3D Image Segmentation with Explicit Surface-Based Priors

Jensen, Patrick Møller

Publication date:
2022

Document Version
Publisher's PDF, also known as Version of record

DTU

3D Image Segmentation with Explicit Surface-Based Priors

Patrick M. Jensen

Visual Computing,
Applied Mathematics and Computer Science,
Technical Univesity of Denmark,
Kongens Lyngby, Denmark

Supervisors:
Vedrana A. Dahl
Anders B. Dahl

PhD Thesis

Abstract

In the past decades, the amount and quality of 3D image data being produced have increased tremendously. This has brought the need for automated tools to analyze these images. In some domains, models based on deep learning have become dominant. However, there are still areas where the application of such models is challenging due to the lack of annotated data and the sheer size of 3D images.

This thesis explores the use of explicit surfaces in image segmentation to address this issue. Explicit surfaces in the form of meshes have two concrete benefits: First, they easily allow the incorporation of prior knowledge on shape and topology. This helps alleviate the issues of no annotated data for machine learning models. Second, they move the computational scaling to the size of the mesh instead of the size of the image. This allows mesh-based models to scale to very large segmentation problems, as this thesis will show.

The first part of the thesis focuses on fitting meshes to images with minimum cut/maximum flow (min-cut/max-flow). We review the current state of the art for serial and parallel min-cut/max-flow algorithms, develop a new min-cut/max-flow-based parallel algorithm for quadratic pseudo-Boolean optimization (QPBO), and a new method for multi-object segmentation with exclusion constraints based on QPBO. We demonstrate that mesh fitting with min-cut/max-flow can be scaled to large segmentation problems with thousands of interacting objects. Furthermore, for large problems, their solution can be parallelized efficiently, although we show parallel min-cut/max-flow algorithms struggle with speeding up smaller problems.

Next, we contribute a method for segmentation of 4D (3D + time) images, which focuses on objects that either split or merge over time. We demonstrate how a simple topology constraint leads to a highly efficient method that fully captures object dynamics.

In the last part of the thesis, we investigate the use of explicit surfaces in learning-based segmentation. Using surface fitting, we develop a method that allows the training of neural network segmentation models from only sparse point annotations. We show that this allows training more accurate models from significantly less annotation effort. Finally, we develop a new mesh-based shape model which can learn a latent shape space and then smoothly combine multiple latent representations. We demonstrate that this makes for a highly flexible model which can easily be used for a variety of tasks — for example, efficient image annotation and segmentation post-processing.

In conclusion, the contributions in this thesis show meshes to be a powerful tool for image segmentation from both a modeling and computational standpoint. Therefore, meshes have great potential to form the basis of accurate and scalable segmentation models to handle the ever-growing amount of 3D image data.

Resumé

I de seneste årtier er mængden og kvaliteten af 3D-billeddata, der produceres, steget enormt. Dette har medført et behov for automatiserede værktøjer til at analysere disse billeder. I nogle domæner er modeller baseret på dyb læring blevet dominerende, men der er stadig områder, hvor anvendelsen af sådanne modeller er udfordrende på grund af manglen på annoteret data og den enorme størrelse af 3D-billeder.

Denne afhandling udforsker brugen af eksplicitte overflader i billedsegmentering for at løse dette problem. Eksplicitte overflader i form af trekantsnet, har to konkrete fordele: For det første tillader de let at indarbejde forhåndsviden om form og topologi. Dette hjælper på problemet med manglen på annoteret data til maskinlæringsmodeller. For det andet flytter de den beregningsmæssige skalering til størrelsen af trekantsnettet i stedet for størrelsen af billedet. Dette tillader modeller baseret på trekantsnet at skalere til meget store segmenteringsproblemer, som denne afhandling vil vise.

Den første del af afhandlingen fokuserer på at tilpasse trekantsnet til billeder med grafoptimering. Vi evaluerer nuværende serielle og parallelle algoritmer til grafoptimering, udvikler en ny parallel algoritme til kvadratisk pseudo-Boolesk optimering (QPBO) baseret på grafoptimering og præsenterer en ny metode til multi-objekt segmentering med seperationsbegrænsninger baseret på QPBO. Vi viser, at tilpasning med trekantsnet baseret på grafoptimering kan skaleres til store segmenteringsproblemer med tusindvis af interagerende objekter. For store problemer kan løsningen af disse desuden paralleliseres effektivt. Vi viser dog, at parallelle grafoptimeringsalgoritmer har problemer med at løse mindre problemer hurtigere end serielle algoritmer.

Som det næste bidrager vi med en metode til segmentering af 4D (3D + tid) billeder, som fokuserer på objekter, der enten deler sig eller smelter sammen over tid. Vi viser, hvordan en simpel topologibegrænsning fører til en yderst effektiv metode, der også fuldt beskriver objektudviklingen.

I den sidste del af afhandlingen undersøger vi brugen af eksplicitte overflader i læringsbaseret segmentering. Ved hjælp af overfladetilpasning udvikler vi en metode, der tillader træning af segmenteringsmodeller baseret på neurale netværk fra sparsomme punktannoteringer. Vi viser, at dette giver mulighed for at træne mere nøjagtige modeller fra en væsentlig mindre annoteringsindsats. Endeligt udvikler vi en ny formmodel baseret på trekantsnet, som kan lære et latent formrum og derefter kontinuert kombinere flere latente repræsentationer. Vi demonstrerer, at dette resulterer i en enormt fleksibel model, som nemt kan bruges til en række forskellige opgaver — for eksempel effektiv billedannotering og efterbehandling af segmenteringer.

Alt i alt viser bidragene i denne afhandling, at trekantsnet er et stærkt værktøj til billedsegmentering fra både et modellerings- og beregningsmæssigt synspunkt. Som følge deraf, har trekantsnet derfor et stort potentiale for at danne grundlag for velfungerende og skalerbare segmenteringsmodeller til at håndtere den stadigt voksende mængde af 3D-billeddata.

Preface

This thesis was prepared at the section for Visual Computing (formerly Image Analysis and Computer Graphics) at the Department of Applied Mathematics and Computer Science (DTU Compute) at the Technical University of Denmark (DTU). The project was funded by a PhD scholarship from DTU Compute. It was completed in the period of 01-08-2019 to 31-10-2022. The project was supervised by Associate Professor Vedrana Andersen Dahl and co-supervised by Researcher Camilla Himmelstrup Trinderup from 01-08-2019 to 29-01-2020 and Professor Anders Bjorholm Dahl from 30-01-2020 to 31-10-2022.

The PhD included two external stays. The first was in Germany from 21-09-2020 to 13-11-2020, which consisted of two weeks at DESY, Hamburg hosted by Dr. Michael Sprung, and five weeks at IRP, Georg-August-Universität Göttingen hosted by Professor Dr. Tim Salditt. This stay was funded by the Hanseatic League of Science (HALOS). The second stay was in Switzerland from 21-09-2021 to 17-12-2021 at CVLAB, École Polytechnique Fédérale de Lausanne (EPFL) hosted by Professor Pascal Fua.

Patrick Møller Jensen,
Kongens Lyngby, 31-10-2022.

Acknowledgements

First of all, I want to thank my supervisors, Vedrana A. Dahl and Anders B. Dahl, for their amazing guidance and support both during and before my PhD. I am immensely grateful for all the encouragement, opportunities, and freedom you have provided during the last few years. Your near-infinite capacity for research work is inspirational. I also wish to thank Camilla H. Trinderup for her supportive and encouraging supervision from my bachelor's project all the way to the early parts of my PhD.

My thanks also go out to my amazing colleagues at the Visual Computing section at DTU, that have made my time in the office so enjoyable. In particular, I want to thank Niels Jeppesen for many interesting hours of discussion and collaboration. The high standards you set for your work are something to strive for. Also, I wish to thank J. Andreas Bærentzen for all the ideas and input he had for the paper on 4D segmentation. It taught me a lot about both geometry and writing. Thank you also to my former colleagues at ROCKWOOL, Lucie Chapelle and Dorthe Lybye, who gave me my first glimpse of what research looks like and a great environment to experience it in.

I also want to thank Tim Salditt, Michael Sprung, Marius Reichardt, and the rest of the group at IRP in Göttingen for hosting me. Seeing all the work that goes into creating the kind of data I spent my PhD analyzing was both great fun and very humbling. Additionally, I owe many thanks to Pascal Fua, Udaranga Wickramasinghe, and the CVLAB group for hosting me in Lausanne. They were all extremely welcoming and it was very inspiring to experience computer vision research at such a high level.

I am also immensely grateful to my girlfriend Vivian for all of her love and support, as well as for proofreading my entire thesis. Doing a PhD is not always easy, but without you, it would have been much harder. Finally, my deepest gratitude goes to my parents. You were always encouraging. Your love and support always felt boundless and unconditional. I am where I am today because of you.

Contents

1 Introduction

3D images (also known as volumetric images or volumes) provide a unique opportunity to investigate the spatial structure of matter. They allow us to understand phenomena such as how blood vessels are organized in tissue (Figure 1.1a), the physics of foam (Figure 1.1b), the morphology of insects (Figure 1.1c), and the size and shape of organs (Figure 1.1d).

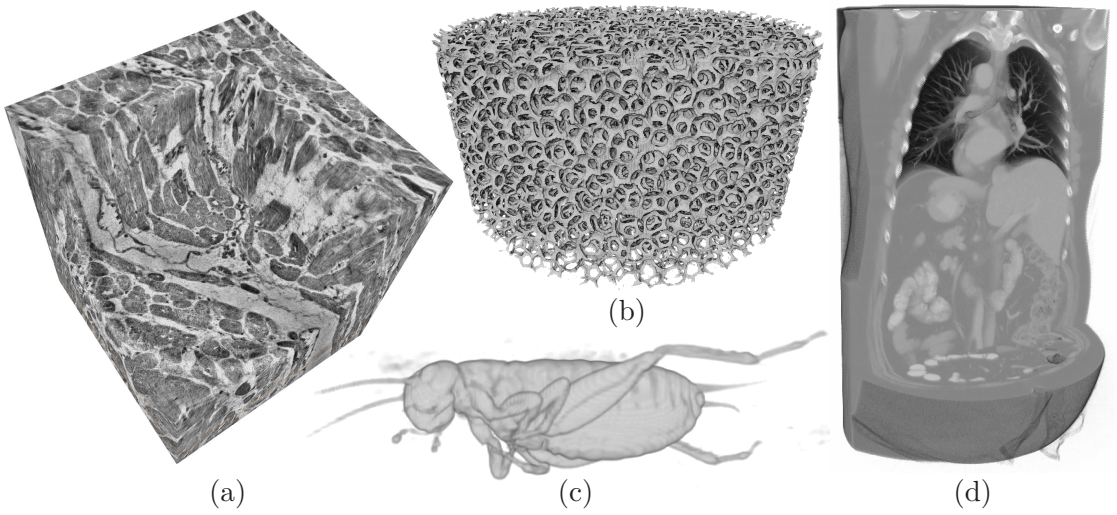
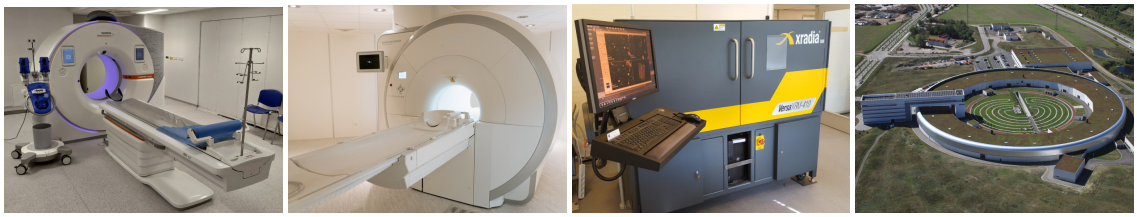


Figure 1.1: Examples of 3D image data. (a) Phase-contrast X-ray CT image of heart tissue from [1]. (b) X-ray CT image of liquid foam from [2]. The image has been thresholded. (c) X-ray CT image of a cricket. The image has been thresholded. (d) Clinical contrast-enhanced X-ray CT from [3].

Due to this power, 3D images have also become increasingly prevalent over the last decade. The best known sources of 3D images are likely clinical X-ray computed tomography (CT) scanners (Figure 1.2a) or magnetic resonance imaging (MRI) machines (Figure 1.2b) — and the number of such machines is steadily increasing worldwide [4–6]. For industrial and research use, laboratory X-ray CT setups (Figure 1.2c) are also becoming common and several companies now sell 'plug and play' scanners aimed at non-experts. At the other end of the scale are synchrotron light sources (Figure 1.2d), which are huge facilities that can generate very powerful X-ray light for CT (or related techniques) [7]. These facilities are mainly used for research (industrial and academic) and the number of publications making use of synchrotron light has been on an upward trend for the past 20 years.[1]

In addition to an increase in the quantity of 3D images, their quality has increased. Spatial resolution keeps improving [7, 10, 11], which means that structures smaller than 1 µm can now be resolved. Simultaneously, imaging times decrease which enables scanning large numbers of specimens or getting a larger field of view by stitching many images together. The shorter scanning times also enable 3D movies (3D + time or 4D images) which opens up possibilities for studying the dynamics of a sample [7, 12]. Indeed, modern synchrotron facilities can now produce up to 1000 full 3D images per second while having spatial

[1] Determined by searching Scopus and Web of Science for publications containing the words "synchrotron" and ("ct" or "tomography") in their title, abstract, or keywords.



(a) (b) (c) (d)

Figure 1.2: Example sources of 3D images. (a) Clinical X-ray CT scanner [8]. (b) Clinical MRI machine. (c) Laboratory X-ray CT scanner. (d) The MAX-IV synchrotron in Lund, Sweden [9].

resolutions in the micrometer range [13]. As a result of these developments, image data sets can now be very large with 3D images using up to 50 gigabytes (GBs) per image while a single 4D image can use multiple terabytes (TBs) [7].

Because of this growth, there is already a strong need for automated analysis of 3D images, and this need is only going to increase with time. While analysis may encompass many things, I will focus on segmentation which is one of the most common tasks in image analysis. In medical imaging, algorithms based on deep learning have become dominant [14, 15]. A large contributor to this success is the abundance of open datasets available [3, 16–21]. Nevertheless, there are still domains where these datasets remain insufficient and techniques for dealing with imperfect data are still important and actively researched [22, 23]. In materials science and pre-clinical research, machine learning is still only emerging [7, 24, 25] and researchers still rely on fairly primitive image processing tools [7, 12, 26]. However, these fields also lack standardized annotated 3D image datasets for training deep learning models. While some preliminary efforts exist [27–30], these datasets are still much smaller than what is available for clinical medical images.

Another obstacle to deep learning in materials science and pre-clinical research lies in the nature of the data. Typically, a clinical dataset will consist of many 3D images where each image consists of less than 512^3 voxels. This makes them ideal for deep learning training pipelines, as a model will see a large variety of samples during training and only a small amount of downsampling and/or cropping is needed to fit in current graphical processing unit (GPU) memory limits. In contrast, materials science and pre-clinical datasets may consist of around 10 images where each image contains 2000^3 voxels or more. Here, a model must learn from only a few samples and they must be heavily downsampled and/or cropped — which essentially negates the advantages of modern 3D imaging systems. As a result, there is currently a disconnect between the needs of researchers working with such datasets and modern (learning-based) image segmentation tools. It is this disconnect I will seek to address in this thesis.

1.1 Thesis Focus

Concretely, in this thesis I focus on the problem of how to perform segmentation of 3D images with the following properties:

P1 They are large (1000^3 voxels and beyond). This places demands on the computational and memory efficiency of methods. Methods that work well enough on small images may become highly impractical, or even infeasible, to use on large images.

P2 No labeled image data exists. This means that models must either work without

a prior learning step or learn very efficiently from limited labels as these will have to be provided from scratch.

P3 We have prior knowledge about object shapes. This knowledge can be low-level information, such as knowledge of object topology or the assumption of a smooth boundary, or higher level, such as a statistical shape model.

It is worth elaborating on P2 and P3 as they may seem contradictory. How can we create shape models if we do not have access to labeled data? This is possible by making a distinction between object *appearance* and *shape*. P2 means that we cannot (easily) learn the *appearance* of an object in the images to segment. However, we may learn the *shape* based on information from other modalities, which is what motivates P3. This is important because 3D images come from many different modalities that may create very different-looking images, even when imaging the same object.

To address the above-mentioned problem, I have focused on methods for fitting surface meshes to images. Meshes are attractive as they open many avenues for incorporating prior shape information. Furthermore, by developing methods that scale with the size of the mesh instead of the size of the image we can potentially tackle segmentation of very large images. Finally, meshes are also very suitable — and sometimes necessary — for downstream analysis tasks such as volume measurement, shape analysis, finite element modeling, etc.

1.2 Thesis Structure

I now give an overview of the thesis and how each chapter addresses the three problem properties stated above. The majority of the thesis is a collection of papers completed during my PhD. At the beginning of Chapters 3 and 4 I provide additional information on each contribution.

- In Chapter 2, I provide background information and describe related work in order to position the work of this thesis in the broader scope of image segmentation.
- In Chapter 3, I focus on P1 and P2 to investigate new segmentation methods that do not make use of learned shape information but instead rely on low-level smoothness priors. The main focus was computational and memory efficiency of segmentation with minimum cut/maximum flow (min-cut/max-flow) algorithms which we reviewed (Paper A) and improved (Paper B). We also developed a new method that can segment multiple objects in large 3D images (Paper C) using min-cut/max-flow. Finally, we developed a new deformable surface method for segmenting objects with changing topology in large 4D images (Paper D).
- In Chapter 4, I focus on P2 and P3 to incorporate shape knowledge into learned segmentation models. We use smoothness-based shape fitting in order to greatly reduce the annotation effort required to train a deep learning-based segmentation model (Paper E). Expanding on that, we then developed a new learnable mesh-based shape model (Paper F) that could further reduce the required annotation effort and also improve existing segmentations from other models.
- In Chapters 5 and 6, I give concluding remarks on the work in this thesis and provide perspectives on future work and developments.

1.3 Project Evolution

The scope of this project was initially very wide and finding the focus of my PhD was a large part of the first year. I began by exploring min-cut/max-flow methods for multi-GB

3D image segmentation, which lead to Paper C. While working on this, it became clear that serial min-cut/max-flow algorithms were a performance bottleneck. This led to a collaboration with Niels Jeppesen, who had observed the same, and we set out to explore what had been done with min-cut/max-flow, especially on the parallel front. The result of this investigation formed the basis for Paper A. We also set out to develop a new parallel method to further speed up our segmentation methods, which resulted in Paper B. Early in the project I also tried adapting the min-cut/max-flow surface fitting to 4D. However, it became clear that a deformable surface approach would be better, which lead to Paper D.

In parallel to this, I became interested in methods from the emerging field of geometric deep learning. The hope was to develop a learned shape prior to further improve image segmentation. We noticed that Udaranga Wickramasinghe from Pascal Fua's lab was also doing work in this direction and initialized a collaboration which led to me participating in Paper E. We continued the collaboration to create a more powerful shape model which would (potentially) improve Paper E and allow me to create the segmentation model I initially wanted. This led to Paper F, which was about that shape model. With the PhD coming to an end, I didn't have enough time to build on Paper F to create my segmentation algorithm — another paper for another time.

During the project, I also had forays into other projects which are not included in this thesis. In the first year, I briefly worked on foam quantification, which never crystallized into a paper. I also spent some time working on fast 3D mathematical morphology, which expanded on a paper[2] I wrote during my master's. While we improved the theoretical framework, it did not result in a new paper during my PhD. Finally, I also spent time working with physicists from Göttingen on segmenting blood vessels in large 3D images using deep learning. This led to a paper[3] and I learned a lot about how the image data I work with was captured. Although not included in the thesis, this was the catalyst for shifting my work to deep learning.

Non-Included Contributions

In this section, I list all contributions I have participated in, but which are not included in this thesis due to being out of scope or my contribution being too minor.

[i] Patrick M. Jensen, Camilla H. Trinderup, Anders B. Dahl, and Vedrana A. Dahl. "Zonohedral approximation of spherical structuring element for volumetric morphology". In: Scandinavian Conference on Image Analysis. Springer. 2019, pp. 128–139.

[ii] Matthias B. Stuart, Patrick M. Jensen, Julian T. R. Olsen, Alexander B. Kristensen, Mikkel Schou, Bernd Dammann, Hans Henrik B. Sørensen, Jørgen A. Jensen. "Fast GPU-beamforming of row-column addressed probe data". In: IEEE International Ultrasonics Symposium (IUS). 2019, pp. 1497–1500.

[iii] Matthias B. Stuart, Patrick M. Jensen, Julian T. R. Olsen, Alexander B. Kristensen, Mikkel Schou, Bernd Dammann, Hans Henrik B. Sørensen, Jørgen A. Jensen. "Real-time volumetric synthetic aperture software beamforming of row–column probe data". In: IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control 68.8 (2021), pp. 2608–2618.

[iv] Marius Reichardt, Patrick M. Jensen, Vedrana A. Dahl, Anders B. Dahl, Maximilian Ackermann, Harshit Shah, Florian Länger, Christopher Werlein, Mark P. Kuehnel, Danny Jonigk, Tim Salditt. "3D virtual histopathology of cardiac tissue from Covid-19 patients based on phase-contrast X-ray tomography". In: eLife 10 (2021), e71359.

[2] Not included in this thesis. See paper [i] in the section Non-Included Contributions.
[3] Not included in this thesis. See paper [iv] in the section Non-Included Contributions.

[v] Jakob Christensen, Patrick M. Jensen, Morten R. Hannemose, Anders B. Dahl, Vedrana A. Dahl. "LayeredCNN: Segmenting Layers with Autoregressive Models". In: Proceedings of the Northern Lights Deep Learning Workshop. 2022.

References

[1] Marius Reichardt, Patrick Moller Jensen, Vedrana Andersen Dahl, Anders Bjorholm Dahl, Maximilian Ackermann, Harshit Shah, Florian Länger, Christopher Werlein, Mark P Kuehnel, Danny Jonigk, et al. "3D virtual histopathology of cardiac tissue from Covid-19 patients based on phase-contrast X-ray tomography". In: *eLife* 10 (2021), e71359.

[2] Christophe Raufaste, Benjamin Dollet, Kevin Mader, Stéphane Santucci, and Rajmund Mokso. "Three-dimensional foam flow resolved by fast X-ray tomographic microscopy". In: *EPL (Europhysics Letters)* 111.3 (2015), p. 38004.

[3] Amber L Simpson, Michela Antonelli, Spyridon Bakas, Michel Bilello, Keyvan Farahani, Bram Van Ginneken, Annette Kopp-Schneider, Bennett A Landman, Geert Litjens, Bjoern Menze, et al. "A large annotated medical image dataset for the development and evaluation of segmentation algorithms". In: *arXiv preprint arXiv:1902.09063* (2019).

[4] Eurostat. *Medical technology*. Accessed 06-07-2022. 2021. URL: https://ec.europa.eu/eurostat/databrowser/bookmark/6c838913-f2d4-4c54-b3b8-d8af9e487989?lang=en.

[5] OECD. *Magnetic resonance imaging (MRI) units (indicator)*. Accessed 07-07-2022. 2022. DOI: 10.1787/1a72e7d1-en.

[6] OECD. *Computed tomography (CT) scanners (indicator)*. Accessed 07-07-2022. 2022. DOI: 10.1787/bedece12-en.

[7] Philip J Withers, Charles Bouman, Simone Carmignato, Veerle Cnudde, David Grimaldi, Charlotte K Hagen, Eric Maire, Marena Manley, Anton Du Plessis, and Stuart R Stock. "X-ray computed tomography". In: *Nature Reviews Methods Primers* 1.1 (2021), pp. 1–21.

[8] Image by Tomáš Vendiš. Image has been cropped from original. Published under CC-BY-SA (https://creativecommons.org/licenses/by-sa/4.0/). URL: https://commons.wikimedia.org/wiki/File:Modern%C3%AD_v%C3%BDpo%C4%8Detn%C3%AD_tomografie_s_p%C5%99%C3%ADmo_digit%C3%A1ln%C3%AD_detekc%C3%AD_rentgenov%C3%A9ho_z%C3%A1%C5%99en%C3%AD.jpg.

[9] Image by Perry Nordeng. Image has been cropped from original.

[10] Giuseppe Barisano, Farshid Sepehrband, Samantha Ma, Kay Jann, Ryan Cabeen, Danny J Wang, Arthur W Toga, and Meng Law. "Clinical 7 T MRI: Are we there yet? A review about magnetic resonance imaging at ultra-high field". In: *The British Journal of Radiology* 92.1094 (2019), p. 20180492.

[11] Mark E Ladd, Peter Bachert, Martin Meyerspeer, Ewald Moser, Armin M Nagel, David G Norris, Sebastian Schmitter, Oliver Speck, Sina Straub, and Moritz Zaiss. "Pros and cons of ultra-high-field MRI/MRS for human application". In: *Progress in nuclear magnetic resonance spectroscopy* 109 (2018), pp. 1–50.

[12] Eric Maire and Philip John Withers. "Quantitative X-ray tomography". In: *International Materials Reviews* 59.1 (2014), pp. 1–43.

[13] Francisco García-Moreno, Paul Hans Kamm, Tillmann Robert Neu, Felix Bülk, Mike Andreas Noack, Mareike Wegener, Nadine von der Eltz, Christian Matthias Schlepütz, Marco Stampanoni, and John Banhart. "Tomoscopy: Time-Resolved Tomography for Dynamic Processes in Materials". In: *Advanced Materials* 33.45 (2021), p. 2104659.

[14] Geert Litjens, Thijs Kooi, Babak Ehteshami Bejnordi, Arnaud Arindra Adiyoso Setio, Francesco Ciompi, Mohsen Ghafoorian, Jeroen Awm Van Der Laak, Bram Van Ginneken, and Clara I Sánchez. "A survey on deep learning in medical image analysis". In: *Medical Image Analysis* 42 (2017), pp. 60–88.

[15] Alexander Selvikvåg Lundervold and Arvid Lundervold. "An overview of deep learning in medical imaging focusing on MRI". In: *Zeitschrift für Medizinische Physik* 29.2 (2019), pp. 102–127.

[16] Ujjwal Baid, Satyam Ghodasara, Suyash Mohan, Michel Bilello, Evan Calabrese, Errol Colak, Keyvan Farahani, Jayashree Kalpathy-Cramer, Felipe C Kitamura, Sarthak Pati, et al. "The rsna-asnr-miccai brats 2021 benchmark on brain tumor segmentation and radiogenomic classification". In: *arXiv preprint arXiv:2107.02314* (2021).

[17] Aaron J Grossberg, Abdallah SR Mohamed, Hesham Elhalawani, William C Bennett, Kirk E Smith, Tracy S Nolan, Bowman Williams, Sasikarn Chamchod, Jolien Heukelom, Michael E Kantor, et al. "Imaging and clinical data archive for head and neck squamous cell carcinoma patients treated with radiotherapy". In: *Scientific data* 5.1 (2018), pp. 1–10.

[18] A Emre Kavur, N Sinem Gezer, Mustafa Barış, Sinem Aslan, Pierre-Henri Conze, Vladimir Groza, Duc Duy Pham, Soumick Chatterjee, Philipp Ernst, Savaş Özkan, et al. "CHAOS challenge-combined (CT-MR) healthy abdominal organ segmentation". In: *Medical Image Analysis* 69 (2021), p. 101950.

[19] Bennett Landman, Zhoubing Xu, J Igelsias, Martin Styner, T Langerak, and Arno Klein. "MICCAI multi-atlas labeling beyond the cranial vault–workshop and challenge". In: *Proc. MICCAI Multi-Atlas Labeling Beyond Cranial Vault—Workshop Challenge*. Vol. 5. 2015, p. 12.

[20] Vladimír Ulman, Martin Maška, Klas EG Magnusson, Olaf Ronneberger, Carsten Haubold, Nathalie Harder, Pavel Matula, Petr Matula, David Svoboda, Miroslav Radojevic, et al. "An objective comparison of cell-tracking algorithms". In: *Nature methods* 14.12 (2017), pp. 1141–1152.

[21] Jiancheng Yang, Rui Shi, Donglai Wei, Zequan Liu, Lin Zhao, Bilian Ke, Hanspeter Pfister, and Bingbing Ni. "Medmnist v2: A large-scale lightweight benchmark for 2d and 3d biomedical image classification". In: *arXiv preprint arXiv:2110.14795* (2021).

[22] Nima Tajbakhsh, Laura Jeyaseelan, Qian Li, Jeffrey N Chiang, Zhihao Wu, and Xiaowei Ding. "Embracing imperfect datasets: A review of deep learning solutions for medical image segmentation". In: *Medical Image Analysis* 63 (2020), p. 101693.

[23] Shervin Minaee, Yuri Y Boykov, Fatih Porikli, Antonio J Plaza, Nasser Kehtarnavaz, and Demetri Terzopoulos. "Image segmentation using deep learning: A survey". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)* (2021).

[24] Sébastien Brisard, Marijana Serdar, and Paulo JM Monteiro. "Multiscale X-ray tomography of cementitious materials: A review". In: *Cement and Concrete Research* 128 (2020), p. 105824.

[25] Agnese Piovesan, Valérie Vancauwenberghe, Tim Van De Looverbosch, Pieter Verboven, and Bart Nicolaï. "X-ray computed tomography for 3D plant imaging". In: *Trends in Plant Science* 26.11 (2021), pp. 1171–1185.

[26] Simone Carmignato, Wim Dewulf, and Richard Leach. *Industrial X-ray computed tomography*. Springer, 2018.

[27] Ajali Goswami. *Phenome10K: a free online repository for 3-D scans of biological and palaeontological specimens*. 2015. URL: www.phenome10k.org.

[28] Masa Prodanovic, Maria Esteva, Matthew Hanlon, Gaurav Nanda, and Prateek Agarwal. *Digital Rocks Portal: a repository for porous media images*. 2015. DOI: 10.17612/P7CC7K.

[29] Daniel Pflugfelder. *3D Magnetic resonance images of three weeks old barley roots grown in different soils*. 2017. DOI: 10.5447/IPK/2017/10.

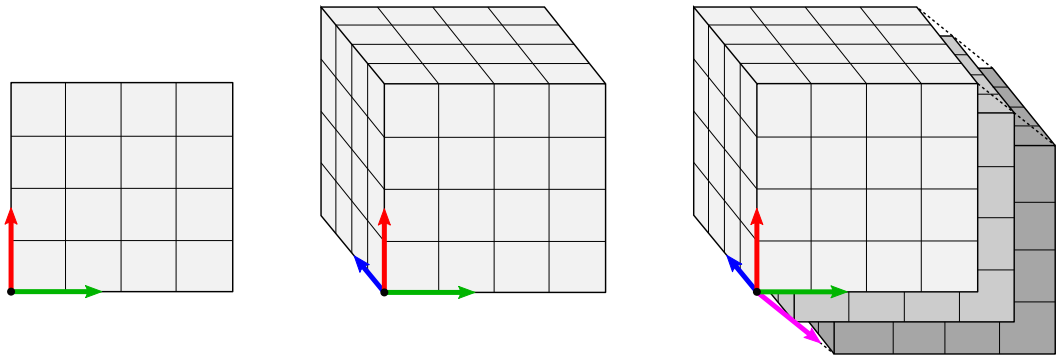
[30] Renaud Lebrun and Maëva J Orliac. "MorphoMuseuM: an online platform for publication and storage of virtual specimens". In: *The Paleontological Society Papers* 22 (2016), pp. 183–195.

2 Background

In this chapter, I provide a brief background on the most important concepts in this thesis which are not defined in the included contributions. The goal is to establish the terms used in the later chapters and to position the work of this thesis within the wider scope of image analysis as a research field.

2.1 Digital Images and Segmentation

While the word *image* has a wide scope in daily speech, we will take it to mean something highly specific in this text. An image, I, refers to a set of values arranged in a finite N-dimensional grid, as shown in Figure 2.1. Commonly, it is assumed that an image represents regular samples from an underlying continuous-coordinate function $\mathcal{I} : \mathbb{R}^N \to \mathbb{R}^M$, as that often reflects the reality of how images are captured. In 2D, we refer to each element of I as a pixel (from *picture element*) and in 3D or above as a voxel (from *volume element*). For simplicity, I will use voxel henceforth, except when explicitly talking about 2D images. Each voxel corresponds to a region of space of size $\delta_1 \times \ldots \times \delta_N$. Typically, it is assumed implicitly that all δ_i are equal which is referred to as isotropic voxel size. However, this is not always the case. For example, images from medical scanners often have anisotropic voxel sizes where the voxels have different side lengths due to how the images were obtained.

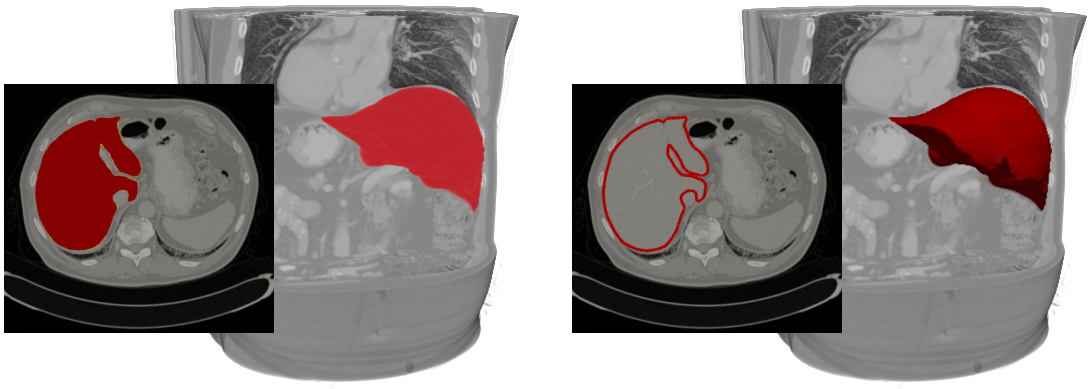


(a) 2D image. (b) 3D image. (c) 4D image.

Figure 2.1: Digital images in multiple dimensions. Images are divided into boxes denoted as pixels in 2D and voxels for 3D and above, which are arranged in an n-dimensional grid.

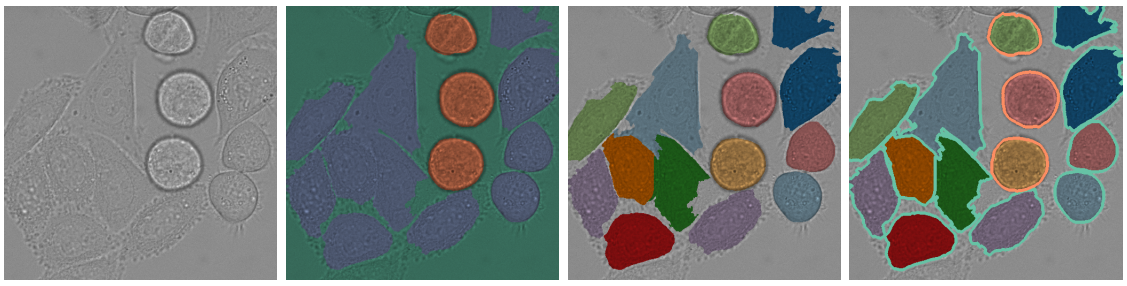
In this thesis, I focus on so-called gray-scale images, where the value stored at each voxel is a scalar. The reason is that in 3D and above, segmentation is typically done on images that are acquired by equipment that only outputs scalars. However, storing multidimensional values is also common. In 2D, color images typically store a vector of red, green, and blue values at each pixel. In 3D, diffusion tensor MRI produces images where each voxel stores a tensor that describes the diffusion properties [1].

By image segmentation, we refer to a partition of the image into regions that each have semantic meaning. The segmentation can either be defined by filled regions, e.g. another image where each voxel stores a label, or by storing the boundary of each region. See Figure 2.2 for an illustration. In this work, we focus on the latter case where the object boundaries are stored as meshes. We elaborate on meshes in Section 2.3.



(a) Liver segmentation defined by filled region. (b) Liver segmentation defined by boundary.

Figure 2.2: Two ways of representing a segmentation of a clinical X-ray CT scan of a human torso from [2]. We show a 3D rendering of the image data with the segmentation in red. The inset shows a horizontal slice through the 3D image and segmentation.



(a) Raw image. (b) Semantic. (c) Instance. (d) Panoptic.

Figure 2.3: Different segmentation types illustrated on an image (a) of dividing cells from [3]. In semantic segmentation (b), the image is divided into background (green), normal cells (blue), and cells about to divide (orange) but we do not differentiate individual cells. In instance segmentation (c), each cell is segmented with a separate label, but we do not distinguish between cell types. In panoptic segmentation (d), each cell is both detected as a separate object and classified as a normal (blue outline) or dividing cell (orange outline).

Image segmentation can be further split into three subcategories [4]: Semantic segmentation, instance segmentation, and panoptic segmentation. The differences between these are illustrated in Figure 2.3 and can be explained as:

- **Semantic segmentation** partitions the image into disjoint regions according to a semantic label. However, no distinction is made between different objects that share the same label. As a result, multiple objects may fuse into one connected component in the segmentation, such as the cells in the left part of Figure 2.3b.
- **Instance segmentation** segments each object of one or more predetermined semantic classes with a separate label. However, no distinction is made between segmented objects of different semantic classes. Also, objects that are not in the predetermined classes are not segmented.
- **Panoptic segmentation** is the combination of semantic segmentation and instance segmentation. The image is partitioned into disjoint regions, each having two labels:

a semantic class and an instance id. This provides the most complete description of the image, but, as a result, is also the most challenging.

The work in this thesis focuses on instance segmentation. The reason is that the developed methods focus on deforming an existing boundary to match the image data. This requires that we specify up front what kind of object to segment and how many. Although it is a limitation, this still encompasses a wealth of important image segmentation tasks.

2.2 Neural Networks

The following section contains a brief introduction to deep learning and neural networks. Since deep learning is an incredibly expansive field, a thorough treatment is beyond the scope of this thesis. The goal is to introduce the most important elements for the works in this thesis, namely multilayer perceptrons (MLPs), which are used in Paper F, and convolutional neural networks (CNNs), which are used in Paper E. For a more comprehensive introduction, the reader is referred to the book by Goodfellow, Bengio, and Courville [5].

2.2.1 Multilayer Perceptrons

Multilayer perceptrons (MLPs), also known as feed-forward neural networks, can be seen as the essence of deep learning. Their goal is to approximate some unknown function $F^* : \mathbb{R}^N \to \mathbb{R}^M$. As an example, F^* may be a classifier mapping a vector input $\mathbf{x}$ to a class y or a continuous function mapping a 3D position to a displacement. The name, neural network, comes from the fact that they are based on a simplified model of biological neurons. Each neuron has several inputs and produces an output. The inputs to each neuron may be data samples, e.g., pixel values, or the outputs of other neurons. As a result, the neurons form a directed graph, i.e., a neural *network*. In MLPs, we do not allow cycles in the graph and can therefore arrange the neurons in layers such as in Figure 2.4. The first layer always contains the input data, $\mathbf{x} = (x_1, x_2, \ldots, x_N)$, and the

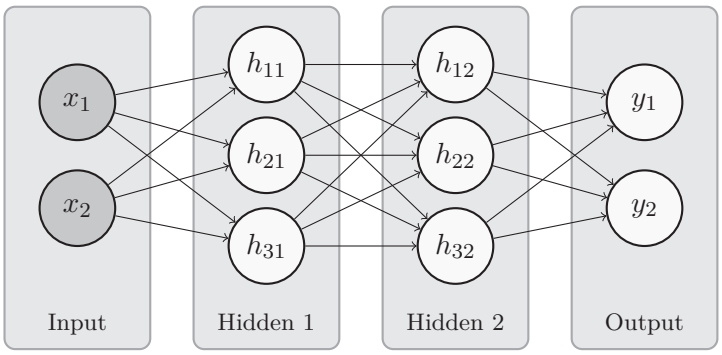


Figure 2.4: A simple multilayer perceptron (MLP) with two hidden layers mapping a 2-dimensional input (x_1, x_2) to a 2-dimensional output (y_1, y_2). The output of each neuron is connected to every neuron in the following layer.

last layer contains the output neurons, $\mathbf{y} = (y_1, y_2, \ldots, y_M)$. We can therefore view an MLP as a function from $\mathbb{R}^N$ to $\mathbb{R}^M$. In between, there may be zero or more so-called hidden layers with additional neurons, h_{kl}. For simplicity, we assume that the output of each neuron is connected to every neuron in the next layer, as shown by the arrows in Figure 2.4.

Mathematically, we model the output of each neuron, f_{kl}, as a linear combination of its C inputs plus a bias, b, followed by a differentiable nonlinear activation function σ. We write it as

$$f_{kl}(\mathbf{x}) = \sigma(w_1 x_1 + w_2 x_2 + \ldots + w_C x_C + b) = \sigma\left(\mathbf{w}^{\mathrm{T}}\mathbf{x} + b\right) . \tag{2.1}$$

Here, $\mathbf{w} \in \mathbb{R}^C$ is a vector that contains the weights for the neurons. Today, the nonlinear activation function σ is usually taken to be the rectified linear unit (ReLU), which is given by $\mathrm{ReLU}(x) = \max\{x, 0\}$. However, several other activation functions have been proposed and the best one is often application dependant. Furthermore, it is common to use a different activation function for the output neurons. We can collect the D neurons in the l'th layer in a single function $f_l : \mathbb{R}^C \to \mathbb{R}^D$ as

$$f_l(\mathbf{x}) = \sigma\left(\mathbf{W}_l^{\mathrm{T}}\mathbf{x} + \mathbf{b}_l\right) , \tag{2.2}$$

where σ is applied elementwise, $\mathbf{W}_l \in \mathbb{R}^{C\times D}$ has the D weight vectors as columns, and $\mathbf{b}_l \in \mathbb{R}^D$ is a vector of the biases. To evaluate an MLP with L layers for an input $\mathbf{x}$, we apply each layer in sequence, using the output of each layer as the input to the next

$$\mathrm{MLP}(\mathbf{x}) = f_L(\dots f_3(f_2(f_1(\mathbf{x})))) . \tag{2.3}$$

The goal now is to find weights and biases such that the MLP approximates the function F^* as well as possible. We call this process training: the MLP learns from prepared training samples $\left(\mathbf{x}^{(i)}, F^*\left(\mathbf{x}^{(i)}\right)\right)$ by comparing its output $\mathrm{MLP}\left(\mathbf{x}^{(i)}\right)$ with the known ground truth $F^*\left(\mathbf{x}^{(i)}\right)$. This comparison is done via a differentiable loss function $L : \mathbb{R}^M \times \mathbb{R}^M \to \mathbb{R}$ where $L\left(\mathrm{MLP}\left(\mathbf{x}^{(i)}\right), F^*\left(\mathbf{x}^{(i)}\right)\right)$ is low when the MLP and F^* agrees and high otherwise. Common choices are cross entropy when training a classifier or mean squared error when training a continuous function.

Training is then performed via (variations of) gradient descent. After computing the loss for a training sample, we can compute the derivative of the loss w.r.t. the weights and biases with a process known as backpropagation [5]. The weights and biases are then updated in the direction of the negative gradient with a given step length called the learning rate. Typically, we compute the loss for a random batch of samples at a time before computing the gradient. This process is repeated until the loss stops decreasing, a given compute budget is exceeded, or some other stop criterium is reached. In addition to this, there is a wealth of other techniques to improve the training of neural networks even further. These include different optimization algorithms, learning rate schedules, parameter regularization, input and output normalization, initialization schemes, and many more. These are all important for the practical performance of neural nets, and more information can be found in [5, 6].

2.2.2 Convolutional Neural Networks

Convolutional neural networks (CNNs) are the cornerstones of modern deep learning models for image segmentation. They are specialized for grid-like data such as images. Conceptually, they are similar to MLPs in that CNNs consist of several layers where the output of each layer is the input for the following layer. The main difference is that, while the basic building blocks of MLPs are matrix multiplication, CNNs are based on convolutions[1] where the kernels are the learnable parameters. As an example, let I be a 2D image and K be the kernel, which itself can be seen as a 2D image. The convolution of I and K, written $I * K$, then results in another 2D image, R, given by

$$R(a, b) = (I * K)(a, b) = \sum_i \sum_j I(a + i, b + j)K(i, j) . \tag{2.4}$$

Typically, K is smaller than I which means each pixel of R only depends on a subset of I. This is illustrated in Figure 2.5 for a 5×5 image and a 3×3 kernel. Note that the output

[1] Technically, most use cross-correlation, which is identical to convolution except the kernel is flipped. In line with most CNN literature, we will refer to both operations as convolution.

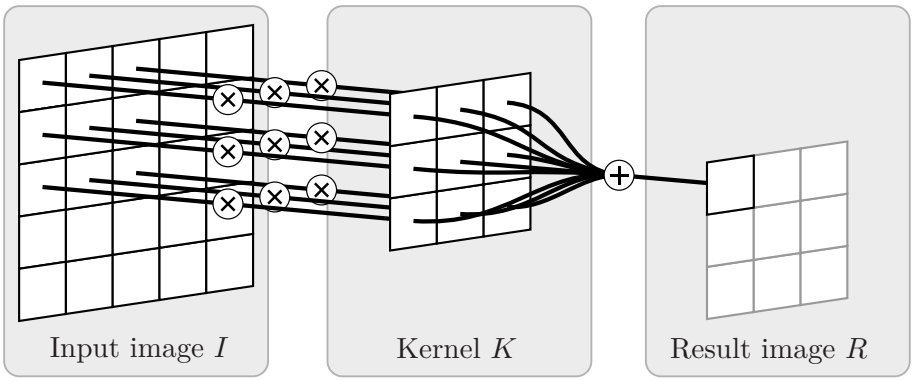


Figure 2.5: Illustration of 2D discrete convolution, which forms the basic building blocks of convolutional neural networks (CNNs). A 5×5 image is convolved with a 3×3 kernel to produce a 3×3 result.

is smaller than the input image. To avoid this, we can pad the input image to make it larger. In the case of vector-valued images, the summation in (2.4) would also run over each pixel component

$$R(a,b) = (I * K)(a,b) = \sum_i \sum_j \sum_c I(a+i, b+j, c) K(i,j,c)\,. \tag{2.5}$$

Note that even if the input image is vector-valued, the result will be a scalar image.

In CNNs, each layer typically contains several kernels which are convolved with the input image separately. After convolution, each result pixel is fed through a non-linear activation function — typically ReLU, but again, this is application specific. The result images are then stacked to form a vector image, which is fed into the next layer. Another important component of CNNs is pooling, which applies a pooling operation — typically maximum or mean — over a sliding window. The stride is usually chosen such that windows do not overlap, which reduces the size of the image, as shown in Figure 2.6. For CNNs that perform segmentation, they commonly include operations to undo this downscaling. Common operations are either simple bilinear upscaling or learned convolutions with extra padding.

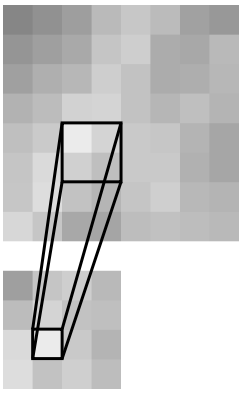


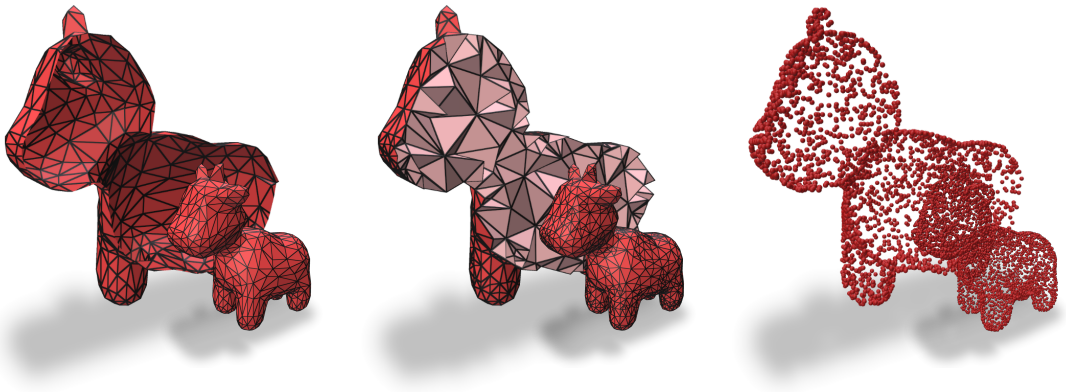
Figure 2.6: Example of max pooling.

To train CNNs, we use the same strategy as with MLPs. Using a modified version of the backpropagation algorithm, we can compute the derivative of each kernel element w.r.t. to a given loss function. Given this, we can train the CNN with gradient-based optimization to minimize this loss function for a set of training examples. As with MLPs, there is a huge amount of additional techniques to improve the training and performance of CNNs, and more information can be found in [5, 7, 8].

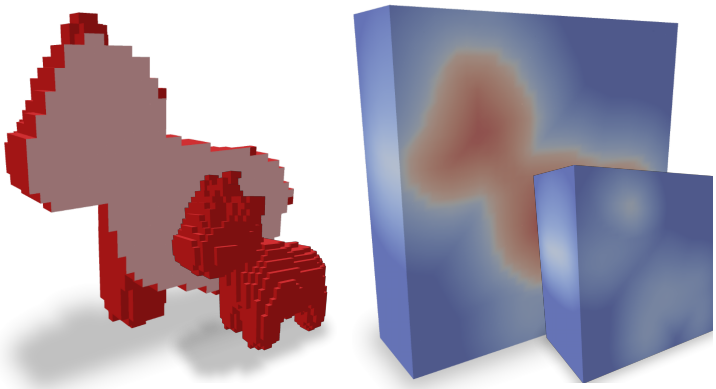
2.3 Shape Representations

As this thesis focuses on segmentation with prior knowledge about shape and topology, it is worth discussing different ways to represent shapes. The major divide in shape representations is whether to represent the shape *explicitly* or *implicitly*.

With explicit representations, we directly represent the boundary of a shape. This gives us direct access to surface points, which makes it easy to compute distance queries to other



(a) Triangle mesh. (b) Tetrahedral mesh. (c) Point cloud.



(d) Voxel volume. (e) Neural Implicit.

Figure 2.7: Examples of common shape representations. Each figure shows the shape sliced in half to display the interior and the inset shows the full shape. The voxel volume in (d) encodes an occupancy function and only inside voxels are shown. The neural implicit in (e) encodes a signed distance function.

shapes and to compute surface properties such as area or curvature. On the other hand, we can not directly check whether an arbitrary point is inside the shape — e.g., to evaluate an image segmentation. The most popular way to implement such an inside/outside query is by shooting a ray from the point to infinity [9, 10]. If the number of intersections with the surface is odd, the point is inside. However, other methods are also possible [11, 12].

With implicit representations, we label every point in space as being either inside or outside the shape. Typically, this is either done using an occupancy function (i.e., a binary labeling) or with a signed distance function (SDF). The surface is then given implicitly as a level set — commonly extracted by evaluating the function on a grid and then using marching cubes [13, 14]. This reverses the situation from explicit representations: We can easily check if a point is inside the shape, but must do work to access the boundary.

We now give an overview of the most prevalent shape representations currently in use:

- **Triangle meshes** (Figure 2.7a) are the classic example of explicit representations and the focus of this thesis. It is a collection of vertices, edges, and triangles where

each triangle represents a small piece of the surface. The vertex positions encode the geometry of the represented shape and the edges and faces encode the topology.

A common way to store triangle meshes is as a list of coordinates for each vertex and a list of vertex indices for each triangle. While simple, it does not store connectivity between edges and triangles which are needed for several mesh operations. If only edge connectivity is needed, common for smoothing, then this may be stored in a separate list or sparse matrix. If general connectivity is needed, e.g., for dynamic mesh changes, the most popular data structure is the so-called half-edge data structure [15], also known as a doubly connected edge list (DCEL). Finally, it should be noted that meshes may be comprised of other polygons than triangles, e.g., quadrilaterals, in which case they are known as polygonal meshes. Nevertheless, we can always convert a polygonal mesh to a triangle mesh by splitting each polygon into triangles.

- **Tetrahedral meshes** (Figure 2.7b) are similar to triangle meshes, but instead represent a volume rather than a surface. They are mainly included here due to their use in Paper D. It is a collection of vertices, edges, triangles, and tetrahedra where each tetrahedron represents a small piece of volume. An advantage over triangle meshes is that we can check if a point is contained inside the shape by checking if it is contained in one of the tetrahedra instead of performing ray-intersection tests.

  As with triangle meshes, tetrahedral meshes are often stored as a list of vertex coordinates, a list of vertex indices for each tetrahedron, and, if needed, a list of edges. For general connectivity, the situation is not as settled as for triangle meshes, and several data structures are used [16–19]. As with triangle meshes, a main application for such data structures is to allow dynamic updates to the mesh.

- **Point clouds** (Figure 2.7c) sit between explicit and implicit representations and are simply a list of points representing samples from the surface of a shape. Their main benefit is their high flexibility, as they can easily represent complex surfaces of any topology. The downside is that topological information is not explicitly encoded, which can introduce ambiguities for highly complex geometries. To convert a point cloud to a triangle mesh common strategies are the Ball Pivoting algorithm [20] and Poisson surface reconstruction [21, 22]. Both require estimates of surface normals for each point, which may be part of the point set or estimated from nearby points.

- **Voxel volumes** (Figure 2.7d) are the classic example of implicit representations. At every voxel, we store either an occupancy value or a signed distance. The main strength of voxel volumes is their simplicity. Also, since they have been in use for a long time, there is a large selection of tools for working with this representation. The major drawbacks are their memory and computational requirements — especially when representing fine geometric details since that requires a very dense grid. To alleviate this, some authors have used sparse data structures such as octrees to represent volumes [23].

- **Neural implicits** (Figure 2.7e) have emerged in recent years as a powerful implicit shape representation. Here, an MLP is trained to map arbitrary 3D positions to occupancy values or signed distances. Since the representation is continuous, neural implicits can represent shapes with fine details without using large amounts of memory — to the point that they sometimes use less memory than a triangle mesh representation. This property makes such representation extremely attractive, which has led to an explosion of papers in the last few years.

2.4 Priors for Image Segmentation

Incorporating prior knowledge has long been a tried and true strategy for improving segmentation algorithms. Allowing for some overlap, we can organize the imposed prior knowledge into three overall groups: appearance priors, abstract priors, and shape priors. Note that many segmentation methods combine several priors, e.g., an appearance and a shape prior. The group on shape priors is the most relevant for this thesis and will therefore be the focus.

2.4.1 Appearance Priors

Appearance priors are based on the distribution of object brightness, color, and texture. A trivial example is knowing that we seek bright objects such as in thresholding [24]. More typically, we seek to learn an appearance model given a set of training examples and then use this model to classify pixels in later images. For brightness and color, this may be done using a Gaussian mixture model (GMM) [25], principal component analysis (PCA) [26] linear discriminant analysis (LDA) or similar methods [27, 28]. For texture, researchers have made use of filter banks [29], engineered feature vectors [30], image patch dictionaries [31], and more [32]. Today, CNNs are arguably the standard way to learn an appearance prior [33] as they make use of both texture and shape [34].

In this thesis, appearance priors have not been the focus. For the contributions in Chapter 3, we make use of very simple appearance models that only seek bright or dark regions. In Chapter 4, we do make use of CNN models but only as backbones for the contributed methods.

2.4.2 Abstract Priors

Abstract priors are those that do not make use of appearance or shape. Topological priors constrain the connectivity and/or genus of the segmentation [35, 36]. The first ensures that each object is represented as a single connected component. The second ensures that objects do not have spurious holes or that they maintain the correct number of holes. Knowledge may also be introduced interactively by a user by constraining some image regions to be background or object [37, 38]. Finally, when segmenting multiple objects, we may enforce that one object contains another, that objects cannot overlap, and that objects must be separated by a minimum or maximum distance [39–41].

Topological priors are especially important for this thesis since representing an object via its surface fixes its topology. Therefore, all methods in this thesis fall under this category. Furthermore, Paper C dealt with how to enforce exclusion during segmentation, and Paper B with how to accelerate segmentation under inclusion/exclusion and separation constraints.

2.4.3 Shape Priors

Shape priors are based on knowledge of the expected shape of objects. Note that in this section, we widen the scope to include shape prior methods used for other tasks than image segmentation, e.g., surface reconstruction. At the low level, we have priors such as favoring boundaries that are smooth or have a small surface area, used in e.g., active contour models (ACMs) [42–45] and Markov Random Fields (MRFs) [46]. At the mid-level, we have priors that favor tubular or flat structures [47, 48]. Other examples are priors which constrain the volume of objects [49, 50]. In recent years, some have used the prior that local surface details are reoccurring over a surface [51–53].

At the high level, we have priors that control the overall shape. Atlas-based segmentation learns a representative mean shape from a set of training examples. This is stored either

as a binary or as a probability volume. Segmentation then corresponds to registering the atlas to a new image [54]. Wu and Chen [55] developed a surface fitting framework based on minimum-cut/maximum-flow (min-cut/max-flow) that constrains the final fit to not deviate significantly from an initial shape [39, 56–59]. Active shape models and active appearance models learn a space of plausible shapes by computing a PCA of a set of aligned training shapes [60].

In recent years, there have been large developments in the application of deep neural networks to shape priors. In [61–66] they use a neural network to deform a template mesh to match the image data — thereby making use of a learned shape prior. In [67, 68] they train generative models to synthesize shapes from incomplete inputs. Neural implicit models have also proven especially powerful as shape priors [69–72] and are starting to find use for 3D images [73–76]. These works aim to represent a shape with a latent vector from a learned shape space. Newer methods are also exploring using multiple latent vectors that each describe a local region [77–80]. This makes the methods more flexible, which makes them drastically better at representing complex geometry. However, most research in this direction is focused on simply representing a specific geometry and less on learning a shape prior.

Every method in this thesis has made use of shape priors to some extent. In Chapter 3 the spatial priors were fairly simple — Paper C uses the shape deviation prior common to the min-cut/max-flow surface fitting methods and Paper D uses a smoothness prior similar to ACMs. In Chapter 4, Paper E also uses an ACM-like smoothness prior. In Paper F, we take inspiration from recent developments in learned local shape priors and create a new mesh shape prior with high flexibility.

References

[1] Roland Bammer. “Basic principles of diffusion-weighted imaging”. In: *European Journal of Radiology* 45 (2003), pp. 169–184.

[2] Amber L Simpson, Michela Antonelli, Spyridon Bakas, Michel Bilello, Keyvan Farahani, Bram Van Ginneken, Annette Kopp-Schneider, Bennett A Landman, Geert Litjens, Bjoern Menze, et al. “A large annotated medical image dataset for the development and evaluation of segmentation algorithms”. In: *arXiv preprint arXiv:1902.09063* (2019).

[3] Vladimír Ulman, Martin Maška, Klas EG Magnusson, Olaf Ronneberger, Carsten Haubold, Nathalie Harder, Pavel Matula, Petr Matula, David Svoboda, Miroslav Radojevic, et al. “An objective comparison of cell-tracking algorithms”. In: *Nature methods* 14.12 (2017), pp. 1141–1152.

[4] Alexander Kirillov, Kaiming He, Ross Girshick, Carsten Rother, and Piotr Dollár. “Panoptic segmentation”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019, pp. 9404–9413.

[5] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[6] Jan Kukačka, Vladimir Golkov, and Daniel Cremers. “Regularization for deep learning: A taxonomy”. In: *arXiv preprint arXiv:1710.10686* (2017).

[7] Laith Alzubaidi, Jinglan Zhang, Amjad J Humaidi, Ayad Al-Dujaili, Ye Duan, Omran Al-Shamma, José Santamaría, Mohammed A Fadhel, Muthana Al-Amidie, and Laith Farhan. “Review of deep learning: Concepts, CNN architectures, challenges, applications, future directions”. In: *Journal of Big Data* 8.1 (2021), pp. 1–74.

[8] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. "A ConvNet for the 2020s". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2022, pp. 11976–11986.

[9] Yehuda E. Kalay. "Determining the spatial containment of a point in general polyhedra". In: *Computer Graphics and Image Processing* 19.4 (1982), pp. 303–334.

[10] Johann Linhart. "A quick point-in-polyhedron test". In: *Computers and Graphics* 14.3-4 (1990), pp. 445–447.

[11] Jakob Andreas Bærentzen and Henrik Aanaes. "Signed distance computation using the angle weighted pseudonormal". In: *IEEE Transactions on Visualization and Computer Graphics* 11.3 (2005), pp. 243–253.

[12] Alec Jacobson, Ladislav Kavan, and Olga Sorkine-Hornung. "Robust inside-outside segmentation using generalized winding numbers". In: *ACM Transactions on Graphics (TOG)* 32.4 (2013), pp. 1–12.

[13] William E Lorensen and Harvey E Cline. "Marching cubes: A high resolution 3D surface construction algorithm". In: *Proceedings of SIGGRAPH* 21.4 (1987), pp. 163–169.

[14] Thomas Lewiner, Hélio Lopes, Antônio Wilson Vieira, and Geovan Tavares. "Efficient implementation of marching cubes' cases with topological guarantees". In: *Journal of Graphics Tools* 8 (2003), pp. 1–15.

[15] Jakob Andreas Bærentzen, Jens Gravesen, François Anton, and Henrik Aanæs. *Guide to computational geometry processing: foundations, algorithms, and methods*. Springer Science & Business Media, 2012.

[16] Jean-Daniel Boissonnat, Olivier Devillers, Monique Teillaud, and Mariette Yvinec. "Triangulations in CGAL". In: *Proceedings of the sixteenth annual symposium on Computational geometry*. 2000, pp. 11–18.

[17] Leila De Floriani, Annie Hui, Daniele Panozzo, and David Canino. "A dimension-independent data structure for simplicial complexes". In: *Proceedings of the SIAM International Meshing Roundtable (IMR)*. 2010, pp. 403–420.

[18] Michael Kremer, David Bommes, and Leif Kobbelt. "OpenVolumeMesh–A versatile index-based data structure for 3D polytopal complexes". In: *Proceedings of the SIAM International Meshing Roundtable (IMR)*. Springer, 2012, pp. 531–548.

[19] Hang Si. "TetGen, a Delaunay-based quality tetrahedral mesh generator". In: *ACM Transactions on Mathematical Software* 41.2 (2015), p. 11.

[20] Fausto Bernardini, Joshua Mittleman, Holly Rushmeier, Cláudio Silva, and Gabriel Taubin. "The ball-pivoting algorithm for surface reconstruction". In: *IEEE Transactions on Visualization and Computer Graphics (VGC)* 5.4 (1999), pp. 349–359.

[21] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. "Poisson surface reconstruction". In: *Proceedings of the Symposium on Geometry Processing (SGP)*. Vol. 7. 2006.

[22] Michael Kazhdan and Hugues Hoppe. "Screened poisson surface reconstruction". In: *ACM Transactions on Graphics (TOG)* 32.3 (2013), pp. 1–13.

[23] Yun-Peng Xiao, Yu-Kun Lai, Fang-Lue Zhang, Chunpeng Li, and Lin Gao. "A survey on deep geometry learning: From a representation perspective". In: *Computational Visual Media* 6.2 (2020), pp. 113–133.

[24] Nobuyuki Otsu. "A threshold selection method from gray-level histograms". In: *IEEE Transactions on Systems, Man, and Cybernetics* 9.1 (1979), pp. 62–66.

[25] Carsten Rother, Vladimir Kolmogorov, and Andrew Blake. ""GrabCut" — interactive foreground extraction using iterated graph cuts". In: *ACM Transactions on Graphics (TOG)* 23.3 (2004), pp. 309–314.

[26] Timothy F. Cootes, Gareth J. Edwards, and Christopher J. Taylor. “Active appearance models”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)* 23.6 (2001), pp. 681–685.

[27] Richard Szeliski. *Computer vision: algorithms and applications*. Springer Nature, 2022.

[28] Trevor Hastie, Robert Tibshirani, and Jerome H. Friedman. *The elements of statistical learning: data mining, inference, and prediction*. Vol. 2. 2009.

[29] Jitendra Malik, Serge Belongie, Thomas Leung, and Jianbo Shi. “Contour and texture analysis for image segmentation”. In: *International Journal of Computer Vision (IJCV)* 43.1 (2001), pp. 7–27.

[30] Michael Crosier and Lewis D Griffin. “Using basic image features for texture classification”. In: *International Journal of Computer Vision (IJCV)* 88.3 (2010), pp. 447–460.

[31] Anders Lindbjerg Dahl and Rasmus Larsen. “Learning Dictionaries of Discriminative Image Patches.” In: *Proceedings of the British Machine Vision Conference (BMVC)*. 2011, pp. 1–11.

[32] Anne Humeau-Heurtier. “Texture Feature Extraction Methods: A Survey”. In: *IEEE access* 7 (2019), pp. 8975–9000.

[33] Shervin Minaee, Yuri Y Boykov, Fatih Porikli, Antonio J Plaza, Nasser Kehtarnavaz, and Demetri Terzopoulos. “Image segmentation using deep learning: A survey”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)* (2021).

[34] Md Amirul Islam, Matthew Kowal, Patrick Esser, Sen Jia, Bjorn Ommer, Konstantinos G Derpanis, and Neil Bruce. “Shape or texture: Understanding discriminative features in cnns”. In: *Proceedings of the International Conference on Learning Representations (ICLR)*. 2021.

[35] Xiao Han, Chenyang Xu, and Jerry L. Prince. “A topology preserving level set method for geometric deformable models”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)* 25.6 (2003), pp. 755–768.

[36] Yun Zeng, Dimitris Samaras, Wei Chen, and Qunsheng Peng. “Topology cuts: A novel min-cut/max-flow algorithm for topology preserving segmentation in N–D images”. In: *Computer Vision and Image Understanding (CVIU)* 112.1 (2008), pp. 81–90.

[37] Yuri Y Boykov and M-P Jolly. “Interactive graph cuts for optimal boundary & region segmentation of objects in ND images”. In: *Proceedings of the International Conference on Computer Vision (ICCV)*. Vol. 1. 2001, pp. 105–112.

[38] Leo Grady. “Random walks for image segmentation”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)* 28.11 (2006), pp. 1768–1783.

[39] Kang Li, Xiaodong Wu, Danny Z Chen, and Milan Sonka. “Optimal surface segmentation in volumetric images-a graph-theoretic approach”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)* 28.1 (2005), pp. 119–134.

[40] Andrew Delong and Yuri Boykov. “Globally optimal segmentation of multi-region objects”. In: *Proceedings of the International Conference on Computer Vision (ICCV)*. 2009, pp. 285–292.

[41] Masoud S Nosrati and Ghassan Hamarneh. “Local optimization based segmentation of spatially-recurring, multi-region objects with part configuration constraints”. In: *IEEE Transactions on Medical Imaging (TMI)* 33.9 (2014), pp. 1845–1859.

[42] Michael Kass, Andrew Witkin, and Demetri Terzopoulos. “Snakes: Active contour models”. In: *International Journal of Computer Vision (IJCV)* 1.4 (1988), pp. 321–331.

[43] T. Mcinerney and D. Terzopoulos. “A Dynamic Finite Element Surface Model for Segmentation and Tracking in Multidimensional Medical Images with Application to Cardiac 4D Image Analysis”. In: *Computerized Medical Imaging and Graphics* 19.1 (1995), pp. 69–83.

[44] Tony F Chan and Luminita A Vese. “Active contours without edges”. In: *IEEE Transactions on Image Processing (TIP)* 10.2 (2001), pp. 266–277.

[45] Udaranga Wickramasinghe, Pascal Fua, and Graham Knott. “Deep active surface models”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2021, pp. 11652–11661.

[46] Stan Z Li. *Markov random field modeling in image analysis*. Springer Science & Business Media, 2009.

[47] Delphine Nain, Anthony Yezzi, and Greg Turk. “Vessel segmentation using a shape driven flow”. In: *Proceedings of the International Conference on Medical Image Computing and Computer Assisted Intervention (MICCAI)*. Springer. 2004, pp. 51–59.

[48] Marie Rochery, Ian H Jermyn, and Josiane Zerubia. “Higher order active contours”. In: *International Journal of Computer Vision (IJCV)* 69.1 (2006), pp. 27–42.

[49] Ismail Ben Ayed, Shuo Li, Ali Islam, Greg Garvin, and Rethy Chhem. “Area prior constrained level set evolution for medical image segmentation”. In: *SPIE Medical Imaging*. Vol. 6914. 2008, pp. 27–32.

[50] Yuyin Zhou, Zhe Li, Song Bai, Chong Wang, Xinlei Chen, Mei Han, Elliot Fishman, and Alan L Yuille. “Prior-aware neural network for partially-supervised multi-organ segmentation”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019, pp. 10672–10681.

[51] Rana Hanocka, Gal Metzer, Raja Giryes, and Daniel Cohen-Or. “Point2Mesh: a self-prior for deformable meshes”. In: *ACM Transactions on Graphics (TOG)* 39.4 (2020), pp. 126–1.

[52] Amir Hertz, Rana Hanocka, Raja Giryes, and Daniel Cohen-Or. “Deep geometric texture synthesis”. In: *ACM Transactions on Graphics (TOG)* 39.4 (2020), pp. 108–1.

[53] Luca Morreale, Noam Aigerman, Paul Guerrero, Vladimir G Kim, and Niloy J Mitra. “Neural Convolutional Surfaces”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2022.

[54] M Bach Cuadra, Valérie Duay, and J-Ph Thiran. “Atlas-based segmentation”. In: *Handbook of biomedical imaging*. Springer, 2015, pp. 221–244.

[55] Xiaodong Wu and Danny Z Chen. “Optimal net surface problems with applications”. In: *International Colloquium on Automata, Languages, and Programming*. Springer. 2002, pp. 1029–1042.

[56] Jan Egger, Miriam HA Bauer, Daniela Kuhnt, Barbara Carl, Christoph Kappus, Bernd Freisleben, and Christopher Nimsky. “Nugget-cut: a segmentation scheme for spherically-and elliptically-shaped 3D objects”. In: *Proceedings of the DAGM Symposium on Pattern Recognition*. 2010, pp. 373–382.

[57] Yao Wang and Reinhard Beichel. “Graph-based segmentation of lymph nodes in CT data”. In: *Proceedings of the International Symposium on Visual Computing (ISVC)*. 2010, pp. 312–321.

[58] Yin Yin, Xiangmin Zhang, Rachel Williams, Xiaodong Wu, Donald D Anderson, and Milan Sonka. “LOGISMOS—layered optimal graph image segmentation of multiple objects and surfaces: cartilage segmentation in the knee joint”. In: *IEEE Transactions on Medical Imaging (TMI)* 29.12 (2010), pp. 2023–2037.

[59] Niels Jeppesen, Anders N Christensen, Vedrana A Dahl, and Anders B Dahl. "Sparse layered graphs for multi-object segmentation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020, pp. 12777–12785.

[60] Timothy F. Cootes, Christopher J Taylor, David H Cooper, and Jim Graham. "Active shape models-their training and application". In: *Computer Vision and Image Understanding (CVIU)* 61.1 (1995), pp. 38–59.

[61] Nanyang Wang, Yinda Zhang, Zhuwen Li, Yanwei Fu, Wei Liu, and Yu-Gang Jiang. "Pixel2mesh: Generating 3d mesh models from single rgb images". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 52–67.

[62] Jiawen Yao, Jinzheng Cai, Dong Yang, Daguang Xu, and Junzhou Huang. "Integrating 3d geometry of organ for improving medical image segmentation". In: *Proceedings of the International Conference on Medical Image Computing and Computer Assisted Intervention (MICCAI)*. 2019, pp. 318–326.

[63] Udaranga Wickramasinghe, Edoardo Remelli, Graham Knott, and Pascal Fua. "Voxel2mesh: 3d mesh model generation from volumetric data". In: *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer. 2020, pp. 299–308.

[64] Fanwei Kong, Nathan Wilson, and Shawn Shadden. "A deep-learning approach for direct whole-heart mesh reconstruction". In: *Medical Image Analysis* 74 (2021), p. 102222.

[65] Fanwei Kong and Shawn C Shadden. "Whole Heart Mesh Generation for Image-Based Computational Simulations by Learning Free-From Deformations". In: *Proceedings of the International Conference on Medical Image Computing and Computer Assisted Intervention (MICCAI)*. 2021, pp. 550–559.

[66] Qiang Ma, Emma C Robinson, Bernhard Kainz, Daniel Rueckert, and Amir Alansary. "PialNN: A fast deep learning framework for cortical pial surface reconstruction". In: *Proceedings of the International Workshop on Machine Learning in Clinical Neuroimaging (MLCN)*. 2021, pp. 73–81.

[67] Or Litany, Alex Bronstein, Michael Bronstein, and Ameesh Makadia. "Deformable shape completion with graph convolutional autoencoders". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018, pp. 1886–1895.

[68] Tianchang Shen, Jun Gao, Kangxue Yin, Ming-Yu Liu, and Sanja Fidler. "Deep Marching Tetrahedra: a Hybrid Representation for High-Resolution 3D Shape Synthesis". In: *Neural Information Processing Systems (NeurIPS)*. 2021.

[69] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. "Deepsdf: Learning continuous signed distance functions for shape representation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019, pp. 165–174.

[70] Zhiqin Chen and Hao Zhang. "Learning implicit fields for generative shape modeling". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019, pp. 5939–5948.

[71] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. "Occupancy networks: Learning 3d reconstruction in function space". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019, pp. 4460–4470.

[72] Vincent Sitzmann, Julien Martel, Alexander Bergman, David Lindell, and Gordon Wetzstein. "Implicit neural representations with periodic activation functions". In: *Neural Information Processing Systems (NeurIPS)*. Vol. 33. 2020, pp. 7462–7473.

[73] Riddhish Bhalodia, Shireen Y Elhabian, Ladislav Kavan, and Ross T Whitaker. "DeepSSM: a deep learning framework for statistical shape modeling from raw images". In: *Proceedings of the International Workshop on Shape in Medical Imaging (ShapeMI)*. 2018, pp. 244–257.

[74] Kristine Sørensen, Oscar Camara, Ole De Backer, Klaus F Kofoed, and Rasmus R Paulsen. "NUDF: Neural Unsigned Distance Fields for High Resolution 3D Medical Image Segmentation". In: *Proceedings of the IEEE International Symposium on Biomedical Imaging (ISBI)*. 2022, pp. 1–5.

[75] Ashwin Raju, Shun Miao, Dakai Jin, Le Lu, Junzhou Huang, and Adam P Harrison. "Deep implicit statistical shape models for 3d medical image delineation". In: *Proceedings of the National Conference on Artificial Intelligence (AAAI)*. Vol. 36. 2. 2022, pp. 2135–2143.

[76] Jiancheng Yang, Udaranga Wickramasinghe, Bingbing Ni, and Pascal Fua. "ImplicitAtlas: Learning Deformable Shape Templates in Medical Imaging". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2022, pp. 15861–15871.

[77] Rohan Chabra, Jan E Lenssen, Eddy Ilg, Tanner Schmidt, Julian Straub, Steven Lovegrove, and Richard Newcombe. "Deep local shapes: Learning local sdf priors for detailed 3d reconstruction". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2020, pp. 608–625.

[78] Ishit Mehta, Michaël Gharbi, Connelly Barnes, Eli Shechtman, Ravi Ramamoorthi, and Manmohan Chandraker. "Modulated periodic activations for generalizable local functional representations". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2021, pp. 14214–14223.

[79] Chiyu Jiang, Avneesh Sud, Ameesh Makadia, Jingwei Huang, Matthias Nießner, Thomas Funkhouser, et al. "Local implicit grid representations for 3d scenes". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020, pp. 6001–6010.

[80] Julian Chibane, Thiemo Alldieck, and Gerard Pons-Moll. "Implicit functions in feature space for 3d shape reconstruction and completion". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020, pp. 6970–6981.

3 Learning-Free Segmentation

This chapter includes the contributions related to image segmentation methods that do not make use of machine learning to learn prior information from data. The focus has been on multi-object segmentation in large multi-GB images where we have a clear expectation of the shapes we are looking for. Here, meshes are used to enforce topological constraints and to incorporate simple shape priors such as smoothness or deviation from an initial shape. Furthermore, meshes are used as a way to avoid working directly with the image grid such that the computational complexity of the methods scale with the mesh size and not the image size.

A large part of this chapter is devoted to methods based on minimum cut/maximum flow (min-cut/max-flow). Specifically, the surface fitting framework originally introduced by Xiaodong Wu and Danny Z. Chen. Here, an initial surface (often a sphere) is deformed by placing new candidate positions for each vertex in a ray along its normal. The optimal placement for each vertex is then selected by solving a min-cut/max-flow problem. See Paper C for additional details. We have focused on using this framework to segment multiple objects under interaction constraints and how to do this in a computationally efficient manner. The contributions are as follows:

A Review of Serial and Parallel Min-Cut/Max-Flow Algorithms for Computer Vision
First, we review and benchmark the current algorithms for min-cut/max-flow in order to assess the current state of the art. We evaluate both serial and parallel algorithms over a wide array of min-cut/max-flow problems from computer vision. For the serial algorithms, we also evaluate multiple variants to investigate the effect of implementation details.

B Faster Multi-Object Segmentation using Parallel Quadratic Pseudo-Boolean Optimization
Next, we develop a new method for solving large quadratic pseudo-Boolean optimization (QPBO) problems in parallel. QPBO becomes necessary when we have exclusion constraints between many objects, but can still be solved via min-cut/max-flow. We demonstrate the performance of the method on a large multi-object segmentation problem making use of both inclusion and exclusion constraints. We also evaluate it on many smaller problems.

C Multi-object Graph-based Segmentation with Non-overlapping Surfaces
Finally, we develop a new method for handling exclusion constraints in multi-object segmentation based on the Wu and Chen surface fitting framework. The method is based on separating each object with a plane. The position of the plane is determined as part of the segmentation which is formulated as a single combined QPBO problem.

The last contribution is dedicated to segmentation of large 4D (3D + time) images. Our contribution belongs to the active contour framework originally introduced by Michael Kass, Andrew Witkin, and Demetri Terzopoulos. Here, an initial surface is iteratively deformed according to two sets of forces: external forces, which pull it toward object boundaries, and internal forces, which regularize the surface. See Paper D for more details. The contribution is:

D Finding Space-Time Boundaries with Deformable Hypersurfaces

We develop a new method for simultaneously segmenting and tracking evolving objects in 4D images which undergo splits or merges but not both. This assumption ensures that the 4D volume traced by the splitting/merging objects is simply connected. We can therefore represent the boundary of this traced volume with a single tetrahedral mesh of (hyper)disk topology. Our method deforms an initial tetrahedral mesh through time and space to fit this boundary. Since we only operate with the deforming boundary mesh, our method is both fast and memory efficient. Once fitted, we can extract a 3D segmentation at any time by computing a cross-section of the 4D tetrahedral mesh. Additionally, we can detect the precise location and time of splits/merges by finding the saddle points of the fitted mesh.

3.1 Paper A: Review of Serial and Parallel Min-Cut/Max-Flow Algorithms for Computer Vision

Patrick M. Jensen, Niels Jeppesen, Anders B. Dahl, Vedrana A. Dahl,
IEEE Transactions on Pattern Analysis and Machine Intelligence, 2022.
DOI: 10.1109/TPAMI.2022.3170096

Review of Serial and Parallel Min-Cut/Max-Flow Algorithms for Computer Vision

Patrick M. Jensen, Niels Jeppesen, Anders B. Dahl, and Vedrana A. Dahl

Abstract—Minimum cut/maximum flow (min-cut/max-flow) algorithms solve a variety of problems in computer vision and thus significant effort has been put into developing fast min-cut/max-flow algorithms. As a result, it is difficult to choose an ideal algorithm for a given problem. Furthermore, parallel algorithms have not been thoroughly compared. In this paper, we evaluate the state-of-the-art serial and parallel min-cut/max-flow algorithms on the largest set of computer vision problems yet. We focus on generic algorithms, *i.e.*, for unstructured graphs, but also compare with the specialized GridCut implementation. When applicable, GridCut performs best. Otherwise, the two pseudoflow algorithms, Hochbaum pseudoflow and excesses incremental breadth first search, achieves the overall best performance. The most memory efficient implementation tested is the Boykov-Kolmogorov algorithm. Amongst generic parallel algorithms, we find the bottom-up merging approach by Liu and Sun to be best, but no method is dominant. Of the generic parallel methods, only the parallel preflow push-relabel algorithm is able to efficiently scale with many processors across problem sizes, and no generic parallel method consistently outperforms serial algorithms. Finally, we provide and evaluate strategies for algorithm selection to obtain good expected performance. We make our dataset and implementations publicly available for further research.

Index Terms—Algorithms, computer vision, graph algorithms, graph-theoretic methods, parallel algorithms, performance evaluation of algorithms and systems

1 Introduction

Min-cut/max-flow algorithms are ubiquitous in computer vision, since a large variety of computer vision problems can be formulated as min-cut/max-flow problems. Example applications include image segmentation [11, 18, 49, 50, 57, 84], stereo matching [14, 64], surface reconstruction [71], surface fitting [24, 60, 70, 74, 97, 101], graph matching [48], and texture restoration [88]. In recent years, min-cut/max-flow algorithms have also found use in conjunction with deep learning methods — for example, to quickly generate training labels [61] or in combination with convolutional neural networks (CNNs) [42, 80, 93].

Greig *et al*. [40] were the first to use min-cut/max-flow algorithms to solve maximum a posterior Markov random field (MRF) problems in computer vision. Later, Boykov and Jolly [9] showed how this could be generalized and Boykov and Kolmogorov [11] proposed a fast min-cut/max-flow algorithm for computer vision problems. Min-cut/max-flow algorithms in computer vision are used to solve a large family of energy minimization problems, and the most commonly used energy function is of the form

$$\mathcal{E}(\mathbf{x}) = \sum_{i \in \mathcal{P}} \mathcal{E}_i(x_i) + \sum_{(i,j) \in \mathcal{N}} \mathcal{E}_{ij}(x_i, x_j), \tag{1}$$

where $\mathcal{P}$ is a set of indices for the binary variables $x_i \in \{0, 1\}$, and $\mathcal{N}$ is a set of index pairs. A unary term $\mathcal{E}_i : \{0, 1\} \to \mathbb{R}$ is associated with variable x_i, and a pairwise term $\mathcal{E}_{ij} : \{0, 1\}^2 \to \mathbb{R}$ is associated with the pair of variables x_i, x_j. As the inputs to the energy terms are binary, the terms are often represented as lookup tables. In a typical application, such as binary segmentation with MRFs [10], $\mathcal{P}$ represents pixels in an image and x_i represents the assignment of pixel i. However, variables can also describe more abstract things [24, 48, 57, 74, 97, 100], *e.g*., candidate positions for mesh vertices.

For energy functions which are *submodular*, meaning that all pairwise energy terms satisfy the condition

$$\mathcal{E}_{ij}(0,0) + \mathcal{E}_{ij}(1,1) \leq \mathcal{E}_{ij}(0,1) + \mathcal{E}_{ij}(1,0), \tag{2}$$

the minimization can be solved directly as a min-cut/max-flow problem [29, 65]. Submodular energies favor neighbors that have the same label, *i.e*., (x_i, x_j) having the labels $(0, 0)$ or $(1, 1)$ rather than $(0, 1)$ or $(1, 0)$. Therefore, submodularity imposes a local smoothness of the solution, which is useful in many computer vision problems. However, some important vision problems are not submodular. In such cases, one can use either a submodular approximation or an approach based on quadratic pseudo-Boolean optimization (QPBO) as described in [6, 44, 63, 88].

Due to the wide applicability of min-cut/max-flow in computer vision, several fast generic min-cut/max-flow algorithms have been developed [11, 36, 45]. In addition, more specialized algorithms have been created that exploit the grid structure of images to reduce memory usage and run time [22, 51, 52, 86, 96]. Furthermore, methods for dynamic problems [36, 62, 102], where a series of similar min-cut/max-flow problems are solved in succession, have been proposed. For such problems, the result of the first solution can be reused to speed up computations of subsequent solutions. Finally, some papers [43, 89, 91, 103] have explored methods that also allow for distributed computation of min-cut/max-flow problems across several computational nodes. This approach is primarily suited for graphs too large to fit in physical memory.

In this paper, we focus on generic min-cut/max-flow algorithms, which do *not* make assumptions about the graph structure (*e.g*., requiring a grid structure). However, for comparison, we include the GridCut algorithm [52] in our evaluation on grid-based graphs. Furthermore, we consider only static problems where a solution is calculated once, without access to a previous solution (as opposed

- *P. M. Jensen, N. Jeppesen, A. B. Dahl, and V. A. Dahl are with the Department of Applied Mathematics and Computer Science, Technical University of Denmark, Kongens Lyngby, Denmark.*
E-mail: {patmjen, niejep, abda, vand}@dtu.dk

to dynamic problems). Finally, for parallel algorithms, we do not consider whether the algorithm works well in a distributed setting, but focus on the shared memory case where the complete graph can be loaded into the memory of one machine.

The goal is that our experimental results can help researchers understand the strengths and weaknesses of the current state-of-the-art min-cut/max-flow algorithms and help practitioners when choosing a min-cut/max-flow algorithm to use for a given problem.

1.1 Related Work

Serial Algorithms Several papers [17, 27, 36, 95] provide comparisons of different serial min-cut/max-flow algorithms on a variety of standard benchmark problems. However, many of these benchmark problems are small w.r.t. the scale of min-cut/max-flow problems that can be solved today — especially when it comes to grid graphs. Also, graphs in which nodes are not based on an image grid are severely underrepresented. Furthermore, [17, 27, 95] do not include all current state-of-the-art algorithms, while other papers do not include initialization times for the min-cut computation. As shown by Verma and Batra [95], it is important for practical use to include the initialization time, as algorithm implementations may spend as much time on initialization as on the min-cut computation. Additionally, existing papers only compare reference implementations (*i.e.*, the implementation released by the authors) of algorithms — the exception being that an optimized version of the BK algorithm is sometimes included, *e.g.*, in [36]. However, as implementation details — *i.e.*, choices that are left unspecified by the algorithm description — can significantly impact performance [95], a systematic investigation of their effect is also important. Finally, existing comparisons focus on determining the overall best algorithm, even though, as we show in this work, the best algorithm depends on the features of the given graph.

Parallel Algorithms To our knowledge, parallel min-cut/max-flow algorithms have not been systematically compared. Papers introducing parallel algorithms only compare with serial algorithms [75, 91, 103] or a single parallel algorithm [5]. The most comprehensive comparison so far was made by Shekhovtsov and Hlaváč [89] who included a generic and grid-based parallel algorithm. However, no paper compares with the approach by Liu and Sun [75], as no public implementation is available, even though it is expected to be the fastest [89, 91]. Additionally, all papers use the same set of computer vision problems used to benchmark serial algorithms. This is not ideal, as the set lacks larger problems which we expect to benefit the most from parallelization [56]. Therefore, how big the performance benefits of parallelization are, and when to expect them, is still to be determined.

1.2 Contributions

We evaluate current state-of-the-art generic serial and parallel min-cut/max-flow algorithms on the largest set of computer vision problems so far. We compare the algorithms on a wide range of graph problems including commonly used benchmarks problems, as well as many new problem instances from recent papers — some of which are significantly larger than previous problems and expose weaknesses in the algorithms not seen with previous datasets. Since the performance of the algorithms varies between problems, we also provide concrete strategies on algorithm selection and evaluate the expected performance of these.

For the serial algorithms, we evaluate the reference implementations of the Hochbaum pseudoflow (HPF) [45, 46], the preflow push-relabel (PPR) [35], and the GridCut [51] algorithms. Moreover, to reduce the influence of implementation details, we evaluate different versions (including our own) of the Excesses Incremental Breadth First Search (EIBFS) [36] and the Boykov-Kolmogorov (BK) [11] algorithm. We chose these for an extended evaluation, as EIBFS is the most recent min-cut/max-flow algorithm and BK is still widely used in the computer vision community.

For the parallel algorithms, we provide the first comprehensive comparison of all major approaches. This includes our own implementation of the bottom-up merging algorithm by Liu and Sun [75], our own version of the dual decomposition algorithm by Strandmark and Kahl [91], the reference implementation of the region discharge algorithm by Shekhovtsov and Hlaváč [89], an implementation of the parallel preflow push-relabel algorithm by Baunstark et al. [5], and the parallel implementation of GridCut (P-GridCut) [51]. In our comparison, we evaluate not just the run time — including both the initialization time and the time for the min-cut/max-flow computations — but also the memory use of the implementations. Memory usage has not received much attention in the literature, despite it often being a limiting factor when working with large problems. Finally, we show that the current parallel algorithm implementations have unpredictable performance and unfortunately often perform worse than serial algorithms.

All tested C++ implementations (except GridCut [51]), including our new implementations of several algorithms, are available at https://github.com/patmjen/maxflow_algorithms and are archived at DOI:10.5281/zenodo.4903945 [54]. We also provide Python wrapper packages for several of the algorithms (including BK and HPF), which can be found at https://github.com/skielex/shrdr. All of our benchmark problems are available at DOI:10.11583/DTU.17091101.

2 Min-Cut/Max-Flow Algorithms for Computer Vision

To illustrate the use of min-cut/max-flow, we will sketch how a vision problem, image segmentation, can be solved using min-cut/max-flow. We start by introducing our notation and defining the min-cut/max-flow problem.

We define a directed graph $G = (V, E)$ by a set of *nodes*, V, and a set of directed *arcs*, E. We let n and m refer to the number of nodes and arcs, respectively. Each arc $(i, j) \in E$ is assigned a non-negative *capacity* c_{ij}. For min-cut/max-flow problems, we define two special *terminal nodes*, s and t, which are referred to as the *source* and *sink*, respectively. The source has only outgoing arcs, while the sink has only incoming arcs. Arcs to and from the terminal nodes are known as *terminal arcs*.

A *feasible flow* in the graph G is an assignment of non-negative numbers (flows), f_{ij}, to each arc $(i, j) \in E$. A feasible flow must satisfy the following two types of constraints: *capacity constraints*, $f_{ij} \leq c_{ij}$, and *conservation constraints*, $\sum_{i|(i,j)\in E} f_{ij} = \sum_{k|(j,k)\in E} f_{jk}$ for all nodes $j \in V \setminus \{s, t\}$. Capacity constraints ensure that the flow along an arc does not exceed its capacity. Conservation constraints ensure that the flow going into a node equals the flow coming out. See Fig. 1(a) for an example of the graph and a feasible flow. The *value* of the flow is the total flow out of the source or, equivalently, into the sink, and the *maximum flow* problem refers to finding a feasible flow that maximizes the flow value.

An *s-t cut* is a partition of the nodes into two disjoint sets S and T such that $s \in S$ and $t \in T$. The sets S and T are referred

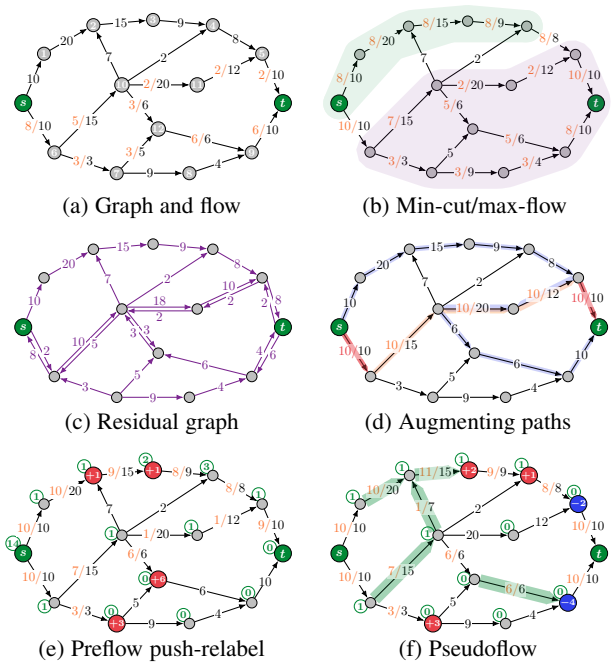


Fig. 1: **Graph basics and serial algorithms.** (a) An example of the graph and a feasible (non-maximal) flow. The flow and capacity for each arc is written as f_{ij}/c_{ij}, and (to reduce clutter) zero-values of the flow are omitted. The flow is 8, which is not maximal, so no s-t cut is evident. (b) The min-cut/max-flow with a value of 18, which all min-cut/max-flow algorithms will eventually arrive at. (c) Residual graph for the flow from (a). (d) An intermediate flow while running the AP algorithm. In the first iteration, 10 units are pushed along the path highlighted in orange and red, saturating two terminal arcs (red). In the next iteration, flow is pushed along the residual path highlighted in blue. (e) A preflow at an intermediate stage of a PPR algorithm. Nodes with excess are shown in red, and a label in green is attached to every node. (f) A pseudoflow at an intermediate stage of the HPF algorithm. Nodes with surplus/deficit are shown in red/blue, a label is attached to every node, and arcs of the tree structure are highlighted in green.

to as the source and sink set, respectively. The *capacity* of the cut is the sum of capacities of the arcs going from S to T. And the *minimum cut* problem refers to finding a cut that minimizes the cut capacity. Often, this partition of the nodes is all that is needed for computer vision applications. Therefore, some algorithms only compute the minimum cut, and an additional step would be needed to extract the flow value for every arc.

Finally, the max-flow min-cut theorem states that the value of the maximum flow is exactly the capacity of the minimum cut. Fig. 1(b) shows min-cut/max-flow on a small graph. This can be shown by formulating both problems as linear programs, which reveals that max-flow is the strong dual of the min-cut.

2.1 Image Segmentation

When formulating segmentation as a min-cut/max-flow problem, one modeling choice involves deciding which structures to represent as graph nodes. Often, nodes of the graph represent individual image pixels, but various other entities may be associated with graph nodes, some of which are illustrated in Fig. 2.

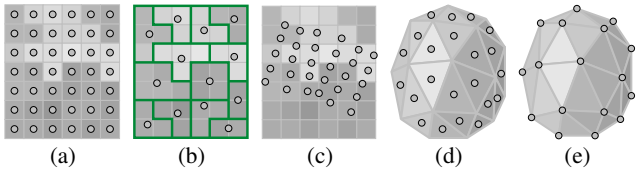


Fig. 2: **Some possibilities for associating graph nodes with entities used for segmentation.** Graph nodes (gray dots) associated with (a) image pixels, (b) superpixels (c) positions in the image (d) mesh faces (e) mesh vertices.

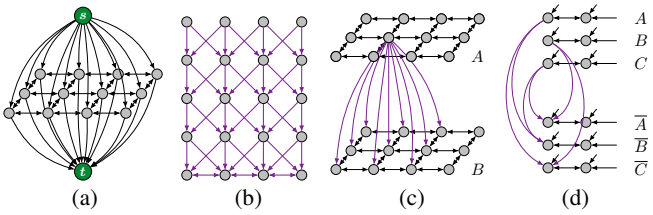


Fig. 3: **Some typical segmentation models.** Terminal arcs are shown only for the first example. Arcs drawn in purple have infinite capacity. (a) A classical MRF segmentation with 4-connected grid graph. (b) A multi-column graph used for segmenting layered structures. (c) Two-object segmentation with inclusion constraint. (d) Three-object segmentation with mutual exclusion using QPBO.

The energy formulation (1) is convenient when min-cut/max-flow algorithms are used to optimize MRFs. Here, each unary energy term is a likelihood energy (negative log likelihood) of a pixel being labeled 0 or 1. Likelihood terms are typically computed directly from image data. The pairwise terms are defined for pairs of pixels, so-called neighbors, and for 2D images, the neighborhood structure is usually given by a 4 or 8-connectivity.

The typical pairwise energy terms used in (1) are

$$\mathcal{E}_{ij}(0,0)=\mathcal{E}_{ij}(1,1)=0 \text{ and } \mathcal{E}_{ij}(0,1)=\mathcal{E}_{ij}(1,0)=\beta_{ij}\ . \qquad (3)$$

These terms penalize neighboring pixels having different labels by a fixed amount, β_{ij}, thus *encouraging* smoothness of the segmentation. In this case, the construction of the s-t graph which exactly represents the energy function is straightforward: The node set is $V = \mathcal{P} \cup \{s, t\}$. For terminal arc capacities, c_{si} and c_{it}, we use the unary terms $\mathcal{E}_i(0)$ and $\mathcal{E}_i(1)$, respectively. Meanwhile, pairwise energy terms correspond to non-terminal arc capacities, such that $c_{ij} = c_{ji} = \beta_{ij}$. Fig. 3(a) shows this construction for a 4-connected grid graph. The binary segmentation of the image corresponds directly to the binary labeling given by the minimum cut. Put in another way, the sets S and T give the optimal labeling of the nodes, and because we have a 1-to-1 mapping between non-terminal nodes and pixels, the node labeling is the segmentation. However, there are many more advanced ways to formulate image segmentation using binary energy optimization and s-t graphs, and ways to formulate other computer vision problems as well [65].

An example closely related to image segmentation is surface fitting, where [74] uses arcs of infinite capacity (*i.e.*, infinite pairwise energy terms) to *impose* a structure to the optimal solution. In Fig. 3(b), downward-pointing arcs ensure that if a pixel is in a source set, the column of pixels below it is also in the source set — so the optimal solution has to be a layer. The slanted arcs impose the smoothness of this layer.

It is also possible to formulate multi-label/multi-object segmentation problems that can be solved with a single s-t cut [22, 50, 57, 74], or by iteratively changing and computing the cut [8, 49]. For the single-cut Ishikawa method [50], it is common to duplicate the graph for each label, *i.e.*, having a sub-graph per label. For example, in Fig. 3, each pixel is represented by two nodes: one for object A and one for object B, so a pixel may be segmented as belonging to A, B, both, or neither. The submodular interaction between the objects may be achieved by adding arcs between the sub-graphs. Fig. 3(c) shows submodular interaction, where arcs with infinite capacity ensure that if a pixel belongs to object A, this pixel and all its neighbors also belong to object B. This is known as *inclusion* or *containment* with a minimum margin of one.

In the examples covered so far, the arcs between the graph nodes correspond to submodular energy terms, which means the energy is lower when the nodes belong to the same set (S or T). Mutual exclusion, in the general case, requires non-submodular energies which are not directly translatable to arcs in the graphs shown so far. An alternative is to use QPBO [63], as illustrated in Fig. 3(d), which can handle any energy function of the form in (1) — submodular or not. When using QPBO, we construct two sub-graphs for each object: one representing the object and another representing its complement. The exclusion of two objects, say A and B, is then achieved by adding inclusion arcs from A to $\overline{B}$ and from B to $\overline{A}$. However, there is no guarantee that the min-cut/max-flow solution yields a complete segmentation of the object as the object and its complement may disagree on the labeling of some nodes leaving them "unlabeled". The number of unlabeled nodes depends on the non-submodularity of the system. Extensions to QPBO, such as QPBO-P and QPBO-I [88], may be used to iteratively assign labels to the nodes that QPBO failed to label.

3 Serial Min-Cut/Max-Flow Algorithms

All min-cut/max-flow algorithms find the solution by iteratively updating a flow that satisfies the capacity constraints. Such a flow induces a *residual graph* with the set of *residual arcs*, R, given by

$$R = \{(i,j) \in V \times V \mid (i,j) \in E, f_{ij} < c_{ij} \text{ or } (j,i) \in E, f_{ji} > 0\}. \quad (4)$$

Each of the residual arcs has a *residual capacity* given by $c'_{ij} = c_{ij} - f_{ij}$ if $(i,j) \in E$ or $c'_{ij} = f_{ji}$ if $(j,i) \in E$. In other words, residual arcs tell us how much flow on the original arc we can increase or decrease, see Fig. 1(c). If the graph contains bidirectional arcs, both conditions from (4) may be met, and the residual capacity then equals the sum of two contributions.

Serial min-cut/max-flow algorithms can be divided into three families: augmenting paths, preflow push-relabel, and pseodoflow algorithms. In this section, we provide an overview of how algorithms from each family work.

3.1 Augmenting Paths

The augmenting paths (AP) family of min-cut/max-flow algorithms is the oldest of the three families and was introduced with the Ford-Fulkerson algorithm [28]. An algorithm from the AP family always maintains a feasible flow. It works by repeatedly finding so-called *augmenting paths*, which are paths from s to t in the residual graph. When an augmenting path is found, a flow is pushed along the path. Pushing flow means increasing flow for each forward arc along the path, and decreasing flow for each reverse arc. To maintain the capacity constraints, the flow that is pushed equals the minimum residual capacity along the path. Conservation constrains are maintained as the algorithm only updates complete paths from s to t. The algorithm terminates when no augmenting paths can be found. Fig. 1(d) shows an intermediate stage of an AP algorithm.

The primary difference between various AP algorithms lies in how the augmenting paths are found. For computer vision applications, the most popular AP algorithm is the Boykov-Kolmogorov (BK) algorithm [11], which works by building search trees from both the source and sink nodes to find augmenting paths and uses a heuristic that favors shorter augmenting paths. The BK algorithm performs well on many computer vision problems, but its theoretical run time bound is worse than other algorithms [95].

In terms of performance, the BK algorithm has been surpassed by the Incremetal Breadth First Search (IBFS) algorithm by Goldberg *et al.* [37]. The main difference between the two algorithms is that IBFS maintains the source and sink search trees as breadth-first search trees, which results in both better theoretical run time and better practical performance [36, 37].

3.2 Preflow Push-Relabel

The second family of algorithms are the preflow push-relabel (PPR) algorithms, which were introduced by Goldberg and Tarjan [35]. These algorithms use a so-called *preflow*, which satisfies capacity constraints but allows nodes to have more incoming than outgoing flow, thus violating conservation constraints. The difference between the incoming and outgoing flows for a node, i, is denoted as its *excess*, $e_i \geq 0$.

The PPR algorithms work by repeatedly pushing flow along the individual arcs. To determine which arcs admit flow, the algorithms maintain an integer labeling (so-called *height*), d_i, for every node. The labeling provides a lower bound on the distance from the node to the sink and has a *no steep drop* property, meaning $d(i) - d(j) \leq 1$ for any residual arc (i,j).

An algorithm from the PPR family starts by saturating the source arcs and raising the source to $d(s) = n$. The algorithm then works by repeatedly selecting a node with excess (after selection called a *selected* node) and applying one of two actions [19, 35]: *push* or *relabel*. If there is an arc in the residual graph leading from the selected node to a lower-labeled node, *push* is performed. This pushes excess along the arc, until all excess is pushed or the arc is saturated. If no residual arc leads to a lower node, the *relabel* operation is used to lift the selected node (increase its label) by one. Fig. 1(e) shows an intermediate step of a PPR algorithm.

When there are no nodes with excess left, the preflow is the maximum flow. It is possible to terminate the algorithm earlier, when no nodes with excess have a label $d_i < n$. At this point, the minimum s-t cut can be extracted by inspecting the node labels. If $d_i \geq n$, then $i \in S$, otherwise $i \in T$. Extracting the maximum flow requires an extra step of pushing all excess back to the source. However, this work generally only represents a small part of the run time [95] and, for computer vision applications, we are typically only interested in the minimum cut anyway.

The difference between various PPR algorithms lies in the order in which the push and relabel operations are performed. Early variants used simple heuristics, such as always pushing flow from the node with the highest label or using a first-in-first-out queue to keep track of nodes with positive excess [17]. More recent versions [3, 33, 34] use sophisticated heuristics and a mix of local and global operations to obtain significant performance improvements over early PPR algorithms.

Unlike other serial algorithms, the algorithms from the PPR family operate locally on nodes and arcs. This, as we shall discuss later, has resulted in a whole family of parallel PPR algorithms.

3.3 Pseudoflow

The most recent family of min-cut/max-flow algorithms is the pseudoflow family, which was introduced with the Hochbaum pseudoflow (HPF) algorithm [45, 46]. These algorithms use a so-called *pseudoflow*, which satisfies capacity constraints but not the conservation constraints, as it has no constraints on the difference between incoming and outgoing flow. As with preflow, we refer to the difference between incoming and outgoing flow for a node as its excess, e_i. A positive excess is referred to as a *surplus* and a negative excess as a *deficit*.

During operation, HPF algorithms maintain two auxiliary structures: the forest of trees and a node labeling function. Only one node in every tree, the root, is allowed to have an excess. The algorithm works by repeatedly pushing the flow along the paths connecting the trees, and growing the trees.

A generic algorithm from the HPF family is initialized by saturating all terminal arcs. At this point, each graph node is a singleton tree in the forest. The algorithm then selects a tree with surplus and containing at least one node with the label less than n (the number of nodes in the graph). In this tree, i denotes the node with the lowest label. If there are no residual arcs from i to a node with a lower label, the label of i is incremented. If there is a residual arc (i, j) that leads to a node j with a lower label, a *merge* is performed. This operation involves pushing surplus along the path from the root of the tree containing i, over i, over j, and to the root of the tree containing j. If the arc capacities along this path allow it, the entire surplus will be pushed and the trees will be merged with j as the root. If the flow along the path saturates an arc (i', j'), a surplus will be collected in i', and a new tree rooted in i' will be created. In contrast to the AP algorithms, the only restrictions on how much flow to push are the individual arc capacities, not the path capacity.

The algorithm terminates when no selection can be made, at which point nodes labeled with n constitute the source set. Additional processing is needed to recover the maximum feasible flow. Fig. 1(f) shows an intermediate step of the HPF algorithm.

There are two main algorithms in this family: HPF and Excesses Incremental Breadth First Search (EIBFS) [36]. The main differences are the order in which they scan through nodes when looking for an arc connecting two trees in the forest, and how they push flow along the paths. Both have sophisticated heuristics for these choices, which makes use of many of the same ideas developed for PPR algorithms.

3.4 Implementation Details

As stressed by [95], the implementation details can significantly affect the measured performance of a given min-cut/max-flow algorithm. In this section, we will highlight the trends of modern implementations and how they differ.

3.4.1 Data Structures and Data Types

The implementations considered in this paper all use a variant of the adjacency list structure [21] to represent the underlying graph. The most common setup mimics the BK algorithm: there is a list of nodes and a list of directed (half-)arcs. Each `Node` structure stores a pointer to its first outgoing half-arc. Each `Arc` stores a pointer to the node it points to, a pointer to the next outgoing arc for the node it points from, a pointer to its reverse arc, and a residual capacity. For algorithms implemented with computer vision applications in mind (*e.g.*, BK, IBFS, and EIBFS), the terminal arcs are stored as a single combined terminal capacity for each `Node`, instead of using the `Arc` structures. Other implementations simply keep track of the source and sink nodes and use `Arc` structures for all arcs. The HPF implementation uses a bidirectional `Arc` structure with a capacity, a flow, and a direction. It is also common to store auxiliary values such as excesses, labels, or more.

As a result of these differences, the memory footprint varies between implementations, as shown in Table 1. The footprint also depends heavily on the data types used to store the data, in particular references to nodes and arcs, as we discuss in the next subsection. For storing arc capacities, integers are common because they are computationally efficient and may use as little as 1 byte. However, some graph constructions involve large capacities to model hard constraints, and here some care must be taken to avoid overflow issues. With floats, this can be modeled using infinite capacity. However, floats are less efficient and some algorithms are not guaranteed to terminate with floats due to numerical errors.

As the size of the data structures influences how much the CPU can store in its caches, which has a large effect on performance, it is generally beneficial to keep the data structures small. Note that some compilers do not pack data structures densely by default, which may significantly increase the size of the `Arc` and `Node` data structures.

3.4.2 Indices vs. Pointers

One way to reduce the size of the `Arc` and `Node` data structures on 64-bit system architectures is to use indices instead of pointers to reference nodes and arcs. As long as the indices can be stored using unsigned 32-bit integers, we can halve the size arc and node references by using unsigned 32-bit integers instead of pointers (which are 64-bit). This approach can significantly reduce the size of the `Arc` and `Node` data structures, as the majority of the structures consist of references to other arcs and nodes [52]. As the performance of min-cut/max-flow algorithms is mainly limited by memory speed, smaller data structures can often lead to improved performance. The downside of indices is that extra computations may be needed for every look-up, although this depends on the exact assembly instructions the compiler chooses to use.

Some grid-based algorithms [52] use 32-bit indices to reduce the size of their data structure. The generic algorithms we have investigated in this work all use pointers to store references between nodes and arcs. Some implementations avoid the extra memory requirement by compiling with 32-bit pointers. However, 32-bit pointers limit the size of the graph much more than 32-bit indices. The reason is that the 32-bit pointers only have 4 GiB of address space, and the `Node` and `Arc` structures they point to take up many bytes. For example, the smallest `Arc` structure we have tested, *c.f.* Table 1, uses 24 bytes, meaning that an implementation based on 32-bit *indices* could handle graphs with 24 times more arcs than an implementation based on 32-bit *pointers*.

3.4.3 Arc Packing

The order in which the arcs are stored may significantly affect performance. *Arc packing* is used to reduce CPU cache misses by storing the arcs in the same order that the algorithm will access them. For example, min-cut/max-flow algorithms often iterate over outgoing arcs from a node, making it beneficial to store outgoing

arcs from the same node adjacent in memory. However, as arcs may be added to the graph in any order, packing the arcs usually incurs an overhead from maintaining the correct ordering or reordering all arcs as an extra step before computing the min-cut/max-flow. Similar to arc packing, *node packing* may improve performance. However, this is not done in practice as opposed to arc packing.

Of the serial reference implementations that we examined, only HI-PR [19], IBFS, and EIBFS implement arc packing. These all implement it as an extra step, where arcs are reordered after building the graph but before the min-cut/max-flow computations start. None of the examined implementations use node packing.

3.4.4 Arc Merging

In practice, it is not uncommon that multiple arcs between the same pair of nodes are added to the graph. Merging these arcs into a single arc with a capacity equal to the sum of capacities of the merged arcs may reduce the graph size significantly. As this decreases both the memory footprint of the graph and the number of arcs to be processed, it can provide substantial performance benefits [52, 89]. However, as redundant arcs can usually be avoided by careful graph construction and they should have approximately the same performance impact on all algorithms, we have not investigated the effects of this further.

4 Parallel Min-Cut/Max-Flow

Like serial algorithms, parallel algorithms for min-cut/max-flow problems can be split into families based on shared characteristics. A key characteristic is whether the algorithms parallelize over individual graph nodes (node-based parallelism) or split the graph into sub-graphs that are then processed in parallel (block-based parallelism). Other important algorithmic traits include whether the algorithm is distributed, which we do not consider in this paper, and the guarantees in terms of convergence, optimality, and completeness provided by the algorithm.

We should note that since many (but not all) min-cut/max-flow problems in computer vision are defined on grid graphs, several algorithms [51, 52, 86, 96] have exploited this structure to create very efficient parallel implementations. However, many important computer vision problems are not defined on grid graphs, so in this paper we focus on generic min-cut/max-flow algorithms.

The category of node-based parallel algorithms is generally dominated by parallel versions of PPR algorithms. In the block-based category, we have identified three main approaches: adaptive bottom-up merging, dual decomposition, and region discharge, which we investigate. In the following sections, we give an overview of each approach and briefly discuss its merits and limitations.

4.1 Parallel Preflow Push-Relabel

PPR algorithms have been the target of most parallelization efforts [2, 4, 5, 22, 32, 47, 96], since both push and relabel are local operations, which makes them well suited for parallelization. Because the operations are local, the algorithms generally parallelize over each node — performing pushes and relabels concurrently. To avoid data races during these operations, PPR algorithms use either locking [2] or atomic operations [47]. As new excesses are created, the corresponding nodes are added to a queue from which threads can poll them. In [5], a different approach is applied, where pushes are performed in parallel, but excesses and labels are updated later in a separate step, rather than immediately after the push.

Since parallel PPR algorithms parallelize over every node, they can achieve good speed-ups and scale well to modern multi-core processors [5], or even GPUs [96]. However, these algorithms have not achieved dominance outside of large grid graphs for min-cut/max-flow problems [103]. Since GPU hardware has advanced considerably in recent years, it is unclear whether GPU method should remain restricted to grid graphs, but this question is not within the scope of this paper.

4.2 Adaptive Bottom-Up Merging

The adaptive bottom-up merging approach introduced by Liu and Sun [75] uses block-based parallelism and has two phases, which are summarized in Fig. 4. In phase one, the graph is partitioned into a number of disjoint sets (blocks), and arcs between blocks have their capacities set to 0 — effectively removing them from the graph. For each pair of blocks connected by arcs, we store a list of the connecting arcs (with capacities now set to 0) along with their original capacities. Disregarding s and t, the nodes in each block now belong to disjoint sub-graphs and we can compute the min-cut/max-flow solution for each sub-graph in parallel. The min-cut/max-flow computations are done with the BK algorithm — although one could in theory use any min-cut/max-flow algorithm.

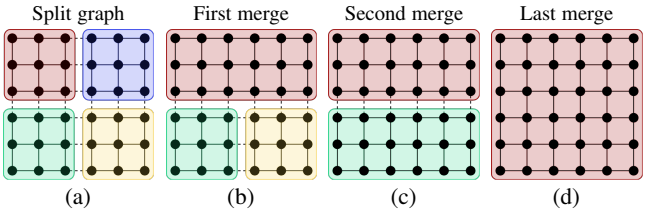


Fig. 4: **Illustration of the adaptive bottom-up merging approach for parallel min-cut/max-flow.** Terminal nodes and arcs are not shown. Note that the underlying graph does *not* have to be a grid graph. Phase one: (a) The graph is split into blocks and the min-cut/max-flow is computed for each block in parallel. Phase two: (b) The topmost blocks are locked, merged, and the min-cut/max-flow recomputed. (c) As the topmost block is locked, the next thread works on the bottom-most blocks (in parallel). (d) Last two blocks are merged and min-cut/max-flow recomputed to achieve the globally optimal solution.

In phase two, we merge the blocks to obtain the complete globally optimal min-cut/max-flow. To merge two blocks, we restore the arc capacities for the connecting arcs and then recompute the min-cut/max-flow for the combined graph. This step makes use of the fact that the BK algorithm can efficiently recompute the min-cut/max-flow when small changes are made to the residual graph for a min-cut/max-flow solution [62].

For merges in phase two to be performed in parallel, the method marks the blocks being merged as locked. The computational threads then scan the list of block pairs, which were originally connected by arcs, until they find a pair of unlocked blocks. The thread then locks both blocks, performs the merge, and unlocks the new combined block. To avoid two threads trying to lock the same block, a global lock prevents more than one thread from scanning the list of block pairs at a time.

As the degree of parallelism decreases towards the end of phase two — since there are few blocks left to merge — performance increases when computationally expensive merges are performed early in phase two. To estimate the cost of merging two blocks, [75]

uses a heuristic based on the potential for new augmenting paths to be formed by merging two blocks. This heuristic determines the merging order of the blocks.

By using block-based, rather than node-based parallelism, adaptive bottom-up merging avoids much of the synchronization overhead that the parallel PPR algorithms suffer from. However, its performance depends on the majority of the work being performed in phase one and in the beginning of phase two, where the degree of parallelism is high.

4.3 Dual Decomposition

The dual decomposition (DD) approach was introduced by Strandmark and Kahl [91] and later refined by Yu et al. [103]. The approach was originally designed to allow for distributed computing, such that it is never necessary to keep the full graph in memory. Their algorithm works as follows: first, the nodes of the graph are divided into a set of overlapping blocks (see Fig. 5(a)). The graph is then split into disjoint blocks, where the nodes in the overlapping regions are duplicated in each block (see Fig. 5(b)). It is important that the blocks overlap such that if node i is connected to node j in block b_j and node k in block b_k, then i is also in both blocks b_j and b_k.

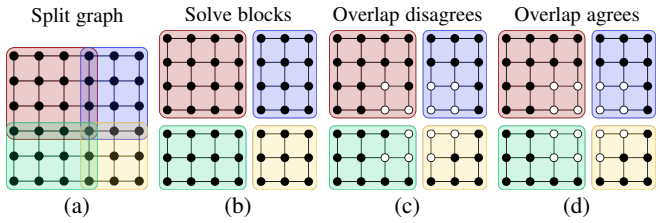


Fig. 5: **Illustration of the dual decomposition approach.** Terminal nodes and arcs are not shown. Note that the underlying graph does *not* have to be a grid graph. (a) Graph nodes are divided into a set of overlapping blocks. (b) The graph is split into disjoint subgraphs and nodes in overlapping regions are duplicated into each blocks. (c) The min-cut/max-flow for each block is computed in parallel which gives an assignment to source set (black) or sink set (white). The source/sink capacities are then adjusted for disagreeing duplicated nodes. (d) The min-cut/max-flow is recomputed and capacities are adjusted until all duplicated nodes agree.

Once the graph has been partitioned into overlapping blocks, the algorithm proceeds iteratively. First, the min-cut/max-flow for each disjoint block is computed in parallel using the BK algorithm. Next, for each duplicated node, it is checked if all duplicates of that node are in the same *s-t* partitioned set, S or T. In that case, we say that the node duplicates agree on their assignment. If all duplicated nodes agree on their assignment, the computed solution is globally optimal and the algorithm terminates. If not, the terminal arc capacities for the disagreeing duplicated nodes are updated according to a supergradient[1] ascent scheme and the min-cut/max-flow is recomputed. This process of updating terminal capacities and recomputing the min-cut/max-flow is repeated until all duplicated nodes agree on their assignment.

A limitation of the original dual decomposition approach is that convergence is not guaranteed. Furthermore, [103] and [89] have demonstrated that the risk of nonconvergence increases as the graph is split into more blocks. To overcome this, Yu et al. [103] introduced a new version with a simple strategy that guarantees convergence: if the duplicated nodes in two blocks do not belong to the same set, S or T, after a fixed number of iterations, the blocks are merged and the algorithm continues. This trivially guarantees convergence since, in the worst case, all blocks will be merged, at which point the global solution will be computed serially. However, performance significantly drops when merging is needed for the algorithm to converge, as merging only happens after a fixed number of iterations and all blocks may (in the worst case) have to be merged for convergence.

1. Analogous to subgradients for convex functions [7].

4.4 Region Discharge

The region discharge (RD) approach was introduced by Delong and Boykov [22] and later generalized by Shekhovtsov and Hlaváč [89]. The idea builds on the vertex discharge operation introduced for PPR in [35]. Similarly to DD by Strandmark and Kahl, RD was designed to allow for distributed computing. The method first partitions the graph into a set of blocks (called regions in [89] following the terminology of [22]). Each block R has an associated *boundary* defined as the set of nodes

$$B^R = \{v \in V \mid v \notin R, (u, v) \in E, u \in R, v \neq s, t\}. \quad (5)$$

Capacities for arcs going from a boundary node to a block node are set to zero. This means that flow can be pushed *out* of the block into the boundary, but not vice versa. Furthermore, each node is allowed to have an excess.

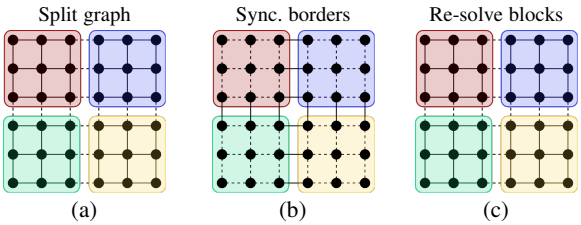


Fig. 6: **Illustration of the region discharge approach.** Terminal nodes and arcs are not shown. Note that the underlying graph does *not* have to be a grid graph. (a) Graph nodes are divided into a set of blocks and the region discharge operation is run on each block, which pushes flow to the sink or boundary. (b) Flow is synchronized between boundaries. (c) Region discharge is run again. The process repeats until no flow crosses the block boundaries.

The method then performs the *region discharge* operation, which aims to push as much excess flow to the sink and/or the boundary nodes as possible (the source, s, is assumed to have infinite excess). This has been done with a PPR [22, 89] or an AP algorithm (specifically BK) [89]. When using a PPR algorithm, the discharge of a block is done by performing only push and relabel operations between nodes in the same block.

When using the BK algorithm, a distance labeling is maintained for the boundary nodes which gives an estimate of how many boundaries must be crossed to reach the sink. Initially, in each block, flow is pushed exclusively to the sink. Then, flow is pushed to the boundary nodes with distance labels less than 1, then less than 2, etc., until no more flow can be pushed. The BK implementation used by Shekhovtsov and Hlaváč has been slightly modified to allow excess in the boundary nodes and for flow to be pushed from the boundary nodes out of the block (but not back).

The discharge operation is performed on all blocks in parallel. Afterward, flow along boundary arcs is synchronized between

neighboring blocks. This may create additional excesses in some blocks, since boundary nodes overlap with another block. The discharge and synchronization process is repeated until no new excesses are created, at which point the algorithm terminates. It is proved in [89] that this process terminates in at most $2n^2$ iterations of discharge and synchronization when using PPR and $2n_B^2 + 1$ when using AP, where n_B is the total number of boundary nodes.

The guarantee of convergence, without having to merge blocks, is beneficial, as it means that the algorithm can maintain a high degree of parallelism while computing the min-cut/max-flow solution. However, because flow must be synchronized between blocks, the practical performance of the method still depends on well-chosen blocks and may be limited by synchronization overhead. For details on the heuristics used in the algorithm, which are also important for its practical performance, see [89].

5 Performance Comparison

We now compare the performance of the algorithms discussed in the previous sections. For all experiments, the source code was compiled with the GCC C++ compiler version 9.2.0 with `-O3` optimizations on a 64-bit Linux-based operating system with kernel release 3.10. Experiments were run on a dual socket NUMA (Non-Uniform Memory Access) system with two Intel Xeon Gold 6226R processors with 16 cores each and HTT (Hyper-Threading Technology) disabled, for a total of 32 parallel CPU threads. The system has 756 GB of RAM, and for all experiments all data were kept in memory. All resources were provided by the DTU Computing Center [23].

For all parallel benchmarks, we prefer local CPU core and memory allocation. This means that for all parallel benchmarks with up to 16 threads, all cores are allocated on the same CPU/NUMA node. If the data fits in the local memory of the active node, we use this memory exclusively. If the data cannot fit in the local memory of one node, memory of both NUMA nodes is used. For benchmarks with more than 16 threads, both CPUs and their memory pools are used.

Run time was measured as the minimum time over three runs and no other processes (apart from the OS) were running during the benchmarks. We split our measured run time into two distinct phases: build time and solve time. Build time refers to the construction of the graph and any additional data structures used by an algorithm. If the algorithm performs arc packing or similar steps, this is included in the build time. To ensure that the build time is a fair representation of the time used by a given algorithm, we precompute a list of nodes and arcs and load these lists fully into memory before starting the timer. Solve time refers to the time required to compute the min-cut/max-flow. For the pseudoflow, PPR, and region discharge algorithms (*c.f.* Table 1), that only compute a minimum cut, we do not include the time to extract the full feasible maximum flow solution. The reason for this is that for most computer vision applications the minimum cut is of principal interest. Furthermore, converting to a maximum flow solution usually only adds a small overhead [95].

5.1 Datasets

We test the algorithms on the following benchmark datasets:

(1) The commonly used University of Waterloo [98] benchmarks problems. Specifically, we use 6 stereo [14, 64], 36 3D voxel segmentation [9, 12, 10], 2 multi-view reconstruction [72, 13], and 1 surface fitting [71] problems.
(2) The 4 super resolution [30, 88], 4 texture restoration [88], 2 deconvolution [88], 78 decision tree field (DTF) [82], and 3 automatic labelling environment (ALE) [25, 67, 68, 69] datasets from Verma's and Batra's survey [95].
(3) New problems that use anisotropic MRFs [38] to segment blood vessels in large voxel volumes from [87]. We include 3 problems where the segmentation is applied directly to the image data and 3 to the output of a trained V-Net [79].
(4) New problems that use MRFs to clean 3D U-Net [20] segmentations of prostate images from [90]. We contribute 4 benchmark problems.
(5) New problems on mesh segmentation based on [76]. We contribute 8 benchmark problems. The original paper uses α-expansion and $\alpha\beta$-swaps [9, 11] to handle the multi-class segmentation problem. For our benchmarks, we instead use QPBO to obtain the segmentation with a single min-cut, which may lead to different results compared with the referenced method.
(6) New problems using the recent Deep LOGISMOS [42] to segment prostate images from [90]. We contribute 8 problems.
(7) New problems performing multi-object image segmentation via surface fitting from two recent papers [53, 57]. We contribute 9 problems using [53] and 8 using [57].
(8) New problems performing graph matching from the recent paper [48]. The original matching problems can be found at https://vislearn.github.io/libmpopt/iccv2021. For each matching several QPBO sub-problems are solved. We contribute the QPBO subproblems (300 per matching problem) for each of the 316 matching problems.

In total, our benchmark includes 495 problems covering a variety of different computer vision applications. Note that some datasets consist of many small sub-problems that must be run in sequence. Here, we report the accumulated times. All the benchmark problems are available at: DOI:10.11583/DTU.17091101 [55].

For the parallel algorithm benchmarks, we only include a subset of all datasets. This is because parallelization is mainly of interest for large problems with long solve times. For the block-based algorithms, we split the graph into blocks in one of the following ways: For graphs based on an underlying image grid, we define blocks by recursively splitting the image grid along its longest axis. For the surface-based segmentation methods [53, 57], we define blocks such that nodes associated with a surface are in their own block. For mesh segmentation, we compute the geodesic distance between face centers and then use agglomerative clustering to divide the nodes associated with each face into blocks. For bottom-up merging, we use 64 blocks for the following dataset: the grid graphs, the mesh segmentation, and the cells, foam, and simcells. For the NT32_tomo data we use two blocks per object. For 4Dpipe we use a block per 2D slice. For P-GridCut we use the same blocks as for bottom-up merging. For dual decomposition and region discharge, we use one and two blocks per thread, respectively.

5.2 Tested Implementations

All tested implementations (except GridCut [51]) are available at https://github.com/patmjen/maxflow_algorithms and are archived at DOI:10.5281/zenodo.4903945 [54]. Beware that the implementations are published under different licenses — some open and some restrictive. See the links above for more information.

In the following, `typewriter` font refers to a specific implementation of a given algorithm. We use this for BK and

EIBFS, where we test more that one implementation of each algorithm, *e.g.*, BK refers to the algorithm, `BK` is the reference implementation, and `MBK` is one of our implementations.

BK [11] We test the reference implementation (`BK`) of the Boykov-Kolmogorov algorithm from http://pub.ist.ac.at/~vnk/software.html. Furthermore, we test our own implementation of BK (`MBK`), which contains several optimizations. Most notably, our version uses indices instead of pointers to reduce the memory footprint of the `Node` and `Arc` data structures. Finally, we test a second version (`MBK-R`), which reorders arcs so that all outgoing arcs from a node are adjacent in memory. This increases cache efficiency, but uses more memory (see Table 1) and requires an extra initialization step. The memory overhead from reordering could be reduced by ordering the arcs in-place; however, this may negatively impact performance. Therefore, we opt for the same sorting strategy as `EIBFS`, where arcs are copied during reordering.

EIBFS [36] We test a slightly modified version [49] (`EIBFS`) of the excesses incremental breadth first search algorithm originally implemented by [36] available from https://github.com/sydbarrett/AlphaPathMoves. This version uses slightly larger data structures to support non-integer arc capacities and larger graphs, compared to the implementation tested in [36]. Although these changes may slightly decrease performance, we think it is reasonable to use the modified version, as several of the other algorithms have made similar sacrifices in terms of performance. Additionally, we test our own modified version of `EIBFS` (`EIBFS-I`), which replaces pointers with indices to reduce the memory footprint. Finally, since both `EIBFS` and `EIBFS-I` perform arc reordering during initialization, we also test a version without arc reordering (`EIBFS-I-NR`) to better compare with other algorithms.

HPF [45] We test the reference implementation of Hochbaum pseudoflow (HPF) from https://riot.ieor.berkeley.edu/Applications/Pseudoflow/maxflow.html. This implementation has four different configurations that we test:

1) Highest label with FIFO buckets (`HPF-H-F`).
2) Highest label with LIFO buckets (`HPF-H-L`).
3) Lowest label with FIFO buckets (`HPF-L-F`).
4) Lowest label with LIFO buckets (`HPF-L-L`).

HI-PR [19] We test the implementation of the preflow push-relabel algorithm from https://cmp.felk.cvut.cz/~shekhovt/d_maxflow/index.html[2].

P-ARD [89] We test the implementation of parallel augmenting paths region discharge (P-ARD) from https://cmp.felk.cvut.cz/~shekhovt/d_maxflow/index.html. P-ARD is an example of the region discharge approach. It uses `BK` as the base solver. Note that, as the implementation is designed for distributed computing, it makes use of disk storage during initialization, which increases the build time.

Liu-Sun [75] Since no public reference implementation is available, we test our own implementation of the adaptive bottom-up merging approach based on the paper by Liu and Sun [75]. Our implementation uses `MBK` as the base solver.

P-PPR [5] We test the implementation of a recent parallel preflow push-relabel algorithm from https://github.com/niklasb/pbbs-maxflow.

Strandmark-Kahl [91] We test our own implementation of the Strandmark-Kahl dual decomposition algorithm based on the implementation at https://cmp.felk.cvut.cz/~shekhovt/d_maxflow/index.html[3]. The original implementation can only handle grid graphs with rectangular blocks, while our implementation can handle arbitrary graphs and arbitrary blocks at the cost of some additional overhead during graph construction. Our implementation uses `MBK` as the base solver. Note that our implementation does not implement the merging strategy proposed by [103] and, therefore, is not guaranteed to converge. We only include results for cases where the algorithm does converge.

GridCut [51, 52] We test both the serial and parallel versions of the highly optimized commercial GridCut implementation from https://gridcut.com. The primary goal is to show how much performance can be gained by using an implementation optimized for grid graphs. GridCut is only tested on problems with graph structures that are supported by the reference implementation, *i.e.*, 4- and 8-connected neighbor grids in 2D, and 6- and 26-connected (serial only) neighbor grids in 3D.

Table 1 lists the tested implementations along with their type and memory footprint. Their memory footprint can be calculated based on the number of nodes and arcs in the graph and will be discussed further in Section 7.

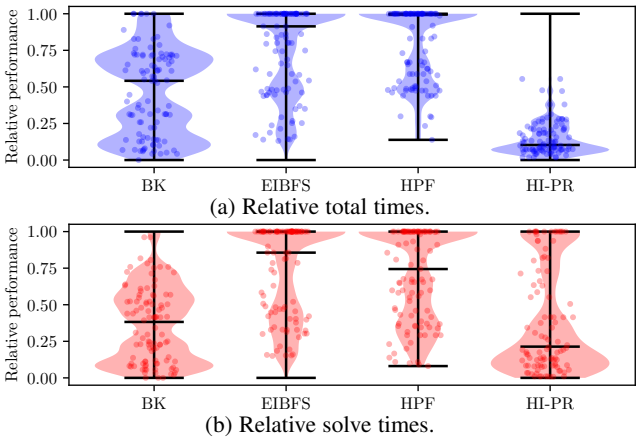
TABLE 1: **Summary of the tested implementations including their memory footprint**. The table shows the bytes required as a function of the number of nodes, n, number of terminal arcs, m_T, and number of neighbor arcs, m_N. We assume the common case of 32-bit capacities and 32-bit indices, which is also what we use for all of our experiments. Since HPF stores undirected arcs, we give all sizes as undirected arcs, *i.e.*, for implementations using directed arcs the size per arc reported here is doubled. Note that the numbers depend on, but are *not* the same as, the `Node` and `Arc` structure sizes, as the footprint reported includes all stored data (connectivity, capacity, and any auxiliary data).

Serial algorithms	Algorithm type	Memory footprint
HI-PR[a] [19]	Preflow push-relabel	$40n + 40m_T + 40m_N$
HPF[b] [45]	Pseudoflow	$104n + 48m_T + 48m_N$
EIBFS [36]	Pseudoflow	
↳`EIBFS`[c]	Pseudoflow	$72n + 72m_N$
↳`EIBFS-I`*[i]	Pseudoflow	$29n + 50m_N$
↳`EIBFS-I-NR`*[i]	Pseudoflow	$49n + 24m_N$
BK [11]	Augmenting path	
↳`BK`[d]	Augmenting path	$48n + 64m_N$
↳`MBK`*[i]	Augmenting path	$23n + 24m_N$
↳`MBK-R`*[i]	Augmenting path	$23n + 48m_N$
Parallel algorithms		
P-PPR[e][i] [5]	Parallel PPR	$48n + 68m_T + 68m_N$
Liu-Sun*[i] [75]	Ada. bot.-up merging†	$25n + 24m_N$
Strandmark-Kahl*[i] [91]	Dual decomposition†	$29n + 24m_N$
P-ARD[a] [89]	Region discharge†	$40n + 32m_N$

†Uses BK (augmenting path)
*Implemented or updated by us: https://github.com/patmjen/maxflow_algorithms
[i]Assuming 32-bit indices
[a]https://cmp.felk.cvut.cz/~shekhovt/d_maxflow/index.html
[b]https://riot.ieor.berkeley.edu/Applications/Pseudoflow/maxflow.html
[c]https://github.com/sydbarrett/AlphaPathMoves
[d]http://pub.ist.ac.at/~vnk/software.html
[e]https://github.com/niklasb/pbbs-maxflow

2. Orignally from http://www.avglab.com/andrew/soft.html, but the link is no longer available.

3. Originally from https://www1.maths.lth.se/matematiklth/personal/petter/cppmaxflow.php but the link is no longer available.



(a) Relative total times.

(b) Relative solve times.

Fig. 7: **Relative performance for the serial algorithms.** For each dataset, the solve and total times for each algorithm were compared to those of the fastest algorithm for that dataset and a relative time was computed. This shows how often an algorithm was fastest and, if it was not fastest, how much slower than the fastest it was. We oversample speed-ups from each problem family (*c.f.* Table 2) so all groups have the same number of entries. This is to avoid bias due to some problem groups having more entries than others. Finally, we overlay a random sample of the (oversampled) speed-ups as jittered points.

5.3 Serial Algorithms

The primary experimental results for the serial algorithms are listed in Table 2 and Fig. 7. Table 2 shows a representative subset of the results, grouped by problem family, while Fig. 7 shows the distribution of the solve time and the total time for each algorithm on each dataset relative to the fastest algorithm on the dataset. Thus, for a given dataset, a relative performance score of 0.5 means that the algorithm used double the amount of time as the fastest algorithm on that dataset. The distribution of these scores indicates how well the different algorithms perform relative to each other.

From Fig. 7(b), we see that EIBFS and HPF outperform the other two algorithms on the majority of the datasets in terms of solve time and total time, as the algorithms have most of their relative times close to 1. Looking at the median, EIBFS has a slightly better relative solve time than HPF, while HPF is faster w.r.t. total time. Furthermore, HPF has the best worst-case performance for both solve and total time. However, despite its overall good performance, HPF performs significantly worse on the oriented MRF and U-Net cleaning datasets. The performance of BK varies significantly depending on the benchmark problem. Although it has a median relative total time of just over 0.5, its relative performance is considerably more inconsistent than that of the three other algorithms. It performs particularly poorly on the 4Dpipe datasets, using over 6 hours on 4Dpipe_small, which both HPF and EIBFS completed in less than 30 seconds. For 4Dpipe_large, BK was not able to find the solution within 45 hours. HI-PR generally has the worst performance but does have the fastest solve time for a few datasets. However, measured on total time, it almost never manages a relative score of more than 0.5. It is worth noting that the distribution of relative times for all algorithms exhibits a bimodality. This indicates that all algorithms have datasets where they are poorly suited compared to the others. We further investigate this in Section 6.

5.3.1 *Algorithm Variants*

The different variants of each algorithm are compared in Fig. 8, which shows the relative performance of each implementation compared to a chosen "reference" implementation. For the BK algorithm, the `BK` implementation is used for reference, for the EIBFS algorithm, the `EIBFS` implementation is used as a reference, and for HPF the `HPF-H-F` configuration is used as reference, since it is the one recommended by the authors. As we are now measuring relative to a specific implementation, rather than the fastest implementation as in Fig. 7, it is possible to obtain a relative performance score of more than one.

For the BK algorithm, both `MBK` and `MBK-R` overall perform similarly or slightly better than `BK`, when measured on total time. Looking at solve time, `MBK-R` shows a large speed-up over the other variants. This clearly reflects the effect of arc packing (reordering the arcs), in that it typically decreases solve at the cost of increased build time. From Table 2, we see that `BK` is generally best for smaller problems where the smaller memory footprint of the index-based variants is less of an advantage. However, the very small difference in absolute time for these small problems will in many cases render the choice of algorithm irrelevant.

For the EIBFS variants, the index-based version (`EIBFS-I`) consistently outperforms the reference implementation with a median improvement of more than 20%. Meanwhile, `EIBFS-I-NR` performs worse than `EIBFS` on almost all datasets w.r.t. to solve time, but better w.r.t. total time for the majority of the problems. In some cases, it also outperforms `EIBFS-I`, again showing that while arc packing generally significantly reduces the solve time, the additional overhead is not always worth it.

For the HPF algorithm, `HPF-H-L` consistently performs the best, while `HPF-L-F` and`HPF-L-L` perform worse than the reference `HPF-H-F` for the majority of datasets. However, for some datasets `HPF-L-F` and `HPF-L-L` show large speed-ups over the other variants. Table 2 reveals that the `HPF-L` variants seem to be better for graph matching and ALE datasets.

5.4 Parallel Algorithms

Our benchmark results for the parallel algorithms are shown in Table 3, where we compare the build and solve time for each algorithm on each dataset. The table includes the number of CPU threads used by each algorithm for the listed solve times. Furthermore, it includes the solve time of the best serial algorithm for each dataset for comparison. We focus on the solve time, as that is what reveals how successfully the algorithms distribute the work as more threads are added. Additionally, a lack of optimization leads to very long build times for some of the parallel implementations, especially P-PPR and P-ARD. Finally, some datasets are omitted for P-PPR due to run-time errors and for Strandmark-Kahl due to excessive run time.

From Table 3, it is clear that no algorithm is dominant, except P-GridCut for 6-connected grid graphs. Every algorithm has datasets where it is the fastest and a serial algorithm often gives the best or close to the best performance. The parallel algorithms show their strength for the large datasets with more than 1 M nodes where significant performance improvements are found. Curiously, only P-ARD shows a significant speed-up for smaller problems.

The parallel benchmarks are also summarized in Fig. 9. All algorithms have median speed-ups less than one. Liu-Sun generally performs best, giving a speed-up for almost half of the dataset and having the largest maximum speed-up. P-PPR and P-ARD still

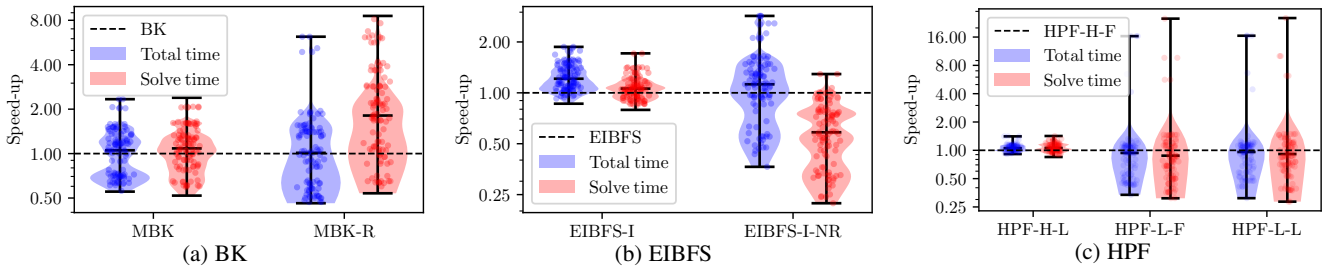


Fig. 8: **Performance comparison of serial algorithm variants.** The solve time and total time is compared against the times for the chosen reference algorithm for each dataset. The violin plots show a Gaussian kernel density estimate of the data and the horizontal bars indicate — from top to bottom — the maximum, median, and minimum. The values were re-sampled as described in Fig. 7.

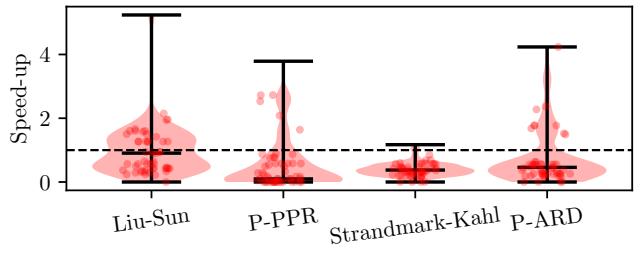


Fig. 9: Speed-up of the parallel algorithms relative to the best serial solve time for each dataset. The values were re-sampled as described in Fig. 7.

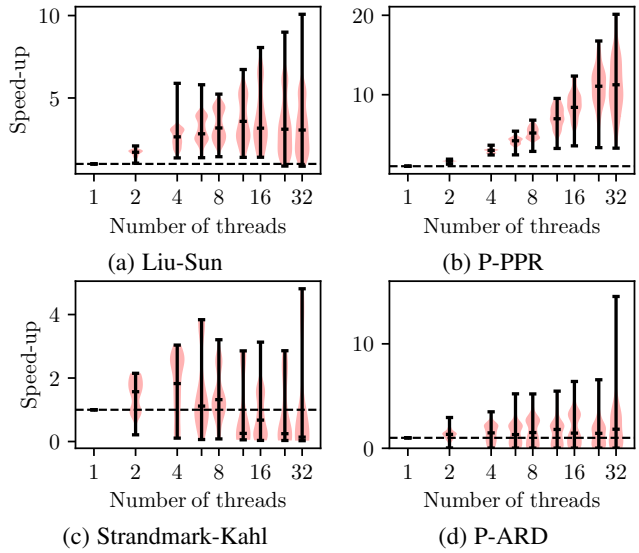


Fig. 10: Speed-up of the the parallel algorithms compared to their single-threaded performance. For each number of threads, the distribution of the speed-ups over all datasets is shown. The values were re-sampled as described in Fig. 7.

provide good speed-ups for some datasets. Strandmark-Kahl comes off the worst, as it rarely beats the best serial algorithm.

Finally, Fig. 10 shows the speed-up distribution of the parallel algorithms compared to their single-threaded performance. Only P-PPR improves consistently as more threads are added. Liu-Sun and P-ARD only show consistent improvements when looking at the maximum speed-up, and for over half of the datasets they have issues scaling beyond 12 threads.

6 Algorithm Selection

As the previous section shows, the performance of the individual min-cut/max-flow algorithms depends on the problem to be solved, *i.e.*, the structure of the graph. Choosing the wrong algorithm may significantly increase the run time. In this section, we investigate strategies for selecting a min-cut/max-flow algorithm that maximize the expected performance given different levels of knowledge about the graph. To quantify the expected performance of a strategy, we will use the relative performance (RP), which we compute as follows: 1. Use the strategy to select an algorithm for each dataset. 2. For each dataset, compute the relative performance of the selected algorithm. For serial algorithms, this is the total time of the selected algorithm divided by the total time of the fastest algorithm for that dataset. For parallel algorithms, we use the solve time. This score shows the expected performance of a given strategy compared to choosing the fastest algorithm.

Scenario 1: No Graph Knowledge If one has no knowledge of the graph to be solved, the best strategy is to choose the overall best algorithm. Table 4 shows summary statistics for the performance scores of each algorithm. To avoid bias in Fig. 8, we oversample scores from each problem family so that they all have the same number of samples.

For the serial algorithms, the best choice is by far GridCut if it is applicable. It is almost always the fastest option and never more than 36% slower than the best option. Otherwise, the best option is `HPF-H-L` in which case the expected performance 64% of the optimal. Another good option is `EIBFS-I` due to its high mean and high minimum RP scores. All implementations, except `EIBFS`, have a maximum RP of 1, meaning that they outperformed all other implementations on at least one problem instance.

For the parallel algorithms, GridCut again dominates when applicable. Otherwise, the best parallel option is Liu-Sun which is slightly better than P-PPR. Surprisingly, using the best serial algorithm for a dataset is the overall best option, although we should note that comparing to the *best* serial algorithm gives some advantage to the serial algorithms. If one compares to a single serial algorithm, the parallel algorithms do give an improvement — although the mean RP is only 1.8x higher in the best case.

Scenario 2: Known Problem Family If one knows from which problem family the graph to be solved comes, a good strategy is to select the algorithm that performs well on that problem family. This could, for example, be established beforehand by running a set of benchmarks on example graphs.

Table 5 shows the best performing serial algorithm for each problem family. Note that, as opposed to Table 2, we split graph matching into sub-groups as papers use different energy functions

TABLE 2: **Performance comparison of serial algorithms** based on both their solve and total (build + solve) times. We show a representative subset of the datasets, which have been grouped according to their problem family. For each problem family we only show the fastest variant of each algorithm measured in total time. The fastest solve time for each dataset has been underlined and the fastest total time has been marked with **bold face**. Datasets which contain many sub-problems are marked with (s).

Dataset	Nodes	Arcs	Solve	Total	Solve	Total	Solve	Total	Solve	Total	Solve	Total
3D segmentation: voxel-based			`MBK-R` [11]		`EIBFS-I` [36]		`HPF-H-L` [45]		HI-PR [19]		GridCut [52, 51]	
adhead.n26c100 [9, 12, 10]	12 M	327 M	65.81 s	92.57 s	22.60 s	33.93 s	24.29 s	29.03 s	225.38 s	424.67 s	25.19 s	**27.79 s**
adhead.n6c100 [9, 12, 10]	12 M	75 M	23.88 s	28.03 s	13.23 s	15.85 s	14.13 s	15.87 s	59.65 s	102.84 s	6.98 s	**7.31 s**
babyface.n26c100 [9, 12, 10]	5 M	131 M	82.29 s	92.87 s	30.13 s	**34.74 s**	54.47 s	56.71 s	183.60 s	228.09 s	53.21 s	54.11 s
babyface.n6c100 [9, 12, 10]	5 M	30 M	7.78 s	9.44 s	5.56 s	6.61 s	11.56 s	12.24 s	57.28 s	69.66 s	2.88 s	**3.00 s**
bone.n26c100 [9, 12, 10]	7 M	202 M	9.01 s	25.62 s	9.18 s	16.28 s	4.24 s	7.16 s	68.39 s	173.75 s	4.52 s	**5.88 s**
bone.n6c100 [9, 12, 10]	7 M	46 M	4.09 s	6.65 s	2.74 s	4.35 s	2.30 s	3.36 s	23.66 s	46.71 s	0.91 s	**1.12 s**
bone_subx.n6c100 [9, 12, 10]	3 M	23 M	4.10 s	5.36 s	2.38 s	3.11 s	1.28 s	1.81 s	10.34 s	21.49 s	1.34 s	**1.44 s**
bone_subx.n26c100 [9, 12, 10]	3 M	101 M	7.70 s	15.78 s	4.74 s	8.23 s	2.14 s	**3.61 s**	25.51 s	75.15 s	3.69 s	4.45 s
liver.n26c100 [9, 12, 10]	4 M	108 M	11.78 s	20.41 s	10.49 s	14.20 s	5.72 s	6.50 s	71.88 s	131.00 s	5.62 s	**6.21 s**
liver.n6c100 [9, 12, 10]	4 M	25 M	10.08 s	11.40 s	5.82 s	6.57 s	5.70 s	6.24 s	30.49 s	42.71 s	3.87 s	**3.99 s**
3D segmentation: oriented MRF			`MBK` [11]		`EIBFS-I-NR` [36]		`HPF-H-L` [45]		HI-PR [19]		GridCut [52, 51]	
vessel.orimrf.256 [11, 38, 87]	16 M	66 M	1.84 s	2.95 s	1.13 s	2.03 s	3.19 s	6.80 s	4.11 s	30.99 s	0.40 s	**1.04 s**
vessel.orimrf.512 [11, 38, 87]	134 M	536 M	12.44 s	21.40 s	7.95 s	15.39 s	25.29 s	55.32 s	32.16 s	321.75 s	2.43 s	**7.73 s**
vessel.orimrf.900 [11, 38, 87]	688 M	2 B	75.23 s	121.82 s	48.13 s	88.09 s	147.22 s	300.79 s	177.38 s	1774.65 s	15.97 s	**44.70 s**
3D U-Net segmentation cleaning			`MBK` [11]		`EIBFS-I-NR` [36]		`HPF-H-L` [45]		HI-PR [19]		GridCut [52, 51]	
clean.orimrf.256 [11, 38, 87]	16 M	66 M	0.97 s	2.09 s	0.69 s	1.61 s	3.21 s	6.89 s	3.93 s	30.83 s	0.13 s	**0.77 s**
clean.orimrf.512 [11, 38, 87]	134 M	536 M	7.87 s	17.03 s	5.51 s	13.51 s	27.10 s	58.22 s	31.40 s	320.87 s	0.91 s	**6.27 s**
clean.orimrf.900 [11, 38, 87]	688 M	2 B	35.83 s	81.92 s	25.96 s	64.22 s	130.22 s	280.73 s	163.88 s	1755.87 s	3.90 s	**31.43 s**
unet_mrfclean_2 [11]	8 M	32 M	0.47 s	1.01 s	0.29 s	0.74 s	3.55 s	5.36 s	9.80 s	22.82 s	62 ms	**0.36 s**
unet_mrfclean_3 [11]	15 M	63 M	0.82 s	1.88 s	0.52 s	1.37 s	5.68 s	9.14 s	20.59 s	46.69 s	0.11 s	**0.68 s**
unet_mrfclean_8 [11]	4 M	19 M	0.48 s	0.81 s	0.24 s	0.50 s	2.39 s	3.46 s	6.55 s	13.89 s	0.11 s	**0.28 s**
Surface fitting			`MBK` [11]		`EIBFS-I` [36]		`HPF-H-L` [45]		HI-PR [19]		GridCut [52, 51]	
LB07-bunny-lrg [71]	49 M	300 M	15.40 s	21.17 s	6.38 s	15.25 s	21.87 s	32.13 s	610.24 s	820.64 s	2.36 s	**3.75 s**
3D segmentation: sparse layered graphs (SLG)			`MBK-R` [11]		`EIBFS-I` [36]		`HPF-H-L` [45]		HI-PR [19]		GridCut [52, 51]	
4Dpipe_small [57]	14 M	124 M	6.03 h	6.03 h	2.06 s	**15.55 s**	17.91 s	28.85 s	202.49 s	266.01 s	-	-
4Dpipe_big [57]	143 M	1 B	-	-	20.59 s	**195.41 s**	222.09 s	332.06 s	2611.65 s	3436.43 s	-	-
NT32_tomo3_.raw_3 [57]	7 M	49 M	15.42 s	18.69 s	24.22 s	27.19 s	15.87 s	**18.33 s**	176.11 s	200.29 s	-	-
NT32_tomo3_.raw_10 [57]	22 M	154 M	52.86 s	63.15 s	50.82 s	60.01 s	36.46 s	**44.14 s**	645.33 s	741.98 s	-	-
NT32_tomo3_.raw_30 [57]	67 M	462 M	145.23 s	**176.37 s**	194.79 s	221.90 s	179.82 s	202.73 s	2939.04 s	3260.63 s	-	-
NT32_tomo3_.raw_100 [57]	183 M	1 B	778.39 s	860.71 s	553.50 s	627.08 s	520.26 s	**583.76 s**	9732.34 s	2.95 h	-	-
3D segmentation: seperating surfaces			`MBK-R` [11]		`EIBFS-I` [36]		`HPF-H-L` [45]		HI-PR [19]		GridCut [52, 51]	
cells.sd3 [53]	13 M	126 M	48.23 s	59.03 s	35.24 s	40.66 s	15.52 s	**21.84 s**	98.25 s	167.56 s	-	-
foam.subset.r160.h210 [53]	15 M	205 M	6.05 s	22.02 s	3.21 s	**12.52 s**	17.14 s	26.18 s	15.16 s	145.58 s	-	-
foam.subset.r60.h210 [53]	1 M	24 M	0.62 s	2.58 s	0.39 s	**1.49 s**	1.98 s	3.01 s	1.85 s	12.82 s	-	-
simcells.sd3 [53]	3 M	27 M	9.93 s	12.10 s	2.94 s	**4.12 s**	3.23 s	4.60 s	21.57 s	33.89 s	-	-
Deep LOGISMOS			`MBK` [11]		`EIBFS-I` [36]		`HPF-H-F` [45]		HI-PR [19]		GridCut [52, 51]	
deeplogismos.2 [42]	511 K	4 M	0.15 s	0.25 s	28 ms	**0.21 s**	0.12 s	0.31 s	0.16 s	1.29 s	-	-
deeplogismos.3 [42]	707 K	5 M	0.18 s	0.31 s	41 ms	**0.30 s**	0.18 s	0.45 s	0.24 s	1.90 s	-	-
deeplogismos.7 [42]	989 K	7 M	0.34 s	**0.54 s**	0.26 s	0.66 s	0.29 s	0.69 s	0.36 s	2.86 s	-	-
Super resolution			`BK` [11]		`EIBFS-I` [36]		`HPF-H-L` [45]		HI-PR [19]		GridCut [52, 51]	
super_res-E1 [30, 88]	10 K	62 K	2 ms	2 ms	1 ms	**2 ms**	2 ms	3 ms	1 ms	7 ms	-	-
super_res-E2 [30, 88]	10 K	103 K	4 ms	5 ms	2 ms	3 ms	2 ms	**3 ms**	2 ms	12 ms	-	-
super_res-Paper1 [30, 88]	10 K	62 K	2 ms	3 ms	1 ms	**2 ms**	2 ms	3 ms	1 ms	7 ms	-	-
superres_graph [30, 88]	43 K	742 K	62 ms	78 ms	10 ms	26 ms	7 ms	**12 ms**	19 ms	0.16 s	-	-
Texture			`MBK-R` [11]		`EIBFS-I` [36]		`HPF-H-L` [45]		HI-PR [19]		GridCut [52, 51]	
texture-Cremer [88]	44 K	783 K	1.54 s	1.58 s	0.35 s	0.37 s	0.17 s	0.19 s	42 ms	**0.19 s**	-	-
texture-OLD-D103 [88]	43 K	742 K	0.60 s	0.65 s	0.19 s	0.21 s	73 ms	**92 ms**	41 ms	0.19 s	-	-
texture-Paper1 [88]	43 K	742 K	0.65 s	0.69 s	0.19 s	0.21 s	76 ms	**95 ms**	36 ms	0.17 s	-	-
texture-Temp [88]	14 K	239 K	0.22 s	0.23 s	30 ms	34 ms	9 ms	**15 ms**	6 ms	32 ms	-	-
Automatic labelling envrionment (ALE)			`MBK-R` [11]		`EIBFS-I-NR` [36]		`HPF-L-L` [45]		HI-PR [19]		GridCut [52, 51]	
graph_1 (s) [68, 69, 26, 67]	185 K	5 M	16.80 s	18.52 s	0.35 s	**0.79 s**	1.00 s	1.60 s	1.58 s	10.60 s	-	-
graph_2 (s) [68, 69, 26, 67]	175 K	3 M	7.38 s	10.47 s	0.83 s	**1.64 s**	2.25 s	3.55 s	2.91 s	20.87 s	-	-
graph_3 (s) [68, 69, 26, 67]	179 K	7 M	27.68 s	35.55 s	2.69 s	**4.51 s**	4.63 s	6.96 s	6.49 s	43.73 s	-	-
Multi-view			`MBK-R` [11]		`EIBFS-I` [36]		`HPF-H-L` [45]		HI-PR [19]		GridCut [52, 51]	
BL06-camel-lrg [13]	18 M	93 M	107.53 s	111.42 s	28.55 s	31.54 s	24.44 s	**28.82 s**	291.71 s	337.91 s	-	-
BL06-gargoyle-lrg [13]	17 M	86 M	238.08 s	241.65 s	33.76 s	36.57 s	26.51 s	**30.61 s**	208.27 s	251.10 s	-	-

TABLE 2: Continued

Dataset	Nodes	Arcs	Solve	Total	Solve	Total	Solve	Total	Solve	Total	Solve	Total
Deconvolution			MBK-R [11]		EIBFS-I [36]		HPF-H-L [45]		HI-PR [19]		GridCut [52, 51]	
graph3x3 [88]	2 K	47 K	9 ms	11 ms	3 ms	3 ms	1 ms	**1 ms**	1 ms	5 ms	-	-
graph5x5 [88]	2 K	139 K	62 ms	67 ms	6 ms	9 ms	3 ms	**4 ms**	2 ms	15 ms	-	-
Stereo 1			BK [11]		EIBFS-I-NR [36]		HPF-H-L [45]		HI-PR [19]		GridCut [52, 51]	
BVZ-sawtooth (s) [14]	164 K	796 K	0.91 s	1.16 s	0.58 s	**0.69 s**	1.39 s	1.85 s	7.89 s	12.27 s	-	-
BVZ-tsukuba (s) [14]	110 K	513 K	0.49 s	0.58 s	0.35 s	**0.41 s**	0.66 s	0.84 s	4.69 s	6.64 s	-	-
BVZ-venus (s) [14]	166 K	795 K	1.72 s	2.03 s	1.30 s	**1.44 s**	1.94 s	2.46 s	15.00 s	20.11 s	-	-
Stereo 2			BK [11]		EIBFS-I [36]		HPF-H-L [45]		HI-PR [19]		GridCut [52, 51]	
KZ2-sawtooth (s) [64]	294 K	1 M	2.59 s	3.40 s	1.14 s	**2.02 s**	3.30 s	4.66 s	23.79 s	36.55 s	-	-
KZ2-tsukuba (s) [64]	199 K	1 M	1.41 s	1.84 s	0.71 s	**1.12 s**	1.92 s	2.55 s	20.95 s	27.14 s	-	-
KZ2-venus (s) [64]	301 K	2 M	3.98 s	4.89 s	2.18 s	**3.16 s**	4.70 s	6.21 s	41.63 s	55.60 s	-	-
Decision tree field (DTF)			MBK-R [11]		EIBFS-I [36]		HPF-H-L [45]		HI-PR [19]		GridCut [52, 51]	
printed_graph1 [82]	20 K	1 M	0.63 s	0.73 s	0.13 s	0.17 s	40 ms	**51 ms**	51 ms	0.25 s	-	-
printed_graph16 [82]	11 K	683 K	0.24 s	0.29 s	44 ms	62 ms	16 ms	**22 ms**	25 ms	0.12 s	-	-
Graph matching: small			BK [11]		EIBFS-I-NR [36]		HPF-L-L [45]		HI-PR [19]		GridCut [52, 51]	
atlas1.dd (s) [58, 48]	1 K	5 K	37 ms	59 ms	34 ms	52 ms	21 ms	**39 ms**	23 ms	0.18 s	-	-
car1.dd (s) [25, 73, 48]	38	131	1 ms	1 ms	0 ms	**1 ms**	1 ms	1 ms	0 ms	3 ms	-	-
hassan1.dd (s) [1, 92, 48]	120	2 K	17 ms	30 ms	5 ms	25 ms	2 ms	**6 ms**	4 ms	65 ms	-	-
matching1.dd (s) [66, 59, 48]	38	380	10 ms	12 ms	5 ms	8 ms	2 ms	**4 ms**	6 ms	14 ms	-	-
Graph matching: big			MBK-R [11]		EIBFS-I [36]		HPF-H-L [45]		HI-PR [19]		GridCut [52, 51]	
pair1.dd (s) [48]	1 K	58 K	1.42 s	1.97 s	0.70 s	0.96 s	92 ms	**0.13 s**	0.82 s	1.68 s	-	-
Mesh segmentation			MBK-R [11]		EIBFS-I [36]		HPF-H-F [45]		HI-PR [19]		GridCut [52, 51]	
bunny.segment [76]	97 K	536 K	0.12 s	0.14 s	63 ms	**75 ms**	68 ms	91 ms	0.20 s	0.30 s	-	-
bunnybig.segment [76]	2 M	13 M	1.01 s	1.59 s	0.62 s	**1.23 s**	1.43 s	2.12 s	4.99 s	9.91 s	-	-
candle.segment [76]	159 K	959 K	87 ms	0.13 s	49 ms	**83 ms**	0.11 s	0.15 s	0.29 s	0.53 s	-	-
candlebig.segment [76]	1 M	5 M	0.51 s	0.72 s	0.26 s	**0.44 s**	0.60 s	0.91 s	2.03 s	3.70 s	-	-
chair.segment [76]	305 K	1 M	0.76 s	0.88 s	0.31 s	0.39 s	0.27 s	**0.37 s**	0.86 s	1.37 s	-	-
chairbig.segment [76]	3 M	26 M	1.62 s	2.89 s	1.02 s	**2.45 s**	3.23 s	4.59 s	9.98 s	20.92 s	-	-
handbig.segment [76]	248 K	1 M	0.15 s	0.19 s	71 ms	**0.11 s**	0.13 s	0.18 s	0.35 s	0.63 s	-	-
handsmall.segment [76]	15 K	69 K	4 ms	5 ms	2 ms	**3 ms**	4 ms	6 ms	10 ms	16 ms	-	-

for the matching. For all but four problem families, the best algorithm achieves a mean relative performance of 95% or higher. Furthermore, for most problem families, one algorithm is always the best. This indicates that the problem family is a strong predictor of algorithm performance. The problem family where this strategy performs the worst is 3D segmentation with sparse layered graphs (SLG). Here, the mean RP is only 81%, which is likely due to the large variation in graph size in this problem family.

Table 6 shows the best performing parallel algorithm for each problem family. For the 6-connected graphs, the parallel GridCut algorithm is clearly superior, but otherwise, the different families appear to favor different algorithms.

Scenario 3: Known Graph Finally, we consider a strategy where the graph is known, but the problem family is not. Here, our strategy is to train a simple decision tree to predict the best algorithm given a feature vector that describes the graph to be solved. Although a single decision tree is not the strongest classifier, it has the benefit of being easily interpretable.

The first components of our feature vector consist of the number of nodes, the number of terminal arcs, the number of neighbor arcs, and whether the graph is a grid graph. Then we include mean, standard deviation, and standard deviation of non-zero values for a number of arc and node properties. For arc properties, we use: source, sink, terminal (source and sink combined), and neighbor capacities. Finally, for node properties we use: sum of in-going neighbor capacities, sum of out-going neighbor capacities, sum of neighbor capacities, degrees, out degrees, and in degrees counts only non-zero arcs. Note that these statistics can be computed efficiently during graph construction. We normalize all capacity statistics by the mean over all arc capacities. In total, our feature vector has 31 entries per graph. Fig. 11 shows a UMAP embedding [78] of the feature vectors for all benchmark datasets. Similar problem families cluster together, despite UMAP receiving no information on this. This suggests the feature vectors provide a good description of the graphs.

We train the decision tree using Scikit-learn [83] version 0.23.1. We use Gini impurity as the split criterion and reduce the tree using minimal cost-complexity pruning [15]. The optimal amount of pruning is determined with 5-fold cross validation. We split each problem family evenly into the folds (if it contains at least 5 datasets). When fitting, each dataset is weighed by one over the number of datasets in its problem family. When evaluating, we oversample the validation data, so that each problem family has the same number of entries. This indicates how well the decision tree will perform with representative training data. We also perform an additional evaluation where we hold out one problem family, fit on the rest, and then evaluate on the held out family. This indicates how well the decision tree will perform for a problem family that it has not yet encountered. We use the mean RP as validation metric.

We first train a decision tree for the serial algorithms; the result is shown in Fig. 12. It achieves a mean RP of 0.82 and 0.82 in the two evaluations, respectively. This means that the tree is

TABLE 3: **Performance of parallel algorithms** based on build and solve times. We show a representative subset of the datasets grouped according to their problem family. See Table 2 for the number of nodes and arcs. The algorithms were run with 1, 2, 4, 6, 8, 12, 16, 24, and 32 threads. Only the best time is shown along with the thread count for that run. For comparison, the solve time for the fastest serial algorithm is also included. All times are in seconds. The fastest solve time for each dataset has been marked with **bold face**

	Liu-Sun [75]			P-PPR [5]			Strandmark-Kahl [91]			P-ARD [89]			P-GridCut [52, 51]			Best serial	
Dataset	Build	Best solve		Build	Best solve		Build	Best solve		Build	Best solve		Build	Best solve		Algo.	Solve
3D segmentation: voxel-based																	
adhead.n26c10 [9, 12, 10]	6.90	17.41	8T	42.71	14.37	12T	26.14	25.40	2T	78.20	35.17	4T	-	-	-	GridCut	**13.21**
adhead.n26c100 [9, 12, 10]	6.86	20.87	8T	41.79	**8.41**	32T	27.65	19.30	6T	75.37	42.91	4T	-	-	-	EIBFS	22.60
babyface.n26c100 [9, 12, 10]	2.89	72.26	32T	15.99	**7.93**	32T	9.15	40.86	4T	32.01	61.20	32T	-	-	-	EIBFS	30.13
bone.n26c100 [9, 12, 10]	4.36	3.68	32T	25.63	4.01	32T	16.38	5.04	8T	48.90	11.21	4T	-	-	-	HPF	**3.48**
bone_subx.n26c100 [9, 12, 10]	2.30	4.04	16T	12.48	2.34	32T	8.03	4.43	8T	24.07	11.71	16T	-	-	-	HPF	**2.14**
liver.n26c10 [9, 12, 10]	2.37	10.79	6T	14.42	7.91	32T	0.89	3.49	1T	21.93	14.95	1T	-	-	-	GridCut	**2.95**
liver.n26c100 [9, 12, 10]	2.36	18.24	6T	13.94	**5.45**	32T	0.86	6.69	1T	27.07	25.74	24T	-	-	-	GridCut	5.62
liver.n6c100 [9, 12, 10]	0.53	7.62	6T	4.78	3.68	16T	0.51	7.24	1T	6.18	7.74	32T	0.11	**2.70**	6T	GridCut	3.87
adhead.n6c100 [9, 12, 10]	1.59	11.17	8T	14.49	4.72	32T	2.52	7.17	4T	15.46	14.41	2T	0.31	**3.83**	4T	GridCut	6.98
babyface.n6c10 [9, 12, 10]	0.66	2.70	32T	5.30	3.09	24T	0.73	3.55	1T	8.87	5.35	1T	0.12	**0.88**	32T	GridCut	1.52
babyface.n6c100 [9, 12, 10]	0.66	5.34	32T	6.53	3.63	24T	0.98	5.24	4T	7.29	7.28	32T	0.12	**1.66**	16T	GridCut	2.88
bone.n6c100 [9, 12, 10]	0.99	0.79	24T	8.30	2.66	32T	1.23	2.01	2T	11.18	2.08	4T	0.20	**0.17**	12T	GridCut	0.91
3D segmentation: oriented MRF																	
vessel.orimrf.256 [11]	1.22	0.69	6T	14.67	1.69	32T	1.89	1.04	2T	16.24	0.82	32T	0.52	**0.14**	12T	GridCut	0.40
vessel.orimrf.512 [11]	9.72	4.92	8T	-	-	-	15.08	6.49	2T	100.95	5.54	16T	4.26	**0.43**	32T	GridCut	2.43
vessel.orimrf.900 [11]	49.62	28.66	8T	-	-	-	79.82	38.45	2T	599.09	24.02	32T	21.98	**2.49**	32T	GridCut	15.97
3D U-Net segmentation cleaning																	
clean.orimrf.256 [11]	1.23	0.39	12T	16.50	2.48	24T	2.39	0.48	4T	15.96	0.84	8T	0.52	**0.06**	32T	GridCut	0.13
clean.orimrf.512 [11]	9.73	2.45	16T	-	-	-	18.26	3.81	4T	131.62	4.81	8T	4.37	**0.27**	32T	GridCut	0.91
clean.orimrf.900 [11]	50.52	12.36	16T	-	-	-	85.77	18.66	4T	578.09	20.17	16T	23.64	**0.82**	32T	GridCut	3.90
unet_mrfclean_3 [11]	1.15	0.27	32T	-	-	-	2.15	0.45	4T	12.66	0.53	4T	0.46	**0.04**	32T	GridCut	0.11
unet_mrfclean_8 [11]	0.37	0.21	8T	-	-	-	0.64	0.23	4T	4.01	0.28	2T	0.14	**0.04**	16T	GridCut	0.11
Surface fitting																	
LB07-bunny-lrg [71]	6.14	1.86	16T	55.88	24.27	32T	7.73	4.14	4T	72.02	4.31	16T	1.24	**0.32**	24T	GridCut	2.36
3D segmentation: sparse layered graphs (SLG)																	
4Dpipe_small [57]	9.93	9.39	12T	-	-	-	-	-	-	47.53	421.60	24T	-	-	-	EIBFS	**2.06**
4Dpipe_big [57]	122.43	86.18	16T	-	-	-	-	-	-	570.44	7485.11	4T	-	-	-	EIBFS	**20.59**
NT32_tomo3_.raw_10 [57]	4.99	18.58	12T	34.90	**14.39**	32T	22.02	85.49	1T	43.95	15.40	12T	-	-	-	HPF	36.46
NT32_tomo3_.raw_30 [57]	14.93	45.70	16T	111.70	59.81	32T	66.56	363.06	1T	132.41	**36.13**	32T	-	-	-	BK	145.23
NT32_tomo3_.raw_100 [57]	38.93	**95.24**	32T	-	-	1T	170.38	1189.78	1T	365.60	158.92	24T	-	-	-	HPF	498.94
3D segmentation: seperating surfaces																	
cells.sd3 [53]	4.18	10.33	16T	25.93	**9.47**	32T	23.90	76.40	1T	21.84	44.98	1T	-	-	-	HPF	15.52
foam.subset.r160.h210 [53]	5.91	8.59	32T	37.08	3.61	32T	52.11	17.24	1T	33.72	7.32	1T	-	-	-	EIBFS	**3.21**
simcells.sd3 [53]	0.69	2.28	16T	5.60	1.99	32T	1.49	2.82	32T	4.96	**0.89**	32T	-	-	-	EIBFS	2.94
Multi-view																	
BL06-camel-lrg [13]	3.99	57.41	8T	-	-	-	1.87	75.56	1T	13.76	95.40	1T	-	-	-	HPF	**24.44**
BL06-gargoyle-lrg [13]	3.70	29.28	16T	-	-	-	1.70	190.07	1T	12.20	102.31	2T	-	-	-	HPF	**26.51**
Mesh segmentation																	
bunnybig.segment [76]	0.30	**0.37**	12T	2.81	1.15	32T	1.12	1.89	1T	3.66	0.41	32T	-	-	-	EIBFS	0.62
chairbig.segment [76]	0.60	0.64	24T	6.10	1.76	32T	2.62	3.29	1T	6.89	**0.57**	32T	-	-	-	EIBFS	1.02
handbig.segment [76]	0.02	0.11	8T	0.25	0.25	16T	0.04	0.16	1T	0.40	0.11	32T	-	-	-	EIBFS	**0.07**

significantly better than naively choosing the overall best algorithm but not as good as knowing the best algorithm for a problem family.

Next, we train a decision tree for the parallel algorithms. We include a category 'Serial', which means that choosing a serial algorithm would be faster. For simplicity, we do not specify which serial algorithm to choose in this scenario. The result is shown in Fig. 13. The decision tree achieves a mean RP of 0.56 and 0.57 in the two evaluations, respectively. Thus, the tree is slightly better than simply choosing the overall best algorithm. However, the best option is to choose the best algorithm for a given category.

7 Discussion

In this section, we discuss the most interesting findings from our experiments.

7.1 Serial Algorithms

Our results clearly show that GridCut is superior to the other tested algorithms for min-cut/max-flow problems with fixed neighborhood grids. This is not surprising since GridCut has been designed and optimized specifically for this type of graph. However, as shown in Table 2, the performance benefit of GridCut decreases significantly

TABLE 4: **Summary of relative performance (RP) scores for each of the min-cut/max-flow algorithm variants.** The best score (higher is better) in each column has been marked with **bold face**. Results were oversampled as described in Fig. 7. We only include results where the algorithm ran to completion.

Serial algorithms	Mean RP ± Std. RP	Min RP	Max RP
EIBFS-I	0.59 ± 0.28	**0.1309**	**1.00**
EIBFS-I-NR	0.56 ± 0.32	0.0535	**1.00**
EIBFS	0.47 ± 0.23	0.1288	0.94
HI-PR	0.16 ± 0.17	0.0046	**1.00**
HPF-H-F	0.59 ± 0.33	0.0279	**1.00**
HPF-H-L	**0.64 ± 0.36**	0.0393	**1.00**
HPF-L-F	0.49 ± 0.29	0.0313	**1.00**
HPF-L-L	0.53 ± 0.31	0.0312	**1.00**
MBK-R	0.27 ± 0.20	0.0006	**1.00**
BK	0.27 ± 0.24	0.0005	**1.00**
MBK	0.28 ± 0.22	0.0005	**1.00**
GridCut*	**0.99 ± 0.03**	**0.6419**	**1.00**
Parallel algorithms			
Liu-Sun	**0.48 ± 0.30**	0.0667	**1.00**
P-PPR	0.46 ± 0.38	0.0133	**1.00**
Strandmark-Kahl	0.23 ± 0.16	0.0667	0.85
P-ARD	0.35 ±0.32	0.0028	**1.00**
P-GridCut*	**1.00 ± 0.00**	**1.0000**	**1.00**
Best serial	**0.59 ± 0.33**	**0.1365**	**1.00**

* Only grid graphs included (6- and 26-conn. for serial, 6-conn for parallel).

TABLE 5: **Relative performance (RP) scores for the best serial algorithm variant for each problem family.** Almost all problem families have one dominant algorithm.

Problem family	Algorithm	Mean RP
3D segmentation: SLG [57]	HPF-H-L	0.81
Multi-view [13]	HPF-H-L	1.00
Surface fitting [71]	GridCut	1.00
3D segmentation: voxel-based [9, 12, 10]	GridCut	0.98
Mesh segmentation [76]	EIBFS-I	0.95
3D segmentation: sep. surfaces [53]	EIBFS-I	0.92
3D MRF [11]	GridCut	1.00
Deep LOGISMOS [42]	EIBFS-I-NR	0.96
Deconvolution [88]	HPF-H-L	0.96
DTF [82]	HPF-H-L	1.00
Super resolution [30, 88]	EIBFS-I	0.87
Stereo 1 [14]	EIBFS-I	0.99
Stereo 2 [64]	EIBFS-I	1.00
ALE [68, 69, 26]	EIBFS-I-NR	1.00
Graph matching: small [58, 48]	HPF-L-L	1.00
Graph matching: small [25, 73, 48]	EIBFS-I-NR	0.91
Graph matching: small [1, 92, 48]	HPF-L-F	1.00
Graph matching: small [94, 16, 48]	HPF-L-L	1.00
Graph matching: small [66, 59, 48]	HPF-L-F	1.00
Graph matching: big [48]	HPF-H-L	1.00
Mean ± std.		0.97 ± 0.05

TABLE 6: **Relative performance (RP) scores for the best parallel algorithm for each problem family.** Since the parallel GridCut implementation can only handle 6-connected graphs '3D segmentation: voxel based' has been split into two subgroups: 6-connected graphs and 26-connected graphs. If an algorithm did not run to completion on a dataset we count the RP as 0.

Problem family	Algorithm	Mean RP
3D segmentation: SLG [57]	Liu-Sun	0.63
Multi-view [13]	Serial	1.00
Surface fitting [71]	P-GridCut	1.00
3D seg.: voxel-based [9, 12, 10] (26-conn.)	Serial	0.86
3D seg.: voxel-based [9, 12, 10] (6-conn.)	P-GridCut	1.00
Mesh segmentation [76]	P-ARD	0.88
3D segmentation: sep. surfaces [53]	P-PPR	0.74
3D MRF [11]	P-GridCut	1.00
Mean ± std.		0.89 ± 0.14

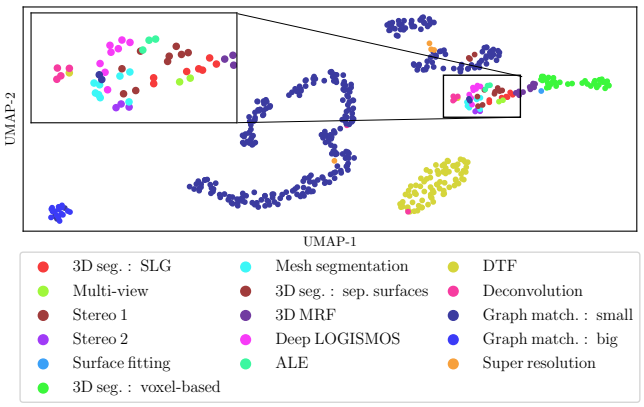


Fig. 11: **UMAP embedding [78] of the extracted graph features.** Each point correspond to a benchmark dataset and is colored according to its problem family. When a benchmark consists of multiple sub-problem we use the mean feature vector. Notice that points from the same problem family tend to cluster together.

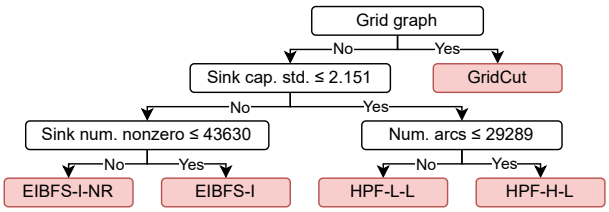


Fig. 12: Decision tree trained to select the best serial algorithm. Note that capacity statistics are normalized, *c.f.* Section 6.

when moving from 6-connected to 26-connected graphs. Actually, both EIBFS and HPF manage to beat GridCut on a couple of the 26-connected problems. This indicates that the benefit of using GridCut significantly decreases for graphs with high connectivity, perhaps because it is an AP algorithm.

In general, the pseudoflow algorithms have the best overall performance. Measured on solve time, EIBFS performs the best, which aligns with existing literature [36]. However, looking at the total time, HPF performs better overall, slightly contradicting previous benchmarks [36]. The reason for the difference between the results may be that [36] compared EIBFS with HPF-H-F, which we show to be inferior to HPF-H-L. Also, in [36] they use 32-bit pointer for most datasets, which may also provide slightly improved performance. Finally, the hardware used in [36] may have different performance characteristics than ours. We observed that EIBFS actually performed better on an older system than on the one we used for our experiments. We speculate if this could be due to the lower cache and memory latency (estimated using Intel® Memory Latency Checker v3.9a) on the older system compared to the one used for our benchmarks. In any case, this raises the question whether HPF implemented with arc packing could outperform EIBFS on even more problems.

Our results also show that the performance of the different

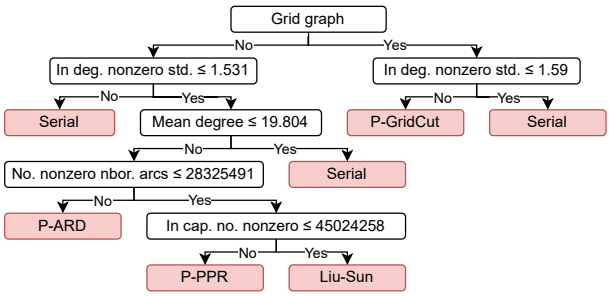


Fig. 13: Decision tree trained to select the best parallel algorithm. 'Serial' means a serial algorithm would be the best option. Note that capacity statistics are normalized, *c.f*. Section 6.

algorithm variants varies, and the choice of variant can significantly affect the run time. Optimizing for cache efficiency seems to be of particular importance, since optimizations such as arc packing and smaller data structures have large effects on the solve times for both BK and EIBFS.

As shown in Section 6, for non-grid problems, the best algorithm most often comes down to a choice between EIBFS or HPF. From Fig. 12 it seems that HPF is faster when the sink (or, more likely, terminal) arc capacities vary a lot. As expected, `EIBFS-I-NR` is preferred for small graphs, while the preferred HPF variant for small graphs appears to be `HPF-L-L`, which aligns with the results in Table 5. However, the best strategy is to test several algorithms on a set of problems from the family at hand.

7.2 Parallel Algorithms

P-GridCut provides the best performance of the parallel algorithms for 6-connected grid graph problems and scales well with many threads. Of the other parallel algorithms, Liu-Sun is overall the best, closely followed by P-PPR, which aligns with previous results [75] and expectations [89, 91]. However, all the block-based algorithms only scale well for large graphs. For small to medium problems, they do not scale to many threads, but seem to peak at 8-12 threads, *c.f*. Fig. 10. This also means that choosing an optimal thread count may be difficult. Only P-PPR scaled consistently with up to 32 threads. In addition, all parallel algorithms were often outperformed by a serial algorithm except on large graphs. In fact, as Table 4 shows, selecting a good serial algorithm has better expected performance than selecting any of the parallel algorithms.

For practical use, only the Liu-Sun, P-PPR, and P-ARD algorithms seem to be relevant as is. However, the block-based algorithms have the additional challenge of dividing the graph into blocks — the result of which significantly affects the run time of the algorithms. This was also shown in [89], where it was noticed that the multiview problems would scale better with more processors when partitioned on vertex numbers vs. the grid. While the graphs tested in this work have a natural way to be split, this may not always be the case. Meanwhile, even though this problem is avoided with P-PPR, it does not perform as well as the block-based algorithms overall, as shown in Fig. 9.

Finally, while all parallel algorithms had datasets where they were best, selecting the best parallel algorithm is difficult (except for 6-connected grid graphs). No algorithm showed dominant performance — neither globally nor per problem family. Furthermore, using the decision tree only gives a small improvement over selecting the best overall algorithm. Fig. 13 indicates that for grid and low-degree graphs, a serial algorithm or GridCut performs best. Otherwise, the choice comes down to graph size, with P-ARD doing better for the smaller graphs, P-PPR being faster for the medium-sized ones, and Liu-Sun performing the best for the largest graphs. However, as Table 6 shows, the best strategy is again to test on a number of graphs from the problem family at hand.

8 Conclusions and Perspectives

We now summarize our findings for the serial and parallel algorithms tested in this work. We also provide perspectives on possible future developments of min-cut/max-flow methods, as well as how these may fit into the future of computer vision.

8.1 Serial Algorithms

For the serial min-cut/max-flow algorithms, we have tested a total of 12 different variants across five of the fastest and most popular algorithms: PPR, BK, EIBFS, HPF, and GridCut. These include representatives for the three families of min-cut/max-flow algorithms: augmenting paths, push-relabel, and pseudoflow.

Our results clearly show that, for simple grid graphs, GridCut has the best performance. In most other cases, the two pseudoflow algorithms, EIBFS and HPF, are significantly faster than the other algorithms and thus should be the first choice for anyone looking for a fast serial min-cut/max-flow algorithm for static computer vision problems. For dynamic problems, we refer to [36].

Contrary to existing literature, we recommend the HPF algorithm in the H-LIFO configuration as the default, since it has the best overall performance. However, the EIBFS algorithm (`EIBFS-I` implementation) is a very close contender and can easily replace HPF with little impact on performance — and indeed may perform better on some problem families. If memory usage is of chief concern, the `MBK` and `EIBFS-I-NR` implementations are both good options, as they use significantly less memory than the reference EIBFS and HPF implementations.

Furthermore, we think significant performance improvements may be gained from further improving the algorithm implementations — especially with a focus on memory use and cache efficiency. In particular, faster and more memory efficient methods for arc (and node) packing could result in significant benefits, since the extra initialization step incurs a large memory and run time overhead. We would like to see a reimplementation of HPF with a half-arc data structure and arc packing.

Finally, we found significant gains through automatic algorithm selection. Based on our results, it seems likely that one could train a robust classifier for selecting the appropriate algorithm based on the min-cut/max-flow problem to be solved. By selecting the right algorithm for the job, run time could in many cases be significantly reduced without the need for new algorithms or implementations. In general, we find it unlikely that a single algorithm will ever be dominant for all types of graphs.

8.2 Parallel Algorithms

We tested five different parallel algorithms for min-cut/max-flow problems: parallel PPR (P-PPR), adaptive bottom-up merging (Liu-Sun), dual decomposition (Strandmark-Kahl), region discharge (P-ARD), and parallel GridCut (P-GridCut).

If the graph is a simple grid, P-GridCut significantly outperforms all other algorithms. For other graphs, we found adaptive

bottom-up merging, as proposed by Liu and Sun [75], to be the best overall parallel approach. However, each parallel algorithm had an area in which it was the best, and it is difficult to predict the best parallel algorithm for a graph (except for 6-connected grid graphs).

Of the parallel algorithms, only P-GridCut and P-PPR improved consistently with more threads. All block-based algorithms failed to scale beyond 12 threads, except on large graphs. Furthermore, except for P-GridCut, all parallel algorithms were often outperformed by a serial algorithm, and consistent improvements over serial algorithms were obtained only for large graphs. These issues reveal a major deficiency in the state of current parallel min-cut/max-flow algorithms and deserve further study. While providing good scaling on any type of graph may be unreachable as min-cut/max-flow is P-complete and therefore hard to parallelize [39], computer vision graphs often come with additional structure. Therefore, it seems highly likely that further improvements in practical performance can be achieved. However, at this time, we only recommend using a parallel algorithm for graphs with more than 5 M nodes or where a serial algorithm uses at least 5 seconds.

To improve the parallel min-cut/max-flow algorithms, one could try to replace BK, which is currently used in all the tested block-based parallel algorithms, with a pseudoflow algorithm. However, this may not be trivial. In [56], results for a Liu-Sun implementation using EIBFS instead of BK showed a significant performance decrease compared to serial EIBFS. Still, given the superior performance of pseudoflow algorithms, this is an important area to investigate. Furthermore, parallelized graph construction is currently only available for P-GridCut. As the build time is a significant part of the total time, reducing build time will be important — especially as solve time decreases.

Finally, choosing an optimal blocking strategy remains an open problem. Generally, when nodes correspond to spatial positions (*e.g.*, pixels or mesh vertices), we find that grouping based on spatial distance works well. However, we recommend that practitioners experiment with different blocking strategies since it can significantly affect the performance. Furthermore, a general method that only considers the graph structure would be of high interest, as this would also make the algorithms more accessible to the average user. An alternative would be to focus on P-PPR algorithms that do not rely on blocking. Further improvements in these areas could also open the door to GPU-based implementations for solving general min-cut/max-flow problems.

8.3 Min-Cut/Max-Flow in Modern Computer Vision

It is no secret that the field of computer vision is currently dominated by deep learning. In this context, it is highly relevant to consider the future role of traditional computer vision tools, such as min-cut/max-flow algorithms.

For 3D images used in medical imaging and materials science research [77], it is common to have images where no relevant training data are available. Here, segmentation methods based on min-cut/max-flow continue to play an important role, as they work without training data and allow geometric prior knowledge to be incorporated. Furthermore, while modern 3D images can already be very large (many GB per image), dynamic imaging (3D + time) with high acquisition rates is now also possible [31, 81]. Computational efficiency is paramount to be able to process this ever increasing amount of data, and for this, parallel min-cut/max-flow algorithms could prove particularly useful.

Finally, as mentioned in [80], there is agreement that the performance of deep learning-based segmentation methods has started to plateau, and investigating how to integrate CNNs with 'classical' approaches should be pursued. Already, combinations with active contours have shown promising results [41, 85, 99] and a combination of CNNs and min-cut/max-flow methods could lead to new advances. As deep learning involves repeated forward and backward passes through a model, it is crucial that the min-cut/max-flow algorithms are fast and efficient. While not the focus of this work, this is also an area where dynamic min-cut/max-flow algorithms can be of great importance, as they are effective at handling repeated solves of graphs where capacities do not change drastically between successive solves.

REFERENCES

[1] Hassan Abu Alhaija et al. "Graphflow–6D large displacement scene flow via graph matching". In: *German Conference on Pattern Recognition*. Springer. 2015, pp. 285–296.

[2] Richard Anderson and Joao C. Setubal. "A parallel implementation of the push-relabel algorithm for the maximum flow problem". In: *Journal of Parallel and Distributed Computing* 29.1 (1995), pp. 17–26.

[3] Chetan Arora et al. "An efficient graph cut algorithm for computer vision problems". In: *European Conference on Computer Vision*. 2010, pp. 552–565.

[4] David A Bader and Vipin Sachdeva. *A cache-aware parallel implementation of the push-relabel network flow algorithm and experimental evaluation of the gap relabeling heuristic*. Tech. rep. Georgia Institute of Technology, 2006.

[5] Niklas Baumstark, Guy Blelloch, and Julian Shun. "Efficient implementation of a synchronous parallel push-relabel algorithm". In: *European Symposium on Algorithms*. 2015, pp. 106–117.

[6] Endre Boros, Peter L Hammer, and Xiaorong Sun. *Network flows and minimization of quadratic pseudo-Boolean functions*. Tech. rep. 17-1991, RUTCOR, 1991.

[7] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge University Press, 2004.

[8] Y. Boykov, O. Veksler, and R. Zabih. "Fast approximate energy minimization via graph cuts". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 23.11 (2001), pp. 1222–1239.

[9] Yuri Y Boykov and M-P Jolly. "Interactive graph cuts for optimal boundary & region segmentation of objects in ND images". In: *International Conference on Computer Vision*. Vol. 1. 2001, pp. 105–112.

[10] Yuri Boykov and Gareth Funka-Lea. "Graph cuts and efficient ND image segmentation". In: *International Journal of Computer Vision* 70 (2006), pp. 109–131.

[11] Yuri Boykov and Vladimir Kolmogorov. "An Experimental Comparison of Min-Cut/Max-Flow Algorithms for Energy Minimization in Vision". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 26.9 (2004), pp. 1124–1137.

[12] Yuri Boykov and Vladimir Kolmogorov. "Computing geodesics and minimal surfaces via graph cuts." In: *International Conference on Computer Vision*. Vol. 3. 2003, pp. 26–33.

[13] Yuri Boykov and Victor S Lempitsky. "From Photohulls to Photoflux Optimization." In: *British Machine Vision Conference*. Vol. 3. 2006, p. 27.

[14] Yuri Boykov, Olga Veksler, and Ramin Zabih. "Markov random fields with efficient approximations". In: *IEEE Conference on Computer Vision and Pattern Recognition*. 1998, pp. 648–655.

[15] Leo Breiman et al. *Classification and regression trees*. Routledge, 2017.

[16] Tibério S Caetano et al. "Learning graph matching". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 31 (2009), pp. 1048–1058.

[17] Bala G Chandran and Dorit S Hochbaum. "A computational study of the pseudoflow and push-relabel algorithms for the maximum flow problem". In: *Operations Research* 57.2 (2009), pp. 358–376.

[18] Xinjian Chen and Lingjiao Pan. "A survey of graph cuts/graph search based medical image segmentation". In: *IEEE Reviews in Biomedical Engineering (RBME)* 11 (2018), pp. 112–124.

[19] Boris V Cherkassky and Andrew V Goldberg. "On implementing the push—relabel method for the maximum flow problem". In: *Algorithmica* 19.4 (1997), pp. 390–410.

[20] Özgün Çiçek et al. "3D U-Net: learning dense volumetric segmentation from sparse annotation". In: *International Conference on Medical Image Computing and Computer Assisted Intervention*. 2016, pp. 424–432.

[21] Thomas H Cormen et al. *Introduction to algorithms*. MIT press, 2009.

[22] Andrew Delong and Yuri Boykov. "A scalable graph-cut algorithm for ND grids". In: *IEEE Conference on Computer Vision and Pattern Recognition*. 2008, pp. 1–8.

[23] DTU Computing Center. *DTU Computing Center resources*. 2021. DOI: 10.48714/DTU.HPC.0001. URL: https://doi.org/10.48714/DTU.HPC.0001.

[24] Jan Egger et al. "Nugget-cut: a segmentation scheme for spherically-and elliptically-shaped 3D objects". In: *Joint Pattern Recognition Symposium*. 2010, pp. 373–382.

[25] M. Everingham et al. *The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results*. http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html.

[26] M. Everingham et al. *The PASCAL Visual Object Classes Challenge 2010 (VOC2010) Results*. http://www.pascal-network.org/challenges/VOC/voc2010/workshop/index.html.

[27] B. Fishbain, Dorit S. Hochbaum, and Stefan Mueller. "A competitive study of the pseudoflow algorithm for the minimum s–t cut problem in vision applications". In: *Journal of Real-Time Image Processing* 11.3 (2016), pp. 589–609.

[28] Lester Randolph Ford Jr and Delbert Ray Fulkerson. *Flows in networks*. Princeton university press, 1962.

[29] Daniel Freedman and Petros Drineas. "Energy minimization via graph cuts: Settling what is possible". In: *IEEE Conference on Computer Vision and Pattern Recognition*. 2005, pp. 939–946.

[30] William T Freeman, Egon C Pasztor, and Owen T Carmichael. "Learning low-level vision". In: *International Journal of Computer Vision* 40 (2000), pp. 25–47.

[31] Francisco Garcı´a-Moreno et al. "Using X-ray tomoscopy to explore the dynamics of foaming metal". In: *Nature communications* 10.1 (2019), pp. 1–9.

[32] Andrew V Goldberg. "Processor-efficient implementation of a maximum flow algorithm". In: *Information Processing Letters* 38.4 (1991), pp. 179–185.

[33] Andrew V Goldberg. "The partial augment–relabel algorithm for the maximum flow problem". In: *European Symposium on Algorithms*. 2008, pp. 466–477.

[34] Andrew V Goldberg. "Two-level push-relabel algorithm for the maximum flow problem". In: *International Conference on Algorithmic Applications in Management*. 2009, pp. 212–225.

[35] Andrew V Goldberg and Robert E Tarjan. "A new approach to the maximum-flow problem". In: *Journal of the ACM* 35.4 (1988), pp. 921–940.

[36] Andrew V Goldberg et al. "Faster and More Dynamic Maximum Flow by Incremental Breadth-First Search". In: *European Symposium on Algorithms*. 2015, pp. 619–630.

[37] Andrew V Goldberg et al. "Maximum Flows by Incremental Breadth-First Search". In: *European Symposium on Algorithms*. 2011, pp. 457–468.

[38] Vicente Grau, J Crawford Downs, and Claude F Burgoyne. "Segmentation of trabeculated structures using an anisotropic Markov random field: application to the study of the optic nerve head in glaucoma". In: *IEEE Transactions on Medical Imaging* 25 (2006), pp. 245–255.

[39] Raymond Greenlaw, H. James Hoover, and Walter L. Ruzzo. *Limits to parallel computation: P-completeness theory*. Oxford University Press on Demand, 1995.

[40] D. M. Greig, B. T. Porteous, and A. H. Seheult. "Exact Maximum A Posteriori Estimation for Binary Images". In: *Journal of the Royal Statistical Society. Series B (Methodological)* 51.2 (1989), pp. 271–279.

[41] Lihong Guo et al. "Learned snakes for 3D image segmentation". In: *Signal Processing* 183 (2021), p. 108013.

[42] Zhihui Guo et al. "Deep LOGISMOS: deep learning graph-based 3D segmentation of pancreatic tumors on CT scans". In: *IEEE International Symposium on Biomedical Imaging*. 2018, pp. 1230–1233.

[43] Felix Halim, Roland HC Yap, and Yongzheng Wu. "A MapReduce-based maximum-flow algorithm for large small-world network graphs". In: *International Conference on Distributed Computing Systems*. 2011, pp. 192–202.

[44] Peter L Hammer, Pierre Hansen, and Bruno Simeone. "Roof duality, complementation and persistency in quadratic 0–1 optimization". In: *Mathematical Programming* 28.2 (1984), pp. 121–155.

[45] Dorit S. Hochbaum. "The pseudoflow algorithm: A new algorithm for the maximum-flow problem". In: *Operations Research* 56.4 (2008), pp. 992–1009.

[46] Dorit S. Hochbaum and James B. Orlin. "Simplifications and speedups of the pseudoflow algorithm". In: *Networks* 61.1 (2013), pp. 40–57.

[47] Bo Hong and Zhengyu He. "An asynchronous multithreaded algorithm for the maximum network flow problem with nonblocking global relabeling heuristic". In: *IEEE Transactions on Parallel and Distributed Systems* 22.6 (2010), pp. 1025–1033.

[48] Lisa Hutschenreiter et al. “Fusion Moves for Graph Matching”. In: *International Conference on Computer Vision*. 2021, pp. 6270–6279.

[49] Hossam Isack et al. “Efficient optimization for hierarchically-structured interacting segments (HINTS)”. In: *IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 1445–1453.

[50] Hiroshi Ishikawa. “Exact optimization for Markov random fields with convex priors”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 25.10 (2003), pp. 1333–1336.

[51] Ondřej Jamriška and Daniel Sỳkora. *GridCut. Version 1.3*. https://gridcut.com. Accessed 2020-06-12. 2015.

[52] Ondřej Jamriška, Daniel Sỳkora, and Alexander Hornung. “Cache-efficient Graph Cuts on Structured Grids”. In: *IEEE Conference on Computer Vision and Pattern Recognition*. 2012, pp. 3673–3680.

[53] Patrick M. Jensen, Anders B. Dahl, and Vedrana A. Dahl. “Multi-Object Graph-Based Segmentation With Non-Overlapping Surfaces”. In: *IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2020, pp. 976–977.

[54] Patrick M. Jensen and Niels Jeppesen. *Max-Flow/Min-Cut Algorithms*. https://doi.org/10.5281/zenodo.4903945. Accessed 2021-06-08. DOI: 10.5281/zenodo.4903945.

[55] Patrick M. Jensen et al. *Min-Cut/Max-Flow Problem Instances for Benchmarking*. https://doi.org/10.11583/DTU.17091101. Accessed 2021-11-29. DOI: 10.11583/DTU.17091101.

[56] Niels Jeppesen et al. “Faster Multi-Object Segmentation Using Parallel Quadratic Pseudo-Boolean Optimization”. In: *International Conference on Computer Vision*. Oct. 2021, pp. 6260–6269.

[57] Niels Jeppesen et al. “Sparse Layered Graphs for Multi-Object Segmentation”. In: *IEEE Conference on Computer Vision and Pattern Recognition*. 2020, pp. 12777–12785.

[58] Dagmar Kainmueller et al. “Active graph matching for automatic joint segmentation and annotation of C. elegans”. In: *International Conference on Medical Image Computing and Computer Assisted Intervention*. 2014, pp. 81–88.

[59] Jörg H Kappes et al. “A comparative study of modern inference techniques for structured discrete energy minimization problems”. In: *International Journal of Computer Vision* 115 (2015), pp. 155–184.

[60] S Kashyap, H Zhang, and M Sonka. “Accurate Fully Automated 4D Segmentation of Osteoarthritic Knee MRI”. In: *Osteoarthritis and Cartilage* 25 (2017), S227–S228.

[61] Anna Khoreva et al. “Simple does it: Weakly supervised instance and semantic segmentation”. In: *IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 876–885.

[62] Pushmeet Kohli and Philip H.S. Torr. “Dynamic graph cuts for efficient inference in Markov random fields”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 29.12 (2007), pp. 2079–2088.

[63] Vladimir Kolmogorov and Carsten Rother. “Minimizing nonsubmodular functions with graph cuts-a review”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 29.7 (2007), pp. 1274–1279.

[64] Vladimir Kolmogorov and Ramin Zabih. “Computing visual correspondence with occlusions using graph cuts”. In: *International Conference on Computer Vision*. Vol. 2. 2001, pp. 508–515.

[65] Vladimir Kolmogorov and Ramin Zabin. “What energy functions can be minimized via graph cuts?” In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 26.2 (2004), pp. 147–159.

[66] Nikos Komodakis and Nikos Paragios. “Beyond loose LP-relaxations: Optimizing MRFs by repairing cycles”. In: *European Conference on Computer Vision*. 2008, pp. 806–820.

[67] L’ubor Ladickỳ and Philip HS Torr. *The automatic labelling environment*. https://www.robots.ox.ac.uk/~phst/ale.htm. Accessed 2021-11-24.

[68] L’ubor Ladickỳ et al. “Associative hierarchical crfs for object class image segmentation”. In: *International Conference on Computer Vision*. 2009, pp. 739–746.

[69] Lubor Ladicky et al. “Graph cut based inference with co-occurrence statistics”. In: *European Conference on Computer Vision*. 2010, pp. 239–253.

[70] Kyungmoo Lee et al. “Multiresolution LOGISMOS graph search for automated choroidal layer segmentation of 3D macular OCT scans”. In: *Medical Imaging 2020: Image Processing*. Vol. 11313. International Society for Optics and Photonics. 2020, 113130B.

[71] Victor Lempitsky and Yuri Boykov. “Global optimization for shape fitting”. In: *IEEE Conference on Computer Vision and Pattern Recognition*. 2007, pp. 1–8.

[72] Victor Lempitsky, Yuri Boykov, and Denis Ivanov. “Oriented visibility for multiview reconstruction”. In: *European Conference on Computer Vision*. 2006, pp. 226–238.

[73] Marius Leordeanu, Rahul Sukthankar, and Martial Hebert. “Unsupervised learning for graph matching”. In: *International Journal of Computer Vision* 96 (2012), pp. 28–45.

[74] Kang Li et al. “Optimal surface segmentation in volumetric images-a graph-theoretic approach”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 28.1 (2005), pp. 119–134.

[75] Jiangyu Liu and Jian Sun. “Parallel Graph-cuts by Adaptive Bottom-up Merging”. In: *IEEE Conference on Computer Vision and Pattern Recognition*. 2010, pp. 2181–2188.

[76] Lei Liu et al. “Graph cut based mesh segmentation using feature points and geodesic distance”. In: *Proceedings of the International Conference on Cyberworlds (CW)*. 2015, pp. 115–120.

[77] Eric Maire and Philip John Withers. “Quantitative X-ray tomography”. In: *International materials reviews* 59.1 (2014), pp. 1–43.

[78] Leland McInnes, John Healy, and James Melville. “Umap: Uniform manifold approximation and projection for dimension reduction”. In: *arXiv:1802.03426* (2018).

[79] Fausto Milletari, Nassir Navab, and Seyed-Ahmad Ahmadi. “V-net: Fully convolutional neural networks for volumetric medical image segmentation”. In: *International Conference on 3D Vision*. 2016, pp. 565–571.

[80] Shervin Minaee et al. “Image segmentation using deep learning: A survey”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2021).

[81] Rajmund Mokso et al. “GigaFRoST: the gigabit fast readout system for tomography”. In: *Journal of synchrotron radiation* 24.6 (2017), pp. 1250–1259.

[82] Sebastian Nowozin et al. “Decision tree fields”. In: *International Conference on Computer Vision*. 2011, pp. 1668–1675.

[83] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning and Research* 12 (2011), pp. 2825–2830.

[84] Bo Peng, Lei Zhang, and David Zhang. “A survey of graph theoretical approaches to image segmentation”. In: *Pattern Recognition* 46.3 (2013), pp. 1020–1038.

[85] Sida Peng et al. “Deep snake for real-time instance segmentation”. In: *IEEE Conference on Computer Vision and Pattern Recognition*. 2020, pp. 8533–8542.

[86] Yi Peng et al. “JF-Cut: A parallel graph cut approach for large-scale image and video”. In: *IEEE Transactions on Image Processing* 24.2 (2015), pp. 655–666.

[87] Marius Reichardt et al. “3D virtual Histopathology of Cardiac Tissue from Covid-19 Patients based on Phase-Contrast X-ray Tomography”. In: *eLife.* (2021).

[88] Carsten Rother et al. “Optimizing binary MRFs via extended roof duality”. In: *IEEE Conference on Computer Vision and Pattern Recognition*. 2007, pp. 1–8.

[89] Alexander Shekhovtsov and Václav Hlaváč. “A distributed mincut/maxflow algorithm combining path augmentation and push-relabel”. In: *International Journal of Computer Vision* 104.3 (2013), pp. 315–342.

[90] Amber L Simpson et al. “A large annotated medical image dataset for the development and evaluation of segmentation algorithms”. In: *arXiv:1902.09063* (2019).

[91] Petter Strandmark and Fredrik Kahl. “Parallel and Distributed Graph Cuts by Dual Decomposition”. In: *IEEE Conference on Computer Vision and Pattern Recognition*. 2010, pp. 2085–2092.

[92] Paul Swoboda et al. “A study of lagrangean decompositions and dual ascent solvers for graph matching”. In: *IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 1607–1616.

[93] Nima Tajbakhsh et al. “Embracing imperfect datasets: A review of deep learning solutions for medical image segmentation”. In: *Medical Image Analysis* 63 (2020), p. 101693.

[94] Lorenzo Torresani, Vladimir Kolmogorov, and Carsten Rother. “Feature correspondence via graph matching: Models and global optimization”. In: *European Conference on Computer Vision*. 2008, pp. 596–609.

[95] Tanmay Verma and Dhruv Batra. “MaxFlow Revisited: An Empirical Comparison of Maxflow Algorithms for Dense Vision Problems”. In: *British Machine Vision Conference*. 2012, pp. 1–12.

[96] Vibhav Vineet and P J Narayanan. “CUDA cuts: Fast graph cuts on the GPU”. In: *IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2008, pp. 1–8.

[97] Yao Wang and Reinhard Beichel. “Graph-based segmentation of lymph nodes in CT data”. In: *International Symposium on Visual Computing*. 2010, pp. 312–321.

[98] University of Waterloo. *Max-flow problem instances in vision*. https://vision.cs.uwaterloo.ca/data/maxflow. Accessed 2021-02-05.

[99] Udaranga Wickramasinghe et al. “Voxel2mesh: 3d mesh model generation from volumetric data”. In: *International Conference on Medical Image Computing and Computer Assisted Intervention*. Springer. 2020, pp. 299–308.

[100] Xiaodong Wu and Danny Z Chen. “Optimal net surface problems with applications”. In: *International Colloquium on Automata, Languages, and Programming*. 2002, pp. 1029–1042.

[101] Yin Yin et al. “LOGISMOS—layered optimal graph image segmentation of multiple objects and surfaces: cartilage segmentation in the knee joint”. In: *IEEE Transactions on Medical Imaging* 29.12 (2010), pp. 2023–2037.

[102] Miao Yu, Shuhan Shen, and Zhanyi Hu. “Dynamic Graph Cuts in Parallel”. In: *IEEE Transactions on Image Processing* 26.8 (2017).

[103] Miao Yu, Shuhan Shen, and Zhanyi Hu. “Dynamic Parallel and Distributed Graph Cuts”. In: *IEEE Transactions on Image Processing* 25.12 (2015), pp. 5511–5525.

Patrick M. Jensen was born in Copenhagen, Denmark, in 1994. He received his B.Sc.Eng degree in 2017 and M.Sc.Eng degree in 2019, both in applied mathematics, at the Technical University of Denmark (DTU), Kgs. Lyngby, Denmark. He is currently pursuing a Ph.D. in 3D image analysis at the Visual Computing group at the Department of Applied Mathematics and Computer Science, Technical University of Denmark. His research interests lie in 3D image segmentation.

Niels Jeppesen is an image analysis and machine learning specialist at FORCE Technology with a Ph.D. degree in image analysis of 3D structures from the Department of Applied Mathematics and Computer Science, Technical University of Denmark (DTU), Kgs. Lyngby, Denmark. His research interests lie in min-cut/max-flow algorithms and quantitative analysis of structures in 3D images. He applies these methods for automated quality control of structures and materials, in particular, in the wind turbine industry.

Anders Bjorholm Dahl is professor in 3D image analysis, and a head of the Section for Visual Computing at the Department of Applied Mathematics and Computer Science, Technical University of Denmark (DTU), Kgs. Lyngby, Denmark. He is heading The Center for Quantification of Imaging Data from MAX IV, focusing on quantitative analysis of 3D images. His research is focused on image segmentation and its applications.

Vedrana Andersen Dahl is an associate professor at the Department of Applied Mathematics and Computer Science, Technical University of Denmark (DTU), Kgs. Lyngby, Denmark. Her primary research interest is in the use of geometric models for the analysis of volumetric data. This includes volumetric segmentation and methods based on deformable meshes. She developed analysis tools with applications in material science, industrial inspection, and biomedicine.

3.2 Paper B: Faster Multi-Object Segmentation using Parallel Quadratic Pseudo-Boolean Optimization

Niels Jeppesen, Patrick M. Jensen, Anders N. Christensen, Anders B. Dahl, Vedrana A. Dahl, IEEE/CVF International Conference on Computer Vision (ICCV), 2021.
DOI: 10.1109/ICCV48922.2021.00620

Faster Multi-Object Segmentation using Parallel Quadratic Pseudo-Boolean Optimization

Niels Jeppesen, Patrick M. Jensen, Anders N. Christensen, Anders B. Dahl, and Vedrana A. Dahl
Department of Applied Mathematics and Computer Science
Technical University of Denmark, Kgs. Lyngby, Denmark
{niejep,patmjen,anym,abda,vand}@dtu.dk

Abstract

We introduce a parallel version of the Quadratic Pseudo-Boolean Optimization (QPBO) algorithm for solving binary optimization tasks, such as image segmentation. The original QPBO implementation by Kolmogorov and Rother relies on the Boykov-Kolmogorov (BK) maxflow/mincut algorithm and performs well for many image analysis tasks. However, the serial nature of their QPBO algorithm results in poor utilization of modern hardware. By redesigning the QPBO algorithm to work with parallel maxflow/mincut algorithms, we significantly reduce solve time of large optimization tasks. We compare our parallel QPBO implementation to other state-of-the-art solvers and benchmark them on two large segmentation tasks and a substantial set of small segmentation tasks. The results show that our parallel QPBO algorithm is over 20 times faster than the serial QPBO algorithm on the large tasks and over three times faster for the majority of the small tasks. Although we focus on image segmentation, our algorithm is generic and can be used for any QPBO problem. Our implementation and experimental results are available at DOI: 10.5281/zenodo.5201620

1. Introduction

Computational parallelism is essential to the performance and thereby the usefulness of many image segmentation algorithms. The best example is perhaps deep learning, which owes much of its success to highly efficient parallel implementations of the matrix operations used during both training and inference. However, not all algorithms used in computer vision rely on easily parallelizable matrix operations.

Graph cut algorithms are popular for solving binary optimization problems in image analysis, due to their speed and guarantee of optimality. Thus, they provide efficient solutions to a variety of computer vision problems – on their own [8, 9, 11, 16, 23, 26, 33, 35], or in combination with other methods [18, 29, 30]. While some popular graph cut algorithms have been parallelized [11, 36, 41, 43], other algorithms have remained serial, which severely limits their ability to utilize modern hardware. One example is the Quadratic Pseudo-Boolean Optimization (QPBO) algorithm [7, 19, 33, 39], which allows non-submodular energy terms, making it particularly useful for instance segmentation. Instance segmentation without training data is common in microscopy and material science, where manually labeling large volumetric datasets can be highly impractical. Often, the input needed for segmentation with QPBO can be obtained much easier.

In this paper, we introduce the first parallel QPBO (P-QPBO) algorithm. Our goal is to provide an efficient and scalable algorithm that can take advantage of modern multi-core processors. With our P-QPBO algorithm, results can be obtained over an order of magnitude faster than with previous serial methods, and the scale of the tasks can be increased significantly. It enables us to segment a volume with hundreds of interacting 3D objects in minutes based on limited user input and no training data. Although we only demonstrate the advantage of P-QPBO for image segmentation, P-QPBO can be used for any QPBO problem.

This work focuses on our parallel algorithm and its time and memory efficiency on image segmentation tasks. Thus, the formulation of suitable energy functions for specific computer vision tasks is outside the scope of this paper.

1.1. Related work

Several algorithms have been developed to solve QPBO problems [19, 32, 33]. In computer vision, the QPBO algorithm [7, 19] implemented by Kolmogorov and Rother [33, 39], utilizing the serial Boykov-Kolmogorov maxflow/mincut (BK) algorithm [9] for solving the optimization problem, is arguably the most popular. Generally, maxflow/mincut algorithms can be separated into three groups [42]: push-relabel algorithms [3, 10, 14, 15], augmenting path algorithms [9, 17], and pseudoflow algorithms [16, 20, 21], which are a hybrid of the two previous categories. BK is an augmenting path algorithm. It is the

most popular maxflow/mincut algorithm in computer vision, due to its performance and ability to handle dynamic maxflow/mincut scenarios, by reusing computations from previous solutions when changes are made to the graph [31]. In the last decade, pseudoflow algorithms like Excesses Incremental Breadth-First Search (EIBFS) [16] have outperformed BK in most static cases, as well as some dynamic cases. However, the overhead associated with graph changes for dynamic problems is still higher for EIBFS than for BK.

Push-relabel algorithms have traditionally been the target of most parallelization efforts [2, 5, 11, 13, 22, 43], as operations mainly act locally, making them well-suited for parallel execution. However, synchronization overhead means that many threads are needed to achieve good performance [6, 40]. More recent works [36, 40, 41, 44, 45] have focused on parallelizing augmenting path algorithms. Here, a graph is partitioned into multiple sub-graphs and a serial algorithm is applied to each sub-graph in parallel. Information is then propagated between sub-graphs, or they are merged. This process is repeated until a global solution is found. Parallel pseudoflow algorithms have not been attempted yet.

Finally, as grid-based graphs are relatively common in computer vision, algorithms specialized for this structure, such as Grid-Cut [24, 25], have also been developed. Grid structured graphs have also been the target of GPU-based implementations [38, 43], as they rely on the highly regular structure of grid graphs to fully utilize the GPU. While grid-based algorithms achieve significant performance improvements, they are only usable on a limited (but certainly important) subset of binary optimization problems. In this paper, we are concerned with a general-purpose parallel QPBO algorithm and will therefore not discuss the grid-based algorithms further.

1.2. Contribution

We introduce a fast parallel algorithm for solving QPBO problems. It is based on the efficient two-stage approach of the QPBO algorithm as presented in [33] and the bottom-up merging approach from [36]. Our algorithm is fully compatible with the original QPBO algorithm and we prove that it is guaranteed to find equivalent solutions.

We show that our parallel algorithm reduces the solve time significantly on a large multi-object 3D segmentation task compared to current state-of-the-art approaches. We also benchmark our algorithm for segmentation using a large set of 2D images and show significant performance improvements, even for smaller segmentation tasks.

Our implementation, benchmark code, results, notebooks, and proof are available at 10.5281/zenodo.5201620.

2. QPBO

We briefly summarize the original QPBO algorithm here. Both the QPBO algorithm and general-purpose maxflow/mincut algorithms can be used to minimize energy functions of the form

$$E(\mathbf{x}) = \sum_{p \in \mathcal{V}} \theta_p(x_p) + \sum_{p,q \in \mathcal{V}} \theta_{pq}(x_p, x_q) \,. \qquad (1)$$

Here, $\mathcal{V}$ is a set of nodes, $x_p \in \{0, 1\}$ are the node labels, θ_p are unary energy terms, and θ_{pq} are pairwise energy terms. If E is submodular, meaning that all pairwise energies satisfy

$$\theta_{pq}(0,0) + \theta_{pq}(1,1) \leq \theta_{pq}(0,1) + \theta_{pq}(1,0) \,, \qquad (2)$$

then maxflow/mincut-based algorithms (including QPBO) are guaranteed to find the global optimal solution to the minimization problem [9, 33]. However, if the energy function contains non-submodular terms, we cannot directly model it as a maxflow/mincut problem [12, 34]. To overcome this limitation, the QPBO algorithm uses an extended graph approach, in which every node is represented by two graph nodes: a node $p \in \mathcal{V}$ and a flipped node $\overline{p} \in \overline{\mathcal{V}}$. Every energy term is then represented by two graph edges (see Table 1). This allows the non-submodular terms to be represented as graph edges between nodes p and q, and the flipped nodes $\overline{p}$ and $\overline{q}$. Computing the maximum flow/minimum cut of this extended graph corresponds to minimizing the energy function (1) [33] given that all terms are non-negative. One can convert a QPBO function to an equivalent non-negative one using the linear time algorithm described in [33].

Table 1. Conversion from energy terms to graph edges [33], where s the source node, t is the sink node, and $\overline{p}, \overline{q} \in \overline{\mathcal{V}}$ are the flipped versions of the nodes $p, q \in \mathcal{V}$.

Energy term	Corresponding edges	Edge capacity
$\theta_p(0)$	$(p \to t), (s \to \overline{p})$	$\frac{1}{2}\theta_p(0)$
$\theta_p(1)$	$(s \to p), (\overline{p} \to t)$	$\frac{1}{2}\theta_p(1)$
$\theta_{pq}(0,1)$	$(p \to q), (\overline{q} \to \overline{p})$	$\frac{1}{2}\theta_{pq}(0,1)$
$\theta_{pq}(1,0)$	$(q \to p), (\overline{p} \to \overline{q})$	$\frac{1}{2}\theta_{pq}(1,0)$
$\theta_{pq}(0,0)$	$(p \to \overline{q}), (q \to \overline{p})$	$\frac{1}{2}\theta_{pq}(0,0)$
$\theta_{pq}(1,1)$	$(\overline{q} \to p), (\overline{p} \to q)$	$\frac{1}{2}\theta_{pq}(1,1)$

However, the support for non-submodular terms means that the guarantee of finding the global optimal solution is replaced by one of finding a partial optimal solution [33]. This means that we may get a solution with unlabeled nodes, which may lead to bad segmentation results [23]. However, when non-submodular terms are used only for exclusion, [26] has shown that unlabeled nodes are rare and hardly affect the resulting segmentation.

The original QPBO algorithm uses an efficient two-stage approach [33]. In Stage 1, only submodular terms are considered and the problem is modelled and solved as a regular

maxflow problem, *i.e.*, without the flipped nodes. In Stage 2, the flipped graph is created by copying the residual graph from Stage 1 and reversing the edges. Finally, the edges for the non-submodular terms are added and the solution is updated. This two-stage approach reduces the solve time significantly, but relies on maxflow solvers that can handle dynamic graphs efficiently, *e.g.*, the BK algorithm [31].

3. Parallel QPBO

We now describe our new Parallel QPBO (P-QPBO) algorithm, which combines the two-stage QPBO approach described in Section 2 with the bottom-up merging parallelization approach by Liu and Sun [36]. Judging from previous work [36, 40, 41, 44, 45], bottom-up merging provides good performance on non-distributed multi-core systems. Like Liu and Sun's algorithm, ours has two phases. In Phase A, the QPBO problem is split into disjoint sub-problems, which are solved independently in parallel. In Phase B, these partial solutions are merged and re-solved, also in parallel, to get the complete solution. Note that Phase A/B strictly refers to the splitting and merging of sub-problems. Stage 1/2 refers to whether the sub-graph associated with a sub-problem has had the flipped graph added or not.

In contrast to Liu and Sun's algorithm, which strictly works as a maxflow/mincut solver, our algorithm also considers each sub-problem as a QPBO problem. Specifically, each sub-problem is kept in Stage 1 (where we do not need the flipped graph) as long as it contains only submodular terms. Thus, in the case of few non-submodular terms, most sub-problems will remain in Stage 1 for most of Phases A and B. This significantly reduces the solve time. We will now describe the two phases, including the specific conditions, which will trigger a conversion to Stage 2 for a sub-problem. Figure 1 shows a visual summary.

Phase A: Partitioning of the QPBO problem is done by splitting the underlying graph $\mathcal{G} = \langle \mathcal{V} \cup \overline{\mathcal{V}}, \mathcal{E} \rangle$. We split the node set $\mathcal{V}$ into N disjoint sets $\mathcal{V}_1, \mathcal{V}_2, ..., \mathcal{V}_N$. This gives a partition of the graph nodes into blocks $\mathcal{W}_1, \mathcal{W}_2, ..., \mathcal{W}_N$ where $\mathcal{W}_i = \{p, \overline{p} \mid p \in \mathcal{V}_i\}$. Then, for each pair of blocks $\mathcal{W}_i, \mathcal{W}_j$ connected by one or more edges, we identify inter-block edges and store these in separate lists. From now on, we refer to these edge lists as *block interfaces*. After building the block interfaces we remove inter-block edges from $\mathcal{G}$.

We now have a series of sub-graphs $\mathcal{G}_i = \langle \{\mathcal{W}_i, s, t\}, \mathcal{E}_i \rangle$ where $\mathcal{E}_i = \{(p \to q) \in \mathcal{E} \mid p, q \in \mathcal{W}_i\}$. Because these sub-graphs are disconnected, except through the source and sink nodes, we can compute their individual maxflow solutions in parallel (see Figure 1a). For each sub-graph, we adapt the two-stage approach from the serial QPBO algorithm. First, we only consider submodular terms and do not add the flipped graph. Then, if (and only if) a sub-graph contains non-submodular terms, we transition the sub-graph to Stage 2. During this transition, the flipped graph is constructed by copying the residual graph from Stage 1, the remaining non-submodular edges are added, and the maxflow solution is updated (see Figure 1b). When all sub-graphs have been solved, we move to Phase B.

Phase B: In this phase we merge the sub-graphs to re-create the original extended graph, $\mathcal{G}$. Merging two sub-graphs is done by re-adding the inter-block edges, which were removed in Phase A. If all sub-graphs and inter-block edges correspond to submodular terms, we keep both sub-graphs in Stage 1 (see Figure 1c). If some of the inter-block edges correspond to non-submodular terms, both sub-graphs are transformed to Stage 2 (see Figure 1d) before the inter-block edges are added (see Figure 1e). Furthermore, if the two sub-graphs are in different stages, the sub-graph in Stage 1 is transformed to Stage 2 before inter-block edges are re-added and the sub-graphs are merged (see Figure 1f). After merging, the solution of the combined graph is updated.

To further reduce the solve time, we want merges to happen in parallel, for which we use the strategy from [36]. Updating the maxflow solution is a serial process, so only one thread can work on a sub-graph at a time. For synchronization, each sub-graph can be locked (meaning it is being worked on) or unlocked (meaning it is free for merging).

To decide which sub-graphs to merge, each thread scans through the list of block interfaces created in Phase A, until it finds one that connects two unlocked sub-graphs. The thread then locks the sub-graphs and merges them. Then, it re-computes the maxflow solution for the merged sub-graph and the sub-graph is unlocked. Note that after sub-graphs have been merged, there may be several block interfaces connecting the previously merged sub-graphs. Therefore, when a thread finds a pair of sub-graphs to merge, it continues to scan the list of block interfaces to find all block interfaces connecting the pair. The block interfaces are then removed from the global list and the merge proceeds. A global synchronization object is used to ensure that only one thread can scan the list of block interfaces at a time.

At the end of Phase B, the number of remaining merges will be less than the number of running threads (unless only one thread is used). Therefore, if a thread scans the whole list of block interfaces without encountering a pair of unlocked sub-graphs, it terminates. As a result, the degree of parallelism is gradually reduced near the end of Phase B. However, for most problems, the time required for the last merge will be small compared to the total solve time. In total, the number of merges performed will be one less than the number of blocks. The process for each thread is summarized in Algorithm 1.

3.1. Correctness

Our P-QPBO algorithm will always give a solution equivalent to that of the serial QPBO algorithm.

Energy: The energy of the solution is given by the unique

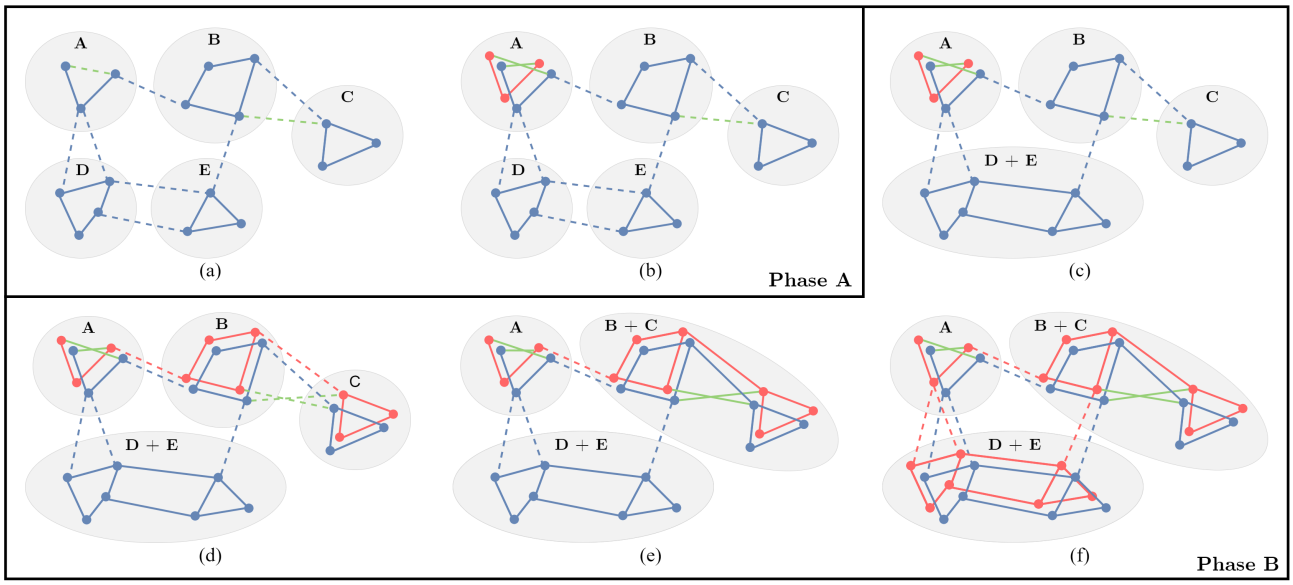


Figure 1. Illustration of merging strategy. Blue dots and lines represent nodes and edges in the non-flipped graph, while red dots and lines are nodes and edges in the flipped graph. Green lines represent edges between a node $p \in \mathcal{V}$ and flipped node $\overline{q} \in \overline{\mathcal{V}}$, corresponding to non-submodular terms. Dashed lines represent inter-block edges, which are re-added when the sub-graphs are merged. The exception is green dashed lines between two blue nodes. These represent non-submodular energy terms, which will be translated to edges, once the flipped graphs are added. Source/sink nodes and edges are not shown. (a) The graph is split into sub-graphs and the Stage 1 solution for each sub-graph is computed in parallel. (b) Sub-graph A contains internal non-submodular terms, so it is transformed to Stage 2 and the solution is updated. (c) Sub-graphs D and E are merged. Inter-block edges are re-added and the sub-graph solution is updated. As all intra- and inter-block terms are submodular the sub-graph remains in Stage 1. (d) A term between B and C is non-submodular, so the sub-graphs are transformed to Stage 2 to prepare for merge. (e) Sub-graphs B and C are merged. (f) Sub-graph D + E is transferred to Stage 2 to allow merges with the remaining sub-graphs. All sub-graphs are now in Stage 2, and merging can proceed as normal bottom-up merging.

value of the minimum cut for the extended graph. Since the final graph is identical for the serial and parallel algorithms, and they both compute a minimum cut, the solutions must have the same energy.

Labeling: There may be several minimum cuts which have the same cost/energy but label a different number of nodes [33]. However, given a residual graph, the algorithm from [4, 33] will choose the minimum cut that labels the maximum number of nodes. It can be shown (proof in supplementary material) that since both QPBO and P-QPBO compute a minimum cut of the same graph, they must label the same nodes after running this extra algorithm. Since this extra step is an insignificant part of the overall runtime, we do not include it in our runtime experiments.

3.2. Efficient graph partitioning and merging

The partitioning of the graph nodes into blocks is important for the performance of the P-QPBO algorithm. While our method allows for any partitioning of nodes, ideally, we want as much work as possible to be done in Phase A (computing the partial solutions) and as little as possible to be done in Phase B (merging sub-graphs and updating solutions). A good way to achieve this is to separate the nodes into blocks that are densely packed (many intra-block terms) and sparsely related (few inter-block terms). This speeds up the merging by reducing the number of changes made to the graph. Of course, the ideal partitioning very much depends on the energy function.

For image segmentation, we can use the spatial position of the nodes/pixels when partitioning them into blocks. Cutting the image into evenly sized rectangular blocks, as done by [36, 41] should result in many intra-block terms, compared to inter-block terms, as long as the blocks are not very small. When solving instance segmentation tasks using Sparse Layered Graphs (SLG) [26], an intuitive way to partition the nodes is to create a block per label/object. This works well when the interaction between the objects is low compared to the size of the objects (which is usually the case), and we have at least as many objects as the number of parallel threads available on the system. We use this natural way of partitioning the nodes for all our experiments, as most of our images contain many objects.

Algorithm 1: Phase B of the parallel QPBO algorithm for each thread.

```
while true do
    Lock synchronization object
    Let S = ∅
    foreach block interface s do
        Let G_i and G_j be sub-graphs connected by s
        if both G_i and G_j are unlocked then
            S = {s_i | s_i connects G_i and G_j}
            break
    Remove entries of S from list of block interfaces
    if S is empty then
        Unlock synchronization object
        return
    Lock sub-graphs G_i and G_j connected by S
    Unlock synchronization object
    /* Ensure sub-graphs are in stage 2 if needed. */
    if S contains non-submodular terms or G_i in stage 2
     or G_j in stage 2 then
        if G_i in stage 1 then Transform G_i to stage 2
        if G_j in stage 1 then Transform G_j to stage 2
    Unite sub-graphs G_i and G_j to sub-graph G_ij
    /* Re-insert boundary edges */
    foreach inter-block edge e in S do
        Reinsert e in graph
        if G_ij in stage 2 then
            Reinsert flipped edge of e in graph
        if nodes of a have different labels then
            Mark nodes of reinserted edges as active
    Update maxflow for subgraph G_ij
    /* Make G_ij available for merges */
    Lock synchronization object
    Unlock sub-graph G_ij
    Unlock synchronization object
```

For determining the merging order, P-QPBO uses the same approach as Liu and Sun [36]. After Phase A, we loop over each block interface and count the number of potential new augmenting paths, when merging the sub-graphs containing the blocks. This serves as a heuristic for how much work must be done when merging the sub-graphs. The list of block interfaces is then sorted in descending order based on the number of potential new augmenting paths, in the hope that threads will perform the most expensive merges first. The goal is to do as much work as possible early in Phase B, while the degree of parallelism is high.

4. Benchmark results

To test the scalability of our P-QPBO algorithm, we compare it with two serial QPBO implementations. The first is a slightly optimized implementation of the original QPBO algorithm by Kolmogorov – we call it K-QPBO. The reason we are using a slightly modified version is that the original implementation has overflow issues for large graphs. The second serial implementation is our own re-implementation of K-QPBO, which contains numerous improvements in data structures and optimizations of the code that improves performance. We call this implementation Modern QPBO (M-QPBO). M-QPBO is included to provide a more fair comparison between a serial and parallel implementation since M-QPBO contains the same performance optimization as P-QPBO. When referring to results for our parallel implementation, we use the notation P-QPBO(t), where t is the number of parallel threads used by P-QPBO.

We test the QPBO implementations on the two datasets used in [26], and use the exact energy functions shared in [27]. Our notebooks (based on [27]), used to formulate the energy functions and to benchmark the QPBO algorithms, are included in our supplementary material (DOI: 10.5281/zenodo.5201620). However, as our focus in this paper is purely on the computational performance, the energy formulations are not included in the paper.

The first dataset used for our experiments is a high-resolution μCT 3D image of nerves [28] shown in Figure 2a. This is a large segmentation task with many non-overlapping objects. It allows us to test the scalability of the parallel QPBO implementation across many CPU threads. The second dataset is the BBBC038v1 `stage1_train` (S1) nuclei image set from the Broad Bioimage Benchmark Collection [37]. An image from the dataset along with the instance segmentation results is shown in Figure 2b. Using these images, we test the performance of the QPBO implementations on a variety of small and medium-sized segmentation tasks. For both datasets, the energy function creates a nontrivial graph topology consisting of irregularly interconnected ordered multi-column sub-graphs. Unlike general maxflow problems, where a number of commonly used benchmark datasets exist [16], there are no commonly used benchmark datasets specifically for QPBO.

We use two Intel Xeon Gold 6226R (16 cores / 16 threads) CPUs in dual socket configuration for all our benchmarks. With this, we test how our implementation scales on a modern architecture with up to 32 threads executing in parallel.

4.1. Large segmentation tasks

The goal of this experiment is to compare the solve times of the K-QPBO and M-QPBO to those of P-QPBO at various parallel thread counts on large segmentation tasks. Although solve times vary between system architectures, this experiment shows the benefit of using P-QPBO, depending on the number of CPU cores available.

In the experiment, we segment the myelin and axon of 216 nerves in a $2048 \times 2048 \times 2048$ volume at two different radial sampling resolutions, using the SLG method of [26]. The first resolution (N1) is the one used by [26], while the second resolution (N2) is higher, resulting in a graph more

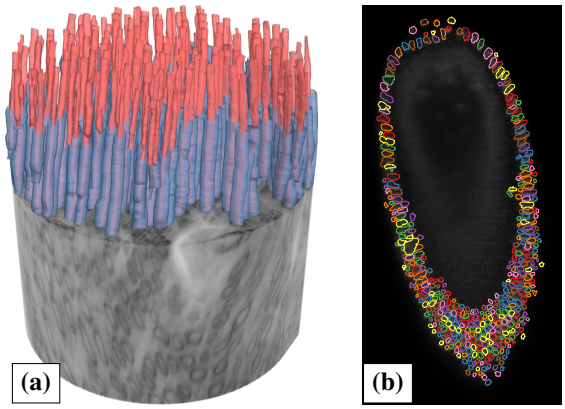


Figure 2. (a) Result of N1 nerve segmentation task. Nodes are split into blocks such that all nodes associated with either the inner (red) or outer (blue) surface of a nerve are in the same block (two blocks per nerve). (b) Example of nuclei segmentation on the image from S1 with the most nuclei. Nodes are split into blocks such that all nodes associated with a cell are in the same block (one block per cell).

	N1	N2
Nodes	363,748,800	818,434,800
Edges	2,124,073,454	4,864,255,488
Memory footprint		
K-QPBO [33]	134.2 GB	306.8 GB
M-QPBO	60.1 GB	182.7 GB
P-QPBO	70.0 GB	224.9 GB
Fastest solve time		
K-QPBO [33]	836 s	4,987 s
M-QPBO	628 s	3,704 s
P-QPBO (1)	558 s	2,429 s
P-QPBO (16)	94 s	323 s
P-QPBO (32)	80 s	264 s
P-QPBO (40)	**74 s**	245 s
P-QPBO (48)	76 s	**239 s**

Table 2. Graph details for the nerve segmentation tasks. Nodes and edges refer to the size of the full extended graph. The memory footprint is the total memory footprint of the graph and relevant bookkeeping. P-QPBO and M-QPBO use 32-bit indices for N1, but 64-bit edge indices for N2, because it has more than 2^{31} edges. K-QPBO always uses 64-bit pointers for indexing. The solve times are shown for each of the three algorithms, with a number of different thread configurations for P-QPBO. Each solve time is the minimum of ten runs for N1 and three runs for N2.

than twice the size of N1 (see Table 2). In both cases, the output is a 3D multi-label segmentation with a total of 432 interacting objects (two per nerve). For P-QPBO we use one block per object (see Figure 2a).

As shown in Table 2, our M-QPBO implementation reduces the solve time for the N1 task by 25% compared to the K-QPBO implementation, while P-QPBO(1) outperforms M-QPBO slightly for this task, with a 33% reduction compared to K-QPBO, using only a single thread. Using two threads, P-QPBO(2) provides a 62% reduction in solve time, compared to K-QPBO, and a 49% reduction compared to M-QPBO. The best result is achieved using 40 threads, in which case P-QPBO is over 11 times faster than K-QPBO. Figure 3a show the relative speed-up when using P-QPBO compared to K-QPBO. We see that the performance increases up to and beyond the number of CPU cores (32) in our test system.

For the larger N2 task, we observe even larger performance improvements and better scaling of P-QPBO than for N1 (see Figure 3b). From the solve times in Table 2, we see that the bottom-up merging strategy, even without parallelism, provides a reduction in solve time of 51% for P-QPBO(1) compared to K-QPBO. Meanwhile, M-QPBO provides a 26% reduction over K-QPBO. In other words, P-QPBO clearly improves its relative performance as the task grows, while M-QPBO performs similarly for N1 and N2, when looking at the relative improvement over K-QPBO.

Both Figures 3a and 3b show that the speed-up increases significantly less past 16 threads. This is expected, as we are testing on a dual socket system, which means we are likely to experience some degree of computational overhead when using both CPUs, especially for cache and memory intensive tasks such as computing the maximum flow. Yet, despite the overhead, the combined 32 CPU cores allow P-QPBO to scale past 32 threads for both N1 and N2, with P-QPBO(40) significantly outperforming P-QPBO(32) in both cases. This is perhaps a result of some threads idling while waiting for the synchronization lock to be released.

Another reason for the way P-QPBO scales with the number of threads is the reduction in the degree of parallelism at the end of Phase B. According to Amdahl's law [1], this puts a theoretical maximum to the speed-up, which in our case will depend on the energies and blocking strategy. For the nerve segmentation tasks we estimate a parallel fraction of 0.88 and 0.92 (including overhead) for N1 and N2, respectively, meaning that most of the work is done in parallel.

We expect that most of the performance improvement of M-QPBO over K-QPBO is due to the smaller memory footprint of the graphs shown in Table 2. We achieve this reduction by using more compact data structures for nodes and edges. Instead of 64-bit pointers, we use 32-bit indices where possible. Furthermore, we store forward and backward edges adjacent in memory to avoid storing pointers between these. P-QPBO and M-QPBO use the same fundamental data structures for nodes and edges. The increased

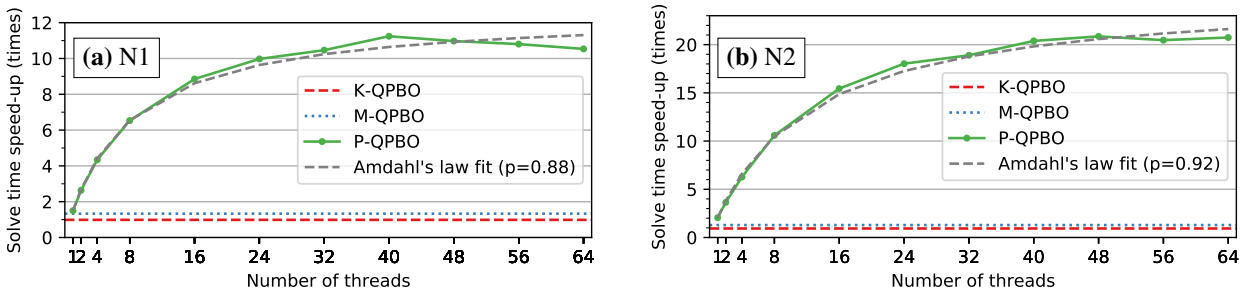


Figure 3. Plots showing the relative speed-up in the solve time when using P-QPBO compared to K-QPBO and M-QPBO. The speed-up is calculated using the fastest solve time out of ten runs for N1 and three runs for N2. K-QPBO and M-QPBO are represented as horizontal lines, as they always use a single thread. We also show a fit of Amdahl's law [1] and the parallel fraction, p. Keep in mind that two 16 core CPUs were used, which means we expect the speed-up to stagnate or even decrease when using more than parallel 32 threads. For these tasks, the stagnation appears to start at 40 threads on our test system.

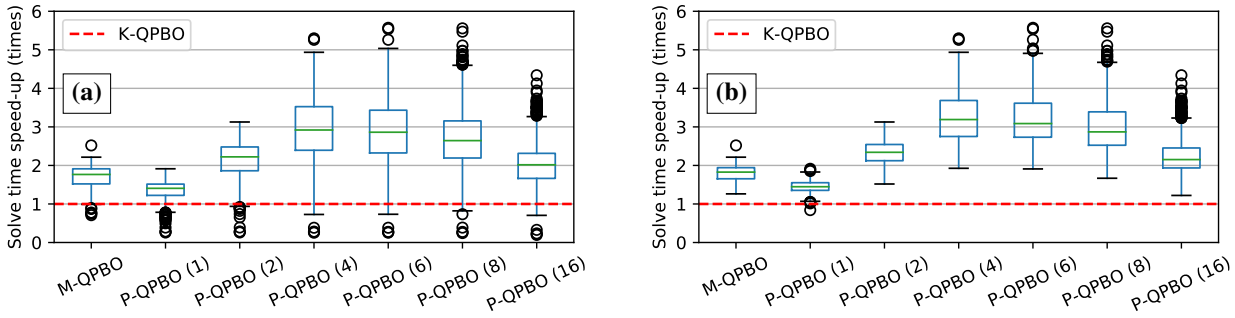


Figure 4. Box plots showing the relative speed-up for each image in the S1 dataset, for M-QPBO and P-QPBO compared to K-QPBO. Following Tukey's definition, the green line in the box is the median, the box marks the two quartiles and the whiskers show the minimum and maximum values, excluding outliers. Outliers are defined as values more than 1.5 times the interquartile range from the nearest quartile and are shown as rings. In (a) the results for all 670 images are shown, while (b) only includes the 502 images with 16 or more nuclei. The relative speed-up is calculated using the fastest solve time for each method, with each method having been run ten times.

memory footprint of the P-QPBO graph is a result of extra bookkeeping needed for the bottom-up merging. Reducing the memory footprint of the graph structures is important for two reasons. 1) It increases performance due to improved CPU cache and memory efficiency. 2) It allows us to solve larger tasks without running out of memory.

It is important to remember that the scaling depends both on the optimization problem and the system architecture. Generally, we would expect M-QPBO to outperform P-QPBO(1) on smaller tasks, due to the overhead of merging the sub-graphs. However, for large tasks, using bottom-up merging, even without parallel computations, actually turns out to be faster. This behavior was previously noted by [16, 36] and is probably due to a combination of shorter augmenting paths and better cache efficiency.

For the N1 task, [26] reported a solve time of 44 minutes for K-QPBO, which is much higher than the 14 minutes we found in our experiments. We suspect that the main reason for the big difference is that their system only had 112 GB memory, while the graph has a footprint of at least 134 GB. This could have caused memory swapping, which would likely impact performance negatively.

4.2. Smaller segmentation tasks

We use the S1 dataset, previously used in [26], to compare the performance of P-QPBO, M-QPBO, and K-QPBO on a large set of 2D (non-grid) segmentation problems of varying sizes. Figure 5 shows the distributions of graph nodes and edges for the images. With a median of 437,400 nodes and 1,598,060 edges, we consider most of these segmentation tasks relatively small. A few of the tasks are significantly larger, with the largest (shown in Figure 2b) having just over six million nodes and 60 million edges. To examine the overall performance of M-QPBO and P-QPBO for these small to medium-sized tasks, we compute the relative speed-up when using our implementations compared to K-QPBO.

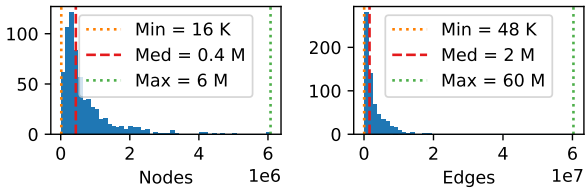


Figure 5. Histograms of the distribution of nodes and edges for the graphs used when segmenting the images in S1.

Figure 4 shows the relative speed-up for M-QPBO and P-QPBO for each image in the dataset (Figure 4a) and for each image with 16 or more nuclei (Figure 4b). Both M-QPBO and P-QPBO show a significant improvement compared to K-QPBO. When we include all images, there are cases where the relative performance drops. In these few cases, the tasks are very small (few nodes and terms), such that the overhead of P-QPBO outweighs the benefits. If we look only at the 502 images with 16 or more nuclei (259,200 nodes or more), M-QPBO and P-QPBO significantly outperform K-QPBO for all images, except when using P-QPBO with a single thread. For these smaller tasks, the overhead of merging blocks is not outweighed by the shorter augmenting paths. Thus, when using only a single thread, the best performance is achieved without bottom-up merging.

For the images with 16 or more nuclei (Figure 4b), M-QPBO gives a median speed-up of 1.8x, with a maximum of 2.5x. P-QPBO(4) achieves the best overall performance, with a median speed-up of 3.2x and a maximum of 5.3x. While P-QPBO(6) and P-QPBO(8) show the best performance in a few of the largest tasks, the overall performance decreases slightly when compared to using four threads, due to the majority of the tasks being relatively small.

4.3. Comparison with other solvers

We compare P-QPBO with other state-of-the-art maxflow/mincut algorithms. To test the effect of the two-stage QPBO strategy, we compare with our own implementation of the parallel maxflow/mincut algorithm by Liu and Sun [36]. Furthermore, we compare with the serial EIBFS algorithm [16], as it is currently the fastest serial maxflow/mincut solver, and we compare with our own parallel version of EIBFS (P-EIBFS) based on bottom-up merging. Finally, we include a best case estimate for a QPBO implementation using EIBFS as its maxflow/mincut solver (EIBFS-QPBO).

We compare the algorithms on the N1 and S1 tasks. The maxflow/mincut algorithms are evaluated by first converting the QPBO problem to the full extended graph and then running the algorithm on this graph. We do not include this conversion time in the benchmark. Results are shown in Table 3. We see that our algorithm significantly outperforms the other methods.

	Memory	Best speed-up	
	N1	N1	S1
M-QPBO	60.1 GB	1.33	1.72±0.28
P-QPBO	70.0 GB	**10.47**	**2.96±0.89**
Liu-Sun [36]	70.0 GB	6.07	0.78±0.13
EIBFS [16]	175.1 GB	1.08	0.33±0.08
P-EIBFS	175.8 GB	0.63	0.68±0.14
EIBFS-QPBO*	175.1 GB	2.17	0.66±0.08

*Best case estimate (half of EIBFS solve time).

Table 3. Results of ablation experiment. We consider up to 32 threads for N1 and up to 16 for S1. We report memory and best speed-up for N1 and best mean±std. speed-up for the S1 data. Speed-ups are computed w.r.t. K-QPBO. The maxflow/mincut solvers were run on the full extended graph. We do not include the time used to convert the QPBO problem to a graph.

5. Conclusion

Our P-QPBO algorithm is the first parallel QPBO algorithm. It scales much better than the serial K-QPBO algorithm on modern multi-core hardware, by partitioning the task into sub-tasks and solving them in parallel. It uses a bottom-up merging strategy to combine the solutions, also in parallel. This allows P-QPBO to solve tasks, such as image segmentation, significantly faster than current algorithms.

Our experiments show that P-QPBO solves large multi-object segmentation tasks over 20 times faster than K-QPBO, with lower memory usage. It does so while remaining fully compatible with K-QPBO, making no constraining assumptions about the graph structure. Even for smaller tasks, with just a few hundred thousand nodes, P-QPBO is 2-5 times faster than K-QPBO, using only four threads. This indicates that P-QPBO will significantly outperform K-QPBO, even on consumer hardware.

The scalability of P-QPBO, when combined with modern hardware, makes P-QPBO suitable for solving much larger optimization tasks than previously possible. Furthermore, because it is a parallel algorithm, we expect the relative performance of P-QPBO to keep increasing in the future. Finally, P-QPBO is a general algorithm, which is suitable for many binary optimization tasks, not just image segmentation. Thus, we are confident that P-QPBO can be used not just for faster image segmentation, but also for a wide range of other tasks, both in computer vision and other fields.

Acknowledgements

This work is supported by FORCE Technology and The Center for Quantification of Imaging Data from MAX IV (QIM).

References

[1] Gene M Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, spring joint computer conference*, pages 483–485, 1967. 6, 7

[2] Richard Anderson and Joao C. Setubal. A parallel implementation of the push-relabel algorithm for the maximum flow problem. *Journal of Parallel and Distributed Computing (JPDC)*, 29(1):17–26, 1995. 2

[3] Chetan Arora, Subhashis Banerjee, Prem Kalra, and SN Maheshwari. An efficient graph cut algorithm for computer vision problems. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 552–565, 2010. 1

[4] Bengt Aspvall, Michael F Plass, and Robert Endre Tarjan. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Information Processing Letters*, 8(3):121–123, 1979. 4

[5] David A Bader and Vipin Sachdeva. A cache-aware parallel implementation of the push-relabel network flow algorithm and experimental evaluation of the gap relabeling heuristic. Technical report, Georgia Institute of Technology, 2006. 2

[6] Niklas Baumstark, Guy Blelloch, and Julian Shun. Efficient implementation of a synchronous parallel push-relabel algorithm. In *Proceedings of the European Symposium on Algorithms (ESA)*, pages 106–117, 2015. 2

[7] Endre Boros, Peter L Hammer, and Xiaorong Sun. Network flows and minimization of quadratic pseudo-boolean functions. Technical report, Technical Report RRR 17-1991, RUTCOR, 1991. 1

[8] Yuri Boykov and Gareth Funka-Lea. Graph Cuts and Efficient N-D Image Segmentation. *International Journal of Computer Vision*, 70(2):109–131, nov 2006. 1

[9] Yuri Boykov and Vladimir Kolmogorov. An Experimental Comparison of Min-Cut/Max-Flow Algorithms for Energy Minimization in Vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 26(9):1124–1137, 2004. 1, 2

[10] Boris V Cherkassky and Andrew V Goldberg. On implementing the push—relabel method for the maximum flow problem. *Algorithmica*, 19(4):390–410, 1997. 1

[11] Andrew Delong and Yuri Boykov. A scalable graph-cut algorithm for nd grids. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8, 2008. 1, 2

[12] Daniel Freedman and Petros Drineas. Energy minimization via graph cuts: Settling what is possible. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 939–946, 2005. 2

[13] Andrew V Goldberg. Processor-efficient implementation of a maximum flow algorithm. *Information Processing Letters*, 38(4):179–185, 1991. 2

[14] Andrew V Goldberg. The partial augment–relabel algorithm for the maximum flow problem. In *Proceedings of the European Symposium on Algorithms (ESA)*, pages 466–477, 2008. 1

[15] Andrew V Goldberg. Two-level push-relabel algorithm for the maximum flow problem. In *International Conference on Algorithmic Applications in Management*, pages 212–225, 2009. 1

[16] Andrew V Goldberg, Sagi Hed, Haim Kaplan, Pushmeet Kohli, Robert E Tarjan, and Renato F Werneck. Faster and More Dynamic Maximum Flow by Incremental Breadth-First Search. In *Proceedings of the European Symposium on Algorithms (ESA)*, pages 619–630, 2015. 1, 2, 5, 7, 8

[17] Andrew V Goldberg, Sagi Hed, Haim Kaplan, Robert E Tarjan, and Renato F Werneck. Maximum Flows by Incremental Breadth-First Search. In *Proceedings of the European Symposium on Algorithms (ESA)*, pages 457–468, 2011. 1

[18] Zhihui Guo, Ling Zhang, Le Lu, Mohammadhadi Bagheri, Ronald M Summers, Milan Sonka, and Jianhua Yao. Deep LOGISMOS: deep learning graph-based 3D segmentation of pancreatic tumors on CT scans. pages 1230–1233, 2018. 1

[19] Peter L Hammer, Pierre Hansen, and Bruno Simeone. Roof duality, complementation and persistency in quadratic 0–1 optimization. *Mathematical Programming*, 28(2):121–155, 1984. 1

[20] Dorit S. Hochbaum. The pseudoflow algorithm: A new algorithm for the maximum-flow problem. *Operations Research*, 56(4):992–1009, 2008. 1

[21] Dorit S. Hochbaum and James B. Orlin. Simplifications and speedups of the pseudoflow algorithm. *Networks*, 61(1):40–57, 2013. 1

[22] Bo Hong and Zhengyu He. An asynchronous multithreaded algorithm for the maximum network flow problem with nonblocking global relabeling heuristic. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 22(6):1025–1033, 2010. 2

[23] Hossam Isack, Olga Veksler, Ipek Oguz, Milan Sonka, and Yuri Boykov. Efficient optimization for hierarchically-structured interacting segments (HINTS). In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1445–1453, 2017. 1, 2

[24] Ondřej Jamriška and Daniel Sỳkora. GridCut. Version 1.3. https://gridcut.com, 2015. Accessed 2020-06-12. 2

[25] Ondřej Jamriška, Daniel Sỳkora, and Alexander Hornung. Cache-efficient Graph Cuts on Structured Grids. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3673–3680, 2012. 2

[26] Niels Jeppesen, Anders N Christensen, Vedrana A Dahl, and Anders B Dahl. Sparse layered graphs for multi-object segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12777–12785, 2020. 1, 2, 4, 5, 7

[27] Niels Jeppesen, Anders Nymark Christensen, Vedrana Andersen Dahl, and Anders Bjorholm Dahl. Sparse Layered Graphs for Multi-Object Segmentation (notebooks). 6 2020. 5

[28] Niels Jeppesen, Anders Nymark Christensen, Vedrana Andersen Dahl, Anders Bjorholm Dahl, Hans Martin Kjer, Martin Bech, and Lars Dahlin. Sparse Layered Graphs for Multi-Object Segmentation (data). 11 2020. 5

[29] Anna Khoreva, Rodrigo Benenson, Jan Hosang, Matthias Hein, and Bernt Schiele. Simple does it: Weakly supervised instance and semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 876–885, 2017. 1

[30] Alexander Kirillov, Evgeny Levinkov, Bjoern Andres, Bogdan Savchynskyy, and Carsten Rother. Instancecut: from edges to instances with multicut. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5008–5017, 2017. 1

[31] Pushmeet Kohli and Philip H.S. Torr. Dynamic graph cuts for efficient inference in Markov random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 29(12):2079–2088, 2007. 2, 3

[32] Vladimir Kolmogorov. Convergent tree-reweighted message passing for energy minimization. In *Proceedings of the International Workshop on Artificial Intelligence and Statistics*, pages 182–189, 2005. 1

[33] Vladimir Kolmogorov and Carsten Rother. Minimizing non-submodular functions with graph cuts-a review. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 29(7):1274–1279, 2007. 1, 2, 4, 6

[34] Vladimir Kolmogorov and Ramin Zabin. What energy functions can be minimized via graph cuts? *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 26(2):147–159, 2004. 2

[35] Kang Li, Xiaodong Wu, Danny Z Chen, and Milan Sonka. Optimal surface segmentation in volumetric images-a graph-theoretic approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 28(1):119–134, 2005. 1

[36] Jiangyu Liu and Jian Sun. Parallel Graph-cuts by Adaptive Bottom-up Merging. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2181–2188, 2010. 1, 2, 3, 4, 5, 7, 8

[37] V. Ljosa, K. L. Sokolnicki, and A. E. Carpenter. Annotated high-throughput microscopy image sets for validation. *Nature Methods*, 9(7):637–637, 2012. 5

[38] Yi Peng, Li Chen, Fang Xin Ou-Yang, Wei Chen, and Jun Hai Yong. JF-Cut: A parallel graph cut approach for large-scale image and video. *IEEE Transactions on Image Processing*, 24(2):655–666, 2015. 2

[39] Carsten Rother, Vladimir Kolmogorov, Victor Lempitsky, and Martin Szummer. Optimizing binary MRFs via extended roof duality. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8, 2007. 1

[40] Alexander Shekhovtsov and Václav Hlaváč. A distributed mincut/maxflow algorithm combining path augmentation and push-relabel. *International Journal of Computer Vision (IJCV)*, 104(3):315–342, 2013. 2, 3

[41] Petter Strandmark and Fredrik Kahl. Parallel and Distributed Graph Cuts by Dual Decomposition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2085–2092, 2010. 1, 2, 3, 4

[42] Tanmay Verma and Dhruv Batra. MaxFlow Revisited: An Empirical Comparison of Maxflow Algorithms for Dense Vision Problems. In *Proceedings of the British Machine Vision Conference (BMVC)*, pages 1–12, 2012. 1

[43] Vibhav Vineet and P J Narayanan. CUDA cuts: Fast graph cuts on the GPU. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 1–8, 2008. 1, 2

[44] Miao Yu, Shuhan Shen, and Zhanyi Hu. Dynamic Parallel and Distributed Graph Cuts. *IEEE Transactions on Image Processing*, 25(12):5511–5525, 2015. 2, 3

[45] Miao Yu, Shuhan Shen, and Zhanyi Hu. Dynamic Graph Cuts in Parallel. *IEEE Transactions on Image Processing*, 26(8), 2017. 2, 3

3.3 Paper C: Multi-object Graph-based Segmentation with Non-overlapping Surfaces

Patrick M. Jensen, Anders B. Dahl, Vedrana A. Dahl,
IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW),
Workshop on Computer Vision for Microscopy Image Analysis (CVMI), 2020
DOI: 10.1109/CVPRW50498.2020.00496

Multi-object Graph-based Segmentation with Non-overlapping Surfaces

Patrick M. Jensen, Anders B. Dahl, Vedrana A. Dahl
Department of Applied Mathematics and Computer Science
Technical University of Denmark, Kgs. Lyngby, Denmark
{patmjen,abda,vand}@dtu.dk

Abstract

For 3D images, segmentation via fitting surface meshes to object boundaries provides an efficient way to handle large images and enforce geometric prior knowledge. Furthermore, fitting such meshes with graph cuts has proven to be a versatile and robust framework. However, when segmenting multiple distinct objects in one image, current methods do not allow the natural constraint that objects should not overlap. In this paper, we present an extension to graph cut based methods which can provide a globally optimal segmentation of thousands of objects while guaranteeing no overlap. Our method works by separating objects with planes whose positions are determined as part of the graph cut. To demonstrate the general applicability of our method, we apply it to several 3D microscopy data sets from both biology and materials science. Our results show both quantitative and qualitative improvements.

1. Introduction

3D microscopy includes techniques such as X-ray micro- or nano-CT, light-sheet microscopy, optical coherence tomography, and confocal microscopy. These techniques, widely used in materials and bio-science, often result in large 3D images that are difficult to interpret through visual inspection. It is therefore important to be able to quantify 3D structures, *e.g.* the size of cells, and here segmentation is an essential tool.

Segmenting multiple objects that are densely packed can easily result in an overlap between the detected objects, which is not physically possible. Therefore, it is often desirable to constrain the segmentation such that the detected objects do not intersect. Besides giving a sensible solution, avoiding overlap also regularizes the segmentation in general. Thus, the resulting segmentation of individual objects is more accurate than without this constraint.

In this work, we propose a method for accurately segmenting multiple densely packed objects while preventing overlap. Our approach uses an s-t graph cut to fit surface meshes to object boundaries. Using surface meshes provides a compact representation of the segmentation, allowing us to process large volumes with a large number of objects.

In general, incorporating non-overlap constraints directly into the graph cut formulation is challenging, and existing approaches are based on assumptions that limits their application. Examples are situations where only a few objects are to be detected, where objects have a certain shape, or where user-provided scribbles are available.

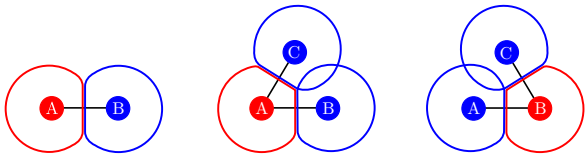
We propose a method to incorporate non-overlap constraints for problems where pairs of interacting objects can be separated by a plane. This is always the case for convex objects [2], but can also be valid for non-convex shapes. As a result, our approach can be applied to a range of problems involving the segmentation of non-overlapping densely packed objects. Our method allows us to simultaneously detect thousands of non-overlapping objects by finding a globally optimal solution to the modeled segmentation problem. We provide an implementation of the method at: https://github.com/patmjen/NOS.

1.1. Approaches for preventing overlap

By definition, an s-t graph cut provides a bipartition of a graph, so binary segmentation is the most straightforward use of the s-t graph cut in image analysis. Segmenting multiple objects with a single graph involves constructing sub-graphs dedicated to each object. This opens a possibility for adding interactions between objects, such as *containment* and *exclusion*, by adding edges between nodes of the sub-graphs. While containment is relatively easy to enforce, exclusion (preventing overlap) is in general challenging.

Several authors have explored the topic of preventing overlap in graph-based multi-object segmentation [1, 7, 12, 20, 25]. The key challenge is that graph cuts can only be used to minimize so-called submodular energy functions. Adding Exclusion typically results in non-submodularity and is therefore not straightforward to model with graphs. A common way to circumvent this is to represent some objects with their complement, which turns exclusion between A and B into containment of A in B^c. This means that non-

overlap may only be enforced between pairs of surfaces. More specifically, we must divide our surfaces into two groups, and separation can only be enforced between two surfaces if they are in different groups. A simple example is illustrated in Figure 1.



(a) Surfaces A and B split into groups {A} and {B}.

(b) Surfaces A, B, and C grouped as {A} and {B, C}.

(c) Surfaces A, B, and C grouped as {A, C} and {B}.

Figure 1: Usual construction of separation constraints for multiple surfaces. For the case in (a) we can easily split two surfaces into groups. For (b) and (c) there will always be a pair of surfaces with no separation constraints.

Pairwise exclusion may suffice for some applications, especially if only a few objects are to be segmented, *e.g.* organ segmentation in medical image analysis. However, for our problem this type of construction is not adequate – we will need to support separation constraints between an arbitrary number of surfaces. Furthermore, previous methods usually require that we identify a region of interaction for pairs of surfaces, and then add edges between corresponding nodes in the graph for each surface [12, 20, 25]. This may require re-meshing of the surfaces or significant changes to the graph, which in turn may be highly non-trivial if several surfaces share a region of interaction.

A way of handling non-submodular energy terms is to use a more general framework, quadratic pseudo-boolean optimization (QPBO). If QPBO provides labeling for all nodes, the solution is guaranteed to be optimal. However, QPBO might leave some graph nodes unlabeled, which is typically problematic when enforcing exclusion in pixel-wise segmentation. Still, unlabeled nodes will not be an issue if relatively few energy terms are non-submodular, and for our approach, QPBO often provides a globally optimal solution.

2. Method

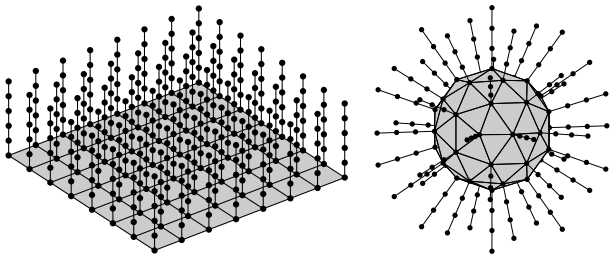
Our method builds upon existing approaches for graph cut segmentation. To make the paper self-contained, we first review the two most relevant algorithms for our work: graph-based surface detection, and quadratic pseudo-boolean optimization. Next, we detail our extension which ensures that the fitted surface meshes are separated by planes. Finally, we cover a few implementational details of our approach. For clarity, we reserve the terms *vertex* and *edge* for meshes and use *node* and *arc* when dealing with graphs.

2.1. Surface fitting with graph cuts

Wu and Chen [24] were the first to suggest using an s-t graph cut, and its polynomial-time solution, to detect surfaces in a 3D image. The approach was initially used for terrain-like and tubular surfaces [15]. For terrain-like surfaces, the solution is found by searching along the columns of the volume. Tubular surfaces are found by searching radially along rays from the tube center. An interpretation of this approach, which can be generalized to any shape, involves a meshed *base-surface*.

In this interpretation, the meshed base-surface is fit to the data by displacing the mesh vertices along the vertex normals. The goal is to place the vertices at the boundary of the object of interest, such that the region inside the mesh gives a segmentation of the object. The optimal displacement of vertices is found using a s-t graph cut.

While base-surfaces for terrain-like and tubular objects may be a regular quad mesh (see Figure 2, left), this may not be suitable for other shapes. For roughly spherical objects, a regular polyhedron may be used [8, 23] (see Figure 2, right). A different strategy involves using a rough initial segmentation as a base-surface [25].



(a) Quad. mesh.

(b) Regular polyhedron.

Figure 2: Examples of different base-surfaces and the positions of the graph nodes computed from the mesh. The nodes belonging to a normal-aligned column are connected by lines.

The underling graph construction is shared by all approaches, and is summarized in the following. With a *base-mesh* as an input, we will construct a graph $G = (\mathcal{V} \cup \{s, t\}, \mathcal{E})$ with nodes $\mathcal{V}$, a source node s, a sink node t, and arcs $\mathcal{E}$.

For a node set, $\mathcal{V}$, we consider each mesh vertex i, with position $\mathbf{v}_i$, and generate a series of candidate positions along the (outward) vertex normal, $\mathbf{n}_i$, as

$$\mathbf{v}_{i,k} = \mathbf{v}_i + k\,\delta_{\text{step}}\,\mathbf{n}_i \quad \text{for} \quad k = 0, 1, ..., n_{\text{step}}. \tag{1}$$

Here, δ_{step} and n_{step} are user defined parameters which define the step length and number of steps, respectively. With each candidate positions $\mathbf{v}_{i,k}$ we associate a graph node $v_{i,k} \in \mathcal{V}$. We will refer to the set of nodes,

$$C_i = \{v_{i,k} \mid k = 0, 1, ..., n_{\text{step}}\}, \tag{2}$$

associated with a mesh vertex i as the *normal-aligned column* of i. Furthermore, we say that two normal aligned columns are neighbors, if their corresponding mesh vertices are connected with an edge.

The arc set $\mathcal{E}$ consists of terminal arcs and internal arcs. Internal arcs impose geometric constraints on the solution, such that vertices have a well-defined position along each normal-aligned column, and that the displacement of two neighboring vertices does not vary more than a pre-set value Δ. The internal arcs have infinite weight and consist of:

Intracolumn arcs $(v_{i,k}, v_{i,k-1})$ between successive nodes of each normal-aligned column C_i. Here $k = 1, 2, ..., n_{\text{step}}$.

Intercolumn arcs $(v_{i,k}, v_{j,l})$ from nodes of each normal-aligned column C_i, to nodes of a neighboring normal-aligned column C_j, where $l = \max\{0, k - \Delta\}$.

These arcs are illustrated in Figure 3. The parameter Δ constrains the displacement of neighbouring vertices relative to the base-mesh. If the base-mesh is chosen to be smooth, then Δ can be thought of as a regularization parameter.

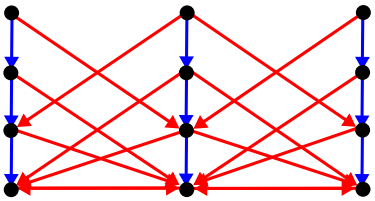


Figure 3: Illustration of intracolumn arcs (blue) and intercolumn arcs (red) for $\Delta = 2$.

The terminal arcs connect graph nodes with either the source s or the sink t. Those arcs have finite weights derived from a cost function which is given by the image values. Several cost functions have been used in the literature, and they typically lead to slightly different weight functions. The most suitable cost function is often problem-specific with popular choices being based on image edges [15] or regions [10]. A region-based cost function is constructed such that it attains negative values where image data supports object, and positive values where image data supports background.

If the node $v_{i,k}$ has a negative weight $-w$ it will be connected to the source with an arc $(s, v_{i,k})$ having a weight w. If the node has a positive weight w it will be connected to the sink with an arc$(v_{i,k}, t)$ having a weight w. However, we always connect the innermost node in each normal aligned column to the source, to guarantee at least one node is included in the object.

It has been shown [15, 24] that the vertex displacements which result in the minimum total cost, while satisfying the constraints set by Δ, can be found by computing a minimum s-t cut in this arc weighted graph. Such a cut splits the nodes into two disjoint subsets; the source set, S, and the sink set, T. Nodes in S are then defined to be inside the object of interest and vice versa for T. Mesh vertices are then moved to the position associated with the outermost node in their normal aligned columns which is in S. The s-t cut is often computed with the efficient algorithm by Boykov and Kolmogorov [3]. Furthermore, several extensions to this basic construction have since been developed [4].

Setting up a segmentation via surface fitting requires some knowledge of the geometry of the segmentation problem, or pre-processing, in order to get an appropriate initialization.

2.2. Quadratic pseudo-boolean optimization

It turns out that the construction from the previous section can be reformulated in a more general framework known as quadratic pseudo-boolean optimization (QPBO). The benefit of this is that QPBO allows us to model constraints that are not possible with the construction in Section 2.1 [9, 14]. Specifically, it enables us to enforce that two graph nodes cannot both be in the source set, which is what allows us to ensure that surfaces do not overlap.

A quadratic pseudo-boolean (QPB) function is a function $f : \{0, 1\}^n \to \mathbb{R}$ which can be written as

$$f(\mathbf{x}) = \theta_{\text{const}} + \sum_{i=1}^{n} \theta_i(x_i) + \sum_{i=1}^{n} \sum_{j>i} \theta_{ij}(x_i, x_j). \quad (3)$$

Computing a minimum s-t cut can then be reformulated as minimizing the following QPB energy function [13]

$$E(\mathbf{x}) = \theta_{\text{const}} + \sum_{v \in \mathcal{V}} \theta_v(x_v) + \sum_{(u,v) \in \mathcal{E}} \theta_{uv}(x_u, x_v), \quad (4)$$

where

$$x_v = \begin{cases} 0, & v \text{ is in the source set } S, \\ 1, & \text{otherwise.} \end{cases} \quad (5)$$

To define the θ_v and θ_{uv} functions, it will be convenient to use the following notation

$$\theta_{v,i} = \theta_v(i) \quad \text{and} \quad \theta_{uv,ij} = \theta_{uv}(i, j). \quad (6)$$

Furthermore, unless we explicitly specify a value for $\theta_{v,i}$ and $\theta_{uv,ij}$ we assume it to be zero. Now, let $w((u, v))$ be the weight of arc (u, v), and add the following unary terms

$$\theta_{v,0} = w((s, v)) \quad \text{and} \quad \theta_{v,1} = w((v, t)), \quad (7)$$

and the following binary terms

$$\theta_{uv,01} = w((u, v)) \quad \text{and} \quad \theta_{uv,10} = w((v, u)). \quad (8)$$

Minimizing $E(\mathbf{x})$ then corresponds to computing the minimum s-t cut on G.

Several papers have explored methods to minimize QPB functions [11, 13, 19]. If (and only if) the function is submodular, meaning that all binary functions satisfy

$$\theta_{uv,00} + \theta_{uv,11} \leq \theta_{uv,01} + \theta_{uv,10}, \quad (9)$$

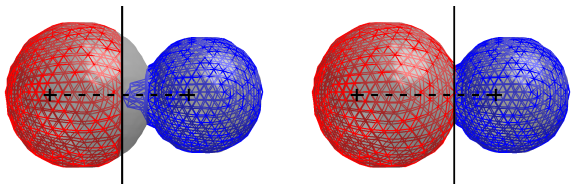
then the minimization can be done with the construction in Section 2.1 [9, 14]. In general, however, the minimization of QPB functions is NP-hard [13]. Thus, many methods focus on computing partial solutions, where each variable, x_i, can be given the value 0, 1, or unknown. The strength is that if 0 or 1 is assigned, then it is guaranteed to be the globally optimal value. The downside is that some variables may remain unassigned.

For our application, we found that even using the basic method of [13] would very often result in fully assigned (globally optimal) solutions. This method works by building a graph, similar to the construction in Section 2.1, with two nodes, a primal and dual, for every binary variable x_i. Non-submodular energy terms can then be modelled by adding edges between primal and dual nodes, and the minimization is done by computing a minimum s-t cut.

Finally, for the few cases where a full assignment was not immediately returned, a small adjustment of the method parameters would then fix it.

2.3. Adding overlap constraints

We now detail our extension, which allows us to enforce that surfaces do not overlap. Our construction proceeds as follows; assume we have two objects, A and B, to segment. Let $\mathbf{c}_\mathrm{A}$ and $\mathbf{c}_\mathrm{B}$ be the center of the base-surface meshes for object A and B, respectively. Now, consider a plane P with normal vector $\mathbf{n}_\mathrm{P} = \mathbf{c}_\mathrm{B} - \mathbf{c}_\mathrm{A}$ placed between A and B. If we constrain each mesh to not cross the plane, we would ensure that they cannot overlap. However, if the plane is not placed optimally the resulting segmentation may be subpar (see Figure 4). Therefore, to make the method more robust, we



(a) Bad plane placement. (b) Good plane placement.

Figure 4: Importance of plane placement for the segmentation of two overlapping balls. The center of each base-mesh is marked with a '+'. In (a) the plane is placed too far to the left. In (b) the plane is placed correctly which has resulted in each object being well segmented.

want the position of the plane to be determined dynamically as part of the segmentation problem.

To achieve this, first build graphs $G_\mathrm{A} = (\mathcal{V}_\mathrm{A} \cup \{s,t\}, \mathcal{E}_\mathrm{A})$ and $G_\mathrm{B} = (\mathcal{V}_\mathrm{B} \cup \{s,t\}, \mathcal{E}_\mathrm{B})$, according to the graph construction from Section 2.1. Then, merge the graphs into $G = (\mathcal{V}, \mathcal{E}) = (\mathcal{V}_\mathrm{A} \cup \mathcal{V}_\mathrm{B} \cup \{s,t\}, \mathcal{E}_\mathrm{A} \cup \mathcal{E}_\mathrm{B})$. Next, generate a number of candidate positions $\mathbf{p}_0, \mathbf{p}_1, ..., \mathbf{p}_n$ where

$$\mathbf{p}_i = \frac{i}{n}\mathbf{c}_\mathrm{B} + \left(1 - \frac{i}{n}\right)\mathbf{c}_\mathrm{A}, \tag{10}$$

and $n = \lfloor \|\mathbf{c}_\mathrm{B} - \mathbf{c}_\mathrm{A}\| / \delta_\mathrm{step} \rfloor$. Recall that δ_step is the step size used for building the normal-aligned columns for each surface graph, cf. Section 2.1. Notice that positions start at object A and then move towards object B. Now, add nodes $u_1, u_2, ..., u_n$ to $\mathcal{V}$ where u_k is associated with $\mathbf{p}_k$. Furthermore, add arcs (u_i, u_{i-1}) for $i = 1, 2, ..., n$ to $\mathcal{E}$ where each arcs has infinite weight. Note that this construction is the same as for the normal-aligned columns, see Figure 5. Thus, we can define the final position of the plane as $\mathbf{p}_i$ where i is the smallest number such that $u_i \in T$.

Next, we add interaction between the surfaces and the planes. Here, it will be more convenient to work with the QPB energy function $E(\mathbf{x})$, cf. Section 2.2. Iterate over all candidate plane positions, $\mathbf{p}_i$, and add energy terms to $E(\mathbf{x})$ according to the following rules:

1. For every normal-aligned column $C \subset \mathcal{V}_\mathrm{A}$, find the first node $v_k \in C$ whose position $\mathbf{v}_k$ satisfies $(\mathbf{v}_k - \mathbf{p}_i)^T \mathbf{n}_P > 0$. Add the term $\theta_{v_k u_i, 01} = \infty$ to $E(\mathbf{x})$ (see Figure 5, left). This will enforce that if $v_k \in S$ then $u_i \in S$. We only need to consider this first node since the construction of the surface graph ensures that if any later node $v_l \in S$ then $v_k \in S$.

2. For all normal-aligned columns $C \subset \mathcal{V}_\mathrm{B}$ find the first node $v_k \in C$ whose position $\mathbf{v}_k$ satisfies $(\mathbf{v}_k - \mathbf{p}_i)^T \mathbf{n}_P < 0$. Add the term $\theta_{v_k u_i, 00} = \infty$ (see Figure 5, right). This term acts like an exclusion arc as it ensures that if either v_k or u_i are in S, then the other cannot. Again, we only consider the first node as $v_k \in T$ will ensure that all later nodes are also in T.

These terms ensure that surface A and B never cross the plane and therefore cannot overlap. The complete construction is visualized in Figure 5. Intuitively, terms from step 1 will cause surface A to 'push' the plane away as it grows, and terms from step 2 will cause the plane to 'push' on surface B. As both surfaces grow to fit the data they will move the plane to a position of equilibrium where no surface can grow without degrading the overall segmentation. Note that the terms from step 2 are not submodular, which means we must use QPBO based methods to minimize the energy.

When dealing with more than two surfaces, we first construct a graph for each surface. Then, all graphs are merged into one graph and the above construction is added for each pair of interacting surfaces. Finally, the method from [13] is used to perform all segmentations simultaneously. If no variables are unassigned, the resulting segmentation is thus guaranteed to be globally optimal while having no overlap.

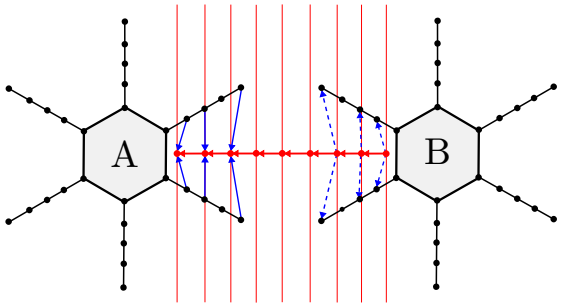


Figure 5: Illustration of the construction used to separate two surfaces with a moving plane. Graphs for two objects A and B are shown, but without intra- and intercolumn arcs. Red nodes indicate candidate plane positions, and each candidate plane is shown as a vertical red line. Red arcs are added to pairs of candidate position nodes pointing left, as candidate points are ordered going from A to B. Interaction terms from step 1 are shown as solid blue arcs, and terms from step 2 are shown as dashed blue arcs.

2.4. Detecting potential overlaps

We now describe a strategy for determining which meshes need to have overlap constraints added. One could add constraints between every pair of surfaces, but this would result in an unnecessarily large graph. As the run-time and resource use of the QPBO method is related to the size of the graph, we want it to be as small as possible.

Often, one chooses a base-surface mesh which approximates a sphere of some radius r centered at a point $\mathbf{c}$. In this case, the node farthest from the center in each normal aligned column will have distance $d = r + \delta_{\text{step}} n_{\text{step}}$ from $\mathbf{c}$. Thus, the fitted mesh will be contained in a bounding sphere of radius d centered at $\mathbf{c}$.

From this, it follows that two meshes (whose base-surface approximates a sphere) can only overlap if their bounding spheres intersect. This happens when $D < d_A + d_B$, where D is the distance between the mesh centers, and d_A, d_B are the radii of the two bounding spheres. Therefore, when adding overlap constraints, we first compute all pairwise center distances, and then only add constraints for meshes whose bounding spheres intersect.

For more general meshes, one could also use distance fields, bounding boxes, or, in the case of a few objects, manual annotation to determine which meshes could overlap.

3. Results and discussion

We now apply the developed method to three data sets with known ground truth segmentations. The first consists of a simulated fluorescence microscopy image of HL60 cell nuclei (see Figure 6a) [22]. As the data is computer-generated, we know the ground truth segmentation for the entire volume. The task is to segment the cells and the main challenge is the low contrast between foreground and background and the small distance between cells. We increased the segmentation difficulty further by adding Poisson and Gaussian noise (std. dev. of 40 with image intensity values between 0 and 255) per the description in [21]. We refer to this data set as *simulated cells*.

The second data set is a 3D confocal microscopy image of C. Elegans embryos (see Figure 6b) [17]. Here, we only have ground truth segmentations for a single xy-slice. Again, the task is to segment the cells and the challenge is the high noise level of the image and poorly defined object boundaries. Furthermore, we again have many closely packed objects. We refer to this data set as *cells*.

The third data set is a 3D X-ray tomographic microscopy image of liquid foam (see Figure 6c) [16, 18] from the TomoBank data repository [6]. As no ground truth is provided, we have manually annotated a single xy-slice to quantitatively evaluate the performance of the methods. The task is to segment individual foam bubbles, which is made difficult since the walls between bubbles are often not visible in the 3D image. We refer to this data set as *foam*.

The sizes of the data sets are shown in Table 1.

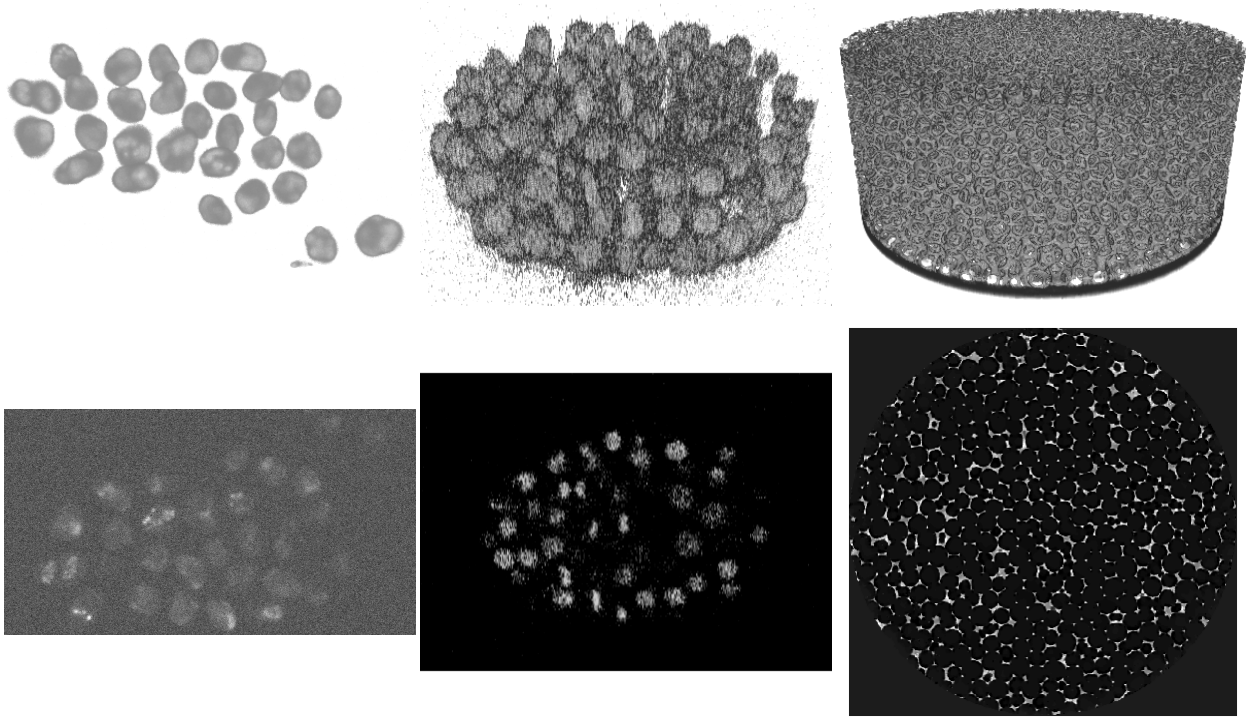
Data set	Size [voxels]	Voxel size [µm]
Sim. cells [22]	349 × 639 × 59	0.125 × 0.125 × 0.200
Cells [17]	512 × 708 × 35	0.090 × 0.090 × 1.000
Foam [18, 16]	504 × 504 × 230	6.000 × 6.000 × 6.000

Table 1: Size of the data sets.

For the segmentation of the images, we place a geodesic 4-frequency subdivided icosahedron at the center of each object. For the simulated cells and cells data set, center positions were manually annotated. For the foam data set, bubble centers were found by first binarizing the image and then finding local maxima of the distance transform. The radius of the base-mesh was chosen as one voxel. When sampling along the normal-aligned columns, steps were scaled to move an equal number of µm along each axis. We use the region based cost function from [10]. The parameters used for segmentation are shown in Table 2. For the cells data set, the in-region and out-region costs were scaled such that their sum over the image region resulted in the same value.

We evaluated the segmentation performance using the Dice similarity coefficient (also known as F1 score), boundary F1 (BF) score [5], and Jaccard score (also known as intersection over union). For the simulated cells data set, a score was computed for each segmented object. For the cells and foam data set, a score was only computed for meshes which intersected the xy-slice with known ground truth. Correspondences between segmentations and ground truth labels were determined a priori. If an intersecting mesh did not have a corresponding label, it was assigned a score of 0.

The segmentation of each data set with and without over-



(a) Sim. cells (b) Cells (c) Foam

Figure 6: Top row: 3D renderings of the data sets. Bottom row: xy-slices of the data sets.

Param.	Sim. cells w/o	Sim. cells w/	Cells w/o	Cells w/	Foam w/o	Foam w/
μ_{in}	105	105	100	100	-0.0002	-0.0002
μ_{out}	90	90	5	5	0.0022	0.0022
σ_{in}	4	4	20	20	0.02	0.02
σ_{out}	4	4	20	20	0.04	0.04
Δ	7	8	9	9	2	2
δ_{step}	0.5	0.5	0.5	0.5	0.5	0.5
n_{step}	90	90	50	50	30	30

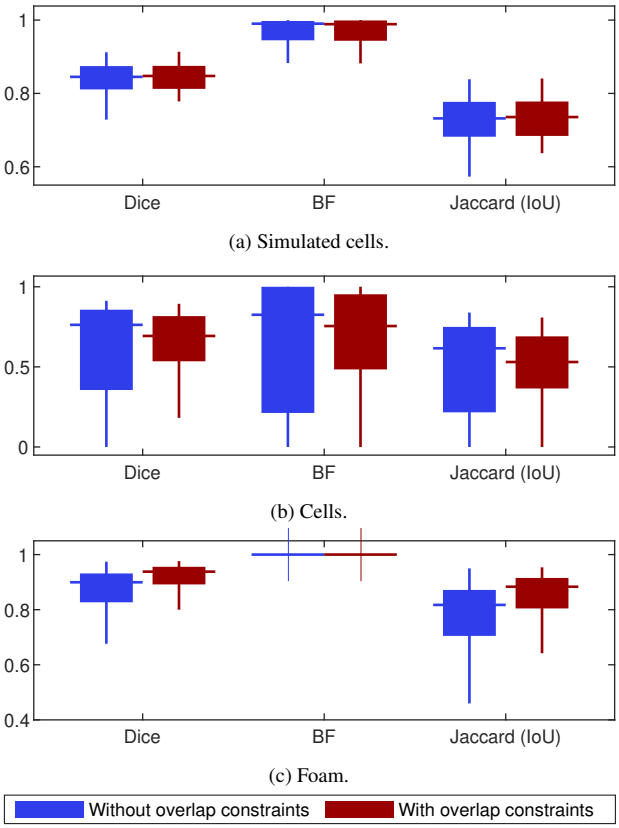
Table 2: Method parameters used for segmentation. See [10] for explanation of the μ and σ parameters.

		Sim. cells w/o	Sim. cells w/	Cells w/o	Cells w/	Foam w/o	Foam w/
Dice	Mean	0.83	**0.84**	0.61	**0.62**	0.81	**0.88**
	Max.	0.91	**0.91**	**0.91**	0.89	0.97	**0.98**
	Min.	0.51	**0.58**	0.00	0.00	0.00	0.00
BF	Mean	0.96	**0.97**	0.66	**0.66**	0.95	**0.98**
	Max.	1.00	1.00	1.00	1.00	1.00	1.00
	Min.	0.77	**0.88**	0.00	0.00	0.00	0.00
Jaccard	Mean	0.72	**0.73**	**0.50**	0.49	0.73	**0.81**
	Max.	0.84	**0.84**	**0.84**	0.81	0.95	**0.95**
	Min.	0.34	**0.41**	0.00	0.00	0.00	0.00

Table 3: Summary of statistics segmentation scores with and without overlap constraints. The best scores for each data set have been marked with bold (the marking is based on additional decimals).

lap constraints are shown in Figure 9. Figure 7 and Table 3 summarize the segmentation scores for each data set. For the segmentations with overlap constraints, we achieved a full assignment for all data sets. Adding overlap constraints generally results in an improved segmentation, especially for the foam data set. For the cells data set, there is a small decrease in the maximum score. This can partly be attributed to the fact that we only measure the segmentation quality in a single xy-slice. Thus, adding the overlap constraints may cause some local degradations of segmentation quality, even if the overall (3D) segmentation may be better. The reason this does not happen for the foam data set is that overlap is a much bigger problem here than for the cells data set. Therefore, fixing it results in a greater overall improvement.

As Figure 7 demonstrates, most of the segmentation improvements come from increasing the quality of the worst segmentations. This is a direct result of removing non-physical segmentation overlap, as shown in Figure 8. When objects are allowed to overlap the segmentation of one object can stray into a neighboring object which increases

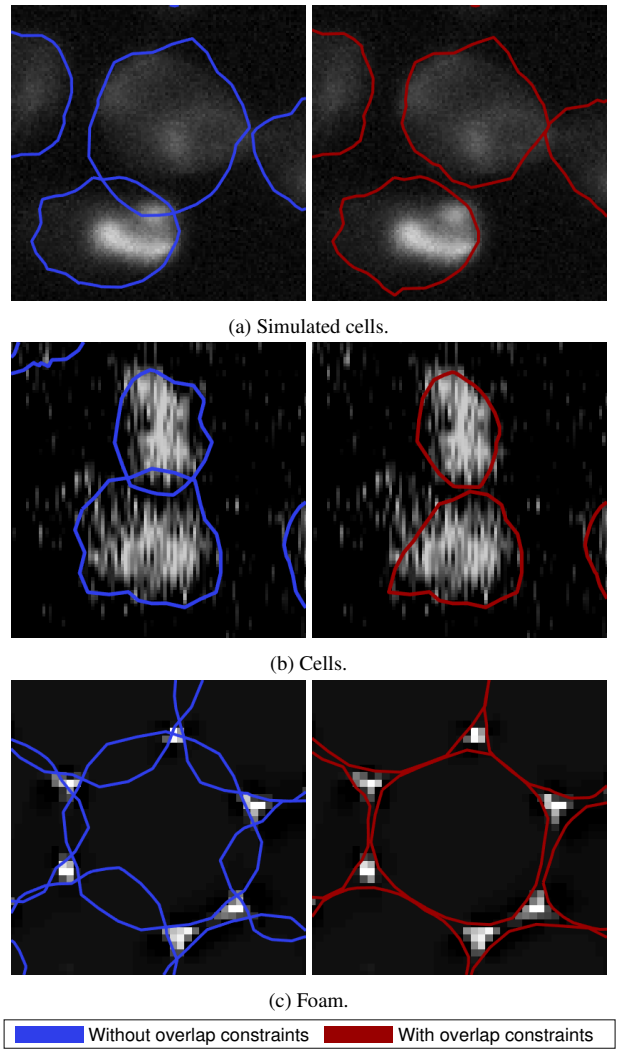


(a) Simulated cells.

(b) Cells.

(c) Foam.

Figure 7: Boxplots of segmentation scores with and without overlap constraints.



(a) Simulated cells.

(b) Cells.

(c) Foam.

Figure 8: Examples of non-physical segmentation overlaps which are corrected by our method. Figures show cross-sections of fitted surface meshes overlaid on xy-slices of the data.

the number of false positives. When this is prevented, the segmentations follow the true object contours more closely. Furthermore, the segmentations now resemble the physical reality more, since distinct object cannot overlap.

Finally, the method was found to be fairly robust to small parameter changes. However, changing Δ, δ_{step}, or δ_{step} could have significant effects on the run time, as they directly affect the size of the graph.

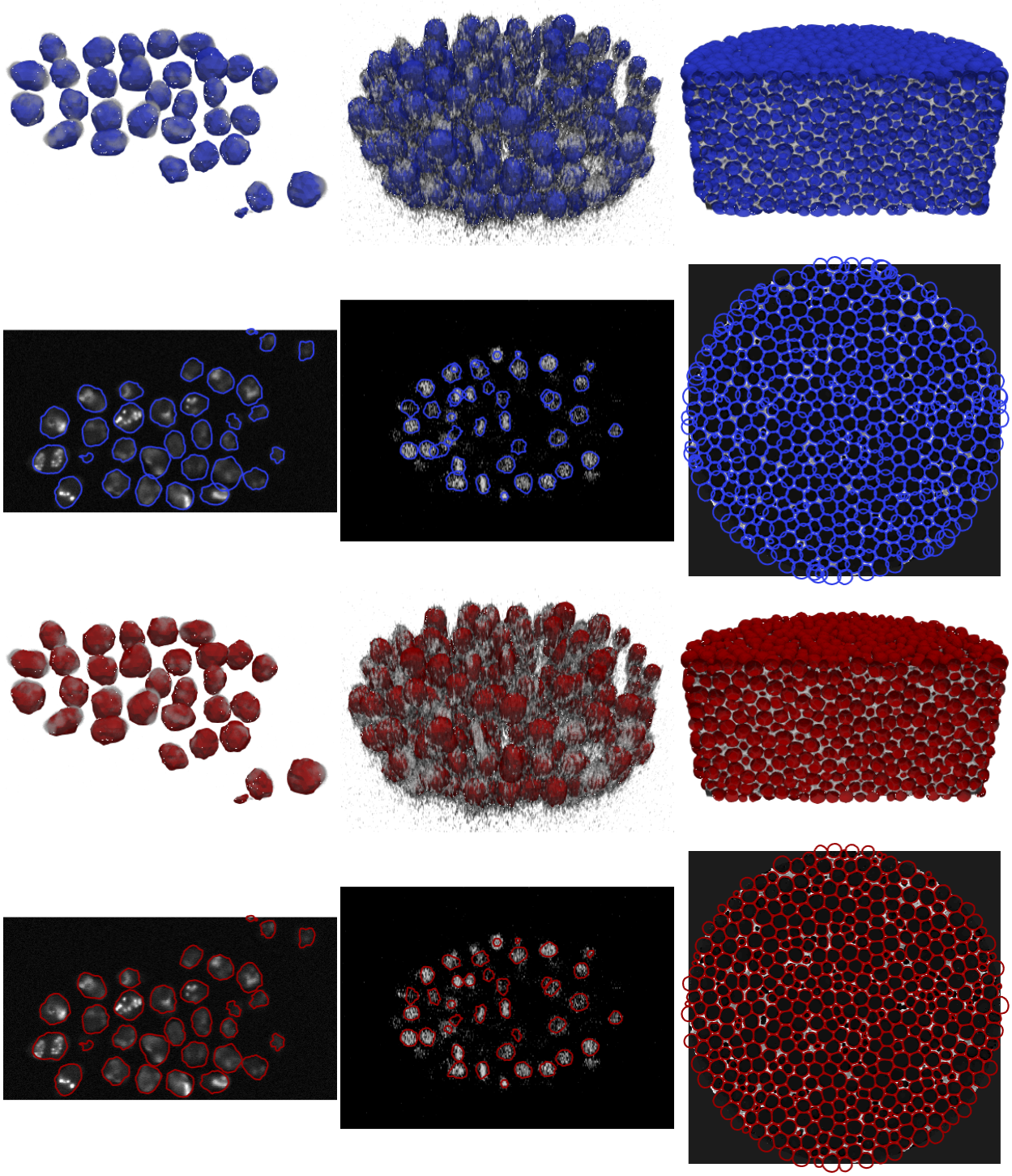
4. Conclusion and further work

In this paper, we have presented an extension for multi-object graph-based segmentation which allows us to enforce that segmented objects may not overlap. The extension worked by separating neighboring objects with planes whose position was determined dynamically. When applied to data sets with known ground truth segmentations, adding overlap constraints resulted in quantitative improvements. Furthermore, overlap prevention also qualitatively improved the segemtnations as they better complied with physical reality.

It is worth noting that our extension is not restricted to the basic version of the graph cut method used in this paper. Indeed, it can be applied to any method in the graph cut family, since it only adds additional constraints to the existing graph structures but does not otherwise modify them.

Furthermore, it is possible to generalize the extension to separate objects with other surfaces than planes, which would increase its usefulness. The current construction could instead use the level sets of any suitable function as the separating surfaces. However, if the separating surfaces have curvature, some amount of overlap will be possible due to the discrete nature of meshes. Developing the mathematical theory for this more general construction will be the subject of future work.

Acknowledgements: This work is partly supported by The Center for Quantification of Imaging Data from MAX IV (QIM) funded by The Capital Region of Denmark



(a) Sim. cells (b) Cells (c) Foam

Figure 9: Segmentation results overlaid on data. Blue surfaces and curves were fitted without overlap constraints, red ones with overlap constraints. Rows 1 and 3 show the fitted surface meshes overlaid on 3D rendering of data. Rows 2 and 4 show cross-sections of surface meshes overlaid on xy-slices of the data.

References

[1] Junjie Bai, Abhay Shah, and Xiaodong Wu. Optimal multi-object segmentation with novel gradient vector flow based shape priors. *Computerized Medical Imaging and Graphics*, 69:96–111, nov 2018. 1

[2] Stephen P. Boyd and Lieven. Vandenberghe. *Convex optimization*. Cambridge University Press, 2004. 1

[3] Yuri Boykov and Vladimir Kolmogorov. An Experimental Comparison of Min-Cut/Max-Flow Algorithms for Energy Minimization in Vision. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 26(9):1124–1137, 2004. 3

[4] Xinjian Chen and Lingjiao Pan. A Survey of Graph Cuts/Graph Search Based Medical Image Segmentation. *IEEE Reviews in Biomedical Engineering*, 11:112–124, 2018. 3

[5] Gabriela Csurka, Diane Larlus, Florent Perronnin, and France Meylan. What is a good evaluation measure for semantic segmentation?. In *British Machine Vision Conference*, volume 27, page 2013, 2013. 5

[6] Francesco De Carlo, Doğa Gürsoy, Daniel J Ching, K Joost Batenburg, Wolfgang Ludwig, Lucia Mancini, Federica Marone, Rajmund Mokso, Daniël M Pelt, Jan Sijbers, et al. TomoBank: a tomographic data repository for computational X-ray science. *Measurement Science and Technology*, 29(3):034004, 2018. 5

[7] Andrew Delong and Yuri Boykov. Globally Optimal Segmentation of Multi-Region Objects. In *International Conference on Computer Vision*, pages 285–292. IEEE, 2009. 1

[8] Jan Egger, Miriam H.A. Bauer, Daniela Kuhnt, Barbara Carl, Christoph Kappus, Bernd Freisleben, and Christopher Nimsky. Nugget-cut: A segmentation scheme for spherically- and elliptically-shaped 3D objects. In *Joint Pattern Recognition Symposium*, pages 373–382. Springer, 2010. 2

[9] Daniel Freedman and Petros Drineas. Energy minimization via graph cuts: Settling what is possible. In *Computer Vision and Pattern Recognition*, pages 939–946. IEEE, 2005. 3, 4

[10] Mona Haeker, Michael Abràmoff, Milan Sonka, Xiaodong Wu, and Randy Kardon. Incorporation of Regional Information in Optimal 3-D Graph Search with Application for Intraretinal Layer Segmentation of Optical Coherence Tomography Images. In *Information Processing in Medical Imaging*, pages 607–618. Springer, 2007. 3, 5, 6

[11] Fredrik Kahl and Petter Strandmark. Generalized roof duality for pseudo-boolean optimization. In *International Conference on Computer Vision*, pages 255–262. IEEE, 2011. 3

[12] Dagmar Kainmueller, Hans Lamecker, Stefan Zachow, and Hans Christian Hege. Coupling deformable models for multi-object segmentation. In *International Symposium on Biomedical Simulation*, pages 69–78. Springer, 2008. 1, 2

[13] Vladimir Kolmogorov and Carsten Rother. Minimizing non-submodular functions with graph cuts - A review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(7):1274–1279, 2007. 3, 4

[14] Vladimir Kolmogorov and Ramin Zabih. What Energy Functions can be Minimized via Graph Cuts? *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(2):147–159, 2004. 3, 4

[15] Kang Li, Xiaodong Wu, Danny Z. Chen, and Milan Sonka. Optimal surface segmentation in volumetric images - A graph-theoretic approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(1):119–134, 2006. 2, 3

[16] Rajmund Mokso, Christian M Schlepütz, Gerd Theidel, Heiner Billich, Elmar Schmid, Tine Celcer, Gordan Mikuljan, Leonardo Sala, Federica Marone, Nick Schlumpf, et al. GigaFRoST: the gigabit fast readout system for tomography. *Journal of synchrotron radiation*, 24(6):1250–1259, 2017. 5

[17] John Isaac Murray, Zhirong Bao, Thomas J Boyle, Max E Boeck, Barbara L Mericle, Thomas J Nicholas, Zhongying Zhao, Matthew J Sandel, and Robert H Waterston. Automated analysis of embryonic gene expression with cellular resolution in c. elegans. *Nature methods*, 5(8):703, 2008. 5

[18] C Raufaste, B Dollet, K Mader, S Santucci, and R Mokso. Three-dimensional foam flow resolved by fast X-ray tomographic microscopy. *Europhysics Letters*, 111(3):38004–38010, 2015. 5

[19] Carsten Rother, Vladimir Kolmogorov, Victor Lempitsky, and Martin Szummer. Optimizing Binary MRFs via Extended Roof Duality. In *Computer Vision and Pattern Recognition*, pages 1–8. IEEE, 2007. 3

[20] Qi Song, Xiaodong Wu, Yunlong Liu, Mark Smith, John Buatti, and Milan Sonka. Optimal graph search segmentation using arc-weighted graph for simultaneous surface detection of bladder and prostate. In *Medical Image Computing and Computer-Assisted Intervention*, pages 827–835. Springer, 2009. 1, 2

[21] David Svoboda, Michal Kozubek, and Stanislav Stejskal. Generation of digital phantoms of cell nuclei and simulation of image formation in 3D image cytometry. *Cytometry Part A*, 75A(6):494–509, 2009. 5

[22] David Svoboda and Vladimír Ulman. MitoGen: A framework for generating 3D synthetic time-lapse sequences of cell populations in fluorescence microscopy. *IEEE Transactions on Medical Imaging*, 36(1):310–321, 2016. 5

[23] Yao Wang and Reinhard Beichel. Graph-based segmentation of lymph nodes in CT data. In *International Symposium on Visual Computing*, pages 312–321. Springer, 2010. 2

[24] Xiaodong Wu and Danny Z. Chen. Optimal Net Surface Problems with Applications. In *International Colloquium on Automata, Languages, and Programming*, pages 1029–1042. Springer, 2002. 2, 3

[25] Yin Yin, Xiangmin Zhang, Rachel Williams, Xiaodong Wu, Donald D. Anderson, and Milan Sonka. LOGISMOS-layered optimal graph image segmentation of multiple objects and surfaces: Cartilage segmentation in the knee joint. *IEEE Transactions on Medical Imaging*, 29(12):2023–2037, dec 2010. 1, 2

3.4 Paper D: Finding Space-Time Boundaries with Deformable Hypersurfaces

Patrick M. Jensen, J. Andreas Bærentzen, Anders B. Dahl, Vedrana A. Dahl
To be submitted to Journal of Mathematical Imaging and Vision.

Finding Space-Time Boundaries with Deformable Hypersurfaces

Patrick M. Jensen[1*], J. Andreas Bærentzen[1], Anders B. Dahl[1] and Vedrana A. Dahl[1]

[1*]Department of Applied Mathematics and Computer Science, Technical University of Denmark, Richard Petersens Plads, Kgs. Lyngby, 2800, Denmark.

*Corresponding author(s). E-mail(s): patmjen@dtu.dk;
Contributing authors: janba@dtu.dk; abda@dtu.dk; vand@dtu.dk;

Abstract

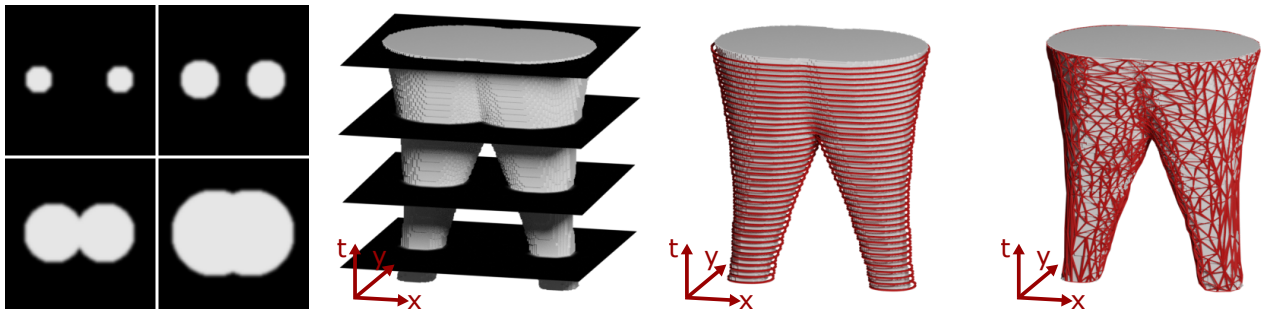
Dynamic 3D imaging is increasingly common as a way of studying evolving objects. Here, we address the problem of detecting and tracking simple 3D objects (like bubbles, or cells) that either merge or split in time. Solving this problem commonly involves detecting topological changes. We instead solve this problem in 4D (space+time) and we exploit one core observation: If the evolving objects only merge, or only split, they appear as a single simply connected component in 4D. We can therefore initiate a topologically simple 3D hypersurface (embedded in 4D) and deform it to fit the surface of all the objects at all times. This gives an extremely compact representation of the objects' continuous evolution over time. We test our method on artificial 4D images and compare it with other established segmentation methods. Furthermore, we apply our method to a 4D dynamic synchrotron X-ray computed tomography data set and use it to quantify evolving topology. We achieve segmentation performance comparable to existing methods with better resource use and improved robustness to segmentation errors.

Keywords: 4D Images, Deformable Models, Segmentation

1 Introduction

4D images refer to a series of 3D volumes acquired over time, for example via dynamic X-ray computed tomography (CT). Such images may be used to study the evolution of 3D objects and have applications in *e.g.* medicine [1] and materials science [2–4]. To use 4D images in quantitative studies, we need segmentation methods that can detect and track objects over time. Additionally, some systems contain objects which merge or split as part of their time evolution, and such events are often of interest on their own [2, 5]. Therefore, a segmentation method should handle topological changes and still obtain an accurate result.

Generally, existing methods for segmenting 4D images fall into two categories: mesh-based methods, and voxel-based methods. Mesh-based methods are mainly used for tracking non-interacting objects over time. In [6, 7], a sequential approach is used where the fit in one 3D volume is used to improve the fit in the next. Here, one needs to establish a correspondence between segmentations in consecutive 3D volumes in order to obtain object tracking. When objects merge or split, this can be highly nontrivial. Others [8–11] use a more global approach where a separate mesh is placed in every 3D volume and then all meshes are fitted simultaneously. Common for these methods is that the mesh connectivity is kept fixed and only vertex positions are changed. This provides a natural correspondence between meshes and makes it easy to track the development of an object over time. However, it does not allow for the object to split or merge.



(a) (b) (c) (d)

Fig. 1 The principle of the 4D segmentation approach illustrated on a lower-dimensional 3D (2D + time) example. (a) Time series of 2D images, where two disks grow and merge. (b) 2D images are stacked to form a 3D (2D + time) volume. Here the merging disks appear as a single simply connected component. (c) Segmentation given by sequentially detecting the disks in each 2D image. (d) Segmentation given by detecting the 2D boundary of the 3D connected component

Voxel-based approaches on the other hand can easily handle topology changes. Here, authors have used methods ranging from level sets [5, 12], Markov random fields [13, 14], to deep learning [6, 15]. However, while voxel-based approaches are more flexible regarding the evolution of the object, they do not (directly) give access to a mesh-based representation, which can make postprocessing more challenging. Furthermore, small errors in the segmentation can have large effects on the resulting topology as two objects may merge prematurely. And finally, due to advancements in imaging technology [16, 17], 4D images can now be acquired with a very high spatial and temporal resolution, i.e. a voxel size $\leq 5\mu$m at hundreds of 3D volumes per second. As a result, modern 4D images have sizes measured in the tens of gigabytes (GB) to several terabytes (TB). Segmenting 4D images with voxel-based methods therefore quickly becomes unwieldy unless the data is severely downsampled.

In order to handle the large size of 4D images, we will use a mesh-based method, where our goal is to detect the surface of the evolving objects. We detect the surface by initiating and iteratively deforming a mesh, similar to classic active contours [18] or deformable surfaces [7, 19]. Our approach differs from existing methods in that we operate in 4D. We can therefore segment a whole system of merging or splitting objects using only one (but relatively uncommon) mesh.

We now describe the main idea of our method. First, we utilize the fact that strictly merging 3D objects exist as a single connected component in 4D. The boundary of this component is a 3D hypersurface embedded in 4D [20]. Such a hypersurface may be represented as a tetrahedral mesh with vertices having four coordinates. Furthermore, we make an additional simplifying assumption; we assume that objects in the 4D image only undergo either splits or merges – but not both. This assumption guarantees that the resulting 4D connected component is *simply connected* (homeomorphic to a 4D ball). For this case, we know the topology of the hypersurface we want to detect. We can therefore initialize the mesh, and fit it to the data without needing to make any topological changes.

The main idea of our approach is also illustrated in Fig. 1 on a lower-dimensional 3D (2D + time) example. With existing methods, we would separately detect the boundary for each time (illustrated in Fig. 1c), and then attempt to connect these segmentations. Instead, considering all images at once, the boundary of merging objects is a surface and may be detected by fitting a mesh to the image data (illustrated in Fig. 1d).

Finally, to get a segmentation for a single time, either in the 4D or 3D case, we can compute a cross-section of the fitted mesh. This results in a mesh describing the boundary of the segmented object(s) at a given time.

While the assumption of only merging or only splitting slightly limits our method, many interesting systems exhibit this behavior. For example, biological cells divide but do not merge, and bubbles in a foam merge but do not split. Furthermore, for such systems, enforcing that objects are only allowed split or merge can indeed be a benefit, as it enforces our prior knowledge of the system.

Our approach has several advantages over alternative methods. First, it provides a very compact description of the evolution of an object. Partly because storing the boundary inherently requires significantly less space. But also because a tetrahedron in 4D can span multiple time values, which exploits the coherence between two time-adjacent 3D volumes. The second

advantage is that it removes the need to explicitly handle merges or splits. As we only have a single simply connected component in 4D, the segmentation can be done with a single mesh. Thirdly, as we enforce the prior knowledge that objects only split or merge, this makes our method more robust to small segmentation errors, as we will demonstrate. Finally, it is worth noting that cross-sections of the fitted tetrahedral can be computed for any time value. Thus, our method provides a continuous characterization of the object evolution, which makes it easy to interpolate object evolution between two 3D volumes.

To summarize; in this paper, we extend the classical methods of deformable surfaces to 4D, to fit tetrahedral meshes to boundaries in 4D images. We investigate the properties of our method by comparing it to mesh-based methods based on consecutive 3D segmentations and voxel-based methods in 4D using a synthetic 4D image. We exemplify the scalability of our method by segmenting a 4D dynamic CT image. Finally, we demonstrate how our method can be used to quantify the evolving topology of merging/splitting objects.

2 Method

Deformable models are widely used in 2D and 3D segmentation in the form of curves and surfaces. In 4D segmentation, hypersurfaces that deform in space-time are rarely encountered, and existing approaches treat the time dimension differently than the space dimensions. In this section, we propose the methodology needed to generalize parametric deformable surfaces to 4D, providing a combined treatment of the deformation in space and time. We begin by introducing the continuous formulation of 4D deformable models and then detail how we discretize the problem. In the subsequent sections, we then detail each component of our 4D deformable model approach.

2.1 Continuous 4D Deformable Model

Given a 4D image, $I : \Omega \subset \mathbb{R}^4 \to \mathbb{R}$, we aim to find a partition of the domain Ω into two disjoint regions, Ω_{in} and Ω_{out}, separated by a boundary Γ. The region Ω_{in} corresponds to the inside of the 4D simply connected component we wish to segment and Ω_{out} to the outside. Since the regions Ω_{in} and Ω_{out} are uniquely determined by Γ we formulate our problem as a search for an optimal boundary Γ^* that minimizes the following energy functional

$$\Gamma^* = \arg\min_{\Gamma} E_{\text{ext}}(\Gamma) + E_{\text{int}}(\Gamma)\,. \tag{1}$$

Here, E_{ext} is an external energy term that ensures that Ω_{in} and Ω_{out} correspond to the image regions we wish to segment and E_{int} is an internal energy that acts as a regularizer by encouraging the boundary Γ to remain smooth. The optimal boundary Γ^* will satisfy the associated Euler-Lagrange equation [18, 19]

$$\nabla E_{\text{ext}}(\Gamma^*) + \nabla E_{\text{int}}(\Gamma^*) = 0\,. \tag{2}$$

For E_{ext}, we use the popular Chan-Vese energy [21], which assumes the image can be modeled as a piecewise constant function with value c_{in} in Ω_{in} and c_{out} in Ω_{out}. It is given as

$$\begin{aligned} E_{\text{ext}}(\Gamma) = &\int_{\Omega_{\text{in}}} (I - c_{\text{in}})^2 \,\mathrm{d}\Omega + \\ &\int_{\Omega_{\text{out}}} (I - c_{\text{out}})^2 \,\mathrm{d}\Omega\,. \end{aligned} \tag{3}$$

Generally, c_{in} and c_{out} may be unknown values which are also optimized over. However, as that adds a significant computational cost, we assume in this paper that c_{in} and c_{out} are known a priori.

For E_{int}, we use the Dirichlet energy

$$E_{\text{int}}(\Gamma) = \gamma \int_{\Gamma} \|\nabla\Gamma(\mathbf{x})\|_2^2 \,\mathrm{d}\mathbf{x}\,, \tag{4}$$

where $\gamma \geq 0$ is a fixed parameter that controls the degree of regularization.

2.2 Discrete 4D Deformable Model

To find the optimal boundary, we use a discretized version of Γ. Since Γ is the boundary between two 4D regions, it is a 3D hypersurface [20] embedded in 4D. We represent Γ with a tetrahedral mesh $\mathcal{T}$ with vertices $\mathbf{X} \in \mathbb{R}^{V\times 4}$ where vertex positions $\mathbf{x}_i$ are given by four coordinates as shown in Fig. 2.

Given an initial mesh $\mathcal{T}^{(0)}$, a solution to the discretized version of (2) can be found by iteratively solving [18, 19]

$$\begin{aligned} \nabla E_{\text{ext}}(\mathcal{T}^{(i-1)}) + \nabla E_{\text{int}}(\mathcal{T}^{(i-1)}) = \\ -\frac{1}{\tau}(\mathbf{X}^{(i)} - \mathbf{X}^{(i-1)})\,, \end{aligned} \tag{5}$$

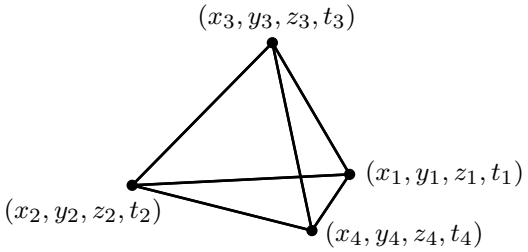


Fig. 2 Single element of the tetrahedral mesh. Each vertex position, $\mathbf{x}_i$, has four coordinates: x_i, y_i, z_i, and t_i

where τ is a step size. The gradient of the membrane energy $\nabla E_{\text{int}}(\Gamma)$ is given by the Laplacian $\Delta\Gamma$ [22]. This gives the update equation

$$(\mathbf{I} + \lambda\mathbf{L})\,\mathbf{X}^{(i)} = \mathbf{X}^{(i-1)} - \tau\nabla E_{\text{ext}}(\mathcal{T}^{(i-1)})\,, \quad (6)$$

where $\mathbf{L}$ is a matrix approximating the Laplacian of Γ, and $\lambda = \tau\gamma$. We describe our initialization strategy in Section 2.3. As the mesh deforms from its initial configuration the tetrahedra will stretch, meaning the mesh will not be able to fit the image to sufficient detail. Therefore, we adaptively subdivide the mesh every n_{sub} iteration to prevent the mesh from becoming too coarse. We detail our subdivision strategy in Section 2.4.

For the external energy gradient in (6), it can be shown that the vertex displacements corresponding to $\nabla E_{\text{ext}}(\mathcal{T})$ are given by

$$\frac{\partial E_{\text{ext}}}{\partial \mathbf{x}_i} = 2(c_{\text{in}} - c_{\text{out}})(I(\mathbf{x}_i) - (c_{\text{in}} + c_{\text{out}})/2)\mathbf{n}_i\,, \quad (7)$$

where $\mathbf{n}_i$ is the normal for vertex i [23]. As we adaptively subdivide the mesh during deformation, we want our normals to be robust to this. Therefore, we have developed an extension of angle-weighted normals to 4D, which we describe in Section 2.5.

For the Laplacian matrix, we use the scale-dependant Laplacian [24], to be robust to irregularly sized tetrahedra [22]. This computes the Laplacian at the i'th vertex as

$$L(\mathbf{x}_i) = \frac{1}{E}\sum_{j\in N(i)} \frac{\mathbf{x}_j - \mathbf{x}_i}{e_{ij}}, \quad E = \sum_{j\in N(i)} e_{ij}\,, \quad (8)$$

where $e_{ij} = \|\mathbf{x}_j - \mathbf{x}_i\|$ and $N(i)$ are the neighbor vertices of vertex i. Since $\mathbf{L}$ is sparse, we can solve the linear system in (6) efficiently using conjugate gradient iteration. However, constructing $\mathbf{L}$ is relatively expensive, and we therefore only update it when we subdivide the mesh. While not strictly correct, we did not observe drawbacks with this strategy.

Finally, to extract the segmentation for a 3D volume at a given time, we compute a cross-section of the fitted 4D mesh using the method detailed in Section 2.6.

2.3 Mesh Initialization

We assume that the object(s) we are segmenting is present in every 3D volume of the 4D image. Furthermore, we assume objects only merge (similar to the example Fig. 1), since splitting can be viewed as merging in reverse. Thus, if the initial mesh is placed in the last 3D volume of the 4D image, it only needs to propagate backward in time. This makes a hyperdisk (*i.e.*, a solid 3D ball) a good candidate for an initial mesh. The boundary of a hyperdisk forms a 2D surface, which we can represent with a triangle mesh.

This leads to the following initialization approach. First, fit a triangle mesh to the 2D object boundary in the last 3D volume. Since we only consider a single 3D volume here, we can ignore the time coordinate and treat it as a traditional 3D segmentation task. As a result, we can use an existing 3D method such as deformable surfaces [7], graph cut approaches [25, 26], or an isosurface [27] from a voxel segmentation to find the triangle mesh. We now let this triangle mesh be the boundary of the hyperdisk, and then tetrahedralize it using the TetGen tool by Si [28]. Finally, we add the time coordinate to the tetrahedron vertices, which results in the initial mesh. An illustration of the initialization approach for a lower dimensional (2D + time) example is shown in Fig. 3.

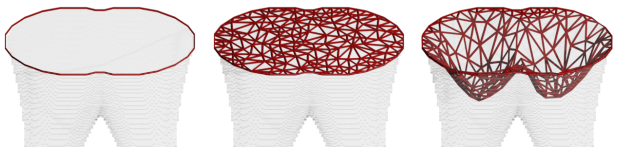


Fig. 3 The principle of initialization process illustrated on a 3D (2D + time) example. (a) First, the boundary of the object in a single image (volume in 4D) is located. (b) The interior of the boundary is triangulated (tetrahedralized in 4D), which gives the initial mesh. (c) The initial mesh is iteratively deformed to fit to the boundary of the connected component

To avoid the mesh leaving the last 3D volume, we constrain the time coordinate of the tetrahedron vertices that correspond to the original triangle vertices to remain unchanged.

2.4 Adaptive 4D Mesh Subdivision

A considerable amount of research has been conducted on creating and subdividing tetrahedral meshes in 3D, and we want to leverage this. We will use an idea similar to that of coordinate charts from differential geometry [29], which are maps from manifolds to Euclidean space. In our case, we know the initial mesh is contained in a single 3D volume. After the mesh has been deformed in 4D, the initial mesh defines a natural mapping between 3D Euclidean space and the 4D mesh. For a point placed anywhere within the initial mesh, we can then use barycentric coordinates to find its corresponding position in 4D. See Fig. 4 for an illustration.

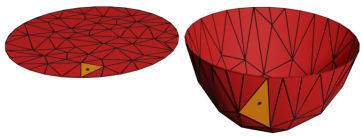


Fig. 4 Illustration of the mapping from the coordinate mesh (left) to the deformed mesh (right). A point inserted in the highlighted face in the coordinate mesh can be mapped to its corresponding position in the deformed mesh using barycentric coordinates.

Specifically, we keep a copy of the initial mesh, $\mathcal{T}^{(0)}$, during deformation — henceforth referred to as the coordinate mesh, $\mathcal{T}^C$. To perform subdivision, we find all tetrahedra whose volume has grown larger than a threshold, s, and insert a new point at the barycenter of each tetrahedron. The volume of a tetrahedra with vertex positions $\mathbf{x}_1$, $\mathbf{x}_2$, $\mathbf{x}_3$, and $\mathbf{x}_4$ is given by [30, 31]

$$\mathrm{vol}(\mathbf{T}) = \frac{1}{6}\sqrt{G(\mathbf{x}_2 - \mathbf{x}_1, \mathbf{x}_3 - \mathbf{x}_1, \mathbf{x}_4 - \mathbf{x}_1)}\,, \quad (9)$$

where G is the Gram determinant. Then, the original vertices of $\mathcal{T}^C$, along with the new points, are re-tetrahedralized with TetGen [28], which gives a new coordinate mesh. This is then re-mapped to 4D, after which the old coordinate and 4D mesh are thrown away.

While performing a full re-tetrahedralization is more expensive than updating the mesh, it has the benefits of resulting in a higher quality mesh, and is simple to implement with tools designed for 3D meshes.

2.5 Angle Weighted Normals in 4D

Analogous to surface meshes in 3D, vertex normals are defined as a weighted average of the adjacent tetrahedral face normals. These are in turn defined using

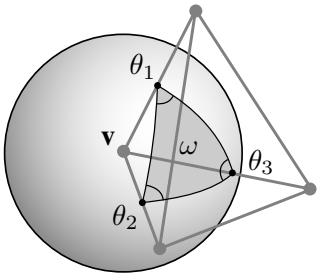


Fig. 5 The solid angle ω at vertex $\mathbf{v}$ is the area of the spherical triangle given by the intersection of the tetrahedron and a unit sphere centered at $\mathbf{v}$. θ_1, θ_2, and θ_3 denote the dihedral angles along the tetrahedron edges connected to v

a 4D analog of the cross product [32]; given 4D vectors $\mathbf{u}$, $\mathbf{v}$, and $\mathbf{w}$, we can find an orthogonal vector, $\mathbf{n} = (n_x, n_y, n_z, n_t)^{\mathrm{T}}$, as

$$\begin{aligned} n_x &= \begin{vmatrix} u_y & u_z & u_t \\ v_y & v_z & v_t \\ w_y & w_z & w_t \end{vmatrix}, \quad n_y = \begin{vmatrix} u_x & u_z & u_t \\ v_x & v_z & v_t \\ w_x & w_z & w_t \end{vmatrix}, \\ n_z &= \begin{vmatrix} u_x & u_y & u_t \\ v_x & v_y & v_t \\ w_x & w_y & w_t \end{vmatrix}, \quad n_t = \begin{vmatrix} u_x & u_y & u_z \\ v_x & v_y & v_z \\ w_x & w_y & w_z \end{vmatrix}, \end{aligned} \quad (10)$$

where $|\cdot|$ is the determinant. For a tetrahedron with vertex positions $\mathbf{x}_1$, $\mathbf{x}_2$, $\mathbf{x}_3$, and $\mathbf{x}_4$, we form the vectors as $\mathbf{u} = \mathbf{x}_2 - \mathbf{x}_1$, $\mathbf{v} = \mathbf{x}_3 - \mathbf{x}_1$, and $\mathbf{w} = \mathbf{x}_4 - \mathbf{x}_1$.

Previous works have used simple averaging or weighted the normal of each tetrahedron by its volume [33]. However, as for triangle meshes [34], these weighting schemes are sensitive to re-tessellation of the mesh. In this work, we extend angle-weighted normals to four dimensions by weighting the contribution of each tetrahedron by the solid angle spanned by the tetrahedron at $\mathbf{v}$. Assume, without loss of generality, that the mesh is scaled such that all edges are longer than 1. The solid angle, ω, is then given by the area of the spherical triangle formed by the intersection of the tetrahedron and a unit sphere, as illustrated in Fig. 5.

The area of the spherical triangle ω is given by

$$\omega = \theta_1 + \theta_2 + \theta_3 - \pi\,, \quad (11)$$

where θ_1, θ_2, and θ_3 are the dihedral angles of the tetrahedron edges connected to $\mathbf{v}$ [35]. The normal at vertex i is then given by

$$\mathbf{n}_i = \frac{\sum_{T\in\mathcal{T}_i} \omega_T \mathbf{n}_T}{\left\|\sum_{T\in\mathcal{T}_i} \omega_T \mathbf{n}_T\right\|}\,, \quad (12)$$

where $\mathcal{T}_i$ are the incident tetrahedra for vertex i with normals $\mathbf{n}_T$ and incident solid angles ω_T. The denominator ensures that the resulting normal is unit-length. Similar to angle-weighted normals for triangle meshes, this weighting scheme is invariant to re-tessellation of the mesh.

As a proof, consider re-tesselating a tetrahedron meaning splitting it into multiple sub-tetrahedra where the original vertices remain in place and all new vertices are convex combinations of the original vertices. In this case, the face normals of the new tetrahedra will be equal to the original face normal. Furthermore, the solid angles of the sub-tetrahedra incident with $\mathbf{v}$ will sum to the original solid angle ω. Therefore, the contribution of the incident sub-tetrahedra to the vertex normal of $\mathbf{v}$ will be equal to the contribution of the original tetrahedron.

2.6 Cross-sections of Tetrahedral Meshes in 4D

Computing a cross-section is equivalent to finding the intersection with a hyperplane or extracting an isosurface. We use a simplified version of the marching tetrahedron method [36], which has been used previously for visualizing 4D tetrahedral meshes [32].

We assume without loss of generality that we compute the intersection with the xyz-hyperplane at $t = 0$. If a tetrahedron intersects a hyperplane, the intersection will always be one of the eight cases illustrated in Fig. 6. The last three cases are problematic as the intersections are not surface elements. To avoid these, we perturb the time coordinate of all vertices which lie on the hyperplane by a small $\varepsilon \ll 1$, which guarantees we will only encounter the first two cases.

3 Results

To assess the performance of our method we perform a series of numerical experiments, where we segment computer-generated 4D images with known ground truth. This allows a quantitative assessment and comparison with other methods. Next, we apply the method to a 4D dataset from dynamic X-ray CT. Here, a ground truth segmentation is not known, so we can only do a qualitative evaluation. Finally, we demonstrate how our method allows for tracking the evolution of splitting/merging objects through time by computing the Reeb graph [37, 38] of the fitted tetrahedral mesh.

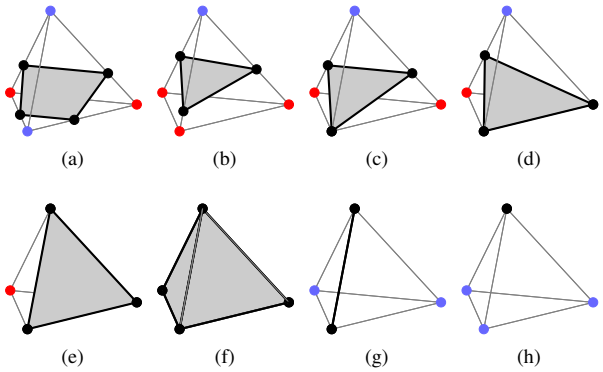
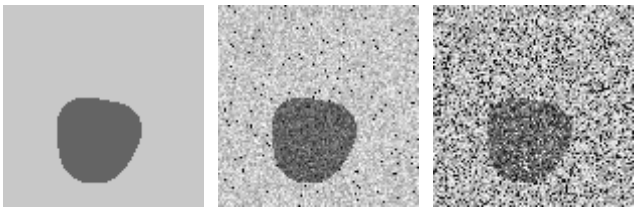


Fig. 6 All possible ways a tetrahedron in 4D may intersect a hyperplane. Black vertices are on the hyperplane ($t = 0$), red vertices are 'below' ($t < 0$), and blue vertices are 'above' ($t > 0$). The intersection can be (a) a quadrilateral (which may be split into two triangles), (b)-(e) a triangle, (f) the entire tetrahedron, (g) a line, or (h) a point

3.1 Numerical experiments

We created an artificial 4D image containing five organic-looking blobs, shown in the top row of Fig. 8. As time progresses, the blobs move toward each other until they collide and merge to a new blob whose volume grows to equal the sum of the previous two blobs. Furthermore, as the blobs move, they rotate and deform to make the data more challenging to segment.

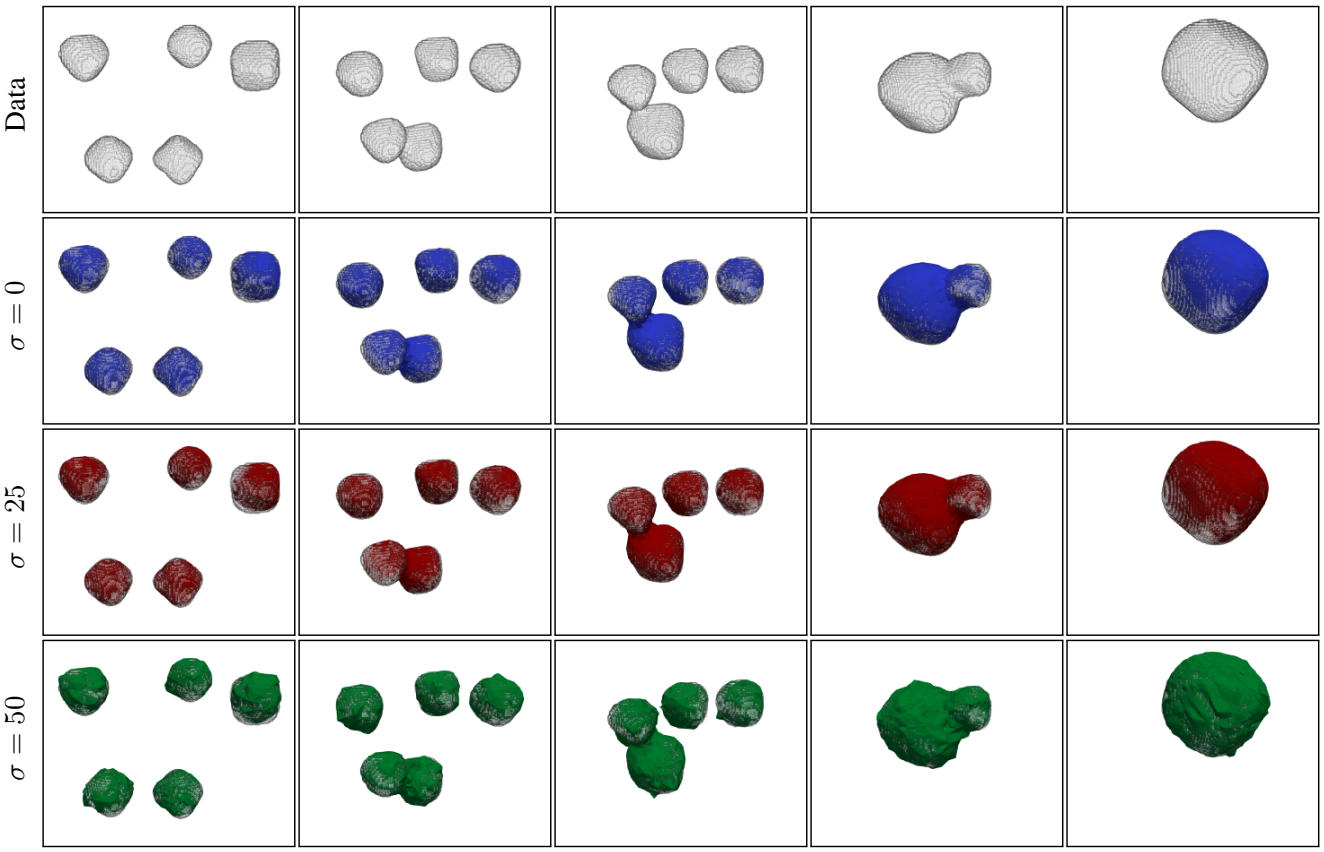


(a) $\sigma = 0$. (b) $\sigma = 25$. (c) $\sigma = 50$.

Fig. 7 2D cross-sections of the last 3D volume in the artificial 4D image for different values of the noise level σ

The 4D image consists of 200 binary label volumes of size $100 \times 100 \times 100$ voxels. Before segmenting, the intensity of the 4D image is transformed so the blobs have an intensity of 100 and the background an intensity of 200. After that, we add zero-mean Gaussian noise with standard deviations of 25 and 50 to create two additional 4D images. Fig. 7 shows a cross-section of the last 3D volume in each of the 4D images.

We segment the artificial images with the proposed method and compare with three other segmentation methods. These methods were chosen because they are well established and form the basis of many segmentation approaches [39]. The first two are sequential 3D Markov random fields and 4D Markov random fields



(a) $t = 1$. (b) $t = 70$. (c) $t = 100$. (d) $t = 145$. (e) $t = 200$.

Fig. 8 Segmentation results for the artificial 4D image. The rows show 3D renderings of (top to bottom): the data, segmentation for $\sigma = 0$, segmentation for $\sigma = 25$, and segmentation for $\sigma = 50$. The segmentations have been overlaid on the data

[40]. The segmentation is computed with the popular algorithm by Boykov and Kolmogorov (BK) [41]. For the sequential 3D version, we compute a separate segmentation for each 3D volume consecutively. Neighbor edges are added from a voxel node to its 6-connected spatial neighbors. For the 4D version, we compute a single segmentation for the entire 4D image and add edges from a voxel node to its 8-connected spatio-temoporal neighbors.

The final method is a graph cut (GC) method for surface detection in 3D volumes [11, 42]. Here, an initial mesh is placed at a user-defined location and deformed to fit the image contours [25, 26, 42] via a graph cut – we again use the BK algorithm and the region-based cost function from [43]. The initial mesh was chosen to be an 8-frequency subdivided icosahedron approximating a sphere. We process each volume sequentially and place a mesh at the centroid of each connected component of the ground truth label volume. Note, that this represents a best-case scenario for this method, as it requires either a good pre-segmentation of the data or manual annotation at every time step. Furthermore, the sequential nature of this method means that we do not have any correspondence between segmentations in different 3D volumes as we do with our method. The correspondences would have to be established afterward which is non-trivial. We apply the graph cut method for two values of the max. difference parameter, Δ, which controls how much the final shape is allowed to deviate from the initial shape (a sphere). A large value gives the method more freedom to fit elongated shapes, while a smaller value makes it more robust to noise.

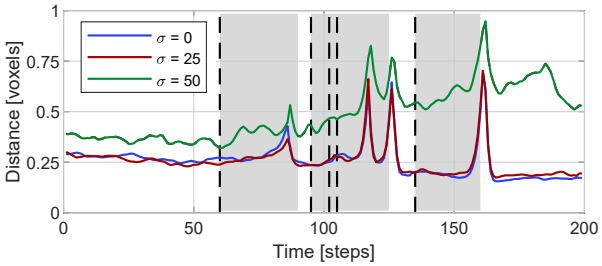
For our method, we scale each spatial axis to $[-1, 1]$ and the time axis to $[-2, 2]$. We use the ground truth segmentation of the last volume to initialize as in Section 2.3. Furthermore, the max. tet. volume, s, starts at $4\times$ the final value to perform a rough initial fit and is then set to the final value for the last 10 iterations to refine the fit.

The parameters for all methods are shown in Table 1 and all experiments were performed with an AMD Ryzen 7 3700X processor. For the noiseless image, we

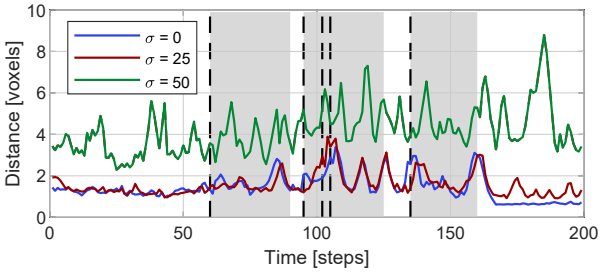
Table 1 Parameters used to segment the artificial 4D images.

Method	Parameter	$\sigma = 0$	$\sigma = 25$	$\sigma = 50$
Ours	Means, c_{in}, c_{out}	100, 200	100, 200	100, 200
	#iter., N	50	50	50
	Step size, τ	0.2	0.2	0.3
	Smoothing, λ	0.004	0.004	0.004
	Max. volume, s	16	16	16
	n_{sub}	10	10	10
MRF 3D/4D	Source term	-	$(I(\mathbf{x})-200)^2$	$(I(\mathbf{x})-200)^2$
	Sink term	-	$(I(\mathbf{x})-100)^2$	$(I(\mathbf{x})-100)^2$
	Neighbor term	-	10^4	10^4
Graph cut (GC/Δ)	Means, c_{in}, c_{out}	-	100 , 200	100 , 200
	Std. dev., σ	-	100	100
	#samples	-	150	150
	Sample step	-	0.5	0.5
	Max. diff., Δ	-	2 and 8	2 and 8

only apply our method. The segmentation results for our method are shown in Fig. 8. Visually, the segmentations correspond well with the data, with some noise observable for $\sigma = 50$.



(a) Mean distance.



(b) Max. distance.

Fig. 9 Plots of mean and max. distance between cross-sections of the fitted tetrahedral meshes and ground truth segmentation boundary. The gray regions signify times when merges are ongoing and the vertical dotted lines mark the beginning of a new merge

To provide a quantitative measure of the quality of the segmentation, a ground truth segmentation boundary – stored as a surface mesh – was created for each 3D volume. These were then compared with cross-sections of the tetrahedral mesh at the corresponding times. The comparison was done with the mean and

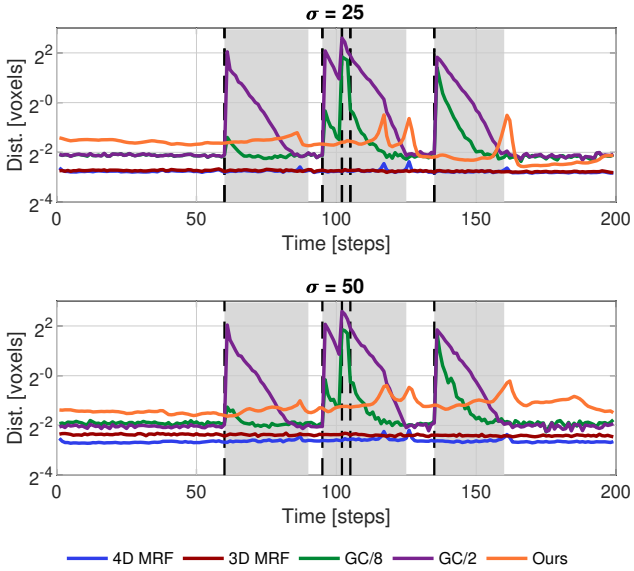


Fig. 10 Plots of the mean distance between the computed segmentation boundary for each method and ground truth segmentation boundary. The gray regions signify times when merges are ongoing and the vertical dotted lines mark the beginning of a new merge

maximum distance between the surface meshes as computed with the MESH tool by Aspert *et al.* [44]. The results are shown in Fig. 9.

The plots support what is seen in Fig. 8 as the mean distances remain small, i.e. below one voxel. There is a spike in the error at the end of the merges, since there is a slight discontinuity in the data when blobs merge. Thus, the temporal part of the chosen regularization introduces a larger error at these times. The same behavior occurs for the maximum distance, although the errors are larger since a single spurious vertex can significantly affect the value.

For the other methods, we also use the distance to the ground truth segmentation boundary. For the GC methods, this can be done directly. For the MRF methods, we extract an isosurface from each 3D label volume. The results are shown in Fig. 10. Additionally, we also compare the resource use of the methods. We measured runtime, peak memory use, and how much memory was needed to store the final segmentations. Table 2 shows the results along with the theoretical scaling of each measured value w.r.t. the image size. Our method performs as well as or better than the compared approaches while resulting in a more compact representation of the final segmentation.

3.2 Application to Metal Foam

We now apply the method to a 4D synchrotron X-ray CT image. The dataset is a time series of foaming

Table 2 Resource use for the segmentation approaches. The memory uses do not include the size of the image data (200 MB). The scaling w.r.t. the image size of each value is also shown. S represents the size of each 3D volume, and T is the number of 3D volumes in the 4D image

Noise	Method	Time		Memory use (peak)		Memory use (seg.)	
		Meas.	Scaling	Meas.	Scaling	Meas.	Scaling
$\sigma = 25$	3D MRF	45 sec.	S, T	408.3 MB	S	200.0 MB	S, T
	4D MRF	205 sec.	S, T	57,500.0 MB	S, T	200.0 MB	S, T
	GC/2	37 sec.	T	62.6 MB	1	9.2 MB	T
	GC/8	39 sec.	T	50.3 MB	1	9.2 MB	T
	Ours	36 sec.	1	40.5 MB	1	4.4 MB	1
$\sigma = 50$	3D MRF	45 sec.	S, T	421.5 MB	S	200.0 MB	S, T
	4D MRF	232 sec.	S, T	57,500.0 MB	S, T	200.0 MB	S, T
	GC/2	38 sec.	T	40.1 MB	1	9.2 MB	T
	GC/8	40 sec.	T	48.3 MB	1	9.2 MB	T
	Ours	38 sec.	1	41.0 MB	1	5.3 MB	1

Table 3 Parameters used to segment the 4D synchrotron X-ray CT data. Please see Table 1 for more descriptive parameter names

c_{in}	c_{out}	N	τ	λ	s	n_{sub}
25	70	342	0.03	0.0003	$5.2 \cdot 10^{-6}$	10

metal, where blowing agent powders have been mixed into a block of aluminum. As the aluminum sample is heated to its melting point the powders release gas, which causes bubbles to form. Over time, these bubbles expand and merge to form even larger bubbles. More details can be found in [2]. The data consists of 55 volumes of size $888 \times 888 \times 600$ and uses 26 GB of memory. It is visualized in the top row of Fig. 11.

We initialize the mesh at the large bubble in the bottom of the last volume (see Fig. 11(d)). We again scale the spatial axes to $[-1, 1]$ and the temporal to $[-0.25, 0.25]$. The goal of the segmentation is to detect which bubbles merged together to form the final bubble. The parameters are shown in Table 3 and the segmentation itself took 5 minutes using an Intel Xeon Gold 6142 processor.

The result of the segmentation is shown in the bottom row of Fig. 11. The method has detected the merging of different bubbles. Furthermore, the 4D cross-sections provide a good match with the data, even though the quality of the segmentation degrades somewhat the further we are from the initial mesh. Also, there are some minor errors in the last volumes where the tetrahedral mesh has moved into an adjacent bubble.

3.3 Tracking Evolving Topology

We now demonstrate how our method allows the quantification of splitting/merging by computing the Reeb graph [37, 38] of the fitted tetrahedral mesh. Reeb graphs are used to describe the evolution of level sets of a function defined on a manifold — in our case the time coordinate of our fitted tetrahedral mesh. As illustrated in Fig. 12, nodes are placed where topological changes occur such as local extrema and saddle points. If the level sets between two points form a connected component, the points are joined by an edge. As a result, the Reeb graph provides a compact description of the evolving topology of an object and also encodes when and where topological changes occur.

We use the Topology Toolkit [45, 46] to compute the Reeb graph for the (noiseless) segmentation of our 4D test image and the segmentation of the metal foam image. Prior to computation, we smooth the tetrahedral mesh with Laplacian smoothing and afterwards we prune edges of the Reeb graph which are shorter than 10% of the mean edge length.

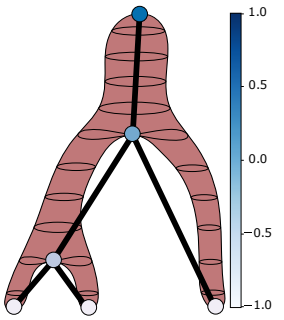
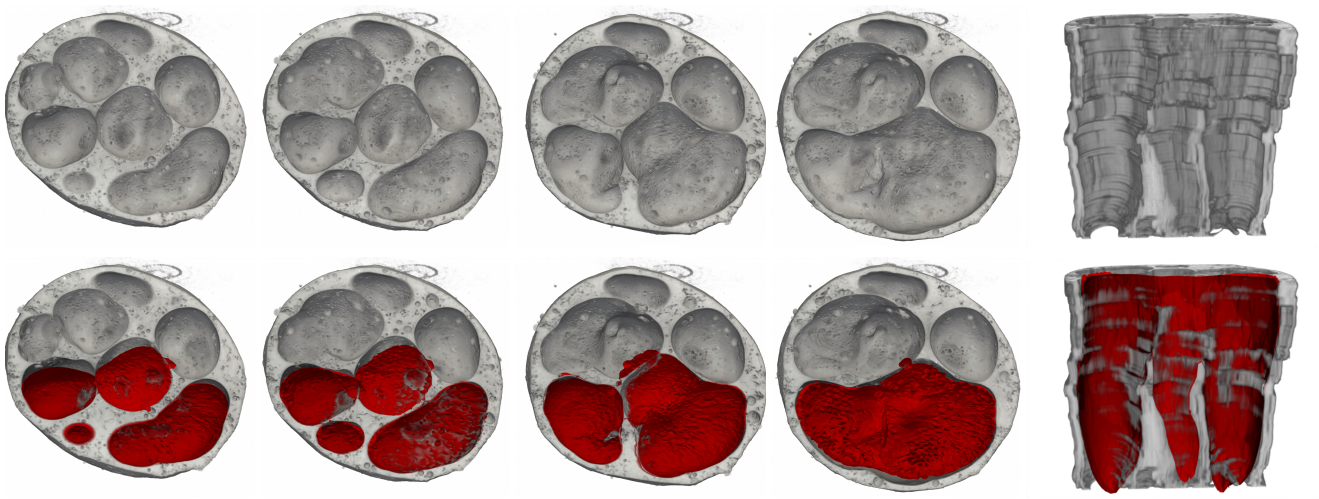


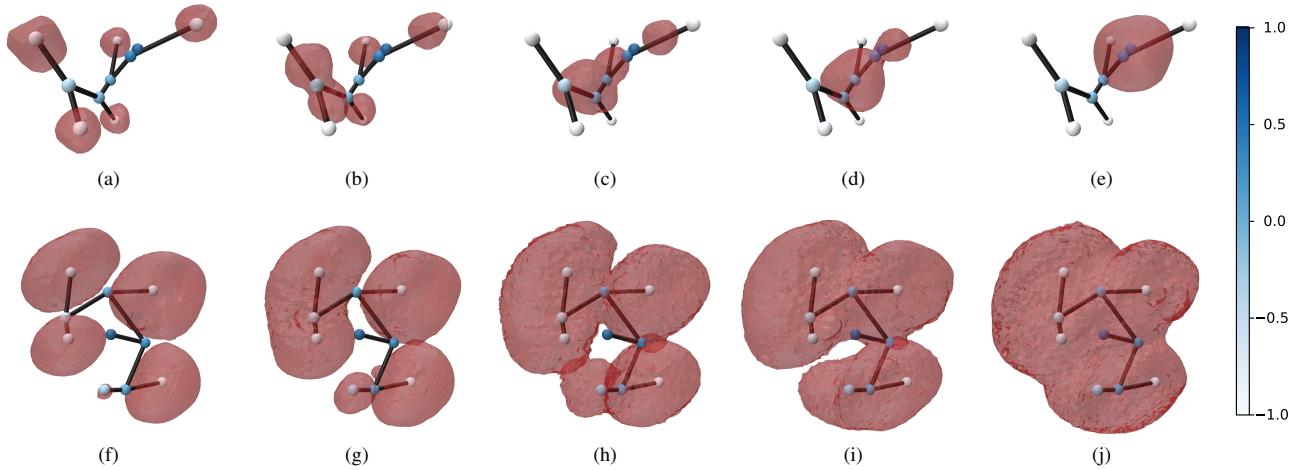
Fig. 12 Illustration of the Reeb graph for a simple 2D+time example. The level sets are shown as black rings and the Reeb graph has nodes where the level sets change topology. The nodes are colored according to their time coordinate

The results are shown in Fig. 13. For both segmentations, the Reeb graph provides a good description of where and how the objects merged together. Note especially how, even though a level set might have some slight self-intersections, e.g., in Fig. 13(h), objects are not counted as having merged since the Reeb graph considers the full spatio-temporal structure.



(a) xyz-volume at $t = 25$ (b) xyz-volume at $t = 35$ (c) xyz-volume at $t = 45$ (d) xyz-volume at $t = 55$ (e) xyt-volume at $x = 232$

Fig. 11 Segmentation results for the 4D synchrotron X-ray CT data. The top row shows a 3D rendering of the 4D data at different times. The bottom row shows the segmentations overlaid on the data. Note that the last column shows an xyt-slice where the vertical axis represents time (bottom = t_0 and top = t_{95})



(a) (b) (c) (d) (e)

(f) (g) (h) (i) (j)

Fig. 13 Reeb graphs for the artificial (top row) and metal foam (bottom row) images. Each column shows a 3D cross-section of the fitted 4D tetrahedral mesh with the Reeb graph overlaid. The Reeb graph has been projected to 3D and its nodes are colored according to their time coordinate (scaled to be in $[-1, 1]$) as in Fig. 12

4 Discussion

The method presented in this paper can segment merging or splitting objects in 4D. As demonstrated in Section 3.1, this can be achieved even in the presence of severe noise, although with a reduced quality. When objects are merging, our method also significantly outperformed the fixed-topology GC methods while being faster and using the same amount of memory. The MRF methods gave the most accurate segmentation, which is expected given that this is a binary segmentation problem with Gaussian noise. However, they used significantly more time and memory.

Furthermore, the time and memory benefits of our method will only improve as the image size increases, since the resource use of our method only depends on the complexity of the object to segment and *not* on the size of the input image. This is not the case for the compared methods. Finally, storing the segmentation as a tetrahedral mesh also results in an extremely compact description of the segmented objects – indeed, of the tested methods, ours provided the most compact segmentation. For the foaming aluminum data, which took up 26 GB of space, the resulting mesh only uses around 8.5 MB which is a reduction factor of more than 3,000.

Our method also offers increased robustness regarding quantifying the evolution of object topology. As we explicitly model the hypersurface boundary in 4D, splits/merges can be automatically detected as saddle points and do not need manual heuristics or collision detection like sequential methods. Furthermore, this also means that if some level sets intersect due to segmentation errors it is easily discarded as an error and not a merge/split — something which is not possible with sequential or 3D/4D voxel-based approaches. These benefits are unique to our object representation.

The main challenge with our method is how to maintain the mesh quality during deformation. With too high a regularization, the method cannot achieve a sufficient quality of fit due to over-smoothing. With too low, the tetrahedral mesh may start to invert itself which results in a degenerate fit. Currently, finding the suitable balance between step size and regularization strength requires manual tuning. Furthermore, while Laplacian smoothing works well to regularize the spatial components of the fit, the spatio-temporal saddle points, that form at splits/merges (see Fig. 11(e)), are sensitive to over smoothing. Finally, for more complex image data (*e.g.* medical scans), an intensity-based deformation force may not be adequate. In these cases, performance may be improved by adapting approaches from [47–50] where the output of a neural network is used to guide the deformation of a contour in 2D images.

5 Conclusion

We presented a method for simultaneously segmenting and tracking merging or splitting objects in 4D images. This was achieved by fitting a discretized hypersurface to the 3D boundary of the 4D simply connected component comprising the object of interest. Applications to artificial and real 4D images showed the method was able to successfully segment, track, and quantify merging or splitting objects, even in the presence of noise. The method achieved comparable accuracy to existing methods with a resource use equal to or better than sequential 3D methods. Furthermore, our method only scales with object complexity and not image size making it suitable for the analysis of very large images.

Acknowledgments. We want to thank Rajmund Mokso and Francisco García-Moreno for providing the 4D image of foaming metal.

References

[1] Peng, P., Lekadir, K., Gooya, A., Shao, L., Petersen, S.E., Frangi, A.F.: A review of heart chamber segmentation for structural and functional analysis using cardiac magnetic resonance imaging. Magnetic Resonance Materials in Physics, Biology and Medicine **29**(2), 155–195 (2016)

[2] García-Moreno, F., Kamm, P.H., Neu, T.R., Bülk, F., Mokso, R., Schlepütz, C.M., Stampanoni, M., Banhart, J.: Using x-ray tomoscopy to explore the dynamics of foaming metal. Nature Communications **10**(1), 1–9 (2019)

[3] Maire, E., Withers, P.J.: Quantitative x-ray tomography. International Materials Reviews **59**(1), 1–43 (2014)

[4] Raufaste, C., Dollet, B., Mader, K., Santucci, S., Mokso, R.: Three-dimensional foam flow resolved by fast x-ray tomographic microscopy. Europhysics Letters **111**(3), 38004 (2015)

[5] Mikula, K., Peyriéeras, N., Remešíková, M., Smíšek, M.: 4d numerical schemes for cell image segmentation and tracking. In: Finite Volumes for Complex Applications VI Problems & Perspectives, pp. 693–701 (2011)

[6] Gao, Y., Phillips, J.M., Zheng, Y., Min, R., Fletcher, P.T., Gerig, G.: Fully convolutional structured lstm networks for joint 4d medical image segmentation. In: IEEE International Symposium on Biomedical Imaging, pp. 1104–1108 (2018). IEEE

[7] McInerney, T., Terzopoulos, D.: A dynamic finite element surface model for segmentation and tracking in multidimensional medical images with application to cardiac 4d image analysis. Computerized Medical Imaging and Graphics **19**(1), 69–83 (1995)

[8] Kashyap, S., Zhang, H., Sonka, M.: Accurate fully automated 4d segmentation of osteoarthritic knee mri. Osteoarthritis and Cartilage **25**, 227–228 (2017)

[9] Montagnat, J., Delingette, H.: Space and time shape constrained deformable surfaces for 4d

medical image segmentation. In: International Conference on Medical Image Computing and Computer-Assisted Intervention, pp. 196–205 (2000). Springer

[10] Oguz, I., Abramoff, M.D., Zhang, L., Lee, K., Zhang, E.Z., Sonka, M.: 4d graph-based segmentation for reproducible and sensitive choroid quantification from longitudinal oct scans. Investigative Ophthalmology & Visual Science **57**(9), 621–630 (2016)

[11] Wu, X., Chen, D.Z.: Optimal net surface problems with applications. In: International Colloquium on Automata, Languages, and Programming, pp. 1029–1042 (2002). Springer

[12] Wang, Y., Zhang, Y., Xuan, W., Kao, E., Cao, P., Tian, B., Ordovas, K., Saloner, D., Liu, J.: Fully automatic segmentation of 4d mri for cardiac functional measurements. Medical Physics **46**(1), 180–189 (2019)

[13] Lorenzo-Valdés, M., Sanchez-Ortiz, G.I., Elkington, A.G., Mohiaddin, R.H., Rueckert, D.: Segmentation of 4d cardiac mr images using a probabilistic atlas and the em algorithm. Medical Image Analysis **8**(3), 255–265 (2004)

[14] Wolz, R., Heckemann, R.A., Aljabar, P., Hajnal, J.V., Hammers, A., Lötjönen, J., Rueckert, D., Initiative, A.D.N., *et al.*: Measurement of hippocampal atrophy using 4d graph-cut segmentation: application to adni. NeuroImage **52**(1), 109–118 (2010)

[15] Choy, C., Gwak, J., Savarese, S.: 4d spatio-temporal convnets: Minkowski convolutional neural networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 3075–3084 (2019)

[16] García-Moreno, F., Kamm, P.H., Neu, T.R., Bülk, F., Noack, M.A., Wegener, M., von der Eltz, N., Schlepütz, C.M., Stampanoni, M., Banhart, J.: Tomoscopy: Time-resolved tomography for dynamic processes in materials. Advanced Materials **33**(45), 2104659 (2021)

[17] Mokso, R., Schlepütz, C.M., Theidel, G., Billich, H., Schmid, E., Celcer, T., Mikuljan, G., Sala, L., Marone, F., Schlumpf, N., *et al.*: Gigafrost: the gigabit fast readout system for tomography. Journal of Synchrotron Radiation **24**(6), 1250–1259 (2017)

[18] Kass, M., Witkin, A., Terzopoulos, D.: Snakes: Active contour models. International Journal of Computer Vision (IJCV) **1**(4), 321–331 (1988)

[19] Cohen, I., Cohen, L.D., Ayache, N.: Using deformable surfaces to segment 3-d images and infer differential structures. Computer Vision, Graphics, and Image Processing **56**(2), 242–263 (1992)

[20] Guillemin, V., Pollack, A.: Differential Topology vol. 370. American Mathematical Society, Providence (2010)

[21] Chan, T.F., Vese, L.A.: Active contours without edges. IEEE Transactions on Image Processing **10**(2), 266–277 (2001)

[22] Desbrun, M., Meyer, M., Schröder, P., Barr, A.H.: Implicit fairing of irregular meshes using diffusion and curvature flow. In: Proceedings of SIGGRAPH, pp. 317–324 (1999)

[23] Dahl, V.A., Dahl, A.B., Hansen, P.C.: Computing segmentations directly from x-ray projection data via parametric deformable curves. Measurement Science and Technology **29**(1), 014003 (2017)

[24] Fujiwara, K.: Eigenvalues of laplacians on a closed riemannian manifold and its nets. Proceedings of the American Mathematical Society **123**, 2585–2594 (1995)

[25] Egger, J., Bauer, M.H., Kuhnt, D., Carl, B., Kappus, C., Freisleben, B., Nimsky, C.: Nugget-cut: a segmentation scheme for spherically-and elliptically-shaped 3d objects. In: Joint Pattern Recognition Symposium, pp. 373–382 (2010). Springer

[26] Wang, Y., Beichel, R.: Graph-based segmentation of lymph nodes in ct data. In: International Symposium on Visual Computing, pp. 312–321 (2010). Springer

[27] Lorensen, W.E., Cline, H.E.: Marching cubes: A high resolution 3d surface construction algorithm.

In: Proceedings of SIGGRAPH, vol. 21, pp. 163–169 (1987). ACM

[28] Si, H.: Tetgen, a delaunay-based quality tetrahedral mesh generator. ACM Transactions on Mathematical Software **41**(2), 11 (2015)

[29] Lee, J.M., Chow, B., Chu, S.-C., Glickenstein, D., Guenther, C., Isenberg, J., Ivey, T., Knopf, D., Lu, P., Luo, F., *et al.*: Manifolds and differential geometry. Topology **643**, 658 (2009)

[30] Axler, S.J.: Linear Algebra Done Right vol. 2. Springer, New York (1997)

[31] Horn, R.A., Johnson, C.R.: Matrix Analysis. Cambridge University Press, Cambridge (2012)

[32] Chu, A., Fu, C.-W., Hanson, A., Heng, P.-A.: Gl4d: A gpu-based architecture for interactive 4d visualization. IEEE Transactions on Visualization and Computer Graphics **15**(6), 1587–1594 (2009)

[33] Southern, R.: Animation manifolds for representing topological alteration. Technical report, University of Cambridge, Computer Laboratory (2008)

[34] Bærentzen, J.A., Aanaes, H.: Signed distance computation using the angle weighted pseudonormal. IEEE Transactions on Visualization and Computer Graphics **11**(3), 243–253 (2005)

[35] Todhunter, I.: Spherical Trigonometry, for the Use of Colleges and Schools: with Numerous Examples. Macmillan, Cambridge (1863)

[36] Zhou, Y., Chen, W., Tang, Z.: An elaborate ambiguity detection method for constructing isosurfaces within tetrahedral meshes. Computers & Graphics **19**(3), 355–364 (1995)

[37] Shinagawa, Y., Kunii, T.L., Kergosien, Y.L.: Surface coding based on morse theory. IEEE Computer Graphics and Applications **11**(05), 66–78 (1991)

[38] Reeb, G.: Sur les points singuliers d'une forme de pfaff completement integrable ou d'une fonction numerique [on the singular points of a completely integrable pfaff form or of a numerical function]. Comptes Rendus Acad. Sciences Paris **222**, 847–849 (1946)

[39] Chen, X., Pan, L.: A Survey of Graph Cuts/-Graph Search Based Medical Image Segmentation. IEEE Reviews in Biomedical Engineering **11**, 112–124 (2018)

[40] Li, S.Z.: Markov Random Field Modeling in Image Analysis. Springer, London (2009)

[41] Boykov, Y., Kolmogorov, V.: An Experimental Comparison of Min-Cut/Max-Flow Algorithms for Energy Minimization in Vision. IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI) **26**(9), 1124–1137 (2004)

[42] Li, K., Wu, X., Chen, D.Z., Sonka, M.: Optimal surface segmentation in volumetric images-a graph-theoretic approach. IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI) **28**(1), 119–134 (2005)

[43] Haeker, M., Wu, X., Abràmoff, M., Kardon, R., Sonka, M.: Incorporation of regional information in optimal 3-d graph search with application for intraretinal layer segmentation of optical coherence tomography images. In: Biennial International Conference on Information Processing in Medical Imaging, pp. 607–618 (2007). Springer

[44] Aspert, N., Santa-Cruz, D., Ebrahimi, T.: Mesh: Measuring errors between surfaces using the hausdorff distance. In: Proceedings of the IEEE International Conference on Multimedia and Expo, vol. 1, pp. 705–708 (2002). IEEE

[45] Masood, T.B., Budin, J., Falk, M., Favelier, G., Garth, C., Gueunet, C., Guillou, P., Hofmann, L., Hristov, P., Kamakshidasan, A., *et al.*: An overview of the topology toolkit. In: Proceedings of Topological Methods in Data Analysis and Visualization (2019)

[46] Tierny, J., Favelier, G., Levine, J.A., Gueunet, C., Michaux, M.: The topology toolkit. IEEE transactions on visualization and computer graphics **24**(1), 832–842 (2017)

[47] Acuna, D., Ling, H., Kar, A., Fidler, S.: Efficient interactive annotation of segmentation datasets with polygon-rnn++. In: Proceedings of the IEEE

Conference on Computer Vision and Pattern Recognition (CVPR), pp. 859–868 (2018)

[48] Castrejon, L., Kundu, K., Urtasun, R., Fidler, S.: Annotating object instances with a polygon-rnn. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 5230–5238 (2017)

[49] Ling, H., Gao, J., Kar, A., Chen, W., Fidler, S.: Fast interactive object annotation with curve-gcn. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 5257–5266 (2019)

[50] Peng, S., Jiang, W., Pi, H., Li, X., Bao, H., Zhou, X.: Deep snake for real-time instance segmentation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 8533–8542 (2020)

4 Learning-Based Segmentation

This chapter includes the contributions related to incorporating topology and shape knowledge into learned segmentation models. The focus has been on reducing the annotation effort required to train deep learning classifiers and improve segmentation results for existing models via post-processing. The contributions are loosely based on the deformable contours framework introduced by Michael Kass, Andrew Witkin, and Demetri Terzolpoulos in that an initial surface is iteratively deformed to fit incomplete or noisy data. Specifically, the contributions are:

E Weakly Supervised Volumetric Image Segmentation with Deformed Templates
Here, we introduce a method to train a 3D convolutional neural network (CNN) to perform segmentation from only point annotations. As a user inputs points through a graphical user interface (GUI) a mesh is deformed to fit the points while being regularized by a low-level smoothness prior. The fitted mesh is then rasterized to a binary label volume and a CNN is trained to predict this. A second CNN then takes the predicted label volume and tries to reconstruct the original image. This improves the segmentation since the labels based on the fitted mesh may be too coarse due to the low-level smoothness prior applied.

F Deep Active Latent Surfaces for Medical Geometries
Next, we introduce a new mesh-based shape model to efficiently learn a shape prior for downstream tasks. Our model represents shapes as a mesh with a latent vector at every vertex. During training, the latent vectors must have the same value which regularizes the model while it learns a shape space. To fit a new shape after training, we update each latent vector independently under a simple smoothness regularization. Crucially, the smoothness regularization is applied *only to the latent vectors*. Hence, our model can be seen as an active contour model, but where the surface is placed in the latent space instead of Euclidean image space. Since the model can smoothly blend different shapes together, it does not require that the latent space contains a single vector that describes the whole shape. This makes the model very flexible which we demonstrate on several tasks: fitting to points (like in Paper E), fitting to planar curve annotations, and fitting to a noisy image segmentation.

4.1 Paper E: Weakly Supervised Volumetric Image Segmentation with Deformed Templates

Udaranga Wickramasinghe, Patrick M. Jensen, Mian Shah, Jiancheng Yang, Pascal Fua, International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI), 2022,
DOI: 10.1007/978-3-031-16443-9_41.

Weakly Supervised Volumetric Image Segmentation with Deformed Templates

Udaranga Wickramasinghe[1(✉)], Patrick Jensen[1,2], Mian Shah[1], Jiancheng Yang[1,3], and Pascal Fua[1]

[1] EPFL, Lausanne, Switzerland
udaranga.wickramasinghe@epfl.ch
[2] DTU, Lyngby, Denmark
[3] Shanghai Jiao Tong University, Shanghai, China

Abstract. There are many approaches to weakly-supervised training of networks to segment 2D images. By contrast, existing approaches to segmenting volumetric images rely on full-supervision of a subset of 2D slices of the 3D volume. We propose an approach to volume segmentation that is truly weakly-supervised in the sense that we only need to provide a sparse set of 3D points on the surface of target objects instead of detailed 2D masks. We use the 3D points to deform a 3D template so that it roughly matches the target object outlines and we introduce an architecture that exploits the supervision it provides to train a network to find accurate boundaries. We evaluate our approach on Computed Tomography (CT), Magnetic Resonance Imagery (MRI) and Electron Microscopy (EM) image datasets and show that it substantially reduces the required amount of effort.

1 Introduction

State-of-the-Art volumetric segmentation techniques rely on Convolutional Neural Networks (CNNs) operating on image volumes [3,23]. However, their performance depends critically on obtaining enough annotated data, which itself requires expert knowledge and is both tedious and expensive.

Weakly-supervised image segmentation techniques can be used to mitigate this problem. They typically rely on tag annotations [8,10] or coarse object annotations in the form of point annotations [20,33], bounding box annotations [9], scribbles [33] or approximate target shapes [14]. However, these techniques have been mostly demonstrated in 2D and do not provide enough information when segmenting complex shapes, such as the liver or the hippocampus. For 3D volume segmentation, the dominant approach is to fully label a subset of 2D slices [3,5]. This is often referred to as *weak supervision*, even though it requires full supervision within individual slices.

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/978-3-031-16443-9_41.

L. Wang et al. (Eds.): MICCAI 2022, LNCS 13435, pp. 422–432, 2022.
https://doi.org/10.1007/978-3-031-16443-9_41

By contrast, we propose a truly weakly-supervised approach that only requires a sparse set of 3D points on the surface of the target objects instead of the usual 2D masks in selected slices. Given an appropriate user-interface, this is much faster and easier because it eliminates the need to painstakingly outline fine details, as shown in Fig. 1. To this end, we introduce the **Weak-Net** architecture depicted by Fig. 2. It comprises two U-Net-like networks. The first one produces a segmentation map that matches a rough model of the target object obtained from the 3D point annotations. The second one takes the map as input and uses it to reconstruct the original image, which forces the segmentation boundaries to be accurate even though those of the template are not.

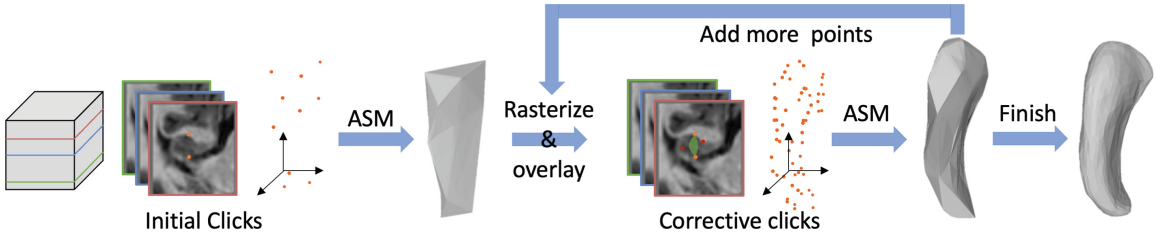


Fig. 1. Iterative annotation strategy. The annotator provides a few 3D points, which we fit to the template using an Active Surface Model (ASM) [28]. The result is rasterized and overlaid on the images. The annotator can then add more and deform the template again as needed.

We evaluate the performance of **Weak-Net** on Computed Tomography (CT), Magnetic Resonance Imagery (MRI) and Electron Microscopy (EM) datasets. We show that it outperforms the standard approach to weak-supervision in 3D at a reduced supervision cost. More specifically, we can deliver the same accuracy as when fully annotating 2D slices for less than a third of the annotation effort. This matters because annotators typically are experts whose time is both scarce and valuable. Furthermore, it creates the basis for interactive annotating strategies that deliver the full accuracy at a lower cost than full supervision.

2 Related Work

We review current approaches to weak-supervision for 2D and 3D image segmentation. We then discuss using atlases to segment biomedical image volumes.

2.1 Weakly-Supervised Image Segmentation

Segmenting 2D Images. Tag and box annotations are among the weakest forms of annotations used to segment natural and medical images [6,9,10]. However, they rarely provide enough supervision for accurate results. By contrast, point annotations [20], scribbles [33], and approximate shape annotation [14] can be used to provide useful shape information.

The annotation process can be sped-up using dynamic programming [19] or deformable contours [12]. This makes it possible to mark only a subset of points along a contour and have the system refine it to match the target object boundary. Unfortunately, these algorithms are hard to deploy effectively in medical imagery because there are many contours besides those of interest and they can easily confuse these algorithms. This is addressed in [21] by introducing *deep deformable contours* that only need a simple approximate contour for initialization purposes. In [2,15], the annotator is brought into the loop by giving corrective clicks when necessary. However, these deep contours require fully labelled data for their own training.

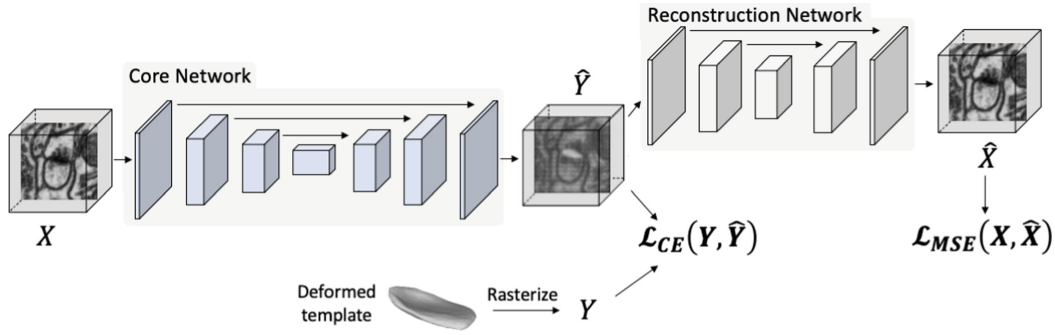


Fig. 2. Weak-Net architecture. A first U-Net takes the image $\mathbf{X}$ as input and outputs a segmentation $\hat{\mathbf{Y}}$, which is in turn fed to a second U-Net that outputs a reconstructed image $\hat{\mathbf{X}}$. Training is achieved by jointly minimizing L_{mse} and L_{ce}. This encourages $\hat{\mathbf{X}}$ to resemble the original image and $\hat{\mathbf{Y}}$ to be similar to $\mathbf{Y}$, a roughly aligned version of a template.

In the result section, we show that supplying a few 3D points, as we do, is faster than using these techniques to delineate whole 2D contours in slices, as is often done.

Segmenting 3D Volumes. By far the most prevalent approach to weak-supervision for 3D segmentation, is to *fully* annotate subsets of slices from the training volumes [3]. Even though this reduces the segmentation effort, the annotator still has to carefully trace the object boundary in those slices. The effort can be reduced by using scribbles in individual slices [5] or any of the semi-automated techniques described above. Unfortunately, the effectiveness of those that do not require training is limited [12,19] while the others [2,15,21] require full supervision and are not applicable in our scenario.

Part of the problem is that these annotation techniques operate purely in 2D without exploiting the 3D nature of the data. Because we do so, we only need a comparatively small number of point annotations for effective training.

2.2 Template Based Approaches

Shape priors have long been used for image segmentation [7] and much recent work use shape priors in conjunction with deep nets [18,29] For medical imaging,

these priors are usually supplied in the form of sophisticated templates known as Probabilistic Atlases (PAs) that assign to each pixel or voxel a probability of belonging to a specific class. They are typically built by fusing multiple manually annotated images and used as auxiliary CNN inputs to provide localization priors that help the network find structures of interest. The PAs can be either very detailed, as in [25], to model structures that are known in detail or very rough, as in [27], to deal with 3D structures whose shape can vary significantly.

PAs built by annotating points have been used in medical imaging [11,22] as a source of prior information. They are often referred to as seed layers and come in two main flavors, Gaussian priors [22] or binary seed layers [11]. These seed layers either indicate points inside the object [11,32] or points on the boundary of the object [17,26]. In the context of deep learning, PAs have been mostly used in fully-supervised approaches [11,22,25]. An exception is the work of [25]. However, they are only used for pre-training purposes. Another are the one-shot and few-shot learning-based segmentation algorithms of [4]. However, they require a few fully annotated target objects as the atlases, which we do not.

2.3 Image Reconstruction

Image reconstruction is used for semi-supervised [16] and unsupervised [31] image segmentation as an auxiliary task to improve the results. In this work, we demonstrate that this idea also applies in a weakly-supervised setting to improve the segmentations produced by the core-network trained with rough annotations. In our framework, the image reconstruction network helps refine the rough initial shapes we obtain from the annotations. There are alternative approaches to boundary refinement [1] but they are designed for 2D segmentation and extending them to 3D would be non-trivial.

3 Method

Our approach to training a network involves an annotator providing only a sparse set of 3D points on the surface of target objects for each image volume, as opposed to carefully annotating several individual slices in each. These points are used to deform a template, such as those shown in Fig. 1, using active surface models [28]. This provides a rough indication of where the target object boundary is. **Weak-Net** uses it to learn weights that yield accurate object boundaries. In short, we provide minimal human input at training time so that, at inference time, the trained network can be used without human intervention.

3.1 Network Architecture and Losses

Weak-Net is depicted by Fig. 2 and comprises two separate U-Net networks [3]. The first takes as input an image volume $\mathbf{X}$ of $D \times H \times W$, where D, H, W stand for depth, height and width. It outputs a tensor $\hat{\mathbf{Y}}$ of dimension $D \times H \times W$ that stores the probabilities of each voxel belonging to the foreground. The second

takes $\hat{\mathbf{Y}}$ as input and yields $\hat{\mathbf{X}}$, which is of the same dimension as $\mathbf{X}$. Ideally, $\hat{\mathbf{Y}}$ should be the desired segmentation and $\hat{\mathbf{X}}$ should be equal to $\mathbf{X}$.

To train **Weak-Net**, we minimize a weighted sum of two losses

$$\mathcal{L} = \mathcal{L}_{ce} + \lambda \mathcal{L}_{mse}, \tag{1}$$

$$\mathcal{L}_{ce} = -\sum_{i,j,k} \mathbf{Y}_{i,j,k} \log(\hat{\mathbf{Y}}_{i,j,k}), \quad \mathcal{L}_{mse} = \sum_{i,j,k} \mathbf{W}^{mse}_{i,j,k} (\mathbf{X}_{i,j,k} - \hat{\mathbf{X}}_{i,j,k})^2,$$

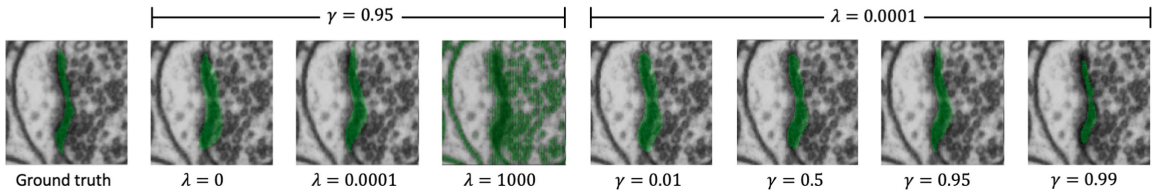


Fig. 3. Impact of the λ parameter in the loss of Eq. 1 and the thresholding parameter γ. (Col. 2–4) When $\lambda = 0$, the segmentation is very similar to the template. When λ is large, the segmentation follows boundaries that exist in the image but are not necessarily the right ones. In between, the boundaries are correct. (Col. 5–8) We set λ to 10^{-4} and vary γ.

where $\mathbf{Y}$ is the rasterized template fitted to the target object and λ is a scalar that controls the influence of the second loss. $\mathcal{L}_{ce}$ is a standard cross entropy loss whose minimization promotes similarity between the rasterized template $\mathbf{Y}$ and the segmentation $\hat{\mathbf{Y}}$. $\mathcal{L}_{mse}$ is a voxel-wise mean squared error in which the individual voxels are given weights $W^{mse}_{i,j,k}$ whose value is high within a distance d from boundaries in $\mathbf{Y}$ and low elsewhere. At inference time, we only use the first U-Net, which we will refer to as our *core network*, and obtain the final segmentation by thresholding its output $\hat{\mathbf{Y}}$ using a threshold γ.

Minimizing $\mathcal{L}_{ce}$ during training ensures that the segmentations will be roughly correct. However, because the template can only be expected to provide a coarse depiction of the object, this is not enough. We therefore also minimize $\mathcal{L}_{mse}$ to force the network to yield accurate boundaries. Figure 3 illustrates the influence of the γ and λ parameters on a real image. In practice, the results are insensitive to how they are chosen over a wide range. However, minimizing $\mathcal{L}_{ce}$ alone ($\lambda = 0$) produces boundaries in $\hat{\mathbf{Y}}$ that are exactly those of the template while minimizing $\mathcal{L}_{mse}$ alone ($\lambda = 1000$) yields boundaries that exist in the image but are not necessarily those we are looking for. Minimizing a properly weighted sum of the two yields segmentations that conform to the template while matching actual image boundaries.

3.2 Template Deformation

The template $\mathbf{Y}$ of Eq. 1 should approximately match the target structure. Hence, the annotator should supply points that are distributed across the object surface. These points are then used to deform the template. In practice, For

structures of genus 0, we start from a simple spherical template but more complex ones are possible. As we increase the number of points, we get increasingly refined templates, as shown in Fig. 4.

To perform this deformation interactively, we developed a GUI that relies on Active Surface Models (ASMs) [28] implemented as a MITK [30] plugin. It lets the annotator supply a few points by clicking on 2D cross sections of the input image volume. The ASM then deforms the template in real-time and overlays it on the image data, both as 2D cross sections and 3D surface renderings. The annotator can then add more points wherever the deformed template is too far from the target organ's boundary and iterate as often as necessary. This effectively puts the human in the loop in a painless and practical way. We illustrate this in a video that can be found in the supp. material.

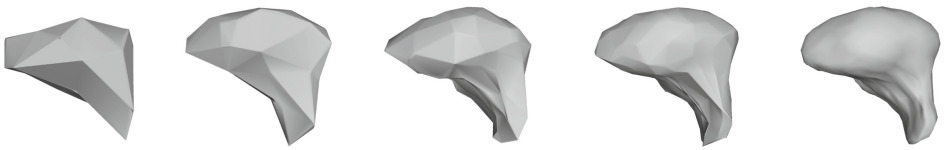


Fig. 4. Deformed templates. (a, b, c, d, e) Deformed template using $N = 25, 50, 125, 250$, and 3661 user-supplied 3D points when segmenting a liver. $N = 3661$ corresponds to full annotations in all slices.

4 Experiments

4.1 Datasets, Metrics and Baseline

We use three datasets acquired using MRI, FIB-SEM, and CT. MRI dataset consists of 260 labeled MRI image cubes of hippocampuses from the Medical Segmentation Decathlon [24]. CT image dataset consists of 20 labeled CT image cubes of the liver from the CHAOS challenge [13]. EM Dataset consist of 26 image cubes of Synaptic Junctions from a $500 \times 500 \times 200$ FIB-SEM image stack of a mouse cortex [29].

Our aim is to produce segmentations of these image values such that they are above a given quality threshold using as few annotations as possible. To quantify this goal, we use two metrics, one for quality and the other for annotation effort. We use the standard IoU metric [3] to quantify segmentation quality. We use the number of points provided by the annotator as a proxy for amount of effort. When the annotator provides individual points, this is clearly proportional to the time spent. When the annotator outlines contours on 2D slices, this becomes the sum of the contour lengths in each slice. It could be argued that this is an overestimate because the task is easier. However, in our experience it is not because precisely outlining a contour requires deliberation.

As discussed in Sect. 2.1, the generally accepted way to provide weak-supervision for 3D image segmentation is to fully annotate a few 2D slices [3]. To provide a baseline, we therefore use this approach to train a 3D U-Net, as described in [3]. For a fair comparison, we use the same one as in **Weak-Net**.

4.2 Comparative Results

We exploit the real-time performance of active surface models to enable the annotator to provide a few points, deform the template accordingly, and then add more points where the deformed shape is not satisfactory. We first benchmark this scenario using human annotations and then provide results using simulated annotation on the object surfaces to eliminate the subjective element it contains.

Table 1. Performance given human annotations. Values are given as mean±std. **Weak-Net** consistently outperforms the baseline in terms of both IoU and annotation effort required to achieve it.

		Hippocampus	Liver	Syn. junction
Baseline	IoU (%)	71.2 ± 0.9	81.9 ± 0.7	68.2 ± 0.9
	Annotation effort (%)	12.4%	6.9%	10.8%
Weak-Net	IoU (%)	74.2 ± 0.7	84.2 ± 1.0	71.5 ± 1.4
	Annotation effort (%)	10.1%	6.8%	9.2%
Full annotation		79.3 ± 0.4	87.3 ± 0.3	73.3 ± 0.6

Table 2. Annotation time. Values are given as mean ± std. Times are in minutes.

	Hippocampus	Liver
Manual Contouring	6.7 ± 0.7 min	125 ± 19.3 min
3D Region Growing	5.4 ± 1.2 min	75.4 ± 30.5 min
3D Fast Marching	5.7 ± 1.4 min	86.1 ± 25.3 min
Ours	4.6 ± 0.9 min	12.3 ± 1.6 min

4.3 Human Annotations

Weak vs Full Supervision. We compare **Weak-Net** and the baseline against providing full supervision and report the results in Table 1. **Weak-Net** delivers higher IoU numbers than the baseline on all three datasets. It delivers 92 to 97% of the accuracy that can be achieved with full supervision for only 7 to 10% of the annotation effort, as defined above.

Annotation Time. In the results of Table 1, we use the number of points supplied by the annotators to gauge the annotation effort. To complement this, we asked the annotators to manually annotate some images using other MITK tools [30], slice-by-slice *manual contouring* of the borders, *3D Region Growing*, and *3D Fast Marching*. *3D Region Growing* and *3D Fast Marching* produce many false positive and negative regions, as shown in the supp. material. Therefore, we had to perform slice-by-slice corrections to obtain final segmentation using

these two tools. In Table 2, we report the average time it took to fully annotate a single sample using each tool. As the Hippocampus volumes are small, point annotation is only ∼1.5x faster than annotating the full volume. For the large Liver volumes however, point annotation is ∼10x faster, which is significant.

Simulated Annotations. To eliminate subjectivity from our experiments, we use the fact that our datasets are fully annotated to simulate the annotation process using the algorithm described in the supp. material. It is driven by two numbers, N the number of points per sample and P the number of samples we annotate. To keep the number of experiments within a manageable range, we vary both N and P when experimenting on the Hippocampus dataset and only N for the other two. To provide a baseline, we randomly pick a number of slices from three image planes to be annotated and use the ground-truth annotations for these slices. When selecting them, we check that they contain the target object. When evaluating the baseline, we vary the number of slices we use and of samples we annotate.

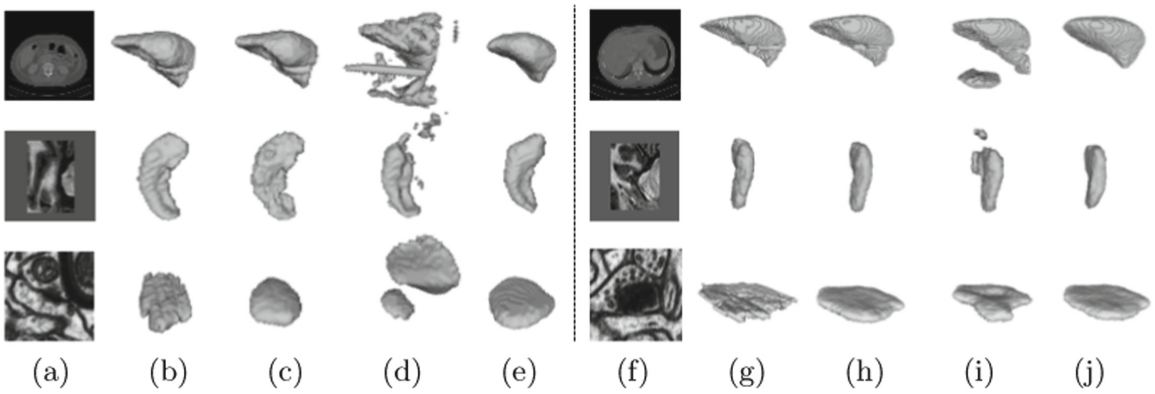


Fig. 5. Segmentation results. (a + f) A slice from the input volume (b + g) Ground truth (c + h) U-Net with full-supervision (d + i) Corresponding Baseline (e) **Weak-Net** trained with 25 points per sample. (f) **Weak-Net** trained with 125 points per sample.

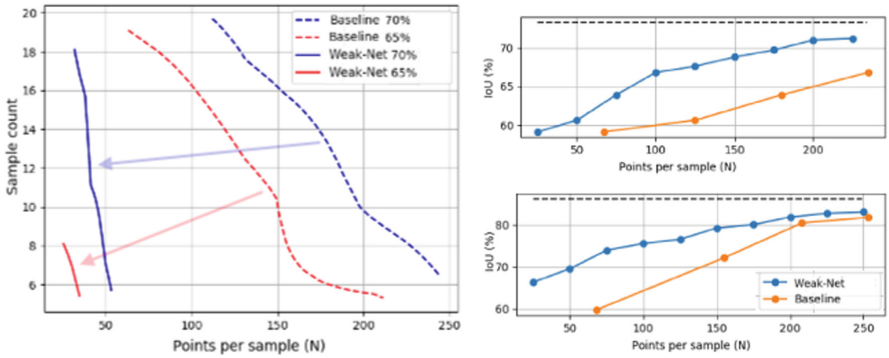


Fig. 6. Performance given synthetic annotations (left). Annotation effort required to achieve 65% and 70% IoU on the hippocampus. The blue and red arrows indicate the effort reduction our approach delivers. (right) IoU as function of the number of points used (N) for the synaptic junction and liver dataset sets. The dashed line denotes fully supervised results. (Color figure online)

We present qualitative results of the experiments in Fig. 5. On the left of Fig. 6, we plot the total annotation effort, that is, the product of the number of points per sample and the number of samples, required to attain a target IoU—either 65% or 70%—in the Hippocampus dataset. As there are many ways to achieve a given IoU by increasing one number while decreasing the other, we draw *iso-IoU* curves. The baseline ones are dashed and ours are full and clearly to the left of the dashed ones. In other words, we need significantly less effort to achieve a similar result. On the right of Fig. 6, we report our results on the other two datasets that contain fewer samples. Hence we used all training samples and plot the IoU as a function of the number of points per sample. For the same number of points, our approach consistently outperforms the baseline. When using it, we cannot annotate less than one slice and, hence, reduce the annotation effort below a certain level, which is why the blue curves extend further to the left than the yellow ones.

5 Conclusion

We have presented a weakly supervised approach to segmenting 3D image volumes that outperforms more traditional approaches that rely on fully annotating individual 2D slices.

It relies on deforming simple spherical templates that incorporate no shape prior. In future work, to further reduce the annotation burden, we will develop more sophisticated templates that are parameterized in terms of low-dimensional latent vectors and can therefore be deformed by specifying even fewer 3D points than we do now.

Acknowledgment. This work was supported in part by a Swiss National Science Foundation grant.

References

1. Acuna, D., Kar, A., Fidler, S.: Devil is in the edges: learning semantic boundaries from noisy annotations. In: Conference on Computer Vision and Pattern Recognition (2019)
2. Akuna, D., Ling, H., Kar, A., Fidler, S.: Efficient interactive annotation of segmentation datasets with Polygon-RNN++. In: Conference on Computer Vision and Pattern Recognition (2018)
3. Çiçek, Ö., Abdulkadir, A., Lienkamp, S.S., Brox, T., Ronneberger, O.: 3D U-Net: learning dense volumetric segmentation from sparse annotation. In: Ourselin, S., Joskowicz, L., Sabuncu, M.R., Unal, G., Wells, W. (eds.) MICCAI 2016. LNCS, vol. 9901, pp. 424–432. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46723-8_49
4. Dalca, A.V., Yu, E., Golland, P., Fischl, B., Sabuncu, M.R., Eugenio Iglesias, J.: Unsupervised deep learning for Bayesian brain MRI segmentation. In: Shen, D., Liu, T., Peters, T.M., Staib, L.H., Essert, C., Zhou, S., Yap, P.-T., Khan, A. (eds.) MICCAI 2019. LNCS, vol. 11766, pp. 356–365. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-32248-9_40

5. Dorent, R., et al.: Scribble-based domain adaptation via co-segmentation. In: Martel, A.L., et al. (eds.) MICCAI 2020. LNCS, vol. 12261, pp. 479–489. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-59710-8_47
6. Feng, X., Yang, J., Laine, A.F., Angelini, E.D.: Discriminative localization in CNNs for weakly-supervised segmentation of pulmonary nodules. In: Descoteaux, M., Maier-Hein, L., Franz, A., Jannin, P., Collins, D.L., Duchesne, S. (eds.) MICCAI 2017. LNCS, vol. 10435, pp. 568–576. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66179-7_65
7. Freedman, D., Zhang, T.: Interactive graph-cut based segmentation with shape priors. In: Conference on Computer Vision and Pattern Recognition, pp. 755–62 (2005)
8. Ge, W., Yanga, S., Yu, Y.: Multi-evidence filtering and fusion for multi-label classification, object detection and semantic segmentation based on weakly supervised learning. In: Conference on Computer Vision and Pattern Recognition (2018)
9. Hsu, C., Hsu, K., Tsai, C., Lin, Y., Chuang, Y.: Weakly supervised instance segmentation using the bounding box tightness prior. In: Advances in Neural Information Processing Systems (2019)
10. Huang, Z., Wang, X., Wang, J., Liu, W., Wang, J.: Weakly-supervised semantic segmentation network with deep seeded region growing. In: Conference on Computer Vision and Pattern Recognition (2018)
11. Januszewski, M., Jain, V.: High-precision automated reconstruction of neurons with flood-filling networks. Nat. Methods **15**, 605–610 (2018)
12. Kass, M., Witkin, A., Terzopoulos, D.: Snakes: active contour models. Int. J. Comput. Vis. **1**(4), 321–331 (1988)
13. Kavur, A., Selver, M.: CHAOS challenge - combined (CT-MR) healthy abdominal organ segmentation. arXiv Preprint (2020)
14. Khoreva, A., Benenson, R., Hosang, J., Hein, M., Schiele, B.: Simple does it: weakly supervised instance and semantic segmentation. In: Conference on Computer Vision and Pattern Recognition, pp. 1665–1674 (2017)
15. Ling, H., Gao, J., Kar, A., Chen, W., Fidler, S.: Fast interactive object annotation with curve-GCN. In: Conference on Computer Vision and Pattern Recognition, pp. 5257–5266 (2019)
16. Liu, X., Thermos, S., O'Neil, A., Tsaftaris, S.A.: Semi-supervised meta-learning with disentanglement for domain-generalised medical image segmentation. In: de Bruijne, M., Cattin, P.C., Cotin, S., Padoy, N., Speidel, S., Zheng, Y., Essert, C. (eds.) MICCAI 2021. LNCS, vol. 12902, pp. 307–317. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-87196-3_29
17. Maninis, K., Caelles, S., Pont-Tuset, J., Gool, L.: Deep extreme cut: from extreme points to object segmentation. In: Conference on Computer Vision and Pattern Recognition (2018)
18. Mirikharaji, Z., Hamarneh, G.: Star shape prior in fully convolutional networks for skin lesion segmentation. In: Frangi, A.F., Schnabel, J.A., Davatzikos, C., Alberola-López, C., Fichtinger, G. (eds.) MICCAI 2018. LNCS, vol. 11073, pp. 737–745. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-00937-3_84
19. Mortensen, E., Barrett, W.: Intelligent scissors for image composition. In: ACM SIGGRAPH, pp. 191–198, August 1995
20. Bearman, A., Russakovsky, O., Ferrari, V., Fei-Fei, L.: What's the point: semantic segmentation with point supervision. In: Leibe, B., Matas, J., Sebe, N., Welling, M. (eds.) ECCV 2016. LNCS, vol. 9911, pp. 549–565. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46478-7_34

21. Peng, S., Jiang, W., Pi, H., Li, X., Bao, H., Zhou, X.: Deep snake for real-time instance segmentation. In: Conference on Computer Vision and Pattern Recognition (2020)
22. Roth, H., et al.: Weakly supervised segmentation from extreme points. In: Zhou, L., et al. (eds.) LABELS/HAL-MICCAI/CuRIOUS 2019. LNCS, vol. 11851, pp. 42–50. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-33642-4_5
23. Shvets, A., Iglovikov, V.: Automatic instrument segmentation in robot-assisted surgery using deep learning. arXiv Preprint (2018)
24. Simpson, A., Menze, B.: A large annotated medical image dataset for the development and evaluation of segmentation algorithms. arXiv Preprint (2019)
25. Spitzer, H., Kiwitz, K., Amunts, K., Harmeling, S., Dickscheid, T.: Improving cytoarchitectonic segmentation of human brain areas with self-supervised Siamese networks. In: Frangi, A.F., Schnabel, J.A., Davatzikos, C., Alberola-López, C., Fichtinger, G. (eds.) MICCAI 2018. LNCS, vol. 11072, pp. 663–671. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-00931-1_76
26. Wang, Z., Acuna, D., Ling, H., Kar, A., Fidler, S.: Object instance annotation with deep extreme level set evolution. In: European Conference on Computer Vision (2020)
27. Wickramasinghe, U., Knott, G., Fua, P.: Probabilistic atlases to enforce topological constraints. In: Shen, D., Liu, T., Peters, T.M., Staib, L.H., Essert, C., Zhou, S., Yap, P.-T., Khan, A. (eds.) MICCAI 2019. LNCS, vol. 11764, pp. 218–226. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-32239-7_25
28. Wickramasinghe, U., Knott, G., Fua, P.: Deep active surface models. In: Conference on Computer Vision and Pattern Recognition (2021)
29. Wickramasinghe, U., Remelli, E., Knott, G., Fua, P.: Voxel2Mesh: 3D mesh model generation from volumetric data. In: Martel, A.L., Abolmaesumi, P., Stoyanov, D., Mateus, D., Zuluaga, M.A., Zhou, S.K., Racoceanu, D., Joskowicz, L. (eds.) MICCAI 2020. LNCS, vol. 12264, pp. 299–308. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-59719-1_30
30. Wolf, I., et al.: The medical imaging interaction toolkit (MITK): a toolkit facilitating the creation of interactive software by extending VTK and ITK. In: Medical Imaging 2004: Visualization, Image-Guided Procedures, and Display (2004)
31. Xia, X., Kulis, B.: W-Net: a deep model for fully unsupervised image segmentation. arXiv Preprint (2017)
32. Yang, L., Wang, Y., Xiong, X., Yang, J., Katsaggelos, A.: Efficient video object segmentation via network modulation. In: Conference on Computer Vision and Pattern Recognition (2018)
33. Zhao, T., Yin, Z.: Weakly supervised cell segmentation by point annotation. IEEE Trans. Med. Imaging **40**, 2736–2747 (2020)

1 Appendix

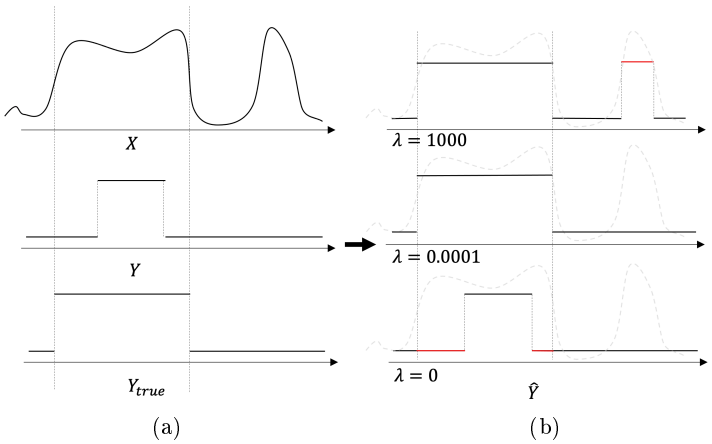


Fig. 1. Using a secondary task to improve segmentations. (a) $\mathbf{X}$ is the input and $\mathbf{Y}_{true}$ is the ground-truth. $\mathbf{Y}$ is the atlas that is roughly correct. (b) When λ is too large, $\mathcal{L}_{mse}$ dominates and the resulting segmentation features a spurious region shown in red that corresponds to image boundaries that are *not* those of the target structure. When λ is zero, only $\mathcal{L}_{ce}$ is minimized and minimizing produces boundaries that are also those of the atlas, which are not at the right place as the denoted by the red lines. For appropriate values of λ, the boundaries are correct and the spurious region is eliminated.

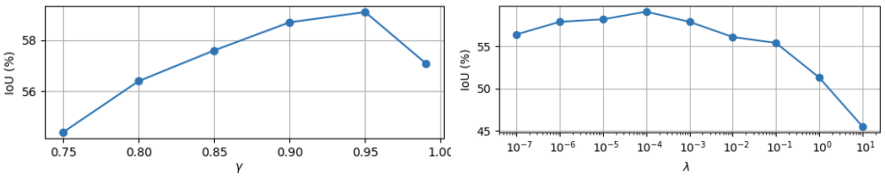


Fig. 2. Influence of λ and γ parameters . IoU (%) as a function of λ in (a), γ in (b) on the Synaptic Junction dataset with $N = 25$.

1.1 Sampling Algorithm

Simulated annotations are generated using the **Improved selection** strategy which is based on **Random selection** strategy.

2

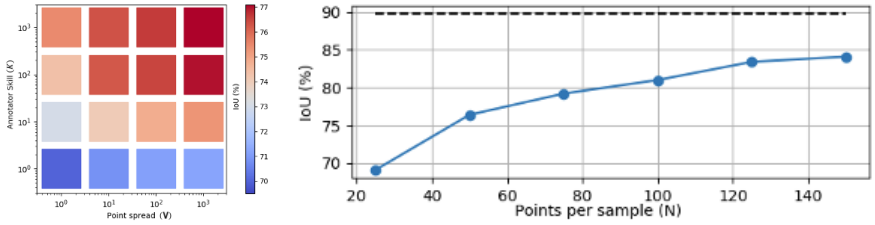


Fig. 3. Annotation simulation on the Liver dataset (a) Reconstruction accuracy (IoU) variation for different values of K and V. (b) Reconstruction accuracy (IoU) variation as a function of the number of annotated points. The black-dashed line indicates the accuracy that would be achieve by annotating all points in each sample, that is 3661 points per sample on average for the liver dataset.

1. **Random selection.** N points are randomly selected across the 3D surface. Ideally, they should be uniformly distributed across the surface but there is no guarantee of that. To emulate the behavior of a conscientious annotator trying to achieve this, we repeat the operation V times and select the set of N points that exhibits the largest intra-point variance. As we increase V, so does the probability that the selected points will indeed be uniformly distributed. The influence of variables N and V is demonstrated in Fig 3.
2. **Improved selection.** We simulate the fact that a skilled annotator will provide the most informative points possible by performing K random annotations as described above, using each one to deform the template, and selecting the one that yields the highest IoU with the ground truth. As K increases, so does the probability that the deformed template will match the ground truth well.

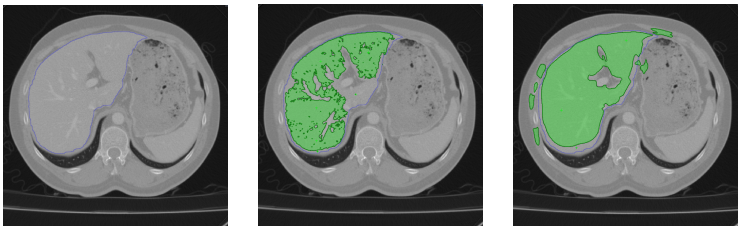


Fig. 4. Use of Fast Marching algorithm from MITK to annotate a liver (a) Image with ground truth contour (in blue) (b) Fast Marching algorithm with small σ (c) Fast Marching algorithm with large σ. In both instances, after the initial segmentation, we have to fix false positive and false negative regions.

4.2 Paper F: Deep Active Latent Surfaces for Medical Geometries

Patrick M. Jensen, Udaranga Wickramasinghe, Anders B. Dahl, Pascal Fua, Vedrana A. Dahl, To be submitted to the International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI).

Deep Active Latent Surfaces for Medical Geometries

Patrick M. Jensen
patmjen@dtu.dk

Udaranga Wickramasinghe
udaranga.wickramasinghe@epfl.ch

Anders B. Dahl
abda@dtu.dk

Pascal Fua*
pascal.fua@epfl.ch

Vedrana A. Dahl*
vand@dtu.dk

Abstract

Shape priors have long been known to be effective when reconstructing 3D shapes from noisy or incomplete data. When using a deep-learning based shape representation, this often involves learning a latent representation, which can be either in the form of a single global vector or of multiple local ones. The latter allows more flexibility but is prone to overfitting. In this paper, we advocate a hybrid approach representing shapes in terms of 3D meshes with a separate latent vector at each vertex. During training the latent vectors are constrained to have the same value, which avoids overfitting. For inference, the latent vectors are updated independently while imposing spatial regularization constraints. We show that this gives us both flexibility and generalization capabilities, which we demonstrate on several medical image processing tasks.

1 Introduction

3D shape reconstruction from noisy or incomplete data usually benefits from the judicious use of shape priors. When using a deep-learning based shape representation, this usually means searching for an appropriate latent representation under regularization losses. This latent vector representation can take the form of either a single global latent vector per shape or a grid of latent vectors.

In either case, it is a challenge to balance regularization against quality of fit to the data. With too much regularization the recovered shapes are too smooth and fine details are lost. With too little, robustness to noise and generalization capabilities will suffer. Our insight is that, when using multiple latent vectors, instead of imposing regularity constraints directly on the surface, we can impose them on the latent vectors. This is effective because, if a latent vector has been trained to model a sharp feature, requiring that this vector be similar to its neighbors will not detract from that. To this end, we represent surfaces as triangulated meshes and advocate using a separate latent vector at each vertex while imposing smoothness constraints on these vectors. To exploit this, we borrow the regularization idea from early approaches to 3D shape modeling [25, 58, 57] that predate deep learning and apply it to the latent vectors instead of the vertex positions.

More specifically, at training time, we use an auto-decoding approach [45] to jointly learn the network weights along with one *common* latent vector for all vertices of each training sample. By contrast, at inference time, we allow the latent vectors to be different at each vertex while enforcing consistency of the vectors by minimizing a regularization loss. Fig. 1 depicts our Deep Active Latent Surfaces (DALS) approach. It makes our model both easy to train from relatively small datasets and very expressive: Because, we learn a single latent vector per shape, we do not require the training set to be huge. At model fitting time, we do not need to find an individual vector that accurately represents a *whole* shape, which can be difficult when the shape is complex. Instead, we can smoothly blend a

*Equal supervision.

Preprint. Under review.

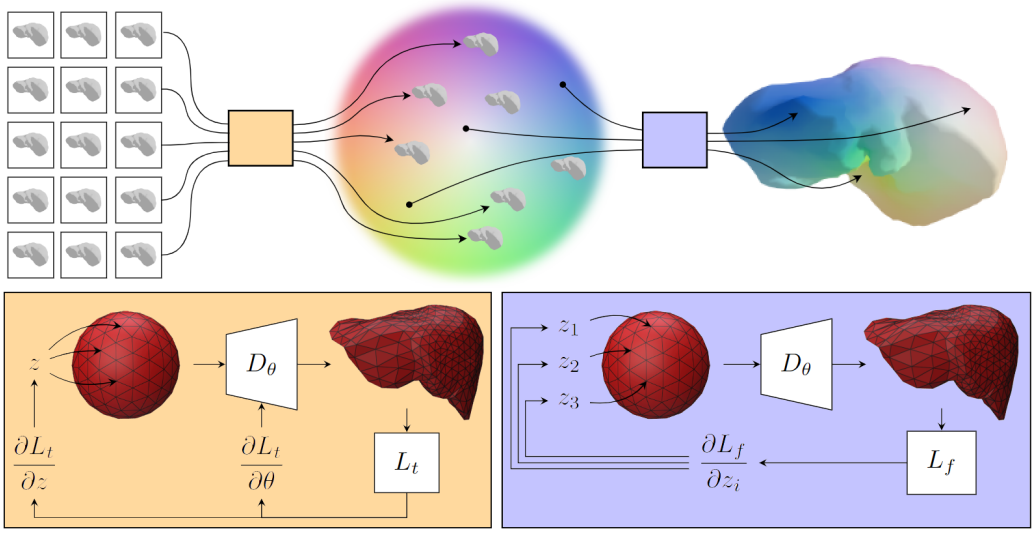


Figure 1: *DALS* **overview. (top)** Given a set of training shapes, we use an auto-decoding approach to learning a latent space of shapes. **(bottom left)** To each training shape is associated a single latent vector $\mathbf{z}$ that is used to compute a translation for each vertex of sphere to minimize the distance between the deformed sphere and the training shape. **(bottom right)** At inference time, the latent vectors $\mathbf{z_1}, \mathbf{z_2}, \ldots, \mathbf{z_N}$ at each one the N sphere vertices are allowed to change independently to minimize a weighted sum of a data loss function and a regularization term that prevents neighboring latent vectors from being too different from each other.

number of vectors to model such a shape, while preserving sharp features. Furthermore, explicitly using a triangulated mesh instead of a grid of latent vectors makes our model more compact and simpler to train because it does not have to waste capacity on modeling empty regions.

2 Related Work

Active Surface Models Active contour models are used to refine contours according to local image properties while remaining smooth. They were first introduced in [25] for interactive delineation and then extended for many different purposes [13]. Active surface models operate on the same principle [57, 58] but replace the contours by triangulated meshes to model 3D surfaces. They have proved very successful for medical [20, 37] and cartographic applications [14], among others, and are still being improved [23, 29, 49]. In recent years, Deep Neural Networks (DNNs) have been used to evaluate the energy that the active contours minimize [18, 35] and, in [31, 32, 46], they are used to directly predict vertex offsets. In [60], active surface models are embedded in special purpose network layers that regularize surface meshes using the same semi-implicit scheme as the original active contours [25].

Correctly balancing the relative influence of the data and regularization terms to avoid over-smoothing while being robust to noise remains a challenge for all these approaches. In [31] smoothing is only added as a loss during training but not during inference. In [60], smoothing is made adaptive to allow sharp edges. Recently, it has been proposed to replace smoothing with preconditioned gradient descent of the external energy [43]. In all these approaches, the regularization tends to flatten sharp geometric features of the mesh geometry, which almost always results in over-smoothing, despite the goodness of fit at a global level. In our work, we side step these issues by focusing the smoothing on the latent space.

Neural Shape Modeling Deep-learning methods are now routinely used to model 3D shapes. Most methods rely on auto-encoders or auto-decoders to produce latent vectors that parameterize the target shapes in terms of triangulated meshes [17, 33, 41], tetrahedral meshes [15, 50], surface patches [16], point clouds [1, 48], voxel grids [6, 12], occupancy functions [8, 39, 47], signed and unsigned distance fields [9, 24], and neural splines [61].

These methods can be classified as those that use a single latent vector to represent a complete shape and those that use multiple ones. Those that use one obtain the latent vector for a shape with an encoder [8, 11] or by directly optimizing a latent vector [11, 24, 33, 45, 59]. These methods are effective but accurately representing all the details may require more than one latent vector as in the methods of [9, 47], who use an encoder, and of [22, 38], who directly optimize the latent vectors. These methods all rely on a grid of latent vectors and a shared decoder to represent the signed distance function of a complete shape. Since each latent vector only has to describe a small part of the complete shape, this greatly increases the model flexibility. It also allows for a much smaller decoder network, which makes inference faster. However, storing a full grid of latent vectors means that some latent vectors are wasted on representing empty space. This increases memory use and training time. The approach of [7, 34] avoids this by using a sparse grid from which unused grid cells are removed. In [36, 55], a tree structures is used to construct sparse multiscale representations. However, these methods either require prior knowledge about the surface or periodic updates of the sparse data structure as it deforms, which makes training more complex. In [42], these issues are alleviated by using a spatial hash encoding which allows the model to implicitly allocate more capacity to regions near the surface. However, this method is designed to represent single shapes and would require non-trivial extensions for model fitting purposes. In our work, we also rely on multiple latent vectors but require neither *a priori* knowledge about the surface nor complex adaptation of a data structure.

3 Deep Active Latent Surfaces

We now describe our Deep Active Latent Surface (*DALS*) approach, which is illustrated by Fig. 1. We represent watertight 3D shapes by triangulated spheres with a latent vector at each vertex. This latent vector along with the vertex coordinates is fed to a decoder $\mathcal{D}_\theta$ that generates an offset vector that is then used to translate the vertex to its final position. Once all the vertices have been translated, we have the final shape such as the one shown on the top right part of Fig. 1. At training time, we use the same latent vector for all vertices whereas, at model fitting time, we allow them to be different but impose spatial consistency on the vectors. This does not preclude the modeling of sharp features because such features can be predicted by individual latent vectors.

3.1 Training Scheme

Formally, let $\mathcal{D}_\Theta$ be a neural network with weights Θ that takes as input a d-dimensional latent vector $\mathbf{z}$ and a 3D location $\mathbf{x}$ and returns an offset $\mathcal{D}_\Theta(\mathbf{z}, \mathbf{x})$. Given a triangulated sphere with V vertices and F facets, we denote by $\mathcal{M}_\theta(\mathbf{z})$ the deformed mesh we obtain by translating each vertex $\mathbf{x}_v$ by $\mathcal{D}_\Theta(\mathbf{z}, \mathbf{x}_v)$ for all v between 1 and V.

Let us further assume we are given a set of N training shapes $S = \{S_1, \ldots, S_N\}$. As in [45], we can simultaneously learn Θ and a $\mathbf{z}_i$ for each S_i by looking for

$$\Theta^*, \mathbf{z}_1^*, \ldots, \mathbf{z}_N^* = \underset{\Theta, \mathbf{z}_1, \ldots, \mathbf{z}_N}{\arg\min} \sum_{i=1}^{N} \mathcal{L}_{\text{dat}}(\Theta, \mathbf{z}_i, S_i) \;, \tag{1}$$

$$\mathcal{L}_{\text{dat}}(\Theta, \mathbf{z}, S_i) = L_{\text{cf}}(\mathcal{M}_\theta(\mathbf{z}), S_i) + \lambda_{\text{reg}} L_{\text{reg}}(\mathcal{M}_\theta(\mathbf{z})) + \lambda_{\text{n}} \|\mathbf{z}\|^2 \;,$$

where L_{cf} is the Chamfer distance [53], L_{reg} is shape regularization term, and λ_{reg} and λ_{n} are weighting constants.

In practice, we take $\mathcal{D}_\Theta$ to be an MLP with three hidden layers of size 724, 724, and 362, which takes as input a concatenation of $\mathbf{x}$ and $\mathbf{z}$. We use ReLU activations for the hidden layers and none for the last layer. Before each ReLU activation we use layer normalization [3]. Our initial spherical triangulation is a subdivided icosahedron. We use a pointwise MLP instead of a mesh based decoder because we want to learn a mapping from the surface of a sphere conditioned on a latent vector rather than a mapping from a specific template mesh. To this end, we also randomly rotate the template during training to specific vertex placement. This enables us to use any template mesh without changing the decoder. We will take advantage of this at inference time by progressively increasing the mesh resolution.

The L_{reg} term in Eq. 1 is intended to encourage the generation of high quality meshes. We could have expressed it in terms of the Laplacian as is often done but we found experimentally that it tends to

result in meshes that are too smooth. Instead, as in [4], we take it to be

$$L_{\text{reg}}(\mathcal{M}) = 1 - \frac{4\sqrt{3}}{|\mathcal{F}|} \sum_{f \in \mathcal{F}} \frac{A_f}{a_f^2 + b_f^2 + c_f^2} \ , \tag{2}$$

where $\mathcal{F}$ stands for the mesh facets, a_f, b_f, c_f for the lengths of the three edges of facet f, and A_f for its area. It is easy to compute and has favorable properties for numerical optimization [51]. In practice, minimizing L_{reg} promotes regularity without directly penalizing high frequency features, as illustrated by Fig. 2.

3.2 Fitting Scheme

Let $\mathcal{D} = \mathcal{D}_{\Theta^*}$ be the network we trained in Section 3.1 and let $\mathbf{Z} \in \mathbb{R}^{V \times d}$ whose V rows are latent vectors, one for each vertex of our triangulated sphere. We now denote by $\mathcal{M}(\mathbf{Z})$ the mesh we obtain by shifting each vertex $\mathbf{x}_v$ by $\mathcal{D}(\mathbf{Z}[v])$. In other words, we now assign to each vertex a *different* latent vector and use it to compute the corresponding translation from the initial sphere to where it should be. To fit our model to data, we look for

$$\mathbf{Z}^* = \arg\min_{\mathbf{Z}} \mathcal{L}_{\text{task}}(\mathcal{M}(\mathbf{Z})) + \lambda_{\text{reg}} L_{\text{reg}}(\mathcal{M}(\mathbf{Z})) + \lambda_{\text{dir}} L_{\text{dir}}(\mathbf{Z}) \ , \tag{3}$$

where $\mathcal{L}_{\text{task}}$ is a task-specific loss function, L_{reg} is the geometric regularization loss of Eq. 2, L_{dir} is a regularization term designed to enforce consistency of the latent vectors across the surface, and λ_{reg} and λ_{dir} are weighting constants. In practice, $\mathcal{L}_{\text{task}}$ can be expressed in terms of Chamfer distance to fit points, slice Chamfer to fit curve annotations, and SDF gradients for segmentation purposes as discussed in Section 4.

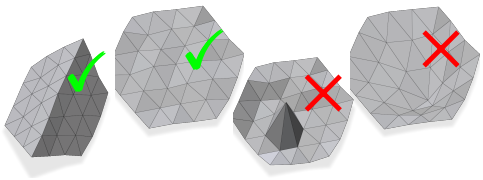


Figure 2: **Behavior of L_{reg}.** Minimizing L_{reg} will leave the regular meshes on the left unchanged, even though the first one exhibits a sharp crease. In contrast, doing so with the meshes on the right, will increase the regularity of the triangles and smooth out the isolated outlier.

Inspired by active surfaces [25, 57, 58, 60], we use Dirichlet energy [43, 54] to define L_{dir}. We write

$$L_{\text{dir}}(\mathbf{Z}) = \text{Tr}(\mathbf{Z}^T \mathbf{L}^p \mathbf{Z}) \ , \tag{4}$$

where $\mathbf{L}$ is the uniform Laplacian matrix and p is an integer power. Note that $\nabla L_{\text{dir}}(\mathbf{Z}) = \mathbf{L}^p \mathbf{Z}$, meaning that a gradient step corresponds to p iterations of Laplacian smoothing of the latent vectors. We found $p = 2$ to work well.

Weight λ_{dir} controls how constrained the fitting is by the prior information contained in the latent space parameterization. When $\lambda_{\text{dir}} \to \infty$, all latent vectors will have the same value and our approach reverts to a global approach with one single latent vector. For small values of λ_{dir}, the model becomes much more flexible as the values of the latent vector can more easily change from vertex to vertex.

One could also allow independant training vectors during training. However, in our experience, this causes the decoder to produce severely self-intersecting meshes as it has too much freedom.

4 Experiments

We demonstrate the benefits of *DALS* on several medical image processing tasks. We train all models to learn a latent representation of livers and spleens using data from the Medical Segmentation Decathlon (MSD) [2] (CC BY-SA 4.0 license). To create ground-truth meshes, we first resampled the annotated images so that their voxel size is $1 \times 1 \times 1$ mm and then used marching cubes [30] to extract isosurfaces. We standardize the surfaces to have zero mean and be contained in the unit sphere.

The datasets contains 111 livers and 41 spleens. We train a model on the first 71 livers and hold out the last 40 for evaluation. We also train another model on the first 31 spleens with the last 10 held out for evaluation. We augment the training data using the recent PointWOLF algorithm [26] to create 100 new shapes for each training shape. PointWOLF applies a smoothly varying non-rigid transformation to mesh vertices and yields diverse and realistic augmentations.

We use 128 dimensional latent vectors and an icosahedron subdivided 3 times for training and 4 times for fitting as a template for the decoder. To learn these vectors and the decoder weights we solve the minimization problem of Eq. 1 with $\lambda_{\text{reg}} = 10^{-4}$ and $\lambda_{\text{n}} = 10^{-3}$. To this end, we use the ADAM optimizer [27]. We set the learning rate to 0.002, the momentum terms to $\beta_1 = 0.9, \beta_2 = 0.999$ and train for 24 hours on a single NVIDIA Tesla V100 GPU (ca. 7,500 epochs). If the loss does not improve for 100 epochs we reduce the learning rate by a factor 2, down to a minimum of 10^{-5}.

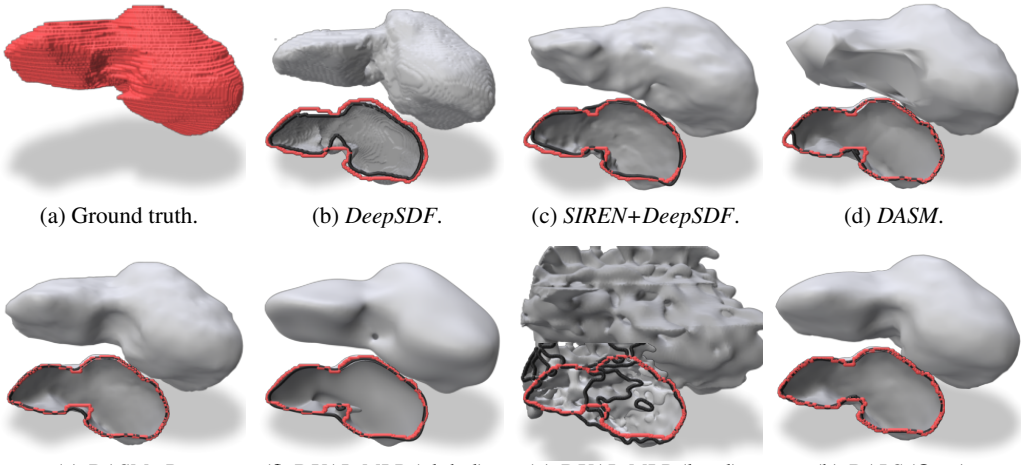
Baselines We compare our model against the following baselines: *DeepSDF* [45], *SIREN+DeepSDF* where the training and inference method is as in *DeepSDF* but with a SIREN [52] based decoder, and the *DUAL-MLP* approach of [38]. These are all auto-decoder based which foregoes the need to train separate encoders for each experiment. We also compare to *DASM*, the active surfaces of [60], along with an improved version that we dub *DASM+R* because it adds a re-meshing step during fitting to avoid self-intersections. These are the only two baselines that do not rely on a learned shape prior and simply promote smoothness.

As the *DUAL-MLP* authors did not release code, we implemented two separate versions of it, one that uses one single latent vector per shape (global) and one that uses several (local). All these methods were trained as recommended in the relevant papers.

4.1 Shape Reconstruction from 3D Point Clouds

Experimental Setup. We test the ability of the latent vector models to reconstruct unknown shapes from a given class, here the liver and the spleen, by randomly and uniformly sampling 2,500 points across the test surface and attempting to reconstruct from them by minimizing the loss of Eq. 3. For *DALS*, *DASM*, and *DASM+R* that use a mesh-based representation, we take $\mathcal{L}_{\text{task}}$ to be the Chamfer distance. For the other methods, we take it to be the mean absolute SDF value at the sample points. For all methods, we use the ADAM optimizer to minimize their fitting losses. For this task, we set $\lambda_{\text{reg}} = 0.001$ and $\lambda_{\text{dir}} = 0.2$. Finally, for *DASM+R* and *DALS* we post process the results using five iterations of Botsch-Kobbelt remeshing [5].

To evaluate the reconstructions, we use the Chamfer distance, the Hausdorff distance, and the F-score [56, 28] at 1% and 2% of the surface's bounding sphere diameter. We also evaluate the mesh quality of the reconstructions using the quality measure of Eq. 2 and the percentage of self-intersecting faces.



(a) Ground truth. (b) *DeepSDF*. (c) *SIREN+DeepSDF*. (d) *DASM*.

(e) *DASM+R*. (f) *DUAL-MLP* (global). (g) *DUAL-MLP* (local). (h) *DALS* (Ours).

Figure 3: **Reconstruction of a previously unseen liver from 2500 3D points.** For each method, we present the full 3D volume and a version of it cut in the middle. The red outline denotes the ground-truth section and the black one that of the reconstructed organ. Note that only ours is smooth while still following closely the ground-truth one.

Results. We report comparative results in Tab. 1. *DALS* consistently outperforms the other approaches, in part because it can model sharp features more accurately, as can be seen in the qualitative

Table 1: **Quantitative results for reconstructing unseen livers from unoriented points.** For each metric we report the mean and standard deviation over the reconstructed shapes. *DALS* concistently produces better reconstructions while still having very good mesh quality.

	Chamfer*↓	Hausdorff↓	F@1%↑	F@2%↑	Quality↑	%self. ints.↓
DeepSDF [45]	40.7 ± 23.7	0.21 ± 0.06	31.3 ± 13.0	63.8 ± 15.6	$\mathbf{0.98 \pm 0.00}$	$\mathbf{0.00 \pm 0.00}$
SIREN+DeepSDF [45, 52]	36.2 ± 34.1	0.20 ± 0.04	51.4 ± 8.68	78.4 ± 9.27	$\mathbf{0.98 \pm 0.00}$	$\mathbf{0.00 \pm 0.00}$
DASM [60]	17.0 ± 10.0	0.23 ± 0.50	87.7 ± 3.25	92.9 ± 2.57	0.74 ± 0.03	7.40 ± 4.57
DASM+R [60]	8.8 ± 7.53	0.19 ± 0.07	94.6 ± 2.57	96.8 ± 1.97	$\mathbf{0.98 \pm 0.00}$	$\mathbf{0.00 \pm 0.00}$
DUAL-MLP (global) [38]	13.6 ± 5.67	0.16 ± 0.05	71.7 ± 6.39	91.4 ± 3.48	$\mathbf{0.98 \pm 0.00}$	$\mathbf{0.00 \pm 0.00}$
DUAL-MLP (local) [38]	161.7 ± 39.6	0.43 ± 0.04	44.5 ± 4.05	62.1 ± 4.08	$\mathbf{0.98 \pm 0.00}$	$\mathbf{0.00 \pm 0.00}$
DALS (Ours)	$\mathbf{2.4 \pm 1.04}$	$\mathbf{0.11 \pm 0.04}$	$\mathbf{95.4 \pm 2.06}$	$\mathbf{99.0 \pm 0.76}$	$\mathbf{0.98 \pm 0.00}$	0.20 ± 0.40

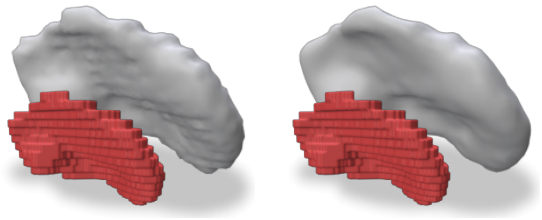
*multiplied with 10,000

results of Fig. 3. Note especially the left side point and the concavity in the lower middle part of the liver. *DALS* also produces excellent mesh quality and keeps the number of intersecting triangles very low although not zero. In future work, we will add an additional loss term to eliminate them.

Table 2: **Quantitative results for reconstructing unseen spleens.** Metrics are reported as in Tab. 1.

	DASM+R [60]	*DALS* (Ours)
Chamfer*↓	$\mathbf{1.6 \pm 0.05}$	2.5 ± 0.89
Hausdorff↓	$\mathbf{0.05 \pm 0.02}$	0.06 ± 0.01
F@1%↑	$\mathbf{97.8 \pm 1.75}$	92.9 ± 2.73
F@2%↑	$\mathbf{100 \pm 0.05}$	99.9 ± 0.11
Quality↑	$\mathbf{0.98 \pm 0.00}$	$\mathbf{0.98 \pm 0.00}$
%self. ints.↓	$\mathbf{0.00 \pm 0.00}$	$\mathbf{0.00 \pm 0.00}$

*multiplied with 10,000



(a) *DASM+R* (b) *DALS* (Ours)

Figure 4: **Reconstruction of previously unseen spleens from 2500 3D points.** The inset shows the ground truth in red.

We repeated the experiment on the much simpler and smoother spleen shapes. We focused on *DALS* and *DASM+R* because they delivered the best results on the liver. As can be seen in Tab. 2 and Fig. 4 *DASM+R* delivers very slightly better metrics but a qualitatively worse reconstruction because it overfits to the staircase artifacts on the ground-truth shape. In contrast, *DALS* yields an organic shape which still fits the data well and can therefore be viewed as a more realistic result. This effect also exists in the liver dataset but did not affect the metrics as obviously because the original images were of higher resolution and the artifacts had much less of an effect.

4.2 Shape Reconstruction from Planar Curve Annotations

Experimental Setup. When annotating medical images, a common time-saving practice is to only annotate three orthogonal 2D slices instead of the entire 3D image. We test the ability of our model to reconstruct shapes from such weak annotations by fitting to the 2D planar boundary curves extracted from the 2D slice annotations. We compute the intersection curves between each held out liver and three orthogonal axis-aligned planes and sample 5,000 points randomly on each curve. As can be seen in Fig. 5(e), this represents a very sparse set of data points and the quality of the shape priors embedded in the models is key to obtaining good results.

For *DALS*, *DASM*, and *DASM+R* we take $\mathcal{L}_{\text{task}}$ to be a modified Chamfer distance that relies on distances within the annotation planes (see supplementary for details). For the other methods, we again use the mean absolute SDF value to compute $\mathcal{L}_{\text{task}}$. We again use the ADAM optimizer and Botsch-Kobbelt remeshing as in the previous section. We set $\lambda_{\text{reg}} = 0.01$ and $\lambda_{\text{dir}} = 100$ as we want to rely heavily on the shape prior in this task.

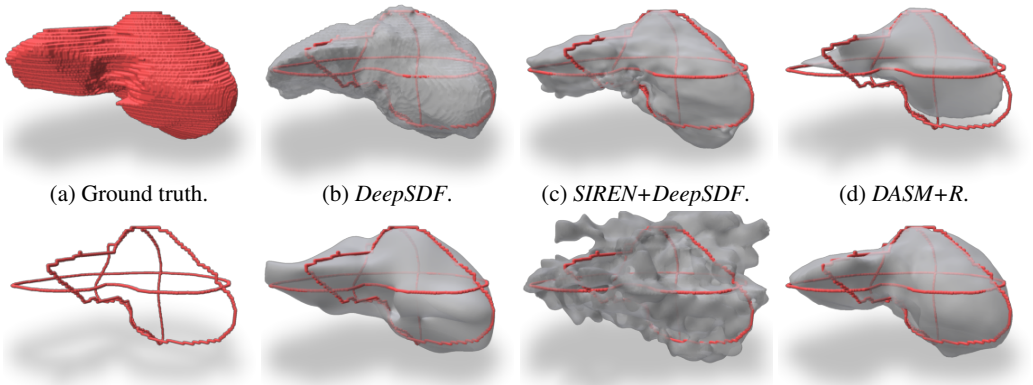
Results. We report comparative results in Tab. 3 and qualitative results in Fig. 5. To generate these results, we only used annotations in the three orthogonal axis-aligned planes. However, we can also annotate additional planes to provide further information. In Fig. 6, we plot the same quality metrics

as in Tab. 3 as a function of the number of annotated planes for one of the livers. Not only are *DASM* results consistently better, but they improve almost monotonically with the number of planes we provide, which is a very desirable behavior in clinical practice.

Table 3: **Quantitative results for reconstructing unseen livers from planar curve annotations.** For each metric (w.r.t. full ground truth) we report the mean and standard deviation over the reconstructed shapes. *DALS* consistently outperforms the baselines while retaining excellent mesh quality.

	Chamfer*↓	Hausdorff↓	F@1%↑	F@2%↑	Quality↑	%self. ints.↓
DeepSDF [45]	4.36 ± 2.35	0.22 ± 0.06	41.5 ± 12.1	69.5 ± 12.2	$\mathbf{0.98 \pm 0.00}$	$\mathbf{0.00 \pm 0.00}$
SIREN+DeepSDF [45, 52]	3.82 ± 2.24	$\mathbf{0.21 \pm 0.05}$	47.0 ± 7.67	74.4 ± 8.25	$\mathbf{0.98 \pm 0.00}$	$\mathbf{0.00 \pm 0.00}$
DASM+R [60]	27.41 ± 7.91	0.47 ± 0.09	14.8 ± 5.99	29.5 ± 9.99	$\mathbf{0.98 \pm 0.00}$	0.02 ± 0.07
DUAL-MLP (global) [38]	4.05 ± 1.58	0.23 ± 0.05	49.9 ± 6.17	74.4 ± 6.08	$\mathbf{0.98 \pm 0.00}$	$\mathbf{0.00 \pm 0.00}$
DUAL-MLP (local) [38]	15.55 ± 4.53	0.39 ± 0.05	28.3 ± 2.87	48.3 ± 4.20	$\mathbf{0.98 \pm 0.00}$	$\mathbf{0.00 \pm 0.00}$
DALS (Ours)	$\mathbf{3.27 \pm 1.48}$	$\mathbf{0.21 \pm 0.05}$	$\mathbf{52.1 \pm 6.56}$	$\mathbf{77.2 \pm 6.87}$	0.99 ± 0.00	$\mathbf{0.00 \pm 0.00}$

*multiplied with 1,000



(a) Ground truth. (b) *DeepSDF*. (c) *SIREN+DeepSDF*. (d) *DASM+R*.

(e) Annotations. (f) *DUAL-MLP* (global). (g) *DUAL-MLP* (local). (h) *DALS* (Ours).

Figure 5: **Reconstruction of a previously unseen liver from outlines in three different planes.** The outlines are shown in the bottom left panel. Again, our reconstruction is smooth while matching the outlines very accurately.

4.3 3D Image Segmentation with Little Training Data

Experimental setup A common medical image analysis task is to segment objects with very few annotations available. Here, we use *DALS* to refine a voxel segmentation produced by a CNN backbone network trained on a few 2D slice annotations. As a backbone, we use the standard U-Net [10] and V-Net [40], along with the more recent nn-U-Net [21] and UNETR [19].

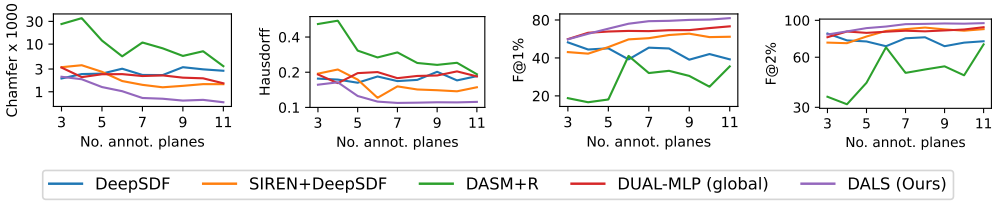
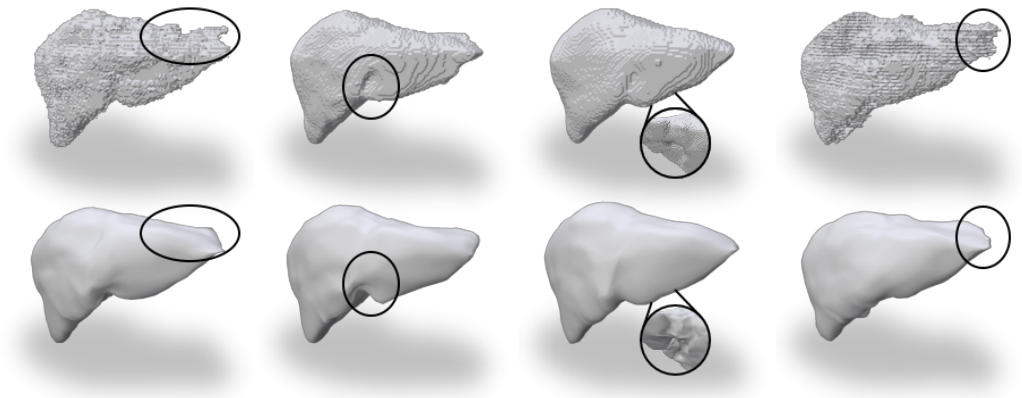


Figure 6: **Reconstruction metrics for a liver as a function of the number of annotated planes.** Unlike those of other approaches, *DALS* results, shown in purple, consistently improve as more planes are added. However, they tend to saturate after 6 or 7.



(a) U-Net. (b) V-Net. (c) nn-U-Net. (d) UNETR.

Figure 7: **Comparison of raw (top) and refined (bottom) segmentations.** The black rings highlight examples of how refinement with *DALS* corrects segmentation mistakes.

For this experiment, we only use the 40 liver images we held off for testing in the previous experiments. We use 20 of them to train the backbone and the other 20 for testing purposes. This simulates a realistic scenario in which we have few training images to train the segmentation network and they have *not* been used to learn the shape priors.

Table 4: **Quantitative results for segmentation refinement.** For each metric we report the mean and standard deviation over the 20 reconstructed shapes. Refinement with *DALS* consistently improves the segmentations for all backbones and metrics.

	Dice ↑		Hausdorff ↓		Chamfer×10k ↓	
	Raw	w/ DALS	Raw	w/ DALS	Raw	w/ DALS
U-Net [10]	0.81 ± 0.08	$\mathbf{0.83 \pm 0.08}$	26.6 ± 10.2	$\mathbf{20.9 \pm 6.32}$	44.9 ± 51.0	$\mathbf{26.0 \pm 22.4}$
V-Net [40]	0.79 ± 0.16	$\mathbf{0.80 \pm 0.16}$	28.5 ± 12.5	$\mathbf{25.4 \pm 9.29}$	56.4 ± 74.8	$\mathbf{43.3 \pm 54.8}$
nn-U-Net [21]	0.84 ± 0.09	$\mathbf{0.85 \pm 0.07}$	25.2 ± 11.3	$\mathbf{19.5 \pm 8.08}$	38.7 ± 48.4	$\mathbf{22.3 \pm 20.4}$
UNETR [19]	0.74 ± 0.13	$\mathbf{0.75 \pm 0.17}$	39.5 ± 24.2	$\mathbf{26.4 \pm 12.5}$	164.5 ± 266	$\mathbf{52.7 \pm 57.0}$

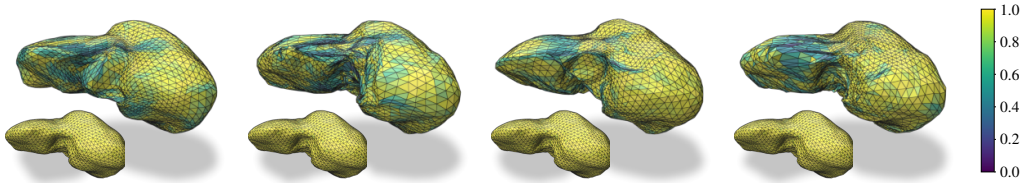
Results. As the models are not sufficiently well trained — a common occurrence in medical imaging — the 'raw' segmentations reported in Tab. 4 are not particularly good. To refine them by enforcing shape priors, we can treat them as noisy data to which we fit a *DALS* model. To this end, we initialize the shape at the center and scale predicted by the raw segmentation. We then fit *DALS* to an unsigned distance function computed from the segmentation binary image (full details in supplementary). We use $\lambda_{\text{reg}} = 0$, $\lambda_{\text{dir}} = \infty$, and no remeshing to heavily rely on the model's learned prior.

As can be seen in Fig. 7 and Tab. 4, this yields much improved segmentations both in terms of visual appearance and quantitative metrics. Note that *DALS* removes spurious growths *and* recovers concavities that were missed in the raw segmentations. In other words, *DALS* does not simply smooth. It really enforces geometric priors.

4.4 Ablation experiments

We perform an ablation study to investigate how using a single or multiple latent vectors and our triangle quality loss L_{reg} affect performance. Additional ablations can be found in the supplementary materials. To use a single vector, we constrain all latent vectors to be the same during fitting. To remove L_{reg} we set λ_{reg} to 0 during training and fitting.

As shown in Tab. 5, both our local latent vector approach and L_{reg} loss significantly improves reconstruction and mesh quality, even more so when combined. This is also apparent in Fig. 8 as the reconstructions with L_{reg} are smoother and fewer triangles are severely distorted. Finally, adding remeshing results in excellent mesh quality at some cost to accuracy, as Botsch-Kobbelt also optimizes vertex positions. In this work, we prioritized mesh quality for our reconstruction results.



(a) Local and $\lambda_{\text{reg}} = 0$. (b) Global and $\lambda_{\text{reg}} = 0$. (c) Local and $\lambda_{\text{reg}} \neq 0$. (d) Global and $\lambda_{\text{reg}} \neq 0$.

Figure 8: **Reconstructions of the ablated models of Tab. 5**. Facets are colored according to their quality in terms of the measure of Eq. (2): yellow is high and blue is low. The insets show the reconstruction by *DALS* including re-meshing. Global/local refers to single/multiple latent vectors.

Table 5: **Quantitative results of the ablation study.** Local inference results in a large boost to reconstruction accuracy and the L_{reg} loss significantly improves triangle quality. Adding remeshing further boosts the triangle quality at some expense to reconstruction accuracy.

Local	L_{reg}	Chamfer*↓	Quality↑
		7.89 ± 2.75	0.75 ± 0.03
✓		1.89 ± 0.63	0.72 ± 0.02
	✓	5.41 ± 1.77	0.83 ± 0.02

Local	L_{reg}	Remeshing	Chamfer*↓	Quality↑
✓	✓		1.50 ± 0.49	0.87 ± 0.01
✓	✓	✓	2.41 ± 1.04	0.98 ± 0.00

5 Conclusion

We have shown that we train a latent vector model to represent a complex surface by a triangulated sphere and deformations at each one of its vertices that are the output of a decoder that takes as input the vertex coordinates and a latent vector. At training time, we use an auto-decoder approach to learn a single latent vector per training shape. However, at model-fitting time, we allow the latent vectors to be different at each vertex but we enforce consistency of these vectors across the triangulation. This enables us to learn the model from a relatively small training set while giving the necessary flexibility to model complex 3D shapes without over-smoothing. A key ingredient is that we impose regularization constraints on the latent vectors but *not* on the vertex 3D locations.

In this work, we have focused on organic shapes represented as watertight surfaces and demonstrated the effectiveness of our approach on liver and spleen reconstruction. In future work, we will extend our approach to non-watertight surfaces by dynamically updating the template mesh, that is, by removing faces that are predicted to be unused as in [44]. We will also integrate re-meshing into *DALS* as we did for *DASM+R* to further improve mesh quality.

Finally, as the purpose of our model is to incorporate prior knowledge into medical image processing tasks, it is important to mention that our model may introduce unwanted biases if the training data is heavily skewed towards certain genders, ages, or other groups. The local nature of our model makes this bias less direct, but representative training data is still a must.

References

[1] Panos Achlioptas et al. “Learning representations and generative models for 3d point clouds”. In: *Proceedings of the International Conference on Machine Learning (ICML)*. 2018, pp. 40–49.

[2] Michela Antonelli et al. “The medical segmentation decathlon”. In: *arXiv preprint arXiv:2106.05735* (2021).

[3] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. “Layer normalization”. In: *arXiv preprint arXiv:1607.06450* (2016).

[4] R. P. Bhatia and Kent L. Lawrence. “Two-dimensional finite element mesh generation based on stripwise automatic triangulation”. In: *Computers & Structures* 36.2 (1990), pp. 309–319.

[5] Mario Botsch and Leif Kobbelt. “A remeshing approach to multiresolution modeling”. In: *Proceedings of Eurographics*. 2004, pp. 185–192.

[6] Andrew Brock et al. “Generative and discriminative voxel modeling with convolutional neural networks”. In: *arXiv preprint arXiv:1608.04236* (2016).

[7] Rohan Chabra et al. “Deep local shapes: Learning local sdf priors for detailed 3d reconstruction”. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2020, pp. 608–625.

[8] Zhiqin Chen and Hao Zhang. “Learning implicit fields for generative shape modeling”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019, pp. 5939–5948.

[9] Julian Chibane, Thiemo Alldieck, and Gerard Pons-Moll. “Implicit functions in feature space for 3d shape reconstruction and completion”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020, pp. 6970–6981.

[10] Özgün Çiçek et al. “3D U-Net: learning dense volumetric segmentation from sparse annotation”. In: *Proceedings of the International Conference on Medical Image Computing and Computer Assisted Intervention (MICCAI)*. Springer. 2016, pp. 424–432.

[11] Luca Cosmo et al. “LIMP: Learning latent shape representations with metric preservation priors”. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2020, pp. 19–35.

[12] Angela Dai, Charles Ruizhongtai Qi, and Matthias Nießner. “Shape completion using 3d-encoder-predictor cnns and shape synthesis”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 5868–5877.

[13] P. Fua. “Model-Based Optimization: Accurate and Consistent Site Modeling”. In: *Congress of International Society for Photogrammetry and Remote Sensing*. July 1996.

[14] P. Fua and Y. G. Leclerc. “Object-Centered Surface Reconstruction: Combining Multi-Image Stereo and Shading”. In: *International Journal of Computer Vision (IJCV)* 16 (Sept. 1995), pp. 35–56.

[15] Jun Gao et al. “Learning deformable tetrahedral meshes for 3d reconstruction”. In: *Neural Information Processing Systems (NeurIPS)* 33 (2020), pp. 9936–9947.

[16] Thibault Groueix et al. “A papier-mâché approach to learning 3d surface generation”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018, pp. 216–224.

[17] Rana Hanocka et al. “Point2Mesh: a self-prior for deformable meshes”. In: *ACM Transactions on Graphics (TOG)* 39.4 (2020), pp. 126–1.

[18] A. Hatamizadeh, D. Sengupta, and D. Terzopoulos. “End-To-End Trainable Deep Active Contour Models for Automated Image Segmentation: Delineating Buildings in Aerial Imagery”. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2020.

[19] Ali Hatamizadeh et al. “UNETR: Transformers for 3d medical image segmentation”. In: *Proceedings of the IEEE Winter Conference on Applications of Computer Vision (WACV)*. 2022, pp. 574–584.

[20] Lei He et al. “A comparative study of deformable contour methods on medical image segmentation”. In: *Image and Vision Computing (IVC)* 26.2 (2008), pp. 141–163.

[21] Fabian Isensee et al. “nnU-Net: a self-configuring method for deep learning-based biomedical image segmentation”. In: *Nature methods* 18.2 (2021), pp. 203–211.

[22] Chiyu Jiang et al. “Local implicit grid representations for 3d scenes”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020, pp. 6001–6010.

[23] Anne Jorstad et al. “NeuroMorph: a toolset for the morphometric analysis and visualization of 3D models derived from electron microscopy image stacks”. In: *Neuroinformatics* 13.1 (2015), pp. 83–92.

[24] Kristine Aavild Juhl et al. “Implicit Neural Distance Representation for Unsupervised and Supervised Classification of Complex Anatomies”. In: *Proceedings of the International Conference on Medical Image Computing and Computer Assisted Intervention (MICCAI)*. 2021, pp. 405–415.

[25] Michael Kass, Andrew Witkin, and Demetri Terzopoulos. “Snakes: Active contour models”. In: *International Journal of Computer Vision (IJCV)* 1.4 (1988), pp. 321–331.

[26] Sihyeon Kim et al. “Point Cloud Augmentation with Weighted Local Transformations”. In: *Proceedings of the International Conference on Computer Vision (ICCV)*. 2021, pp. 548–557.

[27] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *Proceedings of the International Conference on Learning Representations (ICLR)*. 2015.

[28] Arno Knapitsch et al. “Tanks and temples: Benchmarking large-scale scene reconstruction”. In: *ACM Transactions on Graphics (TOG)* 36.4 (2017), pp. 1–13.

[29] M. E. Leventon, W. E. Grimson, and O. Faugeras. “Statistical Shape Influence in Geodesic Active Contours”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2000, pp. 316–323.

[30] Thomas Lewiner et al. “Efficient implementation of marching cubes’ cases with topological guarantees”. In: 8 (2003), pp. 1–15.

[31] Justin Liang et al. “Polytransform: Deep polygon transformer for instance segmentation”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020, pp. 9131–9140.

[32] Huan Ling et al. “Fast interactive object annotation with curve-gcn”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019, pp. 5257–5266.

[33] Or Litany et al. “Deformable shape completion with graph convolutional autoencoders”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018, pp. 1886–1895.

[34] Lingjie Liu et al. “Neural sparse voxel fields”. In: *Neural Information Processing Systems (NeurIPS)*. 2020, pp. 15651–15663.

[35] Diego Marcos et al. “Learning deep structured active contours end-to-end”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018, pp. 8877–8885.

[36] Julien N.P. Martel et al. “ACORN: Adaptive Coordinate Networks for Neural Representation”. In: *ACM Transactions on Graphics (TOG)* (2021).

[37] T. Mcinerney and D. Terzopoulos. “A Dynamic Finite Element Surface Model for Segmentation and Tracking in Multidimensional Medical Images with Application to Cardiac 4D Image Analysis”. In: *Computerized Medical Imaging and Graphics* 19.1 (1995), pp. 69–83.

[38] Ishit Mehta et al. “Modulated periodic activations for generalizable local functional representations”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2021, pp. 14214–14223.

[39] Lars Mescheder et al. “Occupancy networks: Learning 3d reconstruction in function space”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019, pp. 4460–4470.

[40] Fausto Milletari, Nassir Navab, and Seyed-Ahmad Ahmadi. “V-net: Fully convolutional neural networks for volumetric medical image segmentation”. In: *Proceedings of the International Conference on 3D Vision (3DV)*. IEEE. 2016, pp. 565–571.

[41] Luca Morreale et al. “Neural Convolutional Surfaces”. In: *arXiv preprint arXiv:2204.02289* (2022).

[42] Thomas Müller et al. “Instant Neural Graphics Primitives with a Multiresolution Hash Encoding”. In: *arXiv preprint arXiv:2201.05989* (2022).

[43] Baptiste Nicolet, Alec Jacobson, and Wenzel Jakob. “Large steps in inverse rendering of geometry”. In: *ACM Transactions on Graphics (TOG)* 40.6 (2021), pp. 1–13.

[44] Junyi Pan et al. “Deep mesh reconstruction from single rgb images via topology modification networks”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019, pp. 9964–9973.

[45] Jeong Joon Park et al. “Deepsdf: Learning continuous signed distance functions for shape representation”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019, pp. 165–174.

[46] Sida Peng et al. “Deep snake for real-time instance segmentation”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020, pp. 8533–8542.

[47] Songyou Peng et al. “Convolutional occupancy networks”. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2020, pp. 523–540.

[48] Songyou Peng et al. “Shape As Points: A Differentiable Poisson Solver”. In: *Neural Information Processing Systems (NeurIPS)*. 2021.

[49] R. Prevost et al. “Incorporating Shape Variability in Image Segmentation via Implicit Template Deformation”. In: *Proceedings of the International Conference on Medical Image Computing and Computer Assisted Intervention (MICCAI)* (2013), pp. 82–89.

[50] Tianchang Shen et al. “Deep Marching Tetrahedra: a Hybrid Representation for High-Resolution 3D Shape Synthesis”. In: *Neural Information Processing Systems (NeurIPS)*. 2021.

[51] Jonathan Shewchuk. “What is a good linear finite element? interpolation, conditioning, anisotropy, and quality measures (preprint)”. In: *University of California at Berkeley* (2002).

[52] Vincent Sitzmann et al. “Implicit neural representations with periodic activation functions”. In: *Neural Information Processing Systems (NeurIPS)*. Vol. 33. 2020, pp. 7462–7473.

[53] Edward J. Smith et al. “GEOMetrics: Exploiting Geometric Structure for Graph-Encoded Objects”. In: *Proceedings of the International Conference on Machine Learning (ICML)*. 2019, pp. 5866–5876.

[54] Justin Solomon, Keenan Crane, and Etienne Vouga. “Laplace-Beltrami: The Swiss army knife of geometry processing”. In: *Symposium on Geometry Processing Graduate School (Cardiff, UK, 2014)*. Vol. 2. 2014.

[55] Towaki Takikawa et al. “Neural Geometric Level of Detail: Real-time Rendering with Implicit 3D Shapes”. In: (2021).

[56] Maxim Tatarchenko et al. “What do single-view 3d reconstruction networks learn?” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019, pp. 3405–3414.

[57] Demetri Terzopoulos, Andrew Witkin, and Michael Kass. “Constraints on deformable models: Recovering 3D shape and nonrigid motion”. In: *Artificial Intelligence* 36.1 (1988), pp. 91–123.

[58] Demetri Terzopoulos, Andrew Witkin, and Michael Kass. “Symmetry-seeking models and 3D object reconstruction”. In: *International Journal of Computer Vision (IJCV)* 1.3 (1988), pp. 211–221.

[59] Giovanni Trappolini et al. “Shape registration in the time of transformers”. In: *Neural Information Processing Systems (NeurIPS)* 34 (2021), pp. 5731–5744.

[60] Udaranga Wickramasinghe, Pascal Fua, and Graham Knott. “Deep active surface models”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2021, pp. 11652–11661.

[61] Francis Williams et al. *Neural Fields as Learnable Kernels for 3D Reconstruction*. 2021. arXiv: 2111.13674 [cs.CV].

We provide additional details on how we fit our models to planar curves and to voxel segmentations, as described in Sections 4.2 and 4.3 respectively. Finally, we also provide additional experiments.

A Fitting *DALS* to Planar Curve Annotations

When annotations are only provided in 2D planes, we only wish to evaluate the reconstruction in these planes. This is similar to how plane annotations are handled for 3D voxel segmentations [10].

Formally, assume we are given a plane $\mathcal{P}$ and a triangle mesh $\mathcal{M}$. To differentially sample points on the intersection between $\mathcal{P}$ and $\mathcal{M}$ we first find the intersection between the plane and each triangle facet. The intersection of a plane and triangle is either empty or a line segment spanned by two points $\mathbf{x}_1$ and $\mathbf{x}_2$. We ignore the degenerate cases where the intersection is the entire triangle or only one of its vertices. We can then sample a point $\mathbf{p}$ on the intersecting line segment as $\mathbf{p} = r\mathbf{x}_1 + (1-r)\mathbf{x}_2$, where $r \in U(0,1)$. Let $(\alpha_1, \beta_1, \gamma_1)$ and $(\alpha_2, \beta_2, \gamma_2)$ be the barycentric coordinates of, respectively, $\mathbf{x}_1$ and $\mathbf{x}_2$. We can then write $\mathbf{p}$ in terms of the triangle vertices $\mathbf{u}$, $\mathbf{v}$, and $\mathbf{w}$ as

$$\mathbf{p} = \begin{bmatrix}\mathbf{u} & \mathbf{v} & \mathbf{w}\end{bmatrix} \begin{bmatrix}\alpha_1 & \alpha_2 \\ \beta_1 & \beta_2 \\ \gamma_1 & \gamma_2\end{bmatrix} \begin{bmatrix} r \\ 1-r \end{bmatrix} . \tag{5}$$

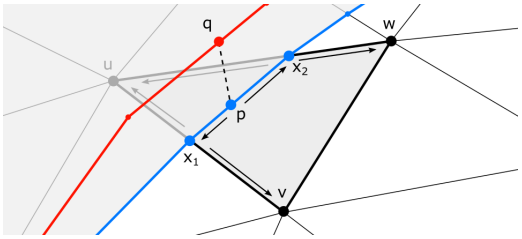


Figure 9: **Gradient propagation for samples on the intersection between a triangle mesh and a plane.** The blue curve is the intersection of the mesh and the plane, and the red curve is the ground truth boundary curve. The highlighted triangle has vertices $\mathbf{u}$, $\mathbf{v}$, and $\mathbf{w}$ and intersects the plane in the line segment spanned by $\mathbf{x}_1$ and $\mathbf{x}_2$.

As a result, $\mathbf{p}$ is a linear combination of the triangle vertices and r is an independent stochastic term. Therefore, we can propagate a gradient from the point $\mathbf{p}$ to the triangle vertices $\mathbf{u}$, $\mathbf{v}$, and $\mathbf{w}$ [53], see Fig. 9.

Now, let $\mathcal{S}_\mathcal{P}(\mathcal{M})$ denote a set of M points sampled differentially on the intersection of $\mathcal{M}$ and $\mathcal{P}$. Further, let $\mathcal{T}_\mathcal{P}$ be a set of N points sampled uniformly on the planar curve annotations for plane $\mathcal{P}$. In this work we use $M = 5000$ sample points. The loss for plane $\mathcal{P}$ is then

$$L_{\text{cf}}(\mathcal{S}_\mathcal{P}(\mathcal{M}), \mathcal{T}_\mathcal{P}) = \frac{1}{M} \sum_{\mathbf{q}\in\mathcal{T}_\mathcal{P}} \min_{\mathbf{p}\in\mathcal{S}_\mathcal{P}(\mathcal{M})} \|\mathbf{p}-\mathbf{q}\|_2^2 + \frac{1}{M} \sum_{\mathbf{p}\in\mathcal{S}_\mathcal{P}(\mathcal{M})} \min_{\mathbf{q}\in\mathcal{T}_\mathcal{P}} \|\mathbf{q}-\mathbf{p}\|_2^2 . \tag{6}$$

Note that the above is the Chamfer loss between the plane sample points [53].

Finally, given a collection of planes $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_P$, the fitting loss, $\mathcal{L}_{\text{task}}$, is

$$\mathcal{L}_{\text{task}}(\mathcal{M}) = \frac{1}{P} \sum_{i=1}^{P} L_{\text{cf}}(\mathcal{S}_{\mathcal{P}_i}(\mathcal{M}), \mathcal{T}_{\mathcal{P}_i}) . \tag{7}$$

B Fitting *DALS* to Voxel Segmentations

To fit *DALS* to a binary 3D voxel image $\mathbf{B} \in \mathbb{R}^{W\times H\times D}$ we first use the Euclidean distance transform to create a new image $\mathbf{U} \in \mathbb{R}^{W\times H\times D}$ where each voxel contains the unsigned distance to the segmentation boundary. Given a mesh $\mathcal{M}$ with V vertices, the fitting loss is then given by

$$\mathcal{L}_{\text{task}}(\mathcal{M}) = \frac{1}{V} \sum_{v=1}^{V} \mathbf{U}(\mathbf{x}_v) , \tag{8}$$

where $\mathbf{U}(\mathbf{x}_v)$ is trilinear interpolation of $\mathbf{U}$ at vertex position $\mathbf{x}_v$.

To optimize the latent vectors $\mathbf{Z} \in \mathbb{R}^{V\times d}$ we require the gradient of $\mathbf{U}$ at $\mathbf{x}_v$. To get a robust estimate, we use a Sobel operator to pre-compute a gradient image $\mathbf{G} \in \mathbb{R}^{W\times H\times D\times 3}$ which contains the gradient of $\mathbf{U}$ at each voxel position. We then use trilinear interpolation to evaluate the gradient of $\mathbf{U}$ as $\nabla\mathbf{U}(\mathbf{x}_v) = \mathbf{G}(\mathbf{x}_v)$.

We also attempted to fit *DALS* directly to the binary segmentation or the softmax outputs of the CNN backbone. In practice, we found that the distance field gradients made it easier for the model to fit the images.

C Additional experiments

To further explore the properties of our model, we perform additional ablation experiments.

Local latent vectors We investigate whether *DALS* actually uses multiple *different* latent vectors during fitting. We use *DALS* to reconstruct a liver from unoriented points as in section 4.1 and illustrate the result in Fig. 10. The livers corresponding to the latent vectors at the four highlighted vertices are noticeably different and the coloring of the reconstruction clearly demonstrates the smooth transition between latent vectors over the surface.

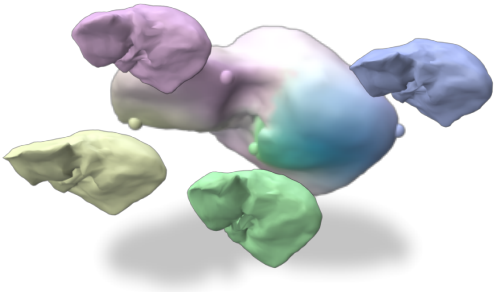


Figure 10: *DALS* **combining multiple shapes.** The large liver is colored according to the latent vector at each vertex. For the highlighted vertices, we show the corresponding decoded livers.

Reconstruction time We compare the reconstruction time for *DALS* against its closest competitors for shape reconstruction: *DUAL-MLP* and *DASM+R*. We use the same setup as in section 4.1 and vary the number of iterations used for fitting. Since each model need a different number of iterations, we only report the time used. Quantitative results are shown in Fig. 11 and *DALS* consistently provides better reconstructions at faster times than the compared baselines.

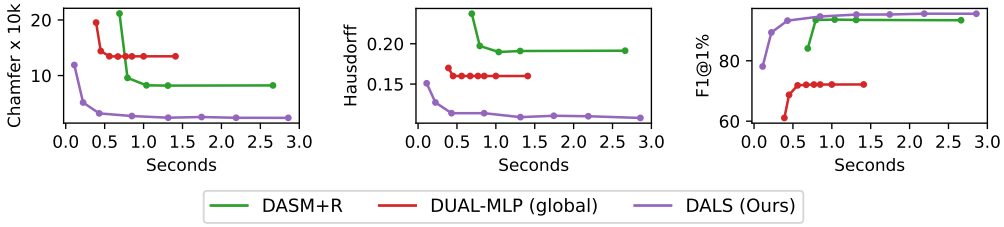


Figure 11: **Metrics over time for reconstructing unseen livers from 2500 unoriented points**. Each point shows the mean metric over all shapes at the given time. *DALS*, shown in purple, is consistently better than the compared methods.

For the results in the paper, the number of fitting iterations was set high enough to ensure convergence. Specifically, for section 4.1, *DASM+R* used 2.9 seconds, *DUAL-MLP* used 1.3 seconds, and *DALS* used 2.4 seconds.

Template mesh resolution We investigate the effect of the template mesh resolution. Again, we use the same setup as in section 4.1 and vary the resolution of the template. Specifically, we use a subdivided icosahedron subdivided $n = 2, \ldots, 6$ times. Note that we use the same model for all n, which was trained using an icosahedron subdivided 3 times — it is only the template used for fitting that varies. Quantitative results are shown in Fig. 12. Increasing the mesh resolution improves the reconstruction metrics at the cost of higher fitting time and worse mesh quality. We used $n = 4$ subdivision for the experiments in this paper, as we find it provides the best trade-off between reconstruction quality, time, and mesh quality.

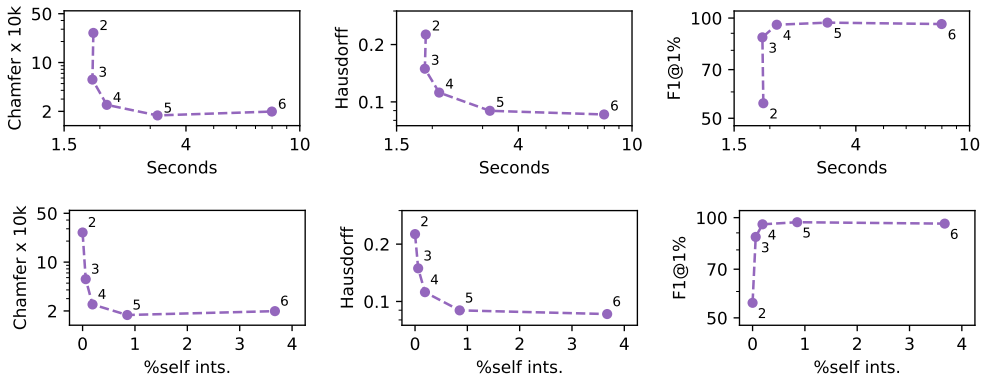


Figure 12: **Metrics for reconstructing unseen livers for varying mesh resolution.** Reconstructions are based on 2500 unoriented points. Each point shows the mean metric over all shapes using an icosahedron subdivided n times as a template, where n is indicted next to each point. The top row shows metrics over time and the bottom row shows metrics over self intersections.

5 Conclusion

This thesis studied the problem of segmenting large 3D images where little to no labeled data is available by incorporating prior information on shape and topology. The methodological focus of the thesis was on methods for fitting surface meshes to image data, where the meshes were used to incorporate topological and geometric priors. Another significant focus of the thesis was on the computational aspects of these methods, in order to make them usable for large segmentation problems. In this chapter, I begin with summarizing the most important findings from each contribution and then provide general concluding statements on the thesis as a whole.

5.1 Summary of Contributions

In Paper A, we reviewed and benchmarked the current state of the art of serial and parallel minimum cut/maximum flow (min-cut/max-flow) algorithms. Our benchmarks were performed over a new dataset with a wide selection of benchmark problems from computer vision. Our main finding was the disappointing performance of parallel algorithms. Except for very large problems, they scaled poorly with added computational resources and were often beaten by a strong serial algorithm. We also found that the best-performing algorithm could be predicted from graph statistics. Our findings show that, while the field of min-cut/max-flow is very mature, there is still room to squeeze out yet more performance.

In Paper B, we developed a parallel algorithm for quadratic pseudo-Boolean optimization (QPBO). When tested on large problems, it provided a significant speed-up with good memory efficiency. When tested on smaller problems, our parallel algorithm, like those in Paper A, provided less of a speed-up but was faster than a serial approach in the majority of cases. As a result, our algorithm significantly increases the scale of QPBO problems that can be practically solved.

In Paper C, we developed a new mesh-based method for multi-object segmentation that allows for exclusion constraints between objects. The meshes are fitted to the image by solving a combined QPBO problem. Our method increased the segmentation accuracy for three multi-object segmentation problems and could segment thousands of objects in a few minutes. Using the method from Paper B, this runtime could be reduced even further.

In Paper D, we developed a new method for simultaneously tracking and segmenting evolving objects in 4D (3D + time) images. Our method is designed for objects that either split or merge over time, but not both, which constrains the 4D volume traced by the objects to be simply connected. We exploited this by extending deformable surfaces to 4D such that we can fit the boundary of this simply connected component. Since we only model the boundary, our method is computationally efficient and has a low memory use. Hence, our method allows the segmentation of images that would be highly impractical with other 4D methods. Furthermore, by explicitly modeling the topology, our method has increased robustness and allows detecting the precise time and location of splits/merges as saddle points in the fitted mesh.

In Paper E, we developed a method to train a 3D convolutional neural network (CNN) from boundary point annotations. A user inputs the boundary points in a graphical user interface (GUI) and a mesh is then fitted to these. The CNN is then supervised by the fitted meshes during training. Compared to existing approaches, ours achieved

higher accuracy for less annotation effort. As a result, our work reduced the barrier to training CNNs in cases where little to no training data is available. Additionally, since our annotation approach is based on mesh fitting, the effort required does not increase for higher-resolution images.

In Paper F, we developed a new mesh-based shape model which learns a shape prior that can easily be coupled with other methods for further downstream tasks. We represent shapes as a mesh with a separate latent vector at each vertex. By allowing the latent vectors to differ smoothly over the surface, the model can blend different shapes together. This gives our model great flexibility. Our first two experiments showed that our model was effective at completing shapes from points or planar curve annotations. Therefore, combining the model with Paper E could further reduce the annotation effort required to train CNNs. Finally, we also showed that the model could be used to correct noisy image segmentations. This is important because a CNN trained on a few labels will often have a noisy output.

5.2 General Conclusion

We consider the work in this thesis according to the data properties listed in Section 1.1.

P1 They are large (1000^3 voxels and beyond).
We have shown that methods based on mesh fitting with min-cut/max-flow can be scaled to very large segmentation problems. The same applies to the 4D deformable model in Paper D. Common for these methods is that their computational cost does not depend on the size of the image but on the size of the mesh — which, again, is related to the complexity of the segmentation target. Consequently, these techniques are very valuable when dealing with large multi-GB images.

The method from Paper E on its own is less suited for larger volumes since it still relies on a CNN. However, the developed method is still relevant, as it reduces the annotation burden for large images — something that is often a significant bottleneck. Furthermore, the model from Paper F could potentially be used to upscale outputs, similar to the post-processing case in Paper F.

P2 No labeled image data exists.
The min-cut/max-flow methods and deformable model from Chapter 3 do not rely on learned appearance information, and therefore do not need any labeled data. However, they are restricted to images where the voxel intensities are sufficient to distinguish between object and non-object regions. For this case, we have shown the developed methods to work very well. With images from materials science, such cases are common but less so in biomedical imaging. Here, the methods from Chapter 3 must be coupled with a more advanced appearance model, e.g., a CNN, where labeled image data is necessary.

To this end, the learning-based methods from Chapter 4 have been shown to reduce the amount of annotation effort needed to train segmentation models for fixed topology objects. Hence, in the cases where intensity does not suffice to distinguish objects, these methods may be used to train an appearance model that can then be used with the methods from Chapter 3 or on its own.

P3 We have prior knowledge about object shapes.
All methods in this thesis made use of meshes in a way that constrains object topology. In Paper D, we demonstrate how restricting the 4D topology leads to a powerful segmentation approach. In Paper E, restricting the topology enables training from sparse point annotations. Both of these methods also made use of a low-level smoothness prior

for regularization. However, especially Paper E could also make use of a more advanced shape prior. As shown in Paper F, meshes are very effective for modeling shapes of fixed topology and our model showed advantages over other methods when fitting to sparse data.

The min-cut/max-flow methods in Chapter 3 make use of the simple shape prior that the fitted shape should not deviate significantly from an initial shape. In this thesis, these shapes have been rather simple — spheres and cylinders — and we have shown these to be highly effective when segmenting simpler objects. For more complex geometries, one could use, e.g., a learned atlas as the initial shape. However, capturing more complex shape variation is not possible in the studied framework. As a result, these methods are less suitable for segmenting objects whose range of possible shapes varies a lot from a given template. In such a case, it is likely more valuable to use an approach like in Paper E.

All in all, we have shown meshes to be a powerful tool for 3D image segmentation due to the strong priors they can bring and their computational efficiency. As the size of 3D images continues to grow, meshes should therefore remain in our minds as an effective way to escape the ever-larger image grid.

6 Perspectives

This chapter outlines directions for future work and discusses broader perspectives for the work in this thesis. Proceeding in the order of the contributions, we start with segmentation based on minimum cut/maximum flow (min-cut/max-flow). As we pointed out in Paper A, there is ample room for improvement concerning parallel algorithms for min-cut/max-flow in computer vision. Current parallel algorithms are based on augmenting paths or preflow push-relabel, while pseudoflow algorithms were demonstrated as faster. In Paper B, as part of an ablation experiment, we attempted to insert a pseudoflow algorithm into an existing parallel algorithm. However. we found that it did not improve performance. This suggests that incorporating pseudoflow may be more than a simple engineering challenge. Nevertheless, finding ways to parallelize pseudoflow algorithms seems like an important direction with great potential.

Another important direction is the tighter coupling of min-cut/max-flow with deep learning methods. Here, an important part is to make min-cut/max-flow differentiable such that gradient information can be propagated through it and back to other learnable parameters. For voxel-wise segmentation, this has already been explored in [1]. Other works [2, 3] have also formulated ways to propagate gradients through more general optimization problems. To my mind, exploring this direction further, and combining it with the mesh fitting frameworks used in this thesis, has the potential to create robust and easy-to-train segmentation models. For example, one could use min-cut/max-flow to fit a mesh-like in Paper C but based on neural network predictions. That mesh could then be compared with user point annotations like in Paper E for an end-to-end training pipeline.

Moving on to shape priors, there are also several ways to continue. One immediate avenue is to build on active contours. Current work has already coupled active contours with deep learning [4, 5] to learn external forces. A shape model like in Paper F could then be used to regularize the surface instead of the common low-level smoothness prior. Another important direction is local shape priors. Paper F and several other works [6–11] have shown that modelling shapes locally makes models much more flexible and allows for smaller faster models. This makes them attractive for limited data and/or interactive scenarios like in Paper E, and could therefore be a promising research direction.

Expanding to a broader context, we should also consider whether meshes will remain useful for image segmentation. While they have their strengths, their inflexibility limits their application areas. Hence, other methods could eclipse them over time. In this context, it is helpful to distinguish between two use cases: 0%-90% and 90%-100%. The first case represents starting from scratch with a set of images and going to a 90% satisfying segmentation. Here, I think meshes will remain an extremely useful tool precisely because of the constraints they impose on the segmentation. For the second case, the long-run success of mesh-based methods is likely determined by one factor: Whether we manage to meaningfully incorporate them with learning-based methods in a scalable way.

Finally, we consider broader societal aspects. This thesis deals with incorporating prior information into segmentation models. This is good, in the sense that it biases the models towards correct segmentations. However, it can also have unintended consequences — especially when the prior is learned from data. Studies have shown that biased datasets can lead to models with worse performance on some target groups [12–14]. Since the models in this thesis impose biases in a very direct way, it is essential to understand

this effect when using them for real applications. Both when it comes to racial and gender biases, but also more abstractly in the form of healthy/sick tissue, cell types, or normal/defective materials. Otherwise, we risk creating segmentation models that may appear to help us, but only because they always show us what we expect to find anyway.

References

[1] Hui Xie. "Model-Based Deep Learning for Medical Image Segmentation with Global Optimality". PhD thesis. The University of Iowa, 2022.

[2] Brandon Amos and J Zico Kolter. "Optnet: Differentiable optimization as a layer in neural networks". In: *Proceedings of the International Conference on Machine Learning (ICML)*. 2017, pp. 136–145.

[3] Akshay Agrawal, Brandon Amos, Shane Barratt, Stephen Boyd, Steven Diamond, and J Zico Kolter. "Differentiable convex optimization layers". In: *Neural Information Processing Systems (NeurIPS)* 32 (2019).

[4] Diego Marcos, Devis Tuia, Benjamin Kellenberger, Lisa Zhang, Min Bai, Renjie Liao, and Raquel Urtasun. "Learning deep structured active contours end-to-end". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018, pp. 8877–8885.

[5] Udaranga Wickramasinghe, Pascal Fua, and Graham Knott. "Deep active surface models". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2021, pp. 11652–11661.

[6] Julian Chibane, Thiemo Alldieck, and Gerard Pons-Moll. "Implicit functions in feature space for 3d shape reconstruction and completion". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020, pp. 6970–6981.

[7] Songyou Peng, Michael Niemeyer, Lars Mescheder, Marc Pollefeys, and Andreas Geiger. "Convolutional occupancy networks". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2020, pp. 523–540.

[8] Rohan Chabra, Jan E Lenssen, Eddy Ilg, Tanner Schmidt, Julian Straub, Steven Lovegrove, and Richard Newcombe. "Deep local shapes: Learning local sdf priors for detailed 3d reconstruction". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2020, pp. 608–625.

[9] Chiyu Jiang, Avneesh Sud, Ameesh Makadia, Jingwei Huang, Matthias Nießner, Thomas Funkhouser, et al. "Local implicit grid representations for 3d scenes". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020, pp. 6001–6010.

[10] Ishit Mehta, Michaël Gharbi, Connelly Barnes, Eli Shechtman, Ravi Ramamoorthi, and Manmohan Chandraker. "Modulated periodic activations for generalizable local functional representations". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2021, pp. 14214–14223.

[11] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. "Instant Neural Graphics Primitives with a Multiresolution Hash Encoding". In: *Proceedings of SIGGRAPH*. 2022.

[12] Stefanos Ioannou, Hana Chockler, Alexander Hammers, and Andrew P King. "A Study of Demographic Bias in CNN-Based Brain MR Segmentation". In: *Proceedings of the International Workshop on Machine Learning in Clinical Neuroimaging (MLCN)*. 2022, pp. 13–22.

[13] Esther Puyol-Antón, Bram Ruijsink, Jorge Mariscal Harana, Stefan K Piechnik, Stefan Neubauer, Steffen E Petersen, Reza Razavi, Phil Chowienczyk, and Andrew P King. "Fairness in cardiac magnetic resonance imaging: assessing sex and racial

bias in deep learning-based segmentation". In: *Frontiers in Cardiovascular Medicine* (2022), p. 664.

[14] Tiarna Lee, Esther Puyol-Anton, Bram Ruijsink, Miaojing Shi, and Andrew P King. "A systematic study of race and sex bias in CNN-based cardiac MR segmentation". In: *arXiv preprint arXiv:2209.01627* (2022).