DISSERTATION
submitted
to the
Combined Faculty for the Natural Sciences and
Mathematics
of
Heidelberg University, Germany
for the degree of
Doctor of Natural Sciences

Put forward by
MSc Carsten Haubold
Born in Saarbrücken, Germany
Oral examination:

# Scalable Inference for Multi-Target Tracking of Proliferating Cells

Advisor: Prof. Dr. Fred A. Hamprecht

# Acknowledgments

First of all, I would like to thank Prof. Fred Hamprecht for supervising my work, but especially for always trying to make the group have the newest and best biological data available. It was a tremendous motivation knowing that our work supports researchers around the world. While Prof. Hamprecht is always pushing to take the next step forward that nobody else has pursued yet, Ullrich Koethe wants to dig deep and understand every smallest detail. Combining these two goals under their supervision fosters state-of-the art research across the whole group.

When I first joined the lab, it was Martin Schiegg who introduced me to the world of tracking, and sharpened my understanding of the matter through numerous productive discussions. I highly appreciate his support, and am using the tracking models he developed as the foundation for everything built in this thesis. Furthermore, I thank Steffen Wolf for the great time we had in our offices, for endless discussions about work and beyond, and for even laughing at the frustrating moments together. I would like to thank my students Mathis Brosovski, Jonas Massa, Letitia Parcalabescu, Philip Schmidt and David Stoeckel for their friendly collaboration and important contributions to making tracking work not just in theory, but on real world data. Not to forget all the other current and former lab members who make this group such an enjoyable and welcoming place, especially Anna Kreshuk, Thorsten Beier and Ferran Diego. Behind the scenes, Barbara Werner is pulling the administrative strings to let everyone else focus on science instead of bureaucracy. I do not know how the lab would work without her, so thank you! And even though in his job he is confronted with more screaming than praise whenever something breaks, our administrator Dominic Spangenberger is doing an amazing job providing dozens of people with working machines and services. I am glad that I was able to pursue my PhD in such a great environment, where someone always finds the right pointers or questions to go the next step.

Outside our lab I was fortunate enough to meet and collaborate with several bright and enthusiastic scientists and developers. I appreciate the patience of tracking algorithm users Sabrina Rossberger and Andrea Imle who were not scared off by the occasional bug and waited until the software matured. A big thanks goes to our colleagues Stuart Berg and Jaime Cervantes in Janelia who push the software forward, and also took care of making my visit so much fun. I also had a great time with Florian Jug and Tobias Pietzsch from the MPI Dresden during numerous hackathons and some refreshing runs. And a huge thank you to Virginie Uhlmann from EPFL who – despite the distance – managed to always bring a good mood into the office via Skype during our productive daily chats and discussions. Chapters 5 and 6 are results of our great collaboration.

Lastly, I thank my family for their support and for always having good advice. And I cannot thank my girlfriend Sonja Mohnen enough for helping me turn away from work and enjoy the other things in life from time to time.

# Abstract

With the continuous advancements in microscopy techniques such as improved image quality, faster acquisition and reduced photo-toxicity, the amount of data recorded in the life sciences is rapidly growing. Clearly, the size of the data renders manual analysis intractable, calling for automated cell tracking methods. Cell tracking – in contrast to other tracking scenarios – exhibits several difficulties: low signal to noise ratio in the images, high cell density and sometimes cell clusters, radical morphology changes, but most importantly cells *divide* – which is often the focus of the experiment. These peculiarities have been targeted by *tracking-by-assignment* methods that first extract a set of detection hypotheses and then track those over time. Improving the general quality of these cell tracking methods is difficult, because every cell type, surrounding medium, and microscopy setting leads to recordings with specific properties and problems. This unfortunately implies that automated approaches will not become perfect any time soon but manual proof reading by experts will remain necessary for the time being. In this thesis we focus on two different aspects, firstly on scaling previous and developing new solvers to deal with longer videos and more cells, and secondly on developing a specialized pipeline for detecting and tracking tuberculosis bacteria.

The most powerful *tracking-by-assignment* methods are formulated as probabilistic graphical models and solved as integer linear programs. Because those integer linear programs are in general *NP*-hard, increasing the problem size will lead to an explosion of computational cost. We begin by reformulating one of these models in terms of a constrained network flow, and show that it can be solved more efficiently. Building on the successful application of network flow algorithms in the pedestrian tracking literature, we develop a heuristic to integrate constraints – here for divisions – into such a network flow method. This allows us to obtain high quality approximations to the tracking solution while providing a polynomial runtime guarantee. Our experiments confirm this much better scaling behavior to larger problems. However, this approach is single threaded and does not utilize available resources of multi-core machines yet. To parallelize the tracking problem we present a simple yet effective way of splitting long videos into intervals that can be tracked independently, followed by a sparse global stitching step that resolves disagreements at the cuts. Going one step further, we propose a microservices based software design for *ilastik* that allows to distribute all required computation for segmentation, object feature extraction, object classification and tracking across the nodes of a cluster or in the cloud.

Finally, we discuss the use case of detecting and tracking tuberculosis bacteria in more detail, because no satisfying automated method to this important problem existed before. One peculiarity of these elongated cells is that they build dense clusters in which it is hard to outline individuals. To cope with that we employ a tracking-by-assignment model that allows competing detection hypotheses and selects the best set of detections while considering the temporal context during tracking. To obtain these hypotheses, we develop a novel algorithm that finds diverse $M$-best solutions of tree-shaped graphical models by dynamic programming. First experiments with the pipeline indicate that it can greatly reduce the required amount of human intervention for analyzing tuberculosis treatment.

# Zusammenfassung

Kontinuierliche Fortschritte in der Mikroskopie wie z.B. verbesserte Bildqualität, höhere Aufnahmegeschwindigkeit und weniger fototoxische biologische Marker, lassen die Menge an aufgezeichneten Videos in den Biowissenschaften rapide wachsen. Aufgrund der Größe der Daten wäre eine manuelle Analyse schwer zu bewältigen, weshalb automatisierte Verfahren vonnöten sind. Im Gegensatz zu anderen Tracking-Szenarien hat das Zell-Tracking einige verkomplizierende Besonderheiten: ein schlechtes Signal-zu-Rauschen Verhältnis in den Bildern, eine hohe Dichte von Zellen bis hin zu Zellhaufen, starke Veränderungen der Form. Aber am allerwichtigsten ist, dass Zellen sich *teilen* können – was oft das Hauptaugenmerk der biologischen Studie ist. Diese Besonderheiten wurden in vorangegangenen Arbeiten mit so genannten *Tracking-by-Assignment*-Methoden angegangen, bei denen zuerst Detektionshypothesen extrahiert werden, welche dann im Tracking über die Zeit hinweg miteinander verbunden werden. Es wäre schwierig, die Qualität dieser Zell-Tracking Methoden im Grundsatz zu verbessern, da jeder Zelltyp, jedes die Zellen umgebende Medium und jede Mikroskopeinstellung zu Aufnahmen führen, die ihre ganz speziellen Eigenschaften und Probleme haben. Dies bedeutet leider auch, dass Tracking Methoden in absehbarer Zukunft nicht perfekt werden, sondern dass ein Korrekturlesen der Ergebnisse von Experten vorerst nötig bleiben wird. In dieser Forschungsarbeit widmen wir uns deshalb zwei anderen Aspekten: zum einen der Skalierung von bekannten und der Entwicklung neuer Lösungsmethoden für etablierte Trackingmodelle und zum anderen der Konstruktion einer speziellen Pipeline um Tuberkulosebakterien zu detektieren und zu verfolgen.

Die leistungsfähigsten *Tracking-by-Assignment* Methoden sind als Probabilistische Graphische Modelle formuliert und werden als Integer Lineare Programme gelöst. Weil diese Integer Linearen Programme aber im Allgemeinen *NP*-schwer sind, führt eine Vergrößerung des zu lösenden Problems zu einer Explosion des benötigten Rechenaufwands. Zu Beginn dieser Arbeit formulieren wir eines dieser Modelle im Stil eines Netzwerk-Fluss Problems mit Nebenbedingungen um und zeigen, dass es dadurch effizienter zu lösen wird. Da Netzwerk-Fluss Algorithmen sehr erfolgreich im Personen-Tracking eingesetzt werden, entwickeln wir eine Heuristik um zusätzliche Bedingungen – hier für Zellteilungen – in solch einer Methode zu behandeln. Das erlaubt uns qualitativ hochwertige Annäherungen an die Trackinglösung zu finden und gleichzeitig eine polynomielle Laufzeitschranke anzugeben. In unseren Experimenten bestätigt sich die geringere Komplexität dadurch, dass unsere Methode deutlich besser zu größeren Problemen skaliert. Nichtsdestotrotz ist der Ansatz single-threaded und nutzt die verfügbaren Ressourcen von Mehrprozessormaschinen noch nicht aus. Damit das Tracking parallelisiert werden kann, stellen wir eine effektive Methode vor um lange Videos in einzelne Intervalle zu zerlegen und diese unabhängig voneinander zu lösen. Durch die anschließende Optimierung eines simplen globalen Problems werden die Teillösungen so zusammengesetzt, dass an den Schnittstellen keine Unstimmigkeiten entstehen. Um noch einen Schritt weiter in diese Richtung zu gehen, schlagen wir ein Softwaredesign für *ilastik* vor, das auf Microservices aufbaut und das es erlaubt, die Berechnungsschritte für die Segmentierung, die Extraktion von Objekteigenschaften, das Klassifizieren von Objekten und dem Tracking auf mehrere Maschinen in einem Cluster oder der Cloud zu verteilen.

Letztendlich betrachten wir den Anwendungsfall des Erkennens und Trackings von Tuberkulosebakterien genauer, da es für dieses wichtige Problem bisher keine zufriedenstellende automatische Methode gab. Eine Besonderheit dieser länglichen Zellen ist, dass sie sich so nahe kommen, dass man die Individuen kaum voneinander abgrenzen kann. Wir verwenden daher ein *Tracking-by-Assignment* Modell, das beim Tracking aus einer großen Menge von teils konkurrierenden Detektionshypothesen diejenigen auswählt, die im gesamten zeitlichen Kontext am plausibelsten sind. Um diese Hypothesen zu erlangen, entwickeln wir einen neuen Algorithmus der die $M$ besten diversen Lösungen von baumförmigen graphischen Modellen mittels dynamischem Programmieren finden kann. Erste Versuche mit dieser Pipeline deuten darauf hin, dass wir damit Tuberkulosebakterien zuverlässig tracken können.

# Contents

Chapter 1

# Introduction

Ongoing advancements in microscopy and automated high-throughput pipelines [Wu et al., 2010, Höckendorf et al., 2012, Krzic et al., 2012, Jemielita et al., 2013, Keller, 2013, Santi et al., 2013, Berthet and Maizel, 2016, Sadanandan et al., 2016] allow biologists to record living stem cells and bacteria with ever increasing temporal and spatial resolution, while limiting the amount of photo-toxicity induced on the target. This drives several research areas forward, such as drug development [McKinney, 2000, Zimmer et al., 2002, Chen et al., 2006] or understanding embryogenesis. The goal of embryogenesis is to analyze the development of embryos from stem cells, asking questions like which forces play important roles [Amat and Keller, 2013] and at which point along the lineage tree cells specialize for their later task [Keller et al., 2008, Höckendorf et al., 2012, Amat et al., 2014]. To the biologists' dismay, the amount of data is growing faster than humans can analyze it, thus automated methods are required – a phenomenon that is common in today's uprise of *Big Data*. The analysis of time series microscopy scans often not only requires tracking the position and movement of individual cells or bacteria, but also the reconstruction of their full lineage [Amat and Keller, 2013]. Hence automatic tools need to detect when and where cells undergo mitosis, and which cells are the offspring of a division.

Even though tracking individual cells is deemed crucial for a lot of studies [Schroeder, 2011, Elowitz et al., 2002, Wakamoto et al., 2013], the available tools for automatic cell tracking are unfortunately not very reliable or do not scale to the actual problem size. Hence a lot of experiments are only analyzed on cell colony scale or involve time consuming manual annotation [Coutu and Schroeder, 2013]. In this work we address several aspects like scalability and applicability of single-cell tracking models as outlined in Figure 1.1, and provide easy to use software packages.

## 1.1 Algorithmic Challenges of Cell Tracking

Before we dive in, let us briefly look at the difficulties of cell tracking and available methods. From a Computer Vision perspective, tracking cells is a challenging task due to several reasons:

- Despite all the progress in microscopy techniques, the recorded time-lapse videos are subject to a low signal to noise ratio, making it hard to locate individual cells and to tell them apart. Furthermore the temporal sampling rate is often low compared to the speed

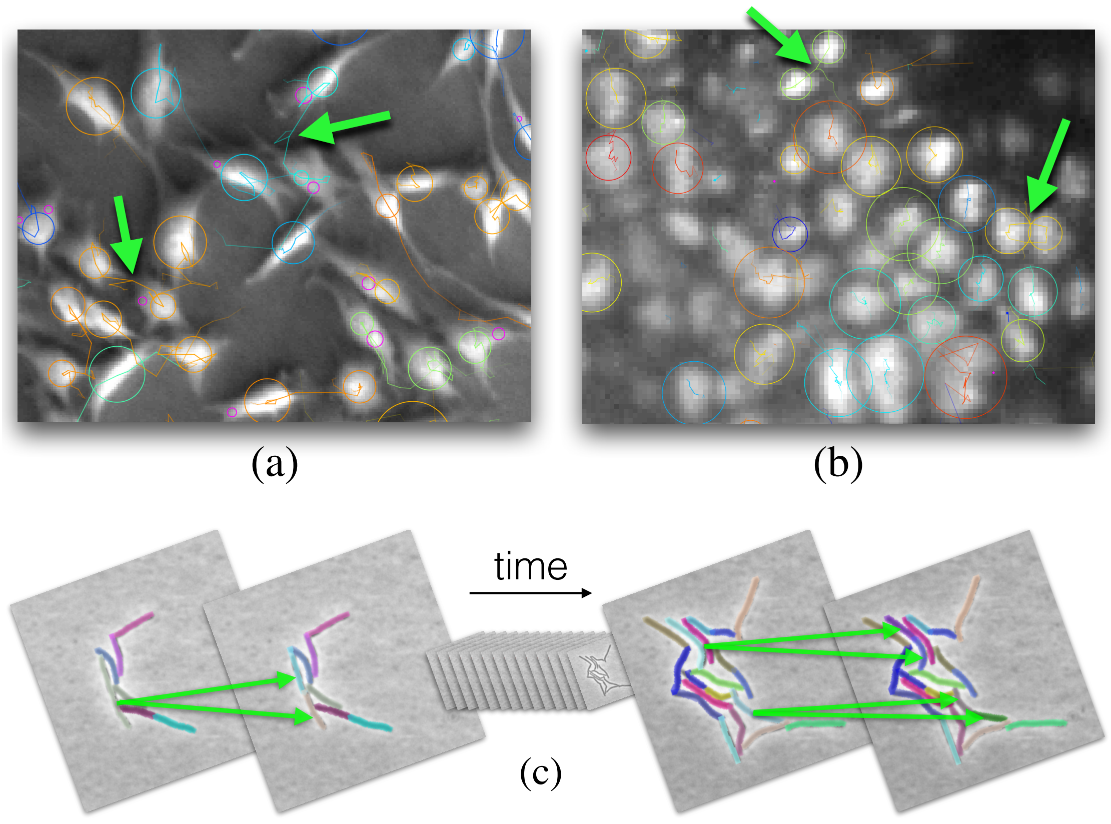


**Figure 1.1:** Tracking result on the datasets used in the course of this thesis. **(a)** 2D recording of Pancreatic rat stem cells (PSC), **(b)** 2D slice from a 3D scan of a developing Drosophila fruit fly embryo (DRO), and **(c)** mycobacteria responsible for tuberculosis. In **(a)** and **(b)**, cell detections are enclosed by a circle, and their previous and future center positions of the next 10 frames are shown as a line of the same color. Offspring of a division take on the color of their parent track, hence cells with the same color are *relatives* in this video. Purple circles denote unused detections, green arrows indicate some visible divisions. In **(c)**, bacteria detections are shown as centerlines where red dots depict possible end points. We represent tracks by colors again, but because these videos start from a single cell we assign new random colors to the children of every division. All results were obtained by the methods presented in this work.

at which cells move between frames, rendering the assignment of cell identities over time difficult.

- Cells and bacteria do not exhibit appearance features that would allow for re-identification of a cell when considering a large neighborhood – even worse, stem cells are subject to constant texture and morphology changes while advancing in the mitotic phase cycle.
- Divisions need to be detected with high accuracy: Mistakes in the lineage trees invalidate the results of all progenitors along the tree. Approaches for pedestrian tracking – a much more common task in Computer Vision – do not model division events, and hence cannot be transferred to this problem without adaptations.
- The recorded samples commonly contain a high number and density of cells with unpredictable motion patterns. The motion could *e.g.* change due to medication, and is not even fully understood for embryogenesis. Thus prior knowledge cannot be included in motion models. On the contrary, it is often exactly the change of motility that is the subject of a medication study.

Several families of tracking methods were applied or developed to deal with these challenges. We mention a few here, see [Maška et al., 2014] for an overview and comparison.

- If the temporal resolution of the recordings was high enough, the 2D+t or 3D+t volume could be segmented such that every pixel in every frame is either background or belongs to the lineage of exactly one ancestor, which is then represented as one connected branching tube-like structure in space-time. Unfortunately this does not work well in practice due to insufficient recording quality. Yet, first attempts have been made by applying multicuts [Jug et al., 2016].
- *Tracking-by-model-evolution* methods assign a state to every tracked object and follow them from frame to frame, updating their states. A state could be just the position and velocity in a basic Kalman filter, a simple shape model like a Gaussian [Amat et al., 2014], or even represent the contour of the object which is then evolved over time [Dufour et al., 2011, Maška et al., 2013]. A big advantage of that setup is that targets are detected based on their state in the previous frame while tracking. These methods can deal with larger displacement when modeling target motion, and have a nice mathematical foundation, but they struggle when targets are densely packed or have unpredictable motion because they cannot consider a larger temporal context when making local decisions. Unfortunately divisions cannot be expressed well in that framework unless one adds heuristics as in [Amat et al., 2014].
- *Tracking-by-assignment* methods assume that an over-complete set of detection hypotheses has been previously extracted, and then construct a graphical model with possible assignments of hypotheses between frames that describe the paths of all tracked targets [Bise et al., 2011, Padfield et al., 2011, Kausler et al., 2012, Magnusson and Jaldén, 2012, Schiegg et al., 2013, Jug et al., 2014, Schiegg et al., 2014, Turetken et al., 2016]. Then these methods strive to find the most likely configuration of the model, using varying optimization strategies.

Of these methods, tracking-by-assignment approaches offer the most modeling power, *e.g.* for incorporating natural constraints, and exhibit a lot of possibilities to incorporate learning from examples specified by an expert, either for detection probabilities [Kausler et al., 2012, Magnusson and Jaldén, 2012, Schiegg et al., 2013], or fully [Lou and Hamprecht, 2011]. Yet, their biggest drawback is that obtaining a good set of detection hypotheses to build a model of possible assignments is not trivial. When tracking objects in natural images, specialized detectors for *e.g.*persons or cars are often employed to generate the set of detection hypotheses [Andriluka et al., 2008, Zhang et al., 2008, Lenz et al., 2015, Pishchulin et al., 2016]. This is a valid approach for biological settings as well. A detector could *e.g.* be specialized for a certain kind of cell and can even include prior knowledge about size, shape, *etc*., such that only plausible hypotheses are chosen. However, the objects tracked in biology exhibit a wide variety of shapes and appearances, and are recorded in various settings from different microscopes (see *e.g.*the datasets used in [Maška et al., 2014]). Hence a different detector would have to be constructed for each of these scenarios, which is time consuming. A more general and more common approach is to segment the objects to track from the background. Unfortunately the image quality often does not allow for a perfect segmentation – which would be to extract exactly one detection hypotheses per object in every frame and then use Hungarian matching [Kuhn, 1955] to find the best assignments – hence there are two main methods to deal with non-optimal sets of detections:

One approach is to start from a binary segmentation of the input frames, assuming that no foreground object is missed, but allowing for unresolved clusters of objects. When multiple objects are falsely contained in a single segment this is called *undersegmentation*. Such a binary segmentation can for instance be obtained by thresholding the raw pixel intensities, or by training a classifier that predicts the probability of every pixel to be a foreground object, and then thresholding these probabilities, as in [Sommer et al., 2011]. To cope with the undersegmentation, the tracking model must allow to assign more than one object to a detection, as in [Magnusson and Jaldén, 2012, Schiegg et al., 2013, Magnusson et al., 2014]. Clusters of objects can be resolved in a post processing step.

Alternatively, one can extract an over-complete set of possibly competing detection hypotheses, and assume that for every possible cluster there are hypotheses that split it into the correct number of objects. The detections – no matter how they are extracted – could be competing in the sense that multiple hypotheses try to explain the same object or pixel in the image. The task of the tracking-by-detection method is then to pick the best globally consistent assignment of a subset of these hypotheses, such that every pixel or object is explained by at most one track. A difficulty of this methodology is that having a large amount of hypotheses can make the model overly complex and hard to solve. [Jug et al., 2014, Schiegg et al., 2014, Turetken et al., 2016] refer to tracking models that pick the best instances from a set of competing detection hypotheses as *Joint Segmentation and Tracking*. The set of segmentation hypotheses is however fixed after the initial detection step, which is why in this work we call those models *Joint Hypotheses Selection and Tracking*.

## 1.2 Focus and Structure of this Thesis

In this work we focus on tracking-by-assignment models and build on the prior work of [Schiegg et al., 2013] and [Schiegg et al., 2014, Turetken et al., 2016]. Our goal is to make this family of algorithms scale better to large videos, and to improve usability by making them more flexible and easier to apply.

- In Chapter 2 we outline and reformulate the *Conservation Tracking* model by [Schiegg et al., 2013] in terms of a constrained network flow graph, and show that this model can be solved more efficiently than the original model using existing solvers.

- We develop an approximate solver for tracking-by-assignment models with clusters in Chapter 3, which is a generalization of successive shortest paths network flow solvers, and show its favorable resource usage and scaling behavior.

- In Chapter 4 we first present a heuristic to solve subproblems of the global tracking objective in parallel and combine their results to a consistent solution. Then we propose a software architecture based on the concept of microservices that allows to parallelize and distribute the computational steps of segmenting and tracking to the cloud.

- We introduce a dynamic programming based-approach for finding the $M$ best solutions of tree-shaped graphical models subject to some diversity constraint in Chapter 5, and show that it can be useful in many Computer Vision applications.

- For elongated tuberculosis bacteria, we develop the first automated *joint hypotheses selection and tracking* pipeline that uses the method from Chapter 5 to extract competing detection hypotheses in Chapter 6. We apply structured learning to reduce the amount of hyper-parameters, and preliminary experiments indicate that the resulting tracking quality is reasonable.

- We close with a brief discussion of the results, limitations, and developed software module in Chapter 7.

Chapter 2

# Tracking Reformulation as Constrained Network Flow Model

In this chapter we briefly review the *Conservation Tracking* model [Schiegg et al., 2013], which we use as the foundation for Chapters 2 through 4. This model is also implemented in ilastik [Sommer et al., 2011, Haubold et al., 2016b], and was amongst the top three performers in the 2015 edition of the Cell Tracking Challenge [Maška et al., 2014][1]. After presenting the model, whose pipeline is sketched in Figure 2.1, we will show that its linear programming local polytope relaxation is rather loose, and hence present a reformulation as constrained network flow model that is much tighter and hence easier to solve.

## 2.1 Conservation Tracking Factor Graph Model and its Energy

*Tracking-by-assignment* problems are commonly depicted by a graph of detection and linking hypotheses. This is a trellis graph for the complete video time span, where all detections in all time steps are represented by nodes, and arcs depict possible assignments of objects across timeframes. The equivalent models presented by [Magnusson and Jaldén, 2012, Magnusson et al., 2014] and [Schiegg et al., 2013] follow that scheme. They additionally assume that detections are given by a binary segmentation of every input image. Hence there are no conflicting detection hypotheses. Because this assumption implies that there is at most one detection explaining any foreground pixel, situations arise where the segmentation is wrong and a cluster of objects is falsely represented by a single segment. We refer to these undersegmented detections as *mergers*. Both approaches cope with mergers by allowing every detection to be used by multiple tracks. Thus, tracks can merge and split – preserving the number of tracks –, appear and disappear, and additionally *divide*, which increases the number of objects in flight. Because events as merging, splitting and divisions could all be trivially modeled in terms of appearing and disappearing tracks, local evidence must be considered together with global context to obtain a probable configuration. Hence [Magnusson and Jaldén, 2012] and [Schiegg et al., 2013] set up this graph for the full video and solve it globally. Yet only Schiegg – who coined the term *Conservation Tracking* model – solves it to optimality, as we will target below in more detail. Once a solution to this model is found, *mergers* can be resolved in a post-processing step *e.g.*by

[1] *HD-Hau-GE* at http://www.codesolorzano.com/Challenges/CTC/Latest_Results.html. Accessed April 13th, 2017.

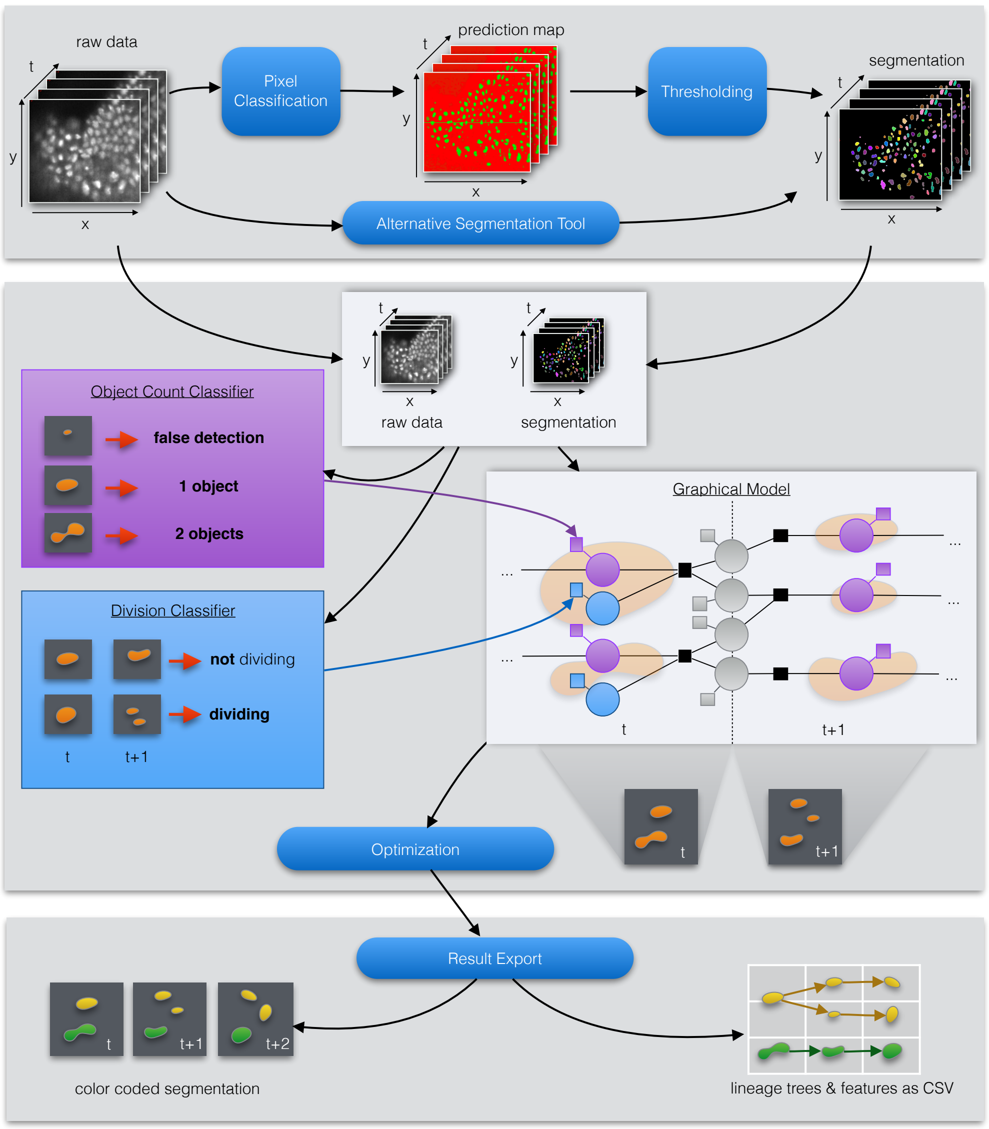


**Figure 2.1:** Overview of the tracking pipeline available in *ilastik* using *Conservation Tracking* [Schiegg et al., 2013]. **Top:** Generating a segmentation using ilastik. **Middle:** Given raw data and segmentation, two classifiers are trained to locate segmentation problems and divisions. A graphical model is built for all possible detections and their assignments, using the classifier predictions as potentials. We then find the MAP solution of the model and resolve merged detections (not shown). **Bottom:** The user can export the tracking results in several formats.

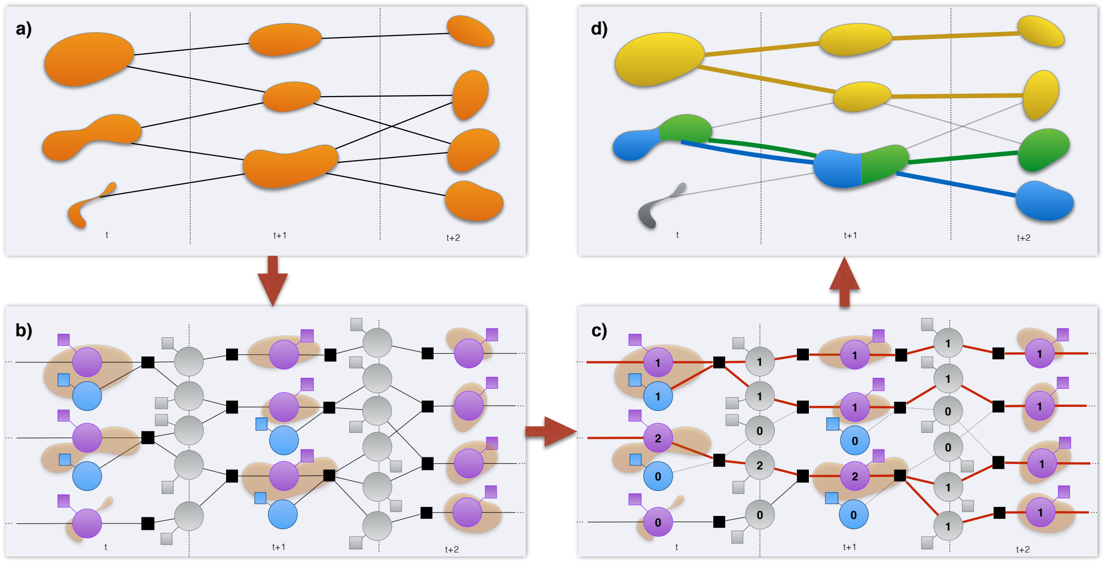


**Figure 2.2:** Illustrative example of the tracking model omitting possible appearances and disappearances. Cells can divide, merge and split, and detections can be ignored, as the solution in **d)** shows. The factor graph **b)** contains three kinds of variables: purple detection hypotheses, blue division hypotheses, and gray possible assignments between consecutive frames, all with their own unary that represents the probability for the variable to represent a certain number of cells. Black squares represent constraints that forbid invalid configurations. In the factor graph solution **c)** the globally consistent configuration is given by numbers of cells inside the nodes, and induced tracks are highlighted in red.

fitting a Gaussian Mixture Model with the appropriate number of clusters. See Figure 2.2 for a visualization of these cases in practice.

Let us briefly restate the model in factor graph notation as presented in [Schiegg et al., 2013]. Factor graphs [Kschischang et al., 2001] are a notation of probabilistic graphical models which do not only show conditional independences, but also make the factorization explicit. Circular nodes represent random variables, non-black boxes depict factors that depend on the connected random variables, and filled black boxes represent hard constraints. The factors are commonly called unary or pairwise for the cases of one or two connected variables, respectively. Each random variable can take a state or label out of a discrete label space, and hence factors can be seen as lookup tables of combinatorial size that represent the probability per variable configuration. The Figures 2.1 and 2.2 contain factor graphs that convey the general idea of the tracking model used here. Yet, for clarity they do not faithfully represent all details of [Schiegg et al., 2013]. The omitted detail is the way appearances and disappearances are modeled, namely by splitting the (purple) detection node into a disappearance and appearance node, and connecting them via a pairwise factor as shown in Figure 2.3 that replaces the detection unary but also ensures consistency by having zero probability for invalid states[2].

More formally, there are four types of nodes, whose states $\{0, \ldots, m\}$ indicate how many cells up to a maximum cluster size of $m$ are participating in the respective events:

- **Appearance** nodes $A_i \in \mathcal{A}$ represent the number of tracks that will be continued from detection $i$ on.

- **Disappearance** nodes $V_i \in \mathcal{V}$ on the other hand sum the number of tracks that arrive at

[2]Section 2.3 introduces an alternative for handling appearance and disappearance

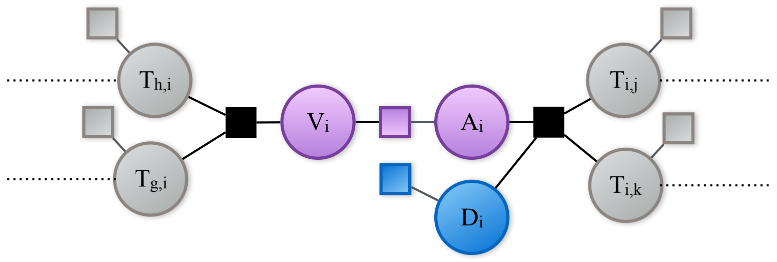


**Figure 2.3:** Detailed view of how in [Schiegg et al., 2013] a detection node $i$ is separated into a disappearance (or vanishing) node $V_i$ and an appearance variable $A_i$, while the purple pairwise factor in between handles both, appearance, disappearance and detection cost as well as forbidding that the number of objects changes within a *merger*.

this detection. If the the disappearance and appearance node agree in the number of tracks, then this is the number of tracks passing through the detection.

- **Transition** nodes $T_{i,j} \in \mathcal{T}$ model how many objects move from the source detection (its disappearance node $V_i$) to the target detection (its appearance node $A_j$).
- **Division** nodes $D_i \in \mathcal{D}$ are always coupled to a specific detection $X_i$ and allow that detection to divide if and only if the detection contains one object, hence their label space is $\{0, 1\}$.

Following [Schiegg et al., 2013], to incorporate local evidence from the images, we train Random Forest classifiers on a few annotated detections and divisions and apply them to the full video. Transition probabilities are modeled based on center of mass distance. Appearance and disappearance probabilities are fixed constants within the video, but decrease linearly in a margin to the boundaries of the field of view.

Instead of finding the most probable solution, we apply the negative logarithm to obtain an energy minimization problem. This problem is an integer linear program (ILP). Despite them being NP-hard in general, they can be approached with commercial general purpose solvers (CPLEX, Gurobi[3]) for reasonably sized problems. Let $\mathcal{Y}$ be the set of all possible configurations of all variables $\mathcal{V} \cup \mathcal{A} \cup \mathcal{T} \cup \mathcal{D}$ in the factor graph. We denote by $\mathbf{y}_i$ the state of variable $i$ in a configuration $\mathbf{y} \in \mathcal{Y}$. Then the minimal energy configuration can be found by solving:

$$\mathbf{y}^* = \operatorname*{argmin}_{\mathbf{y}\in\mathcal{Y}} E(\mathbf{y}) = \operatorname*{argmin}_{\mathbf{y}\in\mathcal{Y}} \sum_{V\in\mathcal{V}} \sum_{A\in\mathcal{A}} E_X(\mathbf{y}_V, \mathbf{y}_A) + \sum_{T\in\mathcal{T}} E_T(\mathbf{y}_T) + \sum_{D\in\mathcal{D}} E_D(\mathbf{y}_D) \tag{2.1}$$

subject to hard constraints that ensure consistency[4]:

- The number of objects in a detection match the number of active incoming and outgoing transitions unless appearance and disappearance events account for the change. This is often referred to as **flow conservation**, hence the name *Conservation Tracking*.
- Only detections containing one cell and having exactly two distinct active descendants can divide.

[3] http://www-03.ibm.com/software/products/en/ibmilogcpleoptistud and http://www.gurobi.com

[4] See [Schiegg et al., 2013] for details.

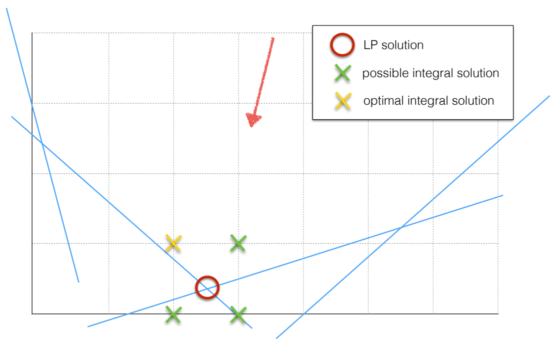


**Figure 2.4:** Linear programming relaxation problems on a 2D toy example. Blue lines depict constraint hyperplanes, the red arrow visualizes the direction in which the objective function decreases. The optimal LP solution lies on a vertex of the polytope defined through the hyperplanes. When trying to decode an integral solution from the LP solution, any of the solutions on the (gray) integer grid in the vicinity of the LP solution could be found. This becomes much more complicated in higher dimensions.

- Mergers cannot partially (dis-)appear.

Note that because $\mathbf{y}$ must be from the set of possible configurations $\mathcal{Y}$ all variables have to take an integral state.

## 2.2 LP Relaxation

Before we look at the problems of the *Conservation Tracking* ILP formulation in more detail, let us briefly restate what linear programming relaxation is. Linear programming (LP) denotes a family of mathematical optimization problems, minimizing (or maximizing) an objective function that linearly depends on the $n$ variables $\mathbf{x} \in \mathbb{R}^n$ in question, in the form:

$$\begin{aligned} & \min_{\mathbf{x}} \mathbf{c} \cdot \mathbf{x}, \\ s.t. \quad & \mathbf{A}\mathbf{x} \leq \mathbf{b}. \end{aligned} \tag{2.2}$$

The constants $\mathbf{c}$ define the direction in the solution space in which the value of the objective function increases. $\mathbf{A}$ is called a constraint matrix, and the constraints define hyperplanes with offsets $\mathbf{b}$ from the origin that disallow the solution to lie on the back side of any hyperplane, see Figure 2.4. Solutions that obey all constraints are called feasible. The space containing all feasible solutions – bounded by the constraint hyperplanes – is called polytope. It can be shown that the optimal solution lies on a vertex of that polytope. LPs can be solved optimally in polynomial time using *e.g.* the network simplex algorithm that walks along vertices of this polytope.

Integer linear programs such as (2.1) are linear programs with additional constraints that ensure that every variable only takes on integral values. This seemingly simple change makes the problem NP hard in general. Solvers approach ILPs using *branch-and-bound* techniques, branching on the different states of a variable, bounding the energy by the best found objective function value, and then recurse along the most promising path.

A common approach to solve ILPs is to ignore the integrality constraints and to solve this relaxed LP. From the found vertex of the polytope defined by the remaining constraints, one tries to *decode* the optimal integral solution, *e.g.* by rounding. If the optimal vertex of the LP is integral – and hence coincides with the optimal ILP solution – the relaxation is said to be *tight* around the optimum. In that situation, in Figure 2.4 the red circle would coincide with an integer grid point.

The experiments in Table 2.2 at the end of this chapter show that the LP relaxation of (2.1) is not tight. Only around $35\%$ of the variables have integral values. This might be due to the pairwise term between the vanishing and appearance node, which obfuscates the required consistency between number of incoming and outgoing objects. In the next section we will therefore present a model that removes the decoupling of detection nodes and show that this reformulation has a much tighter LP relaxation, which makes the problem easier to solve.

## 2.3 Network Flow Formulation

The tracking problem is very related to a well studied family of graph algorithms called *network flow* [Bertsekas, 1998]. Roughly speaking, for these algorithms edges in the graph need an additional property that is a capacity. If one imagines the graph as a network of pipes, where nodes are junctions, and designates a source and a sink node, *max-flow* algorithms can for instance determine the maximum amount of flow that can be pushed through the network from source to sink. If the edges incur a cost when flow traverses them, *min-cost* flow algorithms find the minimal cost routing strategy to send a given amount of flow through the graph in polynomial time. Because of the flow-based nature of these methods, they inherently obey the flow-conservation constraints that we mentioned above. Another important property is that as long as the capacities of all edges are integral, the optimal solution will be to send an integral amount of flow along every edge. This corresponds to the case when an ILP has a totally unimodular (TUM) constraint matrix. It can be shown that then the result of the LP relaxation is always integral, hence the relaxation is always tight [Bertsekas, 1998].

Network flow algorithms have been extensively applied to tracking without divisions [Zhang et al., 2008, Pirsiavash et al., 2011, Lenz et al., 2015]. Then, appearances and disappearances are modeled by direct connections to source or sink respectively, as shown in Figure 2.5[5]. Such network flow models have been applied to cell tracking in [Padfield et al., 2011, Turetken et al., 2016]. But to handle divisions in a network flow, additional constraints need to be integrated which destroy the TUM property of the constraint matrix, and hence we lose guarantee of integral solutions of the LP relaxation. Hence prior methods either round the fractional solution or solve the problem as ILP. The tightness of the LP relaxation of such models has not been studied yet. We will now state a network flow reformulation of the *Conservation Tracking* ILP (2.1), and compare them in the following Section 2.4.

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. [6] By $Source$ and $Sink$ we denote the source and sink node. Let $\mathbf{y} \in \mathcal{Y}$ be a valid labeling, that is,

[5]For this formulation the factor graphs in Figures 2.1 and 2.2 are only missing transitions from a virtual source node for appearances and divisions, and to a sink for disappearances.

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

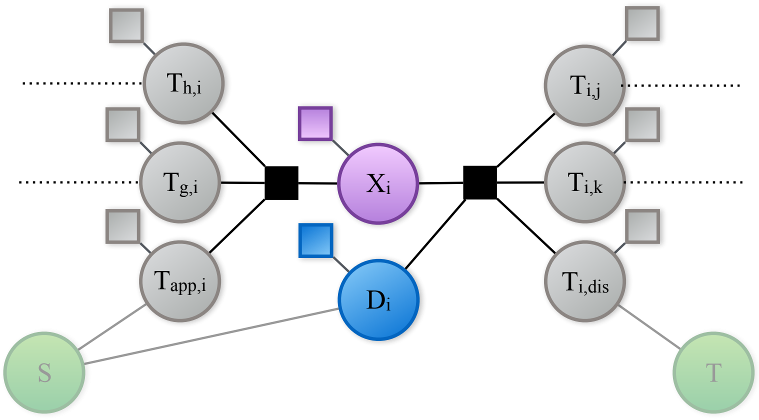


**Figure 2.5:** In the constrained network flow graph, the detection node $X_i$ is only a single node with unary factor, and appearances as well as disappearances are modeled as transitions $T_{app,i}$ from an auxiliary source (green $S$) or $T_{i,dis}$ to an auxiliary sink (green $T$) respectively.

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 $V$ 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

$$\mathbf{y}^* = \operatorname*{argmin}_{\mathbf{y}\in\mathcal{Y}} E(\mathbf{y}) = \operatorname*{argmin}_{\mathbf{y}\in\mathcal{Y}} \sum_{X\in\mathcal{X}} E_X(\mathbf{y}_X) + \sum_{T\in\mathcal{T}} E_T(\mathbf{y}_T) + \sum_{D\in\mathcal{D}} E_D(\mathbf{y}_D) \tag{2.3}$$

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

$$+ \sum_{D\in\mathcal{D}} \sum_{k\in\mathcal{L}(D)} \theta_D(k)\mathbb{1}[\mathbf{y}_D = k] \tag{2.4}$$

subject to:

**Flow conservation:** (2.5)

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

**Division:** (2.6)

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

**Merger** (2.7)

$$\forall_{X\in\mathcal{X}} \forall_{\substack{I\in\mathcal{I}(X)\\ I\neq\mathcal{A}pp(X)}} : \mathbb{1}[y_I = 0] + \mathbb{1}[y_{\mathcal{A}pp(X)} = 0] \geq 1$$

$$\forall_{X\in\mathcal{X}} \forall_{\substack{O\in\mathcal{O}(X)\\ O\neq\mathcal{D}is(X)}} : \mathbb{1}[y_O = 0] + \mathbb{1}[y_{\mathcal{D}is(X)} = 0] \geq 1$$

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 (2.4) is a linear combination of configuration $\mathbf{y}$ and unary potentials $\theta$, 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 (2.5). Furthermore, a detection cannot divide more often than it contains targets (2.6). Lastly, the clusters represented by mergers are not allowed to grow or shrink by cells appearing or disappearing (2.7). The division constraint (2.6) is the key difference to a standard min-cost flow problem [7]. 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. Yet, as we will see in the next section, the LP relaxation of this model is much tighter, nearly all indicator variables are taking on integral states. Additionally, this leads to shorter ILP solver runtimes, up to factors of 2.

## 2.4 Runtime and Tightness Comparison

We evaluate the proposed reformulation on two challenging datasets from developmental biology, a 3D+t drosophila scan [Schiegg et al., 2014] and 2D+t pancreatic rat stem cells (PSC) presented in [Rapoport et al., 2011], both publicly available with ground-truth. Both can be seen in Figure 1.1. 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 [Schiegg et al., 2013] and [Magnusson et al., 2014], we assign to each detection the probability for containing a certain number of cells $P_{det}(k)\ \forall k \in [0, m]$ [8] , as well as a probability for division $P_{div}$. These probabilities are predicted by Random Forest classifiers which were trained on the same subset of the data as described in [Schiegg et al., 2014]. 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 [Schiegg et al., 2013] included in ilastik [Sommer et al., 2011] 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.

We compare the tracking performance for two solutions 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. Firstly, we evaluated the quality of all models with respect to the groundtruth of both datasets. Table 2.1 indicates that all models produce similar results, subject to minimal variations which can be due to the allowed relative optimality gap[9] of $0.05$ for Drosophila and $0.01$ for PSC, as well as the fact that the optimum must not be unique and here we compare based on the resulting configuration.

We omitted before that the network flow optimality guarantees only hold as long as the energies are convex with respect to the state of a node. We achieve this property by finding a convex upper envelope $\bar{\theta}_i$ to each potential $\theta_i$ independently. To verify that this does not corrupt the model, we compare the original *Conservation Tracking* ILP (abbreviated by *CT*) with the

[7]The merger constraint violates the TUM property as well, but the division constraint is what makes cell tracking hard, clusters could be handled differently.

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

[9]ILP solvers usually apply a branch-and-cut algorithm and keep track of the best integral $y$ and fractional $\tilde{y}$ solutions they find. The relative optimality gap is then $\frac{y-\tilde{y}}{y}$.

| Dataset | CT | Flow | cF |
|---|---|---|---|
| DRO | 93.40 | 93.67 | 93.70 |
| PSC | 91.20 | 90.80 | 91.00 |

**Table 2.1:** F-measure of the different models with respect to a manually curated groundtruth. Note that subtle differences can occur due to the allowed optimality gap of $0.05$ for the ILP solvers. The results obtained by the different models and after convexifying the energies are extremely similar.

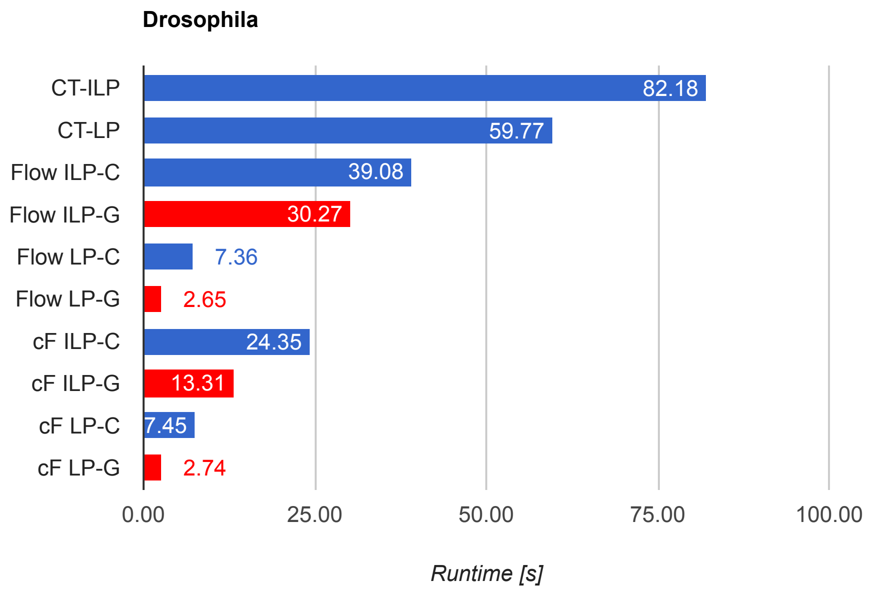
| Dataset | CT | Flow | cF |
|---|---|---|---|
| DRO | 31.58 | 93.02 | 99.21 |
| PSC | 37.10 | 96.70 | 98.50 |

**Table 2.2:** Percentage of integral variables in the solution of the LP relaxation

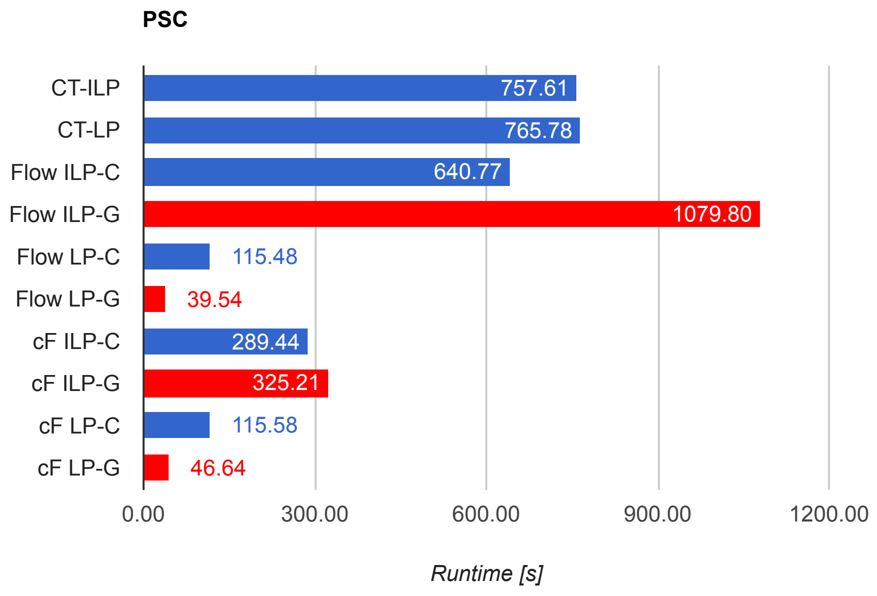
reformulation as constrained network flow (*Flow*), and the reformulated model with convex energies (*cF*). The last column of Table 2.1 shows that the cost convexification does not impact the quality of the final tracking results when applied to the energies of either datasets used.

The runtime of the different formulations and solvers is presented in Figure 2.6. All timings were performed on a machine with $2\times$ Intel Xeon E5-2650 with 10 cores at 2.3Ghz and 256GB RAM. We ran the experiments 5 times and present the median. The deviation between the runs was very little in most cases ($\sigma \leq 0.5s$ for Drosophila and $\sigma \leq 6s$ for PSC). However, the runtimes of the CT model's LP relaxation had a standard deviation of $\approx 300s$ for both datasets, which is why the mean would have been misleading. The experiments show that by switching from the CT model to the constrained network flow ILP the problems can already be solved faster and especially the LP relaxations become easier to optimize. But when the energies are replaced by a convex upper envelope, the ILP and LP solving times drastically improve, as seen in the last four rows of Figure 2.6. This trend also shows in the fraction of integral variables in the solutions of the LP relaxations in Table 2.2. Unfortunately, even though nearly all variables obtain an integral value when using convex costs, the solutions cannot be used directly. When rounding the remaining fractional variables the solutions are valid solutions to the ILP but violate constraints. Still, we evaluated the percentage of agreement with the ILP solution by comparing the states of individual nodes, see Table 2.3.

It would be desirable to make use of the fact that the LP solution is so close to the ILP solution. There are several ways how this could be achieved, for instance by developing a constraint-aware rounding scheme, by softening the constraints by Lagrangian relaxation and only gradually enforcing them [Butt and Collins, 2013], by adding constraints in a cutting planes fashion [Jug et al., 2016], or by developing a heuristic solver as *e.g.* [Magnusson and Jaldén, 2012]. In the next section we will present a network flow-based approximate solver that generalizes that of [Magnusson and Jaldén, 2012]. This choice comes with the additional benefit that no costly license for CPLEX or Gurobi is required, reducing the hurdle to make automatic tracking methods easily available to all biologists.



(a)



(b)

**Figure 2.6:** Runtimes of the *Conservation Tracking* (CT) and flow-based (Flow) ILP formulations and their LP-relaxations. For CT only CPLEX times are available, all other experiments denote by suffix *C* or *G*, and colors blue and red, whether they use CPLEX or Gurobi. The rows labeled *cF* show results for the flow-based model with convex energies

| Dataset | Flow | cF |
|---|---|---|
| DRO | 93.77 | 99.21 |
| PSC | 97.12 | 98.93 |

**Table 2.3:** F-measure of the rounded solution of the LP relaxation with respect to the ILP solution. The constrained network flow model with convex energies yields LP relaxations whose rounded solution agrees with the best ILP configuration at $99\%$ of the nodes in the factor graph. The table shows that by using convex energies, the agreement between the rounded LP and ILP solutions can be increased significantly. Note however, that due to the rounding step these solutions violate constraints of the tracking model and can hence not be used to extract full cell lineages.

Chapter 3

# Conditioned Residual Capacity Successive Shortest Paths Solver

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 introduced in the previous chapters – finding the solution to a global tracking-by-detection model is NP-hard. In this Chapter[1], we present a flow-based approximate solution to the cell tracking model presented in Chapter 2. 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.

## 3.1 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) [Bise et al., 2011, Schiegg et al., 2013, Maška et al., 2014]. 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 [Zhang et al., 2008, Pirsiavash et al., 2011, Lenz et al., 2015, Berclaz et al., 2011]. These methods provide a polynomial runtime guarantee and are very efficient in practice, while solving the problem to global optimality. Unfortunately,

[1]This Chapter is an extended version of [Haubold et al., 2016a].

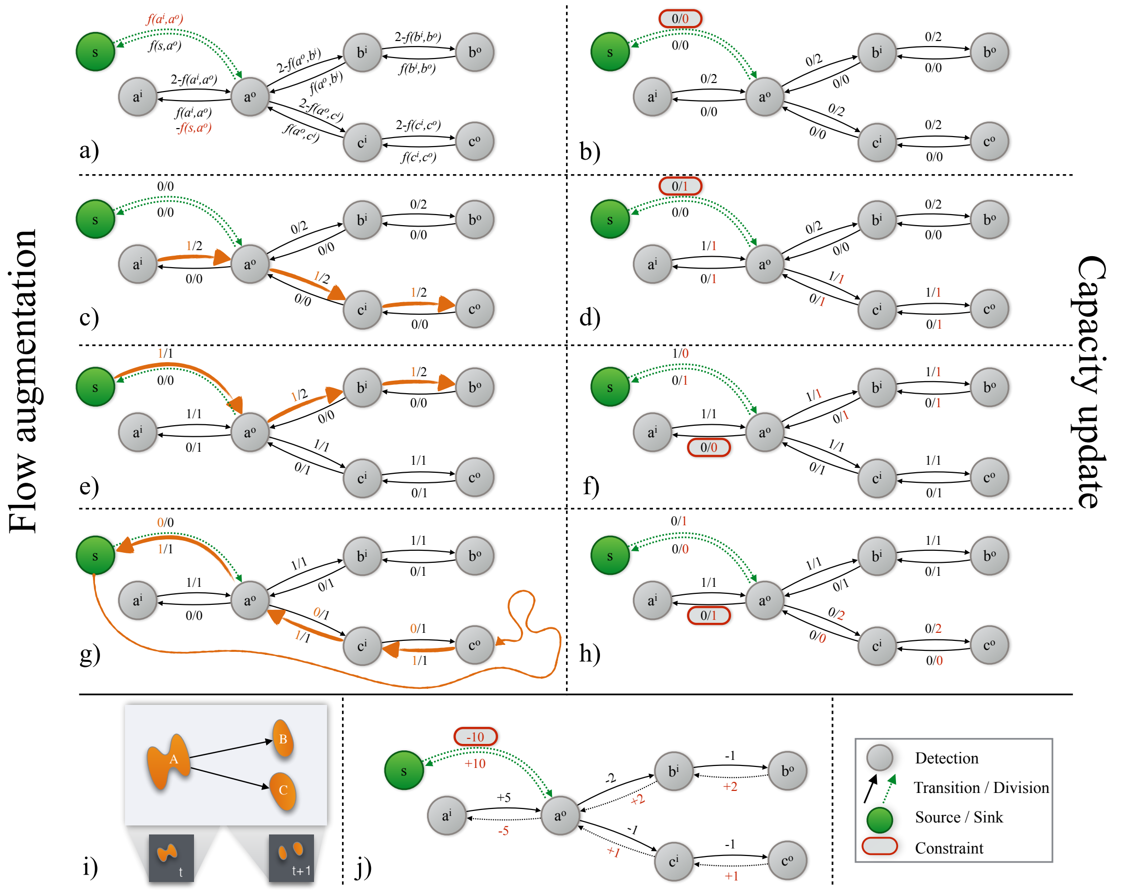


**Figure 3.1:** Case study: A minimal example excerpt of a trellis graph **i)** and a few iterations of the proposed constraint-aware flow algorithm. **a)** shows how residual arc capacities $c^r(u,v)$ are derived from the flow $f$. In general for forward arcs this is $c^r(u,v) = c(u,v) - f(u,v)$ and 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.

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, and hence optimal. Some attempts have been made to apply min-cost flow solvers to network flow problems with side constraints nevertheless [Padfield et al., 2011, Butt and Collins, 2013], but they mostly resort to rounding to finally obtain an integral solution.

In this chapter, 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. 3.1. This leads to a polynomial time algorithm that empirically exhibits attractive anytime performance and gives close to optimal results.

## 3.2 Related Work

Many tracking-by-detection models link the detections of a previously acquired per-frame segmentation between pairs of frames [Kuhn, 1955] or create short chains of detections and stitch them [Xing et al., 2009, Castanon and Finn, 2011, Jaqaman et al., 2008]. Others build a model spanning the entire time sequence to find a globally optimal configuration [Zhang et al., 2008, Pirsiavash et al., 2011, Lenz et al., 2015, Andriyenko et al., 2012, Brendel et al., 2011]. Standard tracking-by-detection expects all targets to be detected individually, which is not necessarily the case. [Wang et al., 2014] 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 [Bise et al., 2011, Schiegg et al., 2013, Padfield et al., 2011, Magnusson et al., 2014] 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. [Bertsekas, 1998, Ahuja et al., 1988, Cormen, 2009]. Multi-target tracking can be solved using such a min-cost flow setup as shown in the seminal work by Zhang and Nevatia [Zhang et al., 2008]. 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, [Pirsiavash et al., 2011] propose to use the SSP algorithm and add tracks as long as they lead to a lower cost solution. Berclaz *et al.* [Berclaz et al., 2011] improve on the runtime by using K-shortest paths instead of a single shortest path in each iteration. Lenz *et al.* [Lenz et al., 2015] 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.* [Butt and Collins, 2013] 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 [Padfield et al., 2011] 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.* [Magnusson and Jaldén, 2012, Magnusson et al., 2014] – who showed outstanding performance at the 2013 and 2014 ISBI Cell Tracking Challenges [Maška et al., 2014] 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 [Magnusson et al., 2014].

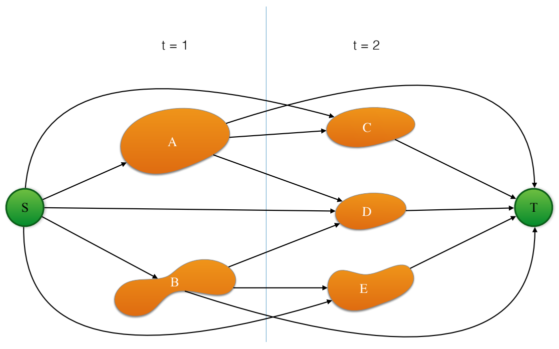
One additional complication is that in microscopy data, it is often not obvious from a single image how many objects are in a cluster. The study [Rapoport et al., 2011] revealed that under-segmentations are the prevailing segmentation error in cell tracking pipelines, and so here we follow the tracking-by-detection model presented in Chapter 2 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 [Bertsekas, 1998].

## 3.3 Tracking Model

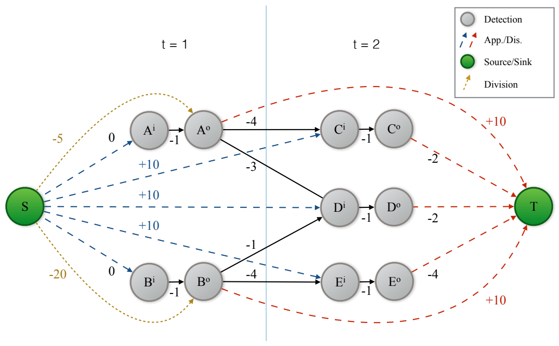
We use the model of [Schiegg et al., 2013] as shown in Section 2.3, but for the sake of brevity in our discussion we disregard the constraint which disallows the appearance/vanishing of merged detections(2.7)[2].

For all ILP results presented in the evaluation section of this work, we build the model as stated in (2.4), add equivalent constraints to those in [Schiegg et al., 2013], and solve it with the ILP solver Gurobi.

[2]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 3.4.



(a) Trellis Graph



(b) Flow Graph

**Figure 3.2:** 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 (2.6). Exemplary costs are written alongside the arcs.

### 3.3.1 Network Flow Graph

Let us now present a transformation of the ILP – first without division constraints (2.6) – into an equivalent network flow graph $\mathcal{G} = (\mathbf{V}, \mathbf{E})$, as outlined in Fig. 3.2 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 [Zhang et al., 2008]. 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 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 $\mathcal{D}$.

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

## 3.4 Approximate Min-Cost Flow: Conditioned Residual Capacities

In the preceding section we blithely ignored the division constraint (2.6). This section shows how to account for that constraint in a min-cost flow setup. Recent work [Pirsiavash et al., 2011, Lenz et al., 2015, Berclaz et al., 2011] on solving the multi-target tracking problem as min-cost flow employed the *successive shortest paths* algorithm [Ahuja et al., 1988, 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.

### 3.4.1 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 [Ahuja et al., 1988, 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.

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

**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
```

## 3.4.2 Successive Shortest Paths with Conditioned Residual Capacities

In section 3.3.1 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 3.1). 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 3.1 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 3.1 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 [Ahuja et al., 1988, p. 104], thus finds a global optimal solution.

## 3.4.3 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 ($\mathcal{O}(|\mathbf{V}|log|\mathbf{V}| + |\mathbf{E}|)$) Dijkstra algorithm to find the shortest path based on these *reduced costs* $w_{>0}$ [Bertsekas, 1998, p. 97], which is used by [Lenz et al., 2015]. 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 [Cormen, 2009]. We base our BF implementation on the LEMON Library [Jüttner et al., ], 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 $\alpha$ iterations of BF. In our experiments we use $\alpha$ 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 [Lenz et al., 2015] 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`.

### 3.4.4 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 $\mathcal{O}(P * N * |\mathcal{X}|(3 + L)) = \mathcal{O}(P * N^2)$.

### 3.4.5 First-order Residual Graph Approximation

Magnusson *et al.* [Magnusson et al., 2014] 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 [Cormen, 2009, p. 592]. They also adjust the arc costs and availability in each iteration according to the current tracking solution and constraints.

Even though [Magnusson et al., 2014] 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 $\mathcal{G}^r(f_i) = (\mathbf{V}, \mathbf{E}^r_i)$ be the residual graph at iteration $i$. Then the set of arcs directed forward in time is given as $\tilde{\mathbf{E}}^r_i = \{(k,l)|(k,l) \in \mathbf{E}^r_i, t(k) < t(l)\}$. A shortest path $\tilde{\mathcal{P}}$ between two nodes found in the restricted residual graph $\tilde{\mathcal{G}}^r(f_i) = (\mathbf{V}, \tilde{\mathbf{E}}^r_i)$ 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 $\mathcal{O}(|\mathbf{V}| * |\mathbf{E}^r_i|)$ to $\mathcal{O}(|\mathbf{V}| + |\tilde{\mathbf{E}}^r_i|)$ for a larger optimality gap, which can be seen in the results section.

To allow the algorithm to escape from local minima, [Magnusson et al., 2014] introduce *swap arcs*, which we will now restate using residual graph terminology. On top of $\tilde{\mathbf{E}}^r_i$ they instantiate every possible 3-arc sub-path of the residual graph $\{(k,l),(l,m),(m,n)\} \in (\mathbf{E}^r_i)^3$

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

– 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 [Magnusson et al., 2014] 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 [Magnusson et al., 2014] and then use `crcSSP` to find negative weight cycles and reduce the total energy even further.

## 3.5 Experiments

In this section we evaluate the performance of the presented algorithm in terms of tracking quality and resource usage, show the impact of the proposed runtime improvements, and show the scaling behavior with problem size. We again use the Drosophila and PSC datasets as presented in Section 2.4, and use the convex upper envelope fitted to the original unaries as energies. As briefly indicated before, the tracking model presented in Section 3.3 is a slight simplification of the model in [Schiegg et al., 2013] and Section 2.3. For the experiments we use the full model where we handle additional constraints similar to the division constraint.

We use two different machines for the experiments. One of them is a laptop with a 2.8GHz Intel Core i7 and 8GB RAM, and the other is a powerful workstation with two 2.3Ghz Intel Xeon E5-2650 CPUs and 256GB RAM. To make the comparison fair, we force CPLEX and Gurobi to use only one thread, hence all experiments run single threaded.

Table 3.1 shows the results – using the same tracking quality measure by counting agreement on frame-by-frame assignments as in Section 2.4 – for the first and second order residual approximation as presented in [Magnusson et al., 2014], our proposed new method using the full residual graph, and the ILP solution found with Gurobi. In Figure 3.3 we compare the anytime performance of the same solvers on the laptop but also include CPLEX. 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. To see the scaling behavior we artificially replicate the bigger PSC model 2, 4 and 8 times and compare the ILP solver runtimes with that of `crcSSP-i-o` on the workstation in Figure 3.4. And because there were significant runtime deviations between the laptop and the workstation, Figure 3.5 provides those runtimes of the three best performing solvers on both datasets. We implemented all methods and models ourselves in C++, used the Lemon graph library [Jüttner et al., ] as base for our improved BF, and OpenGM to interface Gurobi and CPLEX.[6]

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

[6]`http://github.com/opengm/opengm`, `http://www.gurobi.com` and `http://www-03.ibm.com/software/products/en/ibmilogcpleoptistud`.

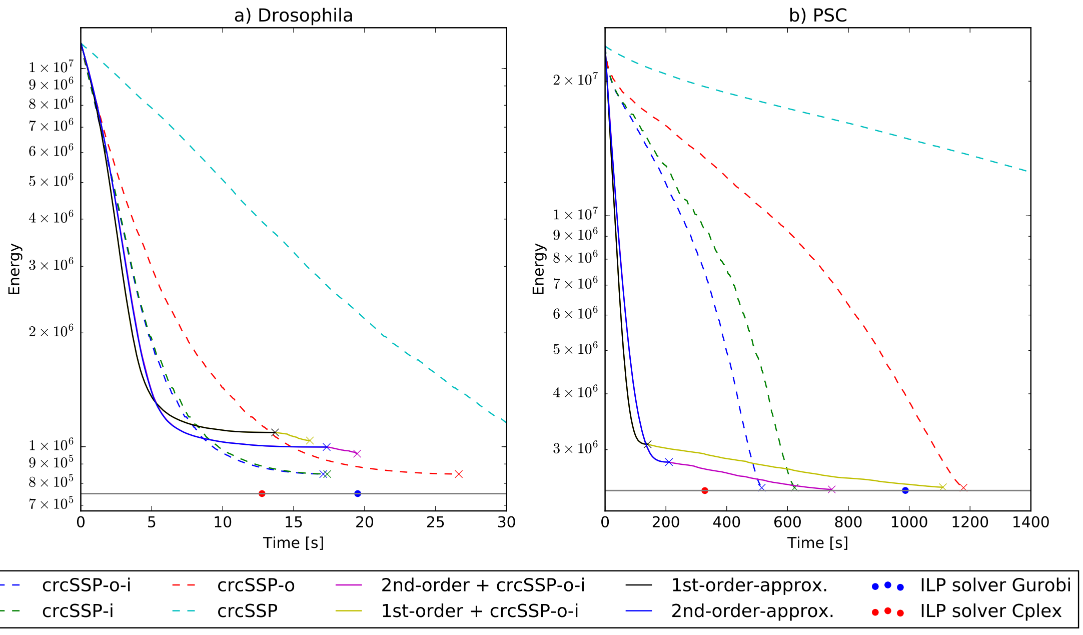


**Figure 3.3:** Anytime performance of the optimal ILP solver, the residual graph approximations by [Magnusson et al., 2014], and our proposed `crcSSP` solvers on two datasets. The `crcSSP` performance improvements of ordering nodes (`-o`) and initializing BF to update only part of the nodes (`-i`) turn out to have a strong impact on the runtime. Refining the solutions found by [Magnusson et al., 2014] by `crcSSP` cannot compete with running `crcSSP-o-i` throughout. Run on the laptop.

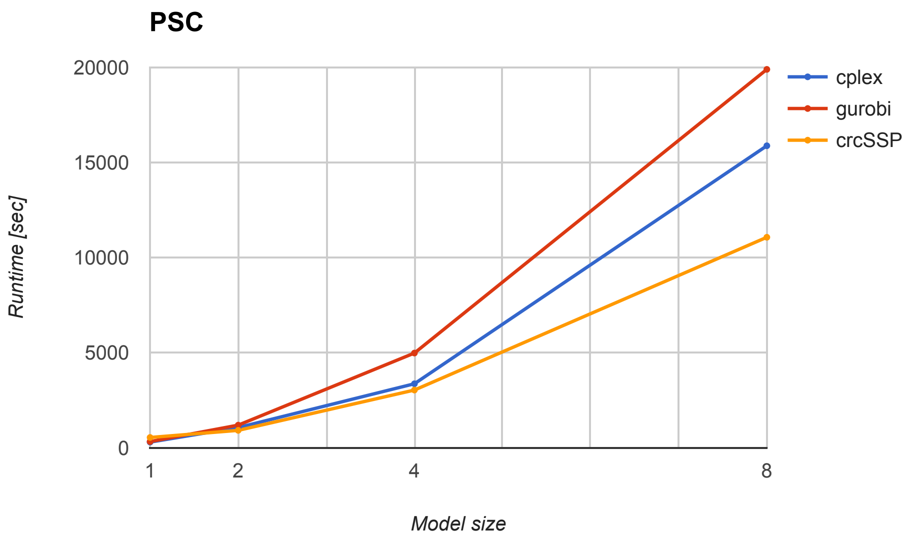


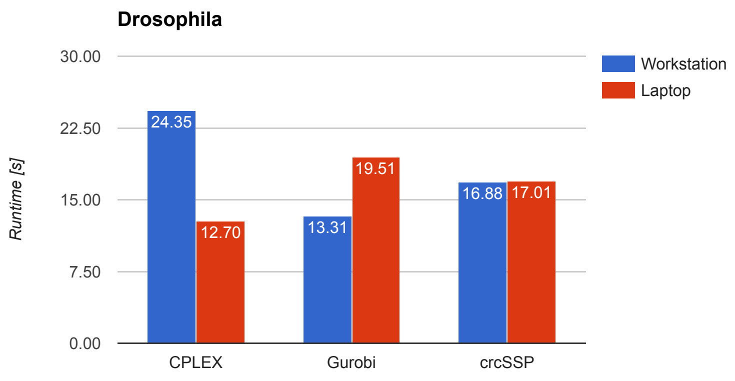
**Figure 3.4:** When replicating the bigger PSC model several times, the polynomial scaling of `crcSSP` pays off compared to CPLEX and Gurobi. Run on the workstation.

## 3.6 Discussion

Figure 3.3 shows that the proposed `crcSSP-o-i` algorithm yields a favorable trade-off between runtime and solution quality. While Magnusson's dynamic programming shortest path search [Magnusson et al., 2014] 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, on the laptop our algorithm outperforms Gurobi in terms of runtime on both datasets. For the larger PSC dataset, our algorithm finishes in about only half the runtime of Gurobi, 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 replicating the PSC model and solving it on the workstation, as seen in Figure 3.4, where Gurobi takes nearly twice as long for a model eight times as big. While CPLEX is much faster than `crcSSP` and Gurobi for the small models on the laptop in Figure 3.3, it also suffers from the NP-hardness of the ILP in the scaling experiment on the workstation in Figure 3.4. In any case, both ILP solvers perform surprisingly good for these large problems.

One striking observation was that there is a large deviation of runtimes between laptop and workstation for Gurobi and CPLEX, as seen in Figure 3.5. The workstation has more RAM and also a much bigger CPU cache, but a slower clock speed than the laptop. Because the



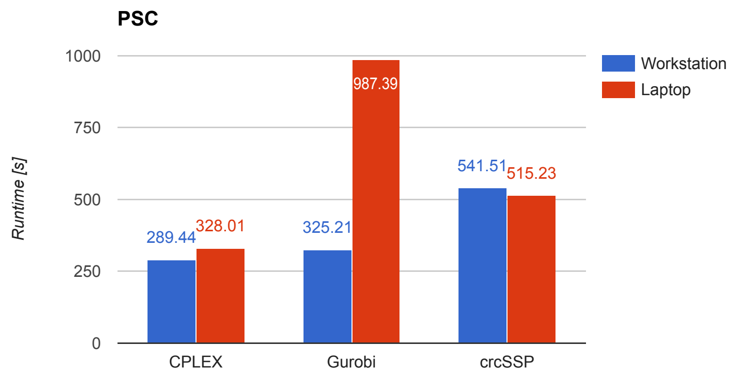


**Figure 3.5:** Runtimes depending on the used machine. Because CPLEX and Gurobi are closed source libraries, it is hard to tell whether the runtime differences are due to the different CPU cache sizes or clock speeds. Our `crcSSP` behaves quite similar on both machines.

RAM usage was rather low (bottom of Table 3.1), this is unlikely to be the limiting factor. For `crcSSP-o-i` it seems that the larger cache of the workstation can make up for the lower clock speed, yielding similar performance. Gurobi on the other hand seems to strongly benefit from the larger cache, whereas CPLEX is much slower on the workstation for the Drosophila model. Unfortunately CPLEX and Gurobi are closed source, so we cannot investigate further what makes the runtimes differ that much.

As one would expect, Figure 3.3 also shows that the first and second order residual graph approximations are fast, but cannot find a very low overall energy. Feeding the solutions found with [Magnusson et al., 2014] as initialization into `crcSSP` allows to improve the energy further, but because then 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 3.1 reveals that our proposed solver does not only exhibit an attractive anytime performance, but that it also produces very accurate tracking results, which are on 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.

## 3.7 Conclusion

In this chapter we 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 and scaling behavior. The core idea is well suited to be adapted to other types of constraints. We have made our code for the ILP model (`github.com/chaubold/multiHypothesesTracking`) and the presented solver publicly available (`github.com/chaubold/dpct`). Both solvers are integrated into *ilastik*, hence a robust automated tracking method is now available to everyone even without a Gurobi or CPLEX license.

| 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 3.1:** 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 laptop with 2.8GHz Intel Core i7 with 8GB RAM

Chapter 4

# Scaling up in the Cloud

The tracking methods that we have presented in the previous chapters required that the problem description fully fits into the host machine's RAM. This is not necessarily true. Also, the tracking runtimes grow quadratically with the number of nodes in the graph even if we apply the flow-based solver from Chapter 3, for ILP solvers the complexity increase is even worse. In Section 4.1 we will thus present a way to track parts of the video independently, and how to combine the results such that no discrepancies occur at the split points.

On top of that, we have taken for granted in Chapters 2 and 3 that we need to obtain a pixel-wise segmentation of every frame, and compute features of all connected components so that for each segment we can predict probabilities whether it divides or is a cluster. Depending on the resolution of the video, and especially if every frame is a 3D scan, performing these operations based on image data requires much more resources in terms of memory and processing time than tracking. *ilastik* provides these operations and goes to great lengths to parallelize[1] these steps on all CPUs of a single machine, but its architecture is not designed to scale to multiple machines in a cluster or the cloud. A recent trend to distribute workload amongst multiple workers – but also to decompose a monolithic application – is to use microservices [Richardson, 2015, Daya et al., 2016]. By applying these ideas to the processing steps performed by *ilastik* we propose a microservices-inspired architecture in Section 4.2 and show that even a prototype has attractive scaling behavior.

## 4.1 Tracking in Parallel by Graph Decomposition

Kalman filtering or sliding window based tracking methods have the advantage over globally optimal models that they scale linearly with the length of the video because they always consider a fixed number of frames as context. Hence they can also be applied in an online setting. We on the other hand approach cell tracking in an offline setting and would like to scale to longer videos by parallelization. Recall that the runtime complexity of the flow-based solver presented in Chapter 3 is quadratic in the number of nodes. Assume that nodes were uniformly distributed over the frames. If we were able to split the video into $N$ windows, we could hence reduce the complexity of every window by $N^2$, giving us a total simplification by a factor of $N$. In this section we present a combination of a graph contraction idea from [Beier et al., 2015] and the sliding window tracking idea in [Lenz et al., 2015] to a heuristic tracking method that analyses

[1]Using the request framework *lazyflow*: `https://github.com/ilastik/lazyflow`.

time windows in parallel and then combines local tracks to a global solution in a sparse graph covering the full video.

### 4.1.1 Related Work

Tracking methods consider different amounts of temporal context for linking. Hungarian matching [Kuhn, 1955] does that frame by frame only. State space (Kalman filtering) methods [Li et al., 2010, Amat et al., 2014] keep a state of every object representing its *history*, but cannot change prior assignments. Sliding window trackers (*e.g*. [Lenz et al., 2015]) perform tracking in batches of multiple time frames, but still keep the solution to everything that happened before the current window fixed. [Lenz et al., 2015] presented multiple ways to perform tracking using successive shortest paths, we lean on their memory bounded sliding window tracking approach in that we also contract tracks into single nodes and use the accumulated energy of the track for the node. We use the term *tracklet* for such a node that represents part of a track. [Castanon and Finn, 2011] grow tracklets to keep the size of the tracking problem manageable, but they have several limitations like no mergers and no false detections, and no focus on combining several windows. Contracted graphs were also used in [Beier et al., 2015]. There, subproblems yield different approximate solutions to the full graph, and are optimally fused into one globally consistent solution in a final step. By deferring only the decisions at locations where subproblem solutions disagree to the the final master problem, even the optimal fusion step remains tractable while considering global context. We combine these ideas by splitting the full video into subproblems, solving those time windows independently, then contract the simple tracks – without merge, split or division events – into tracklet nodes, and stitch tracklets from all subproblems together in a global master problem. This way we perform the easy decisions within the subproblems and defer more complicated assignments to the master problem where the solver can consider global context.

### 4.1.2 Algorithm

We here describe how to decompose a given tracking graph into $N$ windows which can be tracked in parallel, and how to combine their results to a consistent global solution, as outlined in Figure 4.1. To the best of our knowledge this is the first such approach that can cope with merge, split and division events. For didactic purposes we structure our description as if we were given a full graph at the beginning, but in practice one would also parallelize the graph creation over the windows.

Let us define $S$ as the $N+2$ temporally ordered split boundaries including first and last frame of the timespan to track. These splits could be equidistant, such that every window contains approximately the same number of nodes and edges, or such that the split position has low probabilities for merges, splits and divisions. The following steps can then be performed in parallel for each window:

1. Extract the subgraph for each window $T_i$ with $i \in \{0, 1, \dots, N+1\}$ such that $T_i$ ranges from $S_i$ to $S_{i+1}$, without overlap. We define the appearance and disappearance costs to zero at the intermediate split points because we do not want to penalize tracks beginning or ending there.

2. Track each window independently.

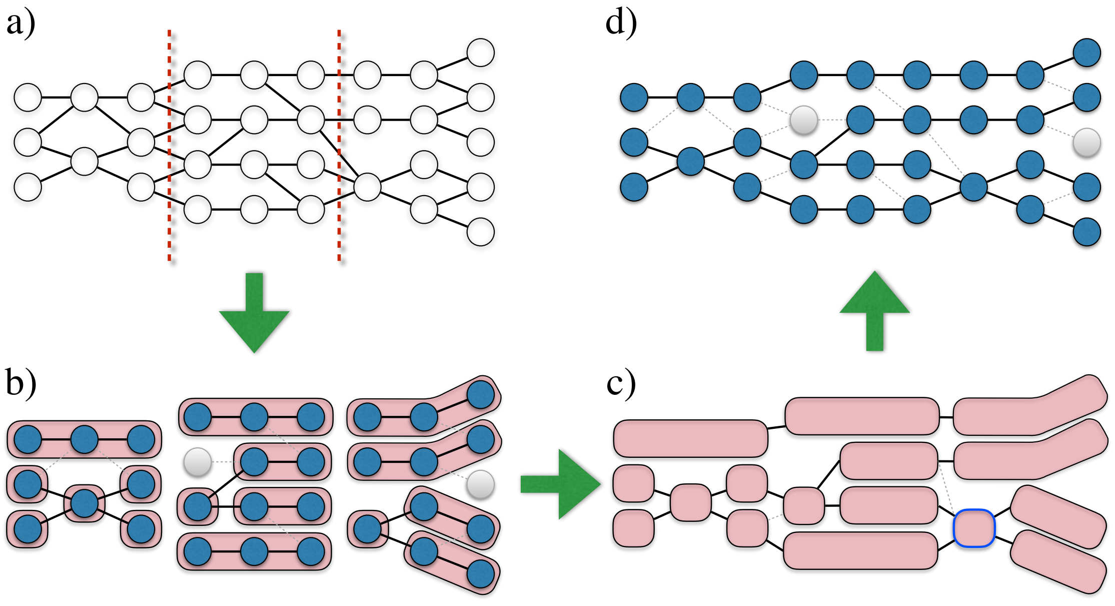


**Figure 4.1:** Illustrative example of graph decomposition and stitching. The full graph in **a)** gets decomposed at the two red lines. **b)** Each of the 3 subgraphs is solved individually. Unused arcs and nodes are indicated in light gray. Chains of linked nodes in the solution get contracted into *tracklet* nodes, shown as red elongated boxes. **c)** The master tracking problem links the contracted nodes from all subgraphs but also re-solves all links where different tracks coincide at clusters or divisions. Note that this way it does not matter whether the lower left node of the right subgraph (blue border in **c)**) had two outgoing links because of a merger or a division. The master problem will solve this linking again while having the global context available for consideration. **d)** The global decoded solution.

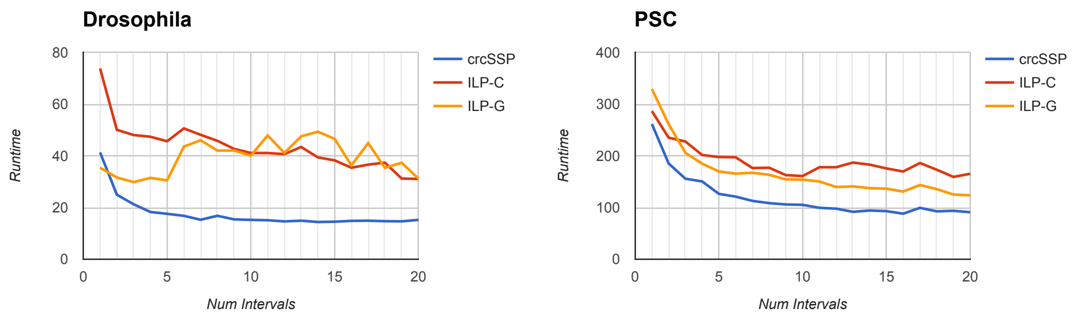


**Figure 4.2:** Accumulated tracking solver runtime of all intervals when splitting the dataset into windows of timeframes. The runtime for the final master problem is included. The different lines show the performance when solving the ILP formulation from Chapter 2 with CPLEX and Gurobi, and the flow-based `crcSSP` solver proposed in Chapter 3.

3. Prune all unused arcs from the subgraphs of all windows. Also remove arcs where there is more than one active incoming or outgoing arc incident at a node. This includes arcs of merging or splitting clusters and those that connect the two children of a division to their mother cell. Then only *simple* tracks remain.

4. Find connected components and contract each component into a tracklet node. Sum the contracted costs into the unary of the new node. See the red elongated boxes in Figure 4.1.

5. Re-insert the used arcs that connect the tracklets in the solution. This creates a much sparser graph representing the same solution.

Finally we concatenate these sparse graphs, adding links at the split points using the same criteria as when setting up a graph normally. This provides a suitable number of possibilities for combining the solutions of the separate windows, but also allows tracks to begin or end at the boundary between windows. We then solve this sparse graph covering the full video time span.

### 4.1.3 Results and Discussion

We evaluate the performance of the proposed decomposition scheme on the same Drosophila and PSC datasets as in Chapters 2 and 3, and run experiments on the big workstation[2]. However, in contrast to the models in the previous chapters, here we do not replace singly-linked chains of detections by one node, which explains the runtime deviations to those in Section 2.4 when tracking only a single interval spanning the whole video. In all experiments, we initialize by equidistant splits but shift the split points by a few frames if we find a pair of adjacent frames with a lower probability for clusters and divisions, hence reducing the sources for discrepancies of the solutions that might occur at the split.

By splitting the videos of the Drosophila and PSC dataset into $N \in 1 \ldots 20$ intervals of a similar number of timeframes, we can evaluate how the runtime and quality change with respect to tracking the full video at once. The scaling behavior of the flow-based `crcSSP` solver is as expected, but the ILP solvers apparently have difficulties finding the optimum for small subgraphs in the complicated Drosophila dataset, as shown in Figure 4.2. In Figure 4.3 we

[2]See Sections 2.4 and 3.5 for details

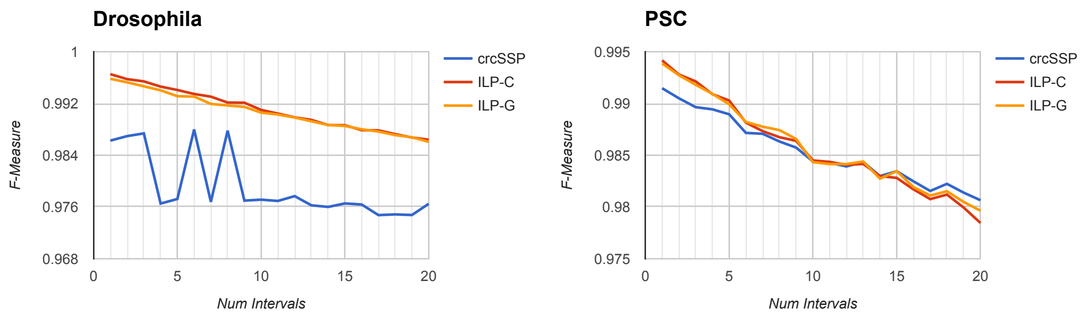


**Figure 4.3:** Quality of the solution after decomposing and stitching compared to the original ILP solution in terms of the f-measure of detection, transition and division agreement.

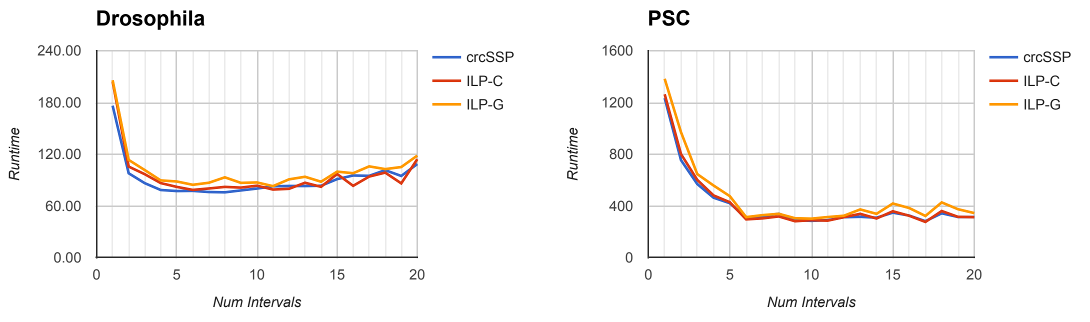


**Figure 4.4:** Wall clock runtime including input loading, graph decomposition, tracking, tracklet contraction and stitching. For the given problem sizes, using too many cores incurs too much decomposition and stitching overhead.

evaluate the quality of the resulting solution after splitting, tracking, contracting, and stitching. While the flow-based solver's runtime scales much better to small splits, it also yields worse results than the ILP solvers for Drosophila. On the PSC dataset all solvers behave similarly, but `crcSSP` has an edge in terms of runtime. The quality for $1$ interval is not always $100\%$ because we solve tracking twice there, once fully, then we build the tracklets, and then these tracklets get linked in a global stitching problem again. For the flow-based solver this represents the deviation from the optimal solution, while for the ILP solvers the minimal difference could be due to the allowed relative optimality gap[3] of $0.05$ for DRO and $0.01$ for PCS, or because the optimum is not unique.

Note that in practice the actual time to finish tracking decreases even further with the number of windows because these can be processed in parallel[4]. In the evaluation above we neglected the time required for graph decomposition and tracklet contraction to highlight the speedup that we gain solely by handing smaller graphs to the solvers. Figure 4.4 shows the total wall clock time of running our unoptimized Python implementation of loading the problem, decomposing the graph, tracking and contracting the results in parallel, and solving the master problem. The decomposition and contraction overhead could probably be made negligible with a C++ implementation. However, the performance gained by splitting and parallelizing the problem is visible much stronger than in Figure 4.2.

As seen in the results, this decomposition scheme exhibits an attractive trade-off between performance and quality. However, allowing for parallelization by tracking time windows independently and only stitching results together in a final *master* tracking problem does not provide all needed context to the solver in the subproblems, and especially for `crcSSP` no flow redirections can occur across the boundaries of the time windows. Another limitation is that if merges, splits and divisions are abundant in the subproblem solutions, the contraction does not reduce the size of the problem sufficiently. In the worst case the master problem remains as hard as the original problem and the decomposition into subproblems actually increases the overall complexity of the problem.

## 4.2 ilastik Microservices Architecture

In the previous Chapters we have simply assumed that we were given a segmentation, as well as the probabilities for every segment to be a cluster or part of a division. Obtaining this information can be a rather complex process, and thus we employed *ilastik* to perform these steps. However, in practice, these often account for even more processing time as the tracking step itself. Fortunately, most of these preprocessing tasks offer some trivial parallelization. *ilastik* already uses a request framework to parallelize the workload – which is decomposed into separate operators with input and output slots in a data flow graph – across multiple CPUs. But with the growing availability of clusters and cloud computing, more and more users are interested in using *ilastik*'s features in a distributed environment, or as a service within a bigger image processing platform. Also, developers of other bio-image processing tools like ImageJ [Schindelin et al., 2012] and Cell Profiler [Carpenter et al., 2006] expressed their interest in connecting to *ilastik* by means of interprocess communication instead of linking the tools through file export and import. *ilastik* already has a modular approach, the source code is split into a viewer, a

[3]See Section 2.4 for details.

[4]On a different tracking dataset of 20 flies over 30.000 frames, with many mergers but no divisions, we have seen a speed increase from 17 hours to 2.5 hours of flow-based solver runtime while achieving 99 percent accuracy of using 100 splits with respect to tracking the full graph.

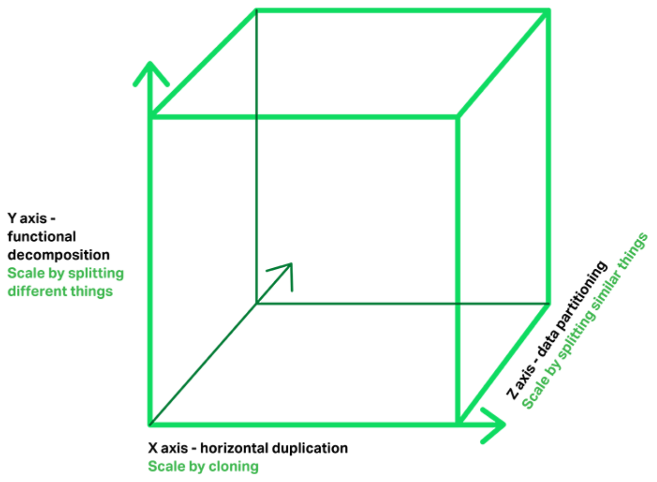

Y axis - functional decomposition
Scale by splitting different things
Z axis - data partitioning
Scale by splitting similar things
X axis - horizontal duplication
Scale by cloning

**Figure 4.5:** Scale cube taken from [Richardson, 2015]. The concept of microservices firstly focuses on functional decomposition (Y axis), but once it is implemented and appropriate methods for communication are employed, running multiple instances of the same service to distribute workload processing (X axis) becomes trivial. Data partitioning (Z axis) can naturally be done in spatial and temporal domain in videos, which *ilastik* is doing already.

request framework to parallelize the workload, and the GUI front end with different workflows that glue the building blocks together. Yet, these modules have been used in conjunction almost exclusively, which led to an entanglement by accessing members deep inside other modules because no clear module API was enforced. Also, the current architecture was never meant to distribute tasks across multiple machines, which is even harder to achieve now that it has grown into a monolith. Hence in this section we propose an architecture prototype that allows *ilastik* to scale to multiple machines.

*Microservices* are a recent trend [Villamizar et al., 2015, Richardson, 2015, Daya et al., 2016] in software design that focuses on decomposing functionality into small services that perform one task as good as possible. An application is then composed of several microservices which communicate in an appropriate way as specified by the developer. One might argue that this is a special case or a simplification of the *service oriented architecture* (SOA) pattern that concentrates more on providing a framework where services are not closely tied together. To ease the communication with other tools or behave like a monolith when deployed as standalone application, a microservices-based application commonly provides a gateway that distributes – and possibly load balances – requests to the responsible services.

The microservices decomposition pattern can solve many of the aforementioned problems: services make their interface explicit in terms of messaging protocols or REST APIs which can also be used for interoperability with other tools; scaling to multiple machines in a cluster or the cloud becomes natural, developers need not navigate the whole project to incorporate a change but usually touch only a small number of services, and services can even be developed in different programming languages if this is required. Microservices would allow *ilastik* to scale along the X axis of the scale cube (Figure 4.5) and make the functional decomposition (in Y) more explicit, while it is handling the data partitioning along the Z axis of the cube quite well already.

Unfortunately this comes at a price. On the one hand developers need to learn how to provide and connect to APIs using interprocess communication channels. To deploy or test, instead of shipping and running one monolithic application, now a group of services needs to be configured and started. And for memory intensive tasks such as image processing, transferring large amounts of data between different machines can come up as a bottleneck. However, minimizing the amount of data sent across the network, which is often approached by considering data locality in a load balancing scheduler, is a research topic on its own that we do not want to address here.

### 4.2.1 Design Prototype

In order to distribute the processing steps of *ilastik* across multiple services, several changes need to be made to its design. We will here present and evaluate a microservices prototype for classifying pixels and finding connected components of foreground pixels after thresholding the probabilities, which gives us a segmentation of the input image. Many of the concepts presented here can also be applied to the tracking workflow, as we discuss below.

The main processing steps involved in *ilastik*'s segmentation method are **(a)** computing features, **(b)** predicting the probability for each pixel – here with a Random Forest [Breiman, 2001] – **(c)** smoothing and thresholding the probability map, and **(d)** extracting connected components of foreground pixels. Not all of these operations can be parallelized equally well. To decompose the work as good as possible while respecting data locality we group the steps into a *pixel classification* service performing **(a)** and **(b)**, and a *thresholding* service with steps **(c)** and **(d)**, as shown in Figure 4.6(a). The microservices architecture and gateway we build here are tailored to perform pixel classification and thresholding only, for tracking one would set up an independent set of microservices and connect them appropriately, *e.g.* as in Figure 4.6(b).

The *pixel classification* service parallelizes its work over blocks in a predefined grid over the full video. For every block, the raw data – including a little margin around the block extents – is loaded from the *data provider* service. The *data provider* service is the processing pipeline's point of access to the data source, which could be a file or some kind of volumetric database (*e.g.* dvid[5]), and would generally provided by the user. From the raw data blocks, the *pixel classification* service computes features[6] **(a)**. The margin is required because the features are computed on different scales by smoothing the image. To compute the value of a smoothing filter at the boundary of some region, the pixel values beyond that boundary need to be considered to yield correct results. Lastly, a previously trained Random Forest then classifies each pixel in the block based on its features. The service then returns the probabilities for each pixel to belong to the different classes.

The *thresholding* service on the other hand works on full frames instead of individual blocks, for two reasons; Firstly, **(c)** also includes a smoothing step of the probabilities, hence adjacent blocks need to be considered. Secondly and more importantly, connected components **(d)** can only be extracted properly when processing the full image, because objects reaching over block boundaries need to be merged at some point.

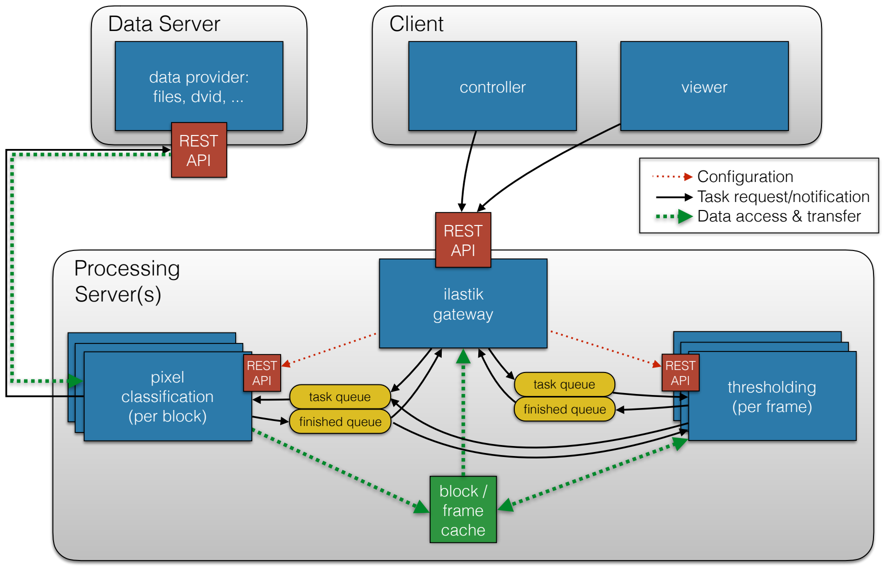
As seen in Figure 4.6(a), we additionally use a central cache that is filled with the results of all freshly computed per-block predictions and per-frame segmentations, and frees up space according to a *least recently used* (LRU) strategy. We use an in-memory instance of a *redis*[7] key-value storage for that[8]. There are multiple reasons why we use a cache to collect the processed

[5]`https://github.com/janelia-flyem/dvid`

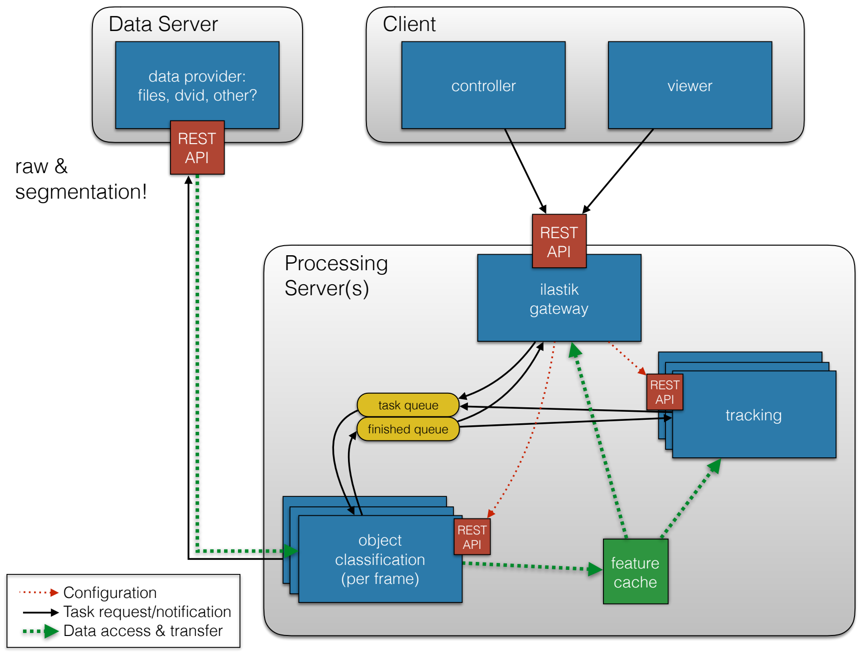
[6]We use Gaussian smoothing, gradient magnitude, Laplacian of Gaussian, structure tensor eigenvalues, and Hessian eigenvalues.

[7]`https://redis.io/`

[8]For production settings one could configure *redis* to be a distributed cache or even extend the available memory



(a) Pixel Classification and Thresholding



(b) Tracking

**Figure 4.6:** Possible decomposition of ilastik's processing steps into services. **(a)** presents the implemented design prototype for pixel classification and thresholding, and **(b)** shows a possible architecture that could be used for tracking when the segmentation is already present and available from the data provider service.

results. Firstly, users could be displaying the predictions and while scrolling through the data with some viewer parallel or sequential requests for predictions could be spatially overlapping.

by saving cached blocks to disk.

Then caching helps for much faster response times. Secondly, the parameters used for thresholding could be changed by a controller or the user. Then cached predictions can be reused. Lastly, *ilastik*'s most famous use case is to train classifiers interactively. To implement that the cache additionally holds the computed features, and cached predictions would be invalidated whenever the classifier is updated.

The crucial change when decomposing a monolith into microservices is how internal and external communication are managed. We chose to make all external access, as well as internal configuration calls synchronous via a REST API. The *data provider* is also accessed via REST. But for the internal distribution of tasks, we employ an asynchronous communication model using task queues and a publish/subscribe pattern for notification about finished tasks, which is sent as soon as results are stored in the central cache. A detailed sequence diagram of how an external request to the gateway fills the pixel classification task queue and collects the probabilities is depicted in Figure 4.7. Because the gateway could receive requests that require the same blocks, but which are not processed yet, we design the communication such that no block is processed twice. Every incoming request first determines which blocks are required. Then it starts a *block collector* thread that listens on the *"block finished"* queue for matching blocks. Only then it checks which required blocks are available in the cache and holds on to them. All other blocks are enqueued in the task queue. To prevent the same block to be enqueued multiple times, the first request to schedule a block for processing inserts a placeholder block into the cache that is taken by other requests as indicator that this block is already enqueued. Only as soon as the *block collector* thread has found all required blocks is the request served to the client. This very scheme is also employed when the *thresholding* service is asked to process a frame, which then requests and waits for all blocks that belong to the frame. With these steps it is trivial to process multiple tasks in parallel by an unlimited number of workers. For thresholding and object classification requests one can use similar task queue communication, but then tasks represent full frames, not blocks.

Another important part of the configuration of microservices is *service discovery*. Every available machine could – depending on its size – run multiple services at once. Still these services need to talk to each other and hence need to know their respective IP addresses and ports. Professional solutions to this problem are *e.g. etcd*[9] or *Consul*[10], but for our prototype we only employ a key-value storage that serves as central registry that every service knows and where it registers its own IP (not shown in Figure 4.6). We also use this *redis* key-value registry server's `list` datatype and publisher/subscriber pattern to implement all message queues.

### 4.2.2 Experiments

To test the proposed architecture, we process a $384 \times 384 \times 192$ volume using $64 \times 64 \times 64$ blocks (108 in total) and evaluate the scaling behavior with the number of machines when using 30 features per pixel. Figure 4.8 shows the total runtime from requesting the thresholding output for the full image until the result is transferred. The pure processing time is very well distributed over the machines, but some overhead remains. As this is only one volume without multiple time steps, thresholding behaves like a global synchronization step. We chose this setup to limit the scaling effects to only one level, because as soon as multiple time frames are requested at the same time, blocks from different frames are competing for pixel classification workers, making it harder to evaluate.

[9] https://github.com/coreos/etcd
[10] https://www.consul.io/

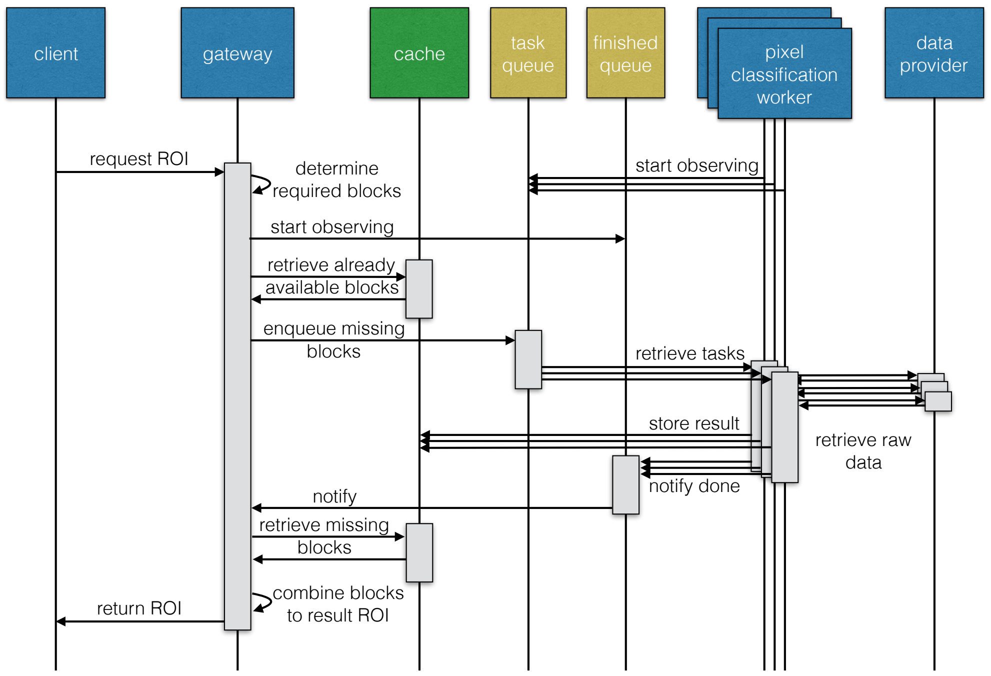


**Figure 4.7:** Sequence diagram for the communication between gateway and pixel classification service such that blocks of data can be processed in parallel by multiple workers that are all subscribed tn the same task queue.

The resulting scaling behavior is nearly perfect, which is even more impressive considering that the total runtime includes the single-threaded thresholding step. This means that our architecture is very well suited to distribute the workload across multiple machines, and that the overhead incurred by data transfer is not affecting the performance much, but even more importantly is nearly constant no matter how many machines are used.

All experiments were run on *Amazon Web Services*'s Elastic Compute Cloud (AWS EC2), using free tier *t2.micro* instances with 1 virtual CPU and 1GB RAM, only thresholding was run on a machine with 8GB RAM because even just storing all predictions of the volume requires 2.6GB of memory – which also means this amount of data transfer must have happened between *pixel classification* workers, cache, and *thresholding* service. The *data provider* service was also run on AWS, because data transfers within their network are faster and incur less charges than outbound traffic.

In terms of pure processing runtime, *ilastik* performs a lot better ($\approx 180s$ without thresholding also on a *t2.micro* AWS EC2 machine). But a lot of this speed difference can be allotted to a *presmoothing* step which reduces the feature computation runtime roughly by a factor of two to three. If this was also implemented in our pixel classification service prototype, the runtimes would be even better. Another reason could be that our choice of block size incurs much more redundant feature computations in the margins than that of *ilastik* ($114^3$). Nevertheless, this has no effect on the near perfect scaling behavior of our architecture with the number of machines.

### 4.2.3 Outlook

Our experiments have shown that employing a microservices architecture for segmentation is not only feasible, but scales extremely well. Once this prototype has matured and the API becomes

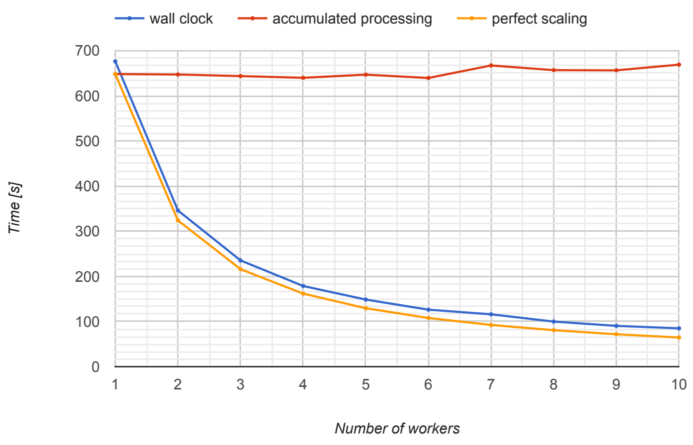


**Figure 4.8:** Scaling of our microservices architecture for pixel classification and thresholding with the number of machines. The *wall clock* time represents the time it takes to deliver the result to the user, including all data transfers. The *accumulated processing* time was required to perform the actual feature computation and prediction, with little fluctuations due to non-exclusive CPU access on AWS. For $1$ machine we see that there is around $25s$ overhead, of which $\approx 8s$ account for thresholding and extracting connected components, which is not parallelized. If we divide the total observed pixel classification processing time for $1$ worker by the number of workers we would obtain runtimes as shown by the *perfect scaling* line. Comparing this to the observed *wall clock* times shows that the overhead of thresholding and data transfer remains constant, which means our task decomposition and messaging schemes do not suffer from competing access to resources.

stable, we can also use these interfaces to communicate with other bio-image analysis tools directly.

If we wanted to build a similar architecture for tracking, as sketched in Figure 4.6(b), then we would want to parallelize the object feature extraction and classification over multiple machines, as this can be done independently per frame. Employing the graph decomposition from Section 4.1 for tracking, multiple tracking workers could process segments of the video, while global synchronization is only required as the very last step when stitching the subgraph solutions to one final tracking result.

However, the most prominent use case for *ilastik* is interactive pixel classification, providing immediate visual feedback to the user after every change to the training annotations for the classifier. To support that, we would have to cache blocks of features instead of just predictions, because the features remain valid even when the classifier is changed. The predictions need to be recomputed whenever a new classifier has been trained. Hence, a training method must be added to the *pixel classification* service that takes a set of annotations as input and invalidates all cached predictions on completion. It remains to see whether the data transfer times stay acceptable in that use case, as interactivity is key. Using a smaller block size could help to increase responsiveness.

Independent of the chosen architecture, there are several things to be aware of when splitting *ilastik* into multiple services. The most severe downside is that debugging failures becomes much more complicated as several services and their communication need to be monitored. Additionally the system must be tolerant to failures and crashes of individual services and machines. To find problems early in the development cycle, *continuous integration* is indispensable. We could for instance let Docker[11] containers be built and tested automatically[12] on every commit. A whole profession called *DevOps* has evolved around that and provides many tools to support developers. Another problem is that if we want to use the same microservices architecture running on Docker but with a GUI front end deployed as end-user software on a single machine, we should be aware that *Docker for Mac* and *Docker for Windows* provide only around 70% of the performance of the host machine due to the virtualized Linux in which the containers run.

[11] https://www.docker.com/
[12] https://circleci.com/integrations/docker/

Chapter 5

# Obtaining Diverse-$M$-Best Hypotheses Through Dynamic Programming

Many computer vision pipelines involve dynamic programming primitives such as finding a shortest path or the minimum energy solution in a tree-shaped probabilistic graphical model. In such cases, extracting not merely the best, but the set of $M$-best solutions is useful to generate a rich collection of candidate proposals that can be used in downstream processing. In this chapter, we show how $M$-best solutions of tree-shaped graphical models can be obtained by dynamic programming on a special graph with $M$ layers. The proposed multi-layer concept is optimal for searching $M$-best solutions, and so flexible that it can also approximate $M$-best diverse solutions. We illustrate the usefulness with applications to object detection, panorama stitching and depth estimation[1]. In Chapter 6 we use this approach to obtain dectection hypotheses for tracking tuberculosis bacteria.

## 5.1 Introduction

A large number of problems in image analysis and computer vision involve the search for the *shortest path* (*e.g.*, finding seams and contours) or for the *maximum-a-posteriori* (MAP) configuration in a tree structured graphical model, as in hierarchies of segmentation hypotheses or deformable part models. To compute the solution to those problems, one relies on efficient and optimal methods from dynamic programming [Bellman, 1952] such as Dijkstra's algorithm [Dijkstra, 1959]. In many of these scenarios, it is of interest to find not merely the single lowest energy (*i.e.*, MAP) solution, but the $M$ solutions of lowest energy (*$M$-best*) [Lawler, 1972, Seroussi and Golmard, 1994, Nilsson, 1998, Rollon et al., 2011, Batra, 2012]. This can *e.g.* be useful for learning [Lampert, 2011], tracking-by-detection methods that allow competing hypotheses [Milan et al., 2013, Jug et al., 2014, Schiegg et al., 2014], or for re-ranking [Yadollahpour et al., 2013] solutions based on higher order features which would be prohibitively complex for the original optimization problem. If these solutions are to differ in more than one label, the problem is referred to as *diverse $M$-best* [Batra et al., 2012, Kirillov et al., 2015, Prasad et al., 2014].

[1]The content of this chapter is under review at the German Conference for Pattern Recognition (GCPR) 2017.

**Contributions:** In this work, we show how the optimal second best ($M = 2$) solution of a tree-shaped graphical model can be found through dynamic programming in a multi-layer graph by using a replica of the original graph as second layer and connecting both layers through edges with special jump potentials (Section 5.3). Using these building blocks, we extend our approach to exactly find the $M > 2$ best solutions sequentially by constructing $M$-layer graphs (Section 5.4). While the above can be seen as a special case of [Yanover and Weiss, 2004], our multi-layer approach is an intuitive interpretation that allows flexible modeling of the desired result. We thus develop two heuristics using multiple layers to find the approximate diverse $M$-best solutions for tree-shaped graphical models (Section 5.5). Lastly, we experimentally compare the different diversity approximations to prior work, and show results for a variety of applications, namely: *i)* panorama stitching, *ii)* nested segmentation hypotheses selection, and *iii)* stereo depth estimation (Section 5.6).

## 5.2 Related Work

**M-best MAP:** An algorithm for sequentially finding the $M$ most probable configurations of general combinatorial problems was first presented in [Lawler, 1972]. To find the next best solution, they solve as many optimization problems as there are variables in the model. This is because they branch on the state of every single variable, resolve, and finally choose the best of all resulting configurations. While this works for any optimization method and model, it is in practice prohibitively expensive. Several works have extended this to junction trees [Seroussi and Golmard, 1994, Nilsson, 1998] that work in $O(|\mathcal{V}|(L^2+M+M\log(|\mathcal{V}|M)))$, while [Rollon et al., 2011, Flerova et al., 2012] developed a similar bucket elimination scheme ($O(M|\mathcal{V}|L^{|\mathcal{V}|})$). All of these methods consider the $M$ best configurations for every clique or bucket, which are then combined to jointly yield $M$ consistent global solutions. A similar idea was applied in [Eppstein, 1998] to find the $M$ shortest paths jointly by building an auxiliary graph with a heap at every node that contains the $M$-best paths to reach that node. For situations where the optimal or approximative max-marginals can be computed, [Yanover and Weiss, 2004] derived an improvement on [Nilsson, 1998] such that the max-marginals have to be computed only $2M$ times, yielding the same runtime complexity ($O(M|\mathcal{V}|L^2)$) as the method we present here. A method that finds the $M$ best solutions on trees in only $O(L^2V + \log(L)|\mathcal{V}|(M-1))$ by an algebraic formulation that is similar to sending messages containing $M$ best values as in [Flerova et al., 2012] was presented by [Schlesinger and Hlavác, 2013, Chapter 8]. However, in contrast to [Yanover and Weiss, 2004, Schlesinger and Hlavác, 2013], our approach provides a lot of modeling flexibility, allowing it to be used to approximate diverse $M$ best solutions as well.

A polyhedral optimization view of the sequential $M$-best MAP problem is given in [Fromer and Globerson, 2009]. There, a linear programming (LP) relaxation of the $M$-best MAP problem is constructed by characterizing the assignment-excluding local polytope through spanning-tree-inequalities, which separate the previous best solutions from the marginal polytope such that all its vertices remain integral. This LP relaxation is tight for trees for $M = 2$, but not for higher $M$ or loopy graphs as the assignment-excluding inequalities could together cut away other integral vertices of the polytope. An efficient message passing algorithm for the same LP relaxation was designed by [Batra, 2012] by exploiting the structure of the polytope.

**Diverse $M$-Best:** For general graphical models, the first formulation of the *diverse $M$-best* problem of successively finding MAP solutions that obey a diversity constraint with respect to all previous solutions can be found in [Batra et al., 2012]. Even though their Lagrangian relaxation of the diversity constraint can work with any choice of metric, it is not even tight for Hamming distances of $k > 1$ between solutions. Different diversity metrics are explored in [Prasad et al., 2014], where a greedy method to find good instances from the (exponentially big) set of possible solutions is designed by setting up a factor graph with higher order potentials, assuming that the diversity metric is submodular. In [Kirillov et al., 2015], the authors construct a factor graph that *jointly* finds the diverse $M$-best solutions by replicating the original model $M$ times and inserting factors for the diversity penalty depending on the structure of the chosen distance. It is shown that maximizing the diversity of the $M$-best solutions jointly, not only sequentially as in [Batra et al., 2012], can yield better results. They propose a reformulation that preserves solvability with $\alpha$-$\beta$-*swap*-like methods. Still, when applied to trees, the factors introduced for diversity unfortunately turn the problem into a loopy graph and prevent the application of dynamic programming. While the aforementioned approaches allow to insert the desired diversity metric (*e.g.*, one of those presented by [Prasad et al., 2014]) into the original optimization problem by Lagrangian relaxation, our model includes the diversity constraints in the graph construction. Hence, we do not need to optimize additional parameters such as the Lagrangian multiplier via subgradient ascent. Instead, we obtain the next best solution in a single shot. Yet, similar to the methods above, our model can only handle diversity measures that decompose over nodes and edges in a graphical model.

On the other hand, [Chen et al., 2013] and [Chen et al., 2014] developed a scheme to find the $M$-best *modes* in chain and tree-shaped graphical models, respectively. For the latter, they build a junction tree of the same structure as the original graph, where auxiliary nodes represent the subgraph contained in a Hamming ball around the respective node of the original graph. For a given scale of this Hamming ball, they find the $M$-best solutions of the junction tree using [Nilsson, 1998], yielding the optimal $M$ modes. While this approach can provide detailed insights into the probability distribution, its computational complexity renders it intractable for large graphs.

## 5.3 Optimal Second-Best Tree Solutions

We now present how the second best solution of a tree-shaped graphical model can be obtained using dynamic programming on a special graph construction. By second best, we mean a solution that differs from the best configuration in at least one node, *i.e.*, that has a Hamming distance of $k \geq 1$ to the best solution. We begin with an informal motivation based on the search for the second best shortest path.

**Motivation: Second Shortest Paths:** The Dual Dijkstra method from [Fujita et al., 2003] allows finding not only the best, but a collection of $M$ shortest paths from a source $\sigma$ to a target $\tau$ in a graph. To do so, two shortest path trees are constructed, one starting at the source and one at the target. Thus, for every node $v$, the shortest path from the source $\mathcal{P}_{\sigma,v}$ and to the target $\mathcal{P}_{v,\tau}$ is known. Summing the distances to source and target gives the length of the shortest path from $\sigma$ to $\tau$ via $v$. An important property is that, for all vertices along the shortest path from $\sigma$ to $\tau$, this sum is equal to the length of the shortest path.

Now imagine these shortest path trees as two copies of the initial graph stacked as two layers, as seen in Figure 5.1 *(a)*. The lower layer indicates the lowest cost to reach every vertex $v$ from $\sigma$, and the upper layer the cost of the shortest path to reach $\tau$ from $v$. By selecting any vertex $v$ and connecting the paths at $v$ in the lower and upper layer, one can again find the shortest path from $\sigma$ to $\tau$ via $v$, this time by introducing an auxiliary *jump* edge between the two layers.

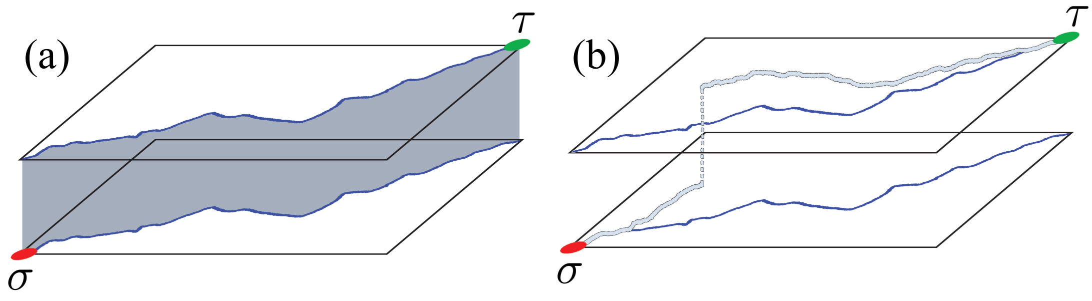


**Figure 5.1:** A schematic two-layer grid-graph construction to find the second best shortest path from the source $\sigma$ in the lower, to the target $\tau$ in the upper layer. A valid path is hence required to jump between layers, which is allowed everywhere for the best path. **(a)** Shown in blue is the best solution, which could have jumped to the upper layer at every node along the path with the same cost. **(b)** To find the second best path, layer jumps are forbidden at the nodes used by the best solution. Thus the second path diverges to the jump location leading to the next minimal cost path.

The benefit of this two layer setup is that to find the *second* best solution, we simply need to search for the vertex that *does not* lie on the best path, at which jumping between the layers leads to the minimal cost path.

**Dynamic Programming:** Let us briefly review the dynamic programming (DP) paradigm on an undirected tree-shaped graph $\bar{G} = (\mathcal{V}, \bar{\mathcal{E}})$. We denote the state of a node $v \in \mathcal{V}$ as $\mathbf{x}_v$, and the full state vector as $\mathbf{x} = \{\mathbf{x}_v : v \in \mathcal{V}\}$. The potentials of node $v$ (*unary* potential) and of the edge connecting nodes $u$ and $v$ (*pairwise* potential) are represented by $\theta_v(\mathbf{x}_v)$ and $\theta_{uv}(\mathbf{x}_u, \mathbf{x}_v)$, respectively. From this, we define the inference problem as an energy minimization task [Koller and Friedman, 2009] with objective

$$\min_{\mathbf{x}} \sum_{v \in \mathcal{V}} \theta_v(\mathbf{x}_v) + \sum_{(u,v) \in \bar{\mathcal{E}}} \theta_{uv}(\mathbf{x}_u, \mathbf{x}_v). \tag{5.1}$$

When applying dynamic programming, one successively computes the energy $E$ of optimal solutions of subproblems of increasing size. One node of the graph $\bar{G}$ is arbitrarily selected as the root node $r$. This results in a directed graph $G = (\mathcal{V}, \mathcal{E})$ where edges point towards the root. Let $\overleftarrow{N}(v)$ denote the neighboring nodes along incoming edges of $v$ in $G$. Using the tree-imposed ordering of edges, one starts processing at the leaves and sends messages embodying the respective subproblem solutions towards the root. Whenever a node $v$ has received a message from all incoming edges, it can – disregarding its successors in $G$ – compute the lowest energies $E_v(\mathbf{x}_v)$ of the subtree rooted at $v$ for every state $\mathbf{x}_v$, and send a message to its parent [Pearl, 1988]. Because leaves have no incoming edges, their energy is equal to their unary potentials. All subsequent nodes combine the incoming messages with their unary potentials to obtain the energy of the subtree rooted at them by

$$E_v(\mathbf{x}_v) := \theta_v(\mathbf{x}_v) + \sum_{u \in \overleftarrow{N}(v)} \min_{\mathbf{x}_u} \left[\theta_{uv}(\mathbf{x}_u, \mathbf{x}_v) + E_u(\mathbf{x}_u)\right]. \tag{5.2}$$

While sending these messages, each node $v$ stores which state $(\operatorname{argmin}_{\mathbf{x}_u})$ of the previous node $(u \in \overleftarrow{N}(v), v)$ along each incoming edge led to the minimal energy of every state $\mathbf{x}_v$. When the root has been processed, the state that led to the minimal energy is selected and, by backtracking all the recorded $\arg\min$, the best global configuration $\mathbf{x}^\star$ can be found. Figure 5.2 shows a minimal tree example.

Regarding DP runtime complexity, consider that (5.2) needs to be evaluated for every state of every node exactly once. In addition, in (5.2), we consider all states of every incoming edge, of which there are $|\mathcal{E}| = |\mathcal{V}| - 1$ in a tree. If $L$ denotes the maximum number of states, one obtains $O(|\mathcal{V}|L^2)$.

**Two-layer Model:** Once the optimal solution is found, we might be interested in the second best solution $\mathbf{x}$, which assigns a different state to at least one node $\exists v \in \mathcal{V} : \mathbf{x}_v \neq \mathbf{x}^\star_v$. Because messages in DP only convey the optimal subtree energies, we cannot immediately extract this second best solution. Hence we are looking for a way to enforce that a different state is attained at least once, but we do not know at which node(s) this should happen to yield the optimal energy. Fortunately, we can apply the same idea as in the second shortest path example: We duplicate the graph to get a second layer and insert edges connecting the two layers such that jumping is only permitted at states not used in the optimal solution $\mathbf{x}^\star$. After propagating messages through both layers, the second best solution can be obtained by backtracking from the minimum energy state of the root in the second layer to leaves in the first layer. This means that messages must have jumped to the second layer at least once at some node $v$ with a state different to $\mathbf{x}^\star_v$, fulfilling our requirement for the second best solution.

To create the two layers, we duplicate graph $G$ (Figure 5.2**a**) such that we get a layer 1, and a layer 2 replica. We address the instances of every node $v \in \mathcal{V}$ by $v^1$ and $v^2$ for layer 1 and layer 2, respectively. When duplicating the graph, the unary and pairwise potentials of nodes and edges are copied to layer 2. At every node $v \in \mathcal{V}$, we insert a *layer-jump-edge* from $v^1$ to $v^2$ (blue edges in Figure 5.2**c**) with a pairwise potential $\theta_{v^1v^2}$ that is only zero if both variables take the same state $\mathbf{x}_{v^1} = \mathbf{x}_{v^2}$ different from $v$'s state in $\mathbf{x}^\star$, and infinity (forbidden) otherwise. This way, finite valued messages in layer two represent configurations that did differ from $\mathbf{x}^\star$ at least once. These jump edges would suffice for a chain graph, but the branching points in a tree need special consideration. When a layer 2 branching point is not reached by a layer jump, the current construction only allows considering incoming messages from layer 2. However, since we only require *one* variable to take a new state, only one branch is necessary to reach layer 2 on a path with finite cost. To cope with this situation, we insert *layer-crossing* edges from $u^1$ to $v^2$ for all edges $(u, v) \in \mathcal{E}$ (dashed purple edges in Figure 5.2**c**) with the same pairwise potential as in the original graph $\theta_{u^1v^2} = \theta_{uv}$.

More formally, we construct the auxiliary directed graph $\tilde{G} = (\tilde{\mathcal{V}}, \tilde{\mathcal{E}})$ composed of two instances of the original directed graph $G = (\mathcal{V}, \mathcal{E})$ stacked up vertically. We address the lower layer with index 1, and the upper one with index 2. The new set of nodes and edges is given by

$$\begin{aligned} \tilde{\mathcal{V}} &:= \{1, 2\} \times \mathcal{V}, \\ \tilde{\mathcal{E}} &:= \mathcal{E}^1_{\text{in}} \cup \mathcal{E}^2_{\text{in}} \cup \mathcal{E}_{\text{jump}} \cup \mathcal{E}_{\text{cross}}, \end{aligned}$$

where

$$\mathcal{E}^1_{\text{in}} \quad := \quad \{(u^1, v^1) | (u, v) \in \mathcal{E}\} \tag{5.3}$$

$$\mathcal{E}^2_{\text{in}} \quad := \quad \{(u^2, v^2) | (u, v) \in \mathcal{E}\} \tag{5.4}$$

$$\mathcal{E}_{\text{jump}} \quad := \quad \{(v^1, v^2) | v \in \mathcal{V}\} \tag{5.5}$$

$$\mathcal{E}_{\text{cross}} \quad := \quad \{(u^1, v^2) | (u, v) \in \mathcal{E}\}. \tag{5.6}$$

Thus, edges are duplicated for each layer (5.3), (5.4), and we introduce *layer-jump-edges* (5.5) that directly go from any node $v^1$ in layer 1 to its duplicate $v^2$ in layer 2, as well as edge duplicates that originate in layer 1 and *cross* to layer 2 for message passing at branching points (5.6).

We can then alter the DP update equation for nodes in layer 2 to

$$E_{v^2}(\mathbf{x}_{v^2}) := \min \Bigg( \theta_{v^1 v^2}(\mathbf{x}_{v^1}, \mathbf{x}_{v^2}) + E_{v^1}(\mathbf{x}_{v^1}), \tag{5.7}$$

$$\theta_{v^2}(\mathbf{x}_{v^2}) + \min_{\substack{L_2 \subseteq \overleftarrow{N}(v) \\ |L_2| \geq 1}} \sum_{u \in L_2} \min_{\mathbf{x}_{u^2}} \left[ \theta_{u^2 v^2}(\mathbf{x}_{u^2}, \mathbf{x}_{v^2}) + E_{u^2}(\mathbf{x}_{u^2}) \right]$$

$$+ \sum_{u \in \overleftarrow{N}(v) \setminus L_2} \min_{\mathbf{x}_{u^1}} \left[ \theta_{u^1 v^2}(\mathbf{x}_{u^1}, \mathbf{x}_{v^2}) + E_{u^1}(\mathbf{x}_{u^1}) \right] \Bigg). \tag{5.8}$$

Compared to (5.2), we now have two options instead of one at every node $v$ in layer 2. Firstly, we can reach $v^2$ by a layer jump. Note that, in case of a jump (5.7), we do not account for the unary $\theta_{v^2}(\mathbf{x}_{v^2})$ as $E_{v^1}(\mathbf{x}_{v^1})$ contains the same term already. Alternatively, at least one of the incoming messages must come from a nonenpty set $L_2$ of predecessors in layer 2 (5.8), while the remaining messages could *cross* layers. These options are visualized in Figure 5.2**c**.

**Optimality and Runtime:** By duplicating the directed graph and inserting two sets of new edges which are oriented towards the root in layer two, the topology of the graph remains a directed acyclic graph, and DP hence yields the optimal configuration. As long as the solution has finite energy, no forbidden *layer-jump-edge* is used, giving us the second best solution. In terms of runtime complexity, we have duplicated the number of vertices and have four times as many edges, which are small constant factors that disappear in $O(|\mathcal{V}|L^2)$. For optimal performance, one can reuse the messages in layer 1 because these do not change.

## 5.4 Optimal $M$-Best Tree Solutions

The two-layer setup can easily be extended to multiple layers, which allows us to search for the $M$-best solutions with a Hamming distance of $k \geq 1$. We use one additional layer per previous best configuration; that is, $M$ layers. Each layer is responsible for one of the previous solutions, hence its *layer-jump-edges* are restricted according to the respective solution. Solutions must be ordered by increasing cost; such that the first layer constrains jumps with respect to the best configuration, the second layer for the second best, and so on. The new update rules from in Section 5.3 can then be applied to every consecutive pair of layers. Figure 5.2**d** shows an example.

**Optimality and Runtime:** When considering more than one previous solution using the multi-layer setup, the jump restrictions encoded in the *layer-jump-edge* potentials are independent at each layer. For any given node and state in a layer, the cost and path to reach it are optimal with respect to all layers below. This straightforwardly holds for the one and two layer cases, and is the reason why layers must be ordered by increasing cost of the represented previous solutions. For the sake of argument, layers could be flattened as they are getting processed, bringing back the problem to a series of $M-1$ optimal two-layer cases, which yields a computational complexity of $O(M|\mathcal{V}|L^2)$.

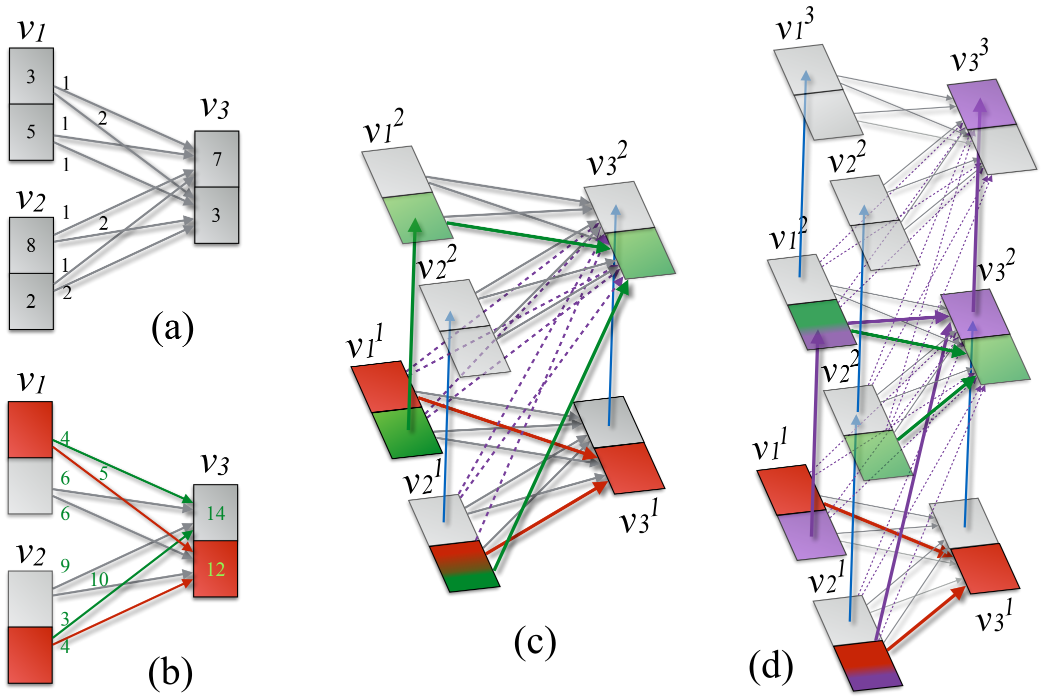


**Figure 5.2:** **(a)** Minimal tree structured graph of nodes $v_1$, $v_2$, and $v_3$, with two states each, visualized as stacked boxes. $v_3$ is arbitrarily designated as the root, or target. Unary and pairwise costs are shown as numbers in boxes and along edges, respectively. Its optimal solution is highlighted in red in **(b)**. Green edges correspond to the argmin incoming configurations for each state, and green numbers depict the accumulated min-sum messages. **(c)** Two-layer tree used to find the second best solution. Blue arcs represent layer-jump-edges with finite potential, which are available at states not occupied by the best solution. Purple dashed edges need to be considered if, at a branching point (such as $v_3^2$), not all incoming messages are coming from the upper layer. The second best solution is represented in green. **(d)** Searching for the $3^{rd}$-best solution (purple) with a Hamming distance of $k = 1$ to the best (red) and second best solution. The new solution must jump twice to reach the upper layer, by taking a state that was not used in the configuration represented by layers 1 and 2.

## 5.5 Approximate Diverse $M$-Best Solutions

In the classical diverse-$M$-best setting [Batra et al., 2012], additional solutions are required to have *e.g.* a Hamming distance of $k > 1$. Here, we look at the straightforward multi-layer extension of Section 5.3 to handle $k > 1$. We argue that this approach is suboptimal, and present a two-layer approximation that trades quality for efficiency. Lastly, we discuss how this could be used to find $M$ diverse solutions.

### 5.5.1 Multi-layer Model

To ensure that the next solution differs by at least $k$ from the best one, we can construct a $k+1$-layer graph using the same jumping criteria between all layers. To reach the top layer, a solution must hence jump $k$ times. This raises two challenges: *(a)* A solution should never jump more than once at a single node, otherwise it will not have the desired diversity. *(b)* A branching point at layer $N$ can be reached by a combination of edges from different layers such that the predecessors *in total* account for a Hamming distance of $N$. Both can be achieved by adjusting the DP update equation to consider a set of admissible incoming edge combinations. We thus generalize the update equation as follows:

$$\begin{aligned} E_{v^N}(\mathbf{x}_{v^N}) := \min \Bigg( & \tilde{\theta}_{v^{N-1}v^N}(\mathbf{x}_{v^{N-1}}, \mathbf{x}_{v^N}) + E_{v^{N-1}}(\mathbf{x}_{v^{N-1}}) \\ & + \infty \cdot \delta[\mathrm{Pred}_{v^{N-1}}(\mathbf{x}_{v^{N-1}}) == (v^{N-2}, \mathbf{x}_{v^{N-2}})], \\ & \tilde{\theta}_{v^N}(\mathbf{x}_{v^N}) + \min_{\mathcal{A} \in \mathscr{A}_{v^N}} \min_{\mathbf{x}_a} \sum_{a \in \mathcal{A}} \tilde{\theta}_{av^N}(\mathbf{x}_a, \mathbf{x}_{v^N}) + E_a(\mathbf{x}_a) \Bigg) \end{aligned} \tag{5.9}$$

$$\begin{aligned} \mathscr{A}_{n^N} = \Bigg\{ \left( u_1^{l_1}, u_2^{l_2}, \ldots, u_{|\overleftarrow{N}(v)|}^{l_{|\overleftarrow{N}(v)|}} \right) \Bigg| \, & u_i^{l_i} \in \overleftarrow{N}(v), \\ & l_i \in \{1, \ldots, N\}, \sum_{i=1}^{|\overleftarrow{N}(v)|} (l_i - 1) \geq N - 1 \Bigg\} \end{aligned} \tag{5.10}$$

To prevent two successive jumps *(a)*, we include a dependence on the previous step. Hence, we denote by $\mathrm{Pred}_v(\mathbf{x}_v)$ the predecessor node of $v$ and its state on the best path to reach $v$'s state $\mathbf{x}_v$. To model that at each junction on layer $N > 1$ the cumulative number of jumps to reach layer $N$ must be $N - 1$ *(b)*, (5.10) defines $\mathscr{A}$ as the set of admissible combinations of selecting incoming nodes $u_i \in \overleftarrow{N}(v)$ from layers $l_i$. Some admissible sets are visualized in Figure 5.3**a**.

**Limitations** To forbid two jumps in a row, equation (5.9) makes jumps available conditioned on a previously made decision. This introduces long range dependencies that invalidate the subproblem optimality criterion for DP to yield the correct result. It is thus possible that DP does not reach the root on layer $k$ (or only with infinite cost), as we show with a counterexample in Figure 5.3**b**. And, due to the same problem, even if a valid solution is found, it is not necessarily optimal. Also note that the set $\mathscr{A}$ of admissible combinations of incoming edges grows combinatorially, making this approach unsuited for large $k$.

### 5.5.2 Diversity Accumulation

Instead of using $k$ layers, one can also formulate a heuristic on a two-layer graph that ensures that any found solution contains the desired amount of diversity. To do so, we reformulate the Hamming distance constraint (that the new solution must differ from the previous one at $k$ nodes) as a constraint on accumulated diversity, *i.e.*, that $\sum_{v \in \mathcal{V}} \alpha_v(\mathbf{x}_v) > T$, where $\alpha$ is a measure of diversity per node and state, and $T$ a threshold.

We can formalize diversity accumulation in the dynamic programming setting as

$$\min_x \sum_{v \in \mathcal{V}} \theta_v(\mathbf{x}_v) + \sum_{(u,v) \in \mathcal{E}} \theta_{uv}(\mathbf{x}_u, \mathbf{x}_v) \tag{5.11}$$

$$\text{s. t. } \sum_{v \in \mathcal{V}} \alpha_v(\mathbf{x}_v) + \sum_{(u,v) \in \mathcal{E}} \alpha_{uv}(\mathbf{x}_u, \mathbf{x}_v) > T. \tag{5.12}$$

We change the DP update rules as follows: First, while propagating messages from the leaves of the tree to the root in layer 1, one must also propagate the amount of diversity accumulated by the corresponding configuration of the subtree. Let us denote nodes and edges of the subtree

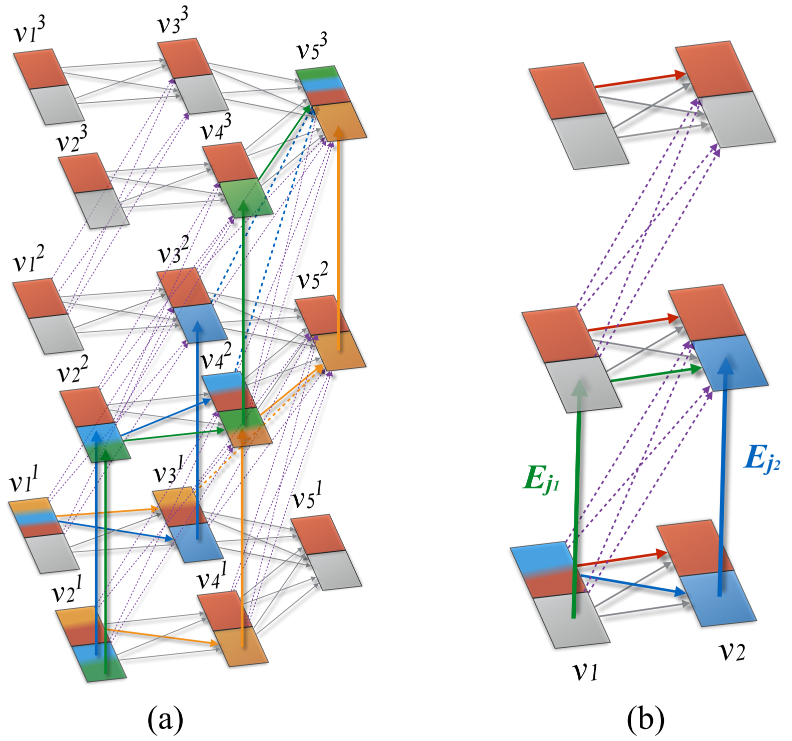


**Figure 5.3:** **(a)** Visualization of different ways of obtaining a Hamming distance of 2, where the red states show the previous solution. To reach $v_5^3$, one can jump two times at different nodes (green) and arrive in layer 3, jump once before to layer 2 and then to 3 at $v_5$ (orange), or to combine two incoming branches from layer 2 to get to layer 3 (blue). **(b)** Counterexample for $k = 2$. If $E_{j_2} < E_{j_1}$, the minimization will pick the predecessors shown in blue, which prevents the algorithm from jumping to layer 3 and finding a valid solution.

rooted at node $v$ in layer 1 by $\overleftarrow{\mathcal{V}}_{v^1}$ and $\overleftarrow{\mathcal{E}}_{v^1}$ respectively. The accumulated diversity $\mathcal{A}$ is given by

$$\mathcal{A}_{v^1}(\mathbf{x}_{v^1}) := \sum_{v \in \overleftarrow{\mathcal{V}}_{v^1}} \alpha_v(\mathbf{x}_v) + \sum_{(u,v) \in \overleftarrow{\mathcal{E}}_{v^1}} \alpha_{uv}(\mathbf{x}_u, \mathbf{x}_v). \tag{5.13}$$

Then, we can set the layer jump potential $\tilde{\theta}_{v^1 v^2}(\mathbf{x}_{v^1}, \mathbf{x}_{v^2})$ to infinity as long as the accumulated diversity is below the desired threshold $\mathcal{A}_{v^1}(\mathbf{x}_{v^1}) < k$.

The limitation of this heuristic is that at each node and state we find the optimal subtree configuration by minimizing the energy without considering diversity. This can prevent us from finding solutions with large diversity, and this approach also suffers from the same problems as the $k$-layer setup. Yet it has an attractive runtime because it only requires 2 layers to find a solution with any Hamming distance $k$.

### 5.5.3 Extension to $M$ Diverse Solutions

Finding $M$ solutions with a Hamming distance of $k$ could be achieved by stacking $M \times (k+1)$ layers, but then the long range dependency problems depicted above are even more prominent. This setup would enforce a too high number of layer jumps than necessary for $M$ solutions with desired Hamming distance $k$, because within each $k+1$ layer stack for a certain solution we forbid successive jumps, but to be diverse with respect to different solutions this is allowed.

For example, if the third solution with distance $k = 2$ would diverge from the two previous configurations at two nodes, it would still just be able to pass through the $k + 1$ layers of the first solution. With diversity accumulation on the other hand, $M$ diverse solutions can be obtained heuristically by using one diversity map $\alpha$ and one accumulator $\mathcal{A}$ per previous solution. The jump criterion must then ensure that enough diversity has been accumulated with respect to each previous solution. Yet, this approach also limited by the fact that diversity is only considered when enabling jumps, not while finding optimal subproblem configurations in the lower layer.

## 5.6 Applications and Experiments

We now evaluate the performance of our heuristics to obtain diverse solutions with prior work, and demonstrate the applicability to several problems in Computer Vision.

**Comparison with Existing Works:** [Batra et al., 2012, Yadollahpour et al., 2013, Kirillov et al., 2015] search for the diverse-$M$-best solutions by incorporating the diversity constraint via Lagrangian relaxation. Our heuristics follow a different approach and turn the constraint into a lower bound instead of relaxing it. The resulting advantage is that we guarantee the set of solutions to be as diverse as required, at the possible expense of a higher cost or the inability to find a solution at all. On $50$ random trees, with $100$ nodes each, all nodes having $3$ states with unary and pairwise potentials drawn uniformly from the range $[0, 1]$, we evaluate different Hamming distances in Figure 5.4. We let the method of [Batra et al., 2012] run for $100$ iterations, with a step size of $1/n$ in iteration $n$. In terms of runtime, diversity accumulation stands out because it constantly requires only two layers. Because the distance to the best configuration is not enforced by hard constraints, the solutions found by [Batra et al., 2012] often offer too little diversity, yielding a too low mean Hamming distance. Diversity accumulation gives solutions with more diversity than required, and hence also deviates more from the optimal energy. In terms of returned diversity and energy, the multi-layer dynamic programming solution yields favorable results compared to the other two methods, but is unfortunately slower – it suffers from the combinatorial explosion of admissible edge sets to consider – and fails to find a valid solution on quite a lot of trees due to the limitations described in Section 5.5.

**Medial Axis Identification in Biological Objects:** Identifying the medial axis of biological objects is a common problem in bio-image analysis, as it serves as a basis for length or growth estimation and tracking-by-assignment. Simple dynamic programming can achieve this task given the end points, although, as biological images tend to get noisy or crowded, designing a robust cost function is difficult. In Figure 5.5, we illustrate the usefulness of searching for a collection of possible best solutions instead of only one shortest path in phase-contrast microscopy images of Mycobacteria – as we will see in the following Chapter 6 – and in brightfield microscopy images of *C. elegans* nematodes.

**Selection of Segmentation Hypotheses:** In datasets with cell clumps, it is often hard to select the correct detections from a set of segmentation hypotheses. We illustrate this problem in images from the Mitocheck project dataset[2] [Held et al., 2010] using the tree model proposed in [Arteta et al., 2013]. There, the task is to assign a class label to each element of a set of nested maximally stable extremal regions. The labels indicate the number of objects that each

[2]http://www.mitocheck.org/

particular region represents. In the tree, nodes correspond to regions, and edges between parent and child node model the nestedness properties. In Figure 5.6, we show the results obtained when constraining dynamic programming with our $M$-best approach. Both of the above are useful to generate segmentation or pose candidates as needed by joint segmentation and tracking procedures, *e.g.* [Jug et al., 2014, Schiegg et al., 2014].

**Panorama Stitching:** In our motivation in Section 5.3 we mentioned that the proposed multi-layer setup can also be used for shortest paths. Here, we apply that in the context of boundary seam computation for panorama stitching [Summa et al., 2012]. We stitch images taken during the Apollo 11 moon landing (Apollo-Armstrong: 2 images of $2349 \times 2366$, courtesy of NASA). As observed in Figure 5.7, the second diverse shortest path also corresponds to a visually correct stitching, although the resulting path significantly differs from the globally optimal one.

**Disparity Map Estimation from Stereo Images** To generate different disparity maps from stereo images, we build a minimal spanning tree of the pixel grid graph using the intensity gradient as edge weight as in [Veksler, 2005]. Neighboring pixels are connected whenever they have similar intensities. Those that are not similar are not connected and hence are not penalized when generating depth discontinuities. We allow disparities of up to $40$ pixels in either direction while computing matching costs on patches of $11 \times 5$ pixels, and use a (non-truncated) quadratic attractive potential on the edges. While this setup is far from state-of-the-art in stereo, it demonstrates that our approach scales to large trees with many labels. We used the proposed diversity accumulation method where one unit of diversity is collected at every state that is at least a distance of $5$ away from the previous solution in label space, and requested a large amount of diversity to obtain visually different depth maps.

## 5.7 Conclusion

We have presented a multi-layer graph construction that allows formulating the $M$-best problem for tree-shaped graphical models efficiently through dynamic programming. This flexible framework can be used to find $M$-best solutions for a Hamming distance of $k = 1$ optimally. For $k > 1$, we present two heuristics, one using a multi-layer graph, and one using two-layers where each new configuration must accumulate diversity before it can reach the upper layer. We evaluate both heuristics against diverse-$M$-best [Batra et al., 2012], revealing that both perform favorably with certain strengths over the baseline. We demonstrated for several practical applications that the presented methods can reveal interesting alternative solutions, and will use it in the next Chapter to extract alternative bacteria detection hypotheses.

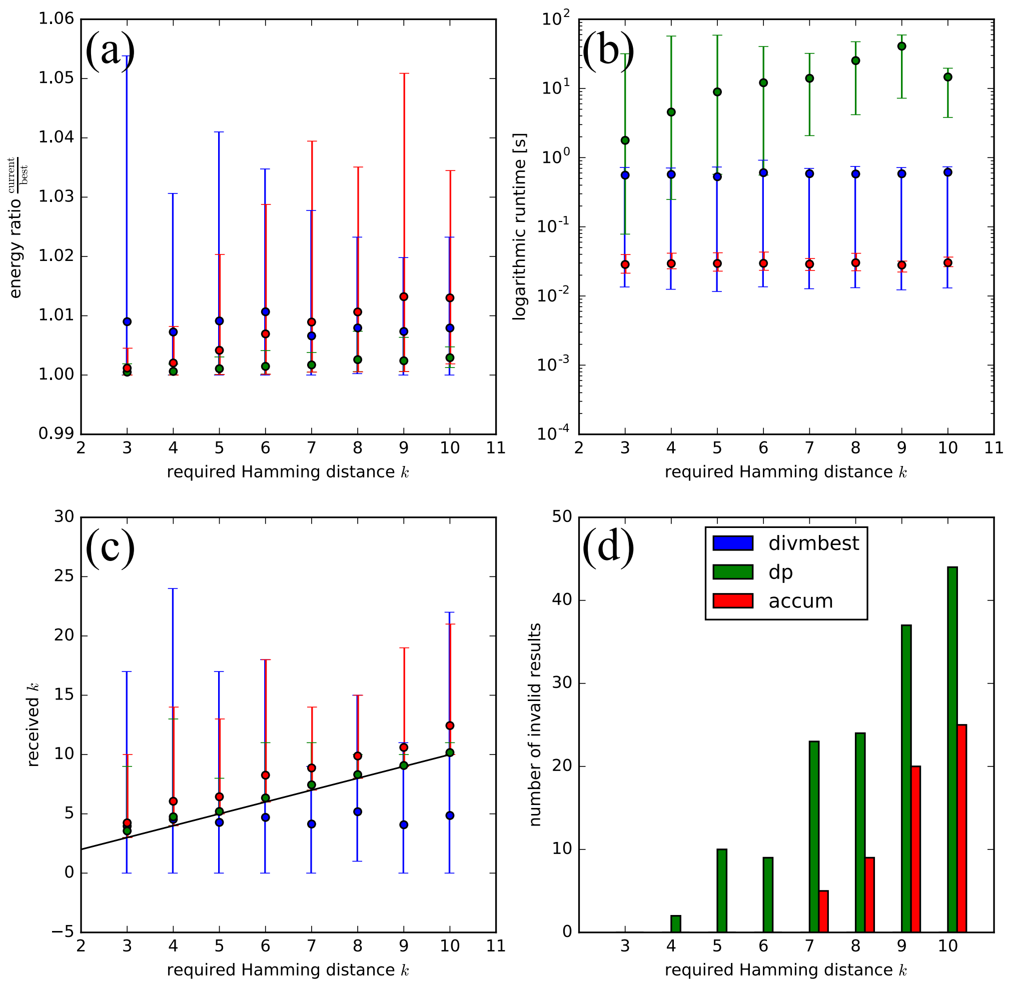


**Figure 5.4:** We compare the $k+1$-layer `dp` and diversity `accumulation` heuristic for obtaining diverse solutions from Section 5.5 against `divmbest` [Batra et al., 2012]. All results show mean, minimum and maximum over the valid solutions obtained for every setting on 50 random trees, where **(d)** shows the number of experiments that did not find a valid solution. **(a)** Energy ratio between the optimal unconstrained solution and the one with Hamming distance $k$. **(b)** Runtime. **(c)** Hamming distance of the resulting solution. Lower is better in all plots but **(c)**, where the returned Hamming distance should be close to, or preferably above the drawn diagonal.

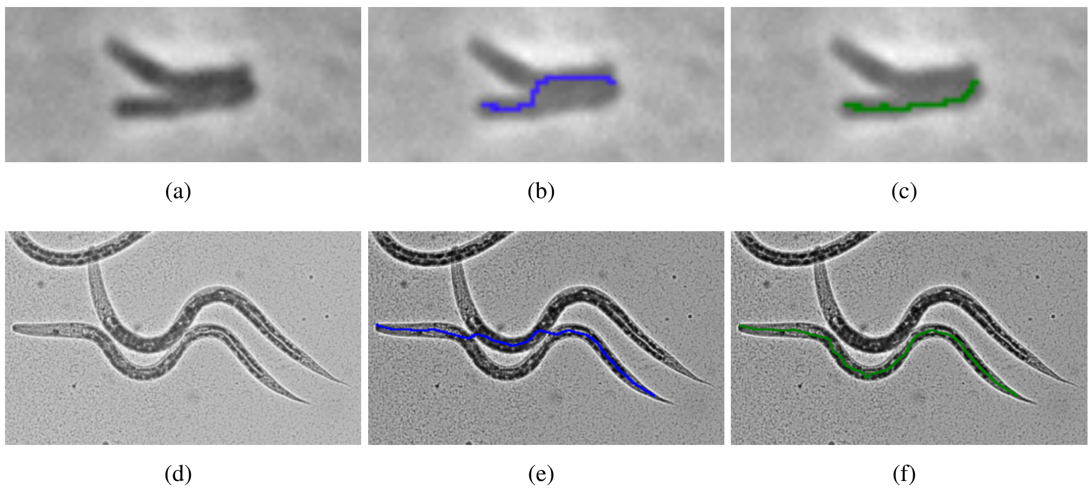


**Figure 5.5:** Diverse shortest path finding in bio-images featuring objects in close contact. **(a)** Raw phase-contrast microscope images of Mycobacteria, **(b)** first best path between two auto-detected end points, and **(c)** 6th best solution using diversity accumulation and $k = 6$. **(d)** Raw brightfield microscope images of *C. elegans*, **(e)** first best path, and **(f)** 5th best path between auto-selected end points using $k = 50$.

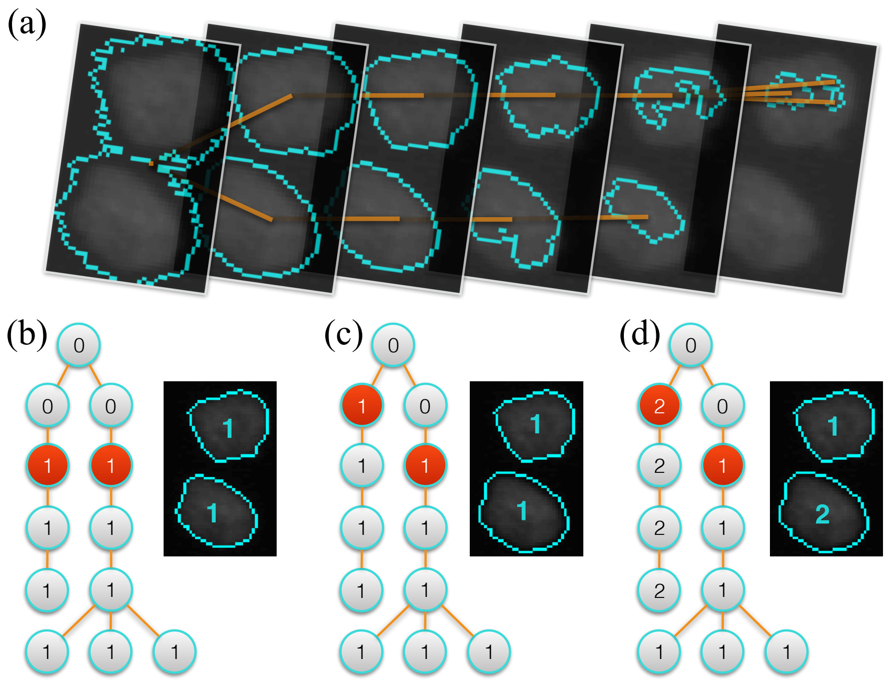


**Figure 5.6:** Finding the $M$-best configurations of a tree **(a)** of MSER segmentation hypotheses as in [Arteta et al., 2013]. The best **(b)**, second **(c)** and third best configuration **(d)** found by blocking the previous solutions in the respective *layer-jump-edge* potentials. The selected label at each node denotes the predicted object count of the first nonzero ancestor in the tree.

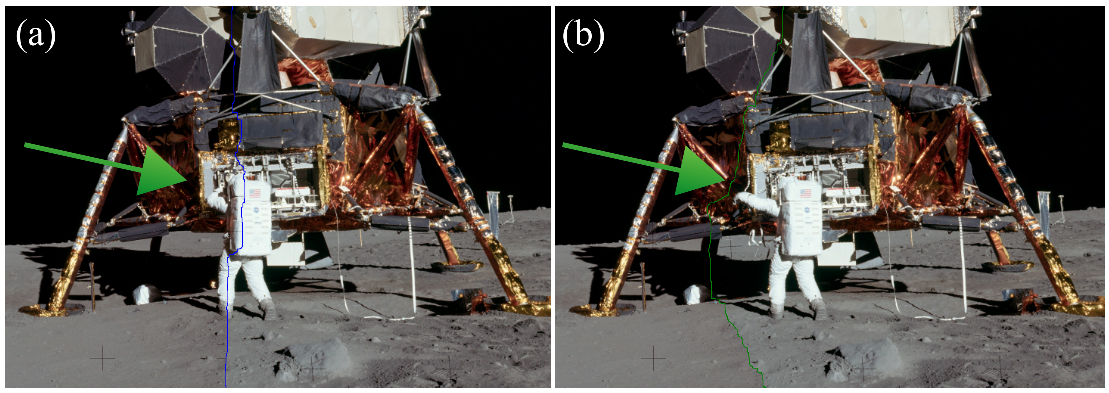


**Figure 5.7:** Finding diverse best paths (seams) for panorama stitching. Once the best solution **(a)** has been found, layer-jump-edges were blocked in a large corridor around it to obtain the diverse second best solution **(b)**.

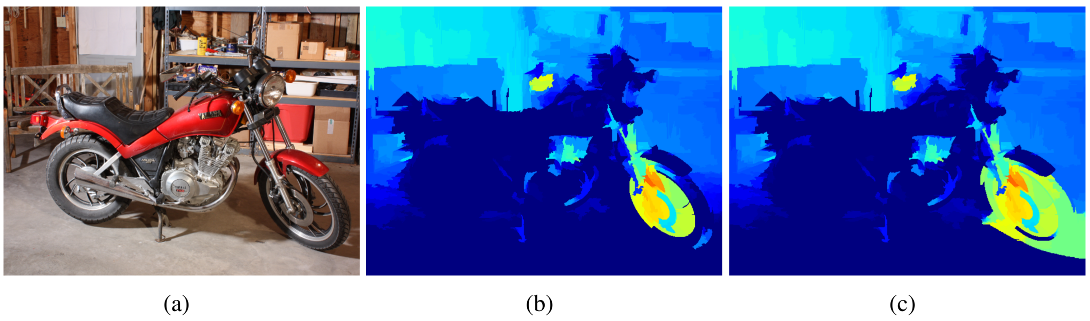


**Figure 5.8:** Exploring diverse solutions for disparity map estimation, on an image from the Middlebury benchmark [Scharstein et al., 2014] resized to $741 \times 500$. **(a)** Left view of the motorbike image pair, and corresponding **(b)** best solution found by [Veksler, 2005] which struggles inside the front wheel. **(c)** Enforcing a large Hamming distance (here 13000) reveals that the area around the front wheel could have been matched differently, exposing ambiguities in the estimation process.

Chapter 6

# Joint Selection of Hypotheses and Tracking of Mycobacteria

In Chapters 2 and 3 we have worked with a tracking model that assumes detections are coming from a simple segmentation of every input frame – and hence are not competing – and where the offset of the center of an object was a good indicator for its movement between frames. There are many tracking scenarios in which these assumptions do not hold. In this Chapter we present a pipeline for such a scenario, develop a specialized mycobacteria detector, enrich the set of detection hypotheses by the approach from Chapter 5, and formulate an optimization problem that jointly selects the most plausible detection hypotheses and tracks them over time. We provide results of an initial evaluation and provide an outlook on the next steps that are required to make this useful in practice.

## 6.1 Introduction

Mycobacteria are the pathogenic agent responsible for tuberculosis, one of the top killing diseases, especially in third-world countries [Raviglione and Sulis, 2016]. Curing infections caused by mycobacteria requires long-term antibiotic treatments in order to completely eradicate bacterial colonies. On top of that, the presence of mutant strains necessitates alternating treatments with different antibiotic compounds, further complexifying the curing process [McKinney, 2000]. Studying the behavior of mycobacteria is the essential initial step both for understanding the mechanisms of resistance and for investigating potential drug candidates. Basic parameters of interest include e.g. growth rate, interdivision time, length at division, a quantification of the expression of specific proteins, *etc.*. For long, such analyses were performed on the cell colony level. It is however now known that cell populations are not normally distributed, neither at the genetic nor at the phenotypic level [Elowitz et al., 2002]. Significant individual cell-to-cell variations are in fact observed between bacteria from a same colony, even derived from a single ancestor [Wakamoto et al., 2013]. It is thus required to perform analysis at the single-cell level in order to be able to identify small subpopulations of mutants.

Automating the tracking of mycobacteria poses two main challenges. Firstly, the nature of these flexible cells prohibits applying a shape prior and the colonies they form are so densely packed that boundaries are not always visible, see Figure 6.1. This causes local ambiguities which make it necessary to consider a bigger context in time around a given frame to provide a

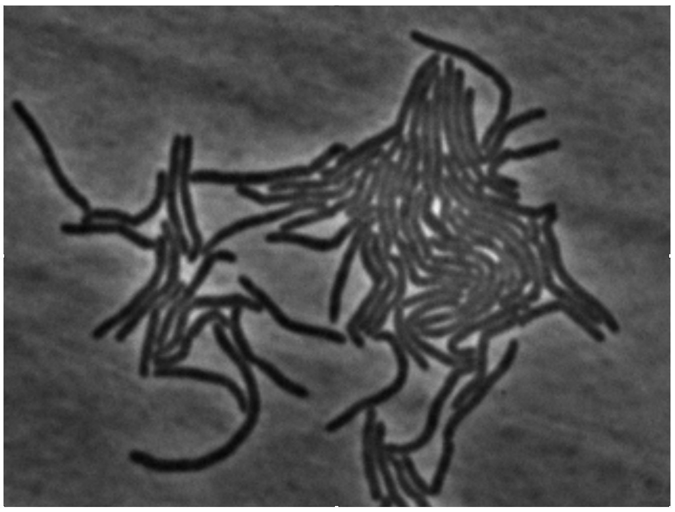


**Figure 6.1:** Sample phase-contrast image illustrating the two main challenges encountered with mycobacteria, namely poorly defined cell boundaries and high variety of shapes in term of length and bending.

solution in which we do not propagate errors. Secondly, divisions of mycobacteria happen in a non-standard way. The division occurs much earlier than cell body separation. As can be seen in the top of Figure 6.3, we cannot simply rely on the phase contrast channel in order to segment the cells, but have to consider a fluorescence signal from a protein located at the cells' poles, called *Wag31* [Santi et al., 2013].

Many approaches and tools are available for tracking worms or bacteria like *E. Coli* or *B. subtilis*, the most famous being MicrobeTracker [Sliusarenko et al., 2011] and Schnitzcells [Young et al., 2012]. But the cell populations they are designed for usually have decent contrast between bacteria or very well defined elliptical shapes. In addition, all these tools expect the cells to physically seperate right after a division. Our mycobacteria have none of those properties, hence those methods cannot be directly applied. One other solution for mycobacteria exists [Mekterović et al., 2014], but it requires users to annotate the divisions manually, defying the purpose of automated tracking.

In this chapter, we present an automated pipeline to analyze time-lapse microscopy images of mycobacteria. We propose to use a joint hypotheses selection and tracking approach as follows. In a preprocessing step, we process each frame of a given image sequence to extract a collection of candidate bacteria, or detection hypotheses. Then, we link detections from successive frames to model cell migration and division, building a graph. In this probabilistic graphical model, nodes represent random variables with features describing the probability – or quality – of each detection, transition or division event. The solution to this joint hypotheses selection and tracking problem is found as the most probable configuration of the graph. If the video was processed sequentially, one would have to take decisions based on locally ambiguous frames and hence impose an order in the assignment of individual tracks. The graphical model formulation on the other hand allows for solving the problem globally, thus we incorporate many more detection and linking possibilities in the graph than needed, and then rule out incorrect hypotheses by considering the sequence as a whole. Such approaches have already demonstrated successful results in other applications [Jug et al., 2014, Schiegg et al., 2014, Turetken et al., 2016]. This is, to the best of our knowledge, the first fully automated approach applicable to mycobacteria.

The data used in this chapter comes from the McKinney lab at EPFL. They feature *M. smegmatis*, the classic model for mycobacteria studies. The sequences are generally composed of 100 frames of $512 \times 512$ pixels of images with 16 bits per pixel.

## 6.2 Pipeline

The pipeline that we propose for automatically tracking mycobacteria can be roughly decomposed into three main steps:

1. Generating detection hypotheses by identifying bacteria candidates
2. Building a graphical model by linking spatially neighboring detection hypotheses of consecutive frames through transition and division hypotheses. We also extract features of all hypotheses and combine them in a per-hypotheses classifier response, which reduces the dimensionality and allows us to prune implausible hypotheses.
3. Finding the globally most probable configuration by solving an Integer Linear Program (ILP).

We will now describe these three steps of the pipeline, assuming we are given trained classifiers and tuned weights. In the next section we present how these classifiers and weights can be trained.

### 6.2.1 Hypotheses Generation

In order to extract detection hypotheses, we have two sources of information per timeframe: the *Wag31* channel shows fluorescent blobs corresponding to the cell extremities, and the phase contrast channel where we can identify cell contours, as seen at the top of Figure 6.3. Ideally we would like to use this information to obtain a full segmentation yielding the contour of every cell. To simplify the problem, we instead search for the medial axis of the bacteria, which we refer to as a *trace* (see Figure 6.2). The rationale behind this design choice is twofold. First, it makes it easier to handle information from both channels as the bacteria identification problem is reduced to finding a meaningful path in the phase channel between two coordinates corresponding to strong signal in the dots channel. Second, the trace provides an easy way for obtaining a complete segmentation of the cell body. Mycobacteria grow mostly at the extremities and have a rather conserved cell width, a good estimate of the cell contour can thus be obtained by searching for areas of strong gradient around the trace and extrapolating this to areas where borders are not visible.

As a preprocessing step, we use *ilastik* [Sommer et al., 2011] to train a Pixel Classification workflow on one dataset to predict the foreground (body) of the mycobacteria in all datasets. This gives us a probability map for every frame in every video that indicates how probable each pixel is to belong to a bacterium. Additionally we train a two-stage auto-context pixel classification *ilastik* workflow on one dataset to predict the endpoints of the mycobacteria in all datasets. These give us two probability maps for every frame in every video.

Then, for each frame, our approach is as follows:

1. We first identify cell extremity candidates by combining detections from two sources. We perform a Laplacian of Gaussian filtering on the *Wag31* channel and consider the probability map for endpoint locations from *ilastik*. Relying on these two sources increases the

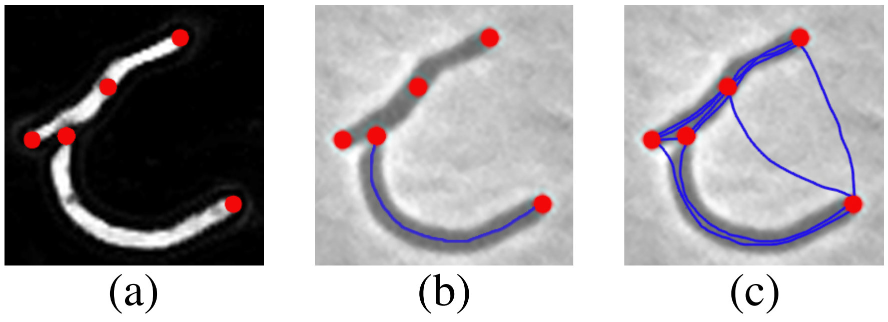


**Figure 6.2:** Traces generation pipeline: **(a)** The selected dot locations are used as seed points for dynamic programming on the foreground probability image. **(b)** A spline snake is fitted on the resulting path and optimized in order to obtain a smoother and more precise trace. **(c)**The procedure is repeated for all pairs of dots in the frame.

robustness of the algorithm, as the *Wag31* channel can robustly identify divisions of cells that have not yet physically separated, and the probability map – for which *ilastik* computed features from both input channels – prevents us from losing extremities when the fluorescence of old end points decays over time. We then threshold these two processed images to identify locations of strong pixel intensity. We merge the detections from the two sources and apply non-maxima suppression.

2. For each pair of points detected in the previous step, we run dynamic programming to find the shortest path between the two points, using the foreground probability map obtained with *ilastik* as costs. To increase the likelihood for finding the medial axis, we search for a collection of $M$-diverse shortest paths per point-pair as presented in the previous Chapter 5. The actual number of shortest paths $M$ is not fixed a priori. Instead, we iteratively compute the next-best path subject to diversity constraints and stop as soon as the cost of the newly found path differs too much from the cost of the previous path. In practice, we continue the search as long as the shortest path cost does not increase by more than $10\%$.

3. Lastly we fit an open Hermite spline-snake [Uhlmann et al., 2014] with loose ends to each shortest path. Using Hermite splines instead of other bases has the advantage that boundary conditions are handled gracefully. We optimize the snake allowing only the endpoints to move towards areas with highest tip probability in order to fine-tune cell extremity localization. This corrects for little endpoint hypotheses offsets that could have been introduced by thresholding and non-maximum suppresion in step 1. Then we extract several informative features of the trace such as *e.g.* curvature.

### 6.2.2 Building the Model

Based on the detections extracted as above, we can now construct a factor graph similar to that in Chapter 2. This graph covers all time frames of the input video. It is set up as a trellis graph, where each column contains nodes representing the detections of a specific time frame. Consecutive columns thus model subsequent time frames. We insert linking and division hypotheses between the columns as separate nodes connected to two or three detection nodes respectively. These linking and division hypotheses outgoing from any detection $i$ at time $t$ are determined by considering the 20 detections $j$ in the following frame $t + 1$ that are closest in terms of the minimal Euclidean distance of all points on the two traces. This is visualized in Figure 6.4, where circular nodes represent random variables for detections, links and divisions, colored boxes depict factors that depend on the connected random variables, and boxes represent hard constraints or factors. Hard constraints allow to model requirements for a consistent solution. We use the black boxes to model flow conservation, and orange for mutual exclusion of detection hypotheses in the same frame based on whether they cross and overlap. Factors on the other hand encode local probabilities and are commonly called unary if they are only connected to one variable. Each random variable can take a state or label out of a discrete label space, and hence factors can be seen as lookup tables of combinatorial size that represent the probability per variable configuration. For this very model, the label space of each variable is binary $0, 1$, simply indicating whether a hypothesis is used in the final tracking result or not. Each unary factor attached to a detection, link, or division node thus encodes the probability for using (state 1) or not using (state 0) this hypothesis.

We obtain these probabilities by first computing an extensive set of features for each hypothesis and then let a Random Forest classifier predict the probability for states *used* and *not used* based on those features. Applying such a classifier has the additional benefit that it allows us to non-linearly combine features of varying dynamic ranges into a single value. Because our initial sets of hypotheses are large to ensure that we do not miss a possibility, we also use those predicted probabilities to filter out implausible hypotheses. We apply very conservative thresholds and discard detections with less than $1\%$ probability and links and divisions with less than $0.01\%$ probability for being used. The whole process of extracting detections and pruning features is visualized in Figure 6.3.

### 6.2.3 Finding the Optimal Solution

To be able to formulate the optimization problem of finding the most probable configuration, let us first define the factor graph more precisely. We represent all detection, linking, and division hypotheses by nodes, where each division is connected by edges to one detection at time $t$ and two detections at time $t+1$, and every transition connects one detection at time $t$ to one detection at time $t+1$. Each detection can appear or disappear, which is modeled by special transition nodes which have one endpoint unconnected – or which could be connected to virtual source or sink. Hence in the equations below, we slightly abuse the notation and include appearances and disappearances in the transitions.

We encode configurations of the graph in the form of a state vector $\mathbf{x}$, composed of as many binary components as there are nodes in the graph. The binary components of $\mathbf{x}$ correspond to the state of each detection, transition, and division node in the graph. We use sub- and superscripts to address the state of individual variables in $\mathbf{x}$ such as $\mathbf{x}_{t,i}^{\mathrm{det}}$ for the state of detection hypotheses $i$ in frame $t$. A segmentation and tracking solution is then given by the subset of active nodes in the state vector.

All detection nodes $i$ in each frame $t$ have a unary factor $\theta_{t,i}^{\mathrm{det}}$ attached, which contains the energy of using ($\mathbf{x}_{t,i}^{\mathrm{det}} = 1$) or not using ($\mathbf{x}_{t,i}^{\mathrm{det}} = 0$) this very detection. Link and division nodes are modeled accordingly, but use 2 or 3 detection node indices respectively for unique addressing. We transform the probabilities predicted by the Random Forest classifiers into energies by applying the negative logarithm, which turns the product over all factors constituting the joint probability into a sum over energies. Thus unary factors all take the form

$$\theta_{t,i}^{\mathrm{det}}(\mathbf{x}_{t,i}^{\mathrm{det}}) := -\log\left(P^{\mathrm{det}}(\mathbf{x}_{t,i}^{\mathrm{det}} = 1)\right) \cdot w^{\mathrm{det}},$$

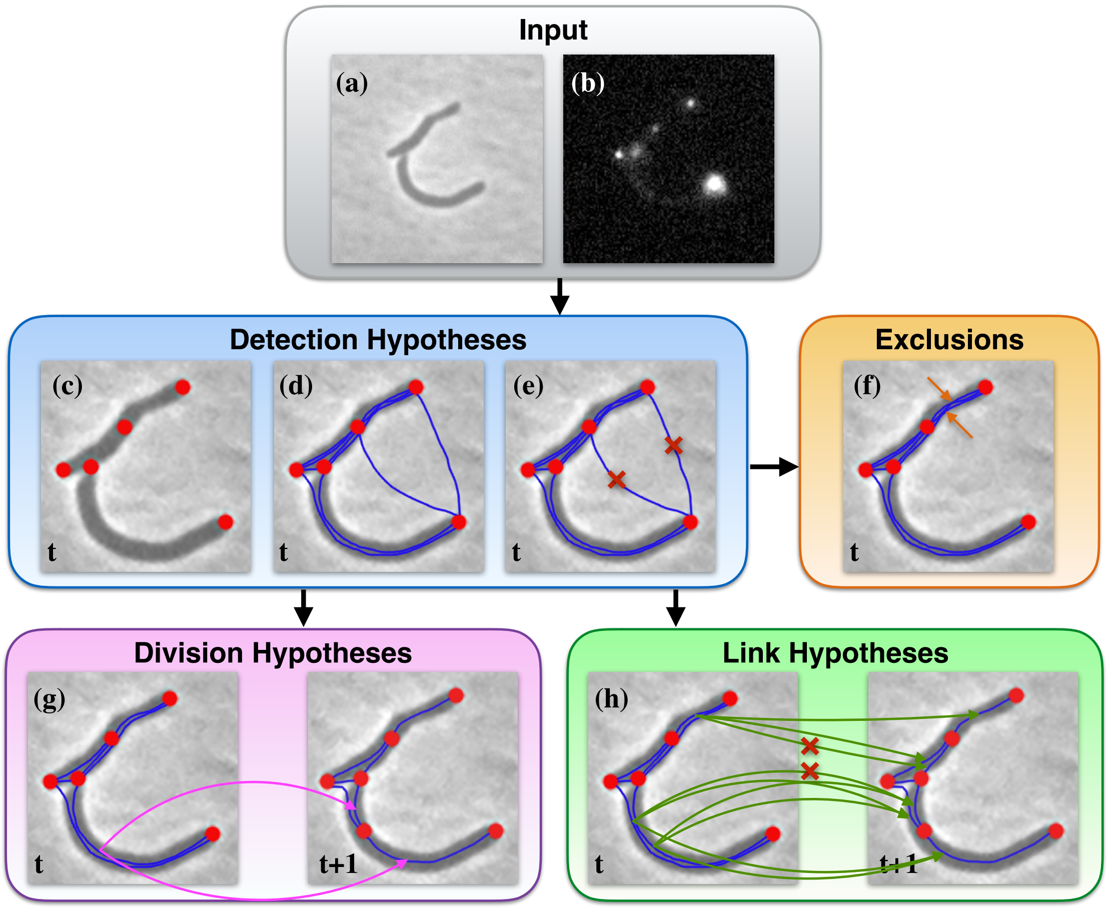


**Figure 6.3:** Extracting detection, linking and division hypotheses. **(a)** Phase contrast channel of the input data. **(b)** The fluorescence channel shows the stained end points of the bacteria. Notice the dot in the upper middle, indicating that a division has happened here even though it is not visible in the phase contrast image yet. **(c)** Extracted dots. **(d)** Shortest path refined by a Hermite snake fit between adjacent dots. **(e)** Pruning of hypotheses based on their probability of being used. **(f)** Detecting overlapping and crossing hypotheses for mutual exclusion during tracking. **(g)** Division hypotheses generation by pairing one detection in the current with two close detection hypotheses in the next frame. **(h)** Link hypotheses generation by pairing detections in the current with close detection hypotheses in the next frame. These hypotheses are also pruned if their probability is too low.

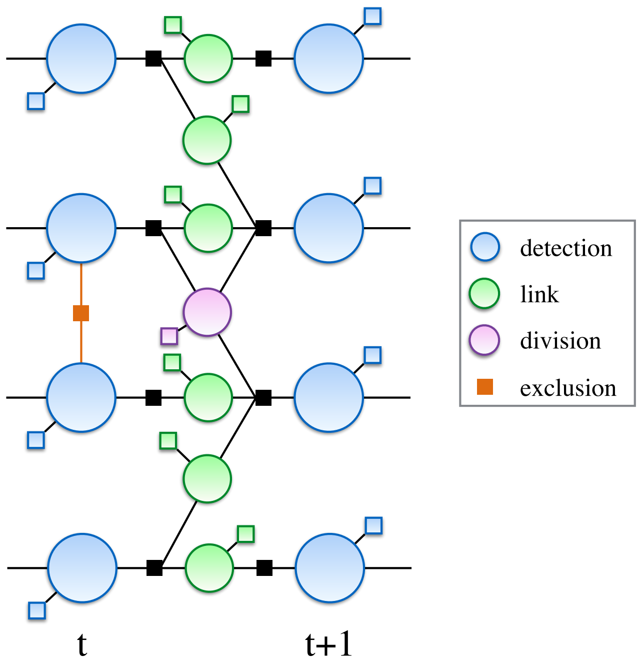


**Figure 6.4:** Schematic view of the graphical model, where blue detection hypotheses can be competing in each frame, and hence orange exclusion constraints ensure that only one of them can be active. Greeen link and purple division nodes connect detections between time frames.

only appearance and disappearance transitions have constant penalty if they are used such as $\theta^{\text{app}}_{t,i}(\mathbf{x}^{\text{app}}_{t,i}) := \mathbf{w}^{\text{app}} \cdot \delta(\mathbf{x}^{\text{app}}_{t,i} = 1)$ using the $\delta$ function which takes value 1 only if the argument is *true*, otherwise 0. We set the appearance penalty in the first, and the disappearance penalty in the last frame to zero to encourage tracks to start and end there. If we denote the set of all detections per frame by $X_t$, we can the formulate the global objective function as

$$\begin{aligned}\min_{\mathbf{x}} \sum_{t=0}^{T} \sum_{i \in \mathcal{X}_t} \theta^{\text{det}}_{t,i}(\mathbf{x}^{\text{det}}_{t,i}) + \sum_{t=0}^{T-1} \sum_{i \in \mathcal{X}_t} \sum_{j \in \mathcal{X}_{t+1}} \theta^{\text{link}}_{t,i,j}(\mathbf{x}^{\text{link}}_{t,i,j}) \\ + \sum_{t=0}^{T-1} \sum_{i \in \mathcal{X}_t} \sum_{j \in \mathcal{X}_{t+1}} \sum_{\substack{k \in \mathcal{X}_{t+1} \\ k \neq j}} \theta^{\text{div}}_{t,i,j,k}(\mathbf{x}^{\text{div}}_{t,i,j,k})\end{aligned} \tag{6.1}$$

*s.t.:*

**Flow conservation:**

$$\forall t\ \forall i \in \mathcal{X}_t : \mathbf{x}^{\text{det}}_{t,i} - \sum_{j \in \mathcal{X}_{t+1}} \mathbf{x}^{\text{link}}_{t,i,j} - \sum_{j \in \mathcal{X}_{t+1}} \sum_{\substack{k \in \mathcal{X}_{t+1} \\ k \neq j}} \mathbf{x}^{\text{div}}_{t,i,j,k} = 0 \tag{6.2}$$

$$\forall t\ \forall i \in \mathcal{X}_t : \mathbf{x}^{\text{det}}_{t,i} - \sum_{j \in \mathcal{X}_{t-1}} \mathbf{x}^{\text{link}}_{t-1,j,i} - \sum_{j \in \mathcal{X}_{t-1}} \sum_{\substack{k \in \mathcal{X}_{t-1} \\ k \neq j}} \mathbf{x}^{\text{div}}_{t-1,j,i,j} = 0 \tag{6.3}$$

**Exclusion constraints:**

$$\forall t\ \forall i,j \in C_t : \mathbf{x}^{\text{det}}_{t,i} + \mathbf{x}^{\text{det}}_{t,j} \leq 1. \tag{6.4}$$

To conserve flow, and to ensure that there is at most one division per parent, we enforce that the sum of active outgoing links and divisions is equal to the detection value (6.2). Analogously, the sum of active incoming hypotheses needs to equal the detection value of this detection (6.3). These are visualized as black boxes in Figure 6.4. If we let $C_t$ be the set of mutually exclusive pairs of detections in frame $t$ – which we find by determining whether two hypotheses overlap or cross – then the exclusion constraints can be defined as in (6.4).

Each unary $\theta_i(\mathbf{x}_i)$ can be written in terms of indicator variables – here implicitly done through the $\delta$ function – as

$$\theta_i(\mathbf{x}_i) := -w \cdot (\log(P_i(\mathbf{x}_i = 1)) \cdot \delta(\mathbf{x}_i = 1) + \log(P_i(\mathbf{x}_i = 0)) \cdot \delta(\mathbf{x}_i = 0)) .$$

This reveals that equation (6.1) is an instance of an integer linear program (ILP). These ILPs can be solved for problems of reasonable size by commercial solvers like Gurobi and CPLEX, we use CPLEX for our experiments. Once we have obtained the optimal configuration $\mathbf{x}$, we can decode the tracking solution in terms of selected detection, linking and division hypotheses.

## 6.3 Training the Pipeline

In the previous Section we have assumed that the Random Forests as well as the weighting terms $w^{\text{det}}, w^{\text{link}}, w^{\text{div}}, w^{\text{app}}$, and $w^{\text{dis}}$ for the different types of hypotheses are given. To make the proposed pipeline useful in practice, these ingredients must be chosen as good as possible. In this section we present how classifiers and weights can be trained from annotated data. To be able to train and evaluate the pipeline, we curated a ground truth (GT) by manually annotating every mycobacterium in every frame as a line of pixels and then assigning them to their successors in the next frame, and did so for multiple datasets.

**Per-Pixel Probability Maps:** Before we can even extract detection hypotheses, we have to generate the tip probability and bacteria probability for every pixel in every frame of all images. We obtain both by interactively training per-pixel Random Forest classifiers in *ilastik* on one dataset. Firstly, for the tip probability, we use the 2-stage auto-context [Tu, 2008] workflow where the first stage is trained to separate pixels that belong to mycobacteria body, mycobacteria tips, background, or mycobacteria boundary. The second stage then performs a binary classification to separate tips from background. To predict the foreground (body) of the mycobacteria in all datasets, we use the pixel classification workflow and again train a Random Forest, but this time only a single stage with the classes mycobacteria, boundary and background.

**Training the Random Forests:** To train Random Forests that can predict whether a detection, linking, or division hypothesis looks plausible, we need to provide positive and negative training examples. Hence, we build the factor graph and compute features of all hypotheses as explained in the previous section. Then – disregarding flow and exclusion constraints – we match the detections from the pixel-wise ground truth into the space of our hypotheses. For every annotated bacterium, we compute its distance to all hypotheses in this frame. We use the area spanned by the two curves as distance measure. Then we find a Hungarian [Kuhn, 1955] matching from GT bacteria to hypotheses. This greedily determines the links and divisions in the naïvely mapped ground truth. For training, we use samples from the full training datasets, because on the one hand the density of cells changes a lot over the course of the videos, and on the other hand the classifier should be robust to some inter-dataset variability.

**Finding optimal weights through Structured Learning:** We propose to seek the optimal weights by training a structured support vector machine [Tsochantaridis et al., 2004] (SSVM) with maximal margin, similar to *e.g.* [Yadollahpour et al., 2013, Funke et al., 2015]. Briefly, a SSVM optimizes a compatibility function $f(c)$ such that, based on observation $c$, a structured output $\mathbf{y} = f(c)$ is predicted that is as close to the desired solution $\mathbf{y}^*$ as possible. By structured we mean that $\mathbf{y}$ is more than just a number or label, it is in our case a full tracking solution. We choose this compatibility function as $f(c) := \operatorname{argmax}_{\mathbf{y}} < \mathbf{w}, \Phi(c)\mathbf{y} >$ to return the configuration that maximizes the dot product of the weights $\mathbf{w}$ and some joint feature matrix $\Phi(c)\mathbf{y}$. Then, given a training example $\mathbf{y}^*$ and the corresponding features $\Phi(c^*)$, one can optimize for $\mathbf{w}$ by the SSVM objective

$$\mathbf{w}^* = \operatorname*{argmin}_{\mathbf{w}} \tfrac{\lambda}{2}||\mathbf{w}^2|| + \max_{y}\big(< \mathbf{w}, \Phi(c^*)\mathbf{y}^* - \Phi(c^*)\mathbf{y} > +\Delta(\mathbf{y}, \mathbf{y}^*)\big), \tag{6.5}$$

where $\lambda$ modulates the influence of the quadratic regularizer on $\mathbf{w}$, and $\Delta(\mathbf{y}, \mathbf{y}^*)$ represents an application specific loss or distance between the two solutions. Roughly speaking this means we seek to find the minimal length weight vector that maximizes the margin by which the optimal solution $\mathbf{y}^*$ gets scored higher than any other solution $\mathbf{y}$, where the margin is additionally scaled by how different the solutions are. Note that the manually annotated ground truth must not necessarily be part of the set of hypotheses we extract, and hence the GT cannot be perfectly represented in our model. As a surrogate, we use the solution that our model can represent which is as close to the GT as possible, and call this the *best effort solution* as in [Funke et al., 2015]. We now first show how we obtain the best effort solution, and then formulate the SSVM training tailored to our tracking problem.

To find the best matching *consistent* solution $\tilde{\mathbf{x}}$ that our model can represent, we build an auxiliary tracking model with one node per possible match of each detection hypotheses $i$ with the three closest GT bacteria $GT(t,i)$ in the same frame $t$. For every matching hypothesis, we replace the detection potential $\theta^{\text{det}}$ by a distance $d$ between our detection hypothesis and the selected GT bacterium $g \in GT(t,i)$ as $\tilde{\theta}^{\text{det}}_{t,i,g}(\mathbf{x}^{\text{det}}_{t,i,g}) := \mathbf{w}^{\text{det}} \cdot d(t,i,g)$. To insert transition and division hypotheses, we proceed exactly as before, but now we replicate those hypotheses for every GT matching possibility of all involved detections. Additional exclusion constraints ensure that at most one of the replicas of each detection is used, and that every GT bacterium is matched at most once. We use empirically chosen weights $\mathbf{w}^{\text{det}} = 30, \mathbf{w}^{\text{link}} = \mathbf{w}^{\text{div}} = 1, \mathbf{w}^{\text{app}} = \mathbf{w}^{\text{dis}} = 100$ for the costs. With the new detection-matching potentials and replicated $\theta^{\text{link}}$ and $\theta^{\text{div}}$, the optimization for the best effort solution $\tilde{\mathbf{x}}$ becomes

$$\min_{\tilde{\mathbf{x}}} \sum_{t=0}^{T} \sum_{i\in\mathcal{X}_t} \sum_{g\in GT(t,i)} \tilde{\theta}^{\text{det}}_{t,i,g}(\tilde{\mathbf{x}}^{\text{det}}_{t,i,g}) + \sum_{t=0}^{T-1} \sum_{i\in\mathcal{X}_t} \sum_{g\in GT(t,i)} \sum_{j\in\mathcal{X}_{t+1}} \sum_{g'\in GT(t+1,j)} \theta^{\text{link}}_{t,i,g,j,g'}(\tilde{\mathbf{x}}^{\text{link}}_{t,i,g,j,g'}) \tag{6.6}$$

$$+ \sum_{t=0}^{T-1} \sum_{i\in\mathcal{X}_t} \sum_{g\in GT(t,i)} \sum_{j\in\mathcal{X}_{t+1}} \sum_{g'\in GT(t+1,j)} \sum_{\substack{k\in\mathcal{X}_{t+1}\\ k\neq j}} \sum_{g''\in GT(t+1,k)} \theta^{\text{div}}_{t,i,g,j,g',k,g''}(\tilde{\mathbf{x}}^{\text{div}}_{t,i,g,j,g',k,g''}), \tag{6.7}$$

subject to the same constraints as in (6.1) plus that every GT object and every detection is matched at most once.

Based on the best effort solution $\tilde{\mathbf{x}}$, we can find the optimal weighting of the different energies predicted by random forest classifiers by solving the structured learning objective from above. For a given annotated tracking dataset, the joint feature matrix $\Phi(c)$ contains the entries of all unary factors without the weights. If we replace our tracking solution $\mathbf{x}$ by an indicator vector $\mathbf{y}$ two times the length of $\mathbf{x}$ – where for every variable $i$'s state $\mathbf{x}_i$, $\mathbf{y}$ contains two values $\delta(\mathbf{x}_i = 0)$ and $\delta(\mathbf{x}_i = 1)$ – it becomes clear that we can write our original ILP optimization problem from (6.1) as $\min_y < \mathbf{w}, \Phi(c)\mathbf{y} >$. If we additionally choose a loss function $\Delta(\tilde{\mathbf{x}}, \mathbf{x})$ that decomposes over all detections, divisions and transitions, then the inner maximization of (6.5) can be performed by solving a loss-augmented version of our original ILP. Hence we use the Hamming distance as loss. We use the OpenGM[1] implementation of the max-margin SSVM objective based on the bundle method by [Teo et al., 2010].

## 6.4 Experiments

We applied the pipeline to seven datasets of 100 time frames each. All datasets start with a single bacterium in the first frame, but due to many divisions they reach around 30 to 40 bacteria at the end of the video. We have a pixel-wise ground truth for all datasets. We trained the pixel classification and auto-context *ilastik* workflows for extremity location and for foreground probabilities on the first dataset (*No3*).

To assess the quality that our pipeline can reach, we trained and predicted on every dataset individually. For detections, links and divisions, we used all samples from the naïvely mapped ground truth of each dataset to train the respective Random Forests. We run structured learning based on the best effort solution up to frame 70. Up to frame 70 there are still only 8 to 10 bacteria in the dataset. Due to the exponential growth, the last 30 frames account for around $60\%$ of all cell detections in the ground truth. Table 6.1 shows the agreement of our solution with the naïvely mapped ground truth for the different datasets, in terms of f-measure of consistent detections, links and divisions.

Because the pipeline should eventually reduce the amount of input from the experts, we tried to randomly sample a small number of positive and negative examples across all datasets and train Random Forests from that. This did not work yield satisfying results, probably because the samples should cover the slight deviations across datasets and the strong change in cell densities between the beginning and the end of the videos. Next we trained Random Forests globally, using all samples of all datasets. Predicting with this classifier on all datasets has a comparable performance to training the Random Forests per dataset individually, as seen in Table 6.1.

The numbers indicate that our pipeline is able to produce results which are not very far from the naïvely mapped ground truth. But this mapping can also contain errors. Hence we compare the results to the pixel-wise ground truth by performing a per-frame Hungarian matching of ground truth bacteria to detections which are closer than an empirically chosen threshold on the integrated area spanned between the two traces. Then we again map the links and divisions onto those matched detections. The tracking solutions turn out to be indeed very sensible. In Table 6.2 we show the quality of one representative solution. The matching numbers of false negatives and false positives for detections and divisions indicate that the tracking solution chose a different path for a few cells, probably because our pipeline missed a detection hypotheses which would have been required to faithfully represent the ground truth tracks, or because a division was detected a frame too early or too late. See Figure 6.5 for an example. Another result, taken from

[1]https://github.com/opengm/opengm

| Dataset | Local RF | Global RF |
|---|---|---|
| No3 | 91.20 | 91.06 |
| No4 | 90.14 | 84.36 |
| No6 | 91.25 | 92.42 |
| No10 | 99.49 | 99.68 |
| No12 | 92.23 | 92.53 |
| No13 | 83.68 | 91.73 |
| No16 | 96.80 | 96.16 |

**Table 6.1:** F-measure agreement in percent with respect to the naïvely mapped ground truth of each dataset, when either training the Random Forests locally per dataset or globally across all datasets.

| Type | Number |
|---|---|
| missed detections | 14 |
| false positive detections | 14 |
| total GT detections | 722 |
| wrong links | 21 |
| total GT links | 669 |
| missed divisions | 2 |
| false positive divisions | 2 |
| total GT divisions | 26 |

**Table 6.2:** Comparing the quality of the tracking result (using the local Random Forest) of Dataset *No16* with the pixel-wise ground truth.

dataset *No10* is shown in Figure 1.1. The weights $\mathbf{w}$ learned through structured learning are similar across the datasets. This indicates that they can probably be learned once and transferred to other datasets, but we would have to validate this in additional experiments.

## 6.5 Conclusion and Outlook

In this section we have presented a pipeline for detecting and tracking the bacteria responsible for tuberculosis. We developed a special detection procedure that copes with the peculiarity that the extremal points of the cells can be made visible through staining even though the cells are not physically separated. We applied the method from Chapter 5 to obtain diverse hypotheses for the centerlines of the bacteria when pairing the detected extremal points. From that we built a graphical model for joint hypotheses selection and tracking and presented how the classifiers and their relative importance can be trained. Lastly we stated the results of first experiments, showing that the pipeline is indeed capable of finding very reasonable results. However, further experiments are required to evaluate the generalization capabilities of the involved machine learning methods. This requires finding a meaningful sub-sampling of the training data for the Random Forests, which should reduce the amount of overfitting while being closer to the desired usage scenario where only little annotation is provided by the expert. On the other hand, the weights could be learned by solving the SSVM objective on a combination of model excerpts from all datasets. By performing cross validation over the available datasets we would be able to judge our pipeline's generalization capabilities to unseen videos. Based on the promising

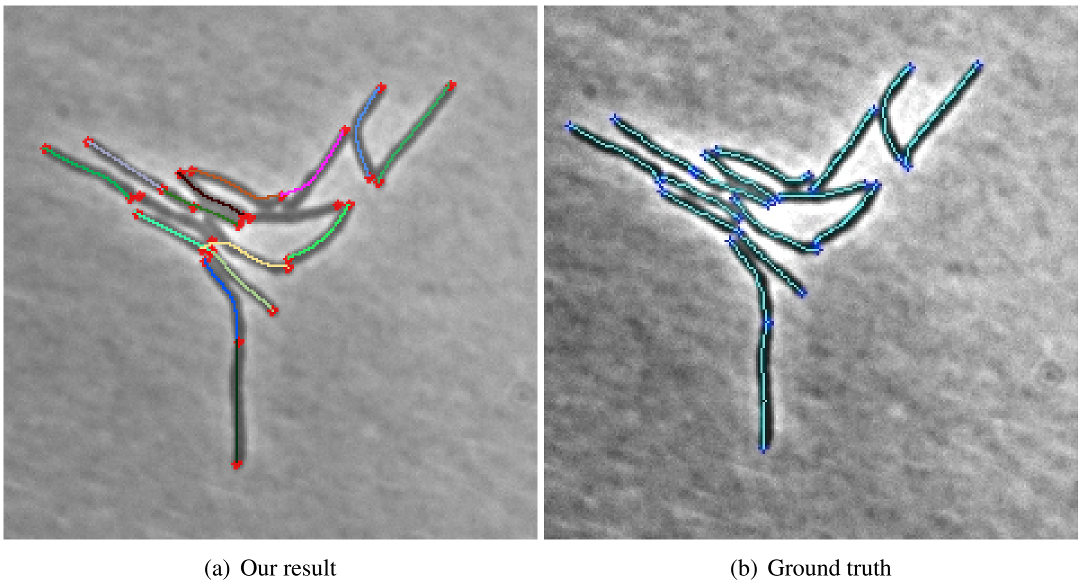


**Figure 6.5:** Our result using the globally trained Random Forest and the manually annotated ground truth of Frame 85 of dataset *No6*. The extremal points that we detected are highlighted in red, the differently colored lines represent the tracked detection hypotheses. While our approach misses an end point, the pipeline manages to let the pink trace take a little detour to be mostly consistent with the optimal solution. However, we also miss two bacteria completely, which is part of why we only achieve $\approx 92\%$ accuracy on this dataset.

preliminary results, we hope the presented pipeline can make the analysis of high throughput screening of tuberculosis bacteria tractable.

Chapter 7

# Discussion

We now first summarize the contributions presented throughout this thesis, then briefly describe the most important resulting software modules, and conclude with an overview of limitations of the used models and algorithms, as well as relevant future research directions.

## 7.1 Summary of Contributions

The abundance of tracking-by-assignment methods formulated as graphical models in the literature for multiple target tracking in general, but also when including additional constraints like divisions, speaks for their relevance and flexibility. In this thesis we have considerably accelerated tracking-by-assignment methods for dividing targets and lowered the hurdles for applying them in practice by providing the following:

- We investigated the tightness of the LP relaxation of a commonly used tracking model in Chapter 2. As it turned out to be rather loose, we reformulated the ILP as a constrained network flow model and show that while having the same expressive power, this formulation has a much tighter relaxation and can thus be solved faster by commercial general purpose ILP solvers. Lastly we show that replacing the unary factors by a convex upper envelope yields solutions of equivalent tracking quality while giving another speedup by a factor of roughly two. This total speedup of around 3 that we gained on two challenging real world tracking problems boosts the practicability of automated cell tracking methods.
- In Chapter 3 we developed a novel heuristic solver for tracking-by-assignment problems for dividing targets based on the successive shortest paths network flow algorithm. The method of incorporating constraints is not tied to divisions but fits any kind of network flow problem with constraints that can be expressed in terms of conditioned residual arc capacities. We compare this heuristic to solutions of the original ILP, revealing that it finds high-quality tracking solutions while having a competitive anytime performance and a much lower RAM usage. We show that the algorithm's runtime complexity is quadratic in the number of nodes, and our experiments confirm that it scales better with problem size than commercial solvers applied to this NP-hard objective. This brings two benefits: Firstly, it allows larger problems to be approached on smaller machines. On the other hand end users of high quality automated tracking pipelines no longer need a license for commercial solvers which are expensive for non-academic institutions.

- In Chapter 4 we first developed a graph decomposition to parallelize tracking that can deal with divisions and undersegmentations, and showed its potential to make analysis of huge videos possible. Then we proposed a prototypical microservices architecture for *ilastik*'s computational back end that would allow to compute segmentations through pixel classification, find connected components, extract object features, and track large amounts of data distributed on a cluster or in the cloud. Our prototype for pixel classification and thresholding performed very well, showing that this architecture can offer the much needed scaling across multiple machines, which paves the road for analysis of much larger data.

- We introduced a dynamic programming based method to generate diverse solutions for tree shaped graphical models, and presented its applicability to a wide range of computer vision problems such as object detection, panorama stitching and depth estimation.

- We developed a tracking pipeline for elongated mycobacteria using the aforementioned diverse detection hypotheses. There we employ a model that includes conflicting hypotheses and hence enforces the mutual exclusions by additional constraints. We showed that the results of this pipeline can reach a high accuracy with respect to the ground truth provided by experts. Further experiments are required to evaluate the generalization to new datasets, but the preliminary results indicate that this pipeline can significantly reduce the amount of manual effort needed to analyze tuberculosis medication studies.

- All source code is publicly available, we briefly describe the main modules in the next section.

We hope that these contributions foster the use of automated tracking methods in the life sciences and beyond.

### 7.1.1 Software Contributions

Several software modules were developed in order to produce the results found throughout this work. All modules are released as open source. The main module is the multiple *hy*potheses *tra*cking toolbox *hytra*[1], that provides the means to build the tracking model from segmented images, and also to export the tracking result into various formats. It is also used as tracking back end in *ilastik*, for which we additionally implemented an exporter to MaMuT [Wolff et al., 2017][2], which allows proof reading and correcting the tracking results. *hytra* defines a human readable JavaScript Object Notation (JSON)[3] format for the graph description and results. This makes it easy to use different solvers which consume models and export the results in this unified format. Two solvers have been developed through the course of this thesis, one for the constrained network flow ILP model presented in Section 2.3[4] – which can also perform structured learning [Joachims et al., 2009] – and another for the conditioned residual capacity successive shortest path heuristic[5] from Chapter 3. The JSON format, as well as both solvers, were designed to additionally handle exclusion constraints and could thus also be applied to the

[1] https://github.com/chaubold/hytra
[2] http://imagej.net/MaMuT
[3] http://json.org/
[4] https://github.com/chaubold/multiHypothesesTracking
[5] https://github.com/chaubold/dpct

problem in Chapter 6. Lastly, in Section 4.2 we proposed a microservices prototype to distribute the computations for pixel classification and thresholding across multiple machines[6].

## 7.2 Limitations and Outlook

The work presented in this thesis has focused on improving the runtime of the *Conservation Tracking* model for merging and dividing cells, as well as on developing a joint hypotheses selection and tracking pipeline with a method to obtain multiple detection hypotheses through dynamic programming. However, we left the expressivity of the underlying tracking model untouched. Hence there remain several open questions for future research on the model, and other ideas arise from the findings presented above.

- Even though our focus was to develop methods that are general enough to be applied to large variety of problems, biological datasets often have unique features and hence require specialized detection (as in Chapter 6) or evaluation strategies. One point we have not touched in this thesis is the **resolution of clusters** in the model presented in Chapters 2 and 3. Currently we fit a Gaussian mixture model to each cluster with the number of objects determined by the tracking result, as in [Schiegg et al., 2013]. But by far not all cells, let alone other tracked targets like animals, have an elliptical shape. For optimal performance this step should always be specialized to the scenario at hand.

- The models presented in Chapters 2, 3 and 6 can only model transitions as pairwise links between a source and a target node. This *first order motion model* does not allow for incorporation of velocity or acceleration and is applicable when the cells undergo Brownian motion, but that is not always the case. Some cells, especially bacteria, can self-propel and hence migrate along a certain path. In such cases velocity and acceleration are features that we humans use excessively to re-identify cells in consecutive frames. Integrating such a **second or third order motion model** would be desirable, but this introduces higher order factors in the graph – ranging over *e.g.* two transitions and the shared detection node for velocity – which make solving the ILP much harder. One possibility for integrating velocity without adding higher order factors is the model constructed by [Butt and Collins, 2013], which contracts two successive detections plus their transition into one node, hence assignments between those pairings consider velocity. It would be an interesting extension to their model if divisions could also be expressed in a similar way.

- In Chapter 6 we have used a classifier for transitions, but the *Conservation Tracking* model from the preceding chapters only used a distance based transition probability. We have performed some experiments with a **transition classifier** in the *Conservation Tracking* model as well. For most transitions the classifier score on validation sets was much higher than that of distance-based decisions, because it can also consider size, shape, and color changes. But as the *Conservation Tracking* model allows clusters of objects, transitions involving the merging or splitting of cell detections look completely different than single object transitions – and hence irritate the classifier. It would be beneficial to develop a more expressive model for transitions than just using distance-based probabilities that supports a varying numbers of objects in source and target detection.

[6]`https://github.com/chaubold/ilastik-backend`

- We showed in Chapter 2 that the LP relaxation of the network flow formulation of Conservation Tracking is very tight. Even more so, we know that without division and merger constraints the LP relaxation will yield the optimal integral solution. By initially solving the problem without those additional constraints, and then iteratively adding only the violated constraints in a **cutting planes** fashion, one might reach the globally optimal solution even faster.

- A lot of tracking mistakes occur when there are segmentation errors, especially **missed detections**. The common way to deal with missing detection hypotheses is to insert transitions over multiple time steps. Unfortunately, their probability can be hard to tune such that the tracking solution does not prematurely skip available detections. And if the importance of these links over multiple frames should be learned by structured learning as in Chapter 6, some instances must be annotated as training examples. But as one strives to find a segmentation with as few mistakes as possible, these can be hard to locate, making annotation impractical. Kalman filtering based approaches on the other hand propagate the state of an object into the next frame, allowing it to continue even if the local evidence is not strong. This can bridge the gap if an object becomes *e.g.* occluded or hard to find due to noise for a short time. By performing a simple Kalman filtering based tracking approach as preprocessing, one could enrich the set of available detections at locations where the segmentation is mistaken.

- Instead of relying on a segmentation to obtain detection hypotheses, we have seen in Section 6 that a specific model of the target can be required. With the revolutionary performance improvements over the last years in **object classification and detection using convolutional neural networks** (CNNs) [Krizhevsky et al., 2012, Redmon et al., 2016], real-world object tracking approaches can build on much better sets of detections. While first attempts have been made to use this for cell detection [Xie et al., 2015], there should be a lot of improvement possible through CNNs. In the tracking domain, it would be especially interesting to apply approaches that direct their attention in subsequent frames to a local neighborhood around the previously known position of the target [Ren et al., 2015].

- While the per-frame segmentation can be improved by CNNs [Ronneberger et al., 2015] and state-of-the-art recurrent neural network-based object detectors can be applied to microscopy data followed by *tracking-by-detection* methods as those presented in this work, completely **new tracking methodologies are evolving based on neural networks** as well [Alahi et al., 2016, Wang et al., 2015, Nam and Han, 2016]. It is to expect that by a clever application of neural network building blocks combined with a suitable loss function, tracking multiple *dividing* targets can be approached as well.

- Lastly, **evaluating the quality** of cell tracking solutions is a difficult task, as metrics do not necessarily capture all aspects of a solution. The Cell Tracking Challenge [Maška et al., 2014] used two scores, one for the segmentation quality (average Jaccard score), and one for the edit distance between two tracking graphs consisting of the selected assignments. In Chapter 2 and 3 we applied a similar metric for the tracking results, assuming a fixed segmentation. For Chapter 6 we additionally provide a distance measure of selected hypotheses to the ones present in the ground truth. Nevertheless none of these measures gracefully handles complex types of errors, like a division being detected a frame too early or too late. This is an error that requires changing several links and detections and hence has a big impact on the edit distance, but in terms of biological quality this is far better than missing the division completely. [Schiegg et al., 2014] used an evaluation tolerant to that, but still they had to report two numbers, one for the segmentation and one for the tracking quality. It would be desirable to design a measure that represents the overall tracking quality, maybe similar to a Jaccard score of 3D+t volumes.

To conclude, in this work we have considerably improved the scaling behavior of automated tracking methods for dividing targets. We developed a polynomial-time heuristic network flow based solver and a decomposition scheme to parallelize tracking, and distributed the preprocessing workload across multiple machines in a cluster. This, the availability of proof reading tools, and the fact that our heuristic solver is available in *ilastik* – which frees biologists from the burden of obtaining a license for a commercial ILP solver – enable more biologists around the world to to analyze the ever growing amount of data of proliferating cells in embryogenesis and drug development. While neural network based methods are revamping the world of Computer Vision and already began to improve automated tracking, they still require large amounts of training data and dedicated hardware to run efficiently. Furthermore, the images obtained by different microscopy and staining techniques, as well as of varying cell types are so diverse that it seems intractable to obtain good results with pre-trained neural networks across the broad range of recording scenarios. Yet, for only very few of the data there is enough annotated ground truth to train a neural network. As long as this is the case, more general methods will prevail, and hence the approaches developed in this thesis will remain an important contribution.

# Bibliography

[Ahuja et al., 1988] Ahuja, R. K., Magnanti, T. L., and Orlin, J. B. (1988). Network flows. Technical report, DTIC Document.

[Alahi et al., 2016] Alahi, A., Goel, K., Ramanathan, V., Robicquet, A., Fei-Fei, L., and Savarese, S. (2016). Social lstm: Human trajectory prediction in crowded spaces. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 961–971.

[Amat and Keller, 2013] Amat, F. and Keller, P. J. (2013). Towards comprehensive cell lineage reconstructions in complex organisms using light-sheet microscopy. *Development, growth & differentiation*, 55(4):563–578.

[Amat et al., 2014] Amat, F., Lemon, W., Mossing, D. P., McDole, K., Wan, Y., Branson, K., Myers, E. W., and Keller, P. J. (2014). Fast, accurate reconstruction of cell lineages from large-scale fluorescence microscopy data. *Nature methods*.

[Andriluka et al., 2008] Andriluka, M., Roth, S., and Schiele, B. (2008). People-tracking-by-detection and people-detection-by-tracking. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Andriyenko et al., 2012] Andriyenko, A., Schindler, K., and Roth, S. (2012). Discrete-continuous optimization for multi-target tracking. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 1926–1933. IEEE.

[Arteta et al., 2013] Arteta, C., Lempitsky, V., Noble, J. A., and Zisserman, A. (2013). Learning to detect partially overlapping instances. In *Proceedings of the IEEE Conference on Computer Vision And Pattern Recognition (CVPR'13)*, pages 3230–3237, Portland, OR, USA.

[Batra, 2012] Batra, D. (2012). An efficient message-passing algorithm for the m-best map problem. In *Proceedings of the 28th Conference on Uncertainty in Artificial Intelligence (UAI 2012)*.

[Batra et al., 2012] Batra, D., Yadollahpour, P., Guzman-Rivera, A., and Shakhnarovich, G. (2012). Diverse m-best solutions in markov random fields. In *Proceedings of the Twelfth European Conference on Computer Vision (ECCV'12)*, pages 1–16, Florence, Italy.

[Beier et al., 2015] Beier, T., Hamprecht, F. A., and Kappes, J. H. (2015). Fusion moves for correlation clustering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3507–3516.

[Bellman, 1952] Bellman, R. (1952). On the theory of dynamic programming. *Proceedings of the National Academy of Sciences*, 38(8):716–719.

[Berclaz et al., 2011] Berclaz, J., Fleuret, F., Türetken, E., and Fua, P. (2011). Multiple object tracking using k-shortest paths optimization. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 33(9):1806–1819.

[Berthet and Maizel, 2016] Berthet, B. and Maizel, A. (2016). Light sheet microscopy and live imaging of plants. *Journal of microscopy*, 263(2):158–164.

[Bertsekas, 1998] Bertsekas, D. P. (1998). *Network optimization: continuous and discrete models*. Athena Scientific Belmont.

[Bise et al., 2011] Bise, R., Yin, Z., and Kanade, T. (2011). Reliable Cell Tracking by Global Data Association. In *IEEE International Symposium on Biomedical Imaging (ISBI)*, pages 1004–1010.

[Breiman, 2001] Breiman, L. (2001). Random forests. *Machine learning*, 45(1):5–32.

[Brendel et al., 2011] Brendel, W., Amer, M., and Todorovic, S. (2011). Multiobject tracking as maximum weight independent set. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 1273–1280. IEEE.

[Butt and Collins, 2013] Butt, A. and Collins, R. (2013). Multi-target tracking by lagrangian relaxation to min-cost network flow. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1846–1853. IEEE.

[Carpenter et al., 2006] Carpenter, A. E., Jones, T. R., Lamprecht, M. R., Clarke, C., Kang, I. H., Friman, O., Guertin, D. A., Chang, J. H., Lindquist, R. A., Moffat, J., et al. (2006). Cellprofiler: image analysis software for identifying and quantifying cell phenotypes. *Genome biology*, 7(10):R100.

[Castanon and Finn, 2011] Castanon, G. and Finn, L. (2011). Multi-target tracklet stitching through network flows. In *IEEE Aerospace Conference*, pages 1–7. IEEE.

[Chen et al., 2013] Chen, C., Kolmogorov, V., Zhu, Y., Metaxas, D. N., and Lampert, C. H. (2013). Computing the m most probable modes of a graphical model. In *AISTATS*, pages 161–169.

[Chen et al., 2014] Chen, C., Liu, H., Metaxas, D., and Zhao, T. (2014). Mode estimation for high dimensional discrete tree graphical models. In *Advances in Neural Information Processing Systems (NIPS'14)*, pages 1323–1331, Montréal, Canada.

[Chen et al., 2006] Chen, X., Zhou, X., and Wong, S. T. (2006). Automated segmentation, classification, and tracking of cancer cell nuclei in time-lapse microscopy. *IEEE Transactions on Biomedical Engineering*, 53(4):762–766.

[Cormen, 2009] Cormen, T. H. (2009). *Introduction to algorithms*. MIT press.

[Coutu and Schroeder, 2013] Coutu, D. L. and Schroeder, T. (2013). Probing cellular processes by long-term live imaging–historic problems and current solutions. *J Cell Sci*, 126(17):3805–3815.

[Daya et al., 2016] Daya, S., Van Duy, N., Eati, K., Ferreira, C. M., Glozic, D., Gucer, V., Gupta, M., Joshi, S., Lampkin, V., Martins, M., et al. (2016). *Microservices from Theory to Practice: Creating Applications in IBM Bluemix Using the Microservices Approach*. IBM Redbooks.

[Dijkstra, 1959] Dijkstra, E. (1959). A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271.

[Dufour et al., 2011] Dufour, A., Thibeaux, R., Labruyere, E., Guillen, N., and Olivo-Marin, J.-C. (2011). 3-d active meshes: fast discrete deformable models for cell tracking in 3-d time-lapse microscopy. *IEEE Transactions on Image Processing*, 20(7):1925–1937.

[Elowitz et al., 2002] Elowitz, M. B., Levine, A. J., Siggia, E. D., and Swain, P. S. (2002). Stochastic gene expression in a single cell. *Science*, 297(5584):1183–1186.

[Eppstein, 1998] Eppstein, D. (1998). Finding the k shortest paths. *SIAM Journal on Computing*, 28(2):652–673.

[Flerova et al., 2012] Flerova, N., Rollon, E., and Dechter, R. (2012). Bucket and mini-bucket schemes for m best solutions over graphical models. In *Graph Structures for Knowledge Representation and Reasoning*, pages 91–118. Springer.

[Fromer and Globerson, 2009] Fromer, M. and Globerson, A. (2009). An lp view of the m-best map problem. In *Advances in Neural Information Processing Systems*, pages 567–575.

[Fujita et al., 2003] Fujita, Y., Nakamura, Y., and Shiller, Z. (2003). Dual Dijkstra search for paths with different topologies. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'03)*, volume 3, pages 3359–3364, Taipei, Taiwan.

[Funke et al., 2015] Funke, J., Hamprecht, F. A., and Zhang, C. (2015). Learning to segment: Training hierarchical segmentation under a topological loss. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 268–275. Springer.

[Haubold et al., 2016a] Haubold, C., Aleš, J., Wolf, S., and Hamprecht, F. A. (2016a). A generalized successive shortest paths solver for tracking dividing targets. In *European Conference on Computer Vision*, pages 566–582. Springer.

[Haubold et al., 2016b] Haubold, C., Schiegg, M., Kreshuk, A., Berg, S., Koethe, U., and Hamprecht, F. A. (2016b). Segmenting and tracking multiple dividing targets using ilastik. In *Focus on Bio-Image Informatics*, pages 199–229. Springer.

[Held et al., 2010] Held, M., Schmitz, M., Fischer, B., Walter, T., Neumann, B., Olma, M., Peter, M., Ellenberg, J., and Gerlich, D. (2010). Cellcognition: time-resolved phenotype annotation in high-throughput live cell imaging. *Nature Methods*, 7(9):747–754.

[Höckendorf et al., 2012] Höckendorf, B., Thumberger, T., and Wittbrodt, J. (2012). Quantitative analysis of embryogenesis: a perspective for light sheet microscopy. *Developmental cell*, 23(6):1111–1120.

[Jaqaman et al., 2008] Jaqaman, K., Loerke, D., Mettlen, M., Kuwata, H., Grinstein, S., Schmid, S. L., and Danuser, G. (2008). Robust single-particle tracking in live-cell time-lapse sequences. *Nature methods*, 5(8):695–702.

[Jemielita et al., 2013] Jemielita, M., Taormina, M. J., DeLaurier, A., Kimmel, C. B., and Parthasarathy, R. (2013). Comparing phototoxicity during the development of a zebrafish craniofacial bone using confocal and light sheet fluorescence microscopy techniques. *Journal of biophotonics*, 6(11-12):920–928.

[Joachims et al., 2009] Joachims, T., Finley, T., and Yu, C.-N. J. (2009). Cutting-plane training of structural svms. *Machine Learning*, 77(1):27–59.

[Jug et al., 2016] Jug, F., Levinkov, E., Blasse, C., Myers, E. W., and Andres, B. (2016). Moral lineage tracing. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5926–5935.

[Jug et al., 2014] Jug, F., Pietzsch, T., Kainmüller, D., Funke, J., Kaiser, M., van Nimwegen, E., Rother, C., and Myers, G. (2014). Optimal joint segmentation and tracking of escherichia coli in the mother machine. In *Bayesian and Graphical Models for Biomedical Imaging*, pages 25–36. Springer.

[Jüttner et al., ] Jüttner, A., Dezsö, B., and Kovács, P. Lemon: Library for efficient modeling and optimization in networks. Technical report, Dept. of Operations Research, Eötvös Loránd University, Budapest.

[Kausler et al., 2012] Kausler, B. X., Schiegg, M., Andres, B., Lindner, M., Koethe, U., Leitte, H., Wittbrodt, J., Hufnagel, L., and Hamprecht, F. A. (2012). A discrete chain graph model for 3d+ t cell tracking with high misdetection robustness. In *European Conference on Computer Vision (ECCV)*, pages 144–157. Springer.

[Keller, 2013] Keller, P. J. (2013). Imaging morphogenesis: technological advances and biological insights. *Science*, 340(6137):1234168.

[Keller et al., 2008] Keller, P. J., Schmidt, A. D., Wittbrodt, J., and Stelzer, E. H. (2008). Reconstruction of zebrafish early embryonic development by scanned light sheet microscopy. *science*, 322(5904):1065–1069.

[Kirillov et al., 2015] Kirillov, A., Savchynskyy, B., Schlesinger, D., Vetrov, D., and Rother, C. (2015). Inferring m-best diverse labelings in a single one. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV'15)*, pages 1814–1822, Santiago, Chile.

[Koller and Friedman, 2009] Koller, D. and Friedman, N. (2009). *Probabilistic graphical models: principles and techniques*. MIT Press, Cambridge, MA, USA.

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

[Krzic et al., 2012] Krzic, U., Gunther, S., Saunders, T. E., Streichan, S. J., and Hufnagel, L. (2012). Multiview light-sheet microscope for rapid in toto imaging. *Nature methods*, 9(7):730–733.

[Kschischang et al., 2001] Kschischang, F. R., Frey, B. J., and Loeliger, H.-A. (2001). Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 47(2):498–519.

[Kuhn, 1955] Kuhn, H. W. (1955). The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97.

[Lampert, 2011] Lampert, C. H. (2011). Maximum margin multi-label structured prediction. In *Advances in Neural Information Processing Systems*, pages 289–297.

[Lawler, 1972] Lawler, E. (1972). A procedure for computing the k best solutions to discrete optimization problems and its application to the shortest path problem. *Management science*, 18(7):401–405.

[Lenz et al., 2015] Lenz, P., Geiger, A., and Urtasun, R. (2015). Followme: Efficient online min-cost flow tracking with bounded memory and computation. In *IEEE International Conference on Computer Vision (ICCV)*, pages 4364–4372.

[Li et al., 2010] Li, X., Wang, K., Wang, W., and Li, Y. (2010). A multiple object tracking method using kalman filter. In *Information and Automation (ICIA), 2010 IEEE International Conference on*, pages 1862–1866. IEEE.

[Lou and Hamprecht, 2011] Lou, X. and Hamprecht, F. A. (2011). Structured learning for cell tracking. In *Advances in neural information processing systems*, pages 1296–1304.

[Magnusson et al., 2014] Magnusson, K., Jalden, J., Gilbert, P., and Blau, H. (2014). Global linking of cell tracks using the viterbi algorithm. *Transactions on Medical Imaging*, 34(4):911 – 929.

[Magnusson and Jaldén, 2012] Magnusson, K. E. and Jaldén, J. (2012). A batch algorithm using iterative application of the viterbi algorithm to track cells and construct cell lineages. In *International Symposium on Biomedical Imaging (ISBI)*, pages 382–385. IEEE.

[Maška et al., 2013] Maška, M., Daněk, O., Garasa, S., Rouzaut, A., Muñoz-Barrutia, A., and Ortiz-de Solorzano, C. (2013). Segmentation and shape tracking of whole fluorescent cells based on the chan–vese model. *IEEE transactions on medical imaging*, 32(6):995–1006.

[Maška et al., 2014] Maška, M., Ulman, V., Svoboda, D., Matula, P., Matula, P., Ederra, C., Urbiola, A., España, T., Venkatesan, S., Balak, D. M., et al. (2014). A benchmark for comparison of cell tracking algorithms. *Bioinformatics*, 30(11):1609–1617.

[McKinney, 2000] McKinney, J. D. (2000). In vivo veritas: the search for tb drug targets goes live. *Nature medicine*, 6(12):1330–1333.

[Mekterović et al., 2014] Mekterović, I., Mekterović, D., et al. (2014). Bactimas: a platform for processing and analysis of bacterial time-lapse microscopy movies. *BMC bioinformatics*, 15(1):251.

[Milan et al., 2013] Milan, A., Schindler, K., and Roth, S. (2013). Detection-and trajectory-level exclusion in multiple object tracking. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3682–3689.

[Nam and Han, 2016] Nam, H. and Han, B. (2016). Learning multi-domain convolutional neural networks for visual tracking. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4293–4302.

[Nilsson, 1998] Nilsson, D. (1998). An efficient algorithm for finding the m most probable configurationsin probabilistic expert systems. *Statistics and Computing*, 8(2):159–173.

[Padfield et al., 2011] Padfield, D., Rittscher, J., and Roysam, B. (2011). Coupled minimum-cost flow cell tracking for high-throughput quantitative analysis. *Medical image analysis*, 15(4):650–668.

[Pearl, 1988] Pearl, J. (1988). *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann.

[Pirsiavash et al., 2011] Pirsiavash, H., Ramanan, D., and Fowlkes, C. C. (2011). Globally-optimal greedy algorithms for tracking a variable number of objects. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1201–1208. IEEE.

[Pishchulin et al., 2016] Pishchulin, L., Insafutdinov, E., Tang, S., Andres, B., Andriluka, M., Gehler, P. V., and Schiele, B. (2016). Deepcut: Joint subset partition and labeling for multi person pose estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4929–4937.

[Prasad et al., 2014] Prasad, A., Jegelka, S., and Batra, D. (2014). Submodular meets structured: Finding diverse subsets in exponentially-large structured item sets. In *Advances in Neural Information Processing Systems (NIPS'14)*, pages 2645–2653, Montréal, Canada.

[Rapoport et al., 2011] Rapoport, D. H., Becker, T., Mamlouk, A. M., Schicktanz, S., and Kruse, C. (2011). A novel validation algorithm allows for automated cell tracking and the extraction of biologically meaningful parameters. *PloS one*, 6(11):e27315.

[Raviglione and Sulis, 2016] Raviglione, M. and Sulis, G. (2016). Tuberculosis 2015: burden, challenges and strategy for control and elimination. *Infectious Disease Reports*, 8(2).

[Redmon et al., 2016] Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 779–788.

[Ren et al., 2015] Ren, S., He, K., Girshick, R., and Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99.

[Richardson, 2015] Richardson, C. (2015). Introduction to microservices. *URL: https://www.nginx.com/blog/introduction-to-microservices*.

[Rollon et al., 2011] Rollon, E., Flerova, N., and Dechter, R. (2011). Inference schemes for m best solutions for soft csps. In *Proceedings of the Seventh International Workshop on Preferences and Soft Constraints*, volume 2, Sitges, Spain.

[Ronneberger et al., 2015] Ronneberger, O., Fischer, P., and Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 234–241. Springer.

[Sadanandan et al., 2016] Sadanandan, S. K., Baltekin, Ö., Magnusson, K. E., Boucharin, A., Ranefall, P., Jaldén, J., Elf, J., and Wählby, C. (2016). Segmentation and track-analysis in time-lapse imaging of bacteria. *IEEE Journal of Selected Topics in Signal Processing*, 10(1):174–184.

[Santi et al., 2013] Santi, I., Dhar, N., Bousbaine, D., Wakamoto, Y., and McKinney, J. D. (2013). Single-cell dynamics of the chromosome replication and cell division cycles in mycobacteria. *Nature communications*, 4.

[Scharstein et al., 2014] Scharstein, D., Hirschmüller, H., Kitajima, Y., Krathwohl, G., Nešić, N., Wang, X., and Westling, P. (2014). High-resolution stereo datasets with subpixel-accurate ground truth. In *German Conference on Pattern Recognition*, pages 31–42. Springer.

[Schiegg et al., 2014] Schiegg, M., Hanslovsky, P., Haubold, C., Koethe, U., Hufnagel, L., and Hamprecht, F. A. (2014). Graphical model for joint segmentation and tracking of multiple dividing cells. *Bioinformatics*, page btu764.

[Schiegg et al., 2013] Schiegg, M., Hanslovsky, P., Kausler, B. X., Hufnagel, L., and Hamprecht, F. A. (2013). Conservation tracking. In *IEEE International Conference on Computer Vision (ICCV)*, pages 2928–2935. IEEE.

[Schindelin et al., 2012] Schindelin, J., Arganda-Carreras, I., Frise, E., Kaynig, V., Longair, M., Pietzsch, T., Preibisch, S., Rueden, C., Saalfeld, S., Schmid, B., et al. (2012). Fiji: an open-source platform for biological-image analysis. *Nature methods*, 9(7):676–682.

[Schlesinger and Hlavác, 2013] Schlesinger, M. I. and Hlavác, V. (2013). *Ten lectures on statistical and structural pattern recognition*, volume 24. Springer Science & Business Media.

[Schroeder, 2011] Schroeder, T. (2011). Long-term single-cell imaging of mammalian stem cells. *Nature methods*, 8(4s):S30–S35.

[Seroussi and Golmard, 1994] Seroussi, B. and Golmard, J.-L. (1994). An algorithm directly finding the k most probable configurations in bayesian networks. *International Journal of Approximate Reasoning*, 11(3):205–233.

[Sliusarenko et al., 2011] Sliusarenko, O., Heinritz, J., Emonet, T., and Jacobs-Wagner, C. (2011). High-throughput, subpixel precision analysis of bacterial morphogenesis and intracellular spatio-temporal dynamics. *Molecular microbiology*, 80(3):612–627.

[Sommer et al., 2011] Sommer, C., Straehle, C., Kothe, U., and Hamprecht, F. A. (2011). Ilastik: Interactive learning and segmentation toolkit. In *IEEE International Symposium on Biomedical Imaging: From Nano to Macro (ISBI)*, pages 230–233. IEEE.

[Summa et al., 2012] Summa, B., Tierny, J., and Pascucci, V. (2012). Panorama weaving: fast and flexible seam processing. *ACM Transactions on Graphics*, 31(4):83:1–83:11.

[Teo et al., 2010] Teo, C. H., Vishwanthan, S., Smola, A. J., and Le, Q. V. (2010). Bundle methods for regularized risk minimization. *Journal of Machine Learning Research*, 11(Jan):311–365.

[Tsochantaridis et al., 2004] Tsochantaridis, I., Hofmann, T., Joachims, T., and Altun, Y. (2004). Support vector machine learning for interdependent and structured output spaces. In *Proceedings of the twenty-first international conference on Machine learning*, page 104. ACM.

[Tu, 2008] Tu, Z. (2008). Auto-context and its application to high-level vision tasks. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE.

[Turetken et al., 2016] Turetken, E., Wang, X., Becker, C. J., Haubold, C., and Fua, P. (2016). Network flow integer programming to track elliptical cells in time-lapse sequences. *IEEE Transactions on Medical Imaging*.

[Uhlmann et al., 2014] Uhlmann, V., Delgado-Gonzalo, R., and Unser, M. (2014). Snakes with tangent-based control and energies for bioimage analysis. In *Biomedical Imaging (ISBI), 2014 IEEE 11th International Symposium on*, pages 806–809. IEEE.

[Veksler, 2005] Veksler, O. (2005). Stereo correspondence by dynamic programming on a tree. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 2, pages 384–390, San Diego, CA, USA.

[Villamizar et al., 2015] Villamizar, M., Garcés, O., Castro, H., Verano, M., Salamanca, L., Casallas, R., and Gil, S. (2015). Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud. In *Computing Colombian Conference (10CCC), 2015 10th*, pages 583–590. IEEE.

[Wakamoto et al., 2013] Wakamoto, Y., Dhar, N., Chait, R., Schneider, K., Signorino-Gelo, F., Leibler, S., and McKinney, J. D. (2013). Dynamic persistence of antibiotic-stressed mycobacteria. *Science*, 339(6115):91–95.

[Wang et al., 2015] Wang, L., Ouyang, W., Wang, X., and Lu, H. (2015). Visual tracking with fully convolutional networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3119–3127.

[Wang et al., 2014] Wang, X., Türetken, E., Fleuret, F., and Fua, P. (2014). Tracking interacting objects optimally using integer programming. In *European Conference on Computer Vision (ECCV)*, pages 17–32. Springer.

[Wolff et al., 2017] Wolff, C., Tinevez, J.-Y., Pietzsch, T., Stamataki, E., Harich, B., Preibisch, S., Shorte, S., Keller, P. J., Tomancak, P., and Pavlopoulos, A. (2017). Reconstruction of cell lineages and behaviors underlying arthropod limb outgrowth with multi-view light-sheet imaging and tracking. *bioRxiv*, page 112623.

[Wu et al., 2010] Wu, Q., Merchant, F., and Castleman, K. (2010). *Microscope image processing*. Academic press.

[Xie et al., 2015] Xie, W., Noble, J. A., and Zisserman, A. (2015). Microscopy cell counting with fully convolutional regression networks. In *MICCAI 1st Workshop on Deep Learning in Medical Image Analysis*.

[Xing et al., 2009] Xing, J., Ai, H., and Lao, S. (2009). Multi-object tracking through occlusions by local tracklets filtering and global tracklets association with detection responses. *CVPR 2013*.

[Yadollahpour et al., 2013] Yadollahpour, P., Batra, D., and Shakhnarovich, G. (2013). Discriminative re-ranking of diverse segmentations. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Yanover and Weiss, 2004] Yanover, C. and Weiss, Y. (2004). Finding the m most probable configurations using loopy belief propagation. *Advances in neural information processing systems*, 16:289.

[Young et al., 2012] Young, J. W., Locke, J. C., Altinok, A., Rosenfeld, N., Bacarian, T., Swain, P. S., Mjolsness, E., and Elowitz, M. B. (2012). Measuring single-cell gene expression dynamics in bacteria using fluorescence time-lapse microscopy. *Nature protocols*, 7(1):80–88.

[Zhang et al., 2008] Zhang, L., Li, Y., and Nevatia, R. (2008). Global data association for multi-object tracking using network flows. In *IEEE Conference on Computer Vision and Pattern Recognition, (CVPR)*, pages 1–8. IEEE.

[Zimmer et al., 2002] Zimmer, C., Labruyere, E., Meas-Yedid, V., Guillen, N., and Olivo-Marin, J.-C. (2002). Segmentation and tracking of migrating cells in videomicroscopy with parametric active contours: A tool for cell-based drug testing. *IEEE transactions on medical imaging*, 21(10):1212–1221.