintain the discrete information about the spatial properties of objects. The events correspond to the transitions in S^3G graph. Since transitions correspond to semantic events, it is possible to perform queries based on semantic concepts following the transitions in S^3G. We should note that we are not interested in the shapes of the objects. We are rather interested in where and when they appear. We are interested in spatio-temporal events that can be denoted at discrete times. We assume that a semantic event causes the difference between two states. The spatial queries are performed with respect to the object-location pairs. Temporal queries based on Linear Temporal Logic are performed by following the transitions in S^3G graph. Spatio-temporal queries use Spatio-Temporal Logic. The spatio-temporal queries, such as what will be the next state (i.e., the spatial properties of objects following a transition on a given state) and whether a particular state eventually occurs in future after a given state has occurred, are implemented using S^3G. Graphical viewing of the query construction is implemented to facilitate easy building complex queries.

The contributions of our approach can be summarized as follows: a) provides a compact representation of a video database, b) supports recurrent databases, c) indexes a database that has multiple sequences with different timelines onto a single data structure, d) links the states with semantic events (or actions), and e) provides an interactive querying interface where the intermediate results are provided and help the user refine his query.

This paper is organized as follows. The following section provides the background and discusses about related work. Section 3 describes the S^3G. Section 4 explains building S^3G. The interface and querying are explained in Section 5. The last section concludes the paper.

2 BACKGROUND AND RELATED WORK

In this section we firstly describe the related work; secondly, provide background on our SMART system; and finally, explain the spatio-temporal logic used in this paper.

2.1 Related Work

Significant research has been performed on indexing temporal and spatial databases [Jensen 1999; Min 2001]. A good survey on indexing temporal data appears in [Salzberg 1999]. A recent survey on spatio-temporal video retrieval is presented in [Ren 2009]. STP [Hadjieleftheriou 2005] provides indexing based on spatial locations. For each spatial location, objects are sorted based on their identifiers. If an object

visits the same location, then visits of an object are sorted based on the time they visit the location. Lagogiannis et al. [Lagogiannis 2009] improve this approach if an object does not visit the same location again. Indexing spatio-temporal data is mostly based on traditional database indexing techniques such as R-trees [Saltenis 2002] and B+-trees [Lin 2005]. One of the major problems of indexing based on these indexing methods is the lack of necessary semantics to build queries that require semantic information. Therefore, semantic retrieval on top of these indexing methods can only be implemented by an upper layer of semantic operations. This puts an additional burden on the retrieval. If the indexing method could capture semantic properties, the retrieval efficiency could be improved.

2.1.1 Tree-Based Indexing

Tree-based methods cannot be used for semantic querying directly since tree-based methods assume that hierarchical organization of data is possible. However, no order can be imposed between the semantic values. The indexing strategy should be able to order semantic components of the database. The ordering between data at different time instants is not necessarily before-after relation. The ordering may also include the cause or event between different time instants. Traditional temporal querying includes storing starting and stopping time of clips and then comparison of these starting and ending points for querying based on Allen's temporal logic based on intervals [Allen 1983].

A survey of indexing methods for multimedia databases that includes spatial data indexing is provided in [Böhm 2001]. The most common way of indexing spatio-temporal data is based on tree indexing. The RS-Tree [Park 2001], X-tree [Berchtold 2002], weR-Tree [Bozanis 2007], NV-tree [Lejsek 2009], SMILe-tree [Lee 2008], Bx-tree [Jensen 2004], ST^2B –tree [Chen 2008], TPR-tree [Saltenis 2000], TPR*-tree [Tao 2003], and 2^n index tree [Ye 2007] are tree-based indexing methodologies.

R-TREE-BASED INDEXING. *RS-tree* [Park 2001] uses R-tree for spatial properties. The non-spatial properties are indexed with respect to R-tree using an S-tree which has the same structure to prune the results of querying the R-tree given non-spatial properties. *X-tree* [Berchtold 2002] improves R-trees by incorporating supernodes to deal with overlaps in R-trees. Supernodes avoid splits in the directory that would cause inefficient directory structure. Hence, X-tree looks like a combination of R-tree like and linear array like directory. *Weighted R-trees* (weR-trees) [Bozanis 2007] propose weight-balanced R-trees for dynamic manipulation of large datasets. The weight is based on the number of values and fanout of a node. The goal is to produce a partial rebuilding by gradual construction of subtrees. *Time-parameterized R-tree* (TPR-tree) [Saltenis 2000] uses spatial positions and velocities in each dimension to index data. An object that can move in 2D space will have 4 four parameters. Moving points are bounded by time-parameterized rectangles and then indexed by R-trees. The queries are assumed to be applied for a window of time. TPR*-tree [Tao 2003] improves TPR-tree by reducing the cost of insertions and

deletions. These methods can aim either past or future but not both since outdated objects are deleted [Tao 2003]. *SMILe-tree* [Lee 2008] is based on K-d trees and R+ trees. At the first level, the data is organized according to the K-d tree with corresponding dimensions at each level. For hierarchical organization of overlapping components, R+ trees are used to avoid overlappings.

B-TREE-BASED INDEXING. *B^x-tree* [Jensen 2004] is based on B+-tree. Object locations are mapped to single-dimensional values using space filling curves. The maximum time between two updates is partitioned into *n* phases. An object with its location is mapped to the corresponding partition of B^x-tree based on its timestamp. It allows queries after the current time. Old partitions of B^x-tree are removed by appending new partitions. B^x-tree returns false hits as it only uses location of the objects. *B^{dual}-tree* [Tao 2008] partitions data in dual space, i.e., both location and velocity, to reduce the number of false hits. So it uses location and velocity to obtain the one-dimensional key. Each internal entry of a B^{dual}-tree maintains a set of moving rectangles, which leads to use of R-tree like query algorithms. Maintaining moving rectangles leads to high computation overload by slowing down the update operations. *A self-tunable spatio-temporal B+-tree* (ST^2B-tree) [Chen 2008] partitions the space with respect to reference points (similar to Voronoi diagram). Each Voronoi cell is assigned a grid, and an object is assigned to the grid of the closest reference point. It has two phases for time; and different reference points are used to deal with data diversity. It supports range queries for space only. In *multicurves* [Valle 2008], each curve is responsible for a subset of the dimensions to reduce the boundary effects inside each curve. The dimensions of data elements are split among a number of space-filling curves. The projections of data are mapped to curves and the one dimensional coordinates of data are computed and stored in a sorted list.

OTHER TREE-BASED INDEXING. *STRIPES* index [Patel 2004] maps 2D moving objects to 4D points using Hough-X transform and then stores them in a PR bucket-quad tree. The objects are represented in a dual transformed space. The trajectories in (d+1) dimensional space are mapped to 2D dual space. STRIPES has high query cost and storage size. Nearest-Vector-tree (NV-tree) [Lejsek 2009] projects data on a line and segments the data and then re-projects data recursively until each segment has enough number of data points. NV-tree allows overlapping of boundaries to overcome the problem of nearest-neighbor search when two close neighbors may belong to different partitions. *2^n index tree* [Ye 2007] has states that include 3D positions, orientation, and speed of objects. Objects are represented as states with timestamps. Objects with limited number of timestamps can be indexed by 2^n index tree.

2.1.2 Sequence Matching

Sequence matching techniques try to find the closest sequence given a proper sequence without gaps. It is also possible that the query sequence can be partially described with some gaps in the sequence. For example, the beginning and ending of a query sequence might be provided. In addition, some sequences

might have a cyclic nature where some parts of the sequence may be repeated several times. However, most sequence matching methods use proper sequence matching without any gaps.

Embedded-based subsequence matching (EBSM) method [Athitsos 2008] maps a query to a sequence of vectors and then closest vectors are searched in the database. Further exploration for close sequences is handled by dynamic time warping based on subsequence matching algorithm without gaps. *Spatiotemporal sequence matching* is used for video copy detection in [Kim 2005]. Two distances are obtained: spatial distance using ordinal measures of 2x2 partitions of the video frames and temporal distance based on the changes in the subsequent frames. This method targets exact match between sequences (i.e., copy detection).

2.1.3 Representation

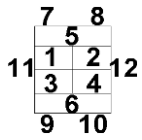
In [Hongeng 2004], an activity is considered to be composed of action threads. A single thread action is represented by a stochastic finite automaton of event states. Events are represented as Bayesian networks (which are acyclic graphs) and event constraints such as "event (or sub-event) A should occur before event (or sub-event) B; and B should occur before event (or sub event) C" are used to identify an event (not for retrieval). While Hongeng et al [Hongeng 2004] deal with event recognition, our S^3G is used to retrieve the clips with desired state (object location values). In [Wattamwar 2008], an extension to MPEG-7 is proposed to detect actions such as “A follows B.” Assfalg et al. [Assfalg 2002] propose a system that semantically annotates the sports videos at different layers of semantic significance. It however uses semantic annotations for multimedia indexing and retrieval without using a graphical representation of the semantic data. Lay and Guan [Lay 2006] propose the use of a grammar for video retrieval. They use an *adjacency matrix* to determine the behavior of player such as baseline player and *inverted index* to retrieve clips based on object locations. They mention about the use of operators such as ADJ (temporally adjacent), W (within as a spatial operator), and temporal ordering with < operator. However, there is no information about the interface for providing queries or the complexity of implementing queries using these operators based on the proposed indexing structures.

2.2 Background on SMART

SMART [Jain 2008; Jain 2009] represents semantic information as a grammar-based string that enables spatio-temporal queries in SQL query language. The player names in sports video, the shots given by the players, and their score are the examples of the semantic content of a video. This semantic information is represented as a grammar-based string that enables spatio-temporal queries in SQL query language. The major semantic contents of video are considered as objects, events, locations, and cameras in SMART. SMART can be applied to most sports games. We use tennis game as an example since it has

an important advantage over most sports videos: the complete view of the game can be captured with one almost-static camera. Three main objects, as $\Sigma_O = \{U, V, b\}$, were identified. Here, U and V represent the players, and b is the ball. Two events, as $\Sigma_E = \{F, B\}$, were identified. Here, F represents the forehand shot and B represents the backhand shot. The tennis court was divided into various regions as shown in Figure 1(a): $\Sigma_L = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, N\}$ where N represents the net. Six different camera views are considered for tennis videos: *A*: Gives a close view of the player at locations 7 and 8 in Figure 1(a); *B*: Gives a close view of the player at locations 9 and10; *C*: Court view; *D*: Action Replay; and *R*: Rest time; *Com*: Commentators. Since we focus only actions in the game, rest time and commentators are not included in our strings.

An Example. Figure 1(b) shows initially the close view of the player1 is captured by camera A. The play is then captured by the court view. The player1 and player2 are at locations 8 and 9 respectively. This sequence can be represented as {A [U] C [U8 b8 V9 b3 Bv9 b5 BU8 b4 FV10 b5] D[ ]}. The player1 serves and the ball hits location 3 (Figure 1(c)). The player2 hits a backhand shot and the ball hits location 5 (Figure 1(d)). Again the player1 hits a backhand shot at location 8 returning the ball to location 4. The player2 hits a forehand shot at location 10 (Figure 1(e)) and the play ends with the ball going to location 5. This sequence is then replayed by camera D.



a) Tennis Court Segmentation


b) Close view of player


c) Player$_1$ serving


d) Player$_2$ hits a backhand shot


e) Player$_2$ hits a forehand shot

Figure 1. Locations of a tennis court and sample sequences [Internet Archive 2011]

Grammar for Tennis Game. SMART's generic grammar was extended to incorporate the semantics of tennis game. Figure 2 provides a grammar for the tennis videos: video is a sequence of clips; a sequence of clips has many clips; a clip has a camera view and a sequence of spatiotemporal instances; a sequence of spatiotemporal instances has a spatiotemporal instance; and a spatiotemporal instance (spt) is represented with an object (obj), location (loc), and an optional event.

```
<obj> ::=                    U | V | b
<event>::=                   F | B
<location> ::=               1|2|3|4|5|6|7|8|9|10|11|12|N
<camera> ::=                 <close-view camera>|C|D
<close-viewcamera>::=        A | B
<spt>:: =                    [<event>] <obj> <loc>
<seq of spt> ::=             <spt> | <spt> <seq of spt>
<clip> ::=                   < close-view camera >'['<obj>']' | C'['< seq of spt > ']' |D'['']'
<sequence of clips> :: =     <clip>| <clip> <seq of clips>
<video> :: =                       <seq of clips>
```

Figure 2. A grammar for tennis videos.

2.3 Spatio-Temporal Logic (STL)

Temporal Logic allows to analyze the properties of a system with respect to time. Linear Temporal Logic (LTL) [Vardi 2001] is a type of temporal logic, that is used to analyze a system that is considered to be composed of a vertex-labeled path $S_0,S_1,\ldots,S_n$, where each vertex S_i corresponds to a point in time as shown by Figure 3(a). We plan to use the spatio-temporal logic (STL) notation provided in [Del Bimbo 1995] with some modifications. Temporal assertion, Θ, on a state diagram (or state graph) σ, for a set of scene sequences is expressed as $\Theta:=(\sigma, S) \models \theta$ where S is a state in the state graph, σ, and θ is a temporal formula which is formed by combining spatial assertions Φ with Boolean connectives such as $\wedge$, $\vee$, $\neg$, $\rightarrow$ and temporal operators. A state is a set of object, location pairs. The temporal operators are global ($\Box$), next ($\circ$), eventually ($\Diamond$), until (U), and releases (R) operators. The temporal formula can be expressed as:

$$\theta := \Phi \mid \Box\Phi \mid \circ\Phi \mid \Diamond\Phi \mid \Phi_1 U \Phi_2 \mid \Phi_1 R \Phi_2 \mid \neg\Phi \mid \Phi_1 \vee \Phi_2 \mid \Phi_1 \wedge \Phi_2 \mid \Phi_1 \rightarrow \Phi_2$$

For a given state a graph, $\mathcal{G}$, these temporal operators are briefly explained as follows:

- **Global/ Always**. This operator is denoted as "G" or "$\Box$". GΦ or $\Box\Phi$, where Φ represents a propositional formula, implies that the condition specified by Φ should be satisfied through the path of vertices. ($\mathcal{G}$, S_1) $\models \Box\Phi$ implies that Φ is true for all states starting from state S_1 in graph $\mathcal{G}$ (Figure 3(b)).
- **Eventually/Finally.** This operator is denoted as "F" or "$\Diamond$". FΦ or $\Diamond\Phi$ implies that the condition specified by Φ should be satisfied eventually at a point of time in the given path of vertices. ($\mathcal{G}$, S_1) $\models \Diamond\Phi$ implies that Φ becomes eventually true after state S_1 in graph $\mathcal{G}$ (Figure 3(c)).
- **Next.** This operator is denoted as "X" or "$\circ$". XΦ or $\circ\Phi$ implies that the condition specified by Φ should be satisfied by the next vertex in the given path of vertices. ($\mathcal{G}$, S_1) $\models \circ\Phi$ implies that Φ becomes true in the state following state S_1 in graph $\mathcal{G}$ (Figure 3(d)).
- **Until.** This operator is denoted as "U". $\psi U\Phi$, where ψ and Φ represent propositional formulae, implies that the condition specified by ψ should be satisfied until a particular vertex in the given path of vertices has Φ satisfied. ($\mathcal{G}$, S_1) $\models \psi U\Phi$ implies that ψ is true until Φ becomes true starting from state S_1 in graph $\mathcal{G}$ (Figure 3(e)).
- **Releases.** This operator is denoted as "R". $\psi R\Phi$ implies that the condition specified by Φ should be satisfied through the path of vertices, until the first vertex in the given path of vertices, has ψ satisfied. ($\mathcal{G}$, S_1) $\models \psi R\Phi$ implies that Φ is true until ψ becomes true starting from state S_1 in graph $\mathcal{G}$ (Figure 3(f)). If there is no vertex that satisfies the condition specified by ψ, then the condition specified by Φ will be satisfied throughout the path of vertices.

In this research, the temporal operators 'Next' and 'Eventually' have been implemented for supporting the spatio-temporal querying on the S^3G, while 'Global' concept is used for optimizing the indexing of S^3G states built for a tennis video database as an application.

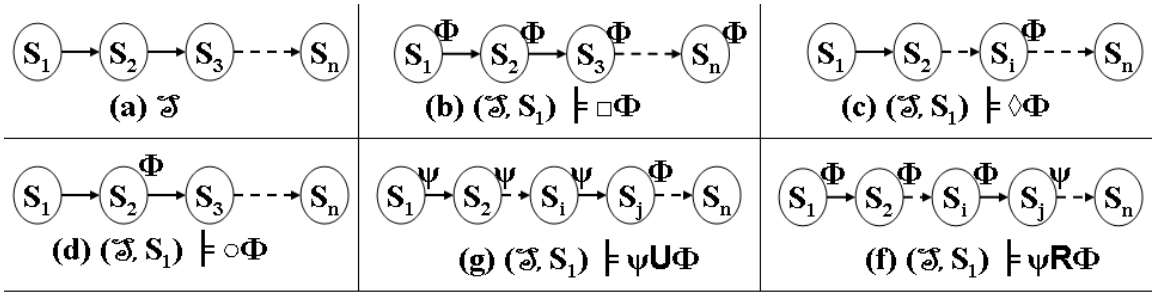


Figure 3. Temporal Operators (a) a sample sequence, (b) globally operator, (c) eventually operator, (d) next operator, (e) until operator, and (f) releases operator.

3. S^3G – A SEMANTIC SEQUENCE STATE GRAPH

S^3G is a graph where the events, objects, and locations are maintained as states and transitions. S^3G resembles to a non-deterministic finite state machine. An S^3G can be defined as $M=[S,\Sigma,\delta,s_0,F]$ where S is a set of internal states, Σ is a finite input alphabet, s_0 is the initial state, F is a set of external states, and δ is a transition function mapping $S \times \Sigma$ to $S \cup F$. An internal state s ($s \subseteq S$) is a set of object-location pairs. An internal state is a subset of cross product of objects spatial locations ($S \subseteq \Sigma_O \times \Sigma_L$). An external state $f \subseteq F$ corresponds to a decision in a sequence of events. The alphabet (Σ) is a subset of the cross product of the object-event pairs ($\Sigma \subseteq \Sigma_O \times \Sigma_E$). It should be noted that not all objects are associated with an event. S^3G is cyclic in nature. Each of its nodes represents a state giving the locations of the objects involved in the video. Each node has also a list of clips displaying the corresponding state.

3.1 Components

The components of SMART grammar have events, objects, locations and camera view as the alphabet. The most important part in building an S^3G is the identification of transition function and the states.

States. A state is identified by objects and their corresponding locations. In addition to object-location pairs, states also maintain pointers for quick retrieval of objects. For each state, there is a list of clip pointers to retrieve the corresponding clips having that state. The maximum number of states for a video is determined by $\prod_{i=1}^{|\Sigma_O|}(|\Sigma_L|) = |\Sigma_L|^{|\Sigma_O|}$.

Transitions. The transitions are determined by the semantic events. In our applications, the semantic events result in the displacement of objects. An action is generally continuous. Dividing an action into

discrete steps is critical in building an S^3G. Determining discrete steps is decided by semantic events/actions.

3.2 An Example: Tennis Video

A video can be considered as a collection of a series of events and a series of states due to these events. Some videos such as a sports video can have finite types of states and events. Since the details of S^3G can be expressed in a simple way using a tennis video, it is chosen as an S^3G application in this paper. For example, in a tennis game the most important object is the ball. Each event causes the ball to move from one location to another location. Whenever a player hits the ball, the ball changes its position. To reduce the number of states, we are interested only in specific situations. For example, we are interested in whenever a player hits the ball or the ball hits the court or net.

States of Tennis. In a tennis game, there are three objects: two players and the tennis ball. At a given instance, the objects with their locations define a state in the video. Since there are 3 objects and 13 locations identified for a tennis video, the total number of states is 13^3. However, it is unlikely to have these many states for a tennis game. Therefore, we create a state as long as that state exists in the database. The start states are generated based on the initial player positions. Each tennis video has a corresponding player 1 and another corresponding player 2.

Each SMART string corresponds to point level. Tennis game can be composed of five hierarchical levels as shown in [Gan 2006]: match-level, set-level, game-level, point-level, and stroke-level. Our strings correspond to the point-level but are composed of stroke-level information. Therefore, each state sequence for a string starts from a serve until one of the players make a mistake. While indexing a clip, we are interested in the plays. Therefore, other parts of the game such as commentators and slow motion replays are not indexed by S^3G.

Transitions of Tennis (Semantic Events). Now, principally, we can say that there can be two types of shots that the tennis players can make, i.e., the forehand shot and the backhand shot. Thus, we can define four types of events for a tennis video, namely a forehand shot by player1, a backhand shot by player1, a forehand shot by player2 and a backhand shot by player2. Hence, there are four types of transitions in a tennis video.

We can thus define a tennis video in terms of a series of states where each state consists of a value for the locations of player1, player2, and the tennis ball. Corresponding to the four types of events possible there can be four types of transitions.

Each transition changes a state with a set of location values for the players and the ball before that transition to another state with a new set of location values after the transition (Figure 4). The next state is represented as $S_{next}=t(A(O,S_{old}))$ where an event/action A by object O from state S_{old} leads to state S_{new}.

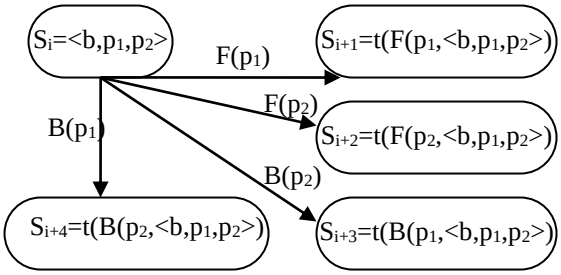


Figure 4. States and types of transitions of S^3G for a tennis video

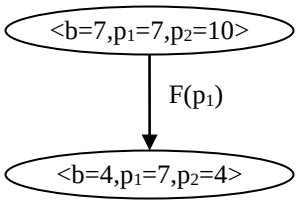


Figure 5. A transition that causes a new state

Example. Consider an initial state that has the values for spatial locations as 7, 10, and 7 for the player1, the player2, and the ball, respectively. This means that the player1 and the ball were in location 7, while the player2 was in location 10. Now, if the player1 hits a forehand shot and the ball goes to location 4 and so does the player2 to hit the ball back, then we will have a new state with values 7, 4, and 4 corresponding to the new location values of the player1, the player2, and the ball, respectively. In this case, the transition for the initial state to the new state should be defined as "player1 hits a forehand shot" ($F(p_1)$) (Figure 5).

Cyclic Nature. Consider a state with the values for spatial locations as 5, 6, and 5 for the player1, the player2, and the ball, respectively. Now, there can be a transition where the player1 hits a forehand shot and the ball goes to location 6 reaching a S^3G state with location values as 5, 6, and 6 for the player1, the player2, and the ball locations, respectively. In return to this player1's shot the player2 may hit a backhand shot hitting the ball to location 5, thus going back to the initial state in the S^3G as shown in Figure 6. Thus, S^3G is a cyclic graph.

List of States. A transition from a state can result in one of the multiple possible states. In the tennis video example, a transition from a state or a set of location values, such as a forehand shot by player1 can result in one of the multiple possible states. For example, the player1 may retain his position after hitting the shot or move to another position. Similarly the player2 can be in any of the locations in his court side. Also the player1 has multiple location options for hitting the ball. Hence for a given state, there should be a list of states for each of the four transitions as in Figure 7.

List of Clips. Now there can be many video clips for a given set of locations values for the player1, the player2, and the ball. So each state can have many video clips corresponding to it. Therefore an array of these clip ids should be maintained for the corresponding clips (Figure 8). In addition, the rank of the state in a clip is also stored along with the clip number. For simplicity, we do not show the ranks in the figures.

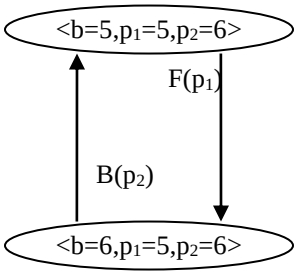


Figure 6. Cyclic nature of S^3G

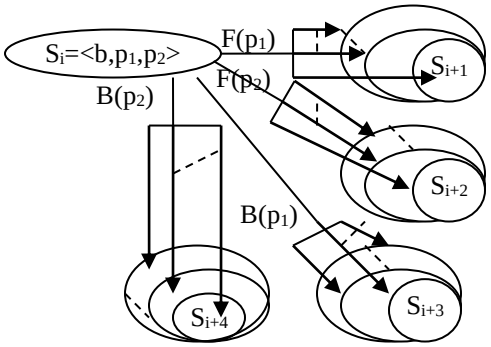


Figure 7. S^3G with a list of states for the different transition types

Finally each state in the S^3G will of the form: State S_i *:[*< $A_1,A_2,...A_n$ >,<C_i>*]*. Here A_i is an action pointer for next S^3G states and C_i is pointer pointing to a list of clips with ranks having the state S_i. For a tennis video, we have 4 action pointers. P_{P1f} and P_{P1b} point to the list of states that are possible after the player1 hits a fore-hand shot and a backhand shot, respectively. Similarly, P_{P2f} and P_{P2b} point to the list of states that are possible after the player2 hits a forehand shot and a backhand shot, respectively.

Complexity. The number of states may look like it grows exponentially with respect to the number of objects and locations. However, the number of states cannot be more than the number of frames in a video in the worst case. A state indicates possible positioning of objects at various locations in many video clips.

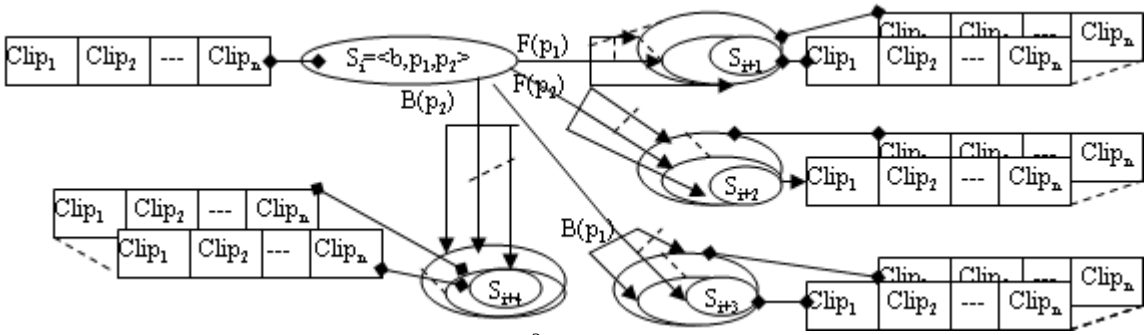


Figure 8. S^3G with a list of Clips

4. BUILDING S^3G

We build a S^3G from SMART [Jain 2008; Jain 2009] strings. For example, consider a string as U7V10b4FV10b8 that states that initially player1 and the ball are in location 7 and player2 is in location 10. The player1 serves the ball that goes to location 4. The player2 in location 10 hits the ball that goes to location 8. Two states can be defined as follows. The initial state has location values 7, 10, and 7 for player1, player2, and ball locations, respectively. The 'following' state has location values as 7, 10, and 8 for player1, player2, and ball locations, respectively. A transition from the initial state to the following state is defined as "player1 hitting a forehand shot".

The assumption used in string representations to reduce string sizes should be taken care of during converting them to state representation. For example, the ball location will be the same as the player location who is serving the ball. The players change their location 7 and 10 to 8 and 9 in alternating fashion, which will not be expressed explicitly in the string representations. For example, if the beginning of the video clip has player locations as 7 and 10, the 'next' play will have the player locations as 8, 9 and vice versa.

4.1 Insertion into S^3G

The S^3G is built in an incremental order as strings are being parsed. This is done by reusing the existing states in the S^3G and creating new states and transitions if required.

Assume that an S^3G as in Figure 9 is already built and we need to process a new substring as "U7b10FV10b7" for a clip with id C2134. This defines an initial state with location values as 10, 7, and 10 for the tennis ball, the player1 and the player2, respectively. Since a corresponding state in Figure 9 does not exist, a new state (S_4 in Figure 10) has to be created. Next, player2 hits a forehand shot and the ball goes to location 7. This corresponds to an existing state (S_1 in Figure 9) with location values 7, 10, and 7 for the tennis ball, the player1, and the player2, respectively. The clip id C2134 is added to the clip list of state S_1 as in Figure 10. In the second case, instead of creating a new state, initial state (S4) for clip C2134 is made to point to the existing state S1. The clip C2134 is added to the list of video clips of S_1.

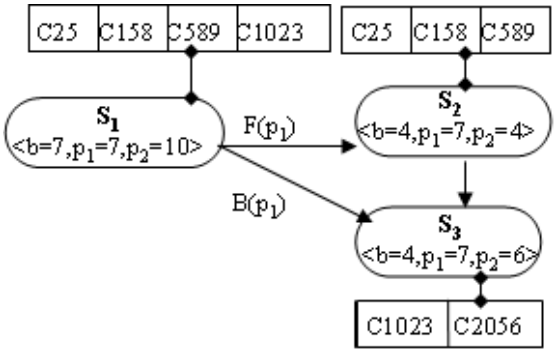


Figure 9. An S^3G graph before inserting clip C2134

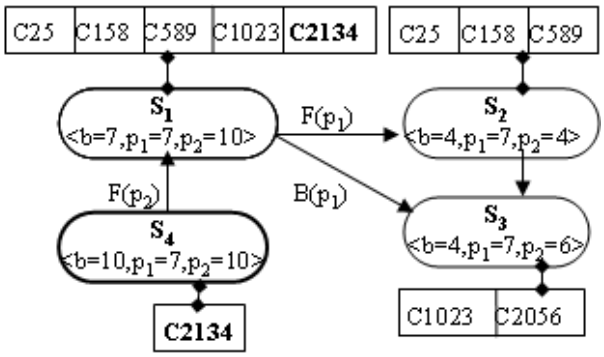


Figure 10. S^3G graph after inserting clip C2134 into a new state and an existing state

```
String2State(StrClip)
IN: string StrClip // applied for each string in the database
OUT: S3G graph
begin
  while not end of string
    read new alphabet α
    if (alphabet α ∈ (ΣO-{b})) // its location is not implied
       Parse the location of α
    endif
    if (alphabet α ∈ {b}) // ball is the last object for a state
       if location of α is implied
         Compute it from another object β // a player
         OP1 ← object-location pairs
           if no state for OP1
              create a state for OP1
              currentState ← OP1_state
           endif
       else
          parse the location of α
          OP2 ← object-location pairs
       endif
    else if (alphabet α ∈ ΣE) // α is an event (transition)
       // for serve, the event is assumed to be with forehand
       if no state is created for OP2
        create a new state for OP2
       endif
       nextState ← OP2_state
       Link currentState through α to nextState
    endif
  end while
end
```

Algorithm 1. Algorithm for converting strings to states.

Algorithm 1 provides the pseudo-code for converting from string to state. Here each alphabet from the SMART [Jain 2008; Jain 2009] string is read. If the alphabet represents an object (players or ball) and the object location is not implied, then the successive alphabets are read to get its location value. Once all the three object locations are determined, a state with these location values is searched using the indexing structure in Section 4.2.1. If the state is found, then it is linked with the previous state with suitable transition type represented by the alphabets in the string denoting the event, else a new state is created and added to the S^3G by linking it with the previous state using the transition specified by the event denoting alphabet. When the strings are parsed, the initial event is 'serve' (not explicitly represented in the string) and represented as a forehand shot as a transition in the graph.

Algorithm 2 determines whether an existing state, OP, in S^3G is reachable from an initial state S. First the system checks the presence of state OP. If OP is present, the set of common clip lists are found. If the intersection of clip lists is empty, state OP is not reachable from state S. If the intersection of clips is non-empty, the ranks of the same clips are compared. If the rank of a clip in state OP is more than the rank of

the clip in S, the state OP is reachable from state S. In the algorithm, C(S) returns the rank of a clip C in state S. This reachability function is used in querying.

```
bool Reachable(OP,S)
IN: Objects with Positions OP={(o1,p1), (o2,p2), …, (on,pn)}
IN: An initial state S
OUT: a boolean value reachable
begin
  check if OP is present using hashing
  reachable=false
  if OP is present then
       CLop = clip list of OP;
       CLs =  clip list of S;
       if (CLop ∩ CLs)=ø then
          reachable = false;
       else
          foreach C in (CLop ∩ CLs)
             if C(OP) > C(S) then
                reachable = true;
             endif
          endfor
       endif
   endif
  return reachable
end
```

Algorithm 2. Algorithm for searching states in S^3G.

A S^3G can grow significantly depending on the number of states and actions involved in a given video. The recursive algorithm for finding states discussed in the previous subsection may no longer be efficient for a S^3G with a large number of states. We use hashing for determining whether a state is present in the graph or not. States may not need to be created ahead of time. They may be created as those states are encountered in video clips. This will avoid the construction of an unnecessary large graph.

5. QUERYING

This section describes the interface to build S^3G and perform spatio-temporal querying on a S^3G. We generate one S^3G for the complete tennis video database. Information other than spatio-temporal content about the tennis games is stored in other tables in the database. The irrelevant clips can be eliminated as the states are visited. The user interface snapshots are provided in the Appendix.

5.1 Input, Backup, and Storage for S^3G

The string representation of the video according to SMART [Jain 2008; Jain 2009] is used as an input for building the S^3G states. The strings can be given as input by (a) manually typing the strings or (b) the SMART [Jain 2008; Jain 2009] strings stored in the GSMART database (Figure A1 in Appendix). S^3G can be mapped to a relational model and stored in the database. When the system restarts, it can easily be

built without processing the strings in the database. If the S^3G index is no longer needed, it can be removed from the database.

5.2 Searching a State

Once a S^3G is formed, a particular state can be queried with respect to object positions. If a state is present in the S^3G, the corresponding clips in the state can be viewed. The 'Location Selector' interface (Figure A3 in Appendix) helps to retrieve clips that are associated with a state. The location of an object is selected by first choosing the object (the player1, the player2, or the ball) and then clicking the location on the tennis court image corresponding to the required object's location. If a state exists in the S^3G, the details of that state (Figure A4 in Appendix) are displayed with a list of the clips associated with the state.

5.3 Querying

We can apply STL on the S^3G and construct spatio-temporal queries. In this paper, we provide examples of 'next' and 'eventually' operators that are applied on the tennis video database. When searching for a state sequence, it is important that the states in the sequence have the common clips. Moreover, the rank of a state for the clip should increase by one in a consecutive state. For example, if a sequence has states S_1 and S_2 that are searched, S_1 and S_2 must have at least one common clip. For the common clips, the rank should increase. For example, consider a partial S^3G in Figure 11 where ranks of states are provided as subscripts of clip numbers. Clip C1 has ranks 1 and 3 for state S_1, and rank 2 for state S_2. This indicates that the states of C1 are in the order of S_1-S_2-S_1. For C7, the order is S_2-S_1. For a sequence from S_1 and S_2 with a backhand shot by player 1, clip C5 will not be retrieved since clip C5 is not common in both clips. Clip C7 is common in both states, but the order of states is not correct. Clip C1 is retrieved because it has a rank 1 for state S_1 and rank 2 for state S_2. This means that S_2 follows S_1 with a backhand shot by player 1 in clip C1.

In the following subsections, we do not delve into ranks assuming that rank conditions are satisfied. We put more emphasis on how temporal operations are handled.

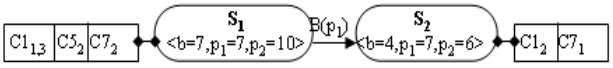


Figure 11. A partial S^3G.

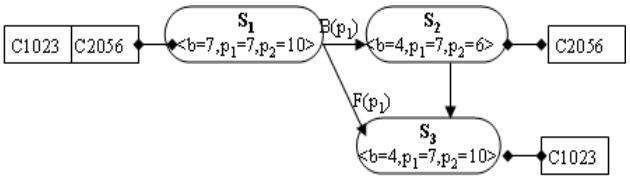


Figure 12. An instance of S^3G

5.3.1 'Next' State Query

The 'next' query helps us retrieve all the possible next states from a given state in the S^3G. Each state in the S^3G can have multiple next states for each of the four kinds of transitions.

Example. In the given instance of S^3G in Figure 12, the state S_2 occurs following S_1 when player1 hits a backhand shot. The state S_3 occurs following S_1 when player1 hits a forehand shot. Therefore, a 'next' query for state S_1, should fetch states S_2 and S_3:

$(\mathbf{S},S_1) \models \circ (S_2)$ and $(\mathbf{S},S_1) \models \circ (S_3)$.

For each of the four transitions types (player1 hitting forehand shot, player2 hitting forehand shot, player1 hitting backhand shot and player2 hitting backhand shot), the 'next' possible states will be retrieved. The interface for querying 'next' states are provided in Appendix A.5. The 'next' state query can also be applied multiple number of times in succession to obtain the clips with all the selected states.

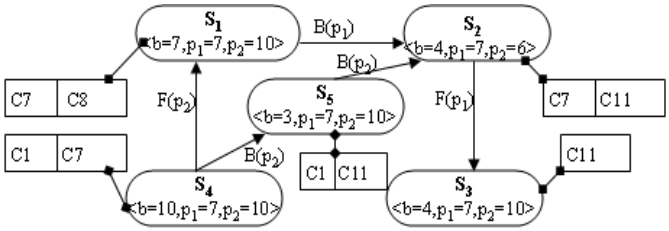


Figure 13. An instance of S^3G

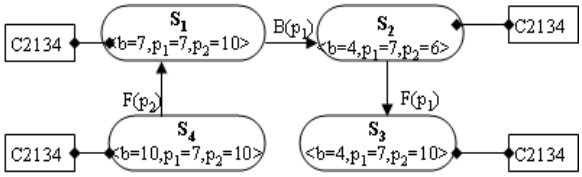


Figure 14. An instance of S^3G

Example. In Figure 13, if S_4 is selected as the current state, its next states will be S_1 and S_5. The states S_4 and S_1 have clip C7 as a common clip, while the states S_4 and S_5 have clip C1 as common. If S_5 is selected as the required 'next' state and the 'next' state is applied to S_5, the state S_2 will be obtained as they have clip 11 as common, but state S_4 does not have this clip. Hence, no clips are retrieved.

5.3.2 'Eventually' State Query

The 'eventually' query helps users check if a state eventually occurs in future after a particular state has already occurred in the S^3G. For given two S^3G states, an initial state and a final state, it can be checked if the final state eventually occurs after the initial state has occurred by checking if it is possible to reach the final state from a given start state in the S^3G .

Example. Figure 14 provides an instance of S^3G where S_1 occurs following the state S_4 when player2 hits a forehand shot. The state S_2 occurs following state S_1 when player1 hits a backhand shot. The state S_3 occurs when player1 hits a forehand shot after the state S_2. Therefore, running an 'eventually' query with S_4 as a start state and S_3 as a final state succeeds while an eventually query with S_2 as a start state and S_4 as a final state does not:

$(\mathbf{S},S_4) \models \Diamond (S_3)$ but $(\mathbf{S},S_2) \not\models \Diamond (S_4)$ is false.

Consider the example shown in Figure 15 where a state with locations 7, 10, and 4 are set for the player1, the player2, and the ball positions, respectively for the initial state. When the user clicks "Set as

initial state", this initial state is shown as a thumbnail in Figure 15. To check if a state 'eventually' occurs after the initial state chosen, the 'Eventually Check' in Figure 16 is selected (see Appendix A.6 for more information).

5.3.3 Combining 'Next' and 'Eventually' Queries

The 'next' and the 'eventually' state queries can be called multiple times in any order. In Figure 17, an instance of S^3G is given where S_1 is a state that is directly reachable from the state S_4 after player2 hits a forehand shot. The state S_2 is directly reachable from S_1 after Player1 hits a backhand shot. The states S_5 and S_3 are directly reachable from the state S2 after the transitions resulting from "player1 hitting a forehand shot" and "player2 hitting a backhand shot" respectively. Therefore, an 'eventually' query with S_4 as the start state and S_2 as the final state will succeed. A 'next' query on the state will retrieve states S_3 and S_5.

$$(\mathbf{S},S_4) \models \Diamond (S_2) \wedge (\mathbf{S},S_2) \models \Diamond (S_3) \text{ and } (\mathbf{S},S_4) \models \circ (S_2) \wedge (\mathbf{S},S_2) \models \circ (S_5)$$

Similarly a 'next' query on the state S_4 gives the state S_1 and then an 'eventually' query with S_1 as the start state and S5 as the final state will also succeed.

$$(\mathbf{S},S_4) \models \circ (S_1) \wedge (\mathbf{S},S_1) \models \Diamond (S_5)$$

Our application has the feature of graphically displaying the states of a query to simplify the process of long queries that result from a series of eventually & next state queries as shown in Figure 18. When a new query is submitted, the image of previous state will be retrieved from a location where all possible state images are stored. This image is then displayed. Thus, S^3G application helps to develop queries interactively. The user can continue building queries on the results obtained from his previous sub-queries, thus enabling creation of lengthy queries dynamically instead of running just a fixed query.

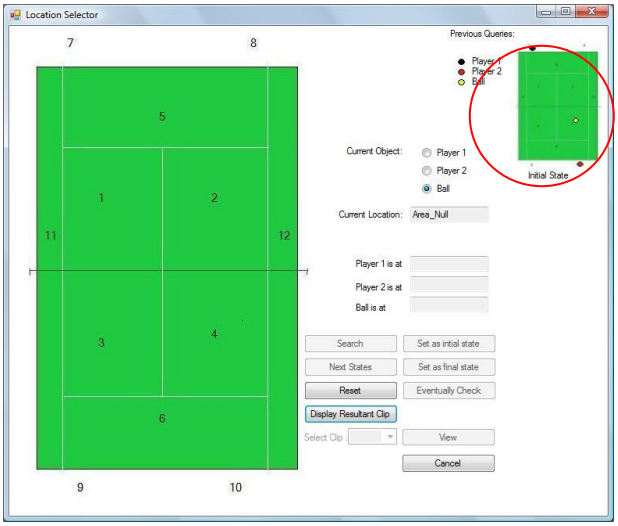


Figure 15. Initial state is set.

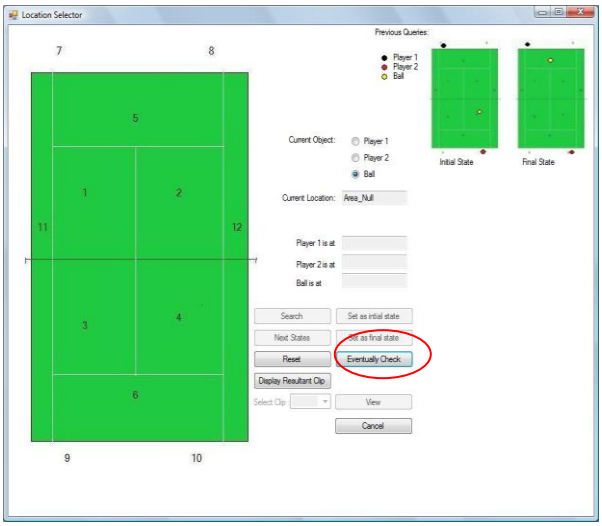


Figure 16. 'Eventually' checking.

Other query types using ‘releases’ or ‘until’ operator can also support queries such as “retrieve clips where both players are at the baseline until one player runs to the net.” In our application, these queries can be represented with a sequence of ‘next’ queries. So, this type of queries is not implemented separately.

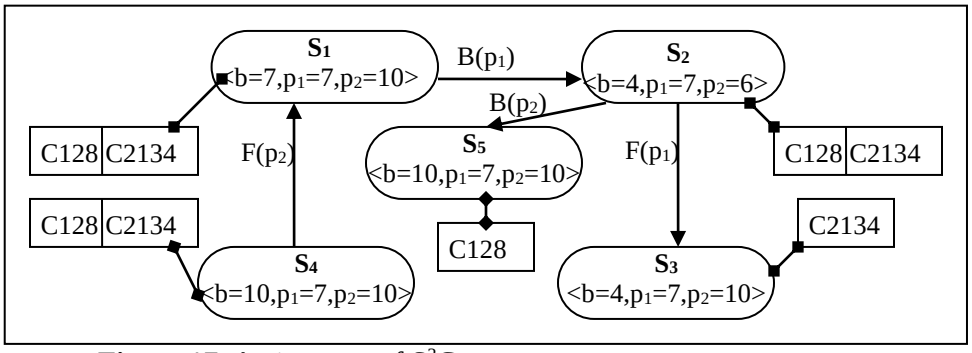


Figure 17. An instance of S^3G

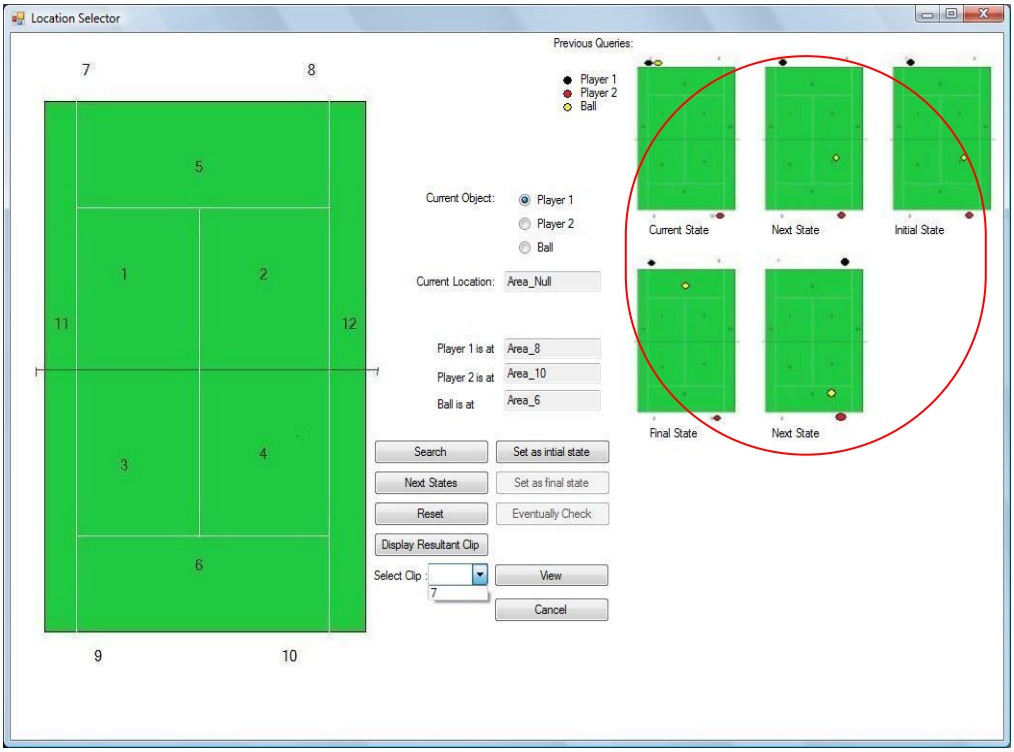


Figure 18. Query building history.

6 EXPERIMENTS

In this section, we first provide experimental results and then provide discussion on extendibility for other videos.

6.1 Performance

Comparison. We compare our method with Spatio-temporal Query Patterns (STP) (Hadjieleftheroui, 2005). There are improvements to STP method with assumptions (Lagogiannis, 2010) which are not applicable to our case (e.g., an object should not visit the same location again). STP provides indexing based on positions usually divided into grids. STP index keeps the time when objects visit a specific location. There are several issues with STP: a) STP supports querying the trajectory of a single object, b) there is a single timeline for trajectories, c) there are no semantic events between changing positions, and d) the gap between any consecutive steps could be anything. To compare our method with STP, the following changes are made for STP. 1) Our queries include multiple objects. Therefore, we first need to query STP for each object and check the results are synchronized (i.e., check whether objects appear at the corresponding positions at the same time). 2) After each tennis point (tennis match is composed of sets, which are composed of games, which are composed of points), the positions of players are reset. There is no need to check the positions of objects across points. We need to keep STP index structure for each point. A sequence of positions across multiple points is not related to each other and meaningless. 3) We consider all semantic transitions (all types of shots) for S^3G since STP does not support transitions. 4) We consider eventual queries to compare S^3G with STP since STP is not designed to support next-querying.

Building S^3G. Since a small number of games are not enough to check the performance of indexing structures, we have simulated tennis games to measure the complexity of building S^3G and querying using S^3G. We have generated around 10,000 points (sequences or strings or clips). Each game has around 100 points. Table I shows the statistics for building S^3G and STP index structures where |IS| is the number of input sequences (or strings) and |s| is the number of states (note |s| is only applicable to S^3G). These timings only include the time for building the index structures and ignore the time for decoding the input strings since strings are decoded differently for each index structure.

|IS| or |q|	|s|	S3G Indexing	STP Indexing	S3G Searching		STP Searching	
				3 states	2 states	3 states	2 states
1	10	0.003	0.003	0.001	0.001	0.020	0.015
2	13	0.003	0.003	0.001	0.001	0.028	0.024
4	19	0.003	0.003	0.001	0.001	0.047	0.040
8	27	0.003	0.003	0.002	0.001	0.076	0.087
16	52	0.003	0.003	0.002	0.002	0.132	0.140
32	82	0.003	0.003	0.003	0.003	0.245	0.248
64	129	0.003	0.003	0.006	0.005	0.473	0.461
128	178	0.003	0.003	0.010	0.008	0.932	0.866
256	217	0.004	0.004	0.019	0.010	1.876	1.638
512	249	0.005	0.006	0.037	0.015	3.735	3.143
1024	287	0.010	0.009	0.073	0.025	7.570	6.175
2048	332	0.014	0.016	0.145	0.049	15.082	12.446
4096	370	0.021	0.029	0.297	0.085	30.104	24.672
8192	387	0.035	0.070	0.600	0.153	59.882	49.496

Table I. Time for building index structures and searching with 2 and 3 states for S^3G and STP.

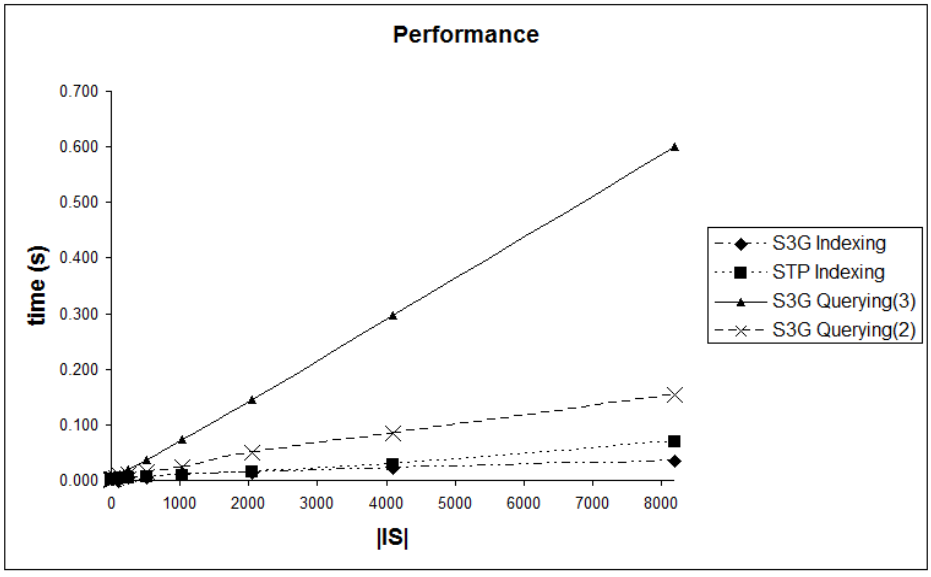


Figure 19. Time for building index structures.

Complexity. It takes 35ms to build S^3G for 10,000 points. Table I shows that the number of states reaches saturation when 4096 input sequences are processed and barely increases after that. Our experimental results indicate that the time complexity is $O(|IS|)+O(|s|)$. Since $|IS|>>|s|$ and the number of states reaches a saturation point, the complexity of building S^3G is $O(|IS|)$ for large databases. The time complexity of inserting a state into S^3G is O(1). Similarly, the time complexity of inserting a clip into a state in S^3G is $O(1)$: finding the state using hashing and adding the clip to the end. Figure 19 shows the comparison of building index structures. Building S^3G is faster than STP despite S^3G also includes the

time to build the index structure with respect to the player shots whereas in STP those transitions are ignored. Although a separate STP index is built for each clip (this results in simple STP index structures), building a complex and single S^3G index was faster than building STP index structures. If a single STP index structure had been built, the querying would not complete in a reasonable time for STP index.

Querying. We compare the performance of S^3G and STP on 'eventually' queries. Table I provides the experimental results when 10,000 points are inserted into the database where |q| indicates the number of queries. A query returns the clips where a destination state is reachable from an initial state. We provide the results with 2 and 3 states (or predicates) for 'eventually' queries. S^3G significantly outperforms STP that we cannot put them into a single graph. Using S^3G, 8196 queries for 3 states are completed in 600ms whereas it takes around 60 seconds for STP. Using S^3G for queries where there is an initial state and a final state just took 153ms. Figure 19 shows the results of querying for S^3G. Note that the time for querying provides the total time for |q| queries. The cost of a 'Next' query is O(1)+O(1). The first part locates the current state using hashing and the second part locates the next state using the corresponding event.

6.2 Discussion on Extendibility

The domain of applications should have the following properties: a) the video data should be discretizable in both spatial domain and temporal domain; b) the states should be repeated to get benefit from this indexing method, and c) there should be preferably semantic events (or transitions) that indicate change from one state to another state. The transitions and discretization are closely related to each other. Now, we give examples of these in several different domains.

We may group sports based on the layout of the field, instruments involved in the game, the number of players, and so on. In some sports ranking is important: auto racing, swimming, and cycling. The events or transitions include completion of a lap, taking a pit stop, or passing of a player by another player. The states may involve the rank of players, the lap, and their locations on the field. A combination of lap and rank could be a better option for auto racing rather than the exact locations of cars. Another set of group of games include games where the field is divided into multiple sections. Examples include tennis, table-tennis, volleyball, and badminton. Tennis example is already covered with some detail. For volleyball the events could be passing, block, or spike etc. Another set of games include games that are played on a board (e.g., chess). In chess, player moves are transitions, and pieces on board make the states. In a single game, it may be rare to repeat states but it is possible to have states repeated in multiple games. 8 moves of a chess game by Kasparov against World Team are: "1.e4 c5 2.Nf3 d6 3.Bb5+ Bd7 4.Bxd7+ Qxd7 5.c4 Nc6 6.Nc3 Nf6 7.0-0 g6 8.d4 cxd4. "

Another set of games include games where players have almost no restriction to move on the field. These games include soccer, football, basketball, ice hockey, etc. The research on these games includes video processing, game summary processing, and commentator speech processing. Not all players might be involved at a play during the game. The positions of players as well as their actions might be important. We investigated games from National Football League (NFL). The critical parts of the state are: the ball location, team having the ball, down-and-distance, and optional team formation (e.g., State(Team: GB; Ball: NO-37; Down; 1-10; Formation: shotgun)). Events have more information about the event type such as rushing, passing, sacks. Events may also have information who is involved in the event (e.g., player A passes the ball to player B which is then stopped by player X of the opponent team). In baseball, the players at the bases, pitcher, batter, catchers, etc. form the states. Home-run, strike, ball, etc. correspond to transitions. In the golf, the player follows holes. The position of the ball after each hit and the hole number may correspond to a state. The hit may correspond to a transition.

If the number of states grows significantly, we propose two ways of reducing the number of states: a) decrease the number of locations by increasing the sizes of locations and b) use only players who are active in the play for that state.

7 CONCLUSION

In this paper, we introduced a new indexing method, semantic sequence state graph (S^3G), for spatio-temporal data using the semantic contents of the video. S^3G utilizes semantic events as transitions in the events while the spatial information is maintained in states. The semantic events result in temporal ordering among the states. The states are indexed with an indexing structure similar to K-d trees. The 'next' and 'eventually' operations of LTL have been implemented using the S^3G platform on the tennis video database. Both these types of queries can be applied multiple times in combination to form a complex query. A sample application of S^3G is shown for a tennis video database application. As future work, we plan to apply S^3G in other domains. We also plan to provide more efficient ways of managing temporal queries of S^3G. We also intend to create new scenarios of events, objects, and locations by reusing the video clip contents and traversing S^3G. S^3G can handle coupled STQ with combination of 'next' and 'eventually' queries. S^3G suits well for applications where there is a finite domain of objects, events, and locations where repetitions are frequent. S^3G also helps the user develop the temporal components of a query by a user interface that shows a history of the built query. The user can see whether his query will lead to any result or not while building his query.

ACKNOWLEDGEMENT

We thank Vineetha Bettaiah for improving the code for searching states, incorporating rank of clips, and editing of the paper.

REFERENCES

ALLEN, J. F. 1983. Maintaining knowledge about temporal intervals. In: Communications of the ACM. 26/11/1983. ACM Press

ASSFALG, J., BERTINI, M., COLOMBO, C., AND DEL BIMBO, A. 2002. Semantic Annotation of Sports Videos. IEEE MultiMedia 9, 2 (Apr. 2002), 52-60.

ATHITSOS, V., PAPAPETROU, P., POTAMIAS, M., KOLLIOS, G., AND GUNOPULOS, D. 2008. Approximate embedding-based subsequence matching of time series. In *Proceedings of the 2008 ACM SIGMOD international Conference on Management of Data* (Vancouver, Canada, June 09 - 12, 2008). SIGMOD '08.

AYGUN, R.S. AND YAZICI, A. 2004 "Modeling and management of fuzzy Information in Multimedia Database Application", Multimedia Tools and Applications, Vol. 24, No 1, p.p. 29-56 ,2004

BERCHTOLD, S., KEIM D.A., AND KRIEGEL, H-P. 2002. The X-tree: An Index Structure for High-Dimensional Data, Readings in Multimedia Computing and Networking, Morgan Kaufmann, San Francisco, 2002, Pages 451-462

BOZANIS, P. AND FOTEINOS, P. 2007. WeR-trees, Data & Knowledge Engineering, Volume 63, Issue 2, November 2007, Pages 397-413

BÖHM, C., BERCHTOLD, S., AND KEIM, D. A. 2001. Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. *ACM Comput. Surv.* 33, 3 (Sep. 2001), 322-373.

CHEN, S., OOI, B., TAN, K., AND NASCIMENTO, M. A. 2008. ST^{2}B-tree: a self-tunable spatio-temporal b$^+$-tree index for moving objects. In *Proceedings of the 2008 ACM SIGMOD international Conference on Management of Data* (Vancouver, Canada, June 09 - 12, 2008). SIGMOD '08

DEL BIMBO, A.; VICARIO, E.; ZINGONI, D. 1995, "Symbolic description and visual querying of image sequences using spatio-temporal logic," *Knowledge and Data Engineering, IEEE Transactions on* , vol.7, no.4, pp.609-622, Aug 1995

GOOGLE VIDEO 2011. Google Video, http://video.google.com/

HADJIELEFTHERIOU, M., KOLLIOS, G., BAKALOV, P., AND TSOTRAS, V.J. 2005. Complex spatio-temporal pattern queries. In *Proceedings of the 31st international conference on Very large data bases* (VLDB '05). VLDB Endowment 877-888.

HONGENG, S., NEVATIA, R., AND BREMOND, F. 2004. Video-based event recognition: activity representation and probabilistic recognition methods. Comput. Vis. Image Underst. 96, 2 (Nov. 2004), 129-162.

INTERNET ARCHIVE 2011. Internet Archive, http://www.archive.org/

JAIN, V. AND AYGUN, R.S. 2008. "SMART: A grammar -based semantic video modeling and representation" , IEEE SouthEast Con 2008

JAIN, V. AND AYGUN, R.S. 2009. "Spatio-Temporal Querying of Video Content Using SQL for Quantizable Video Databases, Journal of Multimedia (to appear)

JENSEN, C.S., LIN, D., AND OOI, B.C. 2004. Query and Update Efficient B+-Tree Based Indexing of Moving Objects. In *VLDB*, pages 768–779, 2004.

JENSEN, C.S. AND SNODGRASS, R. T. 1999. Temporal Data Management. IEEE Trans. on Knowl. and Data Eng. 11, 1 (Jan. 1999), 36-44.

KIM, C., VASUDEV, B. 2005, "Spatiotemporal sequence matching for efficient video copy detection," *Circuits and Systems for Video Technology, IEEE Transactions on* , vol.15, no.1, pp. 127-132, Jan. 2005

KOPRULU, M., CICEKLI, N. K., AND YAZICI, A. 2004. Spatio-temporal querying in video databases. *Inf. Sci. Inf. Comput. Sci.* 160, 1-4 (Mar. 2004), 131-152.

LAGOGIANNIS, G., LORENTZOS, N., SIOUTAS, S. AND THEODORIDIS, E. 2010. A time efficient indexing scheme for complex spatiotemporal retrieval. *SIGMOD Rec.* 38, 3 (December 2010), 11-16.

LAY, J.A.; GUAN, L. (2006), "Semantic retrieval of multimedia by concept languages: treating semantic concepts like words," *Signal Processing Magazine, IEEE* , vol.23, no.2, pp.115-123, March 2006

LEE, M., YOON, H., KIM, Y. J., AND LEE, Y. 2008. SMILE tree: a stream data multi-query indexing technique with level-dimension nodes and extended-range nodes. In *Proceedings of the 2nd international Conference on Ubiquitous information Management and Communication* (Suwon, Korea, January 31 - February 01, 2008). ICUIMC '08

LEJSEK, H., ÁSMUNDSSON, F.H., JÓNSSON, B.Þ., AND AMSALEG, L. 2009 "NV-Tree: An Efficient Disk-Based Index for Approximate Search in Very Large High-Dimensional Collections," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 5, pp. 869-883, May, 2009

LI, J.Z., OZSU, M.T., AND SZAFRON, D. 1997. Modeling of moving objects in a video database, 1997 International Conference on Multimedia Computing and Systems, P.P. 336, 1997

LIN, D., JENSEN, C. S., OOI, B. C., AND ŠALTENIS, S. 2005. Efficient indexing of the historical, present, and future positions of moving objects. In Proceedings of the 6th international Conference on Mobile Data Management (Ayia Napa, Cyprus, May 09 - 13, 2005). MDM '05. ACM, New York, NY, 59-66.

MIN, J.S., KIM, D.H., RYU, K.H. 2001. A spatiotemporal data and indexing, Electrical and Electronic Technology, 2001. TENCON. Proceedings of IEEE Region 10 International Conference on , vol.1, no., pp.110-113 vol.1, 2001

NAIK, M., JAIN, V., AND AYGUN, R.S. 2008. "S^3G: A Semantic Sequence State Graph for Indexing Spatio-temporal Data - A Tennis Video Database Application," ICSC, pp. 66-73, 2008 IEEE International Conf on Semantic Computing, 2008

PARK D-J, HEU S., AND KIM H-J 2001, The RS-tree: An efficient data structure for distance browsing queries, Information Processing Letters, Volume 80, Issue 4, 30 November 2001, Pages 195-203

PATEL, J. M., CHEN, Y., AND CHAKKA, V. P. 2004. STRIPES: an efficient index for predicted trajectories. In *Proceedings of the 2004 ACM SIGMOD international Conference on Management of Data* (Paris, France, June 13 - 18, 2004). SIGMOD '04. ACM, New York, NY, 635-646.

PISSINOU, N., RADEV, I., MAKKI, K., AND CAMPBELL, W.J. 2001. Spatio-Temporal Composition of Video Objects: Representation and Querying in Video Database Systems, IEEE Transactions on Knowledge and Data Engineering, Vol. 13, No 16, P.P. 1033-1040, 2001.

REN, W., SINGH, S., SINGH, M., AND ZHU, Y. S. 2009. State-of-the-art on spatio-temporal information-based video retrieval. *Pattern Recogn.* 42, 2 (Feb. 2009), 267-282.

ŠALTENIS, S., JENSEN, C. S., LEUTENEGGER, S. T., AND LOPEZ, M. A. 2000. Indexing the positions of continuously moving objects. *SIGMOD Rec.* 29, 2 (Jun. 2000), 331-342.

SALTENIS, S. AND JENSEN, C.S. 2002. Indexing of Moving Objects for Location-Based Services, ICDE, 2002: 463-472

SALZBERG, B. AND TSOTRAS, V. J. 1999. Comparison of access methods for time-evolving data. ACM Comput. Surv. 31, 2 (Jun. 1999), 158-221.

TAO, Y., PAPADIAS, D., AND SUN, J. 2003. The TPR*-tree: an optimized spatio-temporal access method for predictive queries, in: Proceedings of the 29th International Conference on Very Large Data Bases (VLDB), Berlin, Germany, 2003, pp. 790–801.

VARDI, M.Y. 2001. Branching vs. Linear Time: Final Showdown. Proceedings of the 7th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'01), 2031 (2001), 1-22.

WATTAMWAR, S. S. AND GHOSH, H. 2008. Spatio-temporal query for multimedia databases. In *Proceeding of the 2nd ACM Workshop on Multimedia Semantics* (Vancouver, British Columbia, Canada, October 31 - 31, 2008). MS '08. ACM, New York, NY, 48-55.

YE, H., LUO, H., SONG, K., XIANG, H., AND CHEN, J. 2007. Indexing moving objects based on 2^n index tree. In *Proceedings of the 6th Conference on 6th WSEAS int. Conf. on Artificial intelligence, Knowledge Engineering and Data Bases - Volume 6* (Corfu Island, Greece, February 16 - 19, 2007).

YIU, M. L., TAO, Y., AND MAMOULIS, N. 2008. The Bdual-Tree: indexing moving objects by space filling curves in the dual space. *The VLDB Journal* 17, 3 (May. 2008), 379-400.

YOUTUBE 2011. YouTube – Broadcast Yourself, http://www.youtube.com/

APPENDIX

In Appendix, we provide information about our user interface for querying.

A USER INTERFACE

In this section, we provide some snapshots from our user interface.

A.1 Building S^3G

Figure A1 displays the user interface where SMART strings are read from the database. When "Proceed to State Conversion" button is clicked, the S^3G is built.

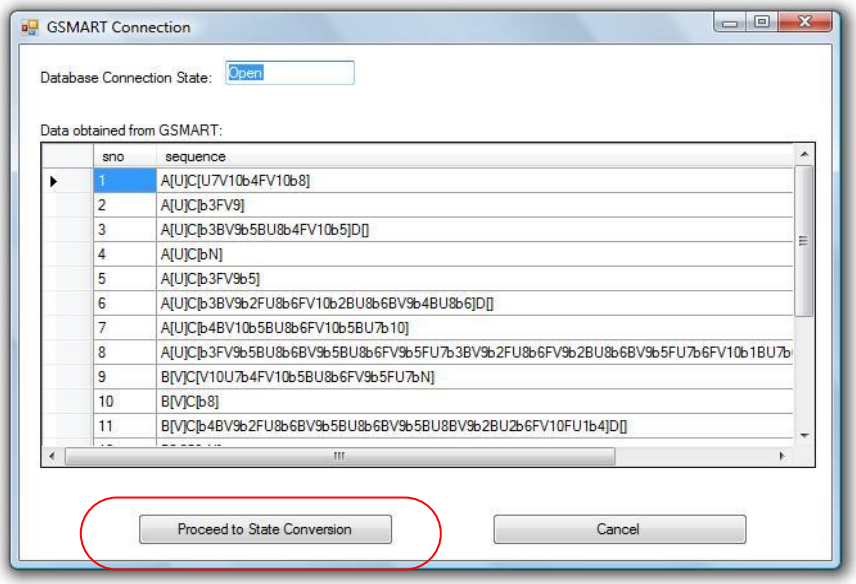


Figure A1. Interface showing the strings obtained from GSMART database

A.2 Displaying States

The interface is prepared based on design in Figure A2 where player1, player2, and the ball images are placed on all possible locations on the tennis court image. All these object images are hidden except the ones that represent the location values for the given state when displaying a state.

A.3 Location Selector Interface

Figure A3 shows Location Selector interface filled with location values for a desired state. In this example, player1 location has a location as 7, player2 has a location as 10, and the ball has a location as 4. To search for a particular state with corresponding location values in the 'Location Selector' interface, the search option in Figure 15 is selected. To retrieve states based on an event, the user needs to follow the links to retrieve the corresponding links. For example, if the user wants to retrieve the clips that result

after an event such as forehand shot by player1, the transition for this event followed and the clips from the relevant states are retrieved.

A.4 Viewing Clips

A particular clip can be selected to view from the list of clips given for that state (Figure A4). The clip will be displayed after pressing the view button. If the selected state does not exist in the S^3G, a message will be displayed saying that the specified state does not exist. Figure A4 also shows player1, player2, and ball locations.

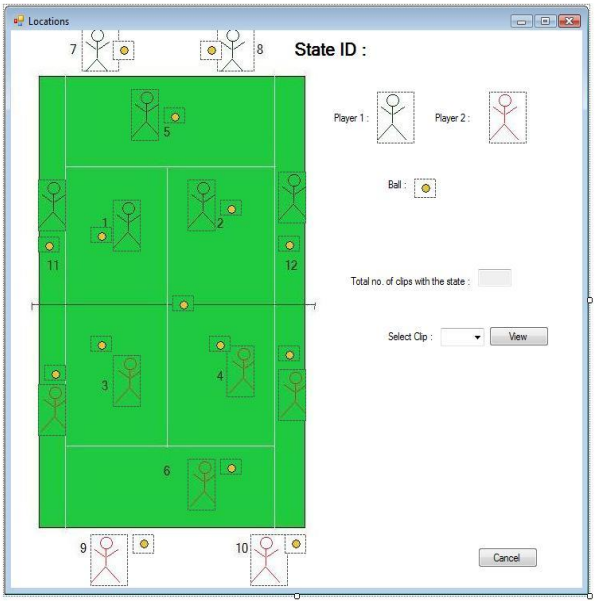


Figure A2. Player1, player2 and ball images on all possible locations on the tennis court

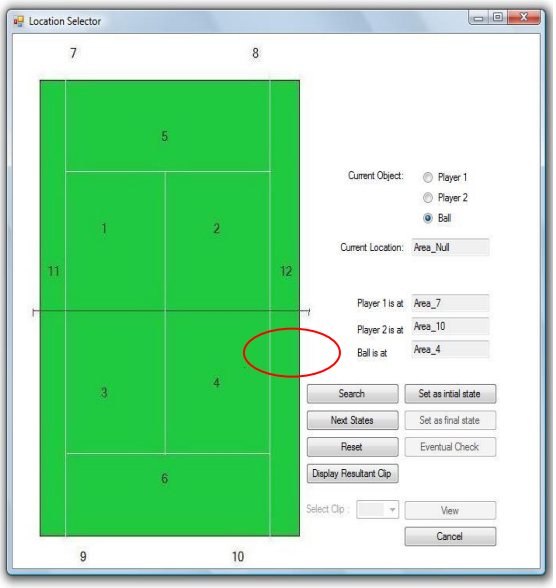


Figure A3. Location Selector interface State with entries filled for player1, player2 and ball locations

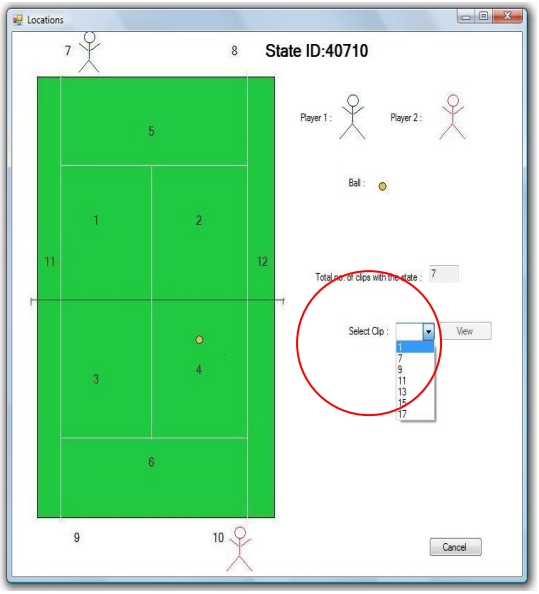


Figure A4. Interface showing details of 40710

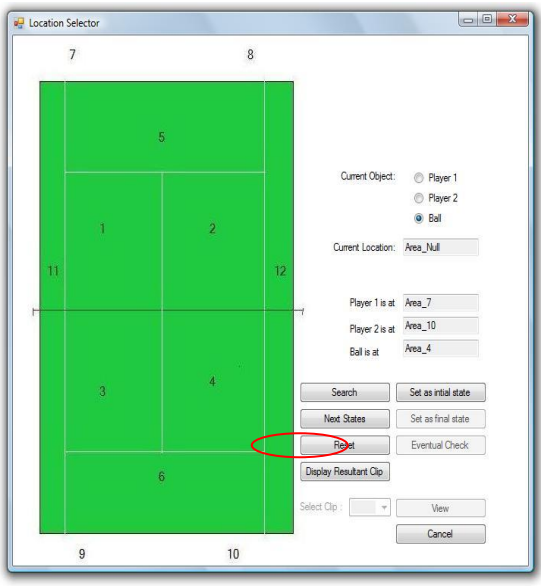


Figure A5. Location Selector interface with next 'state'

A.5 Interface for Querying Next States

A state can be chosen by selecting the player1, the player2 and the ball locations in the tennis court and then corresponding 'next' states in the S^3G can be obtained (Figure A5). The aim is to obtain the clips having the current state and the selected next state.

Figure A6 shows an interface showing the possible 'next' states in the S^3G for the transition, when player2 hits a forehand shot. A 'next' state can be selected by choosing the 'Select State' Options button associated with that state. If the next state does not exist in the S^3G for a transition then a message will be displayed stating that there is no next state for that transition. Once a current state and a next state are selected, the relevant clips can then be viewed (Figure A7).

Only the next states that have common clips with the relevant clip set (clips common to all selected states) are displayed to select. There may be a case where there can be a next state existing for a state in the S^3G but the next state has no common clip with all previously selected states. In such a case, a message stating that "the next state exists in the S^3G, but no common clip exists" is displayed.

A.6 Interface for Querying Eventually

The result, whether the 'eventually' link exists between the chosen initial and final states in the S^3G or not, is obtained by selecting the "Result" option as in Figure A8. Figure A8 shows that an 'eventually' link exists for the chosen initial and final states in the S^3G.

Figure A6. Interface showing 'next' states when player2 hits a forehand shot

Figure A7. Interface showing the relevant clip having the two states (current and 'next' states on top right corner)

Figure A8. Result of 'eventually' check.