Globally-Optimal Greedy Algorithms for Tracking a Variable Number of Objects

Hamed Pirsiavash Deva Ramanan Charless C. Fowlkes
Department of Computer Science, University of California, Irvine
{hpirsiav,dramanan,fowlkes}@ics.uci.edu

Abstract

We analyze the computational problem of multi-object tracking in video sequences. We formulate the problem using a cost function that requires estimating the number of tracks, as well as their birth and death states. We show that the global solution can be obtained with a greedy algorithm that sequentially instantiates tracks using shortest path computations on a flow network. Greedy algorithms allow one to embed pre-processing steps, such as nonmax suppression, within the tracking algorithm. Furthermore, we give a near-optimal algorithm based on dynamic programming which runs in time linear in the number of objects and linear in the sequence length. Our algorithms are fast, simple, and scalable, allowing us to process dense input data. This results in state-of-the-art performance.

1. Introduction

We consider the problem of tracking a variable number of objects in a video sequence. We approach this task as a "spatiotemporal grouping" problem, where all image regions must be labeled as background or as a detection belonging to a particular object track. From such a grouping perspective, one must explicitly estimate (a) the number of unique tracks and (b) the spatiotemporal extent, including the start/termination times, of each track (Fig.1).

Approaches to accomplishing the above tasks typically employ heuristics or expensive algorithms that scale exponentially in the number of objects and/or super-linearly in the length of the video. In this paper, we outline a family of multi-object tracking algorithms that are:

1. Globally optimal (for common objective functions)
2. Locally greedy (and hence easy to implement)
3. Scale linearly in the number of objects and (quasi)linearly with video-length

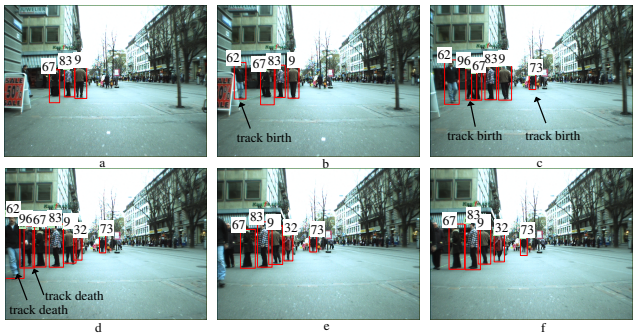

Figure 1. We treat the problem of multi-target tracking through a perspective of "spatiotemporal grouping", where both a large number of groups and their spatiotemporal extent (e.g., the number of objects and their track births and deaths) must be estimated. We show the output of an efficient, linear-time algorithms for solving this computational problem on the ETHMS dataset [9]. In this video clip our method returns hundreds of correct tracks, as evident by the overlaid track number.

Our contribution is grounded in a novel analysis of an integer linear program (ILP) formulation of multi-object tracking [14, 25, 3, 17, 2, 18]. Our work most closely follows the min-cost flow algorithm of [25]. We show that one can exploit the special structure of the tracking problem by using a greedy, successive shortest-path algorithm to reduce the best-previous running time of $O(N^3 \log^2 N)$ to $O(KN \log N)$, where K is the unknown, optimal number of unique tracks, and N is the length of the video sequence. The intuition behind the greedy approach stems from this surprising fact (Fig.2): *the optimal interpretation of a video with $k+1$ tracks can be derived by a local modification to the solution obtained for k tracks*. Guided by this insight, we also introduce an approximate greedy algorithm whose running time scales linearly with sequence length (i.e., $O(KN)$), and is in practice several orders of magnitude faster with no observable loss in accuracy. Finally, our greedy algorithms allow for the embedding of various pre-processing or post-processing heuristics (such as non-maximum suppression) into the tracking algorithm, which can boost performance.

2. Related Work

Classic formulations of multi-object tracking focus on the data association problem of matching instance labels with temporal observations [11, 6, 7, 13]. Many approaches assume manual initialization of tracks and/or a fixed, known number of objects [14]. However, for many real-world tracking problems, such information is not available. A more general spatiotemporal grouping framework is required in which these quantities are automatically estimated from video data.

A popular approach to multi-object tracking is to run a low-level tracker to obtain "tracklets", and then stitch together tracklets using various graph-based formalisms or greedy heuristics [15, 22, 16, 19, 2]. Such graph-based algorithms include flow-networks [25], linear-programming formulations [14], and matching algorithms [15]. One of the contributions of this paper is to show that with a *particular* choice of low-level tracker, and a *particular* schedule of track instantiation, such an algorithm can be *globally-optimal*.

We rely on an increasingly common ILP formulation of tracking [14, 25, 3, 17, 2, 18]. Such approaches restrict the set of possible object locations to a finite set of candidate windows on the pixel grid. Because standard linear programming (LP) relaxations do not scale well, many algorithms process a small set of candidates, with limited or no occlusion modeling. This can produce broken tracks, often requiring a second merging stage. Our scalable algorithm is able to process much larger problems and directly produces state-of-the-art tracks.

Our work relies heavily on the min-cost flow network introduced for temporal data association in [25]. We compare our results with the min-cost solver used in that work [12], and verified that our $O(KN \log N)$ algorithm produces identical results, and that our approximate $O(KN)$ algorithm produces near-identical results when properly tuned.

In concurrent work, Berclaz et al. describe a O(KN log N) algorithm for multi-object tracking in [4]. It is similar in many respects with some differences: Our graph representation has a pair of nodes for each detection. This allows us to explicitly model object dynamics through transition costs, and allows for a simpler flow-based analysis. In addition, our algorithm instantiates tracks in a greedy fashion, allowing for the integration of pre-processing steps (e.g., non-max-suppression) that improve accuracy. Finally, we also describe approximate O(KN) algorithms that perform near-identical in practice.

3. Model

We define an objective function for multi-object tracking equivalent to that of [25]. The objective can be derived from a generative perspective by considering a Hidden Markov Model (HMM) whose state space is the set of true object locations at each frame, along with a prior that specifies likely state transitions (including births and deaths) and an observation likelihood that generates dense image features for all objects and the background.

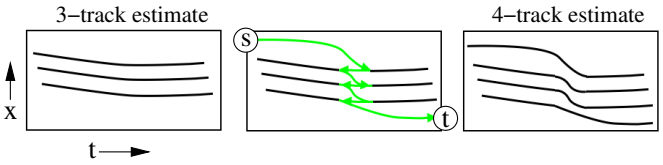

Figure 2. The intuition behind our optimal greedy algorithm. Assume that we are tracking the 'x' location of multiple objects over time. On the **left**, we show the optimal estimate of 3 object trajectories. Given the knowledge that an additional object is present, one may need to adjust the existing tracks. We show that one can do this with a shortest-path/minflow computation that pushes flow from a source to a terminal (**middle**). The solution can reverse flow along existing tracks to "cut and paste" segments, producing the optimal 4-track estimate (**right**). We further speed up this process by approximating such edits using fast dynamic programming algorithms.

3.1. Independent tracks

We write x for a vector-valued random variable that represents the location of a particular object, as given by a pixel position, scale, and frame number:

$$x = (p, \sigma, t) \qquad x \in V \tag{1}$$

where V denotes the set of all spacetime locations in a video.

Prior: We write a single track as an ordered set of state vectors $T = (x_1, \ldots x_N)$, ordered by increasing frame number. We write the collection of tracks as a set $X = \{T_1, \ldots T_K\}$. We assume that tracks behave independently of each other, and that each follows a variable-length Markov model:

$$P(X) = \prod_{T \in X} P(T)$$

$$\text{where} \qquad P(T) = P_s(x_1)\Big(\prod_{n=1}^{N-1} P(x_{n+1}|x_n)\Big)P_t(x_N)$$

The dynamic model $P(x_{n+1}|x_n)$ encodes a smoothness prior for track location. We write $P_s(x_1)$ for the probability of a track starting at location x_1, and $P_t(x_N)$ for the probability of a track transitioning into a termination state from location x_N. If the probability of termination is low, the above prior will tend to favor longer, but fewer tracks so as to minimize the total number of terminations. If these probabilities are dependent on the spatial coordinate of x, they can model the fact that tracks tend to terminate near image borders or start near entry points such as doorways.

Likelihood: We write $Y = \{y_i | i \in V\}$ for the set of feature vectors observed at all space-time locations in a video. For example, these could be the set of gradient histogram features that are scored by a sliding-window object detector. We now describe a likelihood model for generating Y given the set of tracks X. We make two assumptions: 1) there exists a one-to-one mapping between a putative object state x and space-time location index i and 2) tracks do not overlap ($T_k \cap T_l = \emptyset$ for $k \neq l$). Together, both imply that a location can be 'claimed' by at most one track. We write y_x for the image features at location x; these features are generated from a foreground appearance model. Feature vectors for unclaimed windows are generated from a background model:

$$P(Y|X) = \Big(\prod_{T \in X} \prod_{x \in T} P_{fg}(y_x) \Big) \prod_{i \in V \setminus X} P_{bg}(y_i) \qquad (2)$$
$$= Z \prod_{T \in X} \prod_{x \in T} l(y_x)$$
$$\text{where} \quad l(y_x) = \frac{P_{fg}(y_x)}{P_{bg}(y_x)} \quad \text{and} \quad Z = \prod_i P_{bg}(y_i)$$

The likelihood is, up to a constant, only dependent on features of the windows which are part of the set of tracks. If we assume that the foreground and background likelihoods are Gaussian densities with the same covariance,

$$P_{fg}(y_x) = N(y_x; \mu_{fg}, \Sigma) \quad \text{and} \quad P_{bg}(y_x) = N(y_x; \mu_{bg}, \Sigma),$$

we can write the log-likelihood-ratio as a linear function ($\log l(y_x) = w \cdot y_x$), akin to a logistic regression model derived from a class-conditional Gaussian assumption. This model provides a generative motivation for the linear templates that we use as local detectors in our experiments.

3.2. Track interdependence

The above model is reasonable when the tracks do not overlap or occlude each other. However, in practice we need to deal with both occlusion and non-maxima suppression.

Occlusion: To model occlusions, we allow tracks to be composed of state vectors from non-consecutive frames e.g., we allow t_n and t_{n+1} to differ by up to k frames. The dynamic model $P(x_{n+1}|x_n)$ for such k-frame skips captures the probability of observing the given k-frame occlusion.

Non-maxima suppression: When we consider a dense set of locations V, there will be multiple tracks which score well but correspond to the same object (e.g., a good track shifted by one pixel will also have a high probability match to the appearance model). A complete generative model could account for this by producing a cluster of image features around each true object location. Inference would "explain away" evidence and enforce exclusion. In practice, the typical solution is to apply non-max suppression (NMS) as a pre-process to prune the set of candidate locations V prior to multi-object tracking [19, 6, 14, 25].

In our experiments, we also utilize NMS to prune the set V and as a heuristic for explaining away evidence. However, we show that the NMS procedure can be naturally embedded within our iterative algorithm (rather than as a pre-process). By suppressing extra detections around each track as it is instanced, we allow for the possibility that the prior can override the observation term and select a window which is not a local maxima. This allows the NMS procedure to exploit temporal coherence. The recent work of [2] make a similar argument and add an explicit non-overlapping constraint to their ILP, which may sacrifice tractability. We demonstrate in Sec. 6 that our simple and fast approach produces state-of-the-art results.

4. MAP Inference

The *maximuim a posteriori* (MAP) estimate of tracks given the collection of observed features is:

$$X^* = \operatorname*{argmax}_X P(X)P(Y|X) \qquad (3)$$
$$= \operatorname*{argmax}_X \prod_{T \in X} P(T) \prod_{x \in T} l(y_x) \qquad (4)$$
$$= \operatorname*{argmax}_X \sum_{T \in X} \log P(T) + \sum_{x \in T} \log l(y_x) \qquad (5)$$

We drop the constant factor Z and take logarithm of the objective function to simplify the expression while preserving the MAP solution. The above can be re-written as an Integer Linear Program:

$$f^* = \operatorname*{argmin}_f C(f) \qquad (6)$$

$$\text{with} \quad C(f) = \sum_i c_i^s f_i^s + \sum_{ij \in E} c_{ij} f_{ij} + \sum_i c_i f_i + \sum_i c_i^t f_i^t \qquad (7)$$

$$\text{s.t.} \quad f_{ij}, f_i, f_i^s, f_i^t \in \{0, 1\}$$
$$\text{and} \quad f_i^s + \sum_j f_{ji} = f_i = f_i^t + \sum_j f_{ij} \qquad (8)$$

where f_i is a binary indicator variable that is 1 when space-time location i is included in some track. The auxiliary variables f_{ij} along with the second constraint (8) ensures that at most one track claims location i, and that multiple tracks may not split or merge. With a slight abuse of notation, let us write x_i for the putative state corresponding to location i:

$$c_i^s = -\log P_s(x_i), \quad c_i^t = -\log P_t(x_i), \qquad (9)$$
$$c_{ij} = -\log P(x_j|x_i), \quad c_i = -\log l(y_i).$$

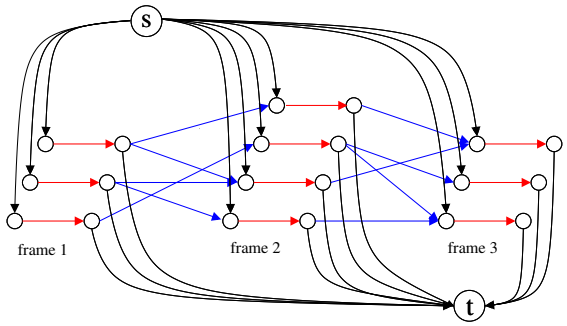

Figure 3. The network model from [25] for three consecutive frames of video. Each space-time location $i \in V$ is represented by a pair of nodes connected by a **red** edge. Possible transitions between locations are modeled by **blue** edges. To allow tracks to start and end at any spatiotemporal point in the video, each location i is connected to both a start and termination node. All edges are directed and unit capacity. The costs are c_i for red edges, c_{ij} for blue edges and c_i^s and c_i^t for black edges.

encode the track start, terminate, transition, and observation likelihoods respectively. We define the edge set E to span the set of permissible state transitions given by our dynamic model (Sec.3.1).

4.1. Equivalence to network flow

To solve the above problem, we can relax the integer constraints in (8) to linear box constraints (e.g., $0 \leq f_i \leq 1$) This relaxation yields a unit capacity network flow problem whose constraint matrix is *totally unimodular*, implying that optimal solutions to the relaxed problem will still be integral [1]. In particular, assume that we knew the number of tracks in a video to be K. Let F_K denote the set of flow conservation and unit capacity constraints along with the additional constraints

$$F_K = \left\{ \begin{array}{cc} f_{ij}, f_i, f_i^s, f_i^t \in [0,1], & \sum_i f_i^s = K, \\ f_i^s + \sum_j f_{ji} = f_i = f_i^t + \sum_j f_{ij}, & \sum_i f_i^t = K \end{array} \right.$$

Minimizing $C(f)$ subject to constraints F_K is an instance of a *minimum cost flow* problem [1, 25]. Such problems are similar to max-flow problems (commonly used in vision for solving graph-cut problems [5]), except that edges in the flow network are labeled with a cost as well as capacity. The cost of a flow is defined to be the sum, over all edges, of the cost of each edge multiplied by the flow through that edge. Finding the MAP estimate of K tracks corresponds to finding a minimum cost flow that pushes K units of flow from the source to the sink.

Figure 3 shows an example flow network constructed from the tracking problem. Each space-time location i, or equivalently putative object state x_i, corresponds to a pair of nodes (u_i, v_i) connected by an edge of cost c_i. Each transition between successive windows is represented by an edge (v_i, u_j) with cost c_{ij}. Finally, nodes s and t are introduced with edges (s, u_i) corresponding to track starts and edges (v_i, t) for terminations (with cost c_i^s and c_i^t respectively). All edges have unit capacity. Pushing K units of flow from s to t yields a set of K disjoint st-paths, each of which corresponds to one of the optimal tracks $T \in X$.

5. Finding min-cost flows

Zhang et al. [25] describe how to solve the above optimization problem in $O(mn^2 \log n)$ time using a push-relabel method [12], where n is the number of nodes (e.g. detection windows) in the network graph and m is the number of edges. Assuming that n and m scale linearly with the number of frames N (reasonable given a fixed number of detections per frame), the algorithm takes $O(N^3 \log N)$ to find K tracks. Furthermore, the cost of the optimal solution, $\min_{f \in F_K} C(f)$ is convex in K [25] so one can use a bisection search over K (upper-bounded by the number of detections) to find the optimal number of tracks with a total running time $O(N^3 \log^2 N)$.

In the following, we show that one can solve the multi-object tracking problem in $O(KN \log N)$ by solving $K+1$ shortest-path problems. This considerable reduction in complexity is due to two particular properties of the network in Fig.3:

1. All edges are unit capacity.
2. The network is a directed acyclic graph (DAG).

The above conditions allow one to use dynamic programming (DP) algorithms to compute shortest paths. We describe a novel DP algorithm that is necessary to construct a globally-optimal $O(KN \log N)$ algorithm. We also show that DP produces the optimal solution for $K = 1$ in $O(N)$ and high-quality approximate solutions for $K > 1$ in $O(KN)$. We begin by describing the optimal $O(KN \log N)$ algorithm based on successive shortest paths (introduced in Fig.2).

5.1. Successive Shortest-paths

We now describe a *successive shortest path* algorithm [1] for solving min-cost flow problems for DAG networks with unit-capacity links. Given a graph G with an integral flow f, define the residual graph $G_r(f)$ to be the same as the original graph except that all edges used in the flow f are reversed in direction and assigned negative their original cost. We initialize the algorithm by setting the flow f to be zero and then iterate the following two steps:

1. Find the minimum-cost path γ from s to t in $G_r(f)$
2. If total cost of the path $C(\gamma)$ is negative, update f by pushing unit-flow along γ

until no negative cost path can be found. Since each path has unit capacity, each iteration increases the total flow by 1 and

decreases the objective by $C(\gamma)$. The algorithm terminates after $K+1$ iterations having found a minimum cost flow. Pushing any further flow from s to t will only increase the cost.

We refer the reader to [1] for a proof of the correctness of the algorithm but give a brief outline. We say a flow f is F_K-*feasible* if it satisfies the constraint set F_K. A necessary and sufficient condition for f to be a minimum cost flow of size K is that it be F_K-feasible and that there does not exist a negative-cost directed cycle in $G_r(f)$. The successive shortest-paths algorithm above starts with a F_0-feasible flow and at each iteration i yields a new flow which is F_i-feasible. Furthermore, each step of the algorithm modifies edges along a single path and can be shown to not introduce any negative weight cycles.

Figure 4 shows example iterations of this algorithm and the resulting sequence of residual graphs. Note that the shortest path in the residual network may instance a new track and/or edit previous tracks by removing flow from them (by pushing flow through the reverse edges).

In each iteration, we need to find a shortest st-path. We would like to use Dijkstra's algorithm to compute the shortest path in $O(N \log N)$, making the overall algorithm $O(KN \log N)$ where K is the optimal number of tracks. Unfortunately, there are negative edge costs in our original network, precluding the direct application of Dijkstra's algorithm. Fortunately, one can convert any min-cost flow network to an equivalent network with non-negative costs [1]. This conversion requires computing the shortest-path of every node from s in the original graph G. For general graphs with negative weights, this computation takes $O(N^2)$ using the Bellman-Ford algorithm [1]. For DAGs, one can use a $O(N)$ dynamic programming algorithm, which we describe below. The successive shortest path algorithm thus runs in $O(KN \log N)$ operations and returns the global minima for our tracking problem (Equation 3).

5.2. Dynamic Programming Solution for $K = 1$

We now present a $O(N)$ dynamic programming (DP) algorithm for computing the shortest path of every node to s. We will also show that this algorithm solves the min cost flow problem for $K = 1$. Because each edge in the network is of unit capacity, the minimum cost unit flow must correspond to the shortest path from node s to t. Because the original network graph is a DAG, one can construct a partial ordering of nodes and use DP to compute shortest paths by sweeping from the first to last frame. This is similar to DP algorithms for tracking but augmented to estimate both the birth and death time of a track.

Assume that nodes are ordered in time, and let $cost(i)$ represent the minimum cost of a track passing through node i. We initialize $cost(i)$ for detections in the first frame to be $cost(i) = c_i + c_i^s$. We can then recursively compute the cost in successive frames as:

$$cost(i) = c_i + \min(\pi, c_i^s) \quad \text{where} \quad \pi = \min_{j \in N(i)} c_{ij} + cost(j) \tag{10}$$

where $N(i)$ is the set of detections from the previous k frames that can transition to detection i. The cost of the optimal ending at node i is then $cost(i) + c_i^t$, and the overall shortest path is computed by taking a min over i. By caching the argmin values at each node, we can reconstruct the shortest path to each node in a single backward sweep.

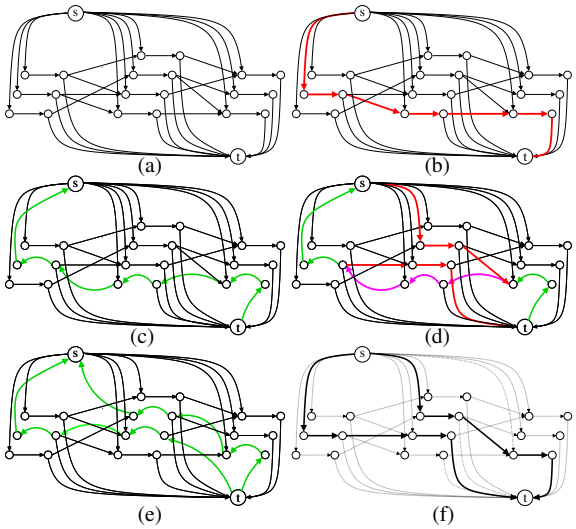

Figure 4. Illustration of successive shortest path algorithm. (a) The tracking problem modeled as a graph as described in Fig.3. The algorithm should send a given amount of flow from source node s to the terminal t. (b) One unit of flow f_1 is passed through the shortest path (in red) from source to terminal. (c) The residual graph $G_r(f_1)$ produced by eliminating the shortest path and adding edges (in green) with unit capacity and negative cost with the opposite direction. (d) The shortest path found in the residual graph. In this example, this path uses previously added edges, pushing flow backwards and "editing" the previously instanced tracks. (e) Residual graph after passing two units of flow. At this point, no negative cost paths exist and so the algorithm terminates and returns the two tracks highlighted in (f). Note that the algorithm ultimately splits the track instanced in the first step in order to produce the final optimal set of tracks. In this example only one split happened in an iteration, but it is possible for a shortest path to use edges from two or more previously instanced tracks, but it is very rare in practice. Our dynamic programming algorithm cannot resolve any splitting since the residual graph has cycles, however the 2-pass dynamic programming algorithm can resolve the situations when any new shortest path splits at most one previously instanced track.

5.3. Approximate DP solution for $K > 1$

We now propose a simple greedy algorithm to instance a variable, unknown number of disjoint, low-cost tracks. Start with the original network-flow graph:

1. Find the shortest path from s to t using DP.

2. If the cost of the path is negative, remove nodes on the path and repeat.

The above algorithm performs $K+1$ iterations of DP to discover K tracks – the last instanced track is ignored since it increases the overall cost. Its running time is $O(KN)$. At each iteration, we have obtained a feasible (but not necessarily minimum cost) k-unit flow. The sub-optimality lies in the fact that the above algorithm cannot adjust any previously instanced tracks based on the demand to produce additional tracks. In successive stages, it operates on a subset of the original graph rather than the residual graph used in the successive shortest paths algorithm. Unfortunately dynamic programming can't be directly applied to the residual graph $G_r(f)$ since the residual graph is no longer a DAG (Fig.4-(c)).

5.4. Approximate 2-pass DP solution for $K > 1$

We now describe generalization of our DP-based algorithm from 5.3 that can also instance new tracks while performing "small" edits of previously instanced track. We observe that most of the time the shortest residual path does not make large edits on previous tracks. We use the same algorithm from Section 5.1, but perform an approximate shortest-path using a 2-pass DP algorithm rather than Dijkstra's algorithm. We perform a forward pass of DP as in (10), but on $G_r(f)$ rather than G with $cost(i)$ defined as the best forward-progressing path from the source to node i (ignoring reversed edges). We then use the costs as initial values for a backward pass starting from the last frame, defining $N(i)$ to be the set of nodes connected through reverse edges to i. After this pass, $cost(i)$ is the cost of the best forward and backward progressing path ending at i. One could add additional passes, but we find experimentally that two passes are sufficient for good performance while saving $O(\log N)$ operations over Dijkstra's approach.

5.5. Caching

Our DP algorithms repeatedly perform computations on a series of reduced or residual graphs. Much of these computations can be cached. Consider the DP computations required for the algorithm from Section 5.3. Once a track is instanced, $cost(i)$ values for nodes whose shortest-paths intersect that track are no longer valid, and it is only this small number of nodes that need to be re-evaluated in the next iteration. This set can be marked using the following fact: any paths that intersect at some node must share the same birth node. Each node can be labeled with its birth node by propagating a birth ID during message-passing in DP. We then only need to recompute $cost(i)$ for nodes that have the same birth node as a newly instanced track. In our experiments, this decreased computation time by *three orders of magnitude*.

Figure 5. We show the results of our algorithm, including estimated track births and deaths, on the Caltech Pedestrian dataset [8]. We show typical results on the ETHMS dataset in Fig.1.

6. Experimental Results

Datasets: Most benchmarks for multi-object tracking (e.g., PETS [24]) are designed for stationary cameras. We are interested in moving camera applications, and so use the Caltech Pedestrian dataset [8] and ETHMS dataset [9] to evaluate our algorithms. The Caltech dataset was captured by a camera installed on a moving car. It contains 71 videos of roughly 1800 frames each, captured at 30 frames per second. Since the testset contains heldout labels, we evaluate ourselves using all annotated pedestrians on the training set. The ETHMS dataset contains footage of a busy sidewalk as seen by a camera mounted on a child stroller. Although this dataset contains both left and right views to facilitate stereo, we use only the left view in our experiments. The dataset contains four videos of roughly 1000 frames each, captured at 14 fps. Both datasets include bounding box annotations for people, while Caltech also provides track IDs. We manually annotated IDs on a portion of ETHMS. In order to compare our results with previous work, we use the same ETHMS video sequence as [25] with 999 frames and ignore detections smaller than 24 pixels as they did.

Setup: We ran an "out-of-the-box" pre-trained part-based HOG pedestrian detector [10] with a conservative NMS threshold, generating around 1000 detections per frame of each video. We set the log-likelihood ratio (the local cost c_i) of each detection to be the negative score of the linear detector (the distance from the decision boundary of an SVM). We use a bounded-velocity dynamic model: we define the transition cost c_{ij} to be 0, but only connect candidate windows across consecutive frames that spatially overlap. We set birth and death costs (c_i^s, c_i^t) to be 10. We experimented with applying an additional NMS step within our greedy algorithm. We also experimented with occlusion modeling by adding transitions which skip over k frames, with k up to 10.

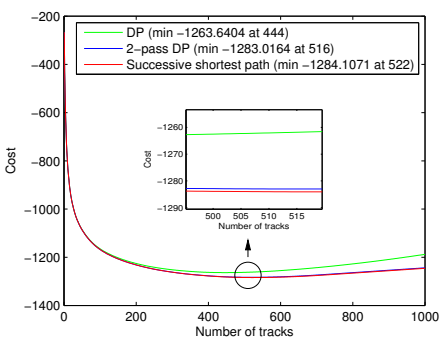

Figure 6. Cost vs. iteration number for all three algorithms on Caltech dataset. The inset shows that our 2-pass DP algorithm produces tracks whose cost is close to optimum while being orders of magnitude faster.

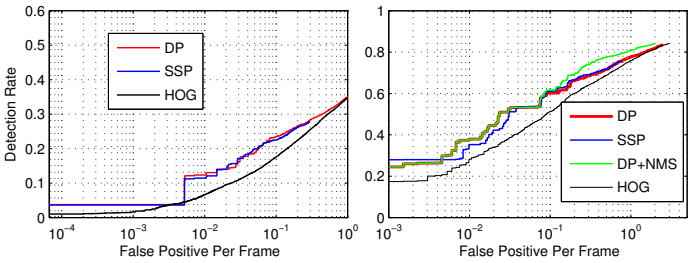

Figure 7. Detection rate versus FPPI on Caltech dataset [8] (**left**) and ETHMS dataset [9] (**right**). We compare our approximate 1-pass DP algorithm with the optimal successive shortest path (SSP) algorithm and a HOG-detector baseline. The DP performs as well as or even better than the shortest path algorithm, while being orders of magnitude faster. We also show that by suppressing overlapping detections after each track is instanced (DP-NMS), we can further improve performance.

Scoring criteria: We use detection accuracy (as measured by detection rate and false positives per frame) as our primary evaluation criteria, as it allows us to compare with a wide body of related work on these datasets. To directly score tracker accuracy, various other criteria (such as track fragmentation, identity switching, etc.) have been proposed [21, 20, 25]. We also use track identity to evaluate our algorithms below.

Approximation quality: We have described three different algorithms for solving the minimum cost flow problem. Figure 6 shows the flow cost, i.e., the objective function, versus iteration number for all three algorithms on the Caltech dataset. The DP algorithms follow the successive shortest path (SSP) algorithm for many iterations but eventually it is necessary to "edit" a previously instanced track (as in Figure 4) and the greedy DP algorithm begins to make suboptimal choices. However DP and SSP do not deviate much before reaching the minimum cost and the 2-pass DP which allows for a single edit follows SSP quite closely. This figure inset shows a close look at the cost at the minimum. Since the 2-pass algorithm can split at most one track in each iteration and it is very rare to see two splits at the same iteration, the cost value for 2-pass DP algorithm is very close to the optimum one.

Rather than scoring the cost function, we can directly compare algorithms using track accuracy. Figure 7 shows detection rate versus FPPI for the baseline detector, DP, and SSP algorithms. These figures show that DP and SSP are similar in accuracy, with DP performing even better in some cases. We suspect the SSP algorithm produces (overly) short tracks because the 1st order Markov model enforces a geometric distribution over track length. The approximate DP algorithm inadvertently produces longer tracks (that better match the ground truth distributions of lengths) because previously instanced tracks are never cut or edited. We henceforth evaluate our one-pass DP algorithm in the subsequent experiments. We also present additional diagnostic

Length of allowable occlusion	% of windows with ID errors
1	14.69
5	13.32
10	9.39

Table 1. Evaluating track label error as a function of the length of the allowable occlusion. We show results for our DP algorithm applied to a portion of the ETHMS dataset given ideal detected windows. Our DP algorithm scales linearly with the length of allowable occlusions. By allowing for longer occlusions (common in this dataset), the % of windows with correct track labels significantly increases.

experiments on the ETHMS data, since it contains on average more objects than Caltech.

Track identities: We evaluate track identities on the ETHMS dataset by using our tracker to compute track labels for ground-truth bounding boxes. This is equivalent to running our tracker on an ideal object detector with zero missed detections and false positives. Given a correspondence between estimated track labels and ground-truth track labels, the misclassification rate is the fraction of bounding boxes with incorrect labels. We compute the correspondence that minimizes this error by bipartite matching [15]. We found occlusion modeling to be crucial for maintaining track identities. Our algorithms can report tracks with k-frame occlusions by adding in transitions between space-time windows spaced k frames apart. Our DP algorithm scales linearly with k, and so we can readily model long 10-frame occlusions (Table 1). This greatly increases the accuracy of track labels on this data because such occlusions are common when nearby people pass the camera, occluding people further away. This result implies that, given ideal local detectors, our tracking algorithm produces track identities with 90% accuracy.

NMS-within-the-loop: In Figure 7, we use the ETHMS dataset to examine the effect of adding a NMS step within

Algorithm	Detection rate	False positive per frame
[9]'s stereo algorithm	47	1.5
[25]'s algorithm 1	68.3	0.85
[25]'s algorithm 2 with occlusion handling	70.4	0.97
[23]'s two-stage algorithm with occlusion handling	75.2	0.939
Our DP	76.6	0.85
Our DP+NMS	79.8	0.85

Table 2. Our algorithm performance compared to the previous state-of-the-art on the ETHMS dataset. Please see the text for further discussion.

our iterative greedy algorithms. When applying [10]'s pedestrian detector, we use their default NMS algorithm as a pre-process to suppress detections that overlap other higher-scoring detection by some threshold. After instancing a track during the DP algorithm, we suppress remaining windows that overlap the instanced tracks using a lower threshold. This suppression is more reliable than the initial one because tracked windows are more likely to be true positives. Our results outperform all previously published results on this data (Table 2).

Running time: For the 999-frame ETHMS dataset, MATLAB's LP solver does not converge, the commercial min-cost-flow solver used in [23] takes 95 seconds, while our MATLAB DP code takes 0.5 seconds.

7. Conclusion

We have described a family of efficient, greedy but globally optimal algorithms for solving the problem of multi-object tracking, including estimating the number of objects and their track births and deaths. Our algorithms are based on a novel analysis of a min-cost flow framework for tracking. Our greedy algorithms allow us to embed pre-processing steps such as NMS within our tracking algorithm. Our scalable algorithms also allow us to process large input sequences and model long occlusions, producing state-of-the-art results on benchmark datasets.

Acknowledgements: Funding for this research was provided by NSF Grants 0954083 and 0812428, and ONR-MURI Grant N00014-10-1-0933.

References

[1] R. Ahuja, T. Magnati, and J. Orlin. *Network flows: Theory, Algorithms, and Applications*. Prentice Hall, 2008.

[2] A. Andriyenko and K. Schindler. Globally optimal multi-target tracking on a hexagonal lattice. In *ECCV*, 2010.

[3] J. Berclaz, F. Fleuret, and P. Fua. Multiple object tracking using flow linear programming. In *Performance Evaluation of Tracking and Surveillance (PETS-Winter), 2009 Twelfth IEEE International Workshop on*, pages 1–8. IEEE, 2010.

[4] J. Berclaz, F. Fleuret, E. Türetken, and P. Fua. Multiple Object Tracking using K-Shortest Paths Optimization. *IEEE Transactions on PAMI*, Accepted for publication in 2011.

[5] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. *IEEE PAMI*, 2001.

[6] Y. Cai, N. de Freitas, and J. Little. Robust visual tracking for multiple targets. *Lecture Notes in Computer Science*, 3954:107, 2006.

[7] W. Choi and S. Savarese. Multiple target tracking in world coordinate with single, minimally calibrated camera. *ECCV 2010*, pages 553–567, 2010.

[8] P. Dollár, C. Wojek, B. Schiele, and P. Perona. Pedestrian detection: A benchmark. In *IEEE CVPR*, June 2009.

[9] A. Ess, B. Leibe, and L. Van Gool. Depth and appearance for mobile scene analysis. In *ICCV*, 2007.

[10] P. Felzenszwalb, D. McAllester, and D. Ramanan. A discriminatively trained, multiscale, deformable part model. *IEEE CVPR*, 2008.

[11] T. Fortmann, Y. Bar-Shalom, and M. Scheffe. Sonar tracking of multiple targets using joint probabilistic data association. *IEEE Journal of Oceanic Engineering*, 8(3):173–184, 1983.

[12] A. Goldberg. An efficient implementation of a scaling minimum-cost flow algorithm. *Journal of Algorithms*, 22(1):1–29, 1997.

[13] M. Isard and J. MacCormick. Bramble: A bayesian multiple-blob tracker. In *ICCV*, 2001.

[14] H. Jiang, S. Fels, and J. Little. A linear programming approach for multiple object tracking. In *IEEE CVPR*, 2007.

[15] H. Kuhn, P. Haas, I. Ilyas, G. Lohman, and V. Markl. The Hungarian method for the assignment problem. *Masthead*, 23(3):151–210, 1993.

[16] S. K. V. G. L. Leibe, B. Coupled detection and trajectory estimation for multi-object tracking. *ICCV 2007*.

[17] Y. Ma, Q. Yu, and I. Cohen. Target tracking with incomplete detection. *CVIU*, 2009.

[18] S. Pellegrini, A. Ess, and L. V. Gool. Improving data association by joint modeling of pedestrian trajectories and groupings. In *ECCV*, 2010.

[19] A. Perera, C. Srinivas, A. Hoogs, G. Brooksby, and W. Hu. Multi-object tracking through simultaneous long occlusions and split-merge conditions. In *IEEE CVPR*, volume 1, 2006.

[20] A. G. A. Perera, A. Hoogs, C. Srinivas, G. Brooksby, and W. Hu. Evaluation of algorithms for tracking multiple objects in video. In *AIPR*, page 35, 2006.

[21] K. Smith, D. Gatica-Perez, J. Odobez, and S. Ba. Evaluating multi-object tracking. In *CVPR Workshops*. IEEE, 2005.

[22] C. Stauffer. Estimating tracking sources and sinks. In *Proc. Event Mining Workshop*. Citeseer.

[23] J. Xing, H. Ai, and S. Lao. Multi-object tracking through occlusions by local tracklets filtering and global tracklets association with detection responses. In *IEEE CVPR*, June 2009.

[24] D. Young and J. Ferryman. Pets metrics: On-line performance evaluation service. In *Joint IEEE International Workshop on Visual Surveillance and Performance Evaluation of Tracking and Surveillance (VS-PETS)*, pages 317–324, 2005.

[25] L. Zhang, Y. Li, and R. Nevatia. Global data association for multi-object tracking using network flows. In *CVPR*, 2008.