All Pairs Shortest Path Formulation for Multiple Object Tracking with Application to Tennis Video Analysis

F.Yan W.Christmas J.Kittler
Centre for Vision, Speech and Signal Processing
University of Surrey
Guildford, GU2 7XH, UK
{f.yan,w.christmas,j.kittler}@surrey.ac.uk

Abstract

In previous work, we developed a novel data association algorithm with graph-theoretic formulation, and used it to track a tennis ball in broadcast tennis video. However, the track initiation/termination was not automatic, and it could not deal with situations in which more than one ball appeared in the scene. In this paper, we extend our previous work to track multiple tennis balls fully automatically. The algorithm presented in this paper requires the set of all-pairs shortest paths in a directed and edge-weighted graph. We also propose an efficient All-Pairs Shortest Path algorithm by exploiting a special topological property of the graph. Comparative experiments show that the proposed data association algorithm performs well both in terms of efficiency and tracking accuracy.

1 Introduction

In automatic video annotation, high-level descriptions rely on low-level features. In the context of a ball game, such as cricket, football, tennis, table tennis or snooker, the trajectory of the ball provides important information for high level annotation. Indeed, reconstructing the ball trajectory is essential for a complete understanding of a ball game. However, tracking a ball in a complex scene can be a difficult task. In the case of tennis ball tracking, the ball's small size, high velocity, abrupt motion change, occlusion, and the presence of multiple balls all pose strong challenges. The scope of this paper is to develop a robust algorithm for tracking tennis balls in broadcast tennis video.

Let us assume we have a ball candidate generation module, where a ball is detected as a candidate with a certain probability along with some clutter-originated false positives. The data association problem, *i.e.* the problem of determining which candidates are ball-originated and which are clutter-originated, is the key problem to solve in tennis ball tracking. In [5], a data association algorithm was proposed under the name of Robust Data Association (RDA), and was used to track a tennis ball. The key idea of RDA is to treat data association as a dynamic model fitting problem. In RDA, a RANSAC [2] paradigm is employed. A sliding window containing several frames is moving over a sequence. Candidate triplets are randomly drawn from all the candidates in the current interval, and are used to fit dynamic models. The fitted models are evaluated using a cost function. The model is found that is best at explaining the candidates inside the interval. An estimate of the ball position in one frame, *e.g.* the middle frame in the interval, is then given by this model. As the sliding window moves, eventually ball positions in all frames are estimated.

RDA works well under moderate clutter level, and when certain assumptions are satisfied. However, several weaknesses of RDA have been noticed (see [8] for details). Inspired by RDA's model fitting approach to the data association problem, in our previous work [8], we proposed a two-layer data association algorithm (dubbed L^{2}DA in this paper) to remedy some of the weaknesses of RDA. Although L^{2}DA provides improved speed and robustness over RDA, like RDA, it is a single-object tracking algorithm, and requires an additional track initiation/termination mechanism. This means L^{2}DA is not applicable for real world tennis sequences that have complex track initiation/termination scenarios of multiple balls. In this paper, we extend L^{2}DA to handle multiple objects and to automate the track initiation/termination. This is achieved by using an All-Pairs Shortest Path (APSP) formulation instead of the Single-Pair Shortest Path (SPSP) formulation at the second layer of L^{2}DA, and by adding a third layer, path level analysis, onto L^{2}DA. The resulting algorithm is dubbed L^{3}DA in this paper.

The rest of this paper is organised as follows: Section 2 gives a brief review of L^{2}DA. Section 3 describes the third layer, path level analysis, of L^{3}DA. This layer works on the set of all-pairs shortest paths in a graph. In Section 4, we propose an efficient APSP algorithm. Experimental results are presented in Section 5. Finally, conclusions are given in Section 6.

2 The L^{2}DA Algorithm

In L^{2}DA, the data association problem is sliced into two layers: candidate level association and tracklet level association. Assume the frames in a sequence are numbered from 1 to K. **At the candidate level**, a sliding window containing $2V+1$ frames is moving over the frames. At time i, the interval I_i centres on frame i and spans frame $i-V$ to frame $i+V$, where $i \in [1+V, K-V]$. Now instead of randomly sampling as in RDA, we exhaustively evaluate for each candidate in frame i whether a small ellipsoid around it in the column-row-time 3D space contains one candidate from frame $i-1$ and one candidate from frame $i+1$. If it does, we call the 3 candidates inside the ellipsoid a "seed triplet", and fit a constant acceleration dynamic model to it. The fitted model is then "improved" by re-fitting another model using candidates in the sliding window that are consistent with it. This process is repeated recursively until convergence, forming what we call a "tracklet": a small segment of a trajectory. Compared to RDA, this "hill-climbing" scheme significantly reduces the algorithm's complexity: as the proportion of true positives drops, the complexity grows approximately linearly.

A tracklet T consists of a parameterised dynamic model M (position and velocity at time i, and the constant acceleration), and a set $\mathscr{S}$ of candidates that support the converged model ("supports" of the model). In other words, $T \triangleq \{M, \mathscr{S}\}$. At time i, there may be multiple tracklets generated. We threshold them based on the number of candidates in their support sets, or their "strengths". Only tracklets that are "strong enough" are

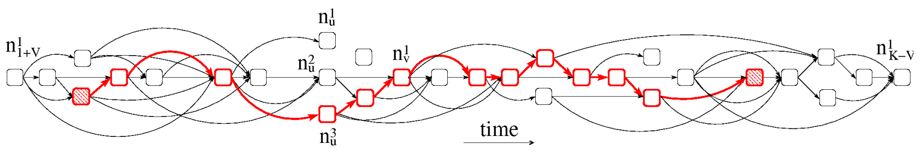


Figure 1: An illustrative example of the topology of $\mathscr{G}$. Each node is a tracklet. Nodes generated in the same sliding window position are aligned vertically. Striped red nodes: the first and last ball-originated nodes. Red nodes and red edges: the shortest path between these two nodes.

retained, and the jth retained tracklet in interval I_i is denoted by $T_i^j = \{M_i^j, \mathscr{S}_i^j\}$.

As the sliding window moves, a sequence of tracklets is generated. These tracklets may have originated from the ball or from clutter. Now we need a data association method **at the tracklet level**. We formulate tracklet level association as a SPSP problem. A directed and edge-weighted graph $\mathscr{G} = \{\mathscr{N}, \mathscr{E}\}$ is constructed, where each node $n_i^j \in \mathscr{N}$ represents the tracklet T_i^j, and the weight $w_{u,v}^{l,m}$ of a directed edge $e_{u,v}^{l,m} \in \mathscr{E}$, which connects n_u^l to n_v^m, is defined according to the "compatibility" of T_u^l and T_v^m, *i.e.* the smaller the $w_{u,v}^{l,m}$, the more likely T_u^l and T_v^m have originated from a same object (see [8] for details). We assume that there is only one ball in the sequence, and that the first and last tracklets (nodes) that have originated from this ball are already known. The ball-originated candidates are then contained in the support sets of the nodes in the shortest path between these two nodes, *i.e.* the path with smallest total edge weight (see Fig. 1).

3 Extending L^{2}DA to L^{3}DA

L^{2}DA assumes there is only one ball to track in a sequence. However, this is not always the case. For example, there may be multiple plays in one sequence, and the second play can start while the ball used for the first play is still in the scene. Moreover, track initiation/termination, which is taken for granted in L^{2}DA, is not a trivial problem, especially when multiple objects are present.

In this section, we extend L^{2}DA to L^{3}DA to deal with multiple objects and to automate the track initiation/termination. This is achieved by using APSP instead of SPSP at the tracklet level, and by introducing one more layer on top of that, namely, path level analysis with a Paths Reduction (PR) algorithm.

For a given pair of nodes n_u^l and n_v^m in $\mathscr{G}$, there may be paths connecting n_u^l to n_v^m, or there may not. Assume the shortest paths between all pairs of nodes that have at least one path connecting them have already been identified. Let $\mathscr{P}$ be the set of such all-pairs shortest paths, and p is the number of paths in $\mathscr{P}$. p is in the order of N^2, where N is the number of nodes in the graph. Now observe that no matter how many balls there are to track, or where each of the ball trajectories starts and terminates in the graph, the paths that correspond to the ball trajectories form a subset of $\mathscr{P}$. The question now is how to reduce the original set of APSP $\mathscr{P}$ to its subset that contains only paths that correspond to the ball trajectories.

We propose a simple Paths Reduction (PR) algorithm to achieve this. The PR algorithm reduces the set of APSP to the Best Set of Compatible Paths (BSCP) $\mathscr{B}$, providing two assumptions are satisfied: first, the p paths in $\mathscr{P}$ can be ordered according to their "qualities"; and second, a pair-wise "compatibility" of the paths in $\mathscr{P}$ is defined. The PR algorithm is summarised as follows:

- **Initialisation:** $\mathscr{P}$ has p paths, and $\mathscr{B}$ is empty.
- **While** $\mathscr{P}$ is not empty:
  - Remove the best path P^* in $\mathscr{P}$ from $\mathscr{P}$;
  - If P^* is compatible with all paths in $\mathscr{B}$, add P^* to $\mathscr{B}$.

Now we define the relative quality of the paths. Recall that the weight of a path is the sum of the weights of all edges the path goes through. Note that the term "shortest path" used in the previous sections should have been "lightest path". However, we chose to use "shortest path" for the sake of consistency with the terminology used in other papers. We define the strength of a path to be the number of supports in all its nodes, or more precisely, the size of the union of the support sets in all its nodes. Intuitively, a "good" path is one that is both "light" and "strong". However, there is usually a trade-off between the weight and the strength of a path: a stronger path tends to be heavier. Taking this into account, we define the relative quality of two path P_1 and P_2 as follows:

$$P_1 \left\{ \begin{array}{c} > \\ = \\ < \end{array} \right\} P_2 \quad \text{if} \quad (W_1 - W_2) \left\{ \begin{array}{c} < \\ = \\ > \end{array} \right\} \alpha \cdot (S_1 - S_2) \tag{1}$$

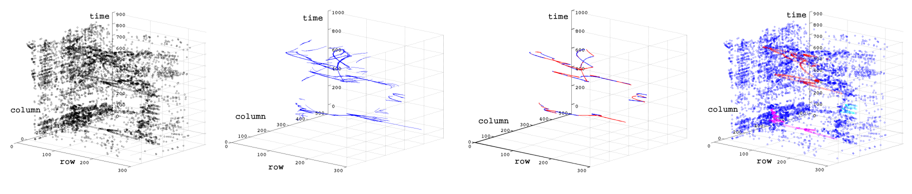
where the relation operators "$>$", "$=$" and "$<$" between P_1 and P_2 stand for "is better than", "has the same quality as", and "is worse than", respectively; W_1 and W_2 are the weights of P_1 and P_2, respectively; S_1 and S_2 are the strengths of P_1 and P_2, respectively; and α is a controllable parameter with the unit of pixel. According to this definition, if a path P_1 is "much stronger" but "slightly heavier" than a path P_2, then P_1 is said to have a better quality than P_2. Note that this definition does not assume any relationship between W_1 and W_2, or relationship between S_1 and S_2.

It easily follows that the set $\mathscr{P}$ equipped with an operator "$\geq$" satisfies the following three statements:

1. **Transitivity:** if $P_1 \geq P_2$ and $P_2 \geq P_3$ then $P_1 \geq P_3$;
2. **Antisymmetry:** if $P_1 \geq P_2$ and $P_2 \geq P_1$ then $P_1 = P_2$;
3. **Totality:** $P_1 \geq P_2$ or $P_2 \geq P_1$.

According to order theory [1], $\mathscr{P}$ associated with operator "$\geq$" is a totally ordered set. The first assumption for the PR algorithm to work is satisfied.

The second assumption, the existence of pair-wise compatibility of the paths, is straightforward. Two paths are said to be compatible if and only if they do not share any common **support**. It should be noted, however, two paths that do not share any common **node** are not necessarily compatible, because different nodes can have common supports.



(a) (b) (c) (d)

Figure 2: (a): ball candidates in an example sequence. Each black circle is a candidate. (b): generated tracklets. (c): results of applying APSP and the PR algorithm: 3 paths in $\mathscr{B}_{th}$. Adjacent nodes in each path are plotted alternatively in blue and red. (d): recovered class labels as given by $\mathscr{B}_{th}$.

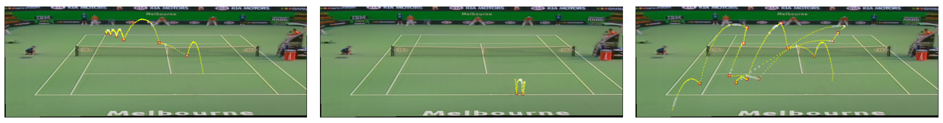


Figure 3: Ball trajectories (after interpolation and key event detection) superimposed on mosaic images. From left to right: the first, second and third play in time order.

Now with SPSP replaced by APSP at the tracklet level, and with the PR algorithm at the path level, we have extended L^{2}DA to L^{3}DA. We apply L^{3}DA to an example sequence (see Fig. 2). Semantically, the ball-originated candidates in this sequence belong to three plays. In time order (from bottom to top in the figures), the first play (magenta circles in Fig. 2 (d)) is a bad serve, where the ball lands outside the service box; the second "play" (cyan circles in Fig. 2 (d)) is a player bouncing the ball on the ground preparing for the next serve; and the third play (red circles in Fig. 2 (d)) is a relatively long one with several exchanges. The objective of data association is to identify the number of plays in this sequence, and to recover the class label of each candidate: clutter, first play, second play, or third play. In other words, the objective is to recover the colour information in Fig. 2 (d), assuming it is lost (see Fig. 2 (a)).

First, we "grow" tracklets from seed triplets (see Fig. 2 (b)), as in L^{2}DA. By looking for all-pairs shortest paths, a set $\mathscr{P}$ with $p = 87961$ paths is obtained. The PR algorithm is then applied, which gives a BSCP $\mathscr{B}$ containing 11 paths. In descending order, the numbers of supports (strengths) of the paths in $\mathscr{B}$ are: 411, 247, 62, 23, 20, 17, 17, 16, 15, 10, 9. It is a reasonable assumption that a path corresponding to a ball trajectory has more supports than a path corresponding to the motion of a non-ball object, *e.g.* a wristband worn by a player (which can be detected as ball candidates and can form smooth trajectory as the player strikes the ball). We set a threshold S_{th} and keep only the

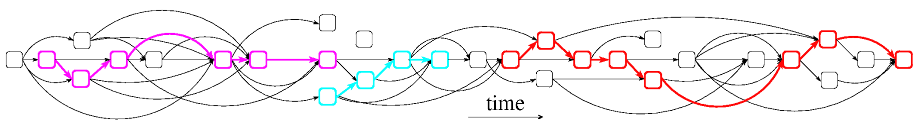


Figure 4: A possible arrangement of the paths in $\mathscr{B}_{th}$. Magenta, cyan, and red paths correspond to the first, second and third play in the sequence, respectively.

paths that have more supports than S_{th}. This results in a thresholded BSCP $\mathscr{B}_{th}$ with 3 paths (see Fig. 2 (c)), where each path corresponds to a play in the sequence.

In tennis ball tracking, the points at which the ball changes its motion abruptly correspond to key events such as hit and bounce, and provide important information for high level annotation. We detect these key events by looking for motion discontinuities in the trajectories. In Fig. 3, the 3 ball trajectories after interpolation and event detection are superimposed on mosaic images.

A suggestion of how the 3 paths might be arranged in the graph $\mathscr{G}$ is shown in Fig. 4. Note that there are 672 tracklets in this sequence. Far fewer nodes are plotted in Fig. 4 for ease of visualisation. Note also that two paths that are temporally overlapping are not necessarily incompatible. In fact, the first and second plays in the example sequence do overlap in time: the first play spans frame 16 to frame 260, and the second spans frame 254 to frame 321.

4 An Efficient APSP Algorithm

In L^{3}DA, at the tracklet level, we need to solve an APSP problem for a graph $\mathscr{G}$ with N nodes. In some sequences, N can be in the order of 10^3. An efficient APSP algorithm is desirable. Several APSP algorithms have been reported in the literature. The Floyd-Warshall algorithm solves APSP in $O(N^3)$ time [3]. Johnson's algorithm has a complexity of $O(N^2 \log N + NE)$, where E is the number of edges in the graph [4]. Neither the Floyd-Warshall algorithm nor Johnson's algorithm makes any assumption about the topology of the graph. Because of the way our graph is constructed, it has a special topological property: its set of nodes $\mathscr{N}$ can be partitioned into subsets $\mathscr{N}_{1+V}, \mathscr{N}_{2+V}, ..., \mathscr{N}_{K-V-1}, \mathscr{N}_{K-V}$, where $\mathscr{N}_i$ is the set of nodes generated in interval I_i, such that edges exist from nodes in subset $\mathscr{N}_u$ to nodes in subset $\mathscr{N}_v$ only if $u < v$ (see [8] for details). Using this property, we derive an $O(N^2)$ APSP algorithm as follows.

The proposed APSP algorithm uses the concept of dynamic programming. Suppose we are in the middle of the tracklet generation process. The sliding window now centres on frame $i-1$, and tracklets in interval I_{i-1} have been generated. Let $\mathscr{G}^{(i-1)} = \{\mathscr{N}^{(i-1)}, \mathscr{E}^{(i-1)}\}$ be the graph constructed so far, where $\mathscr{N}^{(i-1)} = \{\mathscr{N}_{1+V}, \mathscr{N}_{2+V}, ..., \mathscr{N}_{i-1}\}$; $\mathscr{E}^{(i-1)}$ is the set of edges that go into all nodes in $\mathscr{N}^{(i-1)}$. Clearly, $\mathscr{G}^{(i-1)}$ is a sub-graph of the complete graph $\mathscr{G}$. Assume the APSP problem in graph $\mathscr{G}^{(i-1)}$ has been solved. That is, in each node $n_v^m \in \mathscr{G}^{(i-1)}$, a table is maintained, where each entry corresponds to a node in the sub-graph $\mathscr{G}^{(v-1)}$. The entry corresponding to node $n_u^l \in \mathscr{G}^{(v-1)}$ keeps two pieces of information about the shortest path from n_u^l to n_v^m in $\mathscr{G}^{(i-1)}$. The first one is the last node before n_v^m in the shortest path, and the second one is the total weight of the shortest path. With these two pieces of information for each node $n_u^l \in \mathscr{G}^{(v-1)}$ in each node $n_v^m \in \mathscr{G}^{(i-1)}$, the shortest path between any pair of nodes in $\mathscr{G}^{(i-1)}$ can be identified by back tracing.

Next, we show how to solve the APSP problem in $\mathscr{G}^{(i)}$ using the solution of the APSP problem in $\mathscr{G}^{(i-1)}$. Now the sliding window moves one frame forward, and the interval I_i centres on frame i. Assume several tracklets are generated in I_i, forming the set of nodes $\mathscr{N}_i$. Now we need to construct for each node $n_i^j \in \mathscr{N}_i$ a table of APSP knowledge, where each entry contains information about the shortest path in $\mathscr{G}^{(i)}$ from a node in $\mathscr{G}^{(i-1)}$ to n_i^j.

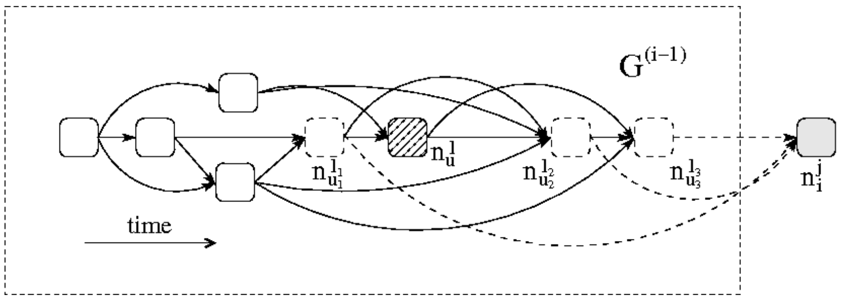


Figure 5: Constructing the table of APSP knowledge for a node $n_i^j \in \mathscr{N}_i$.

In Fig. 5, the sub-graph inside the big rectangle represents $\mathscr{G}^{(i-1)}$, and a new node $n_i^j \in \mathscr{N}_i$ is plotted as a shaded node. Assume s nodes in $\mathscr{G}^{(i-1)}$ are connected to n_i^j with edges. These s nodes are denoted by $n_{u_1}^{l_1}, n_{u_2}^{l_2}, ..., n_{u_s}^{l_s}$, and are plotted as dashed nodes in Fig. 5. Edges that connect these nodes to n_i^j are denoted by $e_{u_1,i}^{l_1,j}, e_{u_2,i}^{l_2,j}, ..., e_{u_s,i}^{l_s,j}$, and are plotted as dashed edges. Obviously, the number of entries in the table of APSP knowledge in n_i^j is equal to the number of nodes in $\mathscr{G}^{(i-1)}$. Without loss of generality, let us consider one entry in the table, which keeps information about the shortest path in $\mathscr{G}^{(i)}$ from a node $n_u^l \in \mathscr{G}^{(i-1)}$ to n_i^j. In Fig. 5, n_u^l is plotted as a striped node. Now observe that the shortest path in $\mathscr{G}^{(i)}$ from n_u^l to n_i^j must go through one of the nodes in $n_{u_1}^{l_1}, n_{u_2}^{l_2}, ..., n_{u_s}^{l_s}$ and the corresponding edge in $e_{u_1,i}^{l_1,j}, e_{u_2,i}^{l_2,j}, ..., e_{u_s,i}^{l_s,j}$. Since APSP has been solved in $\mathscr{G}^{(i-1)}$, the information about the shortest path in $\mathscr{G}^{(i-1)}$ from n_u^l to $n_{u_r}^{l_r}$ is kept in the table in $n_{u_r}^{l_r}$, where $r = 1, 2, ..., s$. Let $W^{(i-1)}(n_u^l, n_{u_r}^{l_r})$ be the total weight of the shortest path in $\mathscr{G}^{(i-1)}$ from n_u^l to $n_{u_r}^{l_r}$, as kept in the table in $n_{u_r}^{l_r}$. Specially, if the table in $n_{u_r}^{l_r}$ does not contain an entry for n_u^l, it means $u_r \leq u$, and we define for this case $W^{(i-1)}(n_u^l, n_{u_r}^{l_r}) = \infty$. The total weight of the shortest path in $\mathscr{G}^{(i)}$ from n_u^l to n_i^j is then:

$$W^{(i)}(n_u^l, n_i^j) = \min[W^{(i-1)}(n_u^l, n_{u_r}^{l_r}) + w_{u_r,i}^{l_r,j}], \forall r \in [1, s] \tag{2}$$

where $w_{u_r,i}^{l_r,j}$ is the weight of edge $e_{u_r,i}^{l_r,j}$. The last node before n_i^j in the shortest path in $\mathscr{G}^{(i)}$ from n_u^l to n_i^j is $n_{u^*}^{l^*}$, where

$$\{u^*, l^*\} = \arg \min_{\{u_r, l_r\}} [W^{(i-1)}(n_u^l, n_{u_r}^{l_r}) + w_{u_r,i}^{l_r,j}], \forall r \in [1, s] \tag{3}$$

The two pieces of information for one entry in the table in node n_i^j are thus obtained: $W^{(i)}(n_u^l, n_i^j)$ and $n_{u^*}^{l^*}$ are put into the entry for n_u^l. This process is applied to each node in $\mathscr{G}^{(i-1)}$, whereupon the complete table in n_i^j is constructed. Using the special topological property of the graph $\mathscr{G}$ discussed at the beginning of this section, the shortest path in $\mathscr{G}^{(i-1)}$ between any pair of nodes in $\mathscr{G}^{(i-1)}$ is also the shortest path in $\mathscr{G}^{(i)}$ between the same pair of nodes. When the new node n_i^j and the associated edges $e_{u_1,i}^{l_1,j}, e_{u_2,i}^{l_2,j}, \cdots, e_{u_s,i}^{l_s,j}$ are added to $\mathscr{G}^{(i-1)}$, the tables in the nodes in $\mathscr{G}^{(i-1)}$ remain the same. This means that, simply by applying the above process as new nodes (and associated edges) are received, when the complete graph $\mathscr{G} = \mathscr{G}^{(K-V)}$ is constructed, the APSP problem in it is solved.

SVM boundary	-4	-3	-2	-1	0	1
r_d	0.917	0.916	0.908	0.874	0.822	0.531
$\bar{N}$	12.2	9.0	5.1	0.9	0.1	0

Table 1: Detection rate and clutter level with various SVM boundaries.

The shortest path between any pair of nodes in $\mathscr{G}$ can be easily identified by back tracing. The proposed APSP algorithm is summarised as follows:

```
• Assume: the APSP problem in $\mathscr{G}^{(i-1)}$ has been solved.
• For each node $n_i^j \in \mathscr{N}_i$:
    – For each node $n_u^l \in \mathscr{G}^{(i-1)}$:
        * add an entry labelled $n_u^l$ to the table of APSP knowledge in $n_i^j$;
        * put $W^{(i)}(n_u^l, n_i^j)$ and $n_{u^*}^{l^*}$ given by (2) and (3) into this entry.
```

Let h_i be the number of nodes in $\mathscr{N}_i$. The number of nodes in sub-graph $\mathscr{G}^{(i-1)}$ is then $\sum_{k=1+V}^{i-1} h_k$. To solve the APSP problem in $\mathscr{G}^{(i)}$, we need to construct a table of APSP knowledge for each node in $\mathscr{N}_i$. The number of operations of this process is in the order of $h_i \sum_{k=1+V}^{i-1} h_k$. The number of operations of the proposed APSP algorithm is then in the order of $\sum_{i=2+V}^{K-V} (h_i \sum_{k=1+V}^{i-1} h_k)$. Simple manipulation shows that the complexity of the proposed APSP algorithm is $O(N^2)$, where $N = \sum_{i=1+V}^{K-V} h_i$ is the number of nodes in $\mathscr{G}$.

5 Experiments

We used 60 sequences from the 2006 Australia Open tournament Men's final game for our experiments. The number of plays in each sequence ranges from 2 to 4. In total the 60 sequences are approximately 16 minutes long, and contain 50,662 frames.

We used frame differencing to extract foreground moving objects. A Support Vector Machine (SVM) was trained and used to classify the foreground blobs into ball candidates and non-candidates. Features used in the SVM are the shape, colour and position of each blob. By moving the decision boundary of the SVM, a trade-off can be made between the ball detection rate r_d and the average number of false candidates $\bar{N}$ in each frame. Table 1 shows 6 SVM boundaries and the corresponding r_d and $\bar{N}$. Using these 6 configurations, we can evaluate a tracker's performance under various detection rate and clutter level.

RDA and another two tennis ball tracking algorithms from our previous work [6, 7], one based on particle filtering, and the other based on the Viterbi algorithm, were also implemented for comparison. For these three trackers, one instance of the tracker was used to track each play in each sequence, and track initiation/termination of each play was manually dealt with. In RDA, the number of trials, N_t, is chosen so that the probability of finding a set that consists entirely of true positives is greater than a threshold γ. In our experiments, γ was set to 0.99.

SVM boundary		-4	-3	-2	-1	0	1
	particle	8.64%	9.61%	6.92%	6.63%	8.29%	19.16%
prop. of	Viterbi	**4.14%**	3.88%	**3.41%**	3.23%	3.64%	4.12%
LOT	RDA	17.06%	15.73%	12.43%	9.18%	6.99%	7.27%
Frames	L^{3}DA	4.40%	**3.68%**	3.57%	**2.81%**	**2.41%**	**2.73%**

Table 2: Proportion of loss-of-track (LOT) frames.

SVM boundary		-4	-3	-2	-1	0	1
	particle	21.0	23.2	25.1	26.6	28.8	30.7
processing speed	Viterbi	31.3	36.7	40.4	42.2	45.9	47.3
(frames per sec)	RDA	0.9	1.7	23.5	**233.0**	**374.8**	**399.1**
	L^{3}DA	**46.3**	**59.4**	**72.8**	93.6	116.2	142.4

Table 3: Processing speed.

To evaluate the performance of the trackers, ground truth of the tennis ball positions in all frames was manually marked. Tracking results were then compared against the ground truth. Tracking error is defined as the Euclidean distance between the ground truth and the tracked (detected or interpolated) ball position. A loss-of-track (LOT) frame is defined as a frame where the tracking error is greater than 6 pixels. Table. 2 shows the proportion of LOT frames of each tracker with each SVM boundary. In brief, L^{3}DA and the Viterbi-based tracker outperform the other two trackers. When looking more carefully at Table. 2, we can see the four algorithms have different failure modes.

When r_d and $\bar{N}$ are both low, the particle-base algorithm performs poorly. This is because the ball changes its motion drastically after being hit by a player. Consequently, the next detected ball-originated candidate can be very far from its predicted position. This is especially the case when r_d is low. As a result, the particle-based tracker can be "trapped" by false candidates that have originated from the player, and cannot recover until the ball is close to the player again. On the other hand, L^{3}DA, being a non-iterative algorithm, is much more robust against sudden change of motion direction.

RDA performs poorly when r_d and $\bar{N}$ are high. This is because in RDA, or more generally in RANSAC, we make the implicit assumption that a model given by an uncontaminated sample set is always "better" than that given by a contaminated sample set. However, in a tennis sequence, especially when multiple balls are present, the ball being tracked is not the only smoothly moving object. Candidates that have originated from other balls, or even from part of a player, *e.g.* a wrist band, can form smooth trajectories. As a result, a model given by candidates that have originated from the ball being tracked can "lose" in the competition with a model given by candidates that have originated from other objects. This problem is tackled in L^{3}DA by enforcing motion consistency with the shortest path formulation.

The Viterbi-based algorithm gives similar performance to that of L^{3}DA. However, L^{3}DA has the advantage of being fully-automatic, while the Viterbi-based algorithm requires an additional track initiation/termination mechanism.

In Table 3, the speed of the four algorithms is compared. L^{3}DA shares the top position with the LDA. The fact that L^{3}DA always starts model fitting from a seed triplet — three candidates that have high probability of containing only true positives — allows it to eliminate false candidates very quickly. The proposed APSP algorithm also helps improve the efficiency of L^{3}DA. It should be noted that as the SVM boundary increases, RDA has the fastest growing processing speed. This is because the time complexity of RDA is determined directly by N_t, which drops rapidly as the proportion of true positives increases.

6 Conclusions

In this paper, we have extended our previous work L^{2}DA, a semi-automatic single-object tracking algorithm, to L^{3}DA, a fully automatic multiple-object tracking algorithm. This was achieve by using APSP instead of SPSP at the tracklet level, and by adding one more layer, path level analysis, on top of L^{2}DA. In this paper, we have also proposed an efficient APSP algorithm by exploiting a special topological property of the graph. The proposed L^{3}DA algorithm was used to track tennis balls in broadcast tennis video. Comparative experiments show that it performs well both in terms of efficiency and tracking accuracy.

Acknowledgements

This work has been supported by the EU IST-507752 MUSCLE Network of Excellence.

References

[1] B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, 2002.

[2] M. A. Fischler and R. C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the Association for Computing Machinery*, 24(6):381–395, 1981.

[3] R. W. Floyd. Algorithm 97: Shortest path. *Communications of the ACM*, 5(6):345, 1962.

[4] D. B. Johnson. Efficient algorithms for shortest paths in sparse networks. *Journal of the ACM*, 24(1):1–13, 1977.

[5] V. Lepetit, A. Shahrokni, and P. Fua. Robust data association for online applications. In *IEEE International Conference on Computer Vision and Pattern Recognition*, volume 1, pages 281–288, 2003.

[6] F. Yan, W. Christmas, and J. Kittler. A tennis ball tracking algorithm for automatic annotation of tennis match. In *British Machine Vision Conference*, volume 2, pages 619–628, 2005.

[7] F. Yan, W. Christmas, and J. Kittler. A maximum a posteriori probability viterbi data association algorithm for ball tracking in sports video. In *IEEE International Conference on Pattern Recognition*, 2006.

[8] F. Yan, A. Kostin, W. Christmas, and J. Kittler. A novel data association algorithm for object tracking in clutter with application to tennis video analysis. In *IEEE International Conference on Computer Vision and Pattern Recognition*, 2006.