tracktour Model

Draga Doncila Pop, Pierre Le Bodic, Juan Nunez-Iglesias

Monash University

1 Introduction

Our tracking optimization model is a type of network-flow model similar to ECLIP [1] but modified to take cell detections as input, omit any initial learning steps in favor of distance-based costs, and allow for track merges to model cell occlusion. We detail its construction below.

2 Tracking Model

To track a set of detected objects, we model the movement of cells from to frame to another as a network flow on a graph, starting from a dummy source vertex that is connected to all cells of the first frame, and terminating at a dummy target vertex connected to all cells of the last frame. The cells in two consecutive frames are connected, and we track a cell between frames by following the flow moving through vertices. In the more realistic version of the tracking problem that we model, cells are allowed to enter and exit at an arbitrary frame, and they are allowed to divide.

2.1 Migration Flow

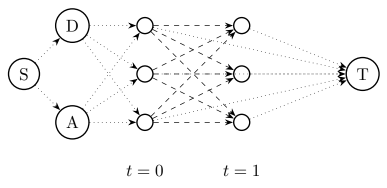
We create the candidate graph of our tracking model by first defining vertices at the center coordinates of all identified cells in each frame. Let $\mathbf{V}$ be the set of all vertices identified in this way.

To represent a cell moving from frame to frame (migration flow), we add edges from each vertex $v \in V$ in frame t to its k-nearest neighbors in frame $t+1$, where k is the only parameter to our model, and takes a default value of 10. This should be sufficient for most of the datasets [2]. While, theoretically, connecting the cell to all other cells in the next frame would ensure we do not exclude the correct neighbor, this would lead to quadratic growth of the candidate graph as the number of cells in each frame increases. Since the cell is unlikely to travel across the entire frame, limiting the number of nearest neighbors retrieved allows us to bound the size of the candidate graph with minimal risk to the optimality of the solution [2]. To further improve performance, the vertex coordinates are stored in KD-Trees [3] to support retrieval in logarithmic time. The cost for these migration edges is simply the Euclidean distance between the cell's position at frame t and the position of its neighbor in frame $t+1$.

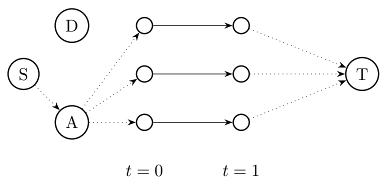
For a constant number of cells per frame, this graph could be turned into a suitable network flow graph by connecting a dummy source vertex S to all vertices $v \in V$ in the first frame, and all vertices in the final frame to a dummy target vertex, T. However, because cells can appear, disappear, and divide, we need a way for additional flow to enter and leave the network.

2.2 Appearance & Disappearance

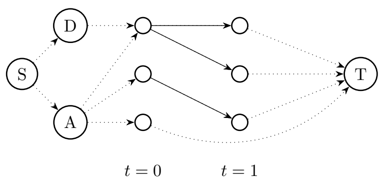
Cells near the borders of the image can come in and out of view in different frames. If a cell exits for one frame and then returns, we cannot determine whether it is the same cell that initially left. Therefore, the track of the cell that left the frame should end, and a new track should begin for the cell that just entered. To represent cells appearing from the borders of the image, we introduce a dummy vertex A. We connect vertex A to S and all vertices v, in all frames. To allow cells to exit the image, we add edges vT for all vertices v. The cost of a given cell v appearing or exiting in any frame is the Euclidean distance of the cell from the nearest border of the image. These costs are associated with edges Av and vT for all vertices v.



(a) A fully defined candidate graph.



(b) A migration-only solution.



(c) A solution including a division. The bottom cell in frame 0 exits the image.

Figure 1: A toy candidate graph and two solutions. Dotted edges are adjacent to virtual vertices and would be removed in the final solution. Dashed edges in 1a may be part of the final solutions. 1b and 1c show the candidate graph after it is solved, and zero-flow edges are removed. Solid edges in 1b and 1c are the selected edges in the final solution.

2.3 Division

Division flow is slightly different from appearance and migration because a cell needs two units of flow incoming if it is to supply flow to two children in the next frame. We therefore introduce a final dummy vertex, D, and connect it to S and all vertices except those in the final frame, which cannot divide. The cost of edges Dv where v is a vertex at frame t is the distance between the two closest neighbors of v in frame $t+1$ (its most likely children) plus the distance from v to its closest neighbor in frame $t+1$. This penalizes children who are too far apart and makes division more costly than migration since it is far less frequent.

Figure 1a shows the full candidate graph for a toy two-frame example with three cells per frame. Figure 1b shows the correct solution to the candidate graph (the three cells simply migrate from one frame to the next), while Figure 1c shows a solution that involves a division and cell exiting instead.

2.4 Flow Constraints

With our graph $G = (V, E)$ defined, we introduce a variable f_{ij} which encodes the amount of flow being sent between vertices i and j for all edges $e \in E$ and use it to define the following constraints.

Flow conservation: Flow entering a vertex must be equal to flow leaving a vertex.

$$\sum_{e_{ij} \in E} f_{ij} + f_{Aj} + f_{Dj} = \sum_{e_{jk} \in E} f_{jk} + f_{jt} \; \forall i, j \in V \tag{1}$$

Vertex demand: Each vertex must have a minimum of one flow passing through it. To make sure D can't be used to fuel appearance or migration, flow from D can't be used to satisfy this demand - it can only provide additional flow if a vertex needs to divide in the next frame.*

$$\sum_{e_{iv} \in E} f_{iv} \geq 1 \; \forall i, v \in V \text{where } i \neq D \tag{2}$$

2.5 Merging

We allow tracks to merge into a single vertex at any frame. While this is not physically possible for cells, we use merges to model cells that may have been missed in the detection/segmentation step. We use these merges in an experimental interface to guide the user to correct the cell detections. To allow tracks to continue in a merged state for more than a single frame, we introduce a final constraint on edge capacity.

Edge capacity: Each edge has a maximum of two units of flow traveling along it.*

$$f_{ij} \leq 2 \; \forall e_{ij} \in E \tag{3}$$

***Note:** Solving with merges can be disabled in tracktour, in which case the maximal capacity along any edge is 1, and the vertex demand must strictly equal 1.

2.6 Solving

As described in detail above, each edge of the flow network has an associated cost: euclidean distance for migration edges, distance to closest image border for appearance/exit edges, and distance to closest child + smallest interchild distance for division edges. Let c_{ij} denote the cost associated with edge f_{ij}.

The objective function of the flow problem is finding the minimum cost flow that satisfies all constraints: $\min \sum_{e_{ij} \in E} f_{ij} * c_{ij}$.

The problem is given to Gurob[4] as a linear program and solved using the dual simplex algorithm. The output is an exact, optimal solution assigning a continuous f_{ij} value for each edge. The final tracks are the subset of edges with $f_{ij} > 0$ and no dummy vertex adjacent - this “discretizes” the solution. Because we enforce a demand of at least 1 flow through each vertex (Equation 2), this subset of edges will connect all detections across time to form the lineage tracks.

References

[1] E. Türetken, X. Wang, C. J. Becker, C. Haubold, and P. Fua, “Network flow integer programming to track elliptical cells in time-lapse sequences,” *IEEE transactions on medical imaging*, vol. 36, no. 4, pp. 942–951, 2016.

[2] W. Jiuqing, C. Xu, and Z. Xianhang, “Cell tracking via structured prediction and learning,” *Machine Vision and Applications*, vol. 28, no. 8, pp. 859–874, 2017.

[3] J. L. Bentley, “Multidimensional binary search trees used for associative searching,” *Communications of the ACM*, vol. 18, no. 9, pp. 509–517, 1975.

[4] Gurobi Optimization, LLC, “Gurobi Optimizer Reference Manual,” 2023.