Multiple Object Tracking using K-Shortest Paths Optimization

Jérôme Berclaz, François Fleuret, Engin Türetken, and Pascal Fua, *Senior Member, IEEE*

Abstract—Multi-object tracking can be achieved by detecting objects in individual frames and then linking detections across frames. Such an approach can be made very robust to the occasional detection failure: If an object is not detected in a frame but is in previous and following ones, a correct trajectory will nevertheless be produced. By contrast, a false-positive detection in a few frames will be ignored. However, when dealing with a multiple target problem, the linking step results in a difficult optimization problem in the space of all possible families of trajectories. This is usually dealt with by sampling or greedy search based on variants of Dynamic Programming, which can easily miss the global optimum. In this paper, we show that reformulating that step as a constrained flow optimization results in a convex problem. We take advantage of its particular structure to solve it using the k-shortest paths algorithm, which is very fast. This new approach is far simpler formally and algorithmically than existing techniques and lets us demonstrate excellent performance in two very different contexts.

Index Terms—Data association, Multi-object tracking, K-shortest paths, Linear programming

1 Introduction

Multi-object tracking can be decomposed into two separate steps that address independent issues. The first is time-independent detection, in which a prediction scheme infers the number and locations of targets from the available signal at every time step independently. It usually involves either a generative model of the signal given the target presence or a discriminative machine learning-based algorithm. The second step relies on modeling detection errors and target motions to link detections into the most likely trajectories.

In theory, at least, such an approach is very robust to the occasional detection failure. For example, false positives are often isolated in time and can readily be discarded. Similarly, if an object fails to be detected in a frame but is detected in previous and following ones, a correct trajectory should nevertheless be produced.

However, while it is easy to design a statistical trajectory model with all the necessary properties for good filtering, estimating the family of trajectories exhibiting maximum posterior probability is NP-Complete. This has been dealt with in the literature either by sampling and particle filtering [1], linking short tracks generated using Kalman filtering [2], or by greedy Dynamic Programming in which trajectories are estimated one after another [3]. While effective, none of these approaches guarantees a global optimum. A notable exception is a recent approach [4] that relies on Linear Programming [5] to find a global optimum with high probability, but at the cost of *a priori* specifying the number of objects being tracked and restricting the potential set of locations where objects can be found to those where the detector has fired. The former is restrictive while the latter is fine as long as the detector never produces false-negatives but may lead to erroneous trajectories in the more realistic case where it does.

By contrast, we show that reformulating the linking step as a constrained flow optimization results in a convex problem that fits into a standard Linear Programming framework. This formulation, however, yields a very large system that is hardly tractable using generic Linear Programming solvers. Therefore, we then demonstrate that, due to its particular structure, our problem can be solved very efficiently using the k-shortest paths algorithm [6], which yields real-time performance on realistically-sized problems. Our method does not present any of the limitations mentioned above, nor does it require an appearance model. The latter does of course not mean that one should not be used if available but making its use optional increases the range of applicability of our approach. Moreover, it is far simpler both formally and algorithmically than existing techniques and we will show that it performs well in two difficult real-world scenarios:

- Tracking multiple balls of similar color, which is a case where an appearance model would not help;
- Tracking multiple people with multiple cameras set at shoulder-level so that there are significant occlusions.

In both cases, we use an object detector that produces a *probabilistic occupancy map*, that is, a set of probabilities

- *J. Berclaz, E. Türetken and P. Fua are with the École Polytechnique Fédérale de Lausanne, CH-1015 Lausanne, Switzerland.*
  E-mail: {jerome.berclaz, engin.turetken, pascal.fua}@epfl.ch
- *F. Fleuret is with the Idiap Research Institute, CH-1920 Martigny, Switzerland, and the École Polytechnique Fédérale de Lausanne, CH-1015 Lausanne, Switzerland.*
  E-mail: francois.fleuret@idiap.ch

This work is supported by the Swiss National Science Foundation (SNSF) Sinergia project Aerial Crowds and by the SNSF under the National Centre of Competence in Research (NCCR) on Interactive Multimodal Information Management (IM2).

of presence of objects at a discrete set of locations at each time step independently. These probabilities may of course be noisy and inaccurate. Our only assumptions are that objects neither appear nor disappear anywhere but at specified entrances and exits, do not move too quickly, and cannot share a location with another object. These assumptions are minimal and generally applicable. We formulate the search for a map that obeys them, while being as close as possible to the original one, as a convex Linear Programming problem. Its solution is a set of flows that are both consistent and binary so that linking detections becomes trivial.

Our main contribution is two-fold: First, we introduce a generic and mathematically sound multiple object tracking framework, which only requires an occupancy map from a detector as input. Very few parameters need to be set and the algorithm handles unknown, and potentially changing, numbers of objects while naturally filtering out false positives and bridging gaps due to false negatives. Second, we demonstrate that this Linear Programming problem can be solved very effectively using the k-shortest paths algorithm [6].

2 Related Work

Multiple object tracking is an intensively studied area of research. A wide range of approaches relies on the recursive update of tracks with the most recent detections. For instance, Kalman filtering is an efficient way to address multi-target tracking [7], [8], [9], [10], [11] when the number of objects remains small. It is also well suited for real-time applications. However, when the number of objects increases, identity switches become more frequent and are difficult to correct, due to the recursive nature of the method. The work of [12], which tracks multiple humans using the mean-shift algorithm, also suffers from the same weakness.

Particle filtering can address some of the limitations of Kalman filtering by exploring multiple hypotheses [13], [1], [14], [15], [16], [17]. This technique has been used to great effect to follow multiple hockey players [18] or to track multiple people in the ground and image planes simultaneously [19]. In the same spirit, [20] relies on data-driven MCMC to recover trajectories of targets using a batch of observations. [21] applies a Probability Hypothesis Density filter to tracking multiple objects from noisy observations, and therefore falls into this family of algorithms. Despite their success, in our experience, those sampling-based methods typically require careful tuning of several meta-parameters, which reduces the generality of systems that rely on it. Besides, they usually look at small time windows, because their state space grows exponentially with the number of frames.

In an attempt to increase tracking reliability, some methods rely on a hybrid approach. Detections are first connected into short tracks, which are then linked together using a higher-level method. For example, [2] relies on Kalman filtering to obtain basic tracks, and then tries to merge and split the tracks using the Hungarian algorithm. [22] explores the hierarchical version of the same concept, while [23] uses a variant of AdaBoost to automatically learn the best criterion for linking low-level tracks together. Similarly, [24] turns observations into trajectory segments using local PCA, and then links those segments based on their spatial proximity and smoothness constraints. [25] relies on mean-shift or particle filtering to generate tracklets from detection results. In a second stage, it uses MCMC data association to combine the tracklets into full tracks, and to automatically estimate the best parameters for the model. [26] uses a motion model and nearest neighbor to build tracks out of heads detected from a top mounted calibrated camera. The tracks thus generated are then merged and split into the final trajectories using heuristics based on overlap, directions and speed. [27] proposes another method to tracklet generation in a crowded environment, without however going all the way to combining them into complete tracks. It detects multiple people and creates tracklets by applying Bayesian clustering on simple tracked image features. By contrast, [28] concentrates on the high level task. The authors assume that a track graph has already been produced and focus on linking identities in the provided track graph. They formulate the multi-object tracking as a Bayesian network inference problem and apply this method to tracking multiple soccer players.

This class of methods is a good compromise: The 2-stage architecture allows them to scale efficiently, while at the same time taking into account a wider observation window. However, while exhibiting good results in some situations, those methods rely on an ad-hoc mathematical formulation, which does not guarantee convergence to a global optimum. They are therefore prone to mistakes such as identity switches. To improve robustness to wrong identity assignment, research has recently focused on linking detections over a larger time window using various optimization schemes. For example, [29] applies graph cuts to extract trajectories from a batch of people detections obtained using homographic constraints on images from multiple cameras. [30] simultaneously optimizes detections and tracks, coupled into a Quadratic Boolean Problem and solved by an EM algorithm.

Dynamic Programming [31] can be used to link multiple detections over time, and therefore solve the multi-target tracking problem. Moreover, it can be extended to enable the optimization of several trajectories simultaneously [32]. Unfortunately, the computational complexity of such an approach can be prohibitive. While efficient for very small state-space, it does not scale to the size of problems we generally deal with. To overcome this limitation, in earlier work [3] we sequentially applied Dynamic Programming over individual trajectories, which were assumed independent. While this approach greatly reduces the optimization cost, it tends to mix trajectories when the targets are densely located. It is also quite sensitive to false negatives and exhibits a tendency to ignore trajectories when the detection information is not good enough. A different formulation is chosen by [33], where a directed graph, with nodes standing for actual detections, represents the multi-frame point correspondence problem. A greedy optimization algorithm is introduced to efficiently solve the problem, but without a guarantee to find a global optimum.

By contrast, Linear Programming is an optimization method that has been applied to find global optima and solve the data association problem for airplane radar tracking [34] or for multiple people tracking [4]. Starting from the output of simple object detectors, this last approach builds a network graph in which every node is an observation fully connected to future and past observations, in much the same way as in [33]. Occlusions among objects are modeled by specifying spatial conflicts between nodes. Additional nodes are created to specifically handle occluded objects. Finally, arc costs are chosen according to object appearances and a motion model, and soft constraints are introduced to ensure spatial layout consistency. A relatively similar graphical model, with nodes representing detections, is built by [35] for multi-people tracking. The global optimum is searched using a min-cost flow algorithm, which exploits the specific structure of the graph to reach the optimum faster than Linear Programming.

Due to their reduced state-space, these methods are computationally efficient. However, [4] requires *a priori* knowledge of the number of objects to be tracked, which seriously limits its applicability in real life situations. Also, with a state-space only consisting of observations, as opposed to all possible locations as in our approach, they cannot smoothly interpolate trajectories when there are false negatives. Finally, the choice of arc costs is ad-hoc and involves many parameters, which have to be tuned for each possible application, reducing the generality of the methods. By contrast, our model is far simpler, with the neighborhood size being the only value that needs to be adapted. We optimize over the entire space of locations for fine trajectory interpolation, and deal with the large resulting size of our problem by trading standard Linear Programming optimization for a very efficient formulation based on the k-shortest paths algorithm.

3 ALGORITHM

In this section, we first formulate multi-target tracking as an Integer Programming (IP) problem. Although such a problem is NP-hard in many cases, we show that a relaxation of it as a Linear Program yields the optimal solution, and hence the problem is solvable in polynomial time. Despite our simple and clean formulation, the large number of variables and constraints makes it only tractable for small areas and short sequences. Thus, in a second step, we demonstrate how the k-shortest paths algorithm can be used to solve this problem much more efficiently than generic Linear Programming solvers can.

TABLE 1
Notation

K	number of spatial locations;
T	number of time steps;
$\mathbf{I}$	$= (\mathbf{I}^1, \ldots, \mathbf{I}^T)$ captured images;
$\mathcal{N}(k)$	$\subset \{1, \ldots, K\}$ neighborhood of location k;
$e^t_{i,j}$	directed edge from location i at time t to location j at time $t+1$;
$f^t_{i,j}$	estimated number of objects moving from location i at time t to location j at time $t+1$;
m^t_i	estimated number of objects at location i at time t;
M^t_i	random variable standing for the true number of objects at location i at time t;
$\mathfrak{F}$	set of occupancy maps physically possible;
$\mathfrak{H}$	set of flows physically possible, i.e. satisfying the constraints of Eqs. 1, 2, 3, and 4.

3.1 Formalization

We discretize the physical area of interest into K locations, and the time interval into T instants. For any location k, let $\mathcal{N}(k) \subset \{1, \ldots, K\}$ denote the neighborhood of k, that is, the locations an object located at k at time t can reach at time $t+1$.

To model occupancy over time, let us consider a labeled directed graph with $K\,T$ vertices, which represents every location at every instant. Its edges correspond to admissible object motions, which means that there is one edge $e^t_{i,j}$ from (t,i) to $(t+1,j)$ if, and only if, $j \in \mathcal{N}(i)$. To allow objects to remain static, there is always an edge from a location at time t to itself at time $t+1$.

Each vertex is labeled with a discrete variable m^t_i standing for the number of objects located at i at time t. Each edge is labeled with a discrete variable $f^t_{i,j}$ standing for the number of objects moving from location i at time t to location j at time $t+1$, as shown in Fig. 1(a). For instance, the fact that an object remains at location i between times t and $t+1$ is represented by $f^t_{i,i} = 1$.

Given these definitions, for all t, the sum of flows arriving at any location j is equal to m^t_j, which also is the sum of outgoing flows from location j at time t. We must therefore have

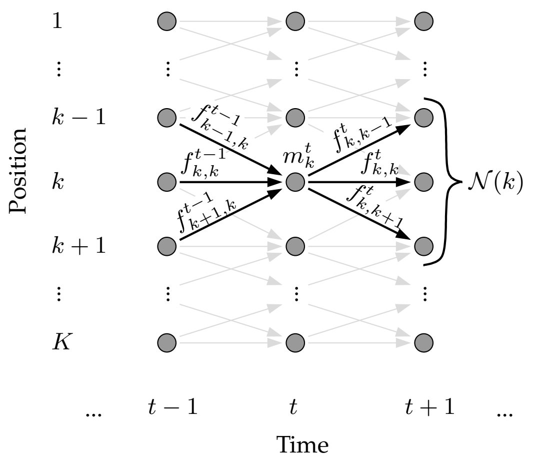
$$\forall t, j, \ \underbrace{\sum_{i:j\in\mathcal{N}(i)} f^{t-1}_{i,j}}_{\text{Arriving at } j \text{ at } t} = m^t_j = \underbrace{\sum_{k\in\mathcal{N}(j)} f^t_{j,k}}_{\text{Leaving from } j \text{ at } t} \ . \tag{1}$$

Furthermore, since a location cannot be occupied by more than one object at a time, we can set an upper-bound of 1 to the sum of all outgoing flows from a given location and impose

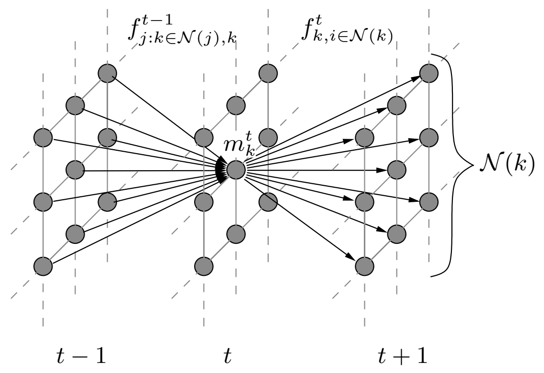
$$\forall k, t, \ \sum_{j\in\mathcal{N}(k)} f^t_{k,j} \leq 1 \, . \tag{2}$$

A similar constraint applies to the incoming flows, but we do not need to explicitly state it, since it is implicitly enforced by Eq. 1. Finally, the flows have to be non-negative and we have

$$\forall k, j, t, \ f^t_{k,j} \geq 0 \, . \tag{3}$$



(a) Simplified flow model



$f^{t-1}_{j:k\in\mathcal{N}(j),k}$
$f^t_{k,i\in\mathcal{N}(k)}$
m^t_k
$\mathcal{N}(k)$
$t-1$ t $t+1$

(b) Grid flow model

Fig. 1. (a) Simplified flow model, which does not use virtual positions. Positions are arranged on one dimension and neighborhood is reduced to 3 positions. (b) Flow model used for tracking objects moving on a 2D grid, such as in pedestrian tracking. For the sake of readability, only the flows to and from location k at time t are printed.

In general, the number of tracked objects may vary over time, meaning that objects may appear inside the tracking area and others may leave. Thus, the total mass of the system changes and we must allow flows to enter and exit the area.

We do this by introducing two additional nodes υ_{source} and υ_{sink} into our graph, which are linked to all the nodes representing positions through which objects can respectively enter or exit the area, such as doors or borders of the camera field of view. In addition, a flow goes from υ_{source} to all the nodes of the first frame, and reciprocally a flow goes from all the nodes of the last frame to υ_{sink}. We call υ_{source} and υ_{sink} *virtual locations*, because, as opposed to the other nodes of the graph, they do not represent any physical location. The resulting complete graph is shown in Fig. 2.

Finally, we introduce an additional constraint that ensures that all flows departing from υ_{source} eventually

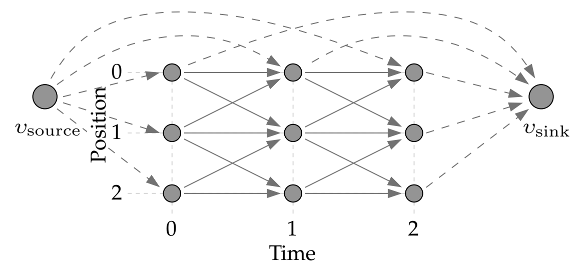


Fig. 2. A complete flow system for a simple graph consisting only of 3 positions and 3 time frames. Here, we assume that position 0 is connected to the virtual positions and therefore a possible entrance and exit point. Flows to and from the virtual positions are shown as dashed lines.

end up in υ_{sink}

$$\underbrace{\sum_{j\in\mathcal{N}(\upsilon_{\text{source}})} f^t_{\upsilon_{\text{source}},j}}_{\text{Leaving } \upsilon_{\text{source}}} = \underbrace{\sum_{k:\upsilon_{\text{sink}}\in\mathcal{N}(k)} f^t_{k,\upsilon_{\text{sink}}}}_{\text{Arriving at } \upsilon_{\text{sink}}} . \tag{4}$$

Let M^t_i denote a random variable standing for the true presence of an object at location i at time t. The object detector used to process the sequence provides, for every location i and every instant t, an estimate of the marginal posterior probability of the presence of an object

$$\rho^t_i = \hat{P}(M^t_i = 1\,|\,\mathbf{I}^t)\,, \tag{5}$$

where $\mathbf{I}^t$ is the signal available at time t. For the multi-camera pedestrian-tracking application described in § 4, $\mathbf{I}^t$ denotes the series of pictures taken by all the cameras at time t.

Let $\mathbf{m}$ be an occupancy map, that is a set of occupancy variables m^t_i, one for each location and for each instant. We say that $\mathbf{m}$ is *feasible* if there exists a set of flows $f^t_{k,j}$ that satisfies Eqs. 1, 2, 3, and 4, and we define $\mathfrak{F}$ the set of feasible maps. Our goal then becomes solving

$$\mathbf{m}^* = \arg\max_{\mathbf{m}\in\mathfrak{F}} \hat{P}(\mathbf{M} = \mathbf{m}\,|\,\mathbf{I}) \ . \tag{6}$$

Assuming conditional independence of the M^t_i, given the $\mathbf{I}^t$, the optimization problem of Eq. 6 can be re-written as

$$\begin{aligned}
\mathbf{m}^* &= \arg\max_{\mathbf{m}\in\mathfrak{F}} \log \prod_{t,i} \hat{P}(M^t_i = m^t_i\,|\,\mathbf{I}^t) && (7)\\
&= \arg\max_{\mathbf{m}\in\mathfrak{F}} \sum_{t,i} \log \hat{P}(M^t_i = m^t_i\,|\,\mathbf{I}^t) \\
&= \arg\max_{\mathbf{m}\in\mathfrak{F}} \sum_{t,i} (1-m^t_i) \log \hat{P}(M^t_i = 0\,|\,\mathbf{I}^t) \\
&\qquad\qquad + m^t_i \log \hat{P}(M^t_i = 1\,|\,\mathbf{I}^t) && (8)\\
&= \arg\max_{\mathbf{m}\in\mathfrak{F}} \sum_{t,i} m^t_i \log \frac{\hat{P}(M^t_i = 1\,|\,\mathbf{I}^t)}{\hat{P}(M^t_i = 0\,|\,\mathbf{I}^t)} && (9)\\
&= \arg\max_{\mathbf{m}\in\mathfrak{F}} \sum_{t,i} \left(\log \frac{\rho^t_i}{1-\rho^t_i}\right) m^t_i\,, && (10)
\end{aligned}$$

where Eq. 7 is true under the assumption of conditional independence of the M^t_i given $\mathbf{I}^t$, Eq. 8 is true because

m_i^t is 0 or 1 according to Eq. 2, and Eq. 9 is obtained by ignoring a term which does not depend on $\mathbf{m}$. Hence, the objective function of Eq. 10 is a linear expression of the m_i^t.

3.2 Linear Programming Formulation

The formulation defined above translates naturally into the Integer Program

$$\begin{aligned}
\text{Maximize} \quad & \sum_{t,i} \log\left(\frac{\rho_i^t}{1-\rho_i^t}\right) \sum_{j\in\mathcal{N}(i)} f_{i,j}^t \\
\text{subject to} \quad & \forall t,i,j,\ \ f_{i,j}^t \geq 0 \\
& \forall t,i, \quad \sum_{j\in\mathcal{N}(i)} f_{i,j}^t \leq 1 \\
& \forall t,i, \quad \sum_{j\in\mathcal{N}(i)} f_{i,j}^t - \sum_{k:i\in\mathcal{N}(k)} f_{k,i}^{t-1} \leq 0 \\
& \sum_{j\in\mathcal{N}(\upsilon_{\text{source}})} f_{\upsilon_{\text{source}},j} - \sum_{k:\upsilon_{\text{sink}}\in\mathcal{N}(k)} f_{k,\upsilon_{\text{sink}}} \leq 0\,.
\end{aligned} \tag{11}$$

In this system, the optimization is carried out with respect to the flows $f_{i,j}^t$ rather than the occupancies m_i^t, because there is no natural way to express the flow continuity constraints in terms of the latter. This is equivalent to maximizing the objective function of Eq. 10 because $\forall t,j, m_j^t = \sum_{k\in\mathcal{N}(j)} f_{j,k}^t$.

Note that the constraints of Eqs. 1, 2, 3, and 4 are expressed as inequalities, to have the linear program in *canonical form*. This new formulation is strictly equivalent to the original one and no additional constraint is needed. The inequalities are indeed sufficient to ensure that no flow can ever appear or disappear within the graph.

Under this formulation, our Integer Program can be solved by any generic LP solver. However, due to the very large size of our problem, this solution would hardly be practical, as IP solving is NP-complete. The usual workaround is to relax the integer assumption and solve a continuous Linear Program instead, which has polynomial-time average-case complexity. The drawback of this method is that the Linear Program is unlikely to converge to the optimal solution of the original IP.

In our case, however, the relaxed Linear Program always converges towards an integer solution because its constraint matrix exhibits a property known as *total unimodularity*, as will be shown in Appendix A. As a consequence, we could use a generic LP solver to optimize our multi-target tracking framework. However, this approach would only be tractable for moderately sized problems, and does not scale to most practical applications. Therefore, in the next section, we introduce a more efficient optimization scheme, which takes into account the specificity of our problem to tremendously reduce the complexity.

3.3 K-Shortest Paths Formulation

The relaxation of the original integer problem yields a large scale LP problem, which can be solved by generic LP solvers, that, in general, rely on variants of the Simplex algorithm [5] or interior point based methods [36]. However, these algorithms do not make use of the specific structure of our problem and have very high worst case time complexities. In the following, we show that this complexity can be drastically reduced by reformulating the problem as a *k shortest node-disjoint paths* problem on a directed acyclic graph (DAG).

Given a pair of nodes, namely the source υ_{source} and the sink υ_{sink}, in a graph G, the k-shortest paths problem is to find the k paths $\{p_1,\ldots,p_k\}$ between these nodes, such that the total cost of the paths is minimum. The problem is well-studied in the network optimization literature and the results have been widely applied in the field of network connection routing and restoration. There exists many variants of the algorithm, each targeted at a specific problem instance [1].

In our specific case, we are interested in the particular instance where the graph is directed and paths are both node-disjoint - i.e. two separate paths cannot share the same node - and node-simple - i.e. a path visits every node in the graph at most once. We use the graph structure with a single source and a single sink illustrated by Fig. 2. Any path between υ_{source} and υ_{sink} in this graph represents the flow of a single object in the original problem along the edges of the path. The node-disjointness constraint means that no location can be shared between two paths, hence two objects. This is thus equivalent to the constraint of Eq. 2. Moreover, by only looking for paths between the source and sink nodes, we ensure that no flow can ever be created nor suppressed anywhere in the graph but at the virtual locations. This enforces the constraints of Eqs. 1 and 4. Finally, the node-simple characteristic of the paths simply stems from the fact that our graph is a DAG, hence acyclic.

A directed edge $e_{i,j}^t$ from location i at time t to location j at time $t+1$ is assigned the cost value

$$c(e_{i,j}^t) = -\log\left(\frac{\rho_i^t}{1-\rho_i^t}\right)\,. \tag{12}$$

The cost value of the edges emanating from the source node is set to zero to allow objects to appear at any entrance position and at any time instant at no cost. We formulate our problem as a minimization problem by negating the objective function of Eq. 11.

Let $\mathfrak{H}$ denote the set of feasible solutions of the original LP formulation of Eq. 11, satisfying the constraints given in Eq. 1, 2, 3, and 4. Then, the optimal solution $\mathbf{f}^*$ of the k-shortest paths problem can be written as

$$\mathbf{f}^* = \arg\min_{\mathbf{f}\in\mathfrak{H}} \sum_{t,i} c(e_{i,j}^t) \sum_{j\in\mathcal{N}(i)} f_{i,j}^t \quad , \tag{13}$$

where $c(e_{i,j}^t)$ represents the cost of the edge $e_{i,j}^t$ defined in Eq. 12. Note that any node-disjoint k paths between

1. for a complete list of references, see the online bibliography at http://liinwww.ira.uka.de/bibliography/Theory/k-path.html

υ_{source} and υ_{sink} with arbitrary k is in the feasible set of solutions $\mathfrak{H}$. In addition, any solution in $\mathfrak{H}$ can be expressed as a set of k node-disjoint paths.

Let p_i^* be the shortest path computed at the i^{th} iteration of the algorithm and $P_l = \{p_1^*, \ldots, p_l^*\}$ be the set of all l shortest paths computed up to iteration l. We start by finding the single shortest path in the graph p_1^* and compute its total cost

$$\text{cost}(p_l^*) = \sum_{e_{i,j}^t \in p_l^*} c(e_{i,j}^t)\,. \tag{14}$$

We then compute iteratively the l-shortest paths for $l = 2, 3, 4, \ldots$, and for each l, we calculate the total cost of the shortest paths

$$\text{cost}(P_l) = \sum_{i=1}^{l} \text{cost}(p_i^*)\,. \tag{15}$$

At each new iteration $l+1$, the total cost $\text{cost}(P_{l+1})$ is compared to the cost at the previous iteration $\text{cost}(P_l)$. The optimal number of paths k^* is obtained when the cost of iteration k^*+1 is higher than the one of iteration k^*. The procedure is summarized by the pseudo-code of Algorithm 1, in Appendix B.

To compute such k-shortest paths, we use the *disjoint paths* algorithm [6], which is an efficient iterative method based on signed paths. For the sake of completeness, we give a brief description of this algorithm in Appendix B.

The equivalence of the LP and the k-shortest paths formulations follows from the exact procedure we use to select an optimal k such that the objective function is minimized. Since path costs are monotonically increasing

$$\text{cost}(p_{i+1}^*) \geq \text{cost}(p_i^*) \qquad \forall i\ , \tag{16}$$

the total cost function $\text{cost}(P_l)$ is convex with respect to l. Therefore, the global minimum is reached when $\text{cost}(p_i^*)$ changes sign and becomes non-negative

$$\text{cost}(P_{k^*-1}) \geq \text{cost}(P_{k^*}) \leq \text{cost}(P_{k^*+1})\ . \tag{17}$$

This is set as the stopping criterion of the algorithm, as presented in Algorithm 1. Finally, among the set of all consecutive values that may satisfy the above condition, we select the smallest one to avoid erroneous splitting of paths.

As discussed in Appendix B, the worst case complexity of the algorithm is $O(k(m + n \cdot \log n))$, where k is the number of objects appearing in a given time interval, m is the number of edges and n the number of nodes in the final transformed graph. This is more efficient than the min-cost flow method of [35], which exhibits a worst case complexity of $O(kn^2 m \log n)$. Furthermore, due to the mostly acyclic nature of our graph, the average complexity is almost linear with the number of nodes, which is reflected by our experimental results in Fig. 7(b). This is much faster than general LP solvers, and a speed gain of up to a factor 1,000 can be expected, as illustrated by the run time comparison in §4.10.

3.4 Batch Processing and Complexity Reduction

Processing a whole video sequence is possible but impractical for applications such as broadcasting, in which the result must be supplied quickly. When dealing with such cases, we split the sequence in batches of 100 frames, which yields good results and can be done in real-time. This results in a constant 4-second delay between input and output, which is nevertheless compatible with many applications.

To enforce temporal consistency across batches, we add the last frame of the previously optimized batch to the current one. We then force the flows out of every location of this frame to sum up to the location's value in the previous batch

$$\forall k \in \{1, \ldots, K\}, \sum_{j \in \mathcal{N}(k)} f_{k,j}^{-1} = \mu_k, \tag{18}$$

where μ_k is the score at location k of the last frame of the previous batch and $f_{k,j}^{-1}$ is a flow from location k of the last frame of the previous batch to location j in the first frame of the current batch. This is implemented as an additional constraint in our framework.

Further reducing the system's size might be needed for extremely large problems, to limit their time and memory consumption. It could be achieved by pruning the detection graph. Since most of the probabilities of presence estimated by the detector are virtually equal to zero, we can use this sparsity to reduce the number of nodes to consider in the optimization, thus reducing the computational cost.

Formally, for every position k and every frame t, we check the maximum detection probability within a given spatio-temporal neighborhood

$$\max_{\substack{\|j-k\|<\tau_1 \\ t-\tau_2<u<t+\tau_2}} \rho_j^u\ . \tag{19}$$

If this value is below a threshold, the location is considered not reachable by an object with any reasonable level of probability. All flows to and from it are then removed from the model. Applying this method would reduce the number of variables and constraints up to an order of magnitude. In the examples presented in this paper, we have not found it necessary to do so.

3.5 Algorithm Output

Estimating the $f_{i,j}^t$ indirectly provides the m_i^t values and the feasible occupancy map $\mathbf{m}^*$ of maximum posterior probability. This data can be used as a cleaned up version of the original occupancy map, in which most false positives and negatives have been filtered out. However, the $f_{i,j}^t$ themselves provide, in addition to the instantaneous occupancy, estimates of the actual motions of objects. From these estimated flows of objects, one can follow the motion back in time by moving along the edges whose $f_{i,j}^t$ are not 0, and build the corresponding long trajectories.

4 Results

In this section, we present results in two very different contexts. First, we use a multi-camera setup in which the cameras are located at shoulder level to track pedestrians who may walk in front of each other. The frequent occlusions between people produce noisy detections, which our algorithm nevertheless links very reliably. As a result, our approach was shown to compare favorably against other state-of-the-art algorithms in the PETS 2009 evaluation [37]. Second, to highlight the fact that we do not depend on an appearance model, we track sets of similar-looking bouncing balls seen from above. In both cases, we compare our results to those of our earlier tracking method based on sequential Dynamic Programming [3], and show that we can obtain good results even when using a single camera.

4.1 PETS 2009 Evaluation

The results of our approach on the PETS 2009 S2/L1 multi-camera sequence have been submitted to the Winter-PETS 2009 workshop. The results of this comparative evaluation are presented in [37] and illustrated by Fig. 3. They show that, for the tracking task, our current approach outperforms the other submitted methods. Furthermore, our earlier Dynamic Programming based approach [3] is also shown to perform well. In the remainder of this section, we thus use this previous approach as the baseline, and extensively compare our new algorithm against it.

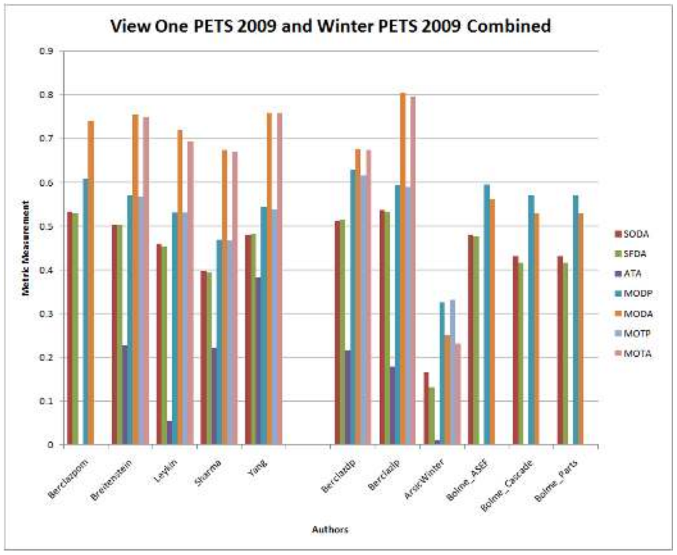


Fig. 3. Winter-PETS 2009 [37] results comparison chart, showing the performance of various tracking methods on the S2/L1 sequence. The results are evaluated with the CLEAR (MODA, MODP, MOTA and MOTP, described below) and VACE metrics [38] (SODA, SFDA and ATA). In the chart, our algorithm is referred to as 'Berclazlp' and our earlier approach based on dynamic programming as 'Berclazdp'. Figure courtesy of James Ferryman and Ali Shahrokni from the University of Reading.

4.2 Test Data

Our main data set consists of a series of multi-camera video sequences of pedestrians. The chosen locations for data acquisition include various environments: a crowded outdoor terrace, an indoor basketball court as well as a very difficult dark underground passageway. Additionally, we also perform our own detailed evaluation on a sequence from the PETS 2009 [39] data set.

4.2.1 *Laboratory sequence*

This 3 1/2-minute outdoor sequence consists of up to 9 people appearing one after the other, and walking in front of the cameras. It tests the ability of our algorithm to cope with a moderately crowded environment.

4.2.2 *Basketball sequence*

This sequence involves 10 basketball players in a game on half a basketball court. It is a difficult sequence with fast moving people and many occlusions.

4.2.3 *Passageway sequences*

These sequences involve several people passing through a public underground passageway. They are very challenging for several reasons. First, lighting conditions are very poor, which is typical in real-world surveillance situations. A large portion of the images is either underexposed or saturated. Second, the area covered by the system is large, and people get very small on the far end of the scene, making their precise localization challenging. Finally, large parts of the scene are filmed by only two or even a single camera.

All these difficulties greatly affect the quality of the probabilistic occupancy maps we use as input and the detection maps can get very noisy, with some people poorly localized or ignored over significant numbers of frames. On these noisy sequences, if we were to detect people by simply thresholding the maps in individual frames, the true positive rate would drop to 70% to 80%, thus making the linking task challenging.

4.2.4 *PETS 2009 sequence*

This sequence was filmed at a road corner of a university campus and involves about 10 people. Important light changes between the background model and the sequence, as well as precision issues in camera calibration make the people detection results noisy. Moreover, the sequence was acquired at a low frame rate of 7 fps, which creates an additional difficulty, since people can move significantly from one frame to the next.

4.2.5 *Camera setup*

In the first three pedestrian environments, the scene was filmed by 4 DV cameras with overlapping fields of view, each of which placed in a corner of the monitored area. The video format is DV PAL, down-sampled to 360×288 pixels at 25 fps and the 4 video streams were synchronized manually after data acquisition. The dimensions of the four areas are summarized by Table 2.

TABLE 2
Dimensions of the areas used for pedestrian tracking.

Scene	dimensions	grid size	locations
Laboratory	7m×10m	30×45	1,350
Basketball	15m×14m	47×50	2,350
Passageway	12m×30m	40×100	4,000
PETS 09	18.5m×20m	56×61	3,416

The PETS 2009 sequence was filmed by seven cameras: three dedicated video surveillance cameras and four DV cameras. The DV cameras were placed at about 2 meters above the ground, whereas the video surveillance cameras were located between 3 to 5 meters above it, and significantly farther from the scene. The frame rate for all cameras was set to 7 fps. Due to calibration imprecision, only five out of the seven available camera views were used for people detection.

Our second data set consists of two video sequences in which 24 table tennis balls were thrown on the ground. Those were filmed by a single DV camera, placed about 1.5m above the ground. The videos were cropped to a resolution of 600×400 pixels, and the corresponding area was discretized into a grid of 60×40 locations.

4.3 Probabilistic Occupancy Map

We used the publicly available implementation [40] of our earlier Probabilistic Occupancy Map (POM) algorithm [3] to create the detection data needed as input by our tracker.

This method first performs binary background/foreground segmentation in all images taken at the same time and then uses a generative model to estimate the most likely locations of targets given the observed foreground regions. More precisely, it relies on a decomposition of the space of possible object positions into a discrete grid. Then, at every time frame t, and for every location i of the grid, it produces an estimate ρ_i^t of the marginal posterior probability of presence of a target at that location, given all input images captured at that instant. POM specifically estimates the ρ_i^t such that the resulting product law closely approximates the joint posterior distribution, which justifies the assumption of conditional independence in Eq. 7.

In the multi-camera setup for which POM was designed, the grid of positions models the ground plane on which people walk, and is made of square elements of typically 30 cm × 30 cm. Correspondence between camera and top views is ensured through camera calibration. The generative model at the heart of POM represents people as cylinders that project to rectangles in the images.

Note that, in our model, the resolution of the ground grid is independent of the target's size. If grid cells are smaller than a target, the spatial precision of detections is improved at the cost of increased computation time, but the detections do not spread over several cells. In effect, POM implicitly performs a non-maximum suppression: The best fitted position receives a high probability while surrounding locations are considered as empty. As a result, POM occupancy maps are normally peaky, which is what our linking algorithm expects.

To process the monocular sequence of bouncing balls, we slightly modified the original POM code to represent the balls as squares and work directly in the top view, without having to project from oblique images into it.

4.4 Baseline

We compare our new algorithm to our previous sequential Dynamic Programming approach [3]. It involves estimating likely trajectories one after another in a greedy way using a standard Dynamic Programming procedure. The most likely trajectories are selected first and, once a trajectory has been found, the corresponding locations are removed from consideration. Note that the results reported in [3] were obtained with both a motion and an appearance model, while our results rely only on the very weak motion model implied by the graph's connectivity. For a fair comparison against our new approach, the method was applied both with and without its appearance model in the evaluation of Figs. 4 and 5. As expected, the appearance model slightly improves Dynamic Programming's results. In the rest of this section we refer to our Linear Programming framework solved using the k-shortest paths algorithm as 'KSP' and to the sequential Dynamic Programming as 'DP'.

4.5 Evaluation Metrics

To quantify our results, we manually labeled some of the test sequences. We marked both the position of the people and their identity, to be able to detect identity switches. The same process was used to label the ball sequences. Our ground truth data therefore consists of:

- two ball sequences of approximately 1,000 frames each, labeled once every 3 frames;
- the 800-frame long PETS 2009 sequence S2/L1, labeled once every 5 frames;
- 4 video sequences from the passageway data set, measuring respectively 2,500, 800, 900 and 800 frames, and labeled once every 25 frames;
- the laboratory sequence of 5,000 frames, labeled once every 25 frames.

Our results are evaluated using the standard CLEAR metrics: *Multiple Object Detection Accuracy* and *Precision* (MODA and MODP), as well as *Multiple Object Tracking Accuracy* and *Precision* (MOTA and MOTP). The detection precision metric (MODP) roughly gauges the quality of the bounding box alignment, in case of correct detection, while its accuracy counterpart (MODA) evaluates the relative number of false positives and missed detections. The tracking metrics also take the identity of detections into account: MOTP evaluates the alignment of tracks with the ground truth, and MOTA produces a score based on the amount of false positives, missed detections, and identity switches. These metrics have become standard for evaluation of detection and tracking algorithms, and we refer the interested reader to [41], [38] for more detailed descriptions and motivations.

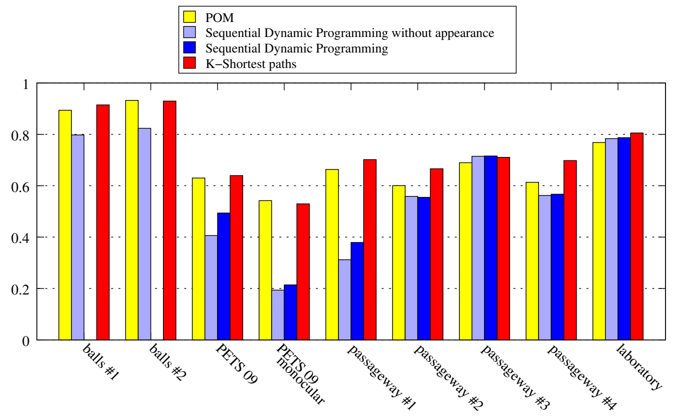
4.6 Pedestrian Tracking Results

For pedestrians tracking, we define the graph of Fig. 1(a) as follows: Every interior location of the ground plane at time t is linked to its 9 direct neighbors at time $t+1$, as illustrated by Fig. 1(b), which means that a pedestrian can only move from one location to its immediate neighbors in a consecutive frame. Border locations through which access to the area is possible are connected to the virtual locations υ_{source} and υ_{sink}. This arrangement is consistent with our chosen grid quantization at 25 fps. It even suits the 7 fps PETS 2009 sequence, since the pedestrians do not move fast. To deal with an even lower frame rate, or faster moving objects, we could extend the neighborhood size, as we do in §4.8 to track table tennis balls. Detection results for all evaluated sequences are shown on Fig. 4, and tracking results on Fig. 5.

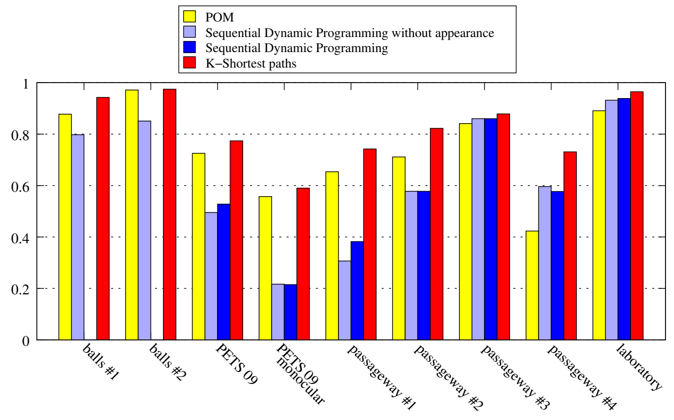
Since both DP and KSP link POM detections together, their precision score rarely exceeds the one of POM itself, although it may happen that the interpolation performed by the trackers corrects some misalignment of POM, such as in the *laboratory* sequence of Fig. 4.a. However, in both detection and tracking *precision*, KSP almost always scores significantly higher than DP.

Detection accuracy varies across sequences, depending on their difficulty. The *passageway* sequences, for instance, have lower accuracy than other sequences, due to their poor image quality. So does the monocular PETS sequence, because of the lack of precision resulting from the use of a single camera. Despite these variations, KSP's detection accuracy is always higher than POM. By accurately linking detections together, while discarding isolated alarms, KSP efficiently filters the detections results, thus decreasing both false positive and missed detection counts. On the other hand, DP's accuracy is often lower than POM, because of its tendency to ignore trajectories with missing detections. Note how the gap between KSP and DP generally widens as POM score gets lower, on Figs. 4 and 5. Both tracking methods deal efficiently with good detection results, but KSP proves much more robust than DP when the detection quality gets worse.

Recall that the occupancy maps are KSP's only input. The algorithm does not use any other source of information, such as the original images. Conversely, DP maintains a color model per tracked individual learned from images, in addition to the occupancy maps. In other words, KSP produces better results, even though it requires less information. This is valuable, because, in some situations such as the ball tracking presented below, appearance models cannot be depended upon. Typical tracking results are illustrated by Fig. 8.



(a) MODP



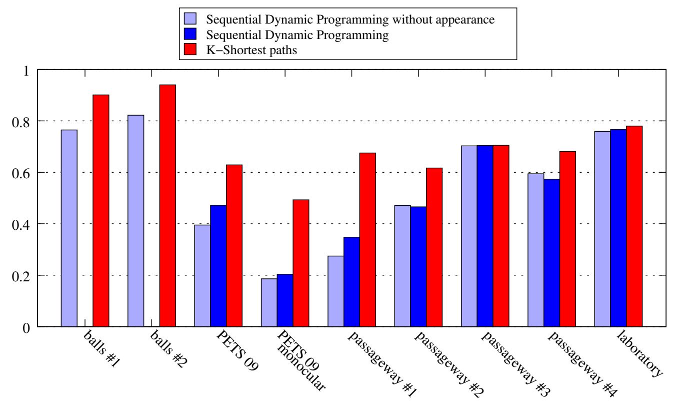
(b) MODA

Fig. 4. Detection precision (MODP) and accuracy (MODA) measures applied to the results of the original detections (POM), the sequential Dynamic Programming (DP) - with and without appearance model - and the proposed (KSP) trackers on various sequences.

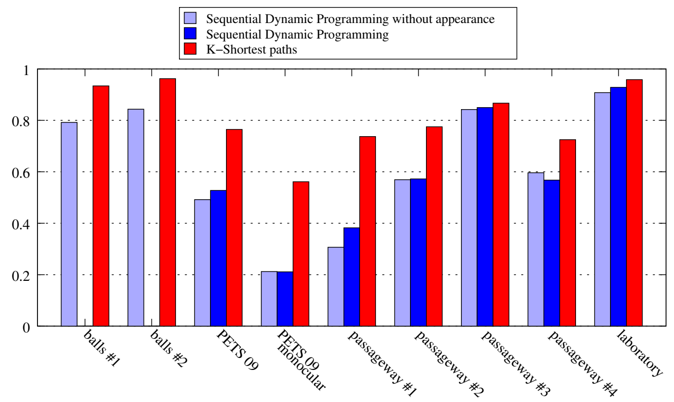
4.7 Monocular Pedestrian Results

To further emphasize the strength of our approach, we generated the detection maps using only one of the 7 available views of the PETS data set. Although POM still works on monocular sequences, ground plane localization is less precise: Without views from different angles, there is a depth ambiguity when estimating a pedestrian's position. Also, in the monocular case, occlusions often result in missed detection.

Under these challenging conditions, our algorithm shows its superiority over the sequential Dynamic Programming, even more clearly than in the multi-camera case. This is illustrated by Figs 4 and 5. In this context, DP's greedy strategy often leaves people outside the grid instead of trying to explain their very noisy detections. By contrast, KSP's joint optimization pays off and interpolates trajectories nicely. Monocular tracking results are depicted by Fig. 9.

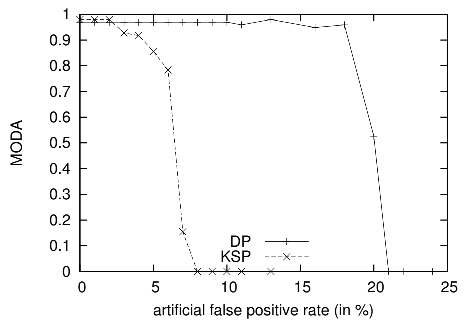


(a) MOTP

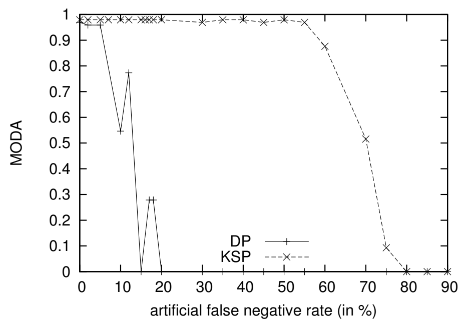


(b) MOTA

Fig. 5. Tracking precision (MOTP) and accuracy (MOTA) measures applied to the results of the sequential Dynamic Programming (DP) - with and without appearance model - and the proposed (KSP) trackers on various sequences.



(a)



(b)

Fig. 6. Artificially increasing the number of detection false positives (a) and false negatives (b), expressed as a percentage on the x-axis of the graphs.

4.8 Ball Tracking Results

Given the difference in grid scale, the balls move much faster than pedestrians, and can cross more than one grid location between consecutive frames. To deal with this environment, we thus had to extend the location neighborhood to include the next closest 49 locations, which increases the maximum distance traveled between consecutive frames to 3 grid locations.

Detection and tracking results for the two ball sequences are also illustrated on Figs 4 and 5, while screen shots are shown on Fig. 10. Detecting ping-pong balls is not a particularly difficult task, and thus POM's results are generally excellent, with very few false positives and false negatives. Because all balls have exactly the same appearance, DP's color model is useless and the comparison between the two algorithms is fair. As for the pedestrian environment, KSP outperforms DP on all 4 metrics. Here again, DP's greedy strategy is a disadvantage. Because it might be less costly to ignore some isolated detections, DP tends to leave out too many of them.

4.9 Failure Modes

Our tracking algorithm can be mainly affected by two elements: false detections, and missing ones. To quantify the effect of both types of detection error, we carried out the following experiment. We selected a 1,000-frame excerpt of the *laboratory* sequence showing high detection accuracy, and added various levels of random detection noise uniformly. We also randomly deleted detections from the same original sequence. That way, we artificially generated controlled amounts of false positives and false negatives. The sequences thus generated were then processed by both DP and KSP tracking methods, and evaluated using a known ground truth.

The results of this evaluation are presented on Fig. 6. The graph of Fig. 6(a) shows that KSP is more sensitive to false positives than DP. Beyond a density value, KSP is able to readily link false detections into - seemingly - coherent trajectories. Here, KSP's lack of motion model is a disadvantage over DP. Conversely, DP's tendency to leave out incomplete trajectories makes it more robust to this kind of noise. The graph of Fig. 6(b) shows the effect of missed detections. Both trackers react the same way: Beyond a false negative rate, the remaining detections are no longer linked together and remain unexplained. KSP shows nevertheless a much higher robustness to missed detections than DP does. This is consistent with our observations on real data: On Figs. 4 and 5, the difference between DP and KSP performance is usually larger when POM's occupancy maps have low accuracy.

Another problem to which our method is potentially vulnerable is identity switch. Since we rely entirely on detection data and do not use any appearance information nor complex motion model, there is no way to distinguish two trajectories intersecting. In practice, we do not suffer much from this, because most of the time, the objects evolve outside of each other's neighborhood. Moreover, the joint optimization of all trajectories pays off in this regard, as opposed to DP's greedy strategy.

4.10 Run time

Finally, we evaluate the speed of our new tracking algorithm. Solving the Linear Program with standard LP libraries [42] is slow, as evidenced by the curve labeled *LP* on the graph of Fig. 7(a). Using the complexity reduction method of §3.4 decreases the computation time by a factor of 10, as shown by the curve labeled *LP w/ compl. red.* Here, we pruned the graph by a radius of $\tau_1 = \tau_2 = 3$ (see Eq. 19).

By contrast, KSP is much faster: As illustrated on Fig. 7(a), there is a considerable speed gain of a factor 100 to 1,000, compared to the generic LP solver [42]. And the gain is still very significant when compared to the LP solver using complexity reduction methods.

Compared to the DP algorithm, KSP is about 10 times faster. DP suffers from having to load videos, in order to maintain its appearance models, and from its redundant batch processing [3]. Interestingly, when dealing with 25 fps videos, KSP can in average process a batch of frames in less than half the time it takes to play it. This means that for a frame rate of 25 fps or less, our tracker can readily operate in real time.

Figure 7(b) illustrates the grid size influence on run time of both DP and KSP. Applying POM on a 200-frame excerpt of the *laboratory* sequence, we generated occupancy grids of different resolutions, which were then processed by the two tracking algorithms. Results show a linear dependency on the grid size, which is consistent with the average complexity of k-shortest paths.

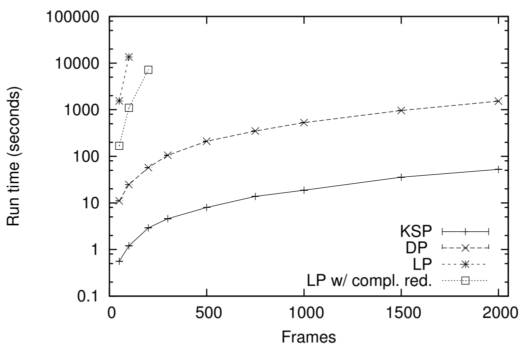
All the above experiments have been performed on a recent Linux PC, equipped with a 2.5 GHz Intel processor and 8 GB of memory. Tracking was applied to a part of the *laboratory* sequence, in which 5 to 7 people are present. For the k-shortest path, no particular optimization was performed, nor did we use any of the complexity reduction methods of §3.4. The results of DP and KSP on Fig. 7 are the average of 20 runs, plotted with 95% confidence interval. This is barely noticeable because the values are very peaked around the average.

Finally note that, whether solved with a generic LP package or the k-shortest paths algorithm, our framework always produces the exact same results, but KSP allows to obtain them much faster.

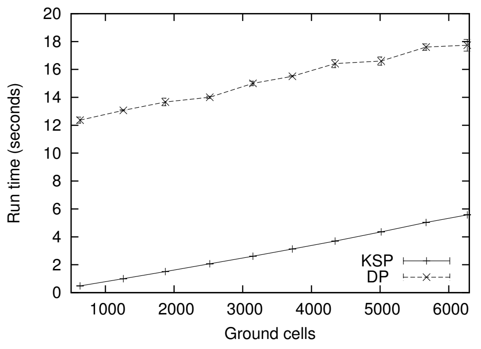
5 Conclusion

Combining frame-by-frame detections to estimate the most likely trajectories of an unknown number of targets, including their entrances and departures to and from the scene, is one of the most difficult components of a multi-object tracking algorithm. We have shown that by formalizing the motions of targets as flows along the edges of a graph of spatio-temporal locations, we can reduce this difficult estimation problem to a standard Linear Programming one. By relying on the k-shortest paths algorithm for the optimization of our problem, we could reduce the complexity to a tiny fraction of the one from the original LP problem, yielding an efficient algorithm performing robust multi-object tracking in real time on a standard computer.

The resulting algorithm is far simpler than current state-of-the-art alternatives and its convexity ensures that a global optimum can be found. It obtains a better performance than a state-of-the-art method on difficult real-word applications, in spite of having access to a more limited signal and requiring fewer meta-parameters. Future work will focus on integrating additional cues to our framework, such as an appearance or a motion models, to robustly handle identities of intersecting trajectories.



(a)



(b)

Fig. 7. (a) Run time comparison between our framework solved with a generic LP package (LP), our framework with a pruned graph solved with a generic LP package (LP w/ comp. red.), our framework solved with the k-shortest paths algorithm (KSP) and our earlier Dynamic Programming method (DP). Note that the y axis represents run time and is plotted in log scale. (b) Both DP and KSP algorithms scale almost linearly with the grid size.

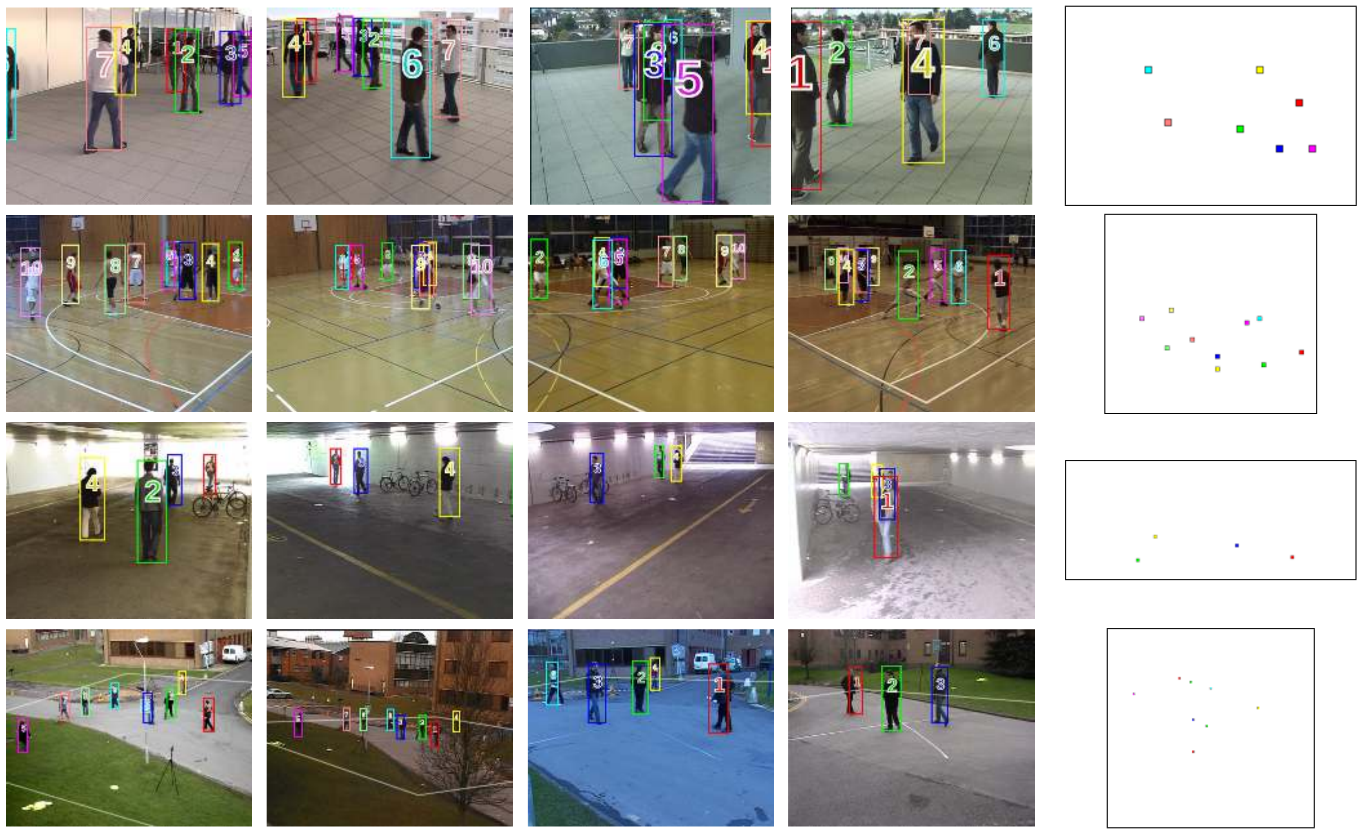


Fig. 8. Multi-camera pedestrian tracking results in various environments. Each of the first four columns shows a different camera view. The fifth column displays the top view. The first row comes from the laboratory sequence, the 2^{nd} from the basketball environment, the 3^{rd} from the passageway, and the last one from PETS 2009.

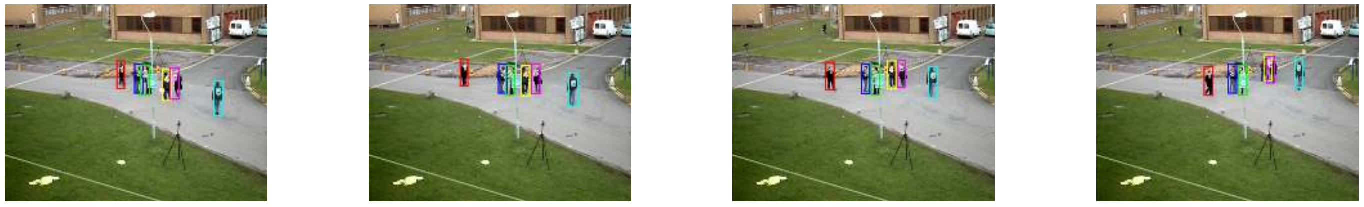


Fig. 9. Monocular pedestrian tracking results, from the PETS 2009 sequence.

Appendix A
Boolean Nature of the Solutions of the Relaxed LP Problem

In this appendix, we prove that, in our problem, the Relaxed Linear Program always converges to the optimal solution of the original Integer Program. For the sake of the proof, we rewrite Eq. 11 in matrix form as

$$\begin{aligned} \text{Maximize} \quad & \sum_{t,i} \log \left(\frac{\rho_i^t}{1-\rho_i^t} \right) \sum_{j \in \mathcal{N}(i)} f_{i,j}^t \ , \\ \text{subject to} \quad & \mathbf{f} \geq 0 \\ & C \cdot \mathbf{f} \leq [1, \dots, 1, 0, \dots, 0]^T \ , \end{aligned} \tag{20}$$

where $\mathbf{f}$ is the vector of all possible flows, and the matrix C is known as the *constraint matrix* and is depicted by Fig. 11.

The set $\{\mathbf{x} : C \cdot \mathbf{x} \leq b, \, \mathbf{x} \geq 0\}$ is known as the *polytope of feasible solutions*, and the optimal solution of the LP problem is to be found on its boundaries. In fact, except in degenerate cases where there are multiple solutions, only one of the vertices of the polytope represents the optimal solution. And even in degenerate cases, at least two of them are polytope vertices.

We use this formulation and the very specific form of the C matrix to prove that the coordinates of those vertices must all be either zero or one, which means the relaxed version of our problem always converges to the optimal solution of the original IP. We first give a couple of theorems that state that the solutions of a LP problem always are integer values, provided that

1) the constraint matrix exhibits the *total unimodularity* property, and
2) the right hand sides of the constraints are integers.

We then show that the C matrix of Eq. 20 is indeed totally unimodular, which in our case means that the solutions must be either zero or one.

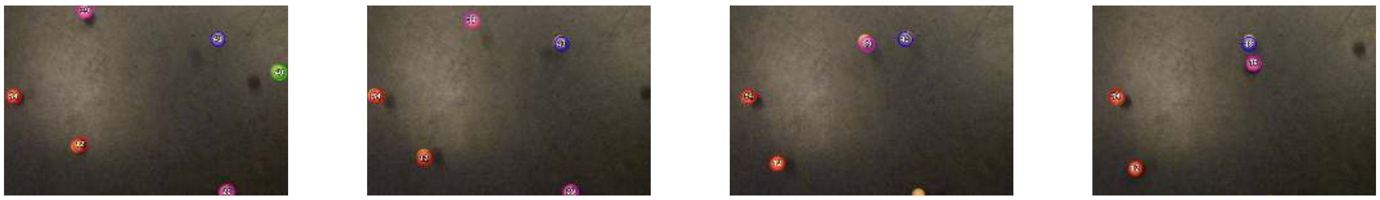


Fig. 10. Multiple ball tracking results. Successive screenshots are separated by 3 time frames.

A.1 Integral Nature of the Solutions

A rectangular matrix is said to be *totally unimodular* if all its square submatrices have determinant 0, -1, or 1. The following two theorems hold for such matrices, as shown in [43] and [44] respectively.

Theorem 1: A matrix $A = \{a_{ij}\} \in \mathbb{Z}^{m\times n}$ is totally unimodular if and only if for every subset $R \subseteq \{1,...,m\}$ of rows, there exists a partition $R = R_1 \cup R_2$, $R_1 \cap R_2 = \emptyset$ such that

$$\forall j = 1,...,n \qquad \sum_{i\in R_1} a_{ij} - \sum_{i\in R_2} a_{ij} \in \{0,-1,1\}. \tag{21}$$

Theorem 2: If C is a totally unimodular matrix, then the vertices of the polytope $\{x : Cx \le b, x \ge 0\}$ are integral, that is, have integer coordinates, for any integral vector b.

Recall that, except in degenerate cases, the solution of our LP is one of the vertices of the polytope of feasible solutions. Therefore, if the C matrix of Eq. 20 is totally unimodular, the vertices of the polytope must have coordinates either zero or one since, in addition to being integer, they must be between zero and one.

A.2 Total Unimodularity of the Constraint Matrix

We now turn to proving that C is totally unimodular. To this end, as depicted by Fig. 11, we split the rows of C into two subsets U_1 and U_2 that respectively correspond to the upper bound on flow and conservation of flow constraints:

$$U_1 : \qquad \left\{\textstyle\sum_{j\in\mathcal{N}(i)} f^t_{i,j} \le 1\right\}, \forall t, i \tag{22}$$

$$U_2 : \quad \left\{\textstyle\sum_{j\in\mathcal{N}(i)} f^t_{i,j} - \sum_{k:i\in\mathcal{N}(k)} f^{t-1}_{k,i} \le 0\right\}, \forall t, i$$
$$\textstyle\sum_{j\in\mathcal{N}(v_{\text{source}})} f_{v_{\text{source}},j} - \sum_{k:v_{\text{sink}}\in\mathcal{N}(k)} f_{k,v_{\text{sink}}} \le 0 \tag{23}$$

Without loss of generality, the columns of C can be ordered in a time-augmented fashion, such that the first column belongs to the first time frame and the last belongs to the last time frame T. In the sketch on Fig. 11, two rows corresponding to two different constraints are explicitly illustrated for a node u appearing at location j of time frame t. The two boxes that include a sequence of 1 and 0 correspond to the set of outgoing edges from node u. Similarly, the box with -1 and 0 correspond to the set of all incoming edges to the node (i.e., for a location i at time $t-1$, the corresponding entry of the matrix for node u is -1 if $j \in \mathcal{N}(i)$ or else 0).

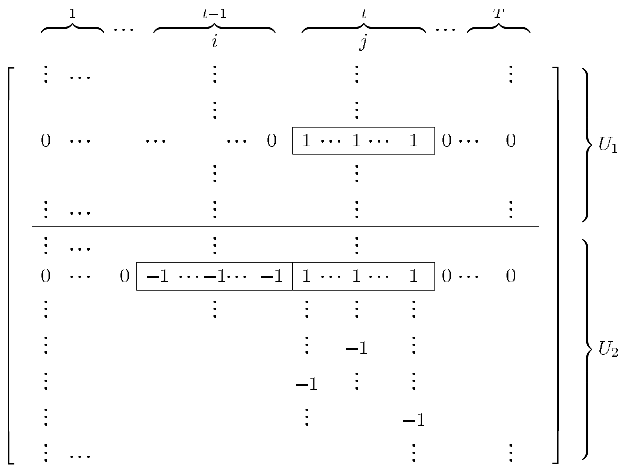


Fig. 11. A sketch of the constraint matrix C of the LP.

Let us also define $\mathbf{f} = \left[f^1_{1,1}, \ldots, f^T_{K,K}\right]^T$ as the vector containing all the flow values. The non-trivial constraints from (11), can be written in matrix format

$$C \cdot \mathbf{f} \le [1, \ldots, 1, 0, \ldots, 0]^T . \tag{24}$$

Let C_R be the sub-matrix, constructed by an arbitrary subset of rows R of the constraint matrix C. As can be observed from the column corresponding to location j on Fig. 11, each column of C_R can have three non-zero elements at most.

A trivial attempt is to partition the subset of rows into two such that the first partition, $R_1 = U_1 \cap R$, corresponds to the first set of nontrivial constraints (22) in the LP and the second partition, $R_2 = U_2 \cap R$, corresponds to the second set (23).

Clearly, for each column of C_R, there are eight different cases to be checked in total. These cases are summarized in Table 3 for a column j of C_R.

TABLE 3
All eight possible cases for a column j as a result of the proposed partitioning.

$\{a_{ij} \| i \in R_1\}$	$\{a_{ij} \| i \in R_2\}$	$\sum_{i\in R_1} a_{ij} - \sum_{i\in R_2} a_{ij}$
$\{0, ..., 0, 1\}$	$\{0, ..., 0\}$	1
$\{0, ..., 0, 1\}$	$\{0, ..., 0, 1\}$	0
$\{0, ..., 0, 1\}$	$\{0, ..., 0, -1\}$	2
$\{0, ..., 0, 1\}$	$\{0, ..., 0, 1, -1\}$	1
$\{0, ..., 0\}$	$\{0, ..., 0\}$	0
$\{0, ..., 0\}$	$\{0, ..., 0, 1\}$	-1
$\{0, ..., 0\}$	$\{0, ..., 0, -1\}$	1
$\{0, ..., 0\}$	$\{0, ..., 0, 1, -1\}$	0

A quick check over the eight possible cases reveals that the property given in Theorem 1 holds for all possible subsets of rows and for all columns, except for those that include a 1 in R_1 and a -1 in R_2 (i.e., third row in Table 3). For each such column, the corresponding row in R_1 that includes the nonzero entry can be moved to the second partition R_2 to satisfy the property. Note that, for all columns corresponding to the non-zero entries of the moved row, $\{a_{ij}|i \in R_1\}$ is $\{0,...,0\}$ and $\{a_{ij}|i \in R_2\}$ is either $\{0,...,0,1\}$ or $\{0,...,0,1,-1\}$, and hence, the property is satisfied.

Since the constraint matrix satisfies all conditions of Theorem 1, it is therefore totally unimodular.

Appendix B
K-Shortest Paths Algorithm

In this second appendix, we give a short description of the k-shortest paths algorithm. We refer the interested reader to [6] for further details.

Given a directed graph $G = (V, E)$, where V is the set of vertices and E is the set of edges, the algorithm computes the k-shortest node-disjoint paths - hereafter referred only as *shortest paths* - between v_{source} and v_{sink}, iteratively for $l = 1, \dots, k$, where k is fixed. Thus, at the l^{th} iteration, the l-shortest paths are computed by using the $l-1$ shortest paths from the previous iteration.

Let P_l be the optimal set of l paths at iteration l. The transition from P_l to P_{l+1} is based on the idea of shortest signed paths. A *signed path* is a sequence of nodes and sign-labeled edges connecting them in order, with each edge assigned a positive label (+) if it is in the direction of the path - that is from the source to the sink - or a negative label (-) otherwise.

At iteration $l+1$ of the algorithm, P_{l+1} can be obtained from P_l by augmenting it with a special kind of signed path p^*, called *interlacing* of P_l, which satisfies the following two conditions [6]:

1) An edge is common to both p^* and P_l if and only if it has a negative label;
2) A node is common to both p^* and P_l if and only if it is incident to an edge with negative label.

Note that the first condition is required to obtain edge-disjoint paths in P_{l+1}, which is necessary but not sufficient for node-disjoint paths. The second condition complements the first one for node-disjointness by excluding those signed paths having single node overlap with P_l.

Given a shortest edge-simple interlacing p^* of P_l, P_{l+1} can be obtained by *augmentation* of p^* and P_{l+1}, which is defined as adding positive labeled edges of p^* to P_l and removing negative labeled edges of p^* from P_l (See [6] for details). Fig. 12 gives an example of such an augmentation, where the shortest path is $P_1 = \{(v_{\text{source}}, v_i, v_j, v_{\text{sink}})\}$ and the shortest interlacing of P_1 is $p^* = (v_{\text{source}}, v_m, v_j, v_i, v_n, v_{\text{sink}})$ with corresponding edges labeled respectively as $(+,+,-,+,+)$. The optimal pair of paths is obtained by augmenting P_1 and p^* as $P_2 = \{(v_{\text{source}}, v_m, v_j, v_{\text{sink}}), (v_{\text{source}}, v_i, v_n, v_{\text{sink}})\}$.

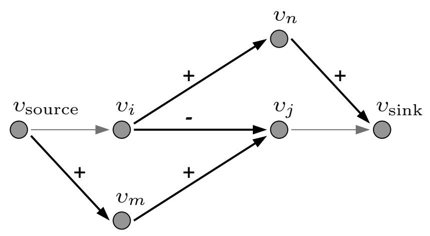


Fig. 12. An example of interlacing and the process of augmentation (only vertices that are in P_2 is shown). The shortest path are $P_1 = \{(v_{\text{source}}, v_i, v_j, v_{\text{sink}})\}$. The shortest interlacing of P_1 (bold lines with corresponding edge labels) is $p^* = (v_{\text{source}}, v_m, v_j, v_i, v_n, v_{\text{sink}})$. Augmentation of P_1 and p^* gives the optimal pair of paths $P_2 = \{(v_{\text{source}}, v_m, v_j, v_{\text{sink}}), (v_{\text{source}} v_i, v_n, v_{\text{sink}})\}$.

Interlacings in the original graph G correspond one-to-one to node-simple directed paths in an extended graph $G_l = (V_l, E_l)$ at iteration l of the algorithm, which can be obtained by a two-phase transformation from G, as specified in Table 4. The first phase addresses the above-described two conditions for being an interlacing since the node-disjointness criteria is relaxed to arc-disjointness. On the other hand, the second phase represents a transformation from signed paths to directed unsigned paths. Therefore, the shortest interlacings in G are equivalent to the shortest node-simple directed paths in G_l. In addition, the cost of an interlacing in G is the same as the cost of the corresponding directed path in G_l. Fig. 13 illustrates an example of this transformation for two nodes.

TABLE 4
Graph Transformation Phases [6]

• Split every node v_i in P_l, except v_{source} and v_{sink} into two nodes, namely $v_i^{'}$ and $v_i^{''}$. Assign all input, resp. output, edges of v_i to $v_i^{'}$, resp. $v_i^{''}$. Add a directed auxiliary edge of zero cost from $v_i^{'}$ to $v_i^{''}$.
• Reverse the direction and algebraic sign of cost for each edge in P_l, including auxiliary edges.

An additional edge cost transformation can be applied to G_l with possibly negative edge costs to obtain a canonic equivalent graph G_l^c with non-negative edge costs. The added benefit of this transformation is the reduction in the complexity of the shortest path computation at each iteration. Let the cost value for an edge $e_{i,j} \in E_l$ between nodes $v_i \in V_l$ and $v_j \in V_l$ be $c_{i,j}$, then G_l is transformed using the following equation [6]

$$c_{i,j}^{'} = c_{i,j} + s_i - s_j \qquad \forall e_{i,j} \in E_l \ , \tag{25}$$

where s_n represents the cost of the shortest path from the source node v_{source} to node v_n. In other words, at the l^{th} iteration, G_l is cost transformed to G_l^c by using the shortest path costs of nodes in G_{l-1}^c. Note that with

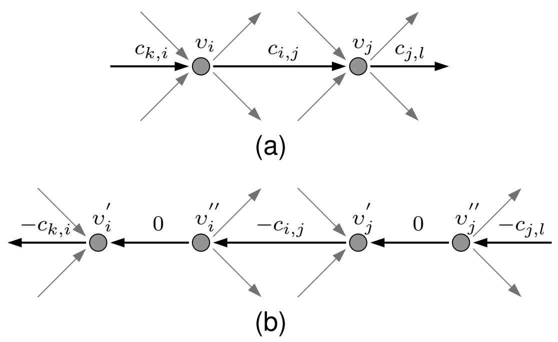


Fig. 13. The two-phase graph transformation. (a) Two nodes v_i and v_j in the original graph. Bold lines (with edge costs $c_{\cdot,\cdot}$) represent the arc of a shortest path incident to these two nodes. (b) The same part of the graph after the transformation.

this transformation, cost values for all paths between the source and the sink nodes change by the same constant factor, and hence, path ordering, in terms of the cost values, remains the same.

A summary of the complete algorithm is given in Algorithm 1 in pseudo-code. The function `efficient_shortest_path` implements a shortest path algorithm that is specifically designed for non-negative edge costs. In our implementation, we used Dijkstra's single source shortest paths algorithm [45] to compute the shortest path trees at each iteration. However, since the initial graph is a DAG, the first tree is computed in linear time by using a topological sort of its vertices [46].

The worst case complexity of the algorithm is $O(k(m+n\cdot\log n))$, where k is the number of objects appearing in a given time interval, m is the number of edges and n is the number of nodes in the final transformed graph. However, given the fact that we start with a DAG and that only very few cycles are introduced later by the interlacings, the shortest path algorithm performs efficiently and the average time complexity of our algorithm is almost linear with n.

Algorithm 1: K-shortest paths algorithm for the tracking problem

```
   input : a set of probabilistic occupancy maps
   output: a set of $k$ paths between $v_{\text{source}}$ and $v_{\text{sink}}$
1  Construct the initial graph $G$, with edge costs from Eq. 12
2  $p_1^* \leftarrow$ generic_shortest_path ($G$, $v_{\text{source}}$, $v_{\text{sink}}$)
3  $P_1 \leftarrow \{p_1^*\}$
4  for $l \leftarrow 1$ to $l_{max}$ do
5      if $l \neq 1$ then
6          if cost($P_l$) $\geq$ cost($P_{l-1}$) then
7              return $P_{l-1} = \{p_1^*, \ldots, p_{l-1}^*\}$
8          end
9      end
10     $G_l \leftarrow$ extend_graph ( $G$ ) /* as in Table 4 */
11     $G_l^c \leftarrow$ transform_edge_cost ($G_l$) /* Eq. 25 */
12     $p_{l+1}^* \leftarrow$ efficient_shortest_path ( $G_l^c$, $v_{\text{source}}$ ,
       $v_{\text{sink}}$ )
13     $p^* \leftarrow$ interlacing ( $P_l$ ) /* corres. to $p_{l+1}^*$ */
14     $P_{l+1} \leftarrow P_l \cup p^*$ /* i.e. augm. of $P_l$ and $p^*$ */
15 end
```

Acknowledgments

The authors would like to thank Ali Shahrokni, Anna Ellis and James Ferryman from the University of Reading for evaluating their results against the PETS 2009 ground truth.

References

[1] J. Giebel, D. Gavrila, and C. Schnorr, "A Bayesian Framework for Multi-Cue 3D Object Tracking," in *European Conference on Computer Vision*, 2004.

[2] A. Perera, C. Srinivas, A. Hoogs, G. Brooksby, and H. Wensheng, "Multi-Object Tracking Through Simultaneous Long Occlusions and Split-Merge Conditions," in *Conference on Computer Vision and Pattern Recognition*, June 2006, pp. 666–673.

[3] F. Fleuret, J. Berclaz, R. Lengagne, and P. Fua, "Multi-Camera People Tracking With a Probabilistic Occupancy Map," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 30, no. 2, pp. 267–282, February 2008.

[4] H. Jiang, S. Fels, and J. Little, "A Linear Programming Approach for Multiple Object Tracking," in *Conference on Computer Vision and Pattern Recognition*, 2007, pp. 744–750.

[5] G. B. Dantzig, *Linear Programming and Extensions.* Princeton University Press, 1963.

[6] J. W. Suurballe, "Disjoint Paths in a Network," *Networks*, vol. 4, pp. 125–145, 1974.

[7] J. Black, T. Ellis, and P. Rosin, "Multi-View Image Surveillance and Tracking," in *IEEE Workshop on Motion and Video Computing*, 2002.

[8] A. Mittal and L. Davis, "M2tracker: a Multi-View Approach to Segmenting and Tracking People in a Cluttered Scene," *Computer Vision and Image Understanding*, vol. 51(3), pp. 189–203, 2003.

[9] S. Iwase and H. Saito, "Parallel Tracking of All Soccer Players by Integrating Detected Positions in Multiple View Images," in *International Conference on Pattern Recognition*, August 2004, pp. 751–754.

[10] M. Xu, J. Orwell, and G. Jones, "Tracking Football Players With Multiple Cameras," in *International Conference on Image Processing*, October 2004, pp. 2909–2912.

[11] D. R. Magee, "Tracking multiple vehicles using foreground, background and motion models," *Image and Vision Computing*, vol. 22, no. 2, pp. 143–155, February 2004.

[12] B. Wu and R. Nevatia, "Tracking of Multiple, Partially Occluded Humans Based on Static Body Part Detection," in *Conference on Computer Vision and Pattern Recognition*, June 2006, pp. 951–958.

[13] J. Vermaak, A. Doucet, and P. Perez, "Maintaining Multimodality Through Mixture Tracking," in *International Conference on Computer Vision*, October 2003, pp. 1110–1116.

[14] K. Smith, D. Gatica-Perez, and J.-M. Odobez, "Using Particles to Track Varying Numbers of Interacting People," in *Conference on Computer Vision and Pattern Recognition*, 2005.

[15] Z. Khan, T. Balch, and F. Dellaert, "Mcmc-Based Particle Filtering for Tracking a Variable Number of Interacting Targets," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, no. 11, pp. 1805–1918, 2005.

[16] C. Yang, R. Duraiswami, and L. Davis, "Fast multiple object tracking via a hierarchical particle filter," in *International Conference on Computer Vision*, 2005.

[17] T. Mauthner, M. Donoser, and H. Bischof, "Robust Tracking of Spatial Related Components," in *International Conference on Pattern Recognition*, 2008.

[18] K. Okuma, A. Taleghani, N. de Freitas, J. Little, and D. Lowe, "A Boosted Particle Filter: Multitarget Detection and Tracking," in *European Conference on Computer Vision*, May 2004.

[19] W. Du and J. Piater, "Multi-Camera People Tracking by Collaborative Particle Filters and Principal Axis-Based Integration," in *Asian Conference on Computer Vision*, 2007, pp. 365–374.

[20] Q. Yu, G. Medioni, and I. Cohen, "Multiple Target Tracking Using Spatio-Temporal Markov Chain Monte Carlo Data Association," in *International Conference on Computer Vision*, 2007.

[21] E. Maggio, M. Taj, and A. Cavallaro, "Efficient Multi-Target Visual Tracking Using Random Finite Sets," *IEEE Transactions On Circuits And Systems For Video Technology*, vol. 18, no. 8, pp. 1016–1027, August 2008.

[22] C. Huang, B. Wu, and R. Nevatia, "Robust Object Tracking by Hierarchical Association of Detection Responses," in *European Conference on Computer Vision*, 2008, pp. 788–801.

[23] Y. Li, C. Huang, and R. Nevatia, "Learning to Associate: Hybrid-boosted Multi-Target Tracker for Crowded Scene," in *Conference on Computer Vision and Pattern Recognition*, June 2009.

[24] C. Beleznai, B. Frühstück, and H. Bischof, "Multiple Object Tracking Using Local Pca," in *International Conference on Image Processing*, 2006.

[25] W. Ge and R. T. Collins, "Multi-target data association by tracklets with unsupervised parameter estimation," in *British Machine Vision Conference*, September 2008.

[26] R. Eshel and Y. Moses, "Homography Based Multiple Camera Detection and Tracking of People in a Dense Crowd," in *Conference on Computer Vision and Pattern Recognition*, 2008.

[27] G. J. Brostow and R. Cipolla, "Unsupervised Bayesian Detection of Independent Motion in Crowds," in *Conference on Computer Vision and Pattern Recognition*, 2006, pp. 594–601.

[28] P. Nillius, J. Sullivan, and S. Carlsson, "Multi-Target Tracking - Linking Identities Using Bayesian Network Inference," in *Conference on Computer Vision and Pattern Recognition*, 2006, pp. 2187–2194.

[29] S. Khan and M. Shah, "Tracking Multiple Occluding People by Localizing on Multiple Scene Planes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 3, pp. 505–519, March 2009.

[30] B. Leibe, K. Schindler, and L. V. Gool, "Coupled Detection and Trajectory Estimation for Multi-Object Tracking," in *International Conference on Computer Vision*, October 2007.

[31] R. E. Bellman, *Dynamic Programming*. Princeton University Press, 1957.

[32] J. Wolf, A. Viterbi, and G. Dixon, "Finding the Best Set of K Paths Through a Trellis With Application to Multitarget Tracking," *Aerospace and Electronic Systems, IEEE Transactions on*, vol. 25, no. 2, pp. 287–296, March 1989.

[33] K. Shafique and M. Shah, "A Noniterative Greedy Algorithm for Multiframe Point Correspondence," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, no. 1, pp. 51–65, January 2005.

[34] P. P. A. Storms and F. C. R. Spieksma, "An Lp-Based Algorithm for the Data Association Problem in Multitarget Tracking," *Computers & Operations Research*, vol. 30, no. 7, pp. 1067–1085, June 2003.

[35] L. Zhang, Y. Li, and R. Nevatia, "Global Data Association for Multi-Object Tracking Using Network Flows," in *Conference on Computer Vision and Pattern Recognition*, 2008.

[36] N. Karmarkar, "A new polynomial time algorithm for linear programming," *Combinatorica*, vol. 4, pp. 373–395, 1984.

[37] A. Ellis, A. Shahrokni, and J. Ferryman, "Pets 2009 and winter-pets 2009 results: A combined evaluation," in *Twelfth IEEE International Workshop on Performance Evaluation of Tracking and Surveillance*, Snowbird, December 2009.

[38] R. Kasturi, D. Goldgof, P. Soundararajan, V. Manohar, J. Garofolo, M. Boonstra, V. Korzhova, and J. Zhang, "Framework for performance evaluation of face, text, and vehicle detection and tracking in video: Data, metrics, and protocol," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 2, pp. 319–336, February 2009.

[39] "Eleventh ieee international workshop on performance evaluation of tracking and surveillance," http://pets2009.net.

[40] J. Berclaz, F. Fleuret, and P. Fua, "Pom: Probabilistic occupancy map," 2007, http://cvlab.epfl.ch/software/pom/index.php.

[41] K. Bernardin and R. Stiefelhagen, "Evaluating multiple object tracking performance: The clear mot metrics," *EURASIP Journal on Image and Video Processing*, vol. 2008, 2008.

[42] A. Makhorin, "Glpk- gnu linear programming kit," 2008, http://www.gnu.org/software/glpk/.

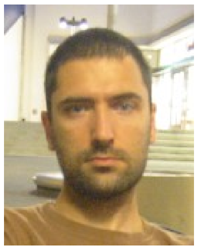
[43] A. Ghouila-Houri, "Caractérisation Des Matrices Totalement Unimodulaires," *Comptes Rendus Hebdomadaires des Séances de l'Académie des Sciences*, vol. 254, pp. 1192–1194, 1962.

[44] A. J. Hoffman and J. B. Kruskal, "Integral Boundary Points of Convex Polyhedra," in *Linear Inequalities and Related Systems*. Princeton University Press, 1956, pp. 223–246.

[45] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, pp. 269–271, 1959.

[46] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. The MIT Press, 2001.

Jérôme Berclaz received a MS degree in Communication Systems in 2004 and a PhD in Computer Vision in 2010 from EPFL (Swiss Federal Institute of Technology). He is now a post-doctoral researcher at the Computer Vision Laboratory and the Signal Processing Laboratory from EPFL. His main research interest is Computer Vision .



François Fleuret received the PhD degree in probability from the University of Paris VI in 2000, and the habilitation degree in Applied Mathematics from the University of Paris XIII in 2006. He holds a Senior Researcher position at the Idiap Research Institute in Switzerland. Prior to that, he held positions at the University of Chicago, at the French Institut de Recherche en Informatique et en Automatique (INRIA), and at the École Polytechnique Fédérale de Lausanne (EPFL). His main research interests are at the interface between statistical methods and algorithmic, centered on the development of algorithmically efficient machine learning techniques.

Engin Türetken received the BSc and the MSc degrees in Electrical and Electronics Engineering from Middle East Technical University in 2005 and 2008, respectively. He is currently a PhD student in the school of Computer and Communication Sciences at Swiss Federal Institute of Technology (EPFL). His research interests include computer vision, graph theory and combinatorial optimization.

Pascal Fua received an engineering degree from Ecole Polytechnique, Paris, in 1984 and the Ph.D. degree in Computer Science from the University of Orsay in 1989. He joined EPFL (Swiss Federal Institute of Technology) in 1996 where he is now a Professor in the School of Computer and Communication Science. Before that, he worked at SRI International and at INRIA Sophia-Antipolis as a Computer Scientist. His research interests include shape modeling and motion recovery from images, analysis of microscopy images, and Augmented Reality. He has (co)authored over 150 publications in refereed journals and conferences. He has been an associate editor of IEEE journal Transactions for Pattern Analysis and Machine Intelligence and has often been a program committee member, area chair, and program chair of major vision conferences.