Debugging Object Tracking Results by a Recommender System with Correction Propagation

Mingzhong Li and Zhaozheng Yin

Department of Computer Science
Missouri University of Science and Technology, USA

Abstract. Achieving error-free object tracking is almost impossible for state-of-the-art tracking algorithms in challenging scenarios such as tracking a large amount of cells over months in microscopy image sequences. Meanwhile, manually debugging (verifying and correcting) tracking results object-by-object and frame-by-frame in thousands of frames is too tedious. In this paper, we propose a novel scheme to debug automated object tracking results with humans in the loop. Tracking data that are highly erroneous are recommended to annotators based on their debugging histories. Since an error found by an annotator may have many analogous errors in the tracking data and the error can also affect its nearby data, we propose a correction propagation scheme to propagate corrections from all human annotators to unchecked data, which efficiently reduces human efforts and accelerates the convergence to high tracking accuracy. Our proposed approach is evaluated on three challenging datasets. The quantitative evaluation and comparison validate that the recommender system with correction propagation is effective and efficient to help humans debug tracking results.

1 Introduction

Automated visual object tracking is very useful to monitor objects over a long period and analyze their behavior. Multi-Hypothesis Tracking (MHT) [10] and Joint Probabilistic Data Association Filters (JPDAF) [4] are two representative examples for multi-object tracking. To reduce the computational cost, tracklet stitching [5] is proposed: first reliable tracklets are generated which are fragments of tracks formed by conservative grouping of detection responses, then the tracklets are connected by the Hungarian algorithm [6]. Bonneau et al. [1] proposes a tracklet linking method in which a minimal path among tracklets is obtained by using dynamic programming in order to track quantum dots in a living cell. Zhang et al. [13] proposes a minimum-cost flow network to resolve the global data association of multiple objects over time.

In real world applications such as uncovering hidden patterns of a complex biological process, high quality object tracking algorithms are required to accurately track bio-specimens over a long period. But, due to the numerous challenges in biomedical data such as appearance similarity, heavy occlusion and clutter, it is extremely difficult to achieve perfect tracking performance without any error. To pursue solid scientific discovery and error-free health diagnosis, biologists and doctors are willing to exchange a small amount of their human efforts to check the automated tracking results manually. Hence, it is worth to consider how to incorporate human efforts to debug (verify and correct) the tracking results, which leads to the following three questions:

(1) Manually checking each object's trajectory frame by frame is very costly for human labors, which we cannot afford. How to find out which tracking data are error-prone thus they are worth to be checked by human?

(2) Checking tracking data on specimens captured over months with thousands of frames is too tedious for an individual. How can we integrate crowdsourcing to check the data collectively?

(3) There might be analogy between different error nodes in the tracking data, and the error can also affect its nearby data. How can we propagate the costly human correction to other unchecked data and automatically correct similar errors such that human burden is alleviated and the convergence to the best tracking accuracy is accelerated?

Recommender systems [2, 8, 9, 11, 14] are capable of using historical data of a user to infer her/his preference on items and then predicting other items that the user might like. Websites such as Google.com, Amazon.com, Ebay.com, etc. have widely equipped their search engines with specialized recommender systems to help their customers find their preferred commodities. Particularly, content-based recommender systems analyze a set of documents and/or descriptions of items previously rated by a user, and build a model to predict the user's interest based on the features of the object ratings [8, 9]. How to construct a proper user profile by collecting data representing the user's preference, is the key of content-based recommender systems.

In this paper, we assume no object tracking algorithm can achieve perfect tracking performance in challenging scenarios. Instead of aiming at developing object detection and tracking algorithms, our focus is to investigate how to debug existing object tracking results with humans in the loop. The main contributions of this paper include:

(1) we propose a novel recommender system to assist multiple human annotators to debug tracking data collectively. Tracking data with high error likelihood are recommended to each individual annotator based on their debugging histories. The verification and correction made by annotators are collected for subsequent correction propagation and user profile updating procedures;

(2) we propose a correction propagation scheme, which propagates the corrected information to other track data affected by the corrected data, based on the verification and corrections made by multi-annotators.

The paper is organized as belows. In Section 2, we describe a basic data-association method for multi-object tracking. In Section 3, we present the recommender system to debug tracking data with multi-annotators in the loop. In Section 4, we introduce how to propagate human corrections to other unchecked tracking data to accelerate the debugging process. Experimental results are presented in Section 5.

2 Multi-Object Tracking

We formulate the multi-object tracking problem in the framework of "tracking-by-detection". First, detected objects in individual frames are considered as *nodes* and they are connected frame-by-frame into short reliable trajectories (a.k.a, *tracklets*). Second, these short tracklets are linked into longer and longer tracklets gradually by a sequential procedure (*fine-to-coarse tracklet association*). Finally, detection-related and tracklet-related features are generated for every node of every tracklet, which are used in the recommender system and correction propagation.

2.1 Tracklet Generation

Every detected object candidate in a frame is represented as a *node* with corresponding features such as color distribution, gradient histogram, object shape, location, etc. We denote $\mathbf{f}(n_i^t) = [\mathbf{f}_1(n_i^t), ..., \mathbf{f}_K(n_i^t)]$ as the vector of K features of node i in frame t. The dissimilarity cost between a pair of nodes in two consecutive frames is computed as

$$c(n_i^t, n_j^{t-1}) = \begin{cases} \frac{1}{K}\sum_{k=1}^{K} \frac{\left\|\mathbf{f}_k(n_i^t)-\mathbf{f}_k(n_j^{t-1})\right\|}{\Delta_k}, \text{if } \left\|\mathbf{f}_k(n_i^t) - \mathbf{f}_k(n_j^{t-1})\right\| \leq \Delta_k, \forall k \in [1, K] \\ \infty, \text{otherwise} \end{cases} \tag{1}$$

where $\|\cdot\|$ is the L_2 norm and Δ_k is the normalization factor of the kth feature. For example, when $\mathbf{f}_k$ is the location feature, Δ_k controls the spatial gating region (i.e., the size of local neighborhood to search a node's correspondence between consecutive frames).

Given I nodes in frame t and J nodes in frame $t-1$, a cost matrix $\mathbf{C} = [c(n_i^t, n_j^{t-1})]$ with size I-by-J is generated and we apply the Hungarian algorithm [6] onto it to solve the linear assignment problem (i.e., corresponding nodes between frames t and $t-1$ are connected). After sequentially performing the Hungarian algorithm between consecutive frames, *tracklets* are generated for a given video (e.g., Fig.1(a)). Note that, (1) we use small gating regions in the frame-by-frame assignment, which generates tracklets with less errors but also causes short broken tracklets when objects move beyond the gating regions; (2) the Hungarian algorithm solves the 1-to-1 bipartite assignment problem but it can not solve the 2-to-1 or 1-to-2 assignment problem when there exists object merging or division, which causes broken or wrong connections among tracklets; (3) it is usually difficult to have perfect detection results for every frame, hence false positives and miss detections will cause broken or wrong connections among the tracklets. In the following two subsections, we describe how to gradually link the short tracklets into longer object trajectories.

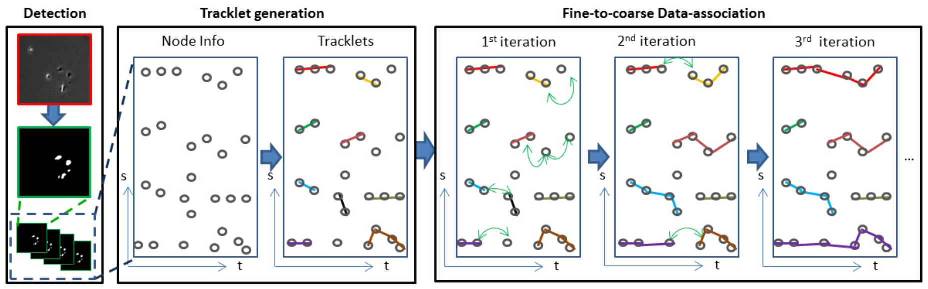

Fig. 1. Multi-object tracking.

2.2 Fine-to-Coarse Tracklet Association

We denote the ith tracklet $\mathbf{T}_i$ by its node set, $\mathbf{T}_i = \{n_i^{s_i}, n_i^{s_i+1}, ..., n_i^{e_i}\}$, where $n_i^{s_i}$ and $n_i^{e_i}$ represent the nodes in the start and end frame s_i and e_i, respectively. Five types of hypotheses are considered when associating tracklets:

(1) Translation (1-to-1): the head of tracklet $\mathbf{T}_j$ is associated with the tail of tracklet $\mathbf{T}_i$ with the cost:

$$c(\mathbf{T}_i \rightarrow \mathbf{T}_j) = \begin{cases} \frac{1}{K+1}\sum_{k=1}^{K+1} \frac{\left\|\mathbf{f}_k^+(n_i^{e_i}) - \mathbf{f}_k^+(n_j^{s_j})\right\|}{\Delta_k}, \\ \quad \text{if} \left\|\mathbf{f}_k^+(n_i^{e_i}) - \mathbf{f}_k^+(n_j^{s_j})\right\| \leq \Delta_k, \forall k \in [1, K+1], \\ \infty, \text{otherwise} \end{cases} \tag{2}$$

where $\mathbf{f}^+(n_i^{e_i}) = [\mathbf{f}(n_i^{e_i}), \theta(n_i^{e_i})]$ and $\mathbf{f}^+(n_j^{s_j}) = [\mathbf{f}(n_j^{s_j}), \theta(n_j^{s_j})]$ are the augmented feature vectors for the end and start nodes of tracklets $\mathbf{T}_i$ and $\mathbf{T}_j$, respectively. $\theta(\cdot)$ denotes a node's trajectory orientation in the tracklet.

(2) Division (1-to-2): the tail of a tracklet is associated with the heads of two tracklets with the cost:

$$c(\mathbf{T}_i \rightarrow (\mathbf{T}_{j_1}, \mathbf{T}_{j_2})) = c(\mathbf{T}_i \rightarrow \mathbf{T}_{j_1}) + c(\mathbf{T}_i \rightarrow \mathbf{T}_{j_2}) + c(n_{j_1}^{s_{j_1}}, n_{j_2}^{s_{j_2}}) \tag{3}$$

(3) Merging (2-to-1): the tails of two tracklets are associated with the head of a tracklet with the cost:

$$c((\mathbf{T}_{i_1}, \mathbf{T}_{i_2}) \rightarrow \mathbf{T}_j) = c(\mathbf{T}_{i_1} \rightarrow \mathbf{T}_j) + c(\mathbf{T}_{i_2} \rightarrow \mathbf{T}_j) + c(n_{i_1}^{e_{i_1}}, n_{i_2}^{e_{i_2}}) \tag{4}$$

(4) Disappearing (1-to-0): the tail of a tracklet is not linked to any other tracklet with the cost:

$$c(\mathbf{T}_i \rightarrow \phi) = \begin{cases} \frac{d^{(t)}(n_i^{e_i}, e)}{\Delta t}, & \text{if } d^{(t)}(n_i^{e_i}, e) \leq \Delta t, \\ \frac{d^{(s)}(n_i^{e_i})}{\Delta s}, & \text{if } d^{(s)}(n_i^{e_i}) \leq \Delta s, \\ \eta, & \text{otherwise.} \end{cases} \tag{5}$$

where $d^{(t)}(n_i^{e_i}, e)$ denotes the temporal distance from the ending node of $\mathbf{T}_i$ to the last frame. $d^{(s)}(n_i^{e_i})$ denotes the spatial distance from the ending node of $\mathbf{T}_i$ to the image boundary. During object tracking, three scenarios cause the disappearing cases: (i) objects at the end of a video will disappear; (ii) objects close to the image boundary may move out of the view field; and (iii) every object may be missed by the detection or occluded by other objects/background so it is associated with a constant cost η (we choose η as the maximum of all non-infinity $c(\cdot \to \cdot)$).

(5)Appearing (0-to-1): similar to 1-to-0 case we define the cost for 0-to-1 hypothesis as:

$$c(\phi \to \mathbf{T}_i) = \begin{cases} \frac{d^{(t)}(n_i^{s_i}, s)}{\Delta t}, & \text{if } d^{(t)}(n_i^{s_i}, s) \leq \Delta t, \\ \frac{d^{(s)}(n_i^{s_i})}{\Delta s}, & \text{if } d^{(s)}(n_i^{s_i}) \leq \Delta s, \\ \eta, & \text{otherwise.} \end{cases} \tag{6}$$

Denoting the number of tracklets in a video as N and the number of all possible hypotheses among the N tracklets as M, we catenate the costs of all hypotheses into a M-by-1 vector $\mathbf{c}$ and define a constraint matrix $\mathbf{Q}$ of size M-by-$2N$. For example, if the hth hypothesis is $\mathbf{T}_i \to (\mathbf{T}_{j_1}, \mathbf{T}_{j_2})$ involving tracklets i, j_1 and j_2, then $\mathbf{Q}(h, i) = 1$, $\mathbf{Q}(h, N + j_1) = 1$, $\mathbf{Q}(h, N + j_2) = 1$ and all other elements of the hth row of $\mathbf{Q}$ are zero. The tracklet association is obtained by solving the following Linear Integer Programing (LIP) problem:

$$\arg\min_{\mathbf{x}} \mathbf{c}^T\mathbf{x}, \quad s.t.\ \mathbf{Q}^T\mathbf{x} = \mathbf{1} \tag{7}$$

where $\mathbf{x}$ is an M-by-1 binary vector and $\mathbf{x}_h = 1$ indicates that the hth hypothesis is selected to be true in the optimal solution. The objective function ($\mathbf{c}^T\mathbf{x}$) aims to find an optimal $\mathbf{x}$ to minimize the total cost of selected hypotheses. The constraint ($\mathbf{Q}^T\mathbf{x} = \mathbf{1}$) ensures that one tracklet is only associated at most once on its head and tail.

Rather than solving the global tracklet association only once with a large gating region which may introduce a significant amount of errors during association, we gradually increase the gating regions (Δ_k) and iteratively solve the corresponding LIP problem, thus the short tracklets are linked into longer and longer ones in a fine-to-coarse manner with less errors (e.g., Fig.1(b)).

2.3 Features for Nodes in Tracklets

In Table 1, we list the features used for nodes in tracklets. Given node n_k^t of tracklet $\mathbf{T}_k$ in frame t, it has features related to both object detection ($\mathbf{f}(n_k^t)$) and tracklet association.

$c_s(n_k^t)$	the cost of the hypothesis involving n_k^t
$c_g(n_k^t)$	number of times the gating region has been increased
$l(n_k^t)$	length of the shortest tracklet among $\{\mathbf{T}_i : \delta(\mathbf{T}_k, \mathbf{T}_i) \neq 0, i \neq k\}$
$c_t(n_k^t)$	$\lvert\{\mathbf{T_j} : \delta(\mathbf{T_k}, \mathbf{T_j}) \neq 0, j \neq k\}\rvert$
$c_h(n_k^t)$	$\lvert\{\mathbf{T_i} : \delta(\mathbf{T_i}, \mathbf{T_k}) \neq 0, i \neq k\}\rvert$

Table 1. Features for nodes in tracklets.

where $|\cdot|$ denotes the cardinality of a set and $\delta(\mathbf{T_k}, \mathbf{T_j})$ is an indicator function ($\delta(\mathbf{T_k}, \mathbf{T_j}) = 1$ when $\mathbf{T_j}$'s head is within the gating region of the tail of $\mathbf{T_k}$; $\delta(\mathbf{T_k}, \mathbf{T_j}) = 0$, otherwise).

For example, in Fig.2(a), if node a is linked to node x by data association, $c_s(a)$ and $c_s(a)$ will be the cost of associate the tracklet of a to the tracklet of x by Eq.2. If the gating region has been increased twice before a and x are linked, $c_g(a)$ and $c_g(x)$ will be 2. $l(a)$ is the length of the shortest tracklet within a's gating region, which is the length of the tracklet with starting node y, hence $l(a) = 2$. Similarly, $l(b) = 3$ and $l(c) = 3$. In the relation graph (Fig.2(b)) of Fig.2(a), $c_t(\cdot)$ and $c_h(\cdot)$ compute the degrees of corresponding nodes, thus $c_t(a) = 2$, $c_t(b) = 2$ and $c_t(c) = 1$; $c_h(x) = 2$, $c_h(y) = 1$ and $c_h(z) = 2$.

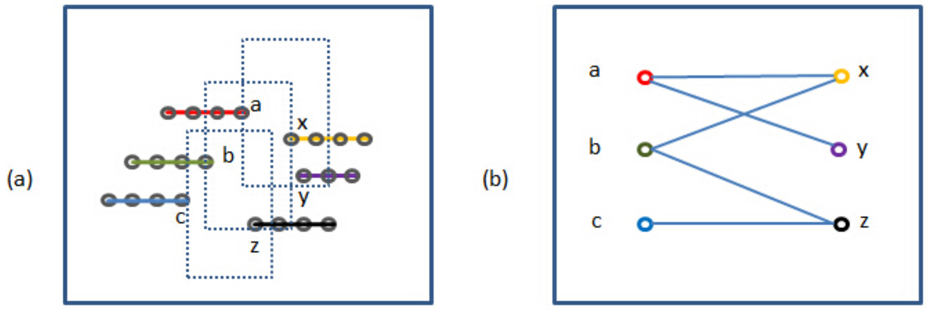

Fig. 2. (a) Tracklets and gating regions (blue dotted windows); (b) Relation graph.

$c_s(n_k^t)$ stores the latest association cost involving the node n_k^t. The higher $c_s(n_k^t)$ is, the more unreliable the association happened on node n_k^t is. The motivation of considering $c_g(n_k^t)$ as one of the features of node n_k^t is that we want to evaluate at which stage node n_k^t is associated to the longest possible trajectory at last. If the stage is high, which means the gating region has been increased many times, the association on node n_k^t is more likely to be a mistake.

Very short tracklets near node n_k^t are highly possible to be false positives and associating them with node n_k^t causes errors, thus we consider $l(n_k^t)$ as one feature. Similar reasons lead us to consider $c_t(n_k^t)$ and $c_h(n_k^t)$: when there are more association possibilities around node n_k^t, the association on node n_k^t may be more erroneous.

All these features are combined with the object-detection-related features into a feature vector $\mathbf{F}(n_k^t) = [\mathbf{f}(n_k^t), c_s(n_k^t), c_g(n_k^t), l(n_k^t), c_t(n_k^t), c_h(n_k^t)]$ to describe node n_k^t. The association-related parts $(c_s(n_k^t), c_g(n_k^t), l(n_k^t), c_t(n_k^t), c_h(n_k^t))$ are updated only when an association hypothesis involving n_k^t is within the optimal solution of the LIP problem in Eq.7. Details of updating the association-related node features are summarized in Algorithm 1 below.

Algorithm 1: Node Feature Updating in Fine-to-Coarse Association

```
Input : $Tracklets : \{\mathbf{T}_i\}$; gating region increasing rate: $\alpha$;
Initialization : $\forall n_k^t$, $c_s(n_k^t) \leftarrow 0$,$c_g(n_k^t) \leftarrow 0$, $l(n_k^t) \leftarrow 0$,$c_t(n_k^t) \leftarrow 0$,$c_h(n_k^t) \leftarrow 0$,
$\beta \leftarrow [\Delta_1, ..., \Delta_{K+1}]$;
Repeat
    Solve the LIP problem in Eq.7;
    for any selected association hypothesis linking tracklets $\mathbf{T_p}$ with $\mathbf{T_q}$
```

```
        $c_s(n_q^{s_q}) \leftarrow c(\mathbf{T}_p, \mathbf{T}_q), c_s(n_p^{e_p}) \leftarrow c(\mathbf{T}_p, \mathbf{T}_q)$;//the current hypothesis' cost
        $c_g(n_p^{e_p}) + +, c_g(n_q^{s_q}) + +$;//the times of gating regions being increased
        compute $l(n_p^{e_p})$, $l(n_q^{s_q})$;//the length of the shortest tracklet nearby
        $c_t(n_p^{e_p}) \leftarrow |\{\mathbf{T_j} : \delta(\mathbf{T_p}, \mathbf{T_j}) \neq 0, j \neq p\}|$;
        $c_h(n_q^{s_q}) \leftarrow |\{\mathbf{T_i} : \delta(\mathbf{T_i}, \mathbf{T_q}) \neq 0, i \neq q\}|$;
    end for
    $\beta \leftarrow \beta + \alpha \cdot \beta$//increase the gating regions
    update $\{\mathbf{T}_i\}$ with the optimization result;
Until no change happens to the association.
```

3 Recommender System

The key idea of content-based recommender system is to estimates the profile parameters of users, $\{\theta^{(u)}, u = 1, ..., U\}$, using the available feature vectors of targets $\{\mathbf{x}^{(i)}, i = 1, ..., n_x\}$

$$\theta^{(u)} : \arg\min_{\theta^{(u)}} \sum_{i:r(i,u)=1} \left(\theta^{(u)^T}\mathbf{x}^{(i)} - \mathbf{y}^{(i,u)}\right)^2 + \lambda\|\theta^{(u)}\| \tag{8}$$

where U and n_x denote the number of users and targets, respectively. $r(i, u) = 1$ if user u has recommendation $(\mathbf{y}^{(i,u)})$ on target i. λ is the coefficient for the regularization term. For any user u, we learn a parameter vector $\theta^{(u)}$ representing the user's preference. Given any new target j with feature $\mathbf{x}^{(j)}$, we predict user u's recommendation on target j as $\theta^{(u)^T}\mathbf{x}^{(j)}$.

In our system the users are annotators who can verify tracking results and correct corresponding errors, and the targets are a large pool of nodes from all linked tracklets with features. Fig.3 shows the workflow of our recommender system and correction propagation:

First, each of the U users selects a small portion of nodes from the large node pool independently and classifies them into *positive* nodes (nodes with tracking errors) and *negative* nodes (nodes without any tracking error). All other nodes unselected by any user are transferred to the uncertain node pool. Note that this manual initialization step only needs to be done once.

Second, each user's profile parameters are learned from the positive and negative node sets.

Third, the recommendation on every node in the uncertain node pool by every user is computed by the user's profile and node's feature vector.

Fourth, top-ranked recommendations are sent to users for verification and correction.

Fifth, the corrections made by human are automatically propagated to other uncertain nodes and their feature vectors are updated accordingly.

Finally, the nodes in uncertain node pool after correction propagation are either relocated to positive/negative sets of users for updating users' profiles or still remain in the uncertain pool if not affected by the correction propagation. The process is iterated until the uncertain node pool is empty.

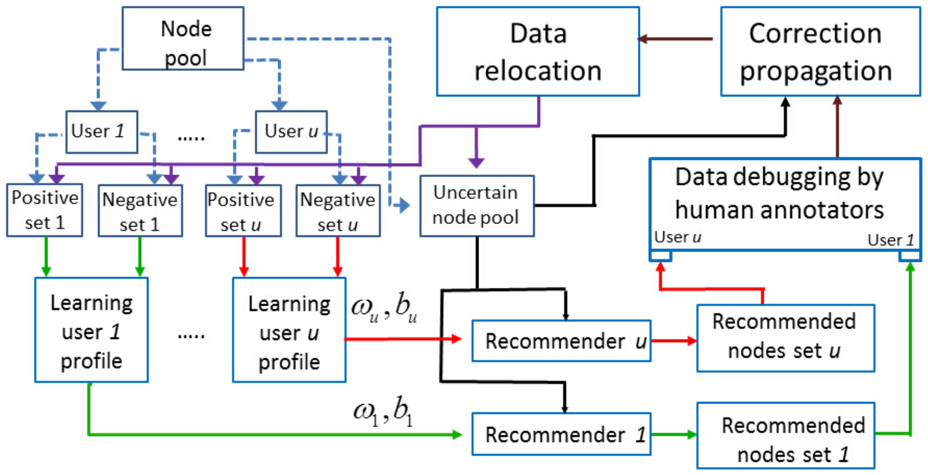

Fig. 3. Workflow of our recommender system.

Different from the least square cost in Eq.3 for recommender learning, we use a linear SVM to learn the profile parameters of any user u

$$\arg\min_{\mathbf{w}_u, b_u, \lambda} \left\{ \sum_i \lambda_i [y^{(i,u)}(\mathbf{w}_u{}^T \psi(\mathbf{F}^{(i,u)}) + b_u) - 1] + \frac{1}{2} \|\mathbf{w}_u\| \right\} \tag{9}$$

where $\mathbf{F}^{(i,u)}$ is the feature vector of node i in user u's positve/negative training node sets. $\mathbf{w}_u$ and b_u define the maximum-margin hyperplane that classifies $\mathbf{F}^{(i,u)}$ according to its class labels $\mathbf{y}^{(i,u)}$. $\psi(\cdot)$ is the kernel function (linear in this paper).

After learning the profile of user u, recommendation on any node j in the uncertain node pool by user u (i.e., verify the node or not) can be computed by the linear SVM score:

$$\mathbf{w}_u{}^T \psi(\mathbf{F}^{(j,u)}) + b_u \tag{10}$$

A small set of nodes from the uncertain node pool which have suspiciously high scores (i.e., high probability of tracking errors) are recommended to user u for verification and correction. For example, top 20 nodes are recommended to a user in each iteration. The profiles of different users are learnt independently hence different sets of nodes are recommended for different users to verify and correct, but their corrections can be collected together for efficient correction propagation discussed below.

4 Correction Propagation

In tracking data, neighboring nodes affect each other. In order to accelerate the debugging efficiency, we propose a correction propagation approach to spread out the correction information of corrected nodes to their neighboring nodes in the uncertain node pool.

First of all, a graph-based Propagation Set Detection (PSD) algorithm is proposed. In the graph G, each vertex is a node in the node pool and edge exists between two nodes if and only if there is an association hypothesis involving both of the nodes in the data-association step (e.g., the relation graph in Fig.2(b)).

Given users' corrections involving nodes $\mathbf{R}$, we detect the propagation set by the algorithm below:

Algorithm 2: Propagation Set Detection

```
Input : Graph $\mathbf{G}$, correction set $\mathbf{R}$
Output : node set $\mathbf{V}_{PSD}$;
Initialization : queue $\mathbf{Q} \leftarrow \mathbf{R}$; node set $\mathbf{V}_{PSD} \leftarrow \mathbf{R}$;
Repeat
    $t \leftarrow \mathbf{Q}.dequeue$;//get the first element of the queue
    for all edges of node t in $\mathbf{G}$, $e$;
        $v \leftarrow \mathbf{G}.adjacentVertex(t, e)$;
        if $v \notin \mathbf{V}_{PSD}$, then add $v$ to $\mathbf{V}_{PSD}$, enqueue $v$ onto $\mathbf{Q}$;
        end if;
    end for;
Until $\mathbf{Q}$ is empty
Return $\mathbf{V}_{PSD}$;
```

By implementing this PSD algorithm, we find all the nodes influenced by the current corrections in tracking data. We denote the affected nodes and corrected nodes as set V_{PSD}. Using the human corrections as hard constraints, we perform the LIP problem in Eq.7 on V_{PSD}, which updates the tracklet association and corresponding node features, i.e., automatically propagates correction information to nodes close to corrected nodes. We use the gating regions to control how far the correction can be propagated in the local neighborhood. While the recommender system is run iteratively, this correction propagation performs like the Butterfly Effect and sweeps gradually over the entire node pool.

After correction propagation, data relocation is performed before we move to the next iteration:

(1) top μ nodes with high scores are recommended for a user to check (e.g., $\mu = 20$) . After human verification and correction, the top μ nodes are separated into two subsets: nodes with errors are assigned to Positive Set and nodes without errors are assigned to Negative Set;

(2) the nodes in V_{PSD} which have low scores after correction propagation are moved to Negative Set. Those with high scores are moved to Positive Set and the rest remains in the uncertain node pool;

(3) rated nodes by the recommender system with low scores are assigned to Negative Set, only if they are not in the propagation set $\mathbf{V}_{PSD}$ found by **PSD** algorithm.

Our iterative recommender system with correction propagation is summarized in Algorithm 3:

Algorithm 3: Iterative Recommender System with Correction Propagation

```
Input : node set $\mathbf{V}$ of all nodes and their features;
```

```
**Initialization** : **Uncertain Node Pool=V**; **Temporary Set P**=$\emptyset$;
**Positive Set**$\leftarrow$ Pick $\mu$ nodes with errors in tracking data;
**Negative Set** $\leftarrow$ Pick $\mu$ nodes without errors in tracking data;
Repeat
    Update **RecommenderProfiles** using Eq.9;
    Compute the scores of nodes in the **Uncertain Node Pool** by Eq.10;
    **for** all node v $\in$ **V**
        **if** node score of v $> \omega$;
            **if** v is one of the top $\mu$ nodes, **then** recommend v for human check;
                **if** v is verified as a node with errors, **then** add v to **Positive Set**;
                **else** add v to **Negative Set**;
                **end if**;
            **else** add v to **P**;
            **end if**;
        **else if** node score of v $< \omega/2$;
            add v to **Negative Set**;
        **end if**;
        **Uncertain Node Pool** $\leftarrow$ **Uncertain Node Pool** $- v$;
    **end for**;
    Find $\mathbf{V_{Propagation}}$ using **Algorithm 2**;
    Implement data-association algorithm within $\mathbf{V_{Propagation}}$ where human
corrections are added as additional hard constraints;
    Add $\mathbf{P} - \mathbf{P} \cap \mathbf{V_{Propagation}}$ to **Uncertain Node Pool**;
until **Uncertain Node Pool** is empty.
```

5 Experimental Results

5.1 Datasets

To test our proposed recommender system, we perform experiments on three different biomedical image sequences. Specifications of these datasets are summarized in Table 1, while some sample images are shown in Fig.4. In Datasets 1 and 2 which are downloaded from [15], the main challenges are the frequent occurring of cell merging and division (causing many tracklets in a small local neighborhood), false positives in detection (causing distractions and wrong associations) and miss detections (causing broken tracklets). In dataset 3 obtained from [7], the main challenges are false positive distractions due to low image contrast, fast motion blurring and object camouflaging.

	#images	Object type	#Objects per frame	Image size
Set1	400	Stem Cells	20-100	1392×1040
Set2	380	Stem Cells	100-400	1392×1040
Set3	10000	Fruit Flies	52	848×480

Table 2. Specifications of datasets.

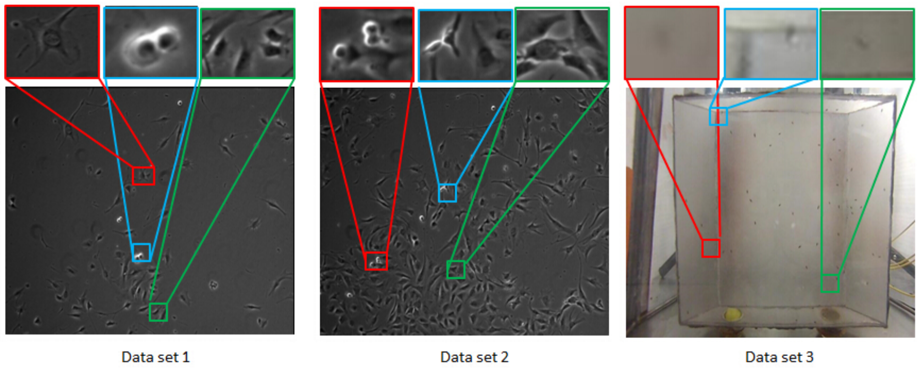

Fig. 4. Examples of 3 datasets.

5.2 Quantitative Evaluation

To evaluate how well our recommender system and correction propagation can assist human annotators on debugging object tracking results, we use two metrics:

(1) efficiency: how fast will the number of uncertain nodes reduce so less human effort is needed to verify and correct nodes?

(2) effectiveness: what percentage of false nodes (nodes with tracking errors) is detected by the recommender system (i.e., how effective can our system guide human annotators towards false nodes)?

Fig.5(a) shows the number of nodes remaining in the uncertain node pool at different iterations for the 3 datasets. We observe that it decreases drastically when using our recommender system and correction propagation to assist human annotators, which proves that our system is efficient with less human effort to debug object tracking results. Fig.5(b) shows "*# of undetected false nodes/# of total false nodes*" at different iterations on the 3 datasets. It is noticeable that the percentage curves drop quickly, and within 20 iterations the number of undetected false nodes falls below 5 percent out of the total false nodes, which proves that our recommender system can effectively guide human annotators to find questionable tracking results for correction.

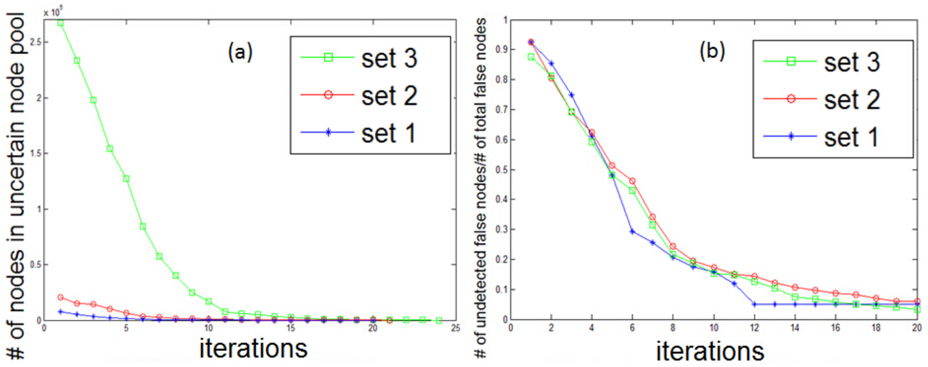

Fig. 5. (a) Number of nodes in the uncertain node pool; (b): # of undetected false nodes/# of total false nodes.

5.3 Quantitative Comparison

In order to demonstrate the effect of our recommender system and correction propagation, we compare the following four approaches:

(1) **Random selection without propagation**. Nodes in the uncertain node pool are randomly selected by humans to verify and correct. Human corrections are not propagated to neighboring nodes.

(2) **Random selection with propagation**. Nodes in the uncertain node pool are randomly selected by humans to verify and correct. Human corrections are propagated to neighboring nodes.

(3) **Recommendation without propagation**. Nodes in the uncertain node pool are recommended by our system for humans to verify and correct. Human corrections are not propagated to neighboring nodes.

(4) **Recommendation with propagation**. Nodes in the uncertain node pool are recommended by our system for humans to verify and correct. Human corrections are propagated to neighboring nodes.

As shown in Fig.6, **Random selection without propagation** is very inefficient for human to debug the object tracking results since the human randomly verifies uncertain nodes without any guidance and no further usage is applied to human correction in each iteration. With correction propagation applied to the random selection, **Random selection with propagation** decreases the number of uncertain nodes faster, but the human annotators still have no clue on what nodes should be checked. **Recommendation without propagation** adds recommendation to humans for debugging tracking results, which reduces the number of uncertain nodes further faster. Finally, when **Recommendation with propagation** is applied together, the uncertain node pool shrinks the fastest, which means that it takes much less time and human labeling costs.

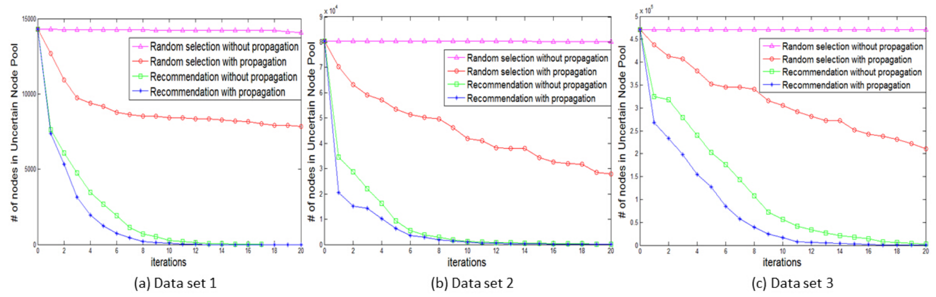

Fig. 6. Uncertain node pool shrinking rates of 4 different approaches on 3 datasets.

From Fig.6, we observe that both our recommender system and correction propagation can efficiently reduce the size of the uncertain node pool. We evaluate the effectiveness of the four approaches in terms of "*# of undetected false nodes/# of total false nodes*". In Fig.7, our recommender system with correction propagation performs beyond the other 3 approaches in helping humans detect nodes with tracking errors. Another observation from Fig.7 is that random selection has much lower effectiveness than recommender system since it has no guidance on which nodes should be verified and corrected.

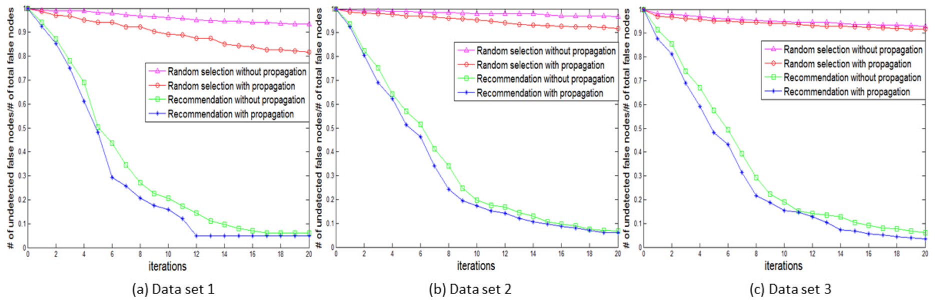

Fig. 7. (# of undetected false nodes/# of total false nodes) of 4 different approaches.

5.4 Qualitative Examples

In Fig.8 we present some examples to show how our recommender system helps humans find similar false tracking data. In the left box, a cell is detected as multiple fragments and similar cases are found by the recommender system. In the right box, figure shows the wrong ID associations due to nearby distractors. In our recommender system, all of these tracking errors can be represented by their nodes' feature vectors. Initially, human selects some false nodes and makes corresponding corrections, then the recommender system search similar false nodes in the uncertain node pool and let human to verify and correct them.

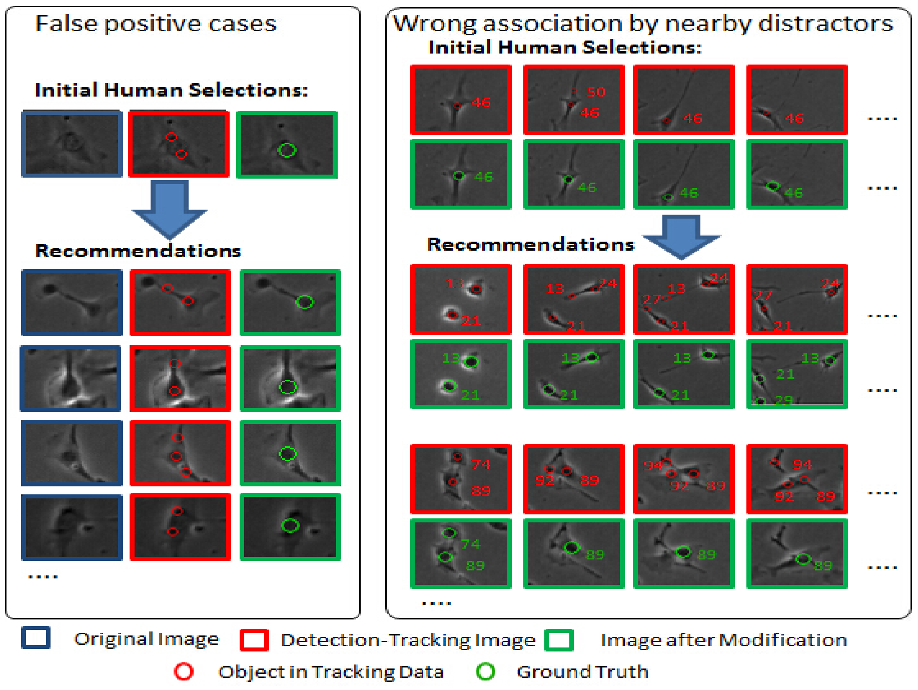

Fig. 8. Examples of recommended nodes for human verification and correction.

6 Conclusion

In this paper, a novel iterative recommender system is proposed to help humans debug tracking errors in data generated by various tracking algorithms. Instead of object-by-object or frame-by-frame checking, human annotators only need to debug a sparse set of nodes recommended by the recommender system in each iteration. Multiple human debuggers work on the tracking data independently and their debugging results are collected together. Since every correction made on one node will result in a chain reaction involving other neighboring nodes, we propagate all the corrections in the neighborhood, which ensures the tracking consistency and speeds up the debugging process. Each annotator's profile parameters on the recommender system is updated based on their new debugged nodes. The process is iterated until the uncertain node pool is empty. We tested our approach on three sets of biomedical image sequences. The results show that our recommender system with correction propagation can efficiently and effectively guide human annotators to debug tracking data.

Acknowledgement

This research was supported by NSF EPSCoR grant IIA-1355406 and NSF CAREER award IIS-1351049, University of Missouri Research Board, ISC and CBSE centers at Missouri University of Science and Technology.

References

1. Bonneau, S., et al.: Single quantum dot tracking based on perceptual grouping using minimal paths in a spatiotemporal volume. IEEE Trans on Image Processing, (2005), 14(9): 1384-1395.
2. Burke, R.: Knowledge-based recommender systems. Encyclopedia of Library and Information Systems, (2000), 69(Supplement 32): 175-186.
3. Cortes, C., et al.: Support-vector networks. Machine Learning, (1995), Springer.
4. Fortmann, T., et al.: Sonar tracking of multiple targets using joint probabilistic data association. IEEE J. Oceanic Engineering, (1983), 8(3): 173-184.
5. Huang, C., et al.: Robust Object Tracking by Hierarchical Association of Detection Responses. ECCV, (2008).
6. Kuhn, H.: The Hungarian Method for the assignment problem. Naval Research Logistics Quarterly, (1955).
7. Li, M., et al.: Track fast-moving tiny flies by adaptive LBP feature and cascaded data association. ICIP, (2013).
8. Lops, P., et al: Content-based recommender systems: State of the art and trends. Recommender Systems Handbook. Springer US, (2011).
9. Park, D., et al: A literature review and classification of recommender systems research. Expert Systems with Applications, (2012).
10. Reid, D.: An algorithm for tracking multiple targets. IEEE Trans on Automatic Control, (1979), 24(6): 843-854.
11. Resnick, P., et al.: Recommender systems. Communications of the ACM, (1997).
12. Yilmaz, A., et al.: Object tracking: A survey. ACM Computing Surveys, (2006), Vol. 38, No. 4, Article 13.
13. Zhang, L., et al.: Global data association for multi-object tracking using network flows. CVPR, (2008).
14. Zhang, T. and Iyengar, Y.: Recommender systems using linear classifiers. The Journal of Machine Learning Research, (2002), 2: 313-334.
15. http://www.celltracking.ri.cmu.edu/downloads.html