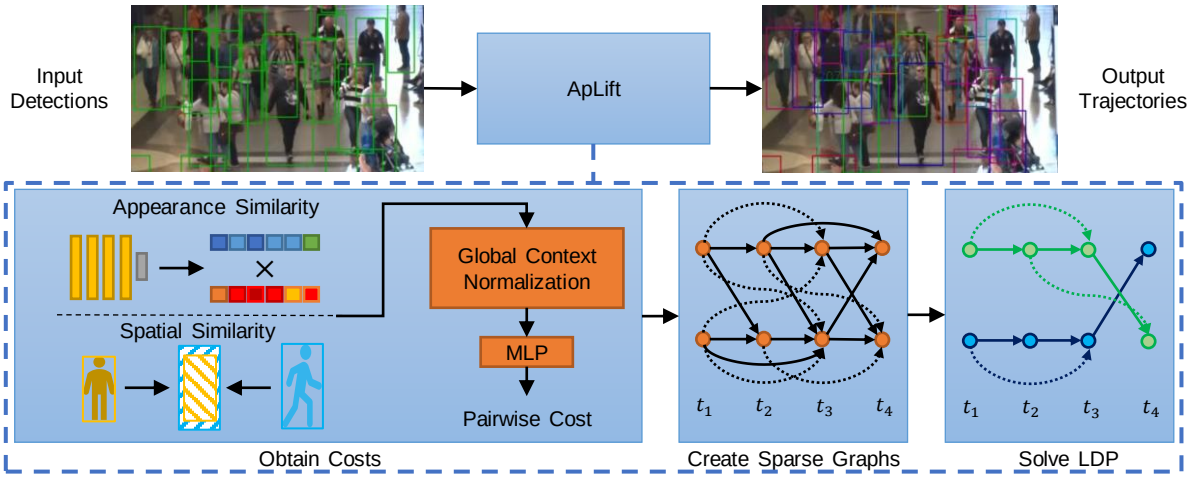

Figure 1. Overview of the ApLift framework. Input detections are used to obtain pairwise costs by an MLP with spatial and appearance features. Based on the costs, two sparse graphs are constructed and passed to our proposed approximate LDP solver. Dashed arrows represent lifted edges and solid arrows base edges. In figure *Solve LDP* equally colored nodes and edges belong to the same trajectory.

8. Appendix

This Appendix contains details about our approximate LDP solver and the whole MOT framework used in ApLift. We depicts this framework in Figure 1.

Appendix outline. We start with providing additional notation and abrreviations list in Section 8.1. Sections 8.2-8.8 present the message passing solver implementation and the algorithms used for it. Sections 8.9-8.13.2 present details about processing of the tracking data. Finally, Section 8.14 discusses theoretical runtime of the solver and Section 8.15 presents examples of qualitative results. The Appendix is rather extensive, especially its algorithmic part. Therefore, we provide its section outline within the context of the whole method bellow.

LDP solver outline. Figure 2 contains a scheme of all algorithms used in our LDP solver. The algorithms are stated either in the main paper or in this Appendix. The solver performs an explicitly given number of message passing iterations. Section 8.7 describes the full solver run and an overview of all methods used within one message passing iteration. Once in five iterations, new primal solution is computed (Sections 4.6 and 8.8). Once in twenty iterations, new subproblems are separated and added to the problem. These are path and cut subproblems (see Sections 4.3 and 4.4). Methods for their separations are described in Sections 8.4 and 8.5.

Message passing. Messages are sent between the subproblems. Each subproblem creates messages to be sent by computing min-marginals of its variables. Section 8.2 presents algorithms used for obtaining min-marginals of inflow and outflow subproblems. The algorithms allow us to efficiently obtain min-marginals of all lifted or all base edges of a subproblem at once. Messages from cut and path subproblems are obtained by modifications of the respective algorithms for their optimization. See Section 4.4 in the main text for the cut subproblem optimization and Section 8.3 for path subproblem optimization.

Tightening by separation. We create the new path and cut subproblems in order to tighten the LP relaxation of the problem (3). Section 8.6 discusses the guaranteed lower bound improvement achieved by separating the new subproblems using algorithms in Sections 8.4 and 8.5.

Tracking. The proposed tracking framework contains additional processing steps, which are briefly mentioned in the main paper. A detailed description and additional evaluation data is provided in this appendix. To construct the graph we calculate costs based on two features as described in Section 5.1 and add multiple scalings which details can be found in Section 8.9. We also determine very confident edges and set their cost based on heuristics explained in Section 8.12. Furthermore, additional implementation and training details for the classifier are presented in Sections 8.10 and 8.11. The efficient inference based on interval solutions is provided in Section 8.13.1. Finally we show details for the post-processing based on heuristics in Section 8.13.2.

8.1. Additional notation and abbreviations

- $x \in A^B$ denotes a mapping $x : B \to A$.
- $[n]$ denotes the set of numbers $\{1, 2, \dots, n\}$.
- LP: Linear programming.
- MP: Message passing.
- DP: disjoint paths problem.
- LDP: Lifted disjoint paths.
- DFS: Depth first search.
- MPLP: Max Product Linear Programming.

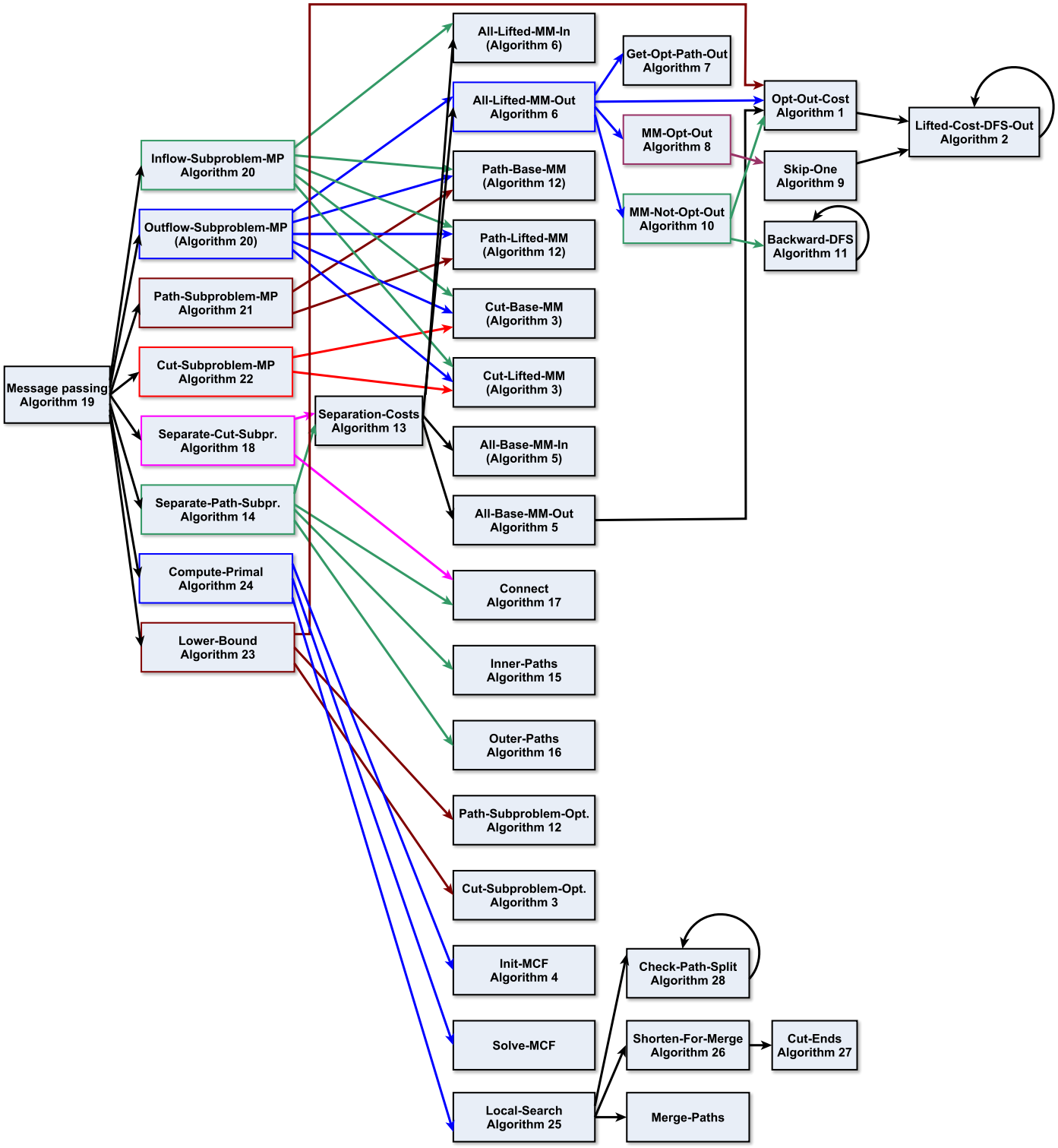

Figure 2. The scheme of our message passing algorithm and all its subroutines described in this work. An arrow from Algorithm X to Algorithm Y means that Algorithm X calls Algorithm Y. Algorithms in brackets denote that their modifications are used as the respective procedures. Abbreviation MP means Messate-Passing. Some algorithms for inflow subproblems are omitted for clarity because they are analogical to outflow subproblem algorithms.

8.2. Min-Marginals for Inflow and Outflow Subproblems

We detail routines for computing min-marginals for all base edges at once (Algorithm 5) and all lifted edges at once (Algorithm 6). All the stated algorithms assume outflow subproblems. Modification to inflow subproblems is done via proceeding in the oposite edge direction.

Iteratively computing min-marginals and performing operation (6) would be inefficient, since it would involve calling Algorithm 1 $\mathcal{O}(|\delta_E^+(v)| + |\delta_{E'}^+(v))|$ times. To speed up iterative min-marginal updates, we can reuse computations as done in Algorithm 5 for base edges and in Algorithm 6 for lifted edges.

Algorithm 5 for computing base edge min-marginals uses the fact that lifted edge costs do not change and therefore Algorithm 1 needs to be called only once. For lifted edges, Algorithm 6 interleaves min-marginal computation and reparametrization updates (6) such that computations can be reused. We introduce auxiliary variables γ'_{vw} in line 3 that keep track of future reparametrization updates.

For the min-marginals, we will need slight extensions of Algorithm 1 and a method to additionally compute a labeling that attains the optimum. These methods are given in Algorithm 9 and 7.

In Algorithm 6, path P^* representing the optimal solution of the outflow problem is found by calling Algorithm 1 followed Algorithm 7. Then, Algorithm 8 computes min-marginals for the lifted edges that are active in the optimal solution. In the end of Algorithm 6, min-marginals are computed for those lifted edges that are not active in the optimal solution.

For computing min-marginals of edges that are active in the optimal solution, we need as a subroutine Algorithm 9, an extended version of Algorithm 1. Algorithm 9 restricts the vertices taken into consideration during the optimization. In particular, a special vertex r is given that is to be excluded from the optimization. Values lifted_cost$[u]$ are reused for those vertices u where $ur \notin \mathcal{R}_G$ because these values are not affected by excluding vertex r.

Min-marginals for vertices inactive in the optimal solution are computed by Algorithms 10 and 11. The algorithms rely on structure back_cost which is an analogy of lifted_cost. Structure back_costs$[u]$ contains the minimum cost of all vu-paths w.r.t. to the costs of all lifted edges connecting v with the vertices of the path plus the cost of the first base edge of the path. Note that lifted_costs$[u]$ is defined analogically but contains the minimum cost of all ut-paths. Therefore, the minimal solution where a lifted edge $vu \in E'$ is active can be obtained as follows:

$$\min_{(z,y,y') \in \mathcal{X}_v^{out} : y'_{vu}=1} \langle (z_v, y, y'), \theta^{out} \rangle = \\ = \text{lifted\_cost}[u] + \text{back\_cost}[u] - \tilde{\theta}'_{vu} \quad (11)$$

The cost of lifted edge $\tilde{\theta}'_{vu}$ must be subtracted because it is involved in both values lifted_cost$[u]$ and back_cost$[u]$.

Algorithm 11 performs two tasks simultaneously. First, it is a DFS procedure for computing back_cost. Contrary to Algorithm 2 that performs DFS for obtaining lifted_cost, Algorithm 11 proceeds in the opposite edge direction. It again uses the fact that a subpath of a minimum-cost path must be minimal. Second, it directly computes min marginal for already processed vertex u on Line 10 and involves this change in setting back_cost$[u]$ on Line 11.

Speeding up DFS: All the algorithms for obtaining optimal solution or min-marginals of inflow and outflow subproblems call DFS procedures. It can be considered that the order of processing the relevant nodes reachable form the central node is always the same. Therefore, we call DFS for each inflow and outflow subproblem only once during their initialization and store the obtained list of processed nodes. The full DFS in Algorithm 2 is replaced by traversing the precomputed node list in the forward direction. Algorithm 11 is replaced by traversing this node list in the backward direction.

Algorithm 5 All-Base-MM-Out$(v, \tilde{\theta})$

Input start vertex v, costs $\tilde{\theta}$
Output base edge min-marginals γ_{vu} $\forall vu \in \delta_E^+(v)$

1: $(\text{opt}, \text{lifted\_cost}, \alpha)$ =Opt-Out-cost(v, θ^{out})
2: $e^* = \underset{vw \in \gamma_E^+}{\text{argmin}} \{\alpha_{vw}\}$, $e^{**} = \underset{vw \in \gamma_E^+ \setminus \{e^*\}}{\text{argmin}} \{\alpha_{vw}\}$
3: $\forall vu \in \delta^+(v) : \gamma_{vu} = \alpha_{vu} - \min(\alpha_{e^{**}}, 0)$

Algorithm 6 All-Lifted-MM-Out$(v, \tilde{\theta})$

Input starting vertex v, $\tilde{\theta}$
Output lifted edge min-marginals γ'_{vu} $\forall vu \in \delta_{E'}^+(v)$

1: $(\text{opt}, \text{lifted\_cost}, \alpha, \text{next})$ =Opt-Out-cost(v, θ^{out})
2: P_V^*=Get-Opt-Path-Out$(\theta^{out}, \alpha, \text{next})$
3: $\forall vw \in \delta_{E'}^+(v) : \gamma'_{vw} = 0$
4: (opt, γ') =MM-Opt-Out$(v, P_V^*, \text{opt}, \gamma', \tilde{\theta})$
5: γ' =MM-Not-Opt-Out$(v, \text{opt}, \gamma', \tilde{\theta})$

Algorithm 7 Get-Opt-Path-Out

Input costs $\tilde{\theta}$, vector α such that $\forall vw \in \delta_E^+(v) : \alpha_{vw}$ is the optimal value if vw is active, next
Output min cost path P_V^*

$w^* = \text{argmin}_{w:vw \in \delta_E^+(v)} \alpha_{vw}$
if $\alpha_{vw^*} < 0$ **then**
  while $w^* \neq t$ **do**
    $P_V^* \leftarrow w^*$
    $w^* = \text{next}[w^*]$
  end while
else
  $P_V^* = \emptyset$
end if

Algorithm 8 MM-Opt-Out

Input starting vertex v, optimal path $P_V^* = (v_1, \dots, v_k)$, value of optimal path opt, γ', costs $\tilde{\theta}$
Output updated cost of optimal path opt, new reparametrization updates γ'

for all $v_i = v_1, \dots, v_k : vv_i \in \delta_{E'}^+(v)$ **do**
  $\alpha = \text{Skip-One}(v, v_i, \tilde{\theta} - (\mathbb{0}, \gamma'), \text{lifted\_cost}, \text{next})$
  $\gamma'_{vv_i} = \text{opt} - \alpha$
  $\text{opt} = \alpha$
end for

Algorithm 9 Skip-One

Input v, ignored vertex r, $\tilde{\theta}$, lifted_cost, next
Output optimal value opt

for $u \in V : vu \in \mathcal{R}_G \wedge ur \in \mathcal{R}_G$ **do**
  $\text{lifted\_cost}[u] = \infty, \text{next}[u] = \emptyset$
end for
$\text{lifted\_cost}[r] = 0, \text{next}[r] = t$
Lifted-Cost-DFS-Out$(v, v, \tilde{\theta}, \text{lifted_cost}, \text{next})$
$\forall w : vw \in \delta_E^+(v) : \alpha_{vw} = \tilde{\theta}_v + \tilde{\theta}_{vw} + \text{lifted\_cost}[w]$
$\text{opt} = \min(\min_{w:vw \in \delta_E^+(v)} \alpha_{vw}, 0)$

8.3. Optimization of path subproblems.

We denote by θ^P the edge costs in subproblem of vw-path P. The optimization over the feasible set $\mathcal{X}^P$ w.r.t. costs θ^P is detailed in Algorithm 12. It checks whether there exists exactly one positive edge and whether it is either a lifted or a strong base edge (Line 2). If so, the optimal solution is either (i) all edges except the two largest ones (Line 6) or (ii) all edges (Line 8), whichever gives smaller objective value. If the above condition does not hold, the optimal solution can be chosen to contain all negative edges (Line 11).

A variation of Algorithm 12 with a specified edge fixed to either 0 or 1 is used for computing min-marginals

Algorithm 10 MM-Not-Opt-Out

Input v, current optimum opt, reparametrization update γ', $\tilde{\theta}$
Output changed reparametrization update γ'

$(\text{opt}, \text{lifted\_cost}) = \text{Opt-Out-cost}(v, \tilde{\theta} - (\mathbb{0}, \gamma'))$
for all $u : vu \in \mathcal{R}_G$ **do**
  if $u \in P_V^*$ **then**
    $\text{visited}[u] = true$
    $\text{back\_cost}[u] = \text{opt} - \text{lifted\_cost}[u]$
    if $vu \in E'$ **then** $\text{back\_cost}[u] \mathrel{+}= \tilde{\theta}'_{vu} - \gamma'_{vw}$
  else
    $\text{visited}[u] = false$
    if $vu \in \delta_E^+(v)$ **then**
      $\text{back\_cost}[u] = \tilde{\theta}_{vu}$
    else
      $\text{back\_cost}[u] = \infty$
    end if
  end if
end for
for all $vu \in \delta_{E'}^+(v)$ **do**
  if $\text{visited}[u] = false$ **then**
    Backward-DFS$(v, u, \tilde{\theta}, \gamma', \text{opt}, \text{back_cost})$
  end if
end for

Algorithm 11 Backward-DFS

Input $v, u, \tilde{\theta}, \gamma', \text{opt}, \text{back\_cost}$
Output γ', back_cost

$\alpha = \text{back\_cost}[u]$
for $wu \in \delta_E^-(u) : vw \in \mathcal{R}_G$ **do**
  if $\text{visited}[w] = false$ **then**
    Backward-DFS$(v, w, \tilde{\theta}, \gamma', \text{back_cost})$
  end if
  $\alpha = \min\{\text{back\_cost}[w], \alpha\}$
end for
if $vu \in E'$ **then**
  $\text{opt}_u = \alpha + \text{lifted\_cost}[u]$
  $\gamma'_{vu} = \text{opt}_u - \text{opt}$
  $\text{back\_cost}[u] = \alpha + \tilde{\theta}_{vu} - \gamma'_{vu}$
else
  $\text{back\_cost}[u] = \alpha$
end if
$\text{visited}[u] = true$

8.4. Separation for Path Subproblems

The path subproblem separation procedure is described in Algorithm 14. The algorithm finds paths together with a lifted edge connecting the start and the end point of the path such that exactly one lifted edge has positive cost, while all the remaining base and lifted edges have negative cost.

Algorithm 12 Path-Subproblem-Optimization

Input Edge costs θ^P
Output optimal value opt of subproblem.

$E^+ = \{kl \in P_{E'} \cup vw | \theta'^P_{kl} > 0\} \cup \{kl \in P_E | \theta^P_{kl} > 0\}$
if $E^+ = \{kl\} \wedge kl \in P_{E'} \cup vw \cup E_0$ **then**
  $\alpha = \min\{\min_{ij \in P_E \setminus E^+} |\theta^P_{ij}|, \min_{ij \in P_{E'} \cup vw \setminus E^+} |\theta'^P_{ij}|\}$
  $\beta = \begin{cases} \theta'^P_{kl}, & kl \in P_{E'} \cup vw \\ \theta^P_{kl}, & kl \in P_E \end{cases}$
  if $\alpha < \beta$ **then**
    $\text{opt} = \sum_{ij \in P_E \setminus E^+} \theta^P_{ij} + \sum_{ij \in P_{E'} \cup vw \setminus E^+} \theta'^P_{ij} + \alpha$
  else
    $\text{opt} = \sum_{ij \in P_E} \theta^P_{ij} + \sum_{ij \in P_{E'} \cup vw} \theta'^P_{ij}$
  end if
else
  $\text{opt} = \sum_{ij \in P_E \setminus E^+} \theta^P_{ij} + \sum_{ij \in P_{E'} \cup vw \setminus E^+} \theta'^P_{ij}$
end if
return opt

First, lifted and base edge costs are obtained in Algorithm 13 by computing min-marginals of inflow and outflow factors. Second, a graph with an empty edge set E^1 is created. Then, edges with negative costs are added to E^1 in ascending order. After adding an edge, we check whether separating path subproblems with edge costs leading to lower bound improvement is possible. Such a factor must contain the newly added edge, one positive lifted edge and edges that already are in the edge set E^1.

Algorithm 15 separates those paths subproblems where the only positive edge is the one connecting the path's endpoints. Algorithm 16 separates those path subproblems where the only positive edge is one of the edges within the path. Algorithm 17 updates connectivity structures by adding edge ij to the edge set E^1.

Each path subproblems has a guaranteed lower bound improvement, see Proposition 2. We add each found path subproblem to priority queue Q, where we sort w.r.t. the guaranteed lower bound improvement. After searching for path subproblems, we add the k best path subproblems from queue Q to the optimization problem.

8.5. Separation for Cut Subproblems

Algorithm 18 separates cut subproblems. The algorithm finds cuts consisting of base edges with positive costs and a lifted edge having endpoints on both sides of the cut and negative cost. Similarly as for the path subproblem separation, lifted and base edge costs are obtained by computing min-marginals of inflow and outflow factors in Algorithm 13. Each edge $uv \in E'^-$ is a candidate lifted edge for a uv-cut factor.

Algorithm 13 Separation-Costs

Input Current cost in inflow and outflow factors $\theta^{in}, \theta^{out}$
Output Cost reparametrization $\forall uv \in E : \tilde{\theta}_{uv}, \forall uv \in E' : \tilde{\theta}'_{uv}$

$\forall uv \in E : \tilde{\theta}_{uv} = 0, \forall uv \in E' : \tilde{\theta}'_{uv} = 0$
for all $u \in V \setminus \{s, t\}$ **do**
  $\forall uv \in \delta^+_E(u) : \gamma^{out}_{uv} = 0$
  $\forall uv \in \delta^-_E(u) : \gamma^{in}_{vu} = 0$
  $\gamma'^{out}_u = 0.5 \cdot$All-Lifted-MM-Out$(u, \theta^{out}_u)$
  $\gamma'^{in}_u = 0.5 \cdot$All-Lifted-MM-In$(u, \theta^{in}_u)$
  $\gamma^{out}_u =$All-Base-MM-Out$(u, \theta^{out}_u - (\gamma^{out}_u, \gamma'^{out}_u))$
  $\gamma^{in}_u =$All-Base-MM-In$(u, \theta^{in}_u - (\gamma^{in}_u, \gamma'^{in}_u))$
  $\forall uv \in \delta^+_E(u) : \tilde{\theta}_{uv} \mathrel{+}= \gamma^{out}_{uv}$
  $\forall uv \in \delta^-_E(u) : \tilde{\theta}_{vu} \mathrel{+}= \gamma^{in}_{vu}$
  $\forall uv \in \delta^+_{E'}(u) : \tilde{\theta}'_{uv} \mathrel{+}= \gamma'^{out}_{uv}$
  $\forall uv \in \delta^-_{E'}(u) : \tilde{\theta}'_{vu} \mathrel{+}= \gamma'^{in}_{vu}$
end for

Algorithm 14 Separate-Path-Subproblem

Input Cost threshold ε

$\tilde{\theta} =$Separation-Costs$(\theta^{in}, \theta^{out})$
$G^1 = (V, E^1 = \emptyset)$
$E^- = \{vw \in E | \tilde{\theta}_{vw} < -\varepsilon\} \cup \{vw \in E' | \tilde{\theta}'_{vw} < -\varepsilon\}$
$E'^+ = \{vw \in E' | \tilde{\theta}'_{vw} > \varepsilon\}$
$\forall v \in V :$ desc$[v] = \{v\}$, pred$[v] = \{v\}$
Priority-Queue $Q = \emptyset$
for all $ij \in E^-$ ascending in $\tilde{\theta}$ **do**
  if $ij \in E$ **then** $c_{ij} = \tilde{\theta}_{ij}$
  else $c_{ij} = \theta'_{ij}$
  Inner-Paths$(i, j, c_{ij}, \text{pred}, \text{desc}, E^+, E^1, Q)$
  Outer-Paths$(i, j, c_{ij}, \text{pred}, \text{desc}, E^+, E^1, Q)$
  Connect$(i, j, \text{pred}, \text{desc}, E^1)$
end for

Algorithm 15 Inner-Paths

Input $i, j, c_{ij}, \text{pred}, \text{desc}, E^+, E^1, Q$

for all $p \in$ pred$[i]$ **do**
  for all $d \in$ desc$[j]$ **do**
    if $pd \in E^+$ **then**
      $P_1 =$Find-Path(p, i, E^1)
      $P_2 =$Find-Path(j, d, E^1)
      $P = (P_1, ij, P_2)$
      $priority = \min\{|c_{ij}|, \tilde{\theta}'_{pd}\}$
      $Q \leftarrow$ (Path-Problem$(P), priority)$
    end if
  end for
end for

The edge set E^1 initially contains all base edges with cost lower than ε. The remaining base edges are added to

Algorithm 16 Outer-Paths

Input i, j, c_{ij}, pred, desc, E^+, E^1, Q

for all $p \in \text{pred}[j]$ **do**
  for all $d \in \text{desc}[i]$ **do**
    if $dp \in E^+$ **then**
      $P_1 = $Find-Path$(i, d, E^1)$
      $P_2 = $Find-Path$(p, j, E^1)$
      $P = (P_1, ij, P_2)$
      $priority = \min\{|c_{ij}|, \tilde{\theta}'_{dp}\}$
      $Q \leftarrow$ (Path-Problem$(P), priority)$
    end if
  end for
end for

Algorithm 17 Connect

Input i, j, pred, desc, E^1

for all $p \in \text{pred}[i]$ **do**
  for all $d \in \text{desc}[j]$ **do**
    $\text{desc}[p] \leftarrow d$
    $\text{pred}[d] \leftarrow p$
    $E^1 \leftarrow ij$
  end for
end for

E^1 in ascending order. Whenever a newly added edge ij causes a connection between u and v where $uv \in E'^-$, a uv-cut C is separated. We select the cut C to contain only those edges that do not belong to E^1. This ensures that ij is the weakest cut edge. In the same time, C is the best uv-cut with respect to the cost of the weakest cut edge.

The found cut factors are added to a priority queue where the priority represents guaranteed lower bound improvement (see Proposition 3) after adding the factor to our problem.

8.6. Tightening Lower Bound Improvement

In order to show that the separation procedures in Algorithms 14 and 18 lead to relaxations that improve the lower bound we show the following: (i) Certain reparametrization used in the above algorithms are non-decreasing in the lower bound. (ii) Separation procedures find new subproblems such that w.r.t. the above reparametrization, a guaranteed lower bound improvement can be achieved.

Points (i) and (ii) guarantee that the same lower bound achievement w.r.t. the original reparametrization can be found. The special reparametrization chosen helps empirically to find good subproblems.

Lemma 1. *Let $\mathsf{s} \in \mathcal{S}$ be a subproblem, θ its cost and $L_{\mathsf{s}}(\theta)$ its lower bound for cost θ. Given a cost reparametrization γ such that*

Algorithm 18 Separate-Cut-Subproblem

Input Cost threshold ε

$\tilde{\theta} =$Separation-Costs$(\theta^{in}, \theta^{out})$
$E'^- = \{vw \in E' | \tilde{\theta}'_{vw} < -\varepsilon\}$
$E^- = \{vw \in E | \tilde{\theta}'_{vw} < \varepsilon\}$, $E^+ = E \setminus E^-$
$E^1 = E^-$, $G^1 = (V, E^1)$
Priority-Queue $Q = \emptyset$
for all $ij \in E^+$ ascending in $\tilde{\theta}$ **do**
  for all $u \in \text{pred}[i]$ **do**
    for all $v \in \text{desc}[j]$ **do**
      if $uv \in E'^-$ **then**
        $C =$ cut between u, v using edges $E \setminus E^1$
        $priority = \min\{\tilde{\theta}_{ij}, |\tilde{\theta}'_{uv}|\}$
        $Q \leftarrow$ (Cut-Problem$(C, u, v), priority)$
      end if
    end for
  end for
  Connect$(i, j, \text{pred}, \text{desc}, E^1)$
end for

1. $\forall i \in [d_{\mathsf{s}}]$

$$\gamma_i = \begin{cases} \leq 0 & \text{if } \exists x^* \in \operatorname{argmin}_{x \in \mathcal{X}^{\mathsf{s}}} \langle \theta, x \rangle : x^*_i = 1 \\ \geq 0 & \text{if } \exists x^* \in \operatorname{argmin}_{x \in \mathcal{X}^{\mathsf{s}}} \langle \theta, x \rangle : x^*_i = 0 \end{cases} \tag{12}$$

2. $\operatorname{argmin}_{x \in \mathcal{X}^{\mathsf{s}}} \langle \theta, x \rangle \subseteq \operatorname{argmin}_{x \in \mathcal{X}^{\mathsf{s}}} \langle \theta - \gamma, x \rangle$

and a coordinate-wise scaled reparametrization $\gamma(\omega)$ defined by coefficients $\omega \in [0,1]^{\mathsf{s}}$ where $\forall i \in [d_{\mathsf{s}}] : \gamma(\omega)_i = \omega_i \gamma_i$, it holds:

1. *The lower bound of s after reparametrization $\gamma(\omega)$ is $L_{\mathsf{s}}(\theta - \gamma(\omega)) = L_{\mathsf{s}}(\theta) - \sum_{i \in [d_{\mathsf{s}}] : \gamma_i < 0} \omega_i \gamma_i$.*
2. $\operatorname{argmin}_{x \in \mathcal{X}^{\mathsf{s}}} \langle \theta, x \rangle \subseteq \operatorname{argmin}_{x \in \mathcal{X}^{\mathsf{s}}} \langle \theta - \gamma(\omega), x \rangle$

Proof. We start with evaluating $\langle \theta - \gamma(\omega), x^* \rangle$ where $x^* \in \operatorname{argmin}_{x \in \mathcal{X}^{\mathsf{s}}} \langle \theta, x \rangle$.

$$\begin{aligned} \langle \theta - \gamma(\omega), x^* \rangle &= \sum_{i \in [d_{\mathsf{s}}]} (\theta_i - \omega_i \gamma_i) x^*_i = \\ &= \sum_{i \in [d_{\mathsf{s}}]} \theta_i x^*_i - \sum_{i \in [d_{\mathsf{s}}] : \gamma_i < 0} \omega_i \gamma_i x^*_i = \\ &= L_{\mathsf{s}}(\theta) - \sum_{i \in [d_{\mathsf{s}}] : \gamma_i < 0} \omega_i \gamma_i \end{aligned} \tag{13}$$

$\forall x \in \mathcal{X}, \forall x^* \in \operatorname{argmin}_{x \in X^{\mathsf{s}}} \langle \theta, x \rangle$:

$$\begin{aligned}\langle \theta - \gamma(\omega), x \rangle &= \sum_{i \in [d_{\mathsf{s}}]} (\theta_i - \omega_i \gamma_i) x_i = \\ &= \sum_{i \in [d_{\mathsf{s}}]} (\theta_i - \gamma_i) x_i + \sum_{i \in [d_{\mathsf{s}}]} (1 - \omega_i) \gamma_i x_i \geq \\ &\geq L_{\mathsf{s}}(\theta - \gamma) + \sum_{i \in [d_{\mathsf{s}}] : \gamma_i < 0} (1 - \omega_i) \gamma_i x_i = \\ &= \sum_{i \in [d_{\mathsf{s}}]} (\theta_i - \gamma_i) x_i^* + \sum_{i \in [d_{\mathsf{s}}]} (1 - \omega_i) \gamma_i x_i^* = \\ &= \langle \theta - \gamma(\omega), x^* \rangle \end{aligned} \tag{14}$$

Formula 14 proves Point 2 of Lemma 1. Formulas 13 and 14 together prove Point 1. □

Lemma 2. *Variables $(\gamma_u^{out}, \gamma_u'^{out})$, resp. $(\gamma_u^{in}, \gamma_u'^{in})$ in Algorithm 13 satisfy the requirements of Lemma 1 for each outflow resp. inflow subproblem of vertex u.*

Proof. Both Algorithms 5 and 6 output reparametrization variables that satisfy the requirements of Lemma 1. We have, for an outflow subproblem of node u:

$$\begin{aligned}\operatorname*{argmin}_{(y,y') \in \mathcal{X}_u^{out}} \langle \theta, (y, y') \rangle &\subseteq \operatorname*{argmin}_{(y,y') \in \mathcal{X}_u^{out}} \langle \theta - (0, \gamma_u'^{out}), (y, y') \rangle \\ &\subseteq \operatorname*{argmin}_{(y,y') \in \mathcal{X}_u^{out}} \langle \theta - (\gamma_u^{out}, \gamma_u'^{out}), (y, y') \rangle \end{aligned} \tag{15}$$

Therefore, also $(\gamma_u^{out}, \gamma_u'^{out})$ together satisfy the requirements of Lemma 1. Analogically, for the inflow subproblems. □

Costs in the new path and cut subproblems. One edge is typically shared among multiple newly added path and cut subproblems. Therefore, the available cost reparametrizations $\tilde{\theta}$ and $\tilde{\theta}'$ from Algorithm 13 must be redistributed to the newly added subproblems. We denote the set of all newly added path subproblems resp. cut subproblems in tightening iteration i by $\mathcal{P}^i$ resp. $\mathcal{C}^i$. For each base resp. lifted edge uv, we sum up the number of newly added path and cut subproblems that contain uv.

$$\begin{aligned} N_{uv} =& |\{P \in \mathcal{P}^i : uv \in P_E\}| + |\{C \in \mathcal{C}^i : uv \in C_E\}|, \\ N'_{uv} =& |\{P \in \mathcal{P}^i : uv \in P_{E'}\}| + |\{P \in \mathcal{P}^i : P \text{ is a } uv\text{-path}\}| \\ &+ |\{C \in \mathcal{C}^i : C \text{ is a } uv\text{-cut}\}| . \end{aligned} \tag{16}$$

Then, we define coefficient ω_{uv} resp. ω'_{uv} for each base edge $uv \in E$ resp. lifted edge $uv \in E'$ that belongs to a newly added subproblem as

$$\omega_{uv} = \frac{1}{N_{uv}}, \quad \omega'_{uv} = \frac{1}{N'_{uv}}. \tag{17}$$

Finally, for each newly added path subproblem P resp. cut subproblem C, we set the cost of base edge $uv \in E$ to $\theta^P_{uv} = \omega_{uv} \tilde{\theta}_{uv}$, resp. $\theta^C_{uv} = \omega_{uv} \tilde{\theta}_{uv}$. Analogically, for the lifted edges.

Cost update in in/outflow subproblems. If we use an edge uv for creating one or more path and cut suproblems, it is necessary to update its cost in the inflow subproblem of vertex v and the outflow subproblem of vertex u accordingly. For instance, we update the cost of base edge uv in the outflow subproblem of u as follows θ^{out}_{uv} $-\!\!=$ γ^{out}_{uv}. Where we adopt the notation from Algorithm 13. Note that $\tilde{\theta}_{uv} = \gamma^{in}_{uv} + \gamma^{out}_{uv}$. Therefore, the total cost of edge variable uv is preserved.

Proposition 2 (Guaranteed lower bound improvement of path subproblem)**.** *If a path subproblem corresponding to vw-path P separated by Algorithm 14 is added to the subproblem set $\mathcal{S}$, the guaranteed improvement of the global lower bound is $\min\{\min_{uv \in P_E} |\theta^P_{uv}|, \min_{uv \in P_{E'} \cup vw} |\theta'^P_{uv}|\}$, where θ^P is the reparametrized cost used for the path factor initialization.*

Proof. Algorithm 14 separates only those subproblems that contain exactly one lifted edge with cost $\theta'^P_{uv} > \varepsilon$ and the rest of the edges have cost lower than $-\varepsilon$. The reparametrized costs of the path factor are fractions of cost reparametrizations obtained by Algorithm 13. We have

$$\begin{aligned} &\forall uv \in P_E : \\ &\quad \theta^P_{uv} = \omega_{uv} \cdot (\gamma^{out}_{uv} + \gamma^{in}_{uv}), \\ &\quad \theta^{out}_{uv} \mathrel{-}= \omega_{uv} \cdot \gamma^{out}_{uv}, \, \theta^{in}_{uv} \mathrel{-}= \omega_{uv} \cdot \gamma^{in}_{uv}, \\ &\forall uv \in P_{E'} : \\ &\quad \theta'^P_{uv} = \omega'_{uv} \cdot (\gamma'^{out}_{uv} + \gamma'^{in}_{uv}), \\ &\quad \theta'^{out}_{uv} \mathrel{-}= \omega'_{uv} \cdot \gamma'^{out}_{uv}, \, \theta'^{in}_{uv} \mathrel{-}= \omega'_{uv} \cdot \gamma'^{in}_{uv} \end{aligned} \tag{18}$$

We evaluate the change of the lower bounds of all relevant inflow and outflow factors after reparametrization given by Formula 18. According to Lemma 1, we have

$$\begin{aligned} &\Delta L^{out} + \Delta L^{in} = \\ &- \sum_{uv \in P_E : \gamma^{out}_{uv} < 0} \omega_{uv} \cdot \gamma^{out}_{uv} - \sum_{uv \in P_{E'} \cup vw : \gamma'^{out}_{uv} < 0} \omega'_{uv} \cdot \gamma'^{out}_{uv} \\ &- \sum_{uv \in P_E : \gamma^{in}_{uv} < 0} \omega_{uv} \cdot \gamma^{in}_{uv} - \sum_{uv \in P_{E'} \cup vw : \gamma'^{in}_{uv} < 0} \omega'_{uv} \cdot \gamma'^{in}_{uv} \geq \\ &- \sum_{uv \in P_E : \gamma^{out}_{uv} + \gamma^{in}_{uv} < 0} \omega_{uv} \cdot (\gamma^{out}_{uv} + \gamma^{in}_{uv}) - \\ &- \sum_{uv \in P_{E'} \cup vw : \gamma'^{in}_{uv} + \gamma'^{out}_{uv} < 0} \omega'_{uv} \cdot (\gamma'^{out}_{uv} + \gamma'^{in}_{uv}) = \\ &- \sum_{uv \in P_E : \theta^P_{uv} < 0} \omega_{uv} \cdot \theta^P_{uv} - \sum_{uv \in P_{E'} \cup vw : \theta'^P_{uv} < 0} \omega'_{uv} \cdot \theta'^P_{uv} \end{aligned} \tag{19}$$

Let $kl \in P_{E'}$ be the only lifted edge with positive cost in the path subproblem. We set α =

$\min\{\min_{ij\in P_E}|\theta^P_{ij}|, \min_{ij\in P_{E'}\cup vw\setminus kl}|\theta'^P_{ij}|\}$ as in Algorithm 12. If we denote by ΔL^P the lower bound of the path subproblem, the global lower bound change after adding the path subproblem is:

$$\Delta L = \Delta L^{out} + \Delta L^{in} + \Delta L^P \tag{20}$$

If $\alpha < \theta'^P_{kl}$

$$\begin{aligned}\Delta L^{out} + \Delta L^{in} + \Delta L^P \geq \\ - \sum_{uv\in P_E:\theta^P_{uv}<0} \omega_{uv}\cdot\theta^P_{uv} - \sum_{uv\in P_{E'}:\theta'^P_{uv}<0} \omega'_{uv}\cdot\theta'^P_{uv} + \\ + \sum_{uv\in P_E:\theta^P_{uv}<0} \omega_{uv}\cdot\theta^P_{uv} + \sum_{uv\in P_{E'}:\theta'^P_{uv}<0} \omega'_{uv}\cdot\theta'^P_{uv} + \\ + \alpha = \alpha\end{aligned} \tag{21}$$

If $\alpha \geq \theta'^P_{kl}$

$$\begin{aligned}\Delta L^{out} + \Delta L^{in} + \Delta L^P \geq \\ - \sum_{uv\in P_E:\theta^P_{uv}<0} \omega_{uv}\cdot\theta^P_{uv} - \sum_{uv\in P_{E'}\cup vw:\theta'^P_{uv}<0} \omega'_{uv}\cdot\theta'^P_{uv} + \\ + \sum_{uv\in P_E:\theta^P_{uv}<0} \omega_{uv}\cdot\theta^P_{uv} + \sum_{uv\in P_{E'}:\theta'^P_{uv}} \omega'_{uv}\cdot\theta'^P_{uv} = \\ = \theta'^P_{kl}\end{aligned} \tag{22}$$

□

Proposition 3 (Guaranteed lower bound improvement of cut subproblem)**.** *If a subproblem corresponding to vw-cut C separated by Algorithm 18 is added to the subproblem set $\mathcal{S}$, the guaranteed improvement of the global lower bound is $\min\{\min_{uv\in C}\theta^C_{uv}, |\theta'^C_{vw}|\}$. Where θ^C is the reparametrized cost used for the cut factor initialization.*

Proof. We obtain the reparametrized cost θ^C for the cut subproblem analogically as in Formula 18 for the path subproblem. Note that Algorithm 18 ensures that all cut edges in the separated cut subproblem have positive cost and the lifted edge vw has negative cost. Using the same arguments as in the proof of Proposition 3, we obtain the lower bound change of inflow and outflow factors after separating the cut subproblem:

$$\begin{aligned}\Delta L^{out} + \Delta L^{in} = \\ - \sum_{uv\in C:\gamma^{out}_{uv}<0} \omega_{uv}\cdot\gamma^{out}_{uv} - \sum_{uv\in C:\gamma^{in}_{uv}<0} \omega'_{uv}\cdot\gamma^{in}_{uv} \\ - \omega'_{vw}\gamma'^{out}_{vw} - \omega'_{vw}\gamma'^{in}_{vw} \geq \\ - \sum_{uv\in C:\gamma^{out}_{uv}+\gamma^{in}_{uv}<0} \omega_{uv}\cdot(\gamma^{out}_{uv}+\gamma^{in}_{uv}) - \omega'_{vw}\gamma'^{out}_{vw} \\ - \omega'_{vw}\gamma'^{in}_{vw} = -\omega'_{vw}\gamma'^{out}_{vw} - \omega'_{vw}\gamma'^{in}_{vw} = -\theta'^C_{vw}\end{aligned} \tag{23}$$

Algorithm 3 shows how we obtain the lower bound of the cut subproblem. We set $\theta^C_{ij} = \text{argmin}_{uv\in C}\theta^C_{uv}$. If $\theta^C_{ij} < |\theta'^C_{vw}|$, we get the overall lower bound improvement

$$\Delta L^{out} + \Delta L^{in} + \Delta L^C \geq -\theta'^C_{vw} + \theta'^C_{vw} + \theta^C_{ij} = \theta^C_{ij}\,.$$

If $\theta^C_{ij} \geq |\theta'^C_{vw}|$, the lower bound of the cut subproblem is 0 and the overall lower bound improvement is

$$\Delta L^{out} + \Delta L^{in} + \Delta L^C \geq -\theta'^C_{vw}\,. \tag{24}$$

□

8.7. Message Passing

One solver run consists of subproblems initialization and a number of message passing iterations. Algorithm 19 details the whole run. Algorithms 23-22 present methods that are called within one iteration.

The number of iterations is predetermined by an input parameter. We use typically tens or maximally one hundred iterations in our experiments.

Algorithm 19 sends in each iteration, messages between all subproblems in the subproblem set $\mathcal{S}$. Each subproblem creates messages to be sent by computing min-marginals of its variables (see Formula (5)). These min-marginals are re-scaled and redistributed between other subproblems that contain the respective variables. These operations are called reparametrization. See Section 4 for details.

Algorithm 23 computes lower bound of the LDP objective by summing up lower bounds of all subproblems. The cost reparametrization realized via our message passing procedures ensures that the lower bound is non-decreasing during the computaiton.

Algorithm 20 shows sending messages from the inflow subproblem of node u. Algorithm 21 shows sending messages from a path subproblem. Algorithm 22 presents sending messages from a cut subproblem.

Algorithm 19 Message Passing

Input Graphs $G = (V, E)$ and $G' = (V, E')$, costs $\theta \in \mathbb{R}^{V \cup E \cup E'}$

Output Best found primal solution $(z, y, y')^{\mathsf{ub}}$, lower bound LB

Initialization:
for $v \in V$ **do**
  Add inflow subproblem for node v.

$$\forall uv \in \delta^-_E(v): \quad \theta^{in}_{uv} = \begin{cases} \theta_{uv} & \text{if } v = s \\ \frac{1}{2}\theta_{uv} & \text{otherwise} \end{cases}$$

  $\forall uv \in \delta^-_{E'}(v)$: $\theta'^{in}_{uv} = \frac{1}{2}\theta'_{uv}$.
  $\theta^{in}_v = \frac{1}{2}\theta_v$.
  Add outflow subproblem for node v with analogu-ous costs.
end for
$\mathcal{C} = \emptyset$
$\mathcal{P} = \emptyset$
Lagrange decomposition optimization
for iter $= 1, \dots,$ max_iter **do**
  Forward Pass:
  for $u = u_1, \dots, u_{|V|}$ **do**
    Inflow-Subproblem-Message-Passing(u)
    Outflow-Subproblem-Message-Passing(u)
  end for
  for $P \in \mathcal{P}$ **do**
    Path-Subproblem-Message-Passing(P)
  end for
  for $C \in \mathcal{C}$ **do**
    Cut-Subproblem-Message-Passing(C)
  end for
  Backward Pass:
  Revert order of nodes and perform above iteration.
  if iter $\equiv 0 \mod k$ **then**
    Separate-Cut-Subproblem(ε)
    Separate-Path-Subproblems(ε)
    Add cut and path subproblems to $\mathcal{C}$ and $\mathcal{P}$
  end if
  if iter $\equiv 0 \mod l$ **then**
    $(z, y, y') =$Compute-Primal$(\mathcal{S}, \theta)$
    if $\langle \theta, (z, y, y') \rangle < \langle \theta, (z, y, y')^{\mathsf{ub}} \rangle$ **then**
      $(z, y, y')^{\mathsf{ub}} = (z, y, y')$
    end if
  end if
  $LB =$Lower-Bound
end for

Algorithm 20 Inflow-Subproblem-Message-Passing

Input central vertex u of the subproblem

$\gamma'^{in} =$ All-Lifted-MM-In(u, θ^{in}).
$\omega = 1$
for $vu \in \delta^-_E(u), P \in \mathcal{P} : vu \in P_E$ **do**
  $\gamma^P_{vu} =$ Path-Base-Min-Marginal(u, v, θ^{in})
  $\omega \mathrel{+}= 1$
end for
for $vu \in \delta^-_{E'}(u), P \in \mathcal{P} : vu \in P_{E'}$ **do**
  $\gamma'^P_{vu} =$ Path-Lifted-Min-Marginal(u, v, θ^{in})
  $\omega \mathrel{+}= 1$
end for
for $vu \in \delta^-_{E'}(u), P \in \mathcal{P} : P \in vu$-paths$(G)$ **do**
  $\gamma'^P_{vu} =$ Path-Lifted-Min-Marginal(u, v, θ^{in})
  $\omega \mathrel{+}= 1$
end for
for $vu \in \delta^-_E(u), C \in \mathcal{C} : vu \in C_E$ **do**
  $\gamma^C_{vu} =$ Cut-Base-Min-Marginal(u, v, θ^{in})
  $\omega \mathrel{+}= 1$
end for
for $vu \in \delta^-_{E'}(u), C \in \mathcal{C} : C$ is a vu-Cut **do**
  $\gamma'^C_{vu} =$ Cut-Lifted-Min-Marginal(u, v, θ^{in})
  $\omega \mathrel{+}= 1$
end for
for $vu \in \delta^-_{E'}(u)$ **do**
  $\theta^{in}_{vu} \mathrel{-}= \frac{1}{\omega} \cdot \gamma'^{in}_{vu}, \quad \theta^{out}_{vu} \mathrel{+}= \frac{1}{\omega} \cdot \gamma'^{in}_{vu}$
end for
for $vu \in \delta^-_E(u), P \in \mathcal{P} : vu \in P_E$ **do**
  $\theta^{in}_{vu} \mathrel{-}= \frac{1}{\omega} \cdot \gamma^P_{vu}, \quad \theta^P_{vu} \mathrel{+}= \frac{1}{\omega} \cdot \gamma^P_{vu}$
end for
for $vu \in \delta^-_{E'}(u), P \in \mathcal{P} : vu \in P_{E'}$ **do**
  $\theta'^{in}_{vu} \mathrel{-}= \frac{1}{\omega} \cdot \gamma'^P_{vu}, \quad \theta'^P_{vu} \mathrel{+}= \frac{1}{\omega} \cdot \gamma'^P_{vu}$
end for
for $vu \in \delta^-_{E'}(u), P \in \mathcal{P} : P \in vu$-paths$(G)$ **do**
  $\theta'^{in}_{vu} \mathrel{-}= \frac{1}{\omega} \cdot \gamma'^P_{vu}, \quad \theta'^P_{vu} \mathrel{+}= \frac{1}{\omega} \cdot \gamma'^P_{vu}$
end for
for $vu \in \delta^-_E(u), C \in \mathcal{C} : vu \in C_E$ **do**
  $\theta^{in}_{vu} \mathrel{-}= \frac{1}{\omega} \cdot \gamma^C_{vu}, \quad \theta^C_{vu} \mathrel{+}= \frac{1}{\omega} \cdot \gamma^C_{vu}$
end for
for $vu \in \delta^-_{E'}(u), C \in \mathcal{C} : C$ is a vu-Cut **do**
  $\theta'^{in}_{vu} \mathrel{-}= \frac{1}{\omega} \cdot \gamma'^C_{vu}, \quad \theta'^C_{vu} \mathrel{+}= \frac{1}{\omega} \cdot \gamma'^C_{vu}$
end for

Algorithm 21 Path-Subproblem-Message-Passing

Input: uv-Path $P \in \mathcal{P}$

$\gamma^P = \text{Path-Min-Marginals}(P, \theta^P)$
$\omega^P = \frac{1}{2 \cdot |P_E| + 2 \cdot |P_{E'}|}$
for $kl \in P_E$ **do**
  $\theta^P_{kl} \mathrel{-}= \gamma^P_{kl}$
  $\theta^{in}_{kl} \mathrel{-}= \omega^P \cdot \gamma^P_{kl}, \quad \theta^{out}_{kl} \mathrel{-}= \omega^P \cdot \gamma^P_{kl}$
end for
for $kl \in P_{E'}$ **do**
  $\theta^P_{kl} \mathrel{-}= \gamma^P_{kl}$
  $\theta^{in}_{kl} \mathrel{-}= \omega^P \cdot \gamma^P_{kl}, \quad \theta^{out}_{kl} \mathrel{-}= \omega^P \cdot \gamma^P_{kl}$
end for

Algorithm 22 Cut-Subproblem-Message-Passing

Input: uv-Cut $C \in \mathcal{C}$

$\gamma^C = \text{Cut-Min-Marginals}(C, \theta^C)$
$\omega^C = \frac{1}{2 \cdot |C_E| + 2}$
for $kl \in C_E$ **do**
  $\theta^C_{kl} \mathrel{-}= 2\omega^C \cdot \gamma^C_{kl}$
  $\theta^{in}_{kl} \mathrel{+}= \omega^C \cdot \gamma^C_{kl}, \quad \theta^{out}_{kl} \mathrel{+}= \omega^C \cdot \gamma^C_{kl}$
end for
$\theta'^C_{uv} \mathrel{-}= 2\omega^C \cdot \gamma'^C_{uv}$
$\theta^{in}_{uv} \mathrel{+}= \omega^C \cdot \gamma'^C_{uv}, \quad \theta^{out}_{uv} \mathrel{+}= \omega^C \cdot \gamma'^C_{uv}$

Algorithm 23 Lower-Bound

Input Subproblems $\mathcal{S}$
Output Lower bound value LB

$LB = 0$
for $u \in V \backslash \{s, t\}$ **do**
  $LB \mathrel{+}= \text{Opt-In-Cost}(u, \theta^{in})$
  $LB \mathrel{+}= \text{Opt-Out-Cost}(u, \theta^{out})$
end for
for $P \in \mathcal{P}$ **do**
  $LB \mathrel{+}= \text{Path-Subproblem-Optimization}(P, \theta^P)$
end for
for $C \in \mathcal{C}$ **do**
  $LB \mathrel{+}= \text{Cut-Subproblem-Optimization}(C, \theta^C)$
end for

8.8. Primal Solution and Local Search

Algorithm 24 summarizes the whole procedure for obtaining a primal solution. As stated in Section 4.6, we obtain an initial primal solution by solving MCF problem.

Given a feasible solution of the LDP, Algorithm 25 improves it by splitting and merging paths. While we obtain the costs for MCF from base and lifted edges costs in inflow and outflow factors (Algorithm 4), the local search procedure uses original input costs of base and lifted edges.

Algorithm 28 finds candidate split point of each path and recursively splits the path if the split leads to decrease of the objective function.

For each vertex of each path, function $split$ evaluates the cost of splitting the path after the vertex:

$$\forall v_j \in P_V = (v_1, \dots v_n): \tag{25}$$
$$split(v_j, P) = - \sum_{\substack{k \le j, l > j, \\ v_k v_l \in E'}} \theta'_{v_k v_l} - \theta_{v_j v_{j+1}} + \theta_{s v_{j+1}} + \theta_{v_j t}$$

The second step of the primal solution post-processing by Algorithm 25 is merging paths. Before the path merging itself, some candidate pairs of paths need to be shortened at their ends in order to enable their feasible merging.

Algorithm 26 identifies pairs of those paths whose merging should lead to objective improvement but that cannot be connected directly due to missing base edge between their endpoints. In order to identify the desired paths pairs, several functions are used.

Function $l^+(P_1, P_2)$ resp $l^-(P_1, P_2)$ is the sum of positive resp. negative lifted edges from path P_1 to path P_2. Function $l(P_1, P_2)$ sums all lifted edges from P_1 to P_2.

$$\begin{aligned} &\forall P_1, P_2 \in \mathcal{P} \\ &\quad l^+(P_1, P_2) = \sum_{uv \in E' : u \in P_1, v \in P_2, \theta'_{uv} \ge 0} \theta'_{uv} \\ &\quad l^-(P_1, P_2) = \sum_{uv \in E' : u \in P_1, v \in P_2, \theta'_{uv} < 0} \theta'_{uv} \\ &\quad l(P_1, P_2) = l^+(P_1, P_2) + l^-(P_1, P_2) \end{aligned} \tag{26}$$

We use the above values in functions $merge$ and $merge_\tau$ that evaluate the gain of merging two paths. Threshold $\tau \le 1$ constraints the ratio between the positive and the negative part of lifted cost function l that is considered acceptable for merging two paths.

$$\forall P_1 = (v_1, \dots, v_n), P_2 = (u_1, \dots, u_m) \in \mathcal{P} \tag{27}$$
$$merge(P_1, P_2) = \begin{cases} \theta_{v_n u_1} + l(P_1, P_2) & \text{if } v_n u_1 \in E \\ \infty & \text{otherwise} \end{cases}$$

$$\begin{aligned} &\forall P_1 = (v_1, \dots, v_n), P_2 = (u_1, \dots, u_m) \in \mathcal{P} \\ &merge_\tau(P_1, P_2) = \\ &\quad = \begin{cases} \infty & \text{if } v_n u_1 \notin E \vee \\ & \quad l^+(P_1, P_2) > \tau |l^-(P_1, P_2)| \\ \theta_{v_n u_1} + l(P_1, P_2) & \text{otherwise} \end{cases} \end{aligned} \tag{28}$$

Algorithm 27 is applied on all paths pairs found by Algorithm 26. It inspects whether shortening of one or both paths leads to a feasible connection that ensures a desired objective improvement. It iteratively removes either the last vertex of the first path or the first vertex of the second path and checks if a connection is possible and how much it costs.

The last part of Algorithm 25 considers merging paths. We use formula $merge_\tau(P_i, P_j) - out(P_i) - in(P_j)$ to evaluate whether merging two paths is beneficial. Here $in(P_j)$ denotes input cost to the first vertex of P_j and $out(P_i)$ denotes output cost from the last vertex of P_i. We state the full formula just for completeness. We set the input and the output costs to zeros in our experiments. Using $merge_\tau$ ensures that we connect the paths only if the ratio between the positive lifted cost l^+ and negative lifted cost l^- between the paths is below the acceptable threshold.

Algorithm 24 Compute-Primal

Input Subproblems $\mathcal{S}$, original costs $\theta \in \mathbb{R}^{|V'|+|E|+|E'|}$
Output Primal solution (z, y, y')

Init-MCF
Obtain primal solution of MCF $y^{mcf} \in \{0,1\}^{E^{mcf}}$
Set (z, y) accroding to y^{mcf}
$y' =$Adjust-Lifted-Solution(z, y)
$(z, y, y') =$Local-Search(z, y, y')

Algorithm 25 Local-Search

Input Input primal solution z, y, y'
Output Improved primal solution z, y, y'

Obtain set of disjoint paths $\mathcal{P} = \{P_1, \dots, P_n\}$ from y
for all $P \in \mathcal{P}$ **do**
    $\mathcal{P} =$Check-Path-Split$(P_i, \mathcal{P})$
end for
$\mathcal{P} =$Shorten-For-Merge$(\mathcal{P})$
while $true$ **do**

$$(P_1, P_2) = \operatorname*{argmin}_{(P_i,P_j)\in\mathcal{P}\times\mathcal{P}} merge_\tau(P_i, P_j) - out(P_i) - in(P_j)$$

    if $merge_\tau(P_1, P_2) - out(P_1) - in(P_2) < 0$ **then**
        $\mathcal{P} =$ Merge-Paths$(P_1, P_2, \mathcal{P})$
    else
        break
    end if
end while
(z, y, y')=Set-From-Paths$(\mathcal{P})$

Algorithm 26 Shorten-For-Merge

Input Set of paths $\mathcal{P}$
Output Updated set of paths $\mathcal{P}$

for all $P_1 = (v_1, \dots, v_n) \in \mathcal{P}$ **do**
    $P = \operatorname*{argmin}_{P_2=(u_1,\dots,u_m)\in\mathcal{P}:v_nu_1\in E} merge(P_1, P_2)$
    $P' = \operatorname*{argmin}_{P_2=(u_1,\dots,u_m)\in\mathcal{P}:v_nu_1\notin E} l(P_1, P_2)$
    if $l(P_1, P') < merge(P_1, P) \wedge l(P_1, P') < 0$ **then**
        $P^* = P'$, $c = l(P_1, P')$
    else
        $P^* = P$, $c = merge(P_1, P)$
    end if
    if pred$[P^*] = \emptyset \vee$ score$[P^*] > c$ **then**
        pred$[P^*] = P_1$, score$[P^*] = c$
    end if
end for
for all $P_2 = (u_1, \dots, u_m) \in \mathcal{P}$ **do**
    if pred$[P_2]$=$P_1 = (v_1, \dots, v_n) \wedge v_nu_1 \notin E$ **then**
        $\mathcal{P} =$Cut-Ends$(P_1, P_2, \mathcal{P})$
    end if
end for

Algorithm 28 Check-Path-Split

Input Input path P, set of all paths $\mathcal{P}$
Output Set of paths $\mathcal{P}$

$v_m = \operatorname{argmax}_{v_j\in P_V} split(v_j, P)$
if $split(v_m, P) < 0$ **then**
    $(P_1, P_2) =$Split-Path(P, v_m)
    $\mathcal{P}$.remove(P), $\mathcal{P}$.insert(P_1), $\mathcal{P}$.insert(P_2)
    $\mathcal{P} =$Check-Path-Split$(P_1, \mathcal{P})$
    $\mathcal{P} =$Check-Path-Split$(P_2, \mathcal{P})$
end if
return $\mathcal{P}$

Algorithm 27 Cut-Ends

Input $P_1 = (v_1, \ldots, v_m), P_2 = (u_1, \ldots, u_m), \mathcal{P}, i_{max}$
Output New set of paths $\mathcal{P}$

$c_1 = \infty, c_2 = \infty$
while $i_1 + i_2 < i_{max}$ **do**
  $P_1' = (v_1, \ldots, v_{n-i_1}), P_2' = (u_{1+i_2}, \ldots, u_m)$
  $P_1'' = (v_1, \ldots, v_{n-i_1-1}), P_2'' = (u_{2+i_2}, \ldots, u_m)$
  if $merge(P_1', P_2'') + merge(P_1'', P_2') < \infty$ **then**
    $\alpha_1 = merge(P_1', P_2'') + split(P_1, v_{n-i_1}) + split(P_2, u_{1+i_2})$
    $\alpha_2 = merge(P_1'', P_2') + split(P_1, v_{n-i_1-1}) + split(P_2, u_{i_2})$
    if $\alpha_1 < \alpha_2$ **then** $c_1 = i_1 - 1, c_2 = i_2$
    else $c_1 = i_1, c_2 = i_2 - 1$
    break
  else if $merge(P_1', P_2'') < \infty$ **then**
    $c_1 = i_1 - 1, c_2 = i_2$
    break
  else if $merge(P_1'', P_2') < \infty$ **then**
    $c_1 = i_1, c_2 = i_2 - 1$
    break
  else
    $\alpha_1 = l(P_1', P_2'') + split(P_1, v_{n-i_1}) + split(P_2, u_{1+i_2})$
    $\alpha_2 = l(P_1'', P_2') + split(P_1, v_{n-i_1-1}) + split(P_2, u_{i_2})$
    if $\alpha_1 < \alpha_2$ **then** i_2 ++
    else i_1 ++
  end if
end while
if $c_1 \neq \infty \wedge c_2 \neq \infty$ **then**
  $P_1' = (v_1, \ldots, v_{n-c_1}), P_2' = (u_{1+c_2}, \ldots, u_m)$
  if $merge_\tau(P_1', P_2') < \infty$ **then**
    if $c_1 > 0$ **then**
      (P_{11}, P_{12}) =Split-Path(P_1, v_{n-c_1})
      $\mathcal{P}$.remove(P_1)
      $\mathcal{P}$.insert(P_{11}), $\mathcal{P}$.insert(P_{12})
    end if
    if $c_2 > 0$ **then**
      (P_{21}, P_{22}) =Split-Path(P_2, u_{c_2})
      $\mathcal{P}$.remove(P_2)
      $\mathcal{P}$.insert(P_{21}), $\mathcal{P}$.insert(P_{22})
    end if
  end if
end if
return $\mathcal{P}$

8.9. Global Context Normalization

Our tracking system employs a global context normalization to obtain accurate features between detections (see Section 5.1). This section elaborates the implementation details.

Global context normalization puts similarity measurements into global perspective to form more meaningful feature values. For instance, global illumination changes will likely decrease measured appearance similarities between any pair of detections. Likewise, a scene where all people are far away from the camera will most likely result in less confident appearance similarity measurements. In both cases, positive matching pairs should have higher similarities than negative matching pairs but the absolute similarity values are reduced by the global effects. These and further effects make the interpretation of the similarity in absolute terms less meaningful. Global context normalization compensates such effects.

To this end, let $\Omega_k = \{\sigma_{vw,k} : vw \in E\}$ comprise all computed similarity measurements for feature $k \in \{\text{Spa}, \text{App}\}$ defined in Section 5.1. For each feature k and similarity measurement $\sigma_{vw,k} \in \Omega_k$, we define sets $\text{GC}_{i,k}$ with $i \in [5]$. Each set $\text{GC}_{i,k}$ induces two global context normalization features: $\sigma_{vw,k} \cdot \max(\text{GC}_{i,k})^{-1}$ and $\sigma_{vw,k}^2 \cdot \max(\text{GC}_{i,k})^{-1}$.

The sets are defined w. r. t. $\sigma_{vw,k} \in \Omega_k$ as follows:

$$\text{GC}_{1,k} = \{s_{vn,k} \in \Omega_k : n \in B\}. \tag{29}$$
$$\text{GC}_{2,k} = \{s_{mw,k} \in \Omega_k : m \in B\}. \tag{30}$$
$$\text{GC}_{3,k} = \{s_{vn,k} \in \Omega_k : n \in B \text{ and } f_n = f_w\}. \tag{31}$$
$$\text{GC}_{4,k} = \{s_{nw,k} \in \Omega_k : n \in B \text{ and } f_n = f_v\}. \tag{32}$$
$$\text{GC}_{5,k} = \{s_{mn,k} \in \Omega_k : m, n \in B\}. \tag{33}$$

where f_x denotes the frame of detection x and B the batch as defined in section 5.1. The sets $\text{GC}_{1,k}$ and $\text{GC}_{2,k}$ result in a normalization of the similarity score $\sigma_{vw,k}$ over all outgoing or incoming edges to v or w, respectively. The set $\text{GC}_{3,k}$ results in a normalization over all similarity scores for outgoing edges from v to a detection in frame f_w. Analogously, the set $\text{GC}_{4,k}$ collects all edges from a detection of frame f_v to node w. Finally, $\text{GC}_{5,k}$ normalizes the similarity score over all existing scores in the batch B.

8.10. Multi Layer Perceptron (MLP)

As reported in Section 5.1 we use a lightweight and scalable MLP to obtain edge costs. We use multiple instances of the same MLP structure. Each MLP is trained on edges that have a specific range of temporal gaps (more details in Section 5.1).

The MLP architecture is based on two fully connected (FC) layers. The input is a 22-dimensional vector (features with corresponding global context normalizations). LeakyReLU activation [43] is used for the first FC layer. The final layer (FC) has one neuron, whose output represents the cost value. For training, an additional sigmoid activation is added. The structure of the neural network is visualized in Figure 3.

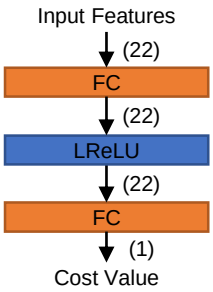

Figure 3. Visualisation of the proposed neural MLP. FC denotes fully connected layers, LReLU denotes LeakyReLU activation [43] and values in parenthesis denote the dimension of the corresponding tensors.

8.11. Batch Creation Using Fixed Frame Shifts

In this section, we elaborate on our batch creation method using fixed frame shifts (see Paragraph *Training* of Section 5.1).

A batch B contains a set of edges with corresponding edge costs. Important to note is that we use the same batch creation strategy to form batches for training as well as inference. During training, batches are augmented with corresponding ground truth labels. A carefully chosen batch creation strategy is crucial to ensure accurate and scalable training and inference.

In order to obtain accurate predictions by our MLPs (Section 5.1), the distribution of a batch should represent the characteristics of the training data. In particular, a batch should comprise edges covering all permissible temporal gaps between detections. Furthermore, the distribution of a batch influences the costs of all edges contained in the batch by the global context normalization (Section 8.9). Thus also during inference, a batch should comprise edges covering all permissible temporal gaps between detections. It is thus important to employ the same batch creation strategy for training and inference.

A naïve strategy is thus to define a batch on all frames within a range f up to $f+\Delta f_{\text{max}}$, where f is a starting frame and Δf_{max} defines the maximal permissible time gap. During training, one could then sample detections from these frames and randomly create true positive and false positive edges. However, such an approach is not tractable during inference for long sequences with many detections and long time gaps. To see this, consider the sequence MOT20-05 of the MOT20 dataset [16]. It contains 3315 frames with 226 detections per frame on average. We use a maximal permissible time gap of 2 seconds, which correspond to 50 frames for sequence MOT20-05. The number of edges per batch and for the entire sequence can then be roughly estimated[4] with $63 \cdot 10^6$ and $4 \cdot 10^9$, respectively, which is intractable.

To decrease the amount of edges per batch while ensuring that batches consists of samples containing all permissible temporal gaps, we adapt batch creation to our needs: For each start frame f of a batch, we subselect the frames to be considered within the range $f, \dots, f_{\text{max}}$. During training, we then subsample detections from these frames. During inference, we utilize all detections of these frames to form our batch.

To this end, we define a sequence of frame shifts that is used to create the frame subselection. Using only few frame shift makes the approach more computationally efficient. Yet we must ensure that all edges are computed at least once during inference. That is if $B(f)$ denotes the batch created according to our strategy with starting frame f, then $\bigcup_f B(f) = E$ must contain all edges.

To ensure that we always cover temporal gaps of up to 2 seconds, the frame shifts depend on the maximal permissible temporal gaps (measured in frames).

For a start frame f and $\Delta f_{\text{max}} = 50$, we define the frame shift set $\text{Sh}(\Delta f_{\text{max}})$ as

$$\text{Sh}(50) = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 17, 26, 35, 44, 50\}\,. \quad (34)$$

For $\Delta f_{\text{max}} = 60$, we define the frame shift set $\text{Sh}(\Delta f_{\text{max}})$ as

$$\begin{aligned}\text{Sh}(60) = \{\,&0, 1, 2, 3, 4, 5, 6, 7, 14, 21, 28, 36, 37,\\ &38, 40, 46, 53, 60\,\}\end{aligned} \quad (35)$$

Then for each start frame f, a batch is created using the frames $\{f + f_{\text{shift}} : f_{\text{shift}} \in \text{Sh}(f_{\text{max}})\}$. To ensure that all edges are computed at least once in the inference stage, one need to calculate batches with start frames $f \in \{1 - \Delta f_{\text{max}}, \dots, \Delta f_{\text{max}}\}$. Compared to the naïve batch creation strategy, our utilized batch creation results in $226 \sum_{i=1}^{13} 226 \cdot (14 - i) = 4647916$ edges for MOT20-20, using the same assumptions as before. The number of edges to be computed is thus significantly lower.

8.12. Determining obviously matching and non-matching detection pairs

During the graph construction in Section 5.2, we employ a simply strategy to detect edges that represent obviously matching or obviously non-matching detection pairs. Corresponding edge costs are set such that they induce must-links or cannot-links as soft constraints. Details are described in this section.

Obviously non-matching pairs. We use optical flow and the object size to calculate the maximal plausible

[4] With 226 detections per frame, there are about $226 \sum_{i=1}^{49} 226(50 - i) = 62568100$ edges per batch. With 3315 frames there are more than $\frac{3315}{50} 62568100 = 4148265030$ edges. Here we assumed non-overlapping batches, thereby missing many connections.

displacement $d_{\max}(v,w)$ and velocities $\mathrm{v}_{x,\max}(v,w)$ and $\mathrm{v}_{y,\max}(v,w)$ between two detections v and w. If v is a detection in frame f_v and w a detection in f_w, we define

$$d_{\max}(v,w) = k_d + \sum_{i=f_v}^{f_w-1} \max(O(i,i+1)) \tag{36}$$

where $\max(O(i,i+1))$ is the maximal magnitude of the optical flow between the frames i and $i+1$ and k_d is a security tolerance, which we set to 175 pixel according to experiments on the training data. If the distance $d(v,w)$ between the center points of detections v and w is greater than $d_{\max}(v,w)$, the detection pair given by vw is regarded as obviously non-matching. We also assume that the maximal velocity of a person is limited. With the height h and width b of the bounding boxes, we define the maximal velocities

$$\mathrm{v}_{x,\max}(v,w) = b \cdot k_e \tag{37}$$

and

$$\mathrm{v}_{y,\max}(v,w) = h \cdot \frac{k_e}{2} \tag{38}$$

in x and y-direction. The factor k_e is set to $k_e = 0.3$ according to experiments on the training data. In sequences with moving cameras, the factor is increased to $k_e = 0.8$ and decreased to $k_e = 0.12$ in sequences with static camera and aerial viewpoint. If the velocity $\mathrm{v}_x(v,w)$ or $\mathrm{v}_y(v,w)$ between the detections v and w is greater than corresponding $\mathrm{v}_{x,\max}(v,w)$ or $\mathrm{v}_{y,\max}(v,w)$, the connection given by vw is regarded as obviously non-matching. To avoid wrong interpretations caused by noise in the velocity calculation, we set k_e to a high value for detection pairs with small temporal distances.

If a detection pair vw is regarded as obviously non-matching, we induce a cannot-link soft constraint on vw by setting its costs to a negative value with a high absolute value, *i.e.* $c_{vw} \ll 0$.

Obviously matching pairs. We induce must-link soft constraints on edges, considering only connections between consecutive frames.

An edge $vw \in E$ with an appearance similarity score $\sigma_{vw,\mathrm{App}}$ close to the maximal achievable score (which is 2 in our implementation) is considered an obviously matching pair. We infer from the training data $k_s = 1.95$ as threshold, so that edges with $\sigma_{vw,\mathrm{App}} > k_s$ are regarded as obviously matching by setting their costs accordingly.

In addition, if two boxes between consecutive frames have a high overlap, we induce a must-link soft constraint on the corresponding edge. In more detail, the intersection over union between the detections must be at least 0.5. However, such spatial measurements are affected by camera motions, thus potentially leading to wrong interpretations. In order to induce link soft-constraints only in confident cases, we employ a simple camera motion compensation beforehand. To this end, we calculate for a considered edge the mean magnitude given by the optical flow between the frames of the respective detections. Before we compute the intersection over union, we translate one of the boxes in horizontal direction by the approximated camera motion, if this decreases the intersection over union. This procedure lowers the likelihood of creating wrong must-links caused by camera motion. Camera motion compensation needs to be applied only to sequences filmed from a non-static camera. Optical flow can be used to detect if a scene has a static camera setup. Note that MOT20 contains only scenes filmed from a static camera.

Table 4. Results with and w/o post-processing on MOT20 train set.

	MOTA↑	IDF1↑	TP↑	FP↓	FN ↓	IDS ↓
w	**74.4**	62.8	**863203**	15778	**271411**	**3511**
w/o	72.3	**63.6**	833473	**8462**	301141	4201

8.13. Inference

8.13.1 Interval Solution

This section explains how we solve MOT20 using solutions of its intervals. First, we solve the problem in independent subgraphs containing detections and edges from time intervals $[il+1,(i+1)l]$ for $i \in \{0,1\ldots,n\}$, where $l = 3 \cdot t_{\max}$, and $t_{\max}$ is the maximum temporal edge length. We fix resulting trajectories in the centres of intervals, namely in time intervals $[il+t_{max}+1,(i+1)l-t_{max}]$ for $i \in \{1,\ldots,n-1\}$. Second, we solve the problem in time intervals covering the end of one initial interval and the beginning of the subsequent interval while allowing connections with the fixed trajectory fragments. The cost of a connection between a detection and a trajectory fragment is obtained as the sum of costs between the unassigned detection and the detections within the trajectory fragment.

8.13.2 Post-Processing

We perform post-processing on the result provided by our solver. As it is common, we recover missing detections within a computed trajectory using linear interpolation. We also correct wrong connections that mostly stem from situations which currently cannot be correctly resolved by current features, independent of the tracking system, *e.g.* pairwise features computed over very long temporal gaps and ambiguous feature information due to multiple people appearing within one detection box.

Consequently, we apply these strategies only to MOT20. As soon as one of these methods detects a connection as false, the corresponding trajectory is split into two new trajectories. Table 4 shows the effect of the post-processing on MOT20 train set.

We noticed an accumulation of wrong connections, where one end of a trajectory (*i.e*. its first or last detection) is connected to the successive detection using a skip-connection over a long temporal gap, and the connection is wrong. This might be explained by a combination of misleading visual features (*e.g*. caused by partial occlusion), not very informative spatial features (due to the high temporal gap) and missing lifted edges, because only one detection is existent at the end of the trajectory. To keep only reliable connections, we split trajectories if only one detection is existing at the start or end of a trajectory, followed by a temporal gap of at least 10 frames.

We also handle cases at the borders of a tracking scene. If a person leaves the scene, and another person enters the scene at a position close by after a short time, the tracking system sometimes joins the trajectories of the two persons. We explain this behaviour by the high visual similarity between partially visual persons at image borders. If a person leaves a scene, normally just one leg, one arm or the head is visible for some frames. However, a single body party is not very discriminative and thus can look similar to a body part of another person. In addition, the spatio-temporal information will indicate a likely match in this scenario. Due to the temporal gap, no or not many meaningful lifted edges are existing which could give contradicting signals. To eliminate this kind of errors, we split trajectories between two detections, if the temporal gap is greater or equal to 10 and both detections are located at the image border.

For all detections which are connected over a temporal time gap greater or equal to 10 frames (skip edges), we perform a motion sanity check and split the corresponding trajectory if its motion is not plausible. To this end, we first determine the highest velocity of obviously correct trajectories, or parts of trajectories with a minimal length of 10 frames (to avoid random noise issues). Then, we split connections at these skip edges, if their velocity is higher than the determined velocity.

Additionally, we verify that motion between trajectory parts are consistent. To this end, we consider the motion described by a trajectory, using the part before a connection, and compare it with the resulting motion described by the trajectory, using the part after the connection. If the velocities differ by a factor of 5 or greater or if the angle differs more than $\pi/2$, the trajectory is split.

8.14. Solver Runtime

Our solver can compute one interval of MOT20 (150 frames) or an entire sequence of MOT17 with less than 20GB RAM, using a single CPU core.

Subsequently, we analyze the runtime in detail, by performing a theoretical analysis in Section 8.14.1, followed by a comparison with an existing LDP solver in Section 8.14.2.

8.14.1 Computational Complexity

The solver terminates if one of the following conditions is satisfied. Either the lower bound is equal to the objective value of the best primal solution, i.e. optimum has been found. Or the maximum number of message passing iterations has been reached. The optimum was not found in our experiments, so the letter condition applied.

The runtime of the solver is, therefore, determined by the input parameter denoting the maximum number of iterations. The dependence on number of iterations is not exactly linear because the problem size grows with the number of path and cut subproblems added to set of subproblems $\mathcal{S}$ via cutting plane separation (see Sections 8.4 and 8.5).

An overview of the whole solver run and the tasks performed within one its iteration is given in Section 8.7. The runtime of the tasks is given by the runtime of computing min-marginals of the subproblems.

We discuss the complexity of the used algorithms in the paragraphs bellow. They all have a polynomial complexity. Therefore, the overall runtime of the solver is polynomial too.

In order to compute messages between the inflow and the outflow subproblem, we apply Algorithm 6. Min-marginals for messages between the path subproblems and the in/outflow subproblems are obtained for one shared variable at the time. The same holds for exchanging messages between the cut subproblems and the in/outflow subproblems. This is done by calling restricted versions of optimization algorithms of the path and cut subproblems (Algorithms 12 and 3). For in/outflow subproblems, we use one call of Algorithm 1 followed by either Algorithm 9 or Algorithm 11 limited to single variable reparametrization.

Messages between inflow and outflow subproblems. Messages between inflow and outflow subproblems are realized on lifted edge variables by calling Algorithm 6. Many subroutines employ full or partial DFS on all nodes reachable from the central node within the relevant time gap. In these cases, we use precomputed node order instead of complete DFS as described in the last paragraph of Section 8.2. One call of the full DFS (Algorithms 1 and 11) requires to process all vertices reachable from v within maximal time gap for edge length (Δf_{max}). This comprises L_{max} video frames (we use $L_{max} = 50$ or 60). Let us denote by n the maximum number of detections in one frame. The complete DFS processes maximally nL_{max} vertices. Incomplete DFS used in Algorithm 9 processes in each step vertices in L layers. In the worst case, this is done for all relevant layers $L = 1, \dots, L_{max}$. Processing one vertex requires to check its neighbors in the base graph. Their amount is bounded by KL_{max} where $K = 3$. See Sparsification paragraph in Section 5. Putting all together, the complexity of Algorithm 6 for one subproblem is $\mathcal{O}(nL_{max}^3)$. We have two subproblems for each (lifted)

graph vertex, yielding complexity $\mathcal{O}(|V'|nL_{max}^3)$ for sending messages between all inflow and outflow subproblems in one message passing iteration.

Messages from path subproblems. Obtaining min marginal for one edge variable of a path subproblem requires two calls of restricted Algorithm 12 whose complexity is linear in the number of path edges. So, min-marginals for all path edges are obtained in $\mathcal{O}(|P|^2)$.

Messages from cut subproblems. Min marginal of one variable of a cut subproblems is obtained by adjusting its optimization Algorithm 3. The complexity is given by the complexity of the employed linear assignment problem which can be solved in polynomial time.

Cutting plane procedures. The cutting plane algorithms are called each 20 iterations. We allow to add maximally $0.5 \cdot |\mathcal{S}_0|$ new factors during one separation call, where $\mathcal{S}_0$ is the initial set of subproblems containing only inflow and outflow factors. So it holds, $|\mathcal{S}_0| = 2|V'|$. Once added, the subproblems influence the runtime via taking part in the message passing (see Section 8.7). Cutting plane itself (Sections 8.4 and 8.5) contains sorting of subsets of base or lifted edges which has complexity $\mathcal{O}(|E^-| \log |E^-|)$ (resp. $\mathcal{O}(|E^+| \log |E^+|)$). The other algorithms run in quadratic time w.r.t. number of vertices within relevant time distance to the currently processed edge.

Primal solution. We compute new primal solution in each five iterations. We use Algorithm 4 for obtaining base edge costs. Then, we use successive shortest paths algorithm for solving minimum cost flow problem and finally local search heuristic given by Algorithm 25, see Section 4.6. The complexity of solving MCF is the complexity of successive shortest path algorithm which is polynomial. Local search heuristic requires to compute and update cummulative costs between candidate paths. They can be computed in time linear in the number of lifted edges $\mathcal{O}(|E'|)$. MCF costs are obtained by calling Algorithm 1. Its complexity is discussed above.

8.14.2 Comparison with LifT

We perform several experiments for comparing our solver with an optimal solver for lifted disjoint paths LifT [28].

LifT global solution vs. two-step procedure. LifT is based on ILP solver Gurobi. It solves the LDP problem optimally. However, it is often not able to solve the problem on the full graphs. Therefore, LifT uses a two-step procedure. First, solutions are found on small time intervals to create tracklets. Second, the problem is solved on tracklets. This approach simplifies the problem significantly but the delivered solutions are not globally optimal anymore. We have observed that using our input costs, LifT is able to solve some problem sequences globally without the two step-procedure. Therefore, we compare our solver with LifT using both the two-step procedure and the global solution.

Influence of input costs. Our input data contain many soft constraints for obviously matching pairs of detections. Those are edges with negative costs significantly higher in absolute value than other edges costs. LifT finds an initial feasible solution using only base edges. This solution may be already very good due to the costs of obviously matching pairs. Moreover, Gurobi contains a lot of efficient precomputing steps, so it can recognize that the respective variables should be active in the optimum and reduce the search space.

Parameters. We adjust parameters of our solver to work with comparable data as LifT. For instance, we do not set cost of any base edges to zero (as described in Section 5.2) because LifT does not enable this option. So, the costs of overlapping base and lifted edges are duplicated as opposed to the most of other experiments. Moreover, if there is no edge between two detections within the maximal time distance in the input data, we can add a lifted edge with high positive cost for such pair in ApLift. This is useful for reducing the input size for MOT20 dataset. LifT does not have this option. Therefore, we disable this option for ApLift too.

Subsequences of MOT20. We present a comparison between our solver and LifT using two-step procedure on an example subsequence of MOT20-01 in Table 3 in the main text. On that subsequence, our solver is faster and has even slightly better IDF1 score than LifT. In Table 5, we present a comparison on first n frames of sequence MOT20-02 where LifT finds solutions faster than our solver using many iterations. We assume that this is caused by the input costs that are convenient for Gurobi, see the discussion above.

Train set of MOT17. We compare our solver with LifT on global training sequences of MOT17. That is, we do not use two-step procedure. Therefore, LifT finds the globally optimal solution if it finishes successfully. The runtime of LifT is exponential in general and it can be often killed because of memory consumption if run on global sequences. Therefore, we perform these experiments on a machine having 2000 GB RAM and multiple CPUs each having 64 cores.

The results are in Table 6. Asterisk in LifT time column indicate that the problem cannot be finished. Some of the processes are killed by the system because of too much memory consumption. Some processes do not finish within more than 27 hours. Moreover, LifT often occupied up to 30 cores for solving one sequence. Our solver uses only one core. In the cases when LifT does not finish, we evaluate the best feasible solution found by LifT. Those were typically the initial feasible solutions. That is, the solutions that ignore the lifted edges. Obtaining the initial solutions for these difficult instances took between 1700 and 4600 seconds. The numbers in brackets relate our results to LifT

results. The time column provides the ratio between our time and LifT time. The IDF1 column presents the difference between ApLift and LifT.

Table 5. Runtime and IDF1 comparison of LDP solvers: ApLift (ours) with 6, 11, 31 and 51 iterations and LifT[28] (two step procedure) on first n frames of sequence *MOT20-01* from MOT20.

n	Measure	LifT	Our6	Our11	Our31	Our51
50	IDF1↑	**83.4**	**83.4**	**83.4**	**83.4**	**83.4**
	time [s]	62	4	7	25	46
100	IDF1↑	**80.6**	79.9	79.9	79.9	79.9
	time [s]	124	30	54	182	360
150	IDF1↑	**78.7**	76.8	76.8	76.8	76.8
	time [s]	222	61	110	378	780
200	IDF1↑	**77.6**	75.8	75.8	75.8	75.8
	time [s]	354	95	177	604	1195

8.15. Qualitative Results

Figure 4 and Figure 5 show qualitative tracking results from the MOT20 [16] and MOT17 [16] datasets. Comparing the samples, it becomes apparent that the density of objects in MOT20 is much higher than in MOT17. The sequence MOT20-04 (Figure 4) has an average density of 178.6 objects per frame and sequence MOT17-12 (Figure 5) only 9.6. The high density in MOT20 results in very crowded groups of persons which are occluding each other completely or partially. Accordingly, appearance information are ambiguous, leading to less discriminative edge costs. An additional challenge arises due to the distance between the camera and the persons, as well as global illumination changes in some sequences. The images shown in Figure 4 are captured in a temporal distance of 40 frames (*i.e.* 1.6 seconds) and the illumination changes heavily. This leads to appearance changes within a short time, which makes re-identification challenging. For instance, the person with id 666 (top right corner) in Figure 4 is wearing a red scarf and a beige jacket. Only a few frames later, the person is barely visible and colors have changed.

Despite these challenges, our system delivers accurate tracking results, as can be seen from the result images. Also the combinatorial and computational challenge in computing optimal trajectories for MOT20, considering for each detections all possible connections within a 50 frame range becomes apparent.

Result video for all test sequences can be obtain from the official evaluation server, for MOT15[5], MOT16[6], MOT17[7], and MOT20[8].

[5] https://motchallenge.net/method/MOT=4031&chl=2

[6] https://motchallenge.net/method/MOT=4031&chl=5

[7] https://motchallenge.net/method/MOT=4031&chl=10

[8] https://motchallenge.net/method/MOT=4031&chl=

8.16. Tracking Metrics

A detailed evaluation of our proposed MOT system in terms of tracking metrics for all sequences of the datasets MOT20 [16] and MOT17 [45] are reported in Table 7. Evaluations on the test set are performed by the official benchmark evaluation server at https://motchallenge.net where our test results are reported as well. The tracking method for training sequences are trained with leave-one-out strategy to avoid overfitting on the corresponding training sequence.

Table 6. Runtime and IDF1 comparison of LDP solvers: ApLift (ours) with 6, 11, 31, 51 and 101 iterations and globally optimal (one step) LifT[28] on MOT17 train. Numbers in parenthesis in the time column show the difference between the solvers, in the IDF1 column the ratio between Lift and ApLift.

	LifT		Ours-6		Ours-11		Ours-31		Ours-51		Ours-101	
Sequence Name	Time↓	IDF1↑	Time↓	IDF1↑	Time↓	IDF1↑	Time↓	IDF1↑	Time↓	IDF1↑	Time↓	IDF1↑
02-DPM	7324 (1.0)	**49.4** (0.0)	94 (0.01)	47.4 (−2.00)	157 (0.02)	47.4 (−2.00)	513 (0.07)	49.1 (−0.30)	989 (0.14)	49.1 (−0.30)	2415 (0.33)	49.1 (−0.30)
02-FRCNN	4073 (1.0)	54.7 (0.0)	97 (0.02)	**54.9** (0.20)	161 (0.04)	**54.9** (0.20)	526 (0.13)	**54.9** (0.20)	1021 (0.25)	**54.9** (0.20)	2503 (0.61)	**54.9** (0.20)
02-SDP	7795 (1.0)	**56.7** (0.0)	131 (0.02)	55.0 (−1.70)	219 (0.03)	55.0 (−1.70)	717 (0.09)	55.0 (−1.70)	1410 (0.18)	55.0 (−1.70)	3685 (0.47)	55.0 (−1.70)
04-DPM	∗ (∗)	**75.4** (0.0)	449 (∗)	74.7 (−0.70)	756 (∗)	74.7 (−0.70)	2220 (∗)	74.7 (−0.70)	3929 (∗)	75.0 (−0.40)	8578 (∗)	75.0 (−0.40)
04-FRCNN	4889 (1.0)	**79.2** (0.0)	383 (0.08)	78.1 (−1.10)	644 (0.13)	78.1 (−1.10)	1811 (0.37)	76.3 (−2.90)	3111 (0.64)	78.2 (−1.00)	6565 (1.34)	78.2 (−1.00)
04-SDP	∗ (∗)	**82.3** (0.0)	499 (∗)	78.0 (−4.30)	839 (∗)	78.0 (−4.30)	2441 (∗)	78.0 (−4.30)	4294 (∗)	77.7 (−4.60)	9269 (∗)	79.9 (−2.40)
05-DPM	535 (1.0)	**65.0** (0.0)	10 (0.02)	62.6 (−2.40)	15 (0.03)	62.6 (−2.40)	57 (0.11)	63.5 (−1.50)	116 (0.22)	63.5 (−1.50)	298 (0.56)	63.5 (−1.50)
05-FRCNN	514 (1.0)	**66.6** (0.0)	10 (0.02)	63.8 (−2.80)	15 (0.03)	63.8 (−2.80)	57 (0.11)	64.0 (−2.60)	118 (0.23)	63.9 (−2.70)	315 (0.61)	65.6 (−1.00)
05-SDP	604 (1.0)	**67.9** (0.0)	11 (0.02)	**67.9** (0.00)	18 (0.03)	**67.9** (0.00)	67 (0.11)	67.1 (−0.80)	137 (0.23)	67.1 (−0.80)	364 (0.60)	67.6 (−0.30)
09-DPM	6692 (1.0)	**67.5** (0.0)	42 (0.01)	66.4 (−1.10)	70 (0.01)	66.4 (−1.10)	232 (0.03)	**67.5** (0.00)	480 (0.07)	**67.5** (0.00)	1281 (0.19)	**67.5** (0.00)
09-FRCNN	11888 (1.0)	**68.2** (0.0)	37 (0.00)	**68.2** (0.00)	61 (0.01)	**68.2** (0.00)	201 (0.02)	**68.2** (0.00)	407 (0.03)	**68.2** (0.00)	1095 (0.09)	**68.2** (0.00)
09-SDP	1462 (1.0)	**68.6** (0.0)	44 (0.03)	67.1 (−1.50)	74 (0.05)	67.1 (−1.50)	247 (0.17)	68.5 (−0.10)	512 (0.35)	68.5 (−0.10)	1443 (0.99)	68.5 (−0.10)
10-DPM	∗ (∗)	66.0 (0.0)	279 (∗)	**68.0** (2.00)	466 (∗)	**68.0** (2.00)	1524 (∗)	66.8 (0.80)	3087 (∗)	67.0 (1.00)	9478 (∗)	67.9 (1.90)
10-FRCNN	∗ (∗)	65.2 (0.0)	310 (∗)	68.8 (3.60)	511 (∗)	68.5 (3.30)	1689 (∗)	**69.4** (4.20)	3428 (∗)	**69.4** (4.20)	10743 (∗)	**69.4** (4.20)
10-SDP	∗ (∗)	65.4 (0.0)	379 (∗)	67.0 (1.60)	630 (∗)	67.0 (1.60)	2090 (∗)	67.4 (2.00)	4294 (∗)	67.1 (1.70)	13379 (∗)	**69.8** (4.40)
11-DPM	1991 (1.0)	**76.3** (0.0)	60 (0.03)	**76.3** (0.00)	99 (0.05)	**76.3** (0.00)	335 (0.17)	**76.3** (0.00)	672 (0.34)	**76.3** (0.00)	1672 (0.84)	**76.3** (0.00)
11-FRCNN	2382 (1.0)	**78.3** (0.0)	68 (0.03)	**78.3** (0.00)	113 (0.05)	**78.3** (0.00)	366 (0.15)	**78.3** (0.00)	729 (0.31)	**78.3** (0.00)	1799 (0.76)	**78.3** (0.00)
11-SDP	3195 (1.0)	80.0 (0.0)	68 (0.02)	79.8 (−0.20)	113 (0.04)	79.8 (−0.20)	370 (0.12)	**80.1** (0.10)	748 (0.23)	80.0 (0.00)	2057 (0.64)	80.0 (0.00)
13-DPM	∗ (∗)	62.8 (0.0)	152 (∗)	**66.8** (4.00)	252 (∗)	**66.8** (4.00)	944 (∗)	**66.8** (4.00)	2008 (∗)	**66.8** (4.00)	6340 (∗)	65.7 (2.90)
13-FRCNN	∗ (∗)	62.5 (0.0)	217 (∗)	**69.8** (7.30)	351 (∗)	**69.8** (7.30)	1331 (∗)	**69.8** (7.30)	2942 (∗)	67.7 (5.20)	9813 (∗)	66.2 (3.70)
13-SDP	∗ (∗)	64.5 (0.0)	196 (∗)	**66.8** (2.30)	326 (∗)	**66.8** (2.30)	1237 (∗)	**66.8** (2.30)	2698 (∗)	66.2 (1.70)	8954 (∗)	65.6 (1.10)
OVERALL	∗ (∗)	**70.7** (0.0)	168 (∗)	70.3 (−0.40)	280 (∗)	70.3 (−0.40)	904 (∗)	70.2 (−0.50)	1768 (∗)	70.3 (−0.40)	4859 (∗)	**70.7** (0.00)

Table 7. Evaluation results for training and test sequences for datasets MOT17 [45] and MOT20 [16]

	Sequence	MOTA↑	IDF1↑	MT↑	ML↓	FP↓	FN↓	IDS↓	Frag. ↓
MOT20 Train	MOT20-01	65.8	62.0	31	10	180	6512	109	87
	MOT20-02	62.3	55.1	108	18	1393	56420	548	534
	MOT20-03	80.4	76.1	427	66	5427	55552	623	591
	MOT20-05	74.6	57.8	643	115	8778	152927	2231	2063
	OVERALL	74.4	62.8	1209	209	15778	271411	3511	3275
MOT20 Test	MOT20-04	79.3	68.8	412	40	8315	47364	968	840
	MOT20-06	36.1	36.8	41	111	4786	79313	740	744
	MOT20-07	56.9	54.7	40	15	936	13135	194	195
	MOT20-08	26.5	33.8	20	98	3702	52924	339	333
	OVERALL	58.9	56.5	513	264	17739	192736	2241	2112
MOT17 Train	MOT17-02-DPM	42.2	52.5	14	29	125	10588	26	26
	MOT17-02-FRCNN	47.3	58.4	15	21	227	9532	27	30
	MOT17-02-SDP	55.1	60.5	17	16	289	7994	53	52
	MOT17-04-DPM	70.9	78.9	40	21	340	13481	17	29
	MOT17-04-FRCNN	68.0	78.4	39	21	179	15044	5	13
	MOT17-04-SDP	77.9	80.8	47	13	439	10035	29	68
	MOT17-05-DPM	60.0	64.5	48	34	475	2260	31	24
	MOT17-05-FRCNN	57.8	64.0	55	32	650	2225	46	41
	MOT17-05-SDP	62.6	67.8	59	19	693	1842	53	46
	MOT17-09-DPM	73.0	72.8	14	1	46	1380	10	9
	MOT17-09-FRCNN	71.5	68.4	14	1	105	1403	10	9
	MOT17-09-SDP	74.1	72.9	14	1	66	1302	10	11
	MOT17-10-DPM	65.3	67.4	32	6	847	3545	61	74
	MOT17-10-FRCNN	62.8	65.8	40	2	2121	2513	139	114
	MOT17-10-SDP	66.3	66.5	43	2	1967	2189	173	120
	MOT17-11-DPM	69.2	75.7	34	21	248	2624	37	17
	MOT17-11-FRCNN	71.5	76.8	38	18	412	2233	47	15
	MOT17-11-SDP	72.6	78.5	42	13	547	1981	58	19
	MOT17-13-DPM	64.4	64.8	55	33	627	3436	83	56
	MOT17-13-FRCNN	67.8	63.4	77	8	1739	1892	120	76
	MOT17-13-SDP	67.2	63.7	72	18	1388	2312	117	60
	OVERALL	66.0	71.4	809	330	13530	99811	1152	909
MOT17 Test	MOT17-01-DPM	48.8	54.3	8	10	113	3181	8	21
	MOT17-01-FRCNN	47.0	57.5	9	10	360	3050	11	21
	MOT17-01-SDP	45.2	55.4	9	10	488	3033	13	29
	MOT17-03-DPM	73.8	73.4	85	17	4360	22905	118	261
	MOT17-03-FRCNN	72.8	74.7	74	17	3471	24883	109	234
	MOT17-03-SDP	77.7	75.4	94	13	4676	18482	139	386
	MOT17-06-DPM	57.7	61.2	94	76	1142	3765	77	91
	MOT17-06-FRCNN	57.3	58.4	102	59	1652	3279	102	140
	MOT17-06-SDP	57.2	59.5	107	58	1700	3251	87	125
	MOT17-07-DPM	45.7	52.5	11	15	1062	8038	80	126
	MOT17-07-FRCNN	45.0	53.1	11	15	1345	7862	75	135
	MOT17-07-SDP	46.6	53.8	13	11	1622	7310	87	166
	MOT17-08-DPM	33.7	44.3	17	37	421	13533	48	67
	MOT17-08-FRCNN	31.5	42.1	17	37	462	13948	53	74
	MOT17-08-SDP	34.5	45.2	18	34	445	13339	63	85
	MOT17-12-DPM	47.6	61.9	23	36	563	3959	20	32
	MOT17-12-FRCNN	47.8	62.3	18	40	296	4219	13	24
	MOT17-12-SDP	50.0	66.1	19	42	488	3836	11	31
	MOT17-14-DPM	37.8	51.0	19	71	1147	10191	151	150
	MOT17-14-FRCNN	33.9	48.4	25	62	2369	9636	206	228
	MOT17-14-SDP	37.0	49.9	25	58	2427	8970	238	246
	OVERALL	60.5	65.6	798	728	30609	190670	1709	2672

Figure 4. Example images from sequence MOT20-04 at frames 1092 and 1132. The images shows a crowded scene captured by a static camera. The above image is captured before an illumination change happens. The lower image is captured after an illumination change happens. The appearance of persons changes consequently to the illumination changes (*e.g.* ID 666 in the top right corner). The result video can be found at https://motchallenge.net/method/MOT=4031&chl=13&vidSeq=MOT20-04.