A Generalized Successive Shortest Paths Solver for Tracking Dividing Targets

Carsten Haubold, Janez Aleš, Steffen Wolf, and Fred A. Hamprecht

IWR/HCI, University of Heidelberg, Germany
firstname.lastname@iwr.uni-heidelberg.de

Abstract. Tracking-by-detection methods are prevailing in many tracking scenarios. One attractive property is that in the absence of additional constraints they can be solved optimally in polynomial time, e.g. by min-cost flow solvers. But when potentially dividing targets need to be tracked – as is the case for biological tasks like cell tracking – finding the solution to a global tracking-by-detection model is NP-hard. In this work, we present a flow-based approximate solution to a common cell tracking model that allows for objects to merge and split or divide. We build on the successive shortest path min-cost flow algorithm but alter the residual graph such that the flow through the graph obeys division constraints and always represents a feasible tracking solution. By conditioning the residual arc capacities on the flow along logically associated arcs we obtain a polynomial time heuristic that achieves close-to-optimal tracking results while exhibiting a good anytime performance. We also show that our method is a generalization of an approximate dynamic programming cell tracking solver by Magnusson *et al.* that stood out in the ISBI Cell Tracking Challenges.

1 Introduction

Tracking proliferating cells is a task that arises e.g. in developmental biology and high-throughput screening for drug development. Tracking-by-detection methods are often the tool of choice because they allow for fine tuned detection algorithms, give room for a lot of modeling decisions, and do not require that the number of targets is known beforehand. One common ingredient in all tracking models for divisible targets is the constraint that a division can only occur in the presence of a parent. These constraints require the formulation of the objective as an integer linear program (ILP) [1,2,3]. Such ILPs can be solved to optimality up to a certain size, in spite of their NP-hardness; but they do not scale to the huge coupled problems that arise from long video.

Recently, min-cost flow solvers have become a popular choice to tracking multiple targets like pedestrians, cars, and other non-dividing objects [4,5,6,7]. These methods provide a polynomial runtime guarantee and are very efficient in practice, while solving the problem to global optimality. Unfortunately, min-cost flow solvers are not directly applicable to tracking problems with additional constraints such as the division constraint. Such additional constraints lead to a coupling of the flow along different arcs, destroying the total unimodularity (TUM) property of the constraint matrix – which is a necessary requirement for the linear programming relaxation solution to be integral,

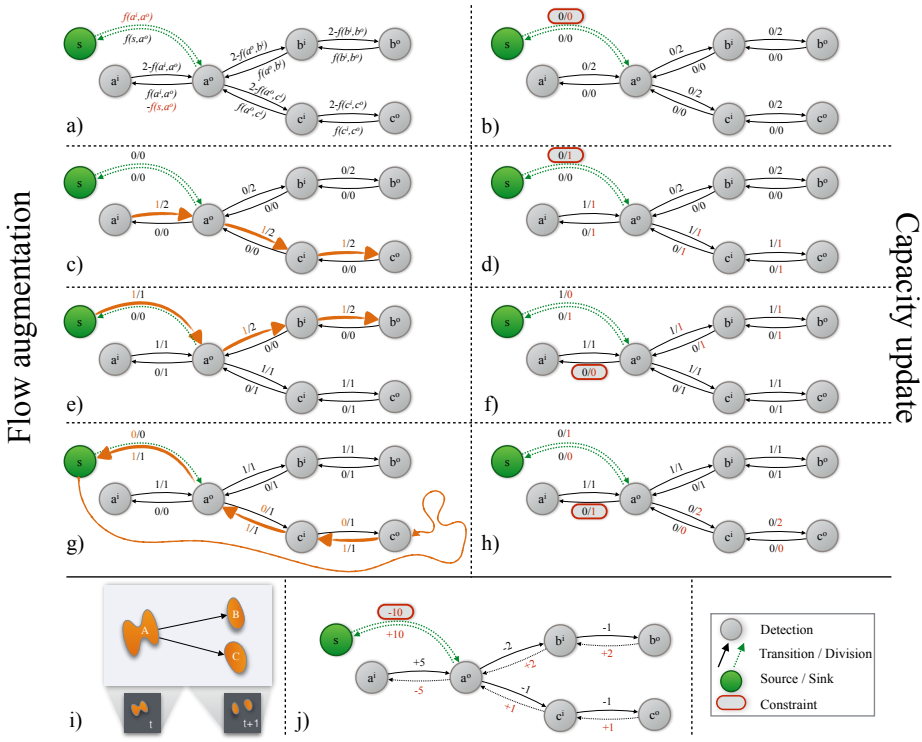


Fig. 1: Case study: A minimal example excerpt of a trellis graph **i)** and a few iterations of the proposed constraint-aware flow algorithm. **a)** shows how residual arc capacities $c^r(u, v)$ are derived from the flow f. In general for forward arcs this is $c^r(u, v) = c(u, v) - f(u, v)$ and for reverse arcs $c^r(v, u) = f(u, v)$. To realize the coupling between parent and division flow, we change how their residual capacity is derived (red). **b)** The graph with zero flow $f := 0$ and resulting initial residual arc capacities. Note the red border indicating that the division arc capacity is zero because of the coupling. Edge annotations are $f(u, v)/c^r(u, v)$. **c)** A shortest path (in orange) is found in the residual graph, and one unit of flow is pushed along that path. **d)** Deriving the new residual arc capacities, changes denoted in red. Because the parent detection a now contains flow, the coupled division arc residual capacity becomes one, making the division available for the next shortest path. **e-f)** The next shortest path and new residual arc capacities. The capacity of the reverse arc of the parent cell is set to zero because the division arc contains flow. **g)** A negative cost cycle is found, pushing flow along the reverse arcs. This is the same as canceling out a formerly found track. **h)** Flow along the division was removed again, leaving a residual graph with proper arc capacities such that the division could still be used in a later path. **j)** Failure case of our algorithm: arcs are now labeled with their costs, where the arc of the parent detection is so expensive that `crcSSP` will never get to the point where the rewarding division arc becomes available because it will not send flow along (a^i, a^o). The optimal solution would be to send flow along both parent and division.

and hence optimal. Some attempts have been made to apply min-cost flow solvers to network flow problems with side constraints nevertheless [8,9], but they mostly resort to rounding to finally obtain an integral solution.

In this work, we present an approximate primal feasible flow-based solver for tracking dividing targets. To achieve this, we modify the successive shortest paths (SSP) algorithm to handle the division constraints by conditioning certain residual arc capacities on the flow along logically associated arcs as shown in Fig. 1. This leads to a polynomial time algorithm that empirically exhibits attractive anytime performance and gives close to optimal results.

2 Related Work

Many tracking-by-detection models link the detections of a previously acquired per-frame segmentation between pairs of frames [10] or create short chains of detections and stitch them [11,12,13]. Others build a model spanning the entire time sequence to find a globally optimal configuration [4,5,6,14,15]. Standard tracking-by-detection expects all targets to be detected individually, which is not necessarily the case. [16] introduces a *contains* relationship employing prior knowledge that e.g. a person entered a car and track both objects at once. In the cell tracking domain such knowledge is usually not applicable: merging of targets occurs due to poor image quality or occlusion, leading to errors in the segmentation, apparent especially in densely populated areas. Furthermore, if cells are merged together into one segment, it is visually barely distinguishable whether this segment is splitting up or dividing, which is why dedicated methods [1,2,8,17] model those events explicitly. Most cell tracking models are solved as ILP because the division constraint prevents the application of optimal and efficient min-cost flow solvers.

Optimization problems that can be formulated as min-cost flow with convex cost and without additional constraints can be solved optimally in polynomial time. A variety of efficient solvers have been proposed: push-relabel, capacity scaling, network simplex, successive shortest paths (SSP), etc. [18,19,20]. Multi-target tracking can be solved using such a min-cost flow setup as shown in the seminal work by Zhang and Nevatia [4]. They model detections as a pair of nodes, with a connecting arc whose capacity limits the number of tracks through each detection to one and whose cost represents the detection cost. They allow negative arc costs so that they do not need to send a predefined amount of flow, but rather solve a series of min-cost flow problems with varying number of tracks to find the globally optimal configuration. Instead of solving a full min-cost flow problem for each number of tracks, [5] propose to use the SSP algorithm and add tracks as long as they lead to a lower cost solution. Berclaz *et al.* [7] improve on the runtime by using K-shortest paths instead of a single shortest path in each iteration. Lenz *et al.* [6] also present several ways to speed up the successive shortest paths search by updating only nodes for which the shortest path has changed due to flow augmentations along the previous shortest path. They transform their costs to be nonnegative, and can thus employ Dijkstra's algorithm to find the shortest paths efficiently. Lastly they develop an optimal and a heuristic but memory limited online tracker with very good runtimes.

For tracking proliferating targets, division constraints are needed. There has been some work on integrating side constraints into min-cost flow trackers, but they all relax the problem and then round the result to get a feasible solution. Butt *et al.* [9] build a tracklet linking model which is stated as a min-cost flow problem with additional exclusion constraints. They build the Lagrangian relaxation to get to a standard min-cost flow problem as subproblem, and then optimize the dual by stochastic gradient descent. As this need not converge to a primal feasible solution, they employ a greedy path selection scheme to resolve exclusion constraint violations. In contrast to this approach, we propose a heuristic that stays in the primal feasible domain and which does not need to solve the full min-cost flow in each iteration.

When tracking dividing targets, one needs to obey the constraint that a division cannot occur if there was no parent object in the first place. To handle this constraint in a flow network, there must be a means to spawn another unit of flow at a division, but this option should only be allowed when the parent detection holds some flow. Unfortunately these constraints violate the necessary criteria for the applicability of min-cost flow solvers. Despite that, Padfield [8] introduced *coupled flow* to handle divisions in a flow network for cell tracking, but they have to resort to a linear programming solution.

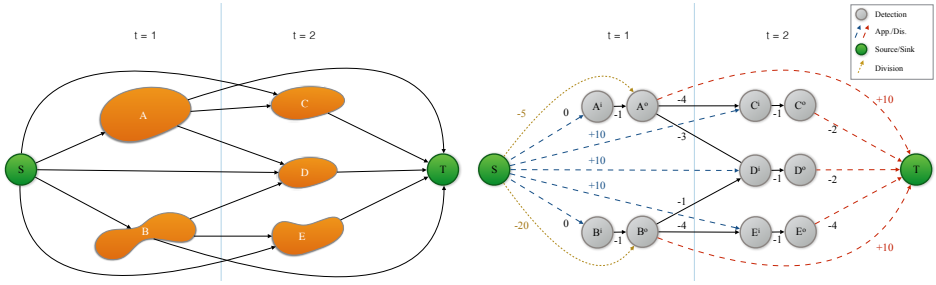
Magnusson *et al.* [21,17] – who showed outstanding performance at the 2013 and 2014 ISBI Cell Tracking Challenges [3] in both segmentation and tracking – set up a similar problem as we do here. They formalize their track linking heuristic as application of the Viterbi algorithm to find the shortest path in an acyclic graph where all arcs are directed forward in time. Instead of resorting to an ILP solver to cope with division constraints, they handle them by hiding arcs that could lead to invalid configurations from the shortest path search in each iteration. We borrow from this idea when developing our approximate min-cost flow based SSP solver and will later reason that our algorithm generalizes that of [17].

One additional complication is that in microscopy data, it is often not obvious from a single image how many objects are in a cluster. The study [22] revealed that undersegmentations are the prevailing segmentation error in cell tracking pipelines, and so here we focus on a tracking-by-detection model that allows for merged detections. Allowing detections to be shared by several tracks means that arc capacities in the network flow graph will be greater than 1. If this arc cost function is non-convex, solving the min-cost flow problem becomes NP-hard even in the absence of additional constraints [18].

3 Tracking Model

Throughout this work, we consider a tracking model where detections can divide, but targets can also temporarily merge into one detection due to undersegmentation, before they split again. This kind of model was used in e.g. [2] and [17]. We follow the design of [2], but for the sake of brevity we disregard their additional constraints which disallow the appearance/vanishing of merged detections and divisions of objects that just appeared[1]. For details we refer the reader to [2].

[1] Our implementation accounts for these constraints nevertheless, as they can be modeled by conditioning residual arc capacities on other arc flows similar to the division constraint, as we will explain in section 4.



(a) Trellis Graph (b) Flow Graph

Fig. 2: a) Trellis graph representing exemplary detection and transition candidates. b) Corresponding network flow graph. Detection nodes are split into two and the cost of the connecting arc accounts for the detection probability. Transition and division probabilities are represented by the other arc costs. This is the base graph without disabling any arcs due to division constraints (4). Exemplary costs are written alongside the arcs.

For our tracking model, we build a trellis graph as in Fig. 2 a) for the complete video time span, where all detections in all time steps are represented by nodes that can hold zero, one, or more than one target. The arcs in the graph depict possible assignments of objects across timeframes. Every arc thus points from a node in timeframe t to a node in $t + 1$. To reduce complexity, we only include transition hypotheses that satisfy a dataset-dependent distance threshold. Target appearance and disappearance are modeled as special assignment arcs, originating at the source or ending at the sink node respectively. Divisions can occur whenever a node has at least two outgoing arcs.

We now first state the optimization problem as integer linear program (ILP), and then present an equivalent network flow graph.

3.1 Integer Linear Program

To transform the aforementioned graph into an ILP, we assign random variables to all detection nodes and transition arcs. Let $\mathcal{V} := \mathcal{X} \cup \mathcal{T} \cup \mathcal{D}$ be the set of all detection nodes $\mathcal{X}$, transition arcs $\mathcal{T}$ and division indicators $\mathcal{D}$ in the graph. Every random variable $V \in \mathcal{V}$ can take a discrete state or label $k \in \mathcal{L}(V) := \{0, \dots, m\}$ indicating the number of contained targets, where m is the upper bound on the number of targets allowed to be merged into one detection. We introduce division random variables $D \in \mathcal{D}$ to indicate whether the corresponding detection $X(D)$ is dividing. [2] By *Source* and *Sink* we denote the source and sink node. Let $\mathbf{y} \in \mathcal{Y}$ be a valid labeling, that is, a vector assigning one state $\mathbf{y}_V \in \mathcal{L}(V)$ to every variable V, where $\mathcal{L}(V)$ is the label space of V.

We introduce a unary potential $\theta_V(k)$ for every random variable $V \in \mathcal{V}$. We set this potential to the negative log of the probability that the respective random variable V

[2] We slightly abuse the notation here and indicate the parent detection X of a division D as $X(D)$ and vice versa $D(X)$.

takes state k. This probability could for instance be estimated by a classifier, given the local observations. This choice of potentials ensures that the minimal energy configuration equals the *maximum-a-posteriori* (MAP) solution.

The energy minimization problem can be stated as

$$\mathbf{y}^* = \operatorname*{argmin}_{\mathbf{y}\in\mathcal{Y}} E(\mathbf{y}) = \operatorname*{argmin}_{\mathbf{y}\in\mathcal{Y}} \sum_{X\in\mathcal{X}} E_X(\mathbf{y}_X) + \sum_{T\in\mathcal{T}} E_T(\mathbf{y}_T) + \sum_{D\in\mathcal{D}} E_D(\mathbf{y}_D) \tag{1}$$

$$= \operatorname*{argmin}_{\mathbf{y}\in\mathcal{Y}} \sum_{X\in\mathcal{X}} \sum_{k\in\mathcal{L}(X)} \theta_X(k)\mathbb{1}[\mathbf{y}_X = k] + \sum_{T\in\mathcal{T}} \sum_{k\in\mathcal{L}(T)} \theta_T(k)\mathbb{1}[\mathbf{y}_T = k]$$

$$+ \sum_{D\in\mathcal{D}} \sum_{k\in\mathcal{L}(D)} \theta_D(k)\mathbb{1}[\mathbf{y}_D = k] \tag{2}$$

subject to:

Flow conservation: (3)

$$\forall_{X\in\mathcal{X}\cup\{Sink\}} : \mathbf{y}_X = \sum_{I\in\mathcal{I}(X)} \mathbf{y}_I, \quad \forall_{X\in\mathcal{X}\cup\{Source\}} : \mathbf{y}_X + \mathbf{y}_{D(X)} = \sum_{O\in\mathcal{O}(X)} \mathbf{y}_O$$

Division: (4)

$$\forall_{D\in\mathcal{D}} : \mathbf{y}_D - \mathbf{y}_{X(D)} \leq 0,$$

where $\mathcal{I}(X)$ denotes all incoming transition variables of detection X, and $\mathcal{O}(X)$ its outgoing transitions respectively. The outgoing transitions of the source $\mathcal{O}(Source)$ include all appearances and divisions, while the incoming transitions at the sink $\mathcal{I}(Sink)$ consist of all disappearances.

The objective (2) is a linear combination of configuration $\mathbf{y}$ and unary potentials θ, where $\mathbb{1}$ is the indicator function. The constraints ensure equality of the number of incoming and outgoing targets at a detection, including appearances and disappearances. Only in the presence of a division the number of outgoing targets can and must be greater than the number of incoming (3). Furthermore, a detection cannot divide more often than it contains targets (4). This last constraint is the key difference to a standard min-cost flow problem. As mentioned before, the full constraint matrix is not totally unimodular and standard flow solvers cannot be applied directly to find a feasible integral solution.

In [2], Schiegg *et al.* present a variant of the model above, introduce a few more constraints to represent design decisions like that the children of a division have to be two separate detections, and use IBM's CPLEX solver to find an optimal assignment $\mathbf{y}^*$ to the ILP. For all ILP results presented in the evaluation section of this work, we build the model as stated in (2), add equivalent constraints to those in [2], and solve it with the ILP solver Gurobi.

3.2 Network Flow Graph

Let us now present a transformation of the ILP – first without division constraints (4) – into an equivalent network flow graph $\mathcal{G} = (\mathbf{V}, \mathbf{E})$, as shown in Fig. 2 b). We are going to iteratively push one unit of flow through this network, where each additional path corresponds to the track of one object. We use the function $w(u, v, k) \in \mathbb{R}$ to denote the cost (which must be convex w.r.t. k) for a directed arc from u to v with current flow $k := f(u, v)$, and $c(u, v) \in \mathbb{N}^+$ to represent the arc capacity.

- Each **detection** $X \in \mathcal{X}$ is represented as a pair of in- and out-nodes x^i and x^o connected by a link with capacity $c(x^i, x^o) := |\mathcal{L}(X)| - 1$ and a weight depending on the detection probability for containing k targets, akin to [4]. The cost for the connecting arc is then $w(x^i, x^o, k) := \theta_X(k+1) - \theta_X(k)$.
- **Transitions** $T \in \mathcal{T}$, including appearances and disappearances, are represented as arcs which can leave from some out-node v^o or the $Source$, and arrive at a detection's in-node x^i or the $Sink$. Let $src(T)$ and $dest(T)$ denote functions that return the source and the destination node of transition T. Costs w are then assigned to arcs with capacity $c(src(T), dest(T)) := |\mathcal{L}(T)| - 1$ as $w(src(T), dest(T), k) := \theta_T(k+1) - \theta_T(k)$.
- Possibly **dividing** detections X get a special division in-arc from the source to x^o with cost $w(Source, x^o, k) := \theta_D(k+1) - \theta_D(k)$. Their capacity is defined in terms of the flow inside the parent detection, as we will see in the next section.

Thus we can state the full graph as

$$\begin{aligned}\mathcal{G} =& (\mathbf{V}, \mathbf{E}), \mathbf{V} = \{x^i, x^o | X \in \mathcal{X}\} \cup \mathcal{S}, \\ \mathbf{E} =& \{(src(T), dest(T)) | T \in \mathcal{T}\} \cup \{(x^i, x^o) | X \in \mathcal{X}\} \cup \{(Source, x^o_{(D)}) | D \in \mathcal{D}\}.\end{aligned}$$

In the next section we will see that shortest paths in $\mathcal{G}$ are found, and flow is pushed through the network along these paths. The path of each unit of flow through the network then corresponds to the track of one target. To make sure that the tracking solutions induced by the ILP and the network flow graph are equivalent, the accumulated cost $w(\mathcal{P}) = \sum_{u,v \in \mathcal{P}} w(u, v, k)$ of each path $\mathcal{P}$ must be equal to the change in energy if one adds one target to the ILP solution $\mathbf{y}$ along that track to get $\hat{\mathbf{y}}$, [3] which can be given as $E(\hat{\mathbf{y}}) - E(\mathbf{y}) = \sum_{V \in \mathcal{V}} \theta_V(\hat{\mathbf{y}}_V) - \theta_V(\mathbf{y}_V)$. To show the equality we decompose path $\mathcal{P}$ into the arcs that correspond to the sets of random variables $\mathcal{X}, \mathcal{T}$, and $\mathcal{D}$.

$$\begin{aligned} w(\mathcal{P}) &= \sum_{T \in \mathcal{P}} w(src(T), dest(T), \mathbf{y}_T) + \sum_{X \in \mathcal{P}} w(x^i, x^o, \mathbf{y}_X) + \sum_{D \in \mathcal{P}} w(Source, x^o_{(D)}, \mathbf{y}_D) \\ &= \sum_{T \in \mathcal{P}} \theta_T(\mathbf{y}_T + 1) - \theta_T(\mathbf{y}_T) + \sum_{X \in \mathcal{P}} \theta_X(\mathbf{y}_X + 1) - \theta_X(\mathbf{y}_X) + \sum_{D \in \mathcal{P}} \theta_D(\mathbf{y}_D + 1) - \theta_D(\mathbf{y}_D) \\ &= \sum_{V \in \mathcal{P}} \theta_V(\mathbf{y}_V + 1) - \theta_V(\mathbf{y}_V) = E(\hat{\mathbf{y}}) - E(\mathbf{y}) \end{aligned}$$

4 Approximate Min-Cost Flow: Conditioned Residual Capacities

In the preceding section we blithely ignored the division constraint (4). This section shows how to account for that constraint in a min-cost flow setup. Recent work [5,6,7] on solving the multi-target tracking problem as min-cost flow employed the *successive shortest paths* algorithm [19, p. 104]. We give a brief summary of SSP and generalize the algorithm to handle division constraints by conditioning the capacities of some arcs in the residual graph on the flow of logically associated arcs in the original graph. As the residual graph costs can be negative, not all shortest path solvers can be used for SSP. We argue why transforming the arc costs to be all positive in order to use Dijkstra's efficient algorithm is too expensive in the given scenario, so we use Bellman-Ford with performance improvements instead.

[3] which increases only the states of variables along the path $V \in \mathcal{P}$

4.1 Successive Shortest Paths

The SSP algorithm finds a global optimal solution to a min-cost flow problem by iteratively finding a path $\mathcal{P}$ with the lowest cost in the residual graph $\mathcal{G}^r(f)$ and then sending maximum feasible flow along this path [19, p. 104].

Let $f(u, v)$ denote the amount of flow traversing an arc (u, v) with capacity $c(u, v)$ in the original graph $\mathcal{G}$. The *residual graph* $\mathcal{G}^r(f)$ is then defined as a graph with the same nodes as $\mathcal{G}$, *forward* arcs (u, v) with *residual capacity* $c^r(u, v) = c(u, v) - f(u, v)$ and cost $w^r(u, v) = w(u, v)$, and *backwards* arcs with residual capacity $c^r(u, v) = f(u, v)$ with cost $w^r(v, u) = -w(u, v)$. By adding reverse arcs with capacity corresponding to the flow along the forward arc in the original graph, flow can be redirected in the residual graph.

Algorithm 1 Successive Shortest Paths with Conditioned Residual Capacities

1: **procedure** CRCSSP($\mathcal{G}, S, T$)
2: | $f \leftarrow 0, \mathcal{P} \leftarrow \emptyset, \mathcal{G}^r(f) \leftarrow \mathcal{G}$
3: | **repeat**
4: | | $f \leftarrow$ AUGMENTFLOW($f, \mathcal{P}$)
5: | | $\mathcal{G}^r(f) \leftarrow$ UPDATERESIDUALGRAPH($\mathcal{G}^r(f), f$)
6: | | $\hat{\mathcal{G}}^r(f) \leftarrow$ UPDATECONDITIONEDRESIDUALCAPACITIES($\mathcal{G}^r(f), f$)
7: | | $\mathcal{P} \leftarrow$FINDSHORTESTPATHORCYCLE($\hat{\mathcal{G}}^r(f), S, T$)
8: | **until** $w(\mathcal{P}) \geq 0$
9: | **return** f
10: **end procedure**

4.2 Successive Shortest Paths with Conditioned Residual Capacities

In section 3.2 we mentioned that the presented network flow setup does not support the division constraints yet. The obvious effect is that flow could be sent along a division arc even though no flow passes through the parent detection, yielding an invalid configuration. Rephrasing the division constraint to "the flow along a division arc is bounded by the amount of flow through the parent detection" directly leads to our main idea: we adjust the residual arc capacity in each iteration of SSP depending on the flow along other arcs in the original graph. In the general SSP algorithm, residual arc capacities $c^r(u, v)$ and $c^r(v, u)$ are derived only from the flow $f(u, v)$ along the corresponding arc in $\mathcal{G}$. Our extension to the SSP algorithm adds rules for deducing residual arc capacities depending on the flow of other arcs.

Let us formally state how we derive the *conditioned residual arc capacities* for the division constraint. According to the rephrased division constraint we define the residual arc capacity as $c^r(Source, x^o) := f(x^i, x^o)$ for each possible division of detection X (see Figure 1). This only covers one half of the division constraint in the residual graph $\mathcal{G}^r(f)$, as sending flow along the reverse residual parent arc could lead to $f(x^i, x^o) < f(Source, x^o)$. To prevent that we also condition $c^r(x^o, x^i) := f(x^i, x^o) - f(Source, x^o)$ on the division arc flow. These adjustments are handled by line 6 in Algorithm 1. Figure 1 walks through an example of using `crcSSP`.

This extension to the SSP algorithm allows us to handle division constraints in a way that maintains a feasible flow-induced tracking solution throughout all iterations of `crcSSP`. However, this comes at the cost of losing the global optimality guarantees and, moreover, introduces a dependency on the order in which paths are found. See Figure 1 j) for an example where the arc costs suggest that using parent detection and division arc together reduces the overall cost, yet our algorithm would use neither because sending flow only along the parent is costly and the division arc is not available yet. Nevertheless, when we apply Algorithm 1 to a dataset with no divisions, then line 6 has no effect and Algorithm 1 executes as the original SSP algorithm [19, p. 104], thus finds a global optimal solution.

4.3 Shortest Path Search: Bellman-Ford

The cyclic nature of the residual graph and the negative arc costs restrict the choice of shortest path algorithms applicable in SSP. One algorithm that can cope with negative cost cycles is *Bellman-Ford* (BF), which has a runtime complexity of $\mathcal{O}(|\mathbf{V}| * |\mathbf{E}|)$.

However, in the absence of negative cost cycles, one could once transform the arc costs to be *non-negative*, and then use the more efficient Dijkstra algorithm ($\mathcal{O}(|\mathbf{V}|log|\mathbf{V}| + |\mathbf{E}|)$) to find the shortest path based on these *reduced costs* $w_{>0}$ [18, p. 97], which is used by [6]. For this transformation, one needs to solve an auxiliary problem where an additional source node is added along with zero cost arcs to all nodes in the graph. Using BF one can now determine the shortest distance $d(v)$ to every node v in the original graph. Reduced costs are then given as $w_{>0}(u, v) := w(u, v) - d(u) + d(v)$. Note that $w_{>0}$ is zero for all arcs on shortest paths. This means that when arc costs are linear, the corresponding reverse oriented residual graph arcs also have zero reduced cost. So one can continue to use Dijkstra to search for SSP without the need to run the transformation again. Unfortunately, this does not hold in our situation for two reasons. First, we have non-linear cost, so after flow augmentation the costs of arcs change which in turn invalidates the distances $d(V)$ and, second, line 6 in our adjusted SSP Algorithm 1 can change the availability of other arcs in the residual graph, which also invalidates $d(V)$ if these arcs happen to have negative cost. This means that we would have to recompute at least part of the distances d after each iteration, where new cycles with negative weight might have been introduced.

Due to the structure of our tracking residual graph, which is a multipartite graph with node partitions indexed by time coordinate, and because of the necessity to have paths from source to sink with overall negative cost, the graph contains long chains of negative accumulated cost. This renders the solution of the auxiliary problem for the transformation very challenging. Our experiments verified that the combined runtime of the transformation plus Dijkstra exceeds the runtime of BF on the residual graph, which is why we chose to employ the latter solution.

Performance Improvements The BF algorithm runs in iterations, where each iteration performs $|\mathbf{E}|$ arc relaxations – which means it checks for each arc whether the current distance to the arc's destination node can be reduced by going along this arc. In the worst case, the number of these iterations is $|\mathbf{V}|$, which BF needs to run to prove the

existence of a negative cost cycle [20]. We base our BF implementation on the LEMON Library [23], which uses an early termination criterion: if nothing changes between two iterations, BF has computed the shortest paths to all nodes in the graph. Another included performance improvement is that only those nodes whose predecessors have changed in the previous iteration are processed in the next iteration.

We add two more stopping criteria to deal with negative cost cycles. Firstly, considering that in our model we perform a single source, single destination shortest path search, it is easy to see that if the shortest distance to the $Source$ node – which is initially zero – gets updated in any iteration of BF, then we definitely found a negative cost cycle. Secondly, we know that our tracking graph has only a fixed number of time frames, so we can check how many iterations it takes in general to find a path. If a negative cost cycle is present in the residual graph, it could be discovered at each BF-iteration using a check which takes $\mathcal{O}(|\mathbf{V}|^2)$. This is costly, but we still know that a cycle can be found much earlier than in iteration $|\mathbf{V}|$, so we check for cycles every α iterations of BF. In our experiments we use α equal to three times the number of time steps. These cycle detection checks are crucial to the practicability of our algorithm, as a considerable amount of negative cost cycles needs to be found. Without the checks this takes up to the order of minutes when a cycle is present.

Furthermore, the BF algorithm needs the least number of arc-relaxation iterations when the arcs are processed in the order of the shortest paths. If there are no arcs pointing backwards in time, BF can terminate after only one iteration by processing arcs in a time-wise order. Our experiments show that this arc ordering yields a significant runtime improvement even in the presence of arcs that are directed backwards in time. We call this `crcSSP-o` in the evaluation.

Lenz [6] improves Dijkstra's runtime when solving the SSP problem as follows. They observe that after augmenting flow along paths $\mathcal{P}$, only the distances to those nodes need to be updated, for which the shortest path to the node was modified by this flow augmentation. They achieve this by initializing Dijkstra's priority queue of unprocessed nodes with exactly those nodes that were influenced by the last path. We apply the same idea when running BF, and initialize as follows: We invalidate the predecessor and shortest path to every node on the path $\mathcal{P}$ and perform a dynamic programming sweep starting with the outgoing arcs which belonged to shortest paths. Next, we construct the set of nodes to be processed with all those nodes in the graph that have an outgoing arc to one of the now invalidated nodes. We only employ this when there was no negative weight cycle. Otherwise we perform the default initialization. We denote the application of improved initialization by `crcSSP-i`.

4.4 Runtime Complexity

The residual graph has N nodes representing the source, sink and split detections, as well as additional division nodes $N = 2+2*|\mathcal{X}|$. The number of arcs M is composed of the transitions, divisions, detections, appearances, and disappearances, so $M = |\mathcal{T}| + |\mathcal{D}| + 3 * |\mathcal{X}|$. BF has runtime $\mathcal{O}(N * M)$, and it is invoked once for each augmenting path. Let $\mathscr{P}$ be the set of paths comprising the final solution and $P = |\mathscr{P}|$, then our overall runtime is $\mathcal{O}(N * M * P)$. In the worst case we have a complete graph where $M = N^2$, and as many paths as there are detections $|\mathscr{P}| \approx \frac{N}{2}$. Hence, the worst case

complexity is $\mathcal{O}(N^4)$. Let L be the average number of possible outgoing transitions from each detection (in practice $L < 10$, for us $L \approx 3$). Hence, we can estimate $M = |\mathcal{X}|(3+L)$, where we have $|\mathcal{X}| * L$ transitions, plus one appearance, disappearance, and one connecting arc between the split detection nodes. Also, P is usually much smaller than $|\mathcal{X}|$, more in the order of thousands in our experiments. The overall runtime is then $\mathcal{O}(P * N * |\mathcal{X}|(3+L)) = \mathcal{O}(P * N^2)$.

4.5 First-order residual graph approximation

Magnusson *et al.* [17] proposed to perform track linking by iteratively augmenting the set of tracks by the highest scoring track in a trellis graph that only has arcs directed forward in time – which can be found in linear time by dynamic programming [20, p. 592]. They also adjust the arc costs and availability in each iteration according to the current tracking solution and constraints.

Even though [17] did not draw the link from their work to network flow solvers, one could interpret their approach as removing backward arcs from the residual graph and finding the shortest path there. Let $t(x)$ denote the time frame of node x[4], and $\mathcal{G}^r(f_i) = (\mathbf{V}, \mathbf{E}_i^r)$ be the residual graph at iteration i. Then the set of arcs directed forward in time is given as $\tilde{\mathbf{E}}_i^r = \{(k,l) | (k,l) \in \mathbf{E}_i^r, t(k) < t(l)\}$. A shortest path $\tilde{\mathcal{P}}$ between two nodes found in the restricted residual graph $\tilde{\mathcal{G}}^r(f_i) = (\mathbf{V}, \tilde{\mathbf{E}}_i^r)$ is always also a valid path in $\mathcal{G}^r(f_i)$, but it is obvious that the cost $w(\tilde{\mathcal{P}})$ is always greater than or equal to the cost $w(\mathcal{P})$ of the shortest path (SP) in $\mathcal{G}^r(f_i)$ because the shortest path in the full residual graph can travel along negative cost arcs directed backwards. Using this restricted graph for the SP search in Algorithm 1 trades an improvement of the runtime of line 7 from $\mathcal{O}(|\mathbf{V}| * |\mathbf{E}_i^r|)$ to $\mathcal{O}(|\mathbf{V}| + |\tilde{\mathbf{E}}_i^r|)$ for a larger optimality gap, which can be seen in the results section.

To allow the algorithm to escape from local minima, [17] introduce *swap arcs*, which we will now restate using residual graph terminology. On top of $\tilde{\mathbf{E}}_i^r$ they instantiate every possible 3-arc sub-path of the residual graph $\{(k,l),(l,m),(m,n)\} \in (\mathbf{E}_i^r)^3$ – where the middle arc is oriented backwards in time – as swap arc (k,n) with cost $w(k,n) = w(k,l) - w(l,m) + w(m,n)$.[5] As all *swap arcs* are also directed forward in time, the shortest path in the graph can still be found by dynamic programming. Once such a swap arc is used, the flow in the original graph is augmented by pushing $1, -1, 1$ units of flow along the 3 arcs respectively, which represents a short flow redirection. If we interpret $\tilde{\mathcal{G}}^r(f_i)$ as a first order approximation of the full residual graph, then including swap arcs leads to a second order approximation.

Because finding the shortest paths in acyclic approximations of the residual graph has linear time complexity, the approach by [17] should run much faster than BF. In our experiments we thus do not only compare against the results by Magnusson *et al.*, but we also try a two-stage approach, where we first run [17] and then use `crcSSP` to find negative weight cycles and reduce the total energy even further.

[4] where $t(x^o) = t(x^i) + 1$

[5] Actually they ignore arc-triplets which contain a division arc, and thus never allow flow to be redirected along divisions.

5 Experiments

We evaluate the proposed algorithm on two challenging datasets from developmental biology, a 3D+t drosophila scan [24] and 2D+t pancreatic rat stem cells (PSC) presented in [22], both publicly available with ground-truth. The former is a time series of a developing embryo where exact cell lineages over long time spans are desired, and the latter presents stem cells in a dish which can overlap and often change their shape. As in [2] and [17], we assign to each detection the probability for containing a certain number of cells $P_{det}(k)\ \forall k \in [0, m]$, as well as a probability for division P_{div}.[6] These probabilities are predicted by Random Forest classifiers which were trained on the same subset of the data as described in [24]. Transition arcs are inserted for nodes that satisfy a forward-backward nearest neighbor check between consecutive frames, and the transition probability is given by the inverted exponential of the Euclidean distance. Energies are derived from those probabilities by taking the negative logarithm. We use the open source implementation of [2] included in ilastik [25] to generate segmentations, predict probabilities with their classifiers and to construct the trellis graph. The resulting network flow graph for the Drosophila dataset then consists of around 45k nodes and 110k arcs, of which $\sim$10k are division arcs. For the much bigger PSC dataset the graph has roughly 260k nodes and 770k arcs including 126k division arcs.

Convex energies are required to obtain an optimal integral solution when applying SSP to a network problem. We obtain these energies by finding a convex upper envelope $\bar{\theta_i}$ to each potential θ_i independently. We first select the state with minimal energy and make sure that when increasing or decreasing the state from there on, the absolute slope of the gradient is monotonically increasing. Our experiments showed that the cost convexification does not impact the quality of the final tracking results when applied to the energies of either datasets used.

As mentioned before, the tracking model presented in section 3 is a slight simplification of the model in [2]. For the experiments we use their full model where we handle additional constraints similar to the division constraint.

We compare the tracking performance w.r.t. the ground truth by checking for the agreement of *move*, *merge*, and *division* events per pair of consecutive frames. A *merge* event in this case means that a detection contains more than one cell in the ground truth, and is only found correctly by a contestant if the number of contained tracks matches. Table 1 shows the results for the first and second order residual approximation as presented in [17], our proposed new method using the full residual graph, and the ILP solution found with Gurobi. In Figure 3 we compare the anytime performance of the same solvers. The anytime performance refers to the energy of a solution obtained after any time point during the optimization. There one can also see the impact of the different BF performance improvements we added, as well as the performance of warmstarting `crcSSP` from the solution of Magnusson's approach. We implemented all methods ourselves in C++ and used the Lemon graph library [23] as base for our improved BF, and OpenGM to interface Gurobi.[7]

[6] $m = 4$ for the Drosophila dataset, $m = 3$ for PSC.

[7] `http://github.com/opengm/opengm` and `http://www.gurobi.com`.

Inference Method	Forward Only			Magnusson			crcSSP-o-i			ILP (Gurobi)		
Dataset and Event	p	r	F	p	r	F	p	r	F	p	r	F
Drosophila: Overall	0.89	0.90	0.89	0.90	0.91	0.90	0.93	**0.92**	0.92	0.96	0.91	0.94
Drosophila: Moves	0.93	0.96	0.95	0.94	0.97	0.96	0.95	**0.98**	0.96	0.97	0.98	0.97
Drosophila: Mergers	0.27	0.60	0.37	0.31	0.63	0.41	0.50	**0.65**	0.56	0.84	0.50	0.63
Drosophila: Divisions	0.49	0.42	0.45	0.70	0.36	0.48	0.80	0.46	0.58	0.85	0.67	0.75
PSC: Overall	0.69	0.89	0.78	0.75	0.91	0.82	**0.89**	0.93	**0.91**	0.88	0.94	0.91
PSC: Moves	0.82	0.92	0.87	0.86	0.95	0.90	**0.92**	0.96	**0.94**	0.92	0.97	0.94
PSC: Mergers	0.08	0.51	0.14	0.11	0.52	0.18	0.33	0.52	0.40	0.34	0.55	0.42
PSC: Divisions	0.08	0.11	0.10	0.13	0.16	0.14	**0.34**	0.32	0.33	0.33	0.37	0.35
Drosophila Runtime/RAM	13s / 0.5GB			17s / 0.5GB			17s / 0.5GB			19s / 1.2GB		
PSC Runtime/RAM	140s / 2.1GB			210s / 2.1GB			515s / 2.2GB			987s / 3.6GB		

Table 1: F-Measure F, precision p and recall r for the different occurring events in solutions obtained by the SSP solver using first and second order residual graph approximation (Magnusson), the full residual graph, and lastly the optimal ILP solver. Our proposed crcSSP solver performs much better than the residual graph approximations in terms of solution quality, and is significantly faster than the ILP solver on the big PSC dataset while giving close to optimal results (**bold** means better or on par). *Last two columns:* runtime and RAM usage on a 2.8GHz Intel Core i7 with 8GB RAM

6 Discussion

Figure 3 shows that the proposed crcSSP-o-i algorithm yields a very good trade-off between runtime and solution quality. While Magnusson's dynamic programming shortest path search [17] leads to a fast energy reduction in the beginning, which is especially apparent for the PSC dataset, crcSSP-o-i is able to find paths with high contribution throughout because it can redirect flow and handle negative weight cycles in each iteration.

As all crcSSP variants are greedy and we use different node ordering and initialization strategies it must not always be that the same heuristic achieves the lowest overall energy. Nevertheless, they all reach an energy close to the optimum. The benefit of applying the different BF performance improvements is huge, ordering the nodes alone yields a speed-up of factor 2 and 3 on the different datasets respectively. Restricting BF to only recompute the shortest paths to those nodes whose minimal distance could have changed brings another significant improvement, and judging from the runtime, it reduces the need for node ordering. With all improvements enabled, our algorithm outperforms Gurobi in terms of runtime on both datasets. On the larger PSC dataset it finishes in about only half the runtime, and the runtime complexity dictates that this gap grows with graph size, making crcSSP-o-i an attractive choice for large scale problems. We verify this by artificially duplicating the PSC model, where Gurobi converges after around 3300 seconds and crcSSP-o-i after roughly 1300 seconds.

As one would expect, the first and second order residual graph approximations are fast, but cannot find a very low overall energy. Feeding the solutions found with [17] as initialization into crcSSP allows to improve the energy further, but because then

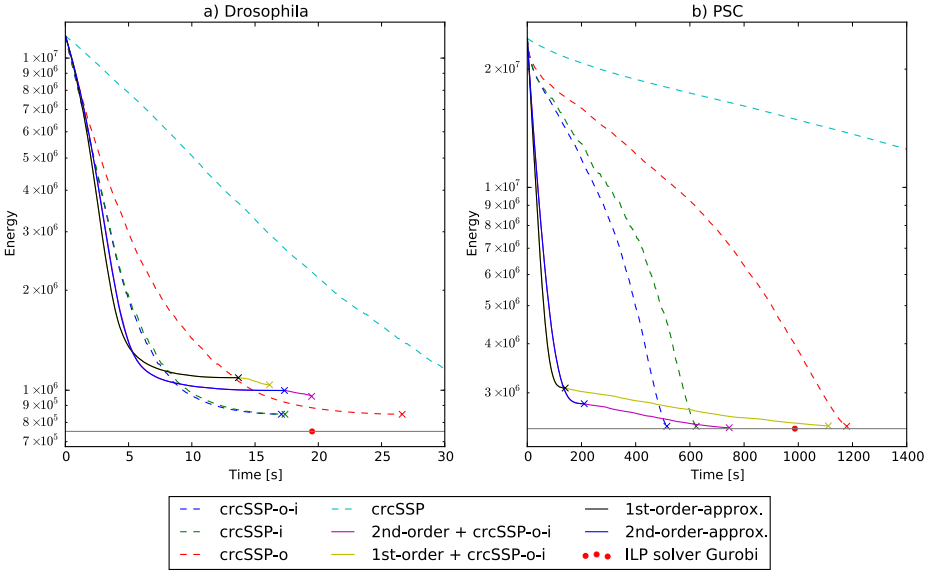


Fig. 3: Anytime performance of the optimal ILP solver, the residual graph approximations by [17], and our proposed `crcSSP` solver on two datasets. The `crcSSP` performance improvements of ordering nodes (`-o`) and initializing BF to update only part of the nodes (`-i`) turn out to have a strong impact on the runtime. Using `crcSSP` to refine the solutions found by [17] cannot compete with running `crcSSP-o-i` throughout.

the graph is quite saturated with flow, `crcSSP` finds negative cost cycles in nearly all iterations. The BF runtime is much higher when a negative cost cycle is present, which is why the anytime performance suffers. The tracking accuracy evaluation in Table 1 reveals that our proposed solver does not only exhibit attractive anytime performance, but that it also produces very accurate tracking results, which are on a par with the optimal ILP solution for the PSC dataset, and still significantly better than the residual graph approximations for *merger* and *move* events in the Drosophila dataset.

7 Conclusion

In this work we have proposed a way to integrate division constraint handling into the successive shortest paths min-cost flow algorithm by conditioning residual arc capacities on the flow along other arcs. While these conditioned residual capacities render our approach greedy, the evaluation shows that it gets close to the optimal energy and yields high quality tracking results for proliferating cells with attractive anytime performance. The core idea is well suited to be adapted to other types of constraints. We have made our code for the ILP model[8] and our presented solver publicly available[9].

[8] http://github.com/chaubold/multiHypothesesTracking
[9] http://github.com/chaubold/dpct

Acknowledgements This work was partially supported by the HGS MathComp Graduate School, the SFB 1129 for integrative analysis of pathogen replication and spread, the RTG 1653 for probabilistic graphical models and the CellNetworks Excellence Cluster / EcTop.

References

1. Bise, R., Yin, Z., Kanade, T.: Reliable Cell Tracking by Global Data Association. In: IEEE International Symposium on Biomedical Imaging (ISBI). (2011) 1004–1010 1, 3
2. Schiegg, M., Hanslovsky, P., Kausler, B.X., Hufnagel, L., Hamprecht, F.A.: Conservation tracking. In: IEEE International Conference on Computer Vision (ICCV), IEEE (2013) 2928–2935 1, 3, 4, 6, 12
3. Maška, M., Ulman, V., Svoboda, D., Matula, P., Matula, P., Ederra, C., Urbiola, A., España, T., Venkatesan, S., Balak, D.M., et al.: A benchmark for comparison of cell tracking algorithms. Bioinformatics **30**(11) (2014) 1609–1617 1, 4
4. Zhang, L., Li, Y., Nevatia, R.: Global data association for multi-object tracking using network flows. In: IEEE Conference on Computer Vision and Pattern Recognition, (CVPR), IEEE (2008) 1–8 1, 3, 7
5. Pirsiavash, H., Ramanan, D., Fowlkes, C.C.: Globally-optimal greedy algorithms for tracking a variable number of objects. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE (2011) 1201–1208 1, 3, 7
6. Lenz, P., Geiger, A., Urtasun, R.: Followme: Efficient online min-cost flow tracking with bounded memory and computation. In: IEEE International Conference on Computer Vision (ICCV). (2015) 4364–4372 1, 3, 7, 9, 10
7. Berclaz, J., Fleuret, F., Türetken, E., Fua, P.: Multiple object tracking using k-shortest paths optimization. Pattern Analysis and Machine Intelligence, IEEE Transactions on **33**(9) (2011) 1806–1819 1, 3, 7
8. Padfield, D., Rittscher, J., Roysam, B.: Coupled minimum-cost flow cell tracking for high-throughput quantitative analysis. Medical image analysis **15**(4) (2011) 650–668 3, 4
9. Butt, A., Collins, R.: Multi-target tracking by lagrangian relaxation to min-cost network flow. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE (2013) 1846–1853 3, 4
10. Kuhn, H.W.: The hungarian method for the assignment problem. Naval research logistics quarterly **2**(1-2) (1955) 83–97 3
11. Xing, J., Ai, H., Lao, S.: Multi-object tracking through occlusions by local tracklets filtering and global tracklets association with detection responses. CVPR 2013 (2009) 3
12. Castanon, G., Finn, L.: Multi-target tracklet stitching through network flows. In: IEEE Aerospace Conference, IEEE (2011) 1–7 3
13. Jaqaman, K., Loerke, D., Mettlen, M., Kuwata, H., Grinstein, S., Schmid, S.L., Danuser, G.: Robust single-particle tracking in live-cell time-lapse sequences. Nature methods **5**(8) (2008) 695–702 3
14. Andriyenko, A., Schindler, K., Roth, S.: Discrete-continuous optimization for multi-target tracking. In: Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on, IEEE (2012) 1926–1933 3
15. Brendel, W., Amer, M., Todorovic, S.: Multiobject tracking as maximum weight independent set. In: Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on, IEEE (2011) 1273–1280 3
16. Wang, X., Türetken, E., Fleuret, F., Fua, P.: Tracking interacting objects optimally using integer programming. In: European Conference on Computer Vision (ECCV). Springer (2014) 17–32 3

17. Magnusson, K., Jalden, J., Gilbert, P., Blau, H.: Global linking of cell tracks using the viterbi algorithm. Transactions on Medical Imaging **34**(4) (2014) 911 – 929 3, 4, 11, 12, 13, 14
18. Bertsekas, D.P.: Network optimization: continuous and discrete models. Athena Scientific Belmont (1998) 3, 4, 9
19. Ahuja, R.K., Magnanti, T.L., Orlin, J.B.: Network flows. Technical report, DTIC Document (1988) 3, 7, 8, 9
20. Cormen, T.H.: Introduction to algorithms. MIT press (2009) 3, 10, 11
21. Magnusson, K.E., Jaldén, J.: A batch algorithm using iterative application of the viterbi algorithm to track cells and construct cell lineages. In: International Symposium on Biomedical Imaging (ISBI), IEEE (2012) 382–385 4
22. Rapoport, D.H., Becker, T., Mamlouk, A.M., Schicktanz, S., Kruse, C.: A novel validation algorithm allows for automated cell tracking and the extraction of biologically meaningful parameters. PloS one **6**(11) (2011) e27315 4, 12
23. Jüttner, A., Dezsö, B., Kovács, P.: Lemon: Library for efficient modeling and optimization in networks. Technical report, Dept. of Operations Research, Eötvös Loránd University, Budapest 10, 12
24. Schiegg, M., Hanslovsky, P., Haubold, C., Koethe, U., Hufnagel, L., Hamprecht, F.A.: Graphical model for joint segmentation and tracking of multiple dividing cells. Bioinformatics (2014) btu764 12
25. Sommer, C., Straehle, C., Kothe, U., Hamprecht, F.A.: Ilastik: Interactive learning and segmentation toolkit. In: IEEE International Symposium on Biomedical Imaging: From Nano to Macro (ISBI), IEEE (2011) 230–233 12