TLBO-based Algorithms for Minimalization of Multi-Ray Path Lengths in Voxel Object Representations on the GPU

Thomas Hudcovic

May 23, 2022

Fachbereich 3 - Mathematik / Informatik

Computer Graphics and Virtual Reality group (CGVR)
https://cgvr.cs.uni-bremen.de/

Master Thesis

TLBO-based Algorithms for Minimalization of Multi-Ray Path Lengths in Voxel Object Representations on the GPU

Thomas Hudcovic

Examiner 1 Prof. Dr. Gabriel Zachmann
Fachbereich 3 - Mathematik / Informatik
Universität Bremen

Examiner 2 Prof. Dr. Ron Kikinis
Harvard Medical School, Department of Radiology
Harvard University

Advisors Prof. Dr. Gabriel Zachmann
Dr. René Weller

May 23, 2022

Thomas Hudcovic

TLBO-based Algorithms for Minimalization of Multi-Ray Path Lengths in Voxel Object Representations on the GPU

Master Thesis, May 23, 2022
Examiners: Prof. Dr. Gabriel Zachmann and Prof. Dr. Ron Kikinis
Advisors: Prof. Dr. Gabriel Zachmann and Dr. René Weller

Universität Bremen
Computer Graphics and Virtual Reality group (CGVR)
https://cgvr.cs.uni-bremen.de/

Fachbereich 3 - Mathematik / Informatik
Bibliothekstraße 5
28359 Bremen

Abstract

This master thesis presents an approach, implemented as a usable system, to solve a subproblem found in the area of ray tracing in heterogeneous media: Determining the "shortest path", where the definition of "shortest" is dependent on the application domain. For example, in nuclear physics, "shortest" can mean the least amount of energy required for a particle/wave to traverse a volume from point A to B. The application domain chosen for this thesis is radiation therapy with charged particles, specifically radiation therapy with protons and how to determine the best beam angles depending constraints and the media given.
This is done by firstly ray tracing through a CT, or similar discretized spatial data in form of a 3D grid, using the appropriate beam model to build a solution space or "cost map".This is then used, along with the domain-specific constraints, to form a binary multidimensional knapsack problem (0/1-MKP). This is then solved using a more recent metaheuristic called "Teaching-Learning-Based Optimization" (TLBO) over a initially generated population of solution candidates. All of this is attempted to be solved with performance in mind: Major components of every step ("cost map generation", "initial population generation", "evaluation", "repair operation") are utilizing the GPU.
In fact, utilizing the GPU to solve TLBO-specific steps is generally not widely explored and to the best of the author's knowledge, one specific operation of this variation of the TLBO-algorithm, the repair operator, has no well-documented GPU-based implementation. Furthermore, to the best of the author's knowledge, the utilization of TLBO as a metaheuristic within the application domain of radiation therapy has also not been widely explored.
The quality of the results as well as the time required for computation indicate that the system developed in this thesis can be usable as a suggestive indicator for a domain expert to include in further considerations.

List of Figures

List of Tables

List of Algorithms

Erklärung

Ich erkläre eidesstattlich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benutzt und die den benutzten Quellen entnommenen Stellen als solche gekennzeichnet habe. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt.

Desweiteren habe ich keine Informatik-Masterprüfung in dem gleichen oder einem fachlich entsprechenden Studiengang an einer wissenschaftlichen Hochschule in der Bundesrepubilk Deutschland endgültig bestanden oder nicht bestanden oder befinde mich in einem entsprechenden Prüfungsverfahren. Ich habe nicht den Prüfungsanspruch durch entgültiges Nichtbestehen einer Prüfung verloren.

Bremen, May 23, 2022

Thomas Hudcovic

Contents

1 Introduction

1.1 Motivation And Problem Statement

In 1525, Albrecht Dürer described a technique to help draw (render) perspective geometry on a flat surface. Utilizing a string (ray) going from the object to be drawn through a frame simulating the flat surface[Hug+13]. By marking the position within the frame where the string intersects (tracing the ray), a perspectively correct corresponding point can be drawn on the flat surface. This principle of tracing a ray to render 3D scenes on a 2D surface has been one of the foundational principles of computer graphics[Hug+13], with the first simple ray tracer described by Arthur Appel in 1968[App68].
A natural evolution of the problem of tracing rays for rendering purposes in homogeneous media like air, and only considering opaque or fully reflective objects, is the consideration of different possible compositions of the medium the rays are traced through. Specifically, ray tracing through a volume with possible sub-volumes of different densities and nuclear reaction characteristics, influencing photons along the path, including objects with varying transparency and/or photon refraction and reflection attributes (e.g. rendering of clouds, fog, windows, water, "thin objects" like leaves or paper).

An occurring subproblem of ray tracing heterogeneous media is determining the "shortest path", where the definition of "shortest" is dependent on the application domain. For example, in nuclear physics, "shortest" can mean the least amount of energy required for a particle/ray to traverse through the volume. In geophysics/-seismology, it can be interpreted as the part of the path of a seismic wave losing the least amount of energy, i.e. where the seismic wave has the longest reach (or the inverse!). Both examples depend on the density and composition of the volume.
Looking at the literature, beam positioning in medial physics within the context of radiation therapy for cancer treatment is a very popular topic. The motivation thereof is easily found:
In 2020, the World Health Organization (WHO) reported cancer as a leading cause of death world wide, with approximately 10 million deaths globally in 2020 alone and with lung and breast cancer sharing the ”top spot” for the most reported cancer types[WHO20]. Official reported numbers of cancer cases and types for the United States alone, beginning from the early 1930s until today, show either a rapid growth

or at least steady growth of cancer cases[Glo18]. Naturally, cancer and the therapy thereof has become one of the key areas of medical research, with one of the earliest reported descriptions made by Hippocrates in 460-370 BCE [WC08]. With the advent of physical advancements, especially in nuclear and particle physics, development and research of possible therapy/treatment forms for cancer, incorporating these new discoveries and developments, has been growing rapidly since the late 19th century[WC08].
Today, the common treatment modalities for cancer can be categorized into surgery, chemotherapy and radiation therapy, with the possibility of incorporating all modalities into a therapy plan for a patient. Radiation therapy specifically has been a focus of medical research for cancer therapy as it promises a non-invasive and localized form of treatment, with the main trade-off being the cost and acquisition of the equipment and trained personnel required[Zie15].

This thesis focuses on external (particle) radiation treatment as an application domain of presenting a method for attaining shortest paths in a heterogeneous medium, where carefully positioned beams around a patient are aimed at a therapeutic volume including the tumor and emitting radiation in such a way that the maximum energy dose of the radiation hits the tumor, while sparing non-cancerous tissue. The main differentiating factor of external radiation treatment procedures is the beam modality, i.e. the type of radiation emitted. The most used and researched being high-energy photon radiation, while ion particles (Protons, Carbon Ions, ...) still remain a niche in clinical practice despite favourable physical attributes making them more suited for treating deeper situated tumors[NZ15]. Especially for tumors situated in more critical areas like the brain, the neck or the lungs, where maximizing the amount of healthy tissues spared is, in theory, easier to achieve than with high-energy photons[NZ15]. Over/under-shooting the target being an especially critical issue with charged particles, because of their nature in delivering the maximum amount of dosage at a specific point (Bragg-Peak) as opposed to broader area of delivery for photons[NZ15]. The favourable attributes of charged particle radiation being the reason as to why proton radiation as beam modality has been chosen for this work.

The shortest path here is defined by the path through the body requiring the least amount of initial energy and proton fluence (influencing the number of nuclear interactions along the way, e.g. irradiation) required to hit the target while travelling through the body (heterogeneous media), sparing healthy tissue as much as possible.

1.1.1 Contribution And Goal

The goal of this thesis is to describe and provide a fast method for selecting the shortest direct paths in heterogeneous media, using the application domain of proton radiation therapy. This translates to the problem of selecting optimal beam angle orientations for intensity-modulated proton therapy in an automatic manner, more broadly known as "Beam Angle Optimization".
The DKFZ's open-source matRad-tool[Wie+17], which is a treatment planning system for radiation therapy, will be used as basis for communicating with the system presented, as well as Matlab[MAT21] itself, in which matRad was written in. With performance as one of the key goals, a lower level of control of memory and computational paths is required. Thus, the system itself will be implemented in C++ and will also utilize the GPU, using nVidia's CUDA[Nic+08] for parallelization. This thesis strives to contribute the following:

- A fully implemented system to determine shortest direct paths through heterogeneous media using ray tracing and teaching-learning-based optimization (TLBO) using the GPU.
- A new approach to parallelizing the repair- and evaluation operations of TLBO on the GPU.
- A fast method to generate an initial population of unique, viable solution candidates for TLBO, using the isomorphic mapping between factoriadic and k-combinations.
- An approach to the beam angle optimization problem in proton radiation therapy, implemented for an open-source treatment planning system (matRad).
- To best of the author's knowledge the first application of TLBO to optimize multi-ray path lengths over a heterogeneous voxel grid.

Regarding the second item: A thorough search of the related literature yielded only one other approach to utilize the GPU for the purpose of parallizing TLBO, developed by Rico-Garcia et al.[Ric+19]. However, the method by Rico-Garcia et al. does not really document or describe in detail how the evaluation operation is being computed on the GPU, nor does it utilize a repair operator as described in the approach of Kern et al.[KLV20], which serves as the basis for the method developed in this thesis.

1.2 Related Work

The problem of attaining shortest direct paths through a voxel grid describing heterogeneous media/volumes can be observed from two perspectives: From a specific application area (e.g. beam positions in medical physics or drill/boring paths for gas or oil extraction in geophysics) or as a system of two major components: Ray tracing and optimization algorithms.

For the first perspective, as this thesis focuses on the application area of proton radiation therapy within medical physics, related work/research has proven to be abundant:

Beam angle optimization is part of a bigger pipeline of algorithms and methods to compute target doses from input CT data, known as treatment planning systems (TPS). Selecting optimal beam angles automatically has been a long-researched problem in photon-based radiation therapy and the general nature of beam angle optimization can be seen as invariant to the radiation modality. However, because of the nature of charged particles, solutions for photon-based beams often aren't directly applicable, so research for proton-based beam angle optimization is not as numerous in comparison. And of that research done on beam angle selection, there is a significant amount concentrating on the robustness of beam angles against uncertainties, since it is a more critical factor in proton therapy[LCM14]. Generally, beam angle optimization is a optimization problem on a concave and usually at least C^0-continuous manifold or search space[BS93]. The newer approach to proton therapy, known as intensity-modulated proton therapy (IMPT), is considered here, with pencil-beam scanning (PBS) as active delivery method, where the target volume is discretized into spots shot at by a proton beam with varying initial energy[Pag17].

A common approach to the problem of beam angle optimization is to use candidate angles, either by using a form of sampling on a spherical surface or by just considering equidistantly placed angles or by candidate angles based on clinical heuristics. Out of this candidate set of beam angles, an optimization technique is applied on to get the angles most suited to given clinical objectives to optimize against.
Kiely et al. have considered two common metrics as criteria for optimizing beam angles. They considered the geometric depth or geometric path length and the variation of Hounsfield units along the path, for lower pelvis targets. Comparison of impact of these two metrics found that the shortest geometric path length is still more impactful on overall dose distribution of the patient than HU variation and therefore more important to consider when optimizing beam angles[KWB15].

Significant work has been done by Gu et al. Also using a given set of candidate angles, comprised of equidistantly placed angles around a sphere, then for every candidate beam angle, the scanning spots covering the target volume were calculated and optimal beams were chosen by formulation of three constraints: Sparsity of spots, group sparsity of beams and dose fidelity to penalize variation of set clinical target doses for organs-at-risk and the target volume. Optimization was done by applying the Fast Iterative Shrinkage-Thresholding algorithm[Gu+18]. Building on this, Gu et al. have considered the heterogeneity of the tissue and incorporated it into the group sparsity constraint. A heterogeneity index evaluates tissue heterogeneity lateral to the beams, with the goal being to choose tissue with less heterogeneity, such that beams and beam angles are more robust to geometric variations/uncertainties[Gu+19]. A newer approach by Gu et al. considered the inter-fractional variation of beam angles to optimize for dosages by choosing from a pool of up to 800 non-coplanar candidate beams, with a varying beam amount of 1 to 4. These beams are then compared to treatment plans with fixed beams between fractions[Gu+20]. For both, similar number of beams were chosen to have a common basis to compare on. It was found that variation of beam angles between fractions spare organs-at-risk better and even allow for less applied number of beams while delivering the same dose.

A hybrid approach to selecting optimal beam angles without the need for a set of candidate angles beforehand was investigated by Bertsimas et al. They used simulated annealing to search over a beam heuristic together with gradient information to solve quickly for local minima, which are then used to solve globally for optimal beam angles[Ber+13].
Taasti et al. considered Bayesian optimization to select optimal beam angles out of an initial set of candidate beams. They argue that Bayesian optimization works well on smooth manifolds, i.e. if the objective function has little variation for each point's epsilon-environment within the function's range[Taa+20]. The resulting treatment plans were then evaluated by a score-function that rates the quality of the generated angles/plan by clinical criteria.

One approach, also used by this thesis, was described by Kim et al. They transform all the volumes of interest (including the target volume and organs-at-risk) into a common coordinate space, where the different tissue densities and heterogeneities are described by a common metric known as water-equivalent path length (WEPL). WEPL describes the corresponding thickness in water of a tissue according to a radiation modality chosen (here proton radiation) and respective energy-levels[Kim+20]. However, their approach was to optimize beam angles for robustness: Using WEPL, they formulated a metric upon which range variations, for example due to setup error or changes in the structure of surrounding tissue in the patient, and their impact

on therapeutic effects can be measured. Beam angles being the most susceptible to range variations were discarded while angles being more robust against range variations were suggested. Range variations were calculated by considering the angular variation of calculated WEPLs for beams, equidistantly in 5 degree intervals between 0 and 180 degrees.
Another, more disputed [Pag17][JMP18][Lüh+18], metric to evaluate beam angle orientations and general effectiveness of a treatment plan is relative biological effectiveness (RBE). RBE describes the ratio of physical doses between two different types of radiation required to produce the same biological endpoint[Pag17], usually in comparison to photon-based radiation. Protons have an RBE of 1.1, i.e. proton radiation is considered to be 10% more effective[Pag17].

The second perspective, observing the problem as a composition of ray tracing and optimization, also delivered an abundance of related work/research:

From the first conceptualization and implementation of a simple ray tracing algorithm in 1968 by Arthur Appel[App68], to the efforts of Purcell et al. of utilizing the GPU as an accelerator for ray tracing via massive parallelization[Pur+05], ray tracing has been a perennial topic of research throughout the whole existence of computer graphics as a field of research. And now with GPU manufacturers pushing real-time ray tracing capabilities as a selling point for consumer-grade GPUs, recent research has been more active again. As part of this thesis' method is to ray trace heterogeneous voxel grids on the GPU, research on ray tracing and related data structures on the GPU were the main topics when looking through related research. Due to the nature of the streaming multiprocessors applied on the GPU, hierarchical sptial data structures employing trees or other recursive hierarchies have been very hard to implement on the GPU without sacrificing a lot of performance in comparison to CPU implementations[Kel+19][Mei+21]. Commonly, suitable spatial data structures to accelerate object rendering are constructed and initialized "offline", i.e. on the CPU, and then sent over to the GPU for processing and rendering. Each update to the data structure is again done on the CPU and again sent over to the GPU.[Mei+21].
It can be observed that a lot of somewhat recent worked focuses on parallelizing either the foundational DDA algorithms by Siddon et al.[Sid85] and Amanatides et al.[AW87] (and others) or driving-axis methods, like Joseph's projection method[Jos82]of ray tracing for usage on the GPU, where the "compuitational direction" of sampling is determined by the biggest component of the direction vector, i.e. the driving-axis. The focus there lies often in optimizing memory consumption and layout, as done by Heinrich et al.[Hei+14] (for Siddon et al's algorithm), by cleverly overlapping host-device memory transfers as well as laying out data for concurrent access in (pinned) page memory. Or by reducing branch divergence, as described by Xiao et al.[Xia+12] for Amanatides et al's algorithm (this will be used as the basis

for the ray tracing algorithm used in this thesis), or a branch-divergence reduction for the algorithm developed by Graetz[Gra20] of the Joseph-projection algorithm. Both of the approaches by Xiao et al. and Graetz for reducing branch-divergence their respective progenitor algorithms are essentially a combination of reformulating if-conditionals as Boolean equations and shifting calculations from within the loop to the outside and only updating fragmental information instead. Ongoing research has also specifically targeted heterogeneous (or rather transparent/translucent volumes) data, as presented by Zhang[Tan21], wherein a GPU-based ray tracer was developed utilizing Vulkan and showing the major trade-off: Accuracy (of the light refracted/absorbed/reflected) vs. speed. Or a special grid to discretize a scene to encode relative distances called (Euclidean) distance fields[FH12], to encode translucency. These distance fields can also be used as a data structure to accelerate ray tracing[KM19] (which served as an inspiration for the "CostMap"-approach discussed in chapter 3) and are used in the current version of Unreal Engine for calculating soft shadows[Epi20b] or ambient occlusion[Epi20a].

Akin to ray tracing, mathematical optimization is also a perennial topic of research with an especially wide application area. Optimization algorithms and more specifically metaheuristics are usually found when dealing with problems in NP. Thus, metaheuristics tend to have non-deterministic components, often related, but not limited to, generating random (usually with a specific direction or bias, i.e. not completely random) solutions as a basis or combining solutions to form new ones with a certain (directed/biased) randomness. As such, they don't guarantee to deliver the best solution for a given problem but attempt to converge towards the (global) optimum, i.e. delivering solutions that are "good enough". As one specific NP-complete optimization problem, known as 0/1 Multidimensional Knapsack Problem (MKP, refer to chapter 2 for a more detailed explanation), is attempted to be solved in this thesis as part of the overall problem statement, the focus will be on related research pertaining to MKP.
A common and proven approach to solving (smaller) MKP-problems is to encode them as a linear programming problem and then applying a relaxation scheme (usually relaxing the binary 0/1 constraint to the real number space, i.e. providing more "wiggle room")[Dye+17]. Another popular approach is to apply evolutionary algorithms, which usually utilize biologically inspired combination operations of two solutions in a population with a certain degree of randomness to form new ones[AMA19]. The most popular algorithm of this category is known as Genetic Algorithm, with seminal (and often cited) work done by Chu at al.[CB98]. Utilizing the GPU for parallelization, population-based metaheuristics like the aforementioned genetic algorithm, have been a recent focus of research as well: Another type of these metaheuristics are particle swarm optimization (PSO), with Hung et al. utilizing the GPU for acceleration via a thread-pool model and step-wise memory access patterns inbetween work-steps[HW12]. Another kind of population-based algorithms that

profit from GPU acceleration to solve (among others) the MKP problem is known as ant colony optimization (ACO), with Huang et al. surveying and discussing various approaches made to parallelize ACO on the GPU[HLY19], with general message of ACO receiving a significant performance boost as well.
However, many metaheuristic methods require other parameters in addition to the problem input as well. And depending on the values of these required parameters, the algorithms may only deliver sub-optimal solutions or fail entirely. This has resulted in the advent of a sub-problem as a dedicated area of research: Parameter tuning, often utilizing domain-specific knowledge by an expert of the problem domain wherein the algorithms are applied or, more recently, using neural networks to learn parameters from past data and applying it to similar problems[MD19].
For this thesis, another category of metaheuristics has been chosen that require very little to no parameter tuning as the author has not enough domain specific knowledge in either medicine nor medical physics to rule out "bad parameter tuning" as a possible cause for sub-optimal results. The area of operations research has delivered a fairly recent metaheuristic described by Rao et al., called "Teaching-Learning-based optimization" (TLBO)[RSV11], consisting of two phases where one phase globally optimizes the set ("Teaching phase") and a second phase locally optimizing the solutions ("Learner phase"). A newer variation on it has been described by Kern et al.[KLV20], adding the repair operator and combination methods used in the work done by Chu et al. for genetic algorithms. The approach by Kern et al. has also been chosen for this thesis as a basis and will be more thoroughly discussed in chapter 3. Rico et al.[Ric+19] described a parallelized version of TLBO utilizing the GPU and compared it with a similar algorithm called Jaya, also parallelized using the GPU, and is to the best of the author's knowledge the only sufficently described method of using TLBO on the GPU.

1.3 Thesis Structure

Following this chapter, in which motivations, goals, contributions and related work is discussed, an overview and explanation of used technologies and methods will be given.

Chapter 3 presents an overview of the system itself and its components, as well as providing detailed explanations of algorithms and methods developed.

Chapter 4 concludes by discussing and evaluating results, as well as describing the used data sets; and concluding by discussing limitations, suggesting possible improvements and giving an outlook for future developments.

2 Preliminaries

This chapter discusses all relevant preliminaries necessary to understand the method developed and evaluated in the following chapters. It is assumed that the reader is familiar with general computer science concepts but not necessarily familiar with concepts used in radiation therapy and related particle physics. The relevant concepts will be introduced here as well as an attempt will be made to build an analogy from concepts of computer science, especially computer graphics, to radiation therapy, hopefully to support understanding.
This is also the reason why, although a fundamental concept in computer graphics, ray tracing will be discussed very briefly. It is assumed the reader is familiar with GPUs and parallelization on GPUs (implemented via CUDA).

2.1 Ray Tracing And Beam Models

When approaching the topic of medical physics, specifically radiation physics and proton radiation therapy as an application area for the system/set of algorithms developed in this thesis from a computer science point of view, it has proven helpful to identify analogous connections that help translate already familiar structures (here from a viewpoint of rendering) to unfamiliar structures (particle physics, radiation physics).
The spectrum of electromagnetic waves largely comprises photons of different energy levels (i.e. wave lengths), under which also the visible light spectrum as well as higher energy photons can be found (x-rays, γ-rays). Computer graphics has since its inception aspired to depict the visual phenomena that can be perceived by humans as accurately as possible but always with respect to computability in general and computability within time constraints (e.g. for real-time rendering), and thus diverged from directly applying (quantum)-physical models for computation: Many of the physical models have proven to be not of closed form and thus, approximations have been developed, often using lower dimensional and/or simplified domains, forgoing certain phenomena considered irrelevant for the problem at hand (e.g. the famous rendering equation by [Kaj86], mostly focusing on phenomena related to geometric optics in homogeneous media related to photon traversal).

Ray tracing, the popular and foundational principle of rendering a (3D) scene on an image plane by tracing the path of a ray of photons and tracking reflectance and radiance behaviour[PJH16] can also be used with other "beam models", i.e. with other particles than photons of the visible light spectrum and in other media than air. This idea will be referred to as beam model or beam modality henceforth. As ray tracing in its core principles essentially remains the same regardless of beam model used, it will be briefly recalled next.

2.1.1 Ray Tracing

In 1968, Arthur Appel formally introduced and described the first simple ray tracer[App68] and has become one of the foundational ideas of rendering in the computer graphics community[Hug+13]. The idea however, can be traced back hundreds of years earlier to 1525 and Albrecht Dürer, describing a technique to help draw (render) perspective geometry on a flat surface. Utilizing a string (ray) going from the object to be drawn through a frame simulating the flat surface. By marking the position within the frame where the string intersects (tracing the ray), a perspectively correct corresponding point can be drawn on the flat surface[Hug+13].

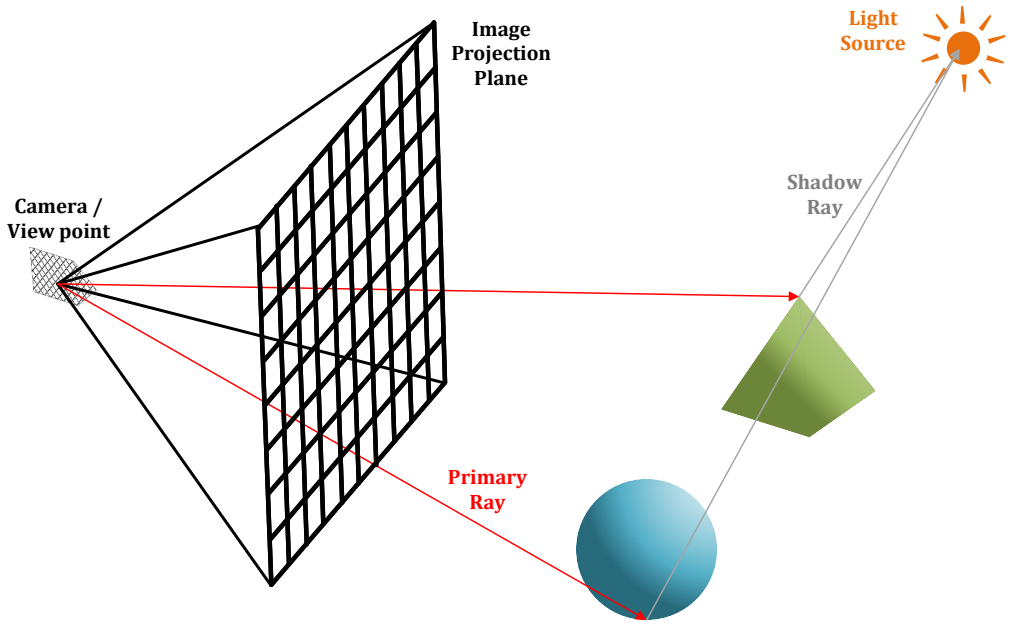


Fig. 2.1: Ray tracing a simple scene, only using primary and (secondary) shadow rays with one light source.

As depicted in figure 2.1, from a view point or camera, (primary) rays are shot at the scene until they collide with an object (determines visibility) or a scene boundary. From the point of impact, a secondary ray to all the light sources in the scene

is traced to determine the coloring and shading. The corresponding pixel on the discretized image projection plane (often a computer display) intersecting with the corresponding ray is receiving the accumulated color of the tracing.
Depending on the properties of the objects and the beam model given, a beam might go through an object, losing energy in the process (and/or emitting new light to simulate emission in more complex beam models), might scatter off in another direction, is reflected or is absorbed. Often, in more advanced ray tracers, additions are made to simulate more complex visual phenomena. E.g. depicting soft shadows often use more rays than just one shadow ray shot out in a specific way (called sampling scheme)[PJH16]. Or introducing additional conceptual planes in the scene used for computational purposes like a focal plane together with a Gaussian convolution of the rays traversing through it to simulate depth-of-field[PJH16].

Even more advanced is to generalize the concept of ray tracing to a method called "Path Tracing", where, after shooting the primary rays, stochastic methods (Monte Carlo methods) are applied to determine a hemisphere of randomly shot out secondary rays, which in turn will shoot out also tertiary rays randomly from their point of impact (often with certain biases depending on the material of the object) to sample light and simulate "indirect illumination" from light bouncing off from other objects to ultimately simulate an effect called global illumination (among other things)[PJH16].

As ray tracing is part of the implementation of the system and method developed in this thesis, it is helpful to list the algorithmic basis on which that part is developed: The seminal work by Amanatides et al[AW87], describing a fast method of ray tracing through a scene or volume discretized by a grid.

Algorithm 1: Xiao et al.'s description of the original DDA-based ray-tracing algorithm by Amanatides et al.[AW87], edited for clarity. The traversal part relating to figure 2.2 is colored in orange.

initialize ray information(source point $\mathbf{S}$, direction $\overrightarrow{d}$);
initialize voxel information(voxel size v_{size});
initialize $\mathbf{V_{current}}$ as indices of the voxel where $\mathbf{S}$ is;
if $\exists d_i \in \overrightarrow{d} : d_i \neq 0$ **then**
    $\mathbf{step} = \{s_i | \forall d_i \in \overrightarrow{d} : s_i = ternary\_op((d_i < 0), -1, 1)\}$;
    $\overrightarrow{d_{inv}} = \frac{1}{\overrightarrow{d}}$;
    $\mathbf{t} = ||\mathbf{S} -$"nearest boundary intersection with ray from $\mathbf{S}$"$||_2$;
else
    $\overrightarrow{d_{inv}} =$"a large value for each component";
    $t =$"a large value for each component";
$\mathbf{\Delta} = v_{size} \cdot \overrightarrow{d_{inv}}$;
while $\mathbf{V_{current}}$ *is inside traversal region* **do**
    if $t_x < t_y$ **then**
        if $t_x < t_z$ **then**
            $t_x += \Delta_x$;
            $\mathbf{V_{current}}_x \mathrel{+}= step_x$;
        else
            $t_z += \Delta_z$;
            $\mathbf{V_{current}}_z \mathrel{+}= step_z$;
    else
        if $t_y < t_z$ **then**
            $t_y += \Delta_y$;
            $\mathbf{V_{current}}_y \mathrel{+}= step_y$;
        else
            $t_z += \Delta_z$;
            $\mathbf{V_{current}}_z \mathrel{+}= step_z$;

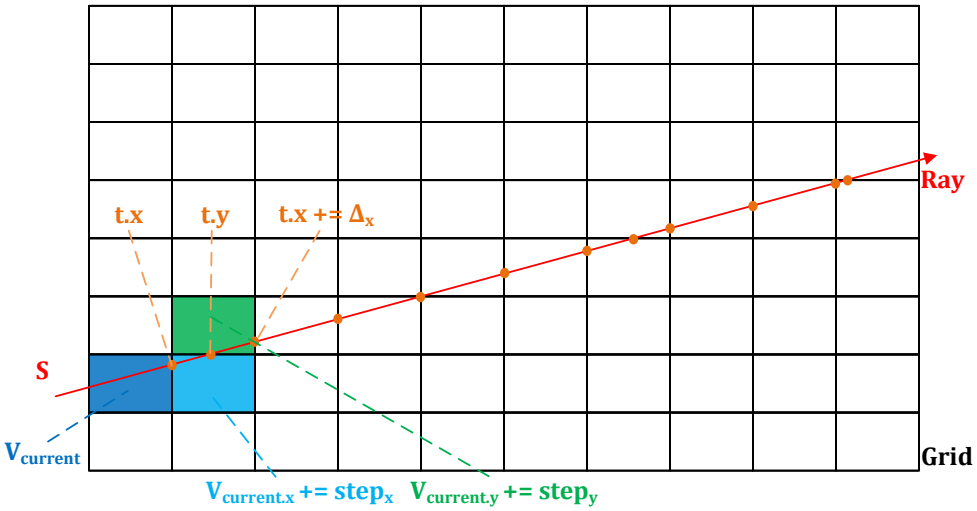


Fig. 2.2: Voxel traversal during ray tracing as described by Amanatides et al.[AW87] (Figure adapted from Xiao et al.[Xia+12])

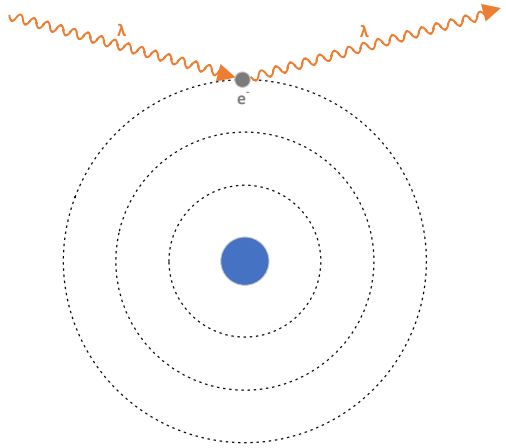
The key part here is the while-loop: Depending on which component of $\mathbf{t}$ is the smallest, i.e. the nearest intersection boundary with the grid, $\mathbf{V_{current}}$ components are accordingly incremented by $\mathbf{step}$. The idea is to view the grid as equidistant hyperplanes of each coordinate axis and using the equidistant attribute to compute the distances to the next intersection boundary (i.e. the next hyperplane) in relation to the ray. $\mathbf{V_{current}}$ is incremented this way along the ray direction $\overrightarrow{d}$ as long as it is inside the grid. Consequently, this approach does not work on irregular grids (e.g. meshes), ray marching might be a better approach in such cases. However, this method will be used later in a parallelized context using Xiao et al.'s contribution, explained in detail in chapter 3.

The purpose for this explanation is to show and familiarize the reader with the idea that regardless the beam model being used, several things about any particle can be "traced" as long as a model exists. So in theory, rendering any scene in any heterogeneous or homogeneous media for any particle phenomenon is possible and applicable via ray tracing (or variations thereof). While it may not be as simple as "plug and play", the general principle can be summarized as such. And tracing protons through a human body is essentially the same category of problem, just with a different model with more considerations.

2.1.2 Photon Radiation Physics

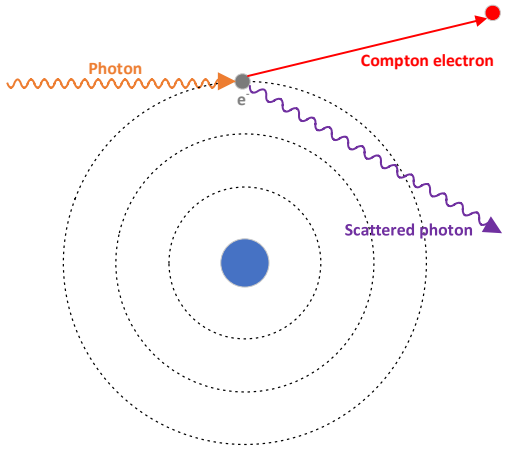
Described in the following are the four main nuclear interaction types of ionizing photon radiation:

Coherent Scattering



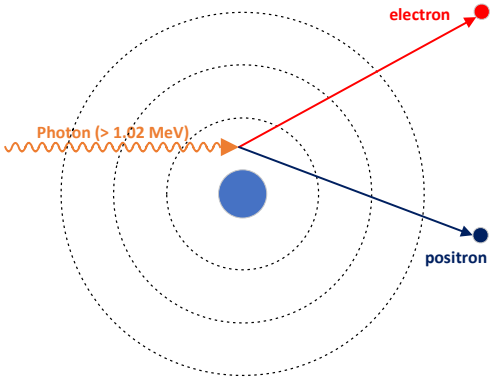
If the dual nature of electromagnetic radiation is considered, coherent scattering of high-energetic photons can be viewed as an incident electromagnetic wave passing near an orbital electron and oscillating it. The oscillation of the electron generates electromagnetic radiation of the same frequency as the incident wave. Since no absorption or attenuation of energy takes place, coherent scattering doesn't have much clinical significance[KG14]. The generated wave is scattered at usually small angles relative to the incident wave and has the highest occurrence probability in lower-energy photons[KG14].

Compton Effect



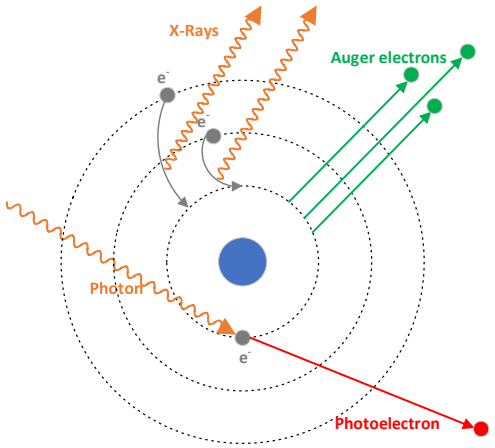
The Compton Effect describes an interaction of an incident high-energy photon with an orbital electron, where the energy of the incident photon is much higher than the binding energy of the electron to its atomic orbit. This collision causes the orbital electron to absorb some of the incident photon's energy and therefore to be emitted from its orbit (Compton electron). The incident photon is scattered (Scattered photon) with a lower energy-level as a result.[KG14] This interaction is the most crucial in photon-based radiation therapy.

Pair Production



Pair Production occurs when an incident high-energetic photon passes near the nucleus of an atom, disappearing and creating an electron and its antimatter (positron). Both particles inherit the kinetic energy of the incident photon. The positron annihilates upon losing all of its energy[KG14].

Photoelectric Effect



The photoelectric effect describes a phenomenon in which an incident high-energetic photon's energy is close to the binding energy of an orbital electron. The photon hits the orbital electron, transferring all its energy, causing the electron to be ejected from its orbit with the kinetic energy equal to the incident photon's energy minus the electron's binding energy (Photoelectron)[KG14]. The vacant spot can be filled by one of the outer shell electrons, which radiate X-Rays during the process of jumping to another shell. There is also a possibility of emitting Auger electrons during the jump, when the energy released by one of the outer shell electrons hits one of the other outer shell electrons, causing them to be ejected[KG14].

Tab. 2.1: Overview of nuclear interaction types of ionizing photon radiation occurring in radiation therapy.

Ionizing photon radiation in the form of a beam of x-rays or γ-rays passing through a medium is transferring energy to the absorber medium (the human tissue) by ejecting orbital electrons from the medium's atoms. The ejected electrons then transfer their energy to other atoms in their paths, ionizing them in turn via inelastic collisions. The energy resulting from this process destroys enough of the tissue's cell components (DNA) such that the cells get destroyed or inhibited in their cell division cycle[KG14]. Because of the indirect nature of ionizing photon radiation, where

x-rays or γ-rays are just responsible for ejecting particles that directly ionize their surroundings, ionizing photon radiation falls into the category of indirect ionizing radiation[KG14].
Of the 4 main nuclear interaction types for high-energetic photons listed, coherent scattering has no practical effect on ionizing the surrounding tissue for therapeutic or diagnostic purposes, as no energy is absorbed by the absorber medium/tissue and no significant attenuation of the incident beam's energy occurs. For the remaining three types, significant energy absorption and beam attenuation occurs; which interaction type predominates in its occurrence depends on the incident photon beam's energy[KG14]. Diagnostic x-rays, usually in the keV energy range, interact predominantly via the photoelectric effect, whereas therapeutic γ-rays, usually in the MeV range, interact predominantly via the Compton effect and via pair production[KG14]. In the lower MeV range ($\leq$24MeV)[KG14], the Compton effect has the highest occurrence probability. Pair production predominates in its occurrence in the higher MeV ranges (>24MeV)[KG14].

Thus, familiar rendering equation by Kajiya is an approximation of the "classical" electromagnetic wave models described by Maxwell and known as Maxwell equations, which were phenomenological models from a macroscopic observation point, where a special focus was given to approximate of the famous rendering equation by Kajiya[Kaj86], given here in its directional form[PJH16]:

$$L(\mathbf{x}, \overrightarrow{\omega_0}) = L_e(\mathbf{x}, \overrightarrow{\omega_0}) + \int_\Omega f_r(\mathbf{x}, \overrightarrow{\omega_i}, \overrightarrow{\omega_0}) L_i(\mathbf{x}, \overrightarrow{\omega_i}) \, |cos\, \theta_i| \, d\overrightarrow{\omega_i} \tag{2.1}$$

With:
$\omega_0 :=$ initial direction, $\mathbf{x} :=$ position,
$f_r(\mathbf{x}, \overrightarrow{\omega_i}, \overrightarrow{\omega_0}) :=$ bidirectional reflectance distribution function (BRDF),
$|cos\, \theta_i| \;\; :=$ the angle of the incoming/incident beam from direction ω_i and the normal vector of position $\mathbf{x}$, essentially equal to $\overrightarrow{n} \cdot \omega_i$,
$\int_\Omega :=$ hemisphere of incident directions of other rays denoted by i, $rho(\overrightarrow{\omega}, \overrightarrow{\omega'}) :=$ phase function.

The rendering equation presents a model for rendering radiation of photons of energies within the visible light spectrum and is widely used. However, as stated by Kajiya himself, it is an approximation of selected phenomena of the classical (macroscopic) electromagnetic wave model based on Maxwell's equations, mainly relating to geometrical optics (movement through homogeneous media, reflectance of radiance, averaged transport loss of radiance)[Kaj86]. Maxwell's models in turn are also not accounting for every phenomenon observed from a quantum mechanical point of view[KG14]. In truth, this model cannot be applied prima facie, since it is not of closed form either. In fact, it can be argued that the whole approach of

rasterization to rendering is a series of composed methods that try to approximate this equation.

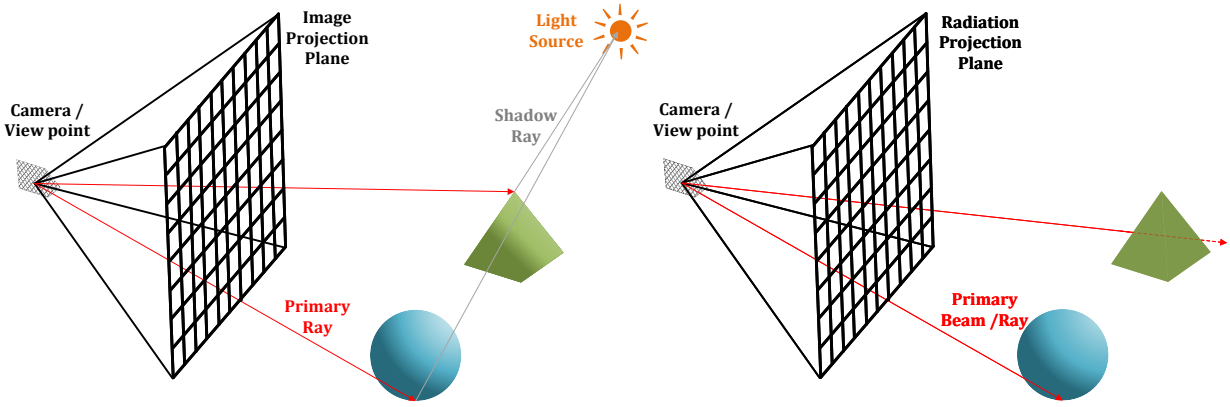


Fig. 2.3: Left: Ray Tracing, i.e. tracing a particle on a linear path (ray) through a medium (here the scene in air and the objects being fully opaque) using the model given by equation 2.1
Right: Ray Tracing here works by the same principle, i.e. tracing a particle on a linear path (ray) through a medium (here the scene in air and the sphere being fully absorbing and the tetrahedron being of different density than air) using a different model, e.g. for protons as described by equation 2.5.
The overall method remains the same but the physical constraints and calculations changed.

A more detailed model (but an approximation as well) approximating the listed photon interactions better is the radiative transfer equation for electromagnetic waves, such as photons with energies of the visible light spectrum.[Jak+10]:

$$(\overrightarrow{\omega}, \overrightarrow{\nabla})L(\mathbf{x}, \overrightarrow{\omega}) = -(\sigma_a + \sigma_s)L(\mathbf{x}, \overrightarrow{\omega}) + Q(\mathbf{x}, \overrightarrow{\omega}) + \sigma_s \int_\Omega L(\mathbf{x}, \overrightarrow{\omega})\rho(\overrightarrow{\omega}, \overrightarrow{\omega'})d\overrightarrow{\omega'} \tag{2.2}$$

With:
$\omega :=$ direction, $\mathbf{x} :=$ position, $(\overrightarrow{\omega}, \overrightarrow{\nabla}) :=$differential change,
$\sigma_a :=$ absorption coefficient, $\sigma_s :=$ scattering coefficient, $Q(\mathbf{x}, \overrightarrow{\omega}) :=$ emission,
$\int_\Omega :=$ all directions in a sphere, $rho(\overrightarrow{\omega}, \overrightarrow{\omega'} :=$ phase function.

As Jakob et al. have processed into a computable form (equation 2.2) to help render anisotropic materials, since the original is not of closed form for three-dimensional space. Again, an (popular) example of a used beam model[PJH16].

2.2 Proton Radiation Physics

Selecting beam angles, which is just one of several optimization problems related to radiation therapy, depends heavily on the modality of radiation chosen as the underlying physics are different enough to make methods not applicable (at least not directly) from one type of radiation to another. The most common modality of radiation used is high-energy photon radiation, also known as gamma radiation. Another increasingly popular type of radiation is using particles, of which protons are this thesis' focus. Generally, there are two broad categories of radiation: Electromagnetic and particle radiation. In radiation therapy however, only the ionizing part of the electromagnetic spectrum is relevant, i.e. wave energies of 1keV and higher. It is still common practice in the clinical application of proton therapy to manually select possible beam emitter angles by trial-and-error together with the experience of the planner and clinical lookup-tables[Gu+19].

Electromagnetic	Particle
X-Rays (0.12keV - 50keV[NAS14])	Electrons
Gamma Rays (>50keV[NAS14])	Neutrons
	Protons(50MeV - 250MeV for RT[NZ15])
	Heavy Ions (Such as carbon ions)

Tab. 2.2: Overview of the significant radiation types used in radiation therapy with typical energy ranges used.

With common units used in literature regarding radiation therapy being:
eV denoting **electron-volt**, which quantifies the kinetic energy gained by an electron accelerating through voltage difference, i.e. potential difference, of a volt with 1eV $\approx 1.602 \cdot 10^{-19} J$[NIS20].
Gy denoting **Gray**, which quantifies ionizing radiation dose absorbed by matter, with 1Gy$= 1\frac{J}{kg}$[BB11]. As an example, $7 \cdot 10^{-4}$Gy is the absorbed dose of an X-Ray, $6 \cdot 10^{-4}$ to $1.4 \cdot 10^{-4}$Gy is the absorbed dose for a CT-Scan. For a mild case of acute radiation sickness a absorbed dose of 0.8 to 2.1Gy is required over a period minutes on sufficient body surface area[BB11].
RBE denoting **Relative Biological Effectiveness**, defined as the ratio of dose required to produce the same biological effect between a applied radiation modality to a reference radiation modality (usually high-energy photons)[KG14]. With high-energy photons having a RBE of 1 and protons having a RBE of 1.1[NZ15].

The main two differences between these radiation modalities being that high-energetic photons carry no charge and were observed to only possess mass while moving, thus adhering to the famous $E = mc^2$ equation. Particles, such as protons,

have a resting mass and can carry charge; positive charge in the case of protons, facilitated by their quark configuration[KG14]. These observed attributes heavily influence the interaction of the radiation modality with the target medium and thus influence treatment planning parameters and methods themselves.
The therapeutic target radiation is usually delivered in small doses over a period of one or two months, depending on the parameters of the tumor[KG14] as delivering the target dosage in one sitting would significantly damage surrounding healthy tissue as well.
In figure 2.4, the curve describing the relative dose distributions for photons (at 18MeV) shows that 100% of energy is transferred right at the beginning of interaction with an absorber medium (here water) and monotonically decreases as the photons travel further. This is one main disadvantages of photon radiation: A target volume at depth cannot be hit by 100% of the emitted energy, unless it is very close to the beam's entry point, meaning that healthy tissue will get hit by unnecessary radiation dose if it is between the target volume and the beam's entry point. Furthermore, after the target volume has been hit by the beam, there is still significant dose being delivered to healthy tissue "behind" the target volume (called *exit dose*) due to the almost linear nature of the monotonic decrease. In comparison with the Bragg curve for hadrons, the maximum dose is delivered with almost pinpoint precision at depth with the curve stopping right after (Bragg peak), meaning that there is no "trail" of radiation dose delivered to healthy tissue "behind" the target volume. However, due to the Bragg-peak denoting the maximum dose delivered, the slight bit of the graph right after the peak, falling sharply off is still a factor and a problem in proton therapy, called "(distal) fall-off dose"[KG14].

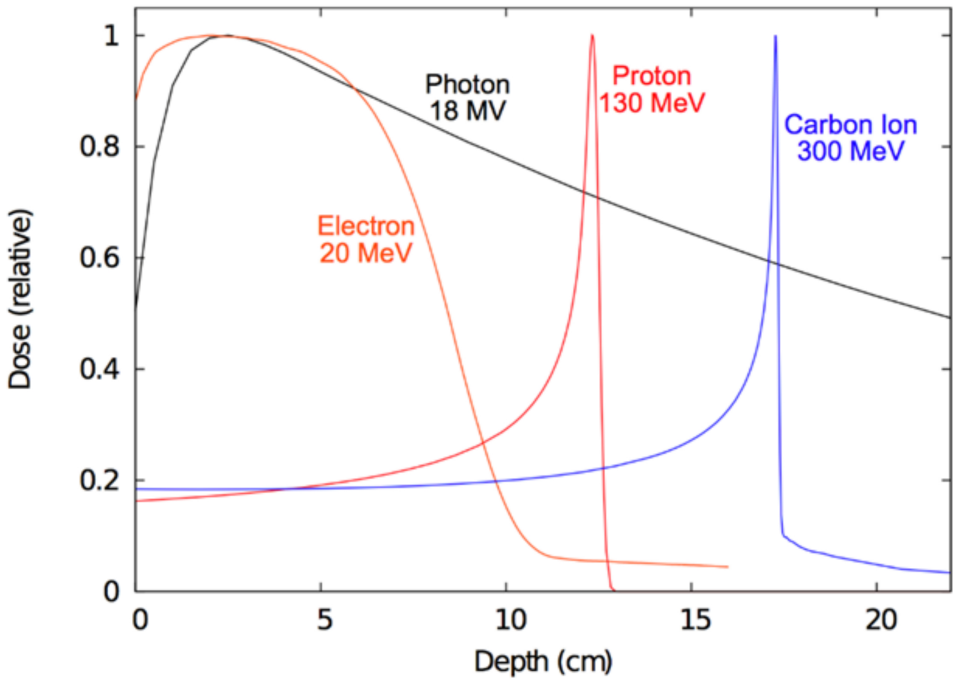


Fig. 2.4: Relative dose distributions in water for common radiation modalities. The Bragg-curve with the corresponding Bragg-peak can be seen for protons (red) and carbon ions (blue). Taken from [Kai+19].

Figure 2.4 visualizes the relative dose delivered for certain energies at a certain depth, this also describes the stopping power $S(E)$ or energy loss rate of particles (here protons) as they travel through matter. It encodes the average energy loss per unit distance[NZ15].

$$\frac{S(E)}{\rho} = -\frac{\delta E}{\rho \delta x} \tag{2.3}$$

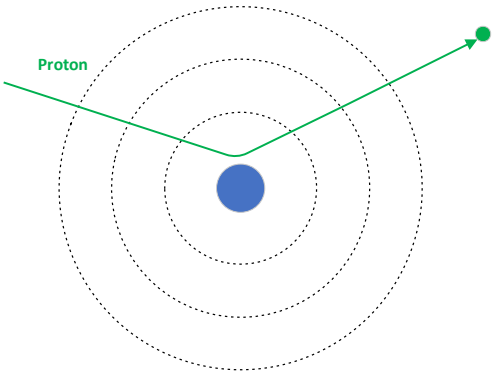
With ρ denoting the mass density, E denoting the particle's energy in eV and x denoting the range in the particle's path. Because this thesis will use water as the reference frame for all subsequent calculations, ρ can be omitted as it behaves like a constant.

$$S(E) = -\frac{\delta E}{\delta x} \tag{2.4}$$

Several models have been developed to calculate this value. For protons, two of the most frequent approaches are presented in sections 2.2.1 and 2.2.2.

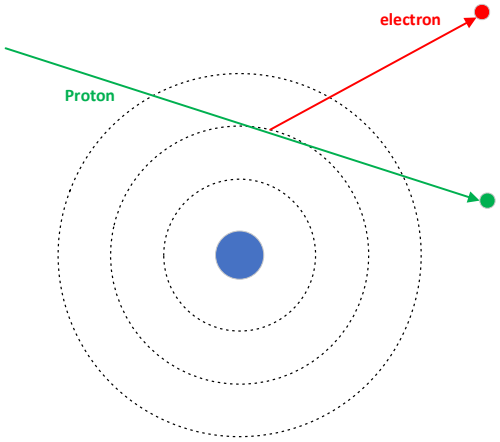
Described in the following are the three main nuclear interaction types of ionizing proton radiation:

Elastic Coulomb Scattering



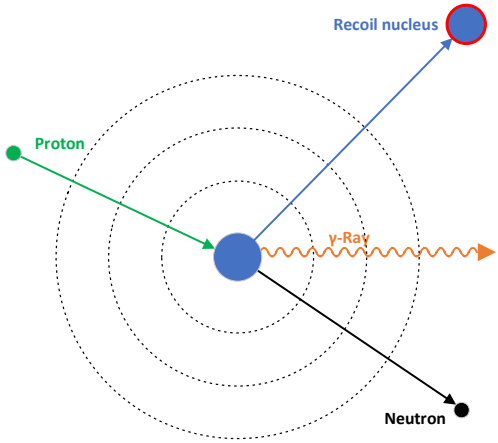
Elastic Coulomb scattering describes a process whereby a incident proton is scattered at an angle, while passing near an atomic nucleus. Due to the proton's positive charge, it is scattered by the positive charge and much higher mass of the nucleus[NZ15]. No significant absorption or attenuation occurs, the scattered proton continues to travel in its new direction[NZ15].

Inelastic Coulomb Scattering



If an incident proton hits an orbital electron along its path, the electron is ejected from its orbit. Due to the proton's mass, which is 1832 electron masses, the proton is not scattered (an important attribute for radiation therapy) and a small loss in its kinetic energy occurs[NZ15]. This process is the main contributor to limiting the proton's range according to its initial energy. Most electrons ejected in this process are absorbed locally in a radius less than 1mm[NZ15].

Nuclear Scattering



Nuclear scattering occurs if an incident proton directly hits a nucleus and its kinetic energy is higher than the energy deflecting equal charged particles. The proton enters the nucleus and either one or more neutrons or a proton, deuteron, triton or heavier ions (here described as recoil nucleus) are emitted, also generating γ-Rays[NZ15].

Tab. 2.3: Overview of nuclear interaction types of ionizing proton radiation occurring in radiation therapy.

In contrast to ionizing photon radiation, ionizing hadron radiation (here protons) is categorized as direct ionizing radiation, as collision with the incident beam's protons causes ejection of the absorber medium's oribtal electrons. Protons, like other hadrons, possess a resting mass. For protons it is 1832 electron masses, carrying a positive charge due to their quark configuration[NZ15]. The advantage of proton radiation over photon radiation comes from both the three interaction types as listed above, due to their mass and charge.
Due to their resting mass, protons of an incident beam carry kinetic energy, which they mainly lose via inelastic Coulomb scattering as this is the interaction type occurring most frequently and directly influencing the depth at which the Bragg peak (see figure 2.4) occurs[NZ15], their mass is also responsible for their travel path, i.e. they aren't deflected significantly during collisions with orbital electrons[NZ15]. However, cumulative deflections can alter the path significantly. The Bragg peak is the result of the energy transferred during the nuclear interactions being inversely proportional to the velocity of the proton. The more the proton slows down, the more energy it transfers per unit path length, the more energy is absorbed by the target medium/tissue per unit path length, i.e. its interaction cross-section (the probability of having interactions per unit area) increases as it slows down.[Pag17].
Protons can undergo nuclear scattering, which occurs less frequently and mostly in the entry region of an absorbing medium[Pag17], producing secondary particles. Lastly, protons can also undergo elastic Coulomb scattering but this interaction doesn't have a significant ionizing effect but contributes to the alteration of a proton's path.
1% of the protons experience a nuclear interaction event per 1cm range in water[NZ15]. This is important to consider for the Bragg peak. One single proton may have a very sharp Bragg peak slightly before or after the target dose due to the interactions, this is called range straggling. However, when considering a beam consisting of many protons, all having statistically slightly different ranges (range straggling), results in the Bragg peak with a width of a few millimeters[NZ15].

Two main approaches to model ranges in terms of stopping power of mono-energetic proton radiation and the corresponding Bragg curve(see fig 2.4) can be commonly found in literature: The Bragg-Kleeman model and more exact models based on the Bethe-Bloch equation.

2.2.1 Bethe-Bloch Equation

The **Bethe-Bloch equation** describes the quantum-relativistic model of hadron movement through an absorbing medium, modelling the Bragg curve as seen in figure 2.4. This equation is an exact but not a closed-form expression, i.e., it requires numerical methods to solve[Mar+19]. One of the reasons for this complexity of the model lies

in the relativistic behavior of the movement of electrons, which macroscopically can be described by an ellipsoidal orbit around an atom but microscopically adhere to an uncertainty in their exact position[KG14]. The relativistic version of the Bethe-Bloch Equation as reported by [GWP17] is:

$$-\frac{\delta E}{\delta x} = \frac{4\pi n z^2}{m_e c^2 \beta^2}\left(\frac{e^2}{4\pi\epsilon_0}\right)\left(ln\left(\frac{2m_e c^2\beta^2}{I(1-\beta^2)}\right) - \beta^2\right) \tag{2.5}$$

With n denoting the electron density of the absorbing material, e denoting the electron charge, m_e denoting the electron mass, I denoting the mean excitation potential of the target material, z denoting the multiple of the electron charge of the projectile (proton), $\beta = \frac{u}{c}$ being the ratio of the projectile velocity u and the speed of light in a vacuum c. ϵ_0 encodes the vacuum permittivity.

Several approaches to approximate the Bethe-Bloch equation have been developed. The main trade-off of each approach lying between computation time and precision. A more popular approach to approximating the equation is to utilize a series approximation. Ulmer et al. have developed a series of exponential terms to approximate the equation (mainly for range calculations), with corresponding parameters fitted to the Bragg curve.[UM10]. Grimes et al. have shifted the "non-closed form problem" by reformulating the equation using the inverse exponential integral function, which itself is a non-closed form expression but is tabularized or can be easily approximated, e.g. via $ln(xln(xln(xknx)))$[GWP17]. Martinez et al. have built on this and developed a Taylor series-based approximation of the Bethe-Bloch equation to calculate the stopping power[Mar+19]. The approach by Ulmer et al. will be used and is discussed in section 3.5.4.

2.2.2 Bragg-Kleeman

The **Bragg-Kleeman** model is a phenomenological model, based two empirical parameters, α and P, fitted such that the equation models the Bragg curve as close as possible, depending on the material traversed[Pet+18]. The exponent P is a scalar value, depending on the proton energy and α approximates the square root of the effective atomic mass of the absorbing medium the hadron (here proton) traverses[Bor97].

$$R - x = \alpha E(x)^P \tag{2.6}$$

Where R denotes the residual range after having traversed to x, denoting a point on the traversed path from which the residual range should be calculated from. Of special interest in this thesis is the residual range from the beginning, i.e. $x = 0$. $E(x)$ denotes the energy of the proton at point x.

Contrary to the Bethe-Bloch Equation, this model is not only a closed-form expression but an analytical one. However, this comes with the cost of being imprecise by definition as it is an empirical model of the Bragg curve/hadron stopping powers modelling mainly non-elastic nuclear interactions of hadrons (in this case protons). Incidentally, this is exactly the main nuclear interaction of protons as they collide with orbital electrons of atoms in the tissues[NZ15].
As the focus of this work also lies on computation times, Bragg-Kleeman has been chosen as the underlying model for calculations. The main disadvantage of this models lies in the two parameters α and P and the related imprecisions. However, a lot of work has been done to fit those parameters and, so far, a clear trend can be observed: For water, P seems to converge around 1.735 and α can be computed from P or taken from various stopping power tables, like SRIM[ZZB10] or PSTAR, the latter of which is based on the official ICRU reported values and methods(ICRU Report 49)[Ber+93].

$$R - x = \alpha E^P \Leftrightarrow E(x) = \left(\frac{R-x}{\alpha}\right)^{\frac{1}{P}} \tag{2.7}$$

Differentiating the energy E with respect to x yields the stopping power $-\frac{\delta E}{\delta x}$.

$$-\frac{\delta E}{\delta x} = \frac{E(x)^{1-P}}{\alpha P} \Rightarrow \frac{(R-x)^{\frac{1-P}{P}}}{\alpha^{\frac{1}{P}} P} \tag{2.8}$$

Several works have been published to fit α and P for different materials. Of special interest for this thesis is water as reference medium as everything is transformed with respect to water to have a common metric space from which to calculate from. Since water is also a very good tissue equivalent, many methods exist to convert from one medium to another, with the usual trade-off between precision and computation time[NZ15].

Model	α in $\frac{MeV}{cm}$	P
Bortfeld 1997[Bor97]	0.0022	1.77
Boon 1998[Pet+18]	0.00256	1.77
Newhauser et al. 2015[NZ15]	0.002633	1.735
Pettersen et al. 2018[Pet+18]	0.00262	1.736

Tab. 2.4: A selection of fitted values for α and P for water. Valid for the proton energy range of 50 to 300 MeV (therapeutic range).

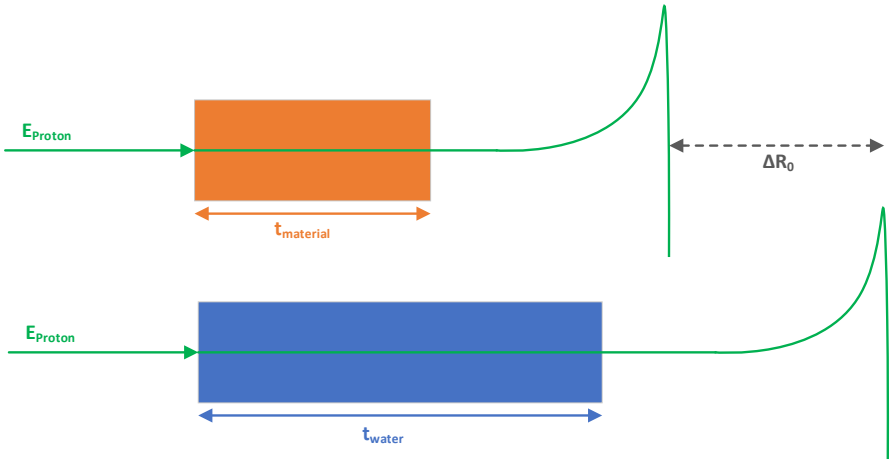


Fig. 2.5: Visualization of the water equivalent thickness t_{water} and a corresponding material $t_{material}$ for proton radiation. The green line models the dose curve for protons as seen in figure 2.4. t_{water} scales accordingly depending on the electron density/thickness of the material traversed, also affecting ΔR_0.

2.2.3 Water Equivalent Thickness And Composite Materials

The (electron) density of the absorbing target material/tissue of a proton beam in radiation therapy directly affects the kinetic energy required for the proton to reach a certain depth, i.e. the frequency at which inelastic Coulomb scattering occurs[NZ15]. As humans do not consist of homogeneous tissue but rather of different types of tissues (skin, muscles, fat, organs, bones, ...) with different material properties, like density, it is an important factor to consider. This thesis uses water as a common calculation space on which other calculations are based on. This requires a transformation of material thicknesses for every material type in a proton beam's path into thicknesses relative to water and the derivation of the relative stopping powers, i.e. the thickness of a material $t_{material}$ and it's stopping power per unit length is transformed into a corresponding thickness t_{water} and its according stopping power relative to water[NZ15]; this is visualized in figure 2.5. This also allows calculation of the range of a proton in composite tissues via the summation of the relative stopping powers of all the tissues in the proton beam's path[NZ15]:

$$R(E) = \int_0^{E_0} \left(\frac{\delta E}{\delta x}_i\right)^{-1} \delta E \quad \approx \sum_0^{E_0} \left(\frac{\delta E}{\delta x}_i\right)^{-1} \Delta E \tag{2.9}$$

Where $-\frac{\delta E}{\delta x}_i$ is the relative stopping power of the i-th material in the beam's path. To calculate the range, the reciprocal is required (because the function of E according to distance x is required) and integrated to yield the distance/range travelled.

The discrete summation is a good enough approximation of the range in clinical contexts[NZ15] and does not require an integrator, like Runge-Kutta or Verlet integration, speeding up calculations.

2.3 Proton Radiation Therapy

As documented in figure 2.8, (proton) radiation therapy is comprised of several sequential steps from the initial diagnostic work to the actual treatment of the patient. The first step is also the most important one: Applying various imaging techniques to determine location, extent and scale of the tumor as well as the situation of its surroundings. The accuracy of the treatment parameters directly rely on the accuracy of the imaging techniques used. CT scans are the most frequently used imaging technique to determine spatial parameters. If necessary, other imaging modalities are used in tandem, like MRI or ultrasound-based imaging[KG14].
After imaging, the collected spatial data is used to segment and register the tumor volume and surrounding volumes of interest or organs that shouldn't be hit by radiation. Then, together with the diagnosis, clinical objectives are defined, i.e. how much dose should the tumor be hit with, if radiation therapy should be used and what type of radiation modality should be used (depending on the tumor position), additional medication that should be administered for chemotherapy, etc.
Based on the clinical objectives and the imaging data, a treatment plan is generated, also with the help of a treatment planning system (TPS), which is a collection of methods and algorithms situated in a pipeline to calculate patient dosimetry based on input parameters (see section 2.3.3). The plan also incorporates how much energy is required for the radiation modality used to achieve the defined dose and what (and how many) beam angles to use and how many fractions/sessions at what interval the patient needs to undergo.[KG14]
Lastly, the generated plan is verified by medical personal and computer programs (usually part of a TPS) and evaluated against the defined clinical objectives for the treatment. If the clinical objectives are achieved by the current plan, patient treatment can begin. Otherwise a new plan needs to be generated.

2.3.1 CT Scans

The basis for all further treatment modalities and parameters, including beam angles, is accurate patient data. For radiation treatment planning, spatial and visual data of the tumor volume, its location and situation, segmentation and registration of it and surrounding volumes of interest (e.g. organs-at-risk) constitute the foundations on which all further calculations are performed on[KG14].
There are different imaging modalities to acquire such data. For 3-dimensional/volumetric data, computed tomography (CT) and magnetic resonance imaging (MRI) are the most commonly chosen ones[KG14]. This thesis focuses on the former as CT data sets are readily available and many decisions made can be directly derived from the structure of the CT data. Contouring and segmentation has been and still is usually done

manually by an expert, using patient CT and/or MRI data[KG14]. However, the application of convolutional neural networks to partially or fully automate these tasks has already been the focus of research and is still ongoing[Lus+18][Don+19][Li+20]. A CT scan effectively consists of a grid of voxels with resolutions up to 0.5mm for each axis[Can17].

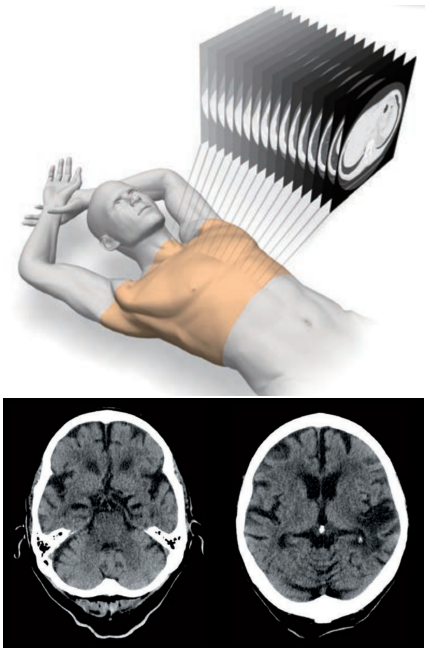


Fig. 2.6: Top: A CT scan consists of slices of volumetric data; here slices along the longitudinal axis.
Bottom: A sample slice of a CT scan of a patient's head. Images taken from [Can17].

Commonly, such as in CT scans done by a tunnel or gantry scanner, the three canonical axes are used, where one slice forms a 2-dimensional grid of voxels along the chosen height- and width-axes with a fixed slice depth. A CT scan data set consists sequence of slices towards the chosen depth-axis. For one slice, photon radiation within the x-ray energy spectrum is emitted in a cone-like field towards to target volume, with the emitter doing one or more revolutions. Depending on the density of the tissue traversed, the x-ray beams are attenuated differently and hit the corresponding diametrically opposed detectors in the tube with different residual energies, forming a set of attenuation coefficients[KG14]. This is done either slice-by-slice or in a continuous movement, where the emitter effectively follows a helical path along the tube. The attenuation coefficients are then used to calculate a visual representation for each voxel, called "CT numbers", which are then normalized against water attenuation to form Hounsfield units (HU), according to the formula reported by Khan et al.[KG14]:

$$HU = \frac{\mu_{tissue} - \mu_{water}}{\mu_{water}} \times 1000 \tag{2.10}$$

with μ denoting the attenuation coefficient for a corresponding material. These Hounsfield units are then used to calculate stopping power values for each voxel, usually via Hounsfield lookup tables (HLUT)[Woh+17]. However, CT scans and the corresponding conversion into stopping powers suffer from inaccuracies, which can significantly impact accuracy of treatment planning[Woh+17]. These range from inaccuracies due to patient positioning changes, patient movement during scanning (even respiratory movement is a factor for lung-related treatment[Sar+06]) to inaccuracies due to different tissue heterogeneities along the path of an x-ray resulting in the same attenuation measured by a detector for different tissue compositions[Woh+17]. To counter the latter issue, Wohlfahrt et al. have suggested a

dual-energy approach to CT scans (DECT), effectively using two different energy ranges of x-rays and then calculate the difference between attenuation of these x-rays along the same path to determine tissue heterogeneities otherwise not seen by the single-energy approach[Woh+17].

2.3.2 Geometric Volume Concepts

After acquisition of spatial data of the treatment region and subsequent image registration, contouring and segmentation of volumes of interest (VOI) is done[KG14], either manually by an expert or (partially) automatic via convolutional neural networks[JKH19] and/or edge (like Canny edge detection, Houghs transforms or watershed algorithms) or feature detection algorithms (like SIFT, ORB or Harris-Corner-Detection). These concepts are then used as input for further processing in treatment planning in order to determine possible treatment modalities and parameters, including the determination of possible beam angles. Figure 2.7 shows a schematic representation of the commonly used clinical and geometric volume concepts as defined by ICRU Reports 50[Lan+93] and 62[Lan+99].

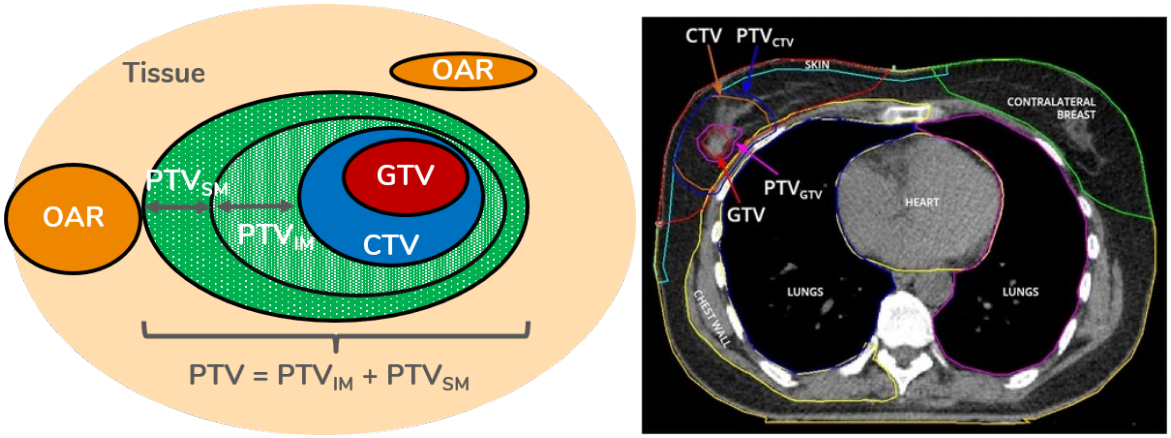


Fig. 2.7: Left: Schematic overview of volume concepts.
Right: Example of segmented volumes in CT scan slice (longitudinal axis) of the upper torso region.

- **Gross Tumor Volume (GTV):** Visible extents of the tumor volume after image registration. This is a clinical concept, independent of treatment modalities.

- **Clinical Target Volume (CTV):** GTV + subclinical, i.e. not directly visible, malignant tissue at certain probability levels depending on the tumor tissue and location. This is a clinical concept, independent of treatment modalities.

- **Planned Target Volume (PTV):** The union of two subsets/volumes: PTV_{IM} and PTV_{SM}. PTV_{IM}, also referred to as Internal Target Volume (ITV), is

defined as CTV + internal margins for motion of the CTV, depending on its location and situation, to compensate for inaccuracies. PTV_{SM} is defined as PTV_{IM} + heuristical uncertainty margins according to treatment modalities. Around 2.5% to 3.5% of volume extents are added for pencil beam scanning used in intensity-modulated proton therapy (IMPT) to account for range straggling and statistical spread of protons due to Coloumb interactions[Tho06].

- **Organs-At-Risk (OAR):** Regions or organs that are defined to be the least desirable to hit or to completely avoid to hit, i.e. region or organs that have a clinically defined penalty associated with them to avoid intersection with radiation beam paths. The extents of OARs is dependent of the beam modality used; for proton beams, lateral and angular spread of the beam is taken into account[Tho06].

2.3.3 Treatment Planning Systems

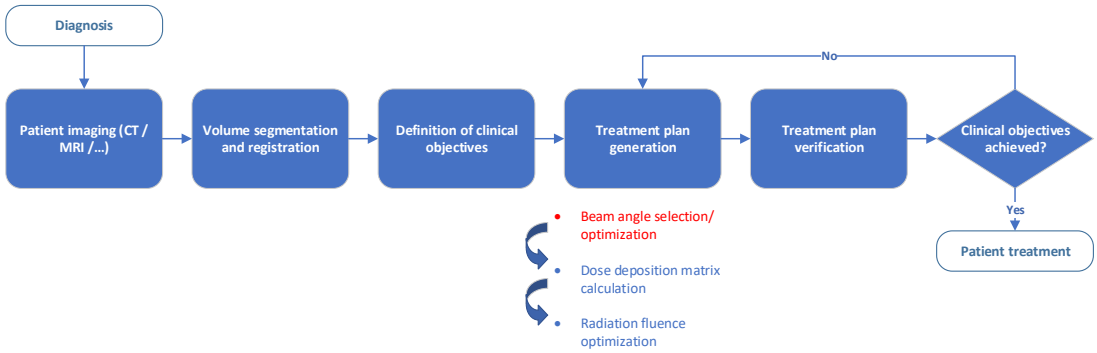


Fig. 2.8: High-level overview of the principal steps included in radiation therapy, with beam angle selection/optimization (marked in red) being the step this thesis focuses on.

Treatment planning systems (TPS) are an integral part of radiation therapy in the current medical context and describe a pipeline of algorithms and methods to compute/simulate radiation dosimetry on a patient's tumor and surrounding tissues while applying clinical and physical constraints.
Since the initial proposition of utilizing protons as beam modality to treat cancer in 1946[Tia+18], advancements have been made in developing methods and algorithms for radiation therapy, predominantly for photon-based radiation.
During the early 90s till earlier 2000s, research and development of proton therapy in general and TPS in particular experienced rapid advances, which were made in tandem with the technological development of computers, spearheaded by dedicated medical research facilities and specialized clinics. The current landscape of TPS and their related research and development has largely been shifted to commercial closed-source systems, naturally only with very limited open documentation on algorithms and methods used[BJB18]. Among the more well-known manufacturers of closed-source TPS are Elektra, Philips, RaySearch and Varian [ITN13], whereas among the open-source solutions, Slicer3D with the SlicerRT-plugin[Pin+12] for radiation therapy is one popular tool. Most other treatment planning systems, or certain subsets thereof, are implemented based on Matlab, and are primarily research and educational tools. Among open-source TPS, FoCa[Sán+14], CERR[DBC03] and DKFZ's matRad[Wie+17] (used in this thesis) are the more popular and still active choices.

After the acquisition of treatment relevant volumetric patient data, treatment parameters are defined according to clinical objectives and input into a TPS. A TPS can be categorized by how its pipeline is laid out: Forward planning or inverse planning[KG14]. The former requires all relevant parameters (desired dose levels, radiation modality, beam delivery modality, radiation energy levels, beam angles, ...) to be known and input beforehand in order to simulate patient dosimetry[KG14].

The latter can infer or suggest at least one of these parameters from a smaller or incomplete set of input parameters[KG14] within the constraints given by the clinical objectives, which, from an abstract point of view, is essentially solving a constraint satisfaction problem. This thesis focuses on beam angles to be inferred in a fast manner.

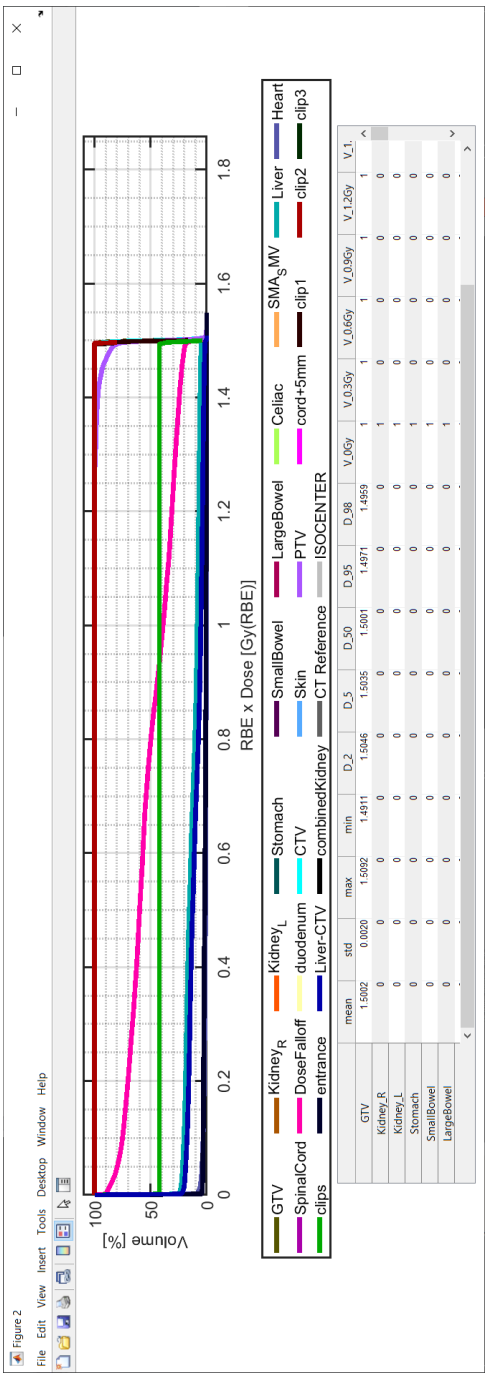


Fig. 2.9: Dose volume histogram (DVH) produced by matRad for the dosimetry in the example shown in figure 2.10.

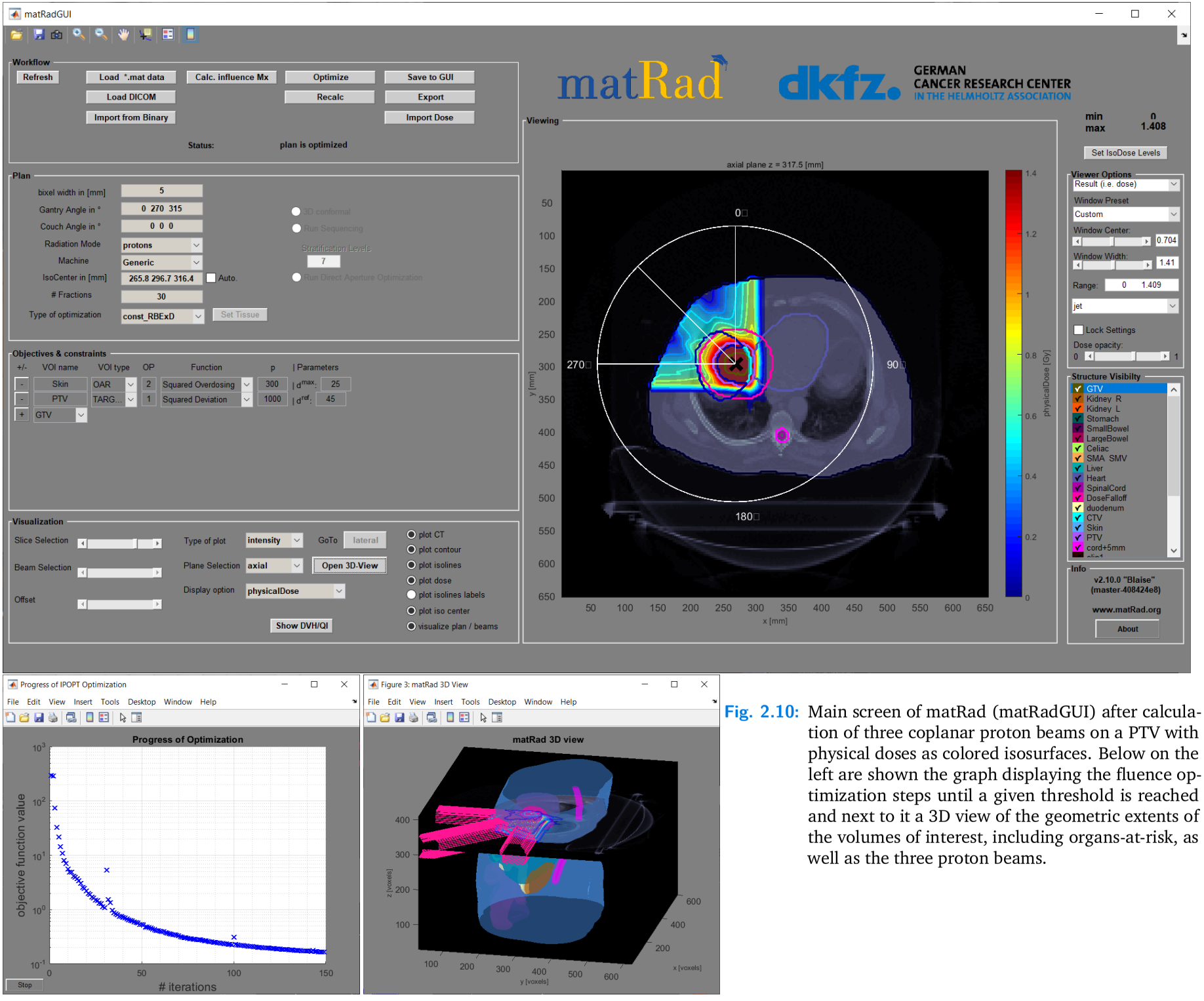


Fig. 2.10: Main screen of matRad (matRadGUI) after calculation of three coplanar proton beams on a PTV with physical doses as colored isosurfaces. Below on the left are shown the graph displaying the fluence optimization steps until a given threshold is reached and next to it a 3D view of the geometric extents of the volumes of interest, including organs-at-risk, as well as the three proton beams.

2.4 Mathematical Optimization And Metaheuristics

An optimization problem has in its most basic form the following structure:

$$\begin{aligned} &\text{minimize } f(x) \\ &\text{subject to } \{g_1(x)\square_1 b_1, \\ &\qquad g_2(x)\square_2 b_2, \\ &\qquad \ldots, \\ &\qquad g_i(x)\square_i b_i, \\ &\qquad \ldots, \\ &\qquad g_m(x)\square_m b_m, \} \end{aligned} \tag{2.11}$$

where $\square_i \in \{<, \leq, =, \geq, >\}$, $i = 1, ..., m$.

With $f(x)$ being referred to as the "objective function" and $g_i(x)\square_i b_i$ referred to as constraints with b_i being "bounds" or "boundaries". Note that $f(x)$ can also be maximized since maximize $f(x) \Leftrightarrow$ minimize $-f(x)$. In words: "*Minimize/Maximize (i.e. get the lowest/highest possible objective function score) $f(x)$ such that the constraints $g_i(x)\square_i b_i$ are still valid.*" According to Stork et al., any optimization algorithm attempting to solve such a problem, or variations thereof, can be decomposed into four major parts: Initialization, generation, selection and control (control being optional, depending on the type of optimization algorithm applied)[SEB20].

Algorithm 2: Basic structure of an optimization algorithm according to Stork et al.[SEB20]

```
1  set initial control parameters;
2  t = 0;
3  initialize candidates;
4  evaluate initialize candidates;
5  while termination condition is not reached do
6      t = t + 1;
7      generate new candidates;
8      evaluate newly generated candidates;
9      select solution(s) for the next iteration;
10     optional: update control parameters;
```

2.4.1 The 0/1 Multidimensional Knapsack Problem

A subclass of optimization problems are known as combinatorial optimization problems. And arguably the most popular problem type of that class is the in(famous) "Knapsack Problem", first formally described in 1896 by Mathews[Mat96]. Informally, true to its given name, the objective is to pack a knapsack with as many items p_j as possible while still keeping the constraint valid; in this example, a constraint could be the weight or size a_j for each corresponding item p_j, x_j would then serves as a "selector", as the value of x_j effectively determines whether item p_j is put into the bag ($x_j \in [0, 1]$) and subsequently the corresponding a_j is considered. Formally, the knapsack problem is described below on the left-hand side.
On the right-hand side, a variation of the knapsack problem can be seen, known as the "0/1 Multidimensional Knapsack Problem" (0/1-MKP)[Laa+18]. The main difference is that the number of allowed constraints is relaxed; it is arbitrary (i.e. "multidimensional"), in contrast to just one constraint for the regular knapsack problem. Note that in the 0/1-MKP, the "selector" values are constraint to $x_j \in \{0, 1\}$. The 0/1-MKP is also one example of a NP-hard problem[Laa+18], just as the regular knapsack problem[Cac+22].

$$\text{maximize} \sum_{j=1}^{n} p_j x_j$$

$$\text{subject to} \sum_{j=1}^{n} a_j x_j \square b,$$

where $\square \in \{<, \leq, =, \geq, >\}$, $x_j \in \{0, 1\}$ and $j = 1, ..., n$.

$$\text{maximize} \sum_{j=1}^{n} p_j x_j$$

$$\text{subject to } \{ \sum_{j=1}^{n} a_{i1} x_j \square_1 b_1,$$

$$\sum_{j=1}^{n} a_{i2} x_j \square_2 b_2,$$

$$...,$$

$$\sum_{j=1}^{n} a_{ij} x_j \square_i b_i,$$

$$...,$$

$$\sum_{j=1}^{n} a_{mj} x_j \square_m b_m \}$$

where $\square_i \in \{<, \leq, =, \geq, >\}$, $x_j \in \{0, 1\}$, $i = 1, ..., m$ and $j = 1, ..., n$.

2.4.2 Metaheuristics And Teaching-Learning-Based Optimization

Solving optimization problems of any kind is a large and popular area of research with wide application domains. Just the above introduced 0/1-MKP can be found

within the area of operations research regarding scheduling problems or loading problems, in the financial industry regarding stock-related problems or capital budgeting, or within the more familiar area of scheduling processes for multi-core processors[Laa+18], to name a few. In fact, Laabadi et al. recalled a study carried out in 1998 wherein the MKP was found to be the 18th most popular and the 4th most needed problem[Laa+18].

As many optimization problems, including combinatorial optimization problems such as the knapsack problem, fall into NP, solving them is not trivial. And due to variety of actual use cases, a lot of research over the centuries yielded a lot of different techniques. At the most basic level, optimization algorithms fall into two broad categories: Exact and heuristical approaches, where the latter do not guarantee to always obtain the globally optimal (i.e. exact) solution but attempt to converge towards an (global) optimum that can be considered "good enough".

The gain for the sacrifice of global optima as solution(s) heuristical optimization algorithms make is requiring less computational effort (relatively, of course. It is still NP) or even actual computability[SG13] in comparison to exact optimization algorithms. Bluntly put, heuristic approaches tackle "harder" problems that may not be computable otherwise. Optimization algorithms utilizing heuristics and/or a stochastic approach are often categorized as "Metaheuristics". Sorensen et al. have a succinct definition: "A metaheuristic is a high-level problem-independent algorithmic framework that provides a set of guidelines or strategies to develop heuristic optimization algorithms"[SG13, Chapter 1, p. 1].

Stork et al. have recently described a new taxonomy of currently relevant optimization algorithms by using the underlying techniques and methods these algorithms apply as criteria for categorization, depicted in figure 2.11.

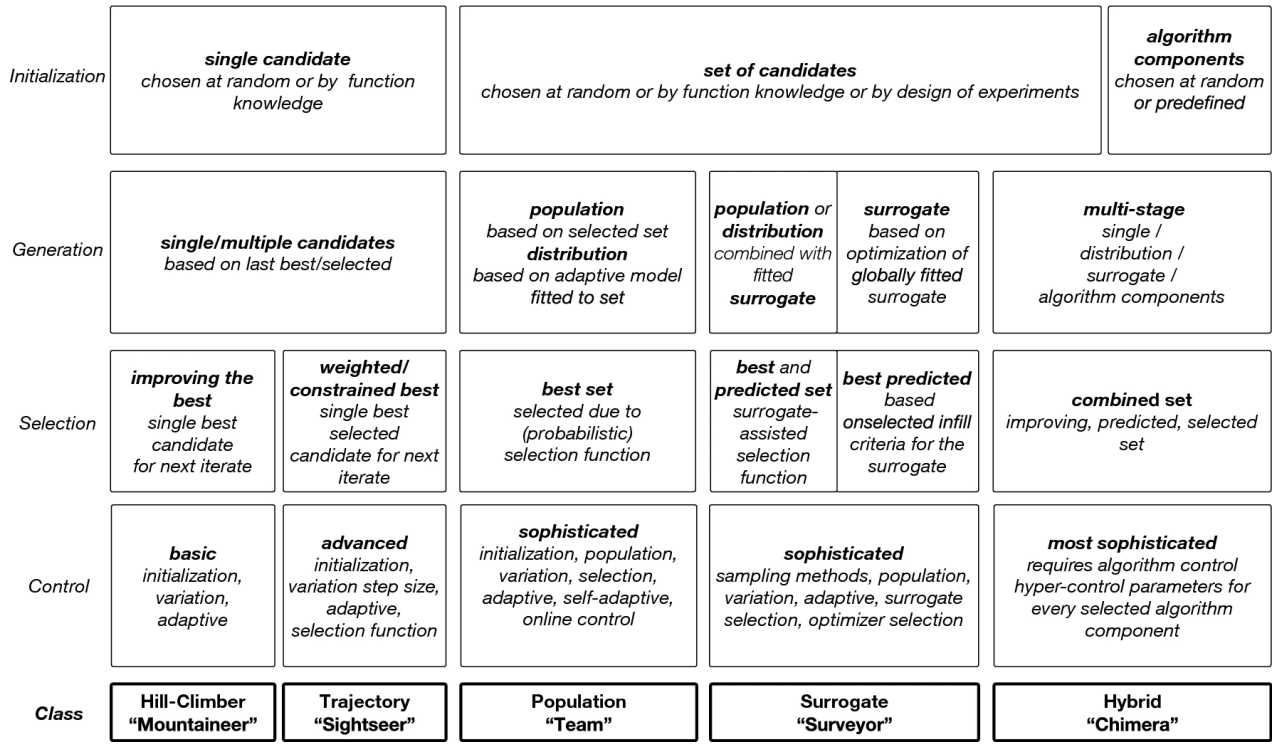


Fig. 2.11: Stork et al's taxonomy of optimization algortihms and metaheuristics categorized in classes according the respective techniques used. Taken from [SEB20].

Notable examples of metaheuristics include genetic/evolutionary algorithms, tabu search, simulated annealing, and ant colony optimization, among others[SG13][SEB20]. However, many of these metaheuristics require, in addition to the actual problem input, control parameters to control various (converging) behaviours, as stated in the beginning of this chapter. And for metaheuristics, they can be considered to be even more important than for exact algorithms.

Among the main sources of error and sub-optimally converging results to the desired optimum, are badly tuned (control) parameters. In fact, there is dedicated sub-branch of research dedicated to control parameters, the tuning and "optimization" thereof[]. This fact and the relatively easy implementation are the main reasons why a class of metaheuristics has been chosen, to be used in the system developed in this thesis, able to work without or only with a minimal amount of control parameters. Additionally, to eliminate possible bad results due to the inexperience of the author in the application field (proton radiation therapy), general badly tuned parameters and to make the whole system more robust for a variety of application domains.

Going by Stork et al.'s taxonomy (see figure 2.11), one fairly recent representative of the class of population-based metaheuristics is known as "Teaching-Learning-based Optimization" (TLBO) by Rao et al.[RSV11]. More specifically, a variation on it by Kern et al. has been chosen as basis for this thesis' system[KLV20] and only requires parameters that all other metaheuristics require at minimum as well: Termination criterion and population size. Kern et al. have utilized some techniques mainly found

in evolutionary algorithms, like Chu et al's version of the genetic algorithm[CB98], to control the candidate generation, thereby achieving better convergence towards the global optimum:

Algorithm 3: The TLBO variation as described by Kern et al.[KLV20].

```
1  g = 0;
2  X = initialize population(size_pop);
3  evaluate(X);
4  repeat
5      X = sort(X);
       // from minimum objective function score to maximum
6      for i = 1 to size_pop do
           // Teacher Phase
7          t_f = round to nearest integer(1 + rand(0, 1));
8          x_mean = x_{⌊|X|/2⌋} ∈ X;
9          x_teacher = x_1 ∈ X;
10         x_{i,new} = x_i + rand(0,1) · (x_teacher − (t_f · x_mean));// x_i ∈ X
11         evaluate(X);
           // If the objective function score of x_{i,new} is better than the score of
              x_i
12         if x_{i,new} > x_i then
13             x_i = x_{i,new};
           // LearnerPhase
14         ii = rand(1, size_pop){ii ≠ i};
15         if x_i > x_ii then
16             x_{i,new} = x_i + rand(0,1) · (x_ii − x_i);
17         else
18             x_{i,new} = x_i + rand(0,1) · (x_i − x_ii);
19         evaluate(X);
20         if x_{i,new} > x_i then
21             x_i = x_{i,new};
22     g = g + 1;
23 until g ≠ generations_max
```

The key idea behind TLBO, which also gave it its name, is that the population of candidates goes through two phases as the algorithm iterates to find better solution candidates (converges toward the optimum): The "Teacher-Phase" and the "Learner-Phase". The algorithm adheres to the structure documented in algorithm 2: Both phases can be found inside the main loop and both phases "generate" new candi-

dates. In the teacher-phase, the current best (i.e. highest objective score) candidate influences the generation of a new solution candidate, i.e. "teaches the class room". In the learner-phase, the overall mean of the population (i.e. the "students") is raised letting the "students teach each other", i.e. a candidate of the current iteration i and a randomly selected candidate are influencing the generation of a new candidate. Thus, the teacher-phase raises the mean quality of candidates by essentially "pulling the population" closer to the current teacher's objective score and can be considered to be a global search phase, while the learner-phase is akin to a local search, where "students"/candidates may or may not improve by using other "students"/candidates for the generation operation (i.e. to "learn from").
As already stated, these phases occur inside the main loop. But more specifically, as can be seen in algorithm 3, they happen in another loop (the for-loop), iterating i through the whole population size. Each loop has therefore an "active" candidate through the index i. The main loop iterates through a specified amount of "generations" and sorts the set of candidates before another for-loop iteration of teaching and learning begins, so that the teacher candidate is always up-to-date as well as the population mean. Kern et al's variant is special in the sense that it takes care that after what is coined an "transformation operation" (lines 11, 16 and 18) has taken place, newly generated candidates are "repaired". I.e. if they are invalid, they will be made valid by changing the corresponding x_j values seen in the definition of the MKP such that all constraints are valid again. These transformation operations are essentially identical in their structure:

$$x_{i,new} = \begin{cases} x_i + r \cdot (x_i - x_j), \text{ if } score(x_i) < score(x_j) \\ x_i + r \cdot (x_j - x_i), \text{ else} \end{cases} \tag{2.12}$$

With a special variation of it being in the teacher-phase (line 10). This operation generates a new candidate by adding to the selected candidate x_i a randomized version of the difference of it and another selected candidate x_j, the difference depending on whether the objective score of x_i or x_j is better. Note that the solution candidates are essentially n-dimensional vectors. A newly generated candidate can be invalid. Instead of trying to generate a new one until a valid one is found, it is "repaired" instead.
However, while Kern et al. describe approaches (and how to "binarize") to generating an initial population as well as the repair operator, they remain on a high-level and leave it to the reader to figure it out in detail. One of the contributions of this thesis will be methods for initial population generation, which doesn't resemble the high-level descriptions of Kern et al. at all anymore, and the parallelization of the repair and evaluation operations, utilizing the GPU. These are presented in detail in chapter 3.

3 System And Methods

This chapter gives an overview of the system that has been implemented for this thesis as well as details on certain aspects of the implementation and reasoning behind its structure. After giving an overview of the system as a whole, each major component will be described in detail and the rationale behind it.

3.1 Overview

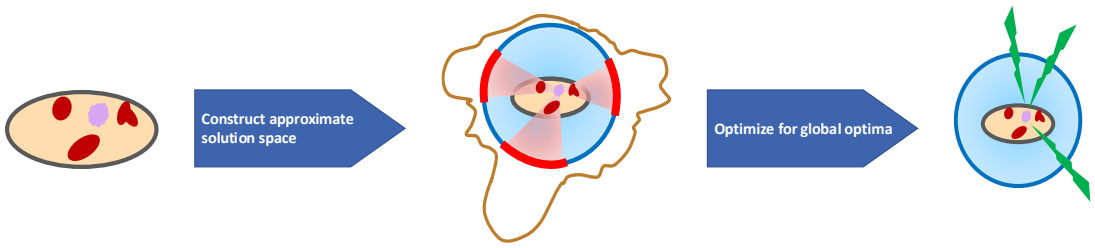


Fig. 3.1: High-level visualization of the main steps of the proposed ray path optimization pipeline. Beginning with the (patient) volume data including the target volume (in pink), the organs at risk (in red) and the rest of the surrounding tissue. After the construction of a solution space on which search for minima is performed, three beam angles have been suggested.

The system/method(s) developed in this thesis can be broadly categorized into two parts: Generation of a solution space and optimizing for a global optimum, utilizing the solution space (see figure 3.1. The generation of a solution space is more aptly termed "Cost Map" within the system and will be used henceforth. Roughly, the main task of the first part is to generate a space in which solutions can be generated in (similar to how an image of a function lies within the codomain) and later optimized. The solution space itself consists of accumulated relative path lengths from a voxel of interest inside a grid of voxels (henceforth called "Cuboid") to a corresponding surface voxel. I.e each surface voxel of a cuboid carries the relative path length value. The path lengths are relative in the sense that, depending on the beam model used (here protons) for ray tracing, material densities according to the beam model are accumulated along the ray, in an inside-out manner.
The last step of the first part is to transform the cuboid from a grid of voxels to a "hull" of voxels, i.e. removing every voxel that is not on the surface of the cuboid. This "hull" of voxels is henceforth called "Cuboid Surface" and serves as the input for

the second part. The main reasoning behind this transformation is to save memory, which can be quite the substantial amount (GBs to MBs).
The second part is taking the surface voxels with the accumulated relative path lengths to form solution candidates, with a solution essentially akin to the solution of the 0/1-multidimensional knapsack problem (0/1-MKP), i.e. $\overrightarrow{x} \in 0, 1^n$ (see 2.4.1). Then optimizing the generated candidates of the solution population such that a set of (global) minimal relative path lengths (according to the beam model) has been found. Lastly, these solutions are translated into a 4-tuple: $(E_0, \rho, \theta, \phi)$, denoting the inital energy E_0 (in MeV) required, and the spherical coordinates (with the voxel of interest as the center).

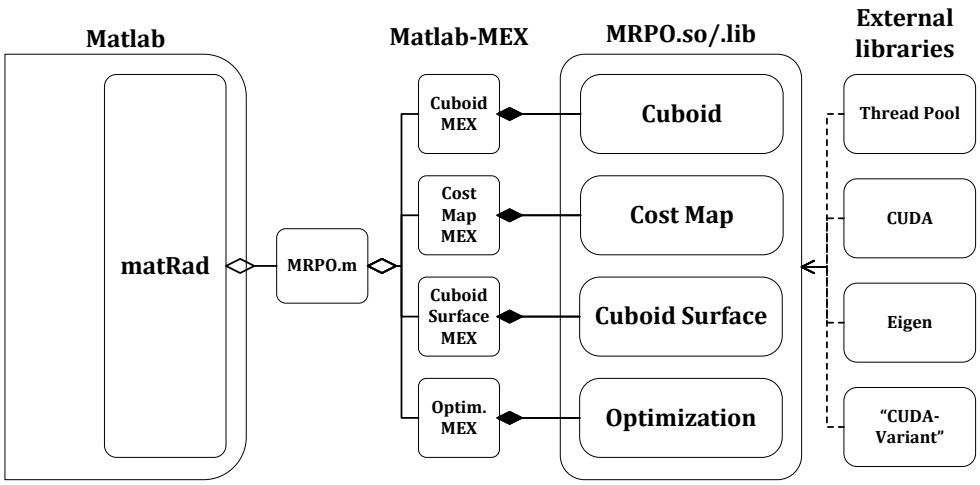


Fig. 3.2: High-level visualization of the system and its major components. Note that MEX is a specific Matlab format, utilizing a C++ Matlab library to allow for access to the Matlab C++ API(MRPO = Multi-Ray Path Optimization)

The system itself is largely implemented as a static library, being then loaded into Matlab using the MEX-API; an API to call in C++ code within Matlab's environment, using C++17 and CUDA (version 11.6, devices supporting at least compute capability 7[nVi22a] or higher). The major components as well as how they are connected to Matlab can be seen in figure 3.2. Each major component (Cuboid, Cost Map, Cuboid Surface, Optimization) is essentially isolated in the sense that they can be composed separately but may require data input from previous components. During development, other libraries than CUDA were used: "CUDA-Variant" by Bryan Cantanzaro, an nVidia employee[Can15], implementing a lighter version of std::variant, i.e. variant type (heterogeneous sum type), also usable on GPUs, the venerable "Eigen"-library used for linear algebra[GJ+10] and "Thread Pool", a library by Barak Shoshany for an easy-to-use thread pool implementation using std::thread behind the scenes[Sho21].
The MEX-files are separately compiled C++ components requiring Matlab as a

dependency. "MRPO" is the main Matlab file that has the actual Matlab-functions to call from within Matlab/matRad.

3.2 Input

The input data of the system developed in this thesis requires discussion as well. Matlab[MAT21] is the chosen "client" application for the system, due to matRad[Wie+17] being implemented in Matlab and it being the only real viable open-source and openly available treatment planning system (at least to the author's knowledge). Thus, the data is loaded from Matlab (and its formats) into the system. However, the data itself originates from a composed data format known as "DICOM"[Nat17], which has been established as one of the common data exchange formats within the medical community.
Beside patient data and medical treatment information, it also allows storing of CT data as well as meta data related to patient geometry, which will be the main source for the system's input. More specifically the CT data, already converted into Hounsfield units (see section 2.3.1). Also a penalty map, which is an overlay the same CT grid, giving each voxel a penalty determined by an expert. The penalty serves to include a medical perspective into the radiation planning, i.e. giving a medical reason as to why a certain volume of voxels should be avoided, although from a purely physical point of view, an optimal path might be considered going through them.
Additionally, other relevant inputs from the DICOM-file are the origin coordinate of the CT, the coordinates for each voxel, the voxel dimensions (a CT can be an non-uniform grid!), the number of voxels for each dimension and contour or volume information, i.e. the PTV, the OARs, etc., together with the isocenters of these geometric volumes.

The scheme described in algorithm 4 outlines the steps with the DICOM-related structs for input. The structs of interest from the DICOM-file that Matlab/matRad extracts are the CT struct and the CST struct, containing the data described before.

Algorithm 4: High-level description of steps inside Matlab/matRad with inputs originating from a DICOM-file

load 'path/to/dataset (DICOM)' `// Generates CT and CST Matlab-structs`
... `// other matRad related steps`
C = Cuboid(CT, CST, ...)
C_{map} = costMap(C, $\mathbf{v}_{\mathbf{of\_interest}}$)
C_s = CuboidSurface(C_{map})
$[(E_0, \rho, \theta, \phi)]$ = OptimizeSolutions(C_s, $size_pop$, ...)
... `// matRad now uses the suggested beam angles for further calculation`

Note that, according to the DICOM-standard, the origin of a CT is the center point of the upper left voxel and the data is laid out in row-major order: From left to right, from top to bottom, from first acquired slice to the last (z-axis). Each voxel coordinate also is that of the voxel's center.

3.3 Cuboid And Cuboid Surface

In order to work with data given from the outside in a uniform and deterministic manner, it seems advisable to fit it into a common structure/format. This is the purpose of the existence of both the "Cuboid" and "Cuboid Surface" components. A cuboid object is the main input (alongside other parameters) for the calculation of the aforementioned solution space or "Cost Map", whereas a cuboid surface object, derived from a cuboid object, is the main input for the optimization component, as can be seen in algorithm 4.

A cuboid object is, in its essence, a 3D array, linearized into 1D, adhering to the common $i_{1D} = x + y * x_dim + z * dim_x * dim_y$ indexing scheme, where dim_x, dim_y and dim_z are denoting the number of voxels for each dimension respectively.

The more interesting object is a cuboid surface, warranting more detailed explanation. In its essence, a cuboid surface object is derived from a cuboid object by "hollowing it out", i.e. by only considering the hull voxels. The main reason for the existence of a cuboid surface object is two-fold: Firstly, the solution space is essentially only required to be a 2D-manifold, which the surface of a cuboid or any volume homeomorphic to a sphere is[BBT20, Chapter 1 and 2]. Secondly, consider a 3D voxel grid of size $m * n * k$ voxels. At worst, the required space complexity in

this example would be determined by the cuboid "side"/dimension with the most amount of cells, i.e. $max(m, , n, k)$. For the purpose of argument, let this be n. Then $O(n^3)$ would be the required space in the worst case. CTs can have a fairly high, i.e. sub-millimeter, resolutions[Can17]. Thereby resulting in a CT with a large amount of voxels. For the purpose of argument, let $n = 1024$. Then $1024^3 = 1073741824$ voxels á 8 bytes (if they each hold a double value) $\Rightarrow 1073741824 \cdot 8$ byte $\approx 8.59GB$. Since utilizing the GPU for parallelization, the available memory on the graphics card need to be considered. Although recent very high-end cards can store up to 24 GB of memory[Wal22], it should be apparent that memory constraints are a key factor. The solution space can be reduced to a 2D-manifold, i.e. the surface, going by the same argument: A cuboid surface needs to consider 6 sides, each side has at worst $n \cdot n \in O(n^2)$ space complexity, $\Rightarrow O(6 \cdot n^2) \in O(n^2)$. Using the same number $n = 1024$ from before: $1024^2 * 8 \approx 8.389MB$, a reduction of three orders of magnitude.

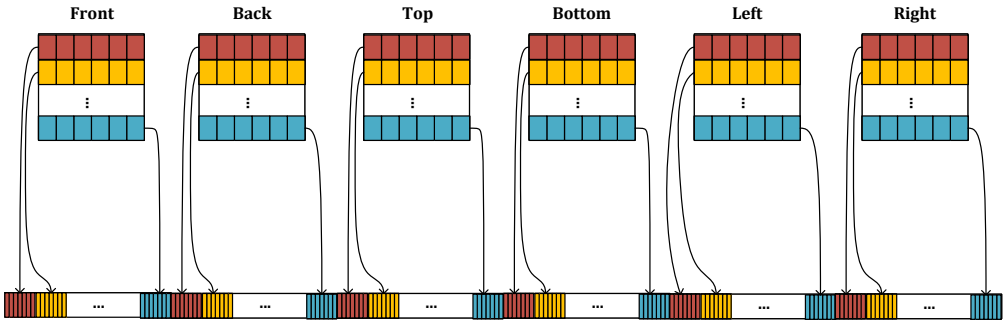


Fig. 3.3: A cuboid surface object consists of the hull voxels of a cuboid laid out sequentially as an array to reduce cache misses.

Figure 3.3 visualizes how the memory is laid out. It also helps to visualize that the process of deriving a cuboid surface object C_s from a cuboid C is simple: A thread pool of 6 threads, on for each face of the cuboid C, is created and each thread is assigned a face from which it copies the surface values into its specific section in a new array. For access via indexing, the common 3D-to-1D indexing scheme is not applicable anymore since in the terms of indices, a cuboid surface C_s is only C^0-continuous, since it is just a a direct sum of the surface voxels of each face:

$$\overrightarrow{f} = (f_1, f_2, ..., f_\alpha) \in \{c_{ijk} | c_{ijk} \in \mathbf{C} : i = 1, ..., dim_x(\mathbf{C}), j = 1, ..., dim_y(\mathbf{C}), k = 1\},$$
$$\alpha = dim_x(\mathbf{C}) \cdot dim_y(\mathbf{C}) \tag{3.1}$$

$$\overrightarrow{b} = (b_1, b_2, ..., b_\alpha) \in \{c_{ijk} | c_{ijk} \in \mathbf{C} : i = 1, ..., dim_x(\mathbf{C}), j = 1, ..., dim_y(\mathbf{C}), k = dim_z(\mathbf{C})\},$$
$$\alpha = dim_x(\mathbf{C}) \cdot dim_y(\mathbf{C}) \tag{3.2}$$

$$\overrightarrow{t} = (t_1, t_2, ..., t_\beta) \in \{c_{ijk} | c_{ijk} \in \mathbf{C} : i = 1, ..., dim_x(\mathbf{C}), j = 1, k = 1, ..., dim_z(\mathbf{C})\},$$
$$\beta = dim_x(\mathbf{C}) \cdot dim_z(\mathbf{C}) \tag{3.3}$$

$$\overrightarrow{o} = (o_1, o_2, ..., o_\beta) \in \{c_{ijk} | c_{ijk} \in \mathbf{C} : i = 1, ..., dim_x(\mathbf{C}), j = dim_y(\mathbf{C}), k = 1, ..., dim_z(\mathbf{C})\},$$
$$\beta = dim_x(\mathbf{C}) \cdot dim_z(\mathbf{C}) \tag{3.4}$$

$$\overrightarrow{l} = (l_1, l_2, ..., l_\gamma) \in \{c_{ijk} | c_{ijk} \in \mathbf{C} : i = 1, j = 1, ..., dim_y(\mathbf{C}), k = 1, ..., dim_z(\mathbf{C})\},$$
$$\gamma = dim_y(\mathbf{C}) \cdot dim_z(\mathbf{C}) \tag{3.5}$$

$$\overrightarrow{r} = (r_1, r_2, ..., r_\gamma) \in \{c_{ijk} | c_{ijk} \in \mathbf{C} : i = dim_x(\mathbf{C}), j = 1, ..., dim_y(\mathbf{C}), k = 1, ..., dim_z(\mathbf{C})\},$$
$$\gamma = dim_y(\mathbf{C}) \cdot dim_z(\mathbf{C}) \tag{3.6}$$

$$\Rightarrow \overrightarrow{C_s} = \overrightarrow{f} \oplus \overrightarrow{b} \oplus \overrightarrow{t} \oplus \overrightarrow{o} \oplus \overrightarrow{l} \oplus \overrightarrow{r} \tag{3.7}$$

A direct sum of vectors is strictly speaking an epimorphism from one n-dimensional (here $|C|$-dimensional) vector space over a field (typically $\mathbb{R}$) to one of different dimensions over the same field, in this case of fewer dimensions. $\overrightarrow{f}$, $\overrightarrow{b}$, $\overrightarrow{t}$, $\overrightarrow{o}$, $\overrightarrow{l}$, $\overrightarrow{r}$ denote the vectors holding the surface voxels for front, back, top, bottom ("o" was chosen for bottom in order to not clash with "b" for "back"), left and right respectively.

3.4 Cost Map Generation

The first step of the system is to utilize a beam model for tracing paths through a discretized volume, representing the whole scene, its sub-volumes (or objects) and the respective densities or material properties according to the model chosen. In the context of proton radiation therapy, these are the water-equivalent path length (WEPL) (see section 2.2.3) together with the constraints later given when optimizing for the optimal path length(s). The reasoning as to why the WEPL is used instead of the Bethe-Bloch equation will be discussed in this chapter, as well as the rationale behind the major steps taken for this component. The input is a cuboid object C, the voxel of interest (VOI) $\mathbf{voi} \in \mathbb{N}^3$, the dimensions of each voxel in mm $\mathbf{dim_{voxel}} \in \mathbb{R}^3$ and the number of voxels in each dimension $\mathbf{dim_{cuboid}} \in \mathbb{N}^3$.

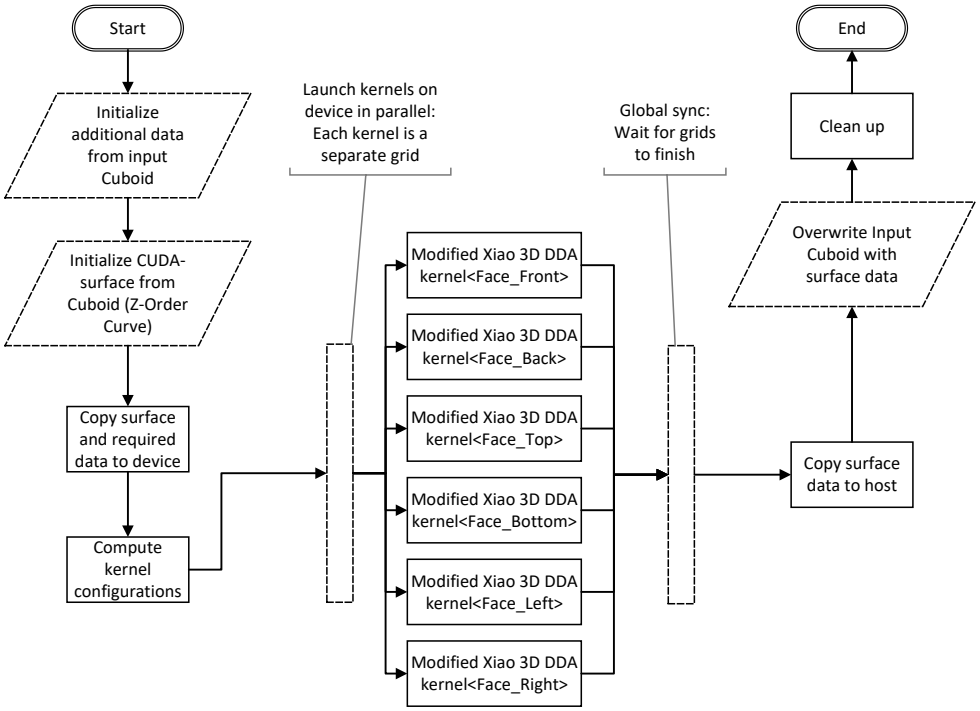


Fig. 3.4: Structure of the main component of cost map generation. Note that in the current CUDA version, it is allowed to "oversubscribe" blocks per grid and grids per device, the CUDA-driver will schedule it accordingly. Therefore, logic to manage data flow is not required. Also CUDA-surfaces (writable CUDA textures) are utilized to reduce cache misses. CUDA surfaces are laid out in memory using a z-order curve[nVi22a].

Figure 3.4 describes the main steps to compute the solution space (i.e. cost map) of a given cuboid. The main idea is to use Xiao et al.'s branch-optimized version[Xia+12] of the original Amanatides et al. DDA algorithm[AW87] in a slightly modified version (a version for each face direction, using templates and "constexpr if" to compile corresponding versions) in parallel for each face. As a cuboid has six

faces, six kernels are launched concurrently on the GPU, each on its respective grid. Fortunately, devices supporting newer versions of the CUDA-standard allow streams of concurrent grids and can be oversubscribed (i.e. more grids can be scheduled than can actually be executed on the device) as well as the number of blocks per grid. The CUDA-driver will take care of the scheduling[nVi22a]. After the first data initialization steps, the cuboid data is copied onto device by fitting it into a CUDA-surface object, which is a writable CUDA-texture object of up to three dimensions. The reason for this is that a CUDA-surface is using a space-filling curve for its data layout, more specifically a Z-Order curve[nVi22a], preserving spatial locality in its linear (1D) indexing. I.e. voxels that are closely together are indexed relatively close together in 1D to reduce cache-misses significantly[MAK03], see figures 3.5 and 3.6. This is especially important since the cuboid is not being rotated or transposed for each kernel/face, but rather the threads compute their direction vector according to the face and go from there.
Kernel configurations (for each face-kernel) are computed using nVidia's provided "Occupancy"-functions, maximizing the number of threads per block and blocks per grid w.r.t. to occupancy. The kernels are then launched concurrently on separate grids until every grid is finished.

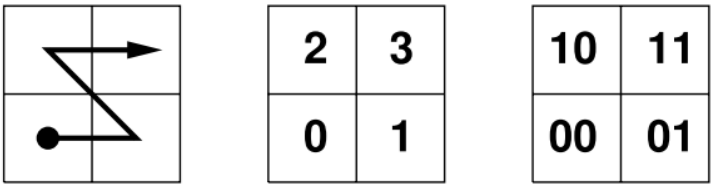


Fig. 1. The 2×2 Z-order (y_0, x_0).

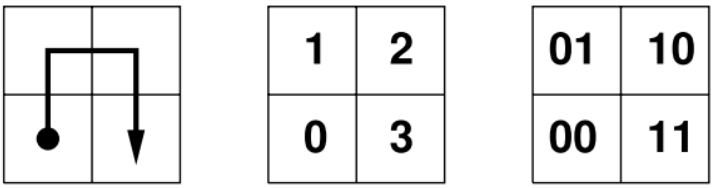


Fig. 2. The 2×2 U-order $(x_0, x_0 \oplus y_0)$.

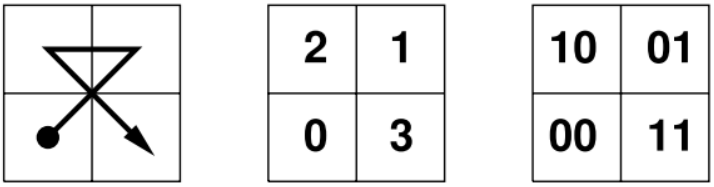


Fig. 3. The 2×2 X-order $(x_0 \oplus y_0, x_0)$.

Fig. 3.5: A selection of space-filling curves in 2D space, their linear indexing scheme (middle) and their respective binary encoding (For the Z-Order curve (also known as the Lebesgue curve) the encoding scheme is known as Morton code) According to nVidia's CUDA technical guide, texture and surface objects utilize a z-order curve. Image taken from [SS15]

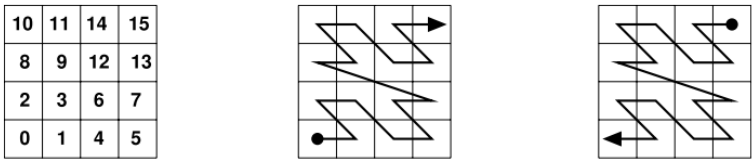


Fig. 4. The 4×4 Z-order and Z space-filling curve (y_1, x_1, y_0, x_0) and its $180°$ rotation.

Fig. 3.6: 2D example of the Z-order (or Z space-filling) curve and its indexing scheme. Note the relatively preserved spatial locality of the 1D indexing. Image taken from [SS15]

3.4.1 Branch-Optimized 3D DDA Kernel

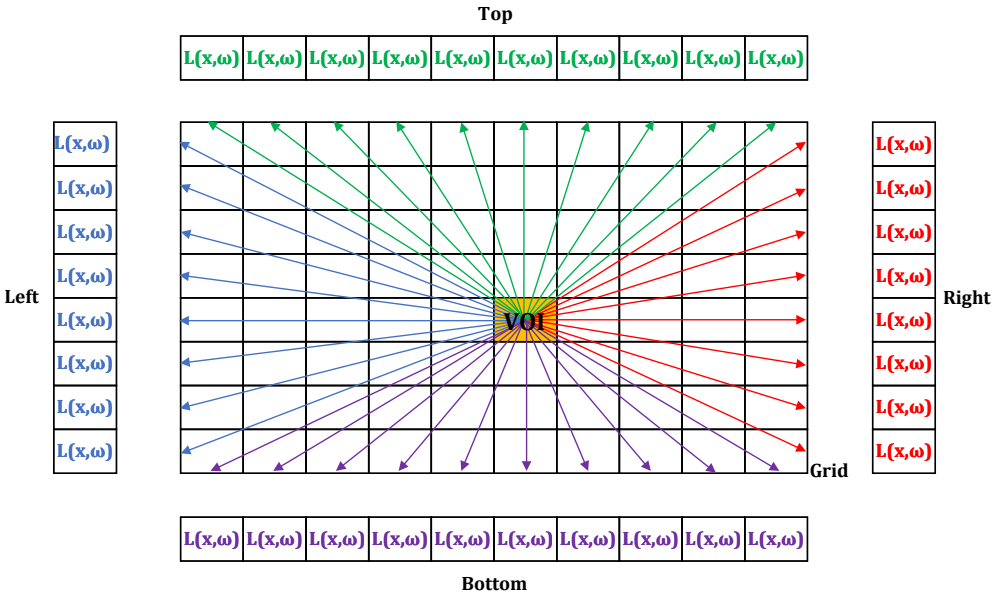


Fig. 3.7: 2D slice showing how the modified algorithm based on Xiao et al.'s work[Xia+12] is working in parallel. A kernel is launched on GPU for each face surface, where a thread corresponds to one surface voxel. Then, the beam model is applied to accumulate the radiance $L(\mathbf{x}, \vec{\omega})$ for each path. The coloring here serves to visualize that each CUDA-kernel for each face surface here is working in parallel (concurrent grids).

Figure 3.7 is showing how the modified algorithm based on Xiao et al.'s work[Xia+12] is working in parallel. Grids for each kernel are launched concurrently on the GPU for each face, one thread corresponding to one respective surface voxel. Then, the beam model is applied to accumulate the radiance $L(\mathbf{x}, \vec{\omega})$ (see section 2.1), through the (partial) path lengths of the beam through each voxel, together with the respective density value of the material the voxel belongs to, from the voxel of interest (VOI) to the corresponding surface voxel. Lastly, the result is written in an additional layer around the grid, which was initialized with 0 previously. These "hull"-voxels serve as the basis for the cuboid surface object later.
Algorithm 5 is showing Xiao et al's branch-optimized version. It might be helpful to

the reader to look at the non-optimized version of the algorithm (algorithm 1) in section 2.1.1.

Algorithm 5: Xiao et al.'s description of the optimized version of the original DDA-based ray-tracing algorithm by Amanatides et al.[AW87], optimized for minimal branch-divergence. Edited for clarity.

1 initialize ray information(source point $\mathbf{S}$, direction $\overrightarrow{d}$);
2 initialize voxel information(voxel size v_{size});
3 initialize $\mathbf{V_{current}}$ as indices of the voxel where $\mathbf{S}$ is;
4 **if** $\exists d_i \in \overrightarrow{d} : d_i \neq 0$ **then**
5 $\quad \mathbf{step} = \{s_i | \forall d_i \in \overrightarrow{d} : s_i = ternary\_op((d_i < 0), -1, 1)\}$;
6 $\quad \overrightarrow{d_{inv}} = \frac{1}{\overrightarrow{d}}$;
7 $\quad \mathbf{t} = ||\mathbf{S} -$"nearest boundary intersection with ray from $\mathbf{S}$"$||_2$;
8 **else**
9 $\quad \overrightarrow{d_{inv}} =$"a large value for each component";
10 $\quad \mathbf{t} =$"a large value for each component";
11 $\mathbf{\Delta} = v_{size} \cdot \overrightarrow{d_{inv}}$;
12 $\mathbf{v_{incr}} = 0$;
13 **while** $\mathbf{V_{current}}$ *is inside traversal region* **do**
$\quad$ `// Note that the whole logical conjunction on the right is being converted to 1 or 0 accordingly.`
14 $\quad v_{incr\_x} = (t_x \leq t_y) \wedge (t_x \leq t_x)$;
15 $\quad v_{incr\_y} = (t_y \leq t_x) \wedge (t_y \leq t_x)$;
16 $\quad v_{incr\_z} = (t_z \leq t_x) \wedge (t_z \leq t_y)$;
17 $\quad \mathbf{t} \mathrel{+}= \mathbf{v_{incr}} \odot \mathbf{\Delta}$; `// Hadamard product`
18 $\quad \mathbf{V_{current}} \mathrel{+}= \mathbf{v_{incr}} \odot \mathbf{step}$;`// Hadamard product`

The step documented in line 7, to initialize $\mathbf{t}$ to the nearest boundary border, is done by using the very efficient (O(1) runtime complexity) 3D ray-AABB intersection algorithm described by Majercik et al.[Maj+18]. As mentioned before, between line 3 and 4, steps were added to compute from given direction $\overrightarrow{d}$ to the direction according to the face. This is done by using templates and "constexpr if"-statements and don't have any impact on runtime complexity.

The common approach is using a variation of Siddon's algorithm. However, this is unfeasible for GPU utilization because of Siddon's memory requirements: It constructs a lookup table and on the GPU, each thread would construct one for itself. Using the previous example, loading a 8GB cuboid on the device and then having at most an additional lookup table of 8GB being built is not feasible for the approach. The trade-off would be the reduce block size, defeating the initial purpose of using the GPU for parallelization.

In sections 2.1.1 and 1.2, another interesting algorithm for this purpose was briefly discussed, based on the original projection algorithm by Joseph[Jos82] using a driving-axis approach and Graetz's version, which is a GPU-executable variant of it[Gra20]. This algorithm was originally considered as well, however, CT data can be based on non-uniform grids, where each voxel "side" has a different length. The approach described by Graetz does not consider this fact in the path length calculation. It considers a relative path based on the relative percentage the partial ray/beam travelled through the volume but not the actual volume size.
For this purpose, it would add imprecision to the whole path length, especially when the voxel in question is only traversed for a very slight bit, e.g. through a corner. However, if input grids were uniform, Graetz's method might be preferable as it is easier to implement while at least as fast.

3.4.2 On The Beam Model

It is also necessary to discuss the utilized beam model for the application in proton radiation therapy. As documented in section 2.2.1, the Bethe-Bloch model determines the stopping power of protons travelling (linearly) through materials and serves as the basis for proton radiation therapy. However, similar to the rendering equation (section 2.1.2), computing it using this model prima face is impossible due to it being not closed form (in addition to being relatively expensive to compute regardless). Thus, several models have been developed to approximate the phenomena described in the Bethe-Bloch model. A very popular approximation, using the Bragg-Kleeman range-energy relationship, to compute radiation dosage transmitted (z) in relation to a given target-depth R_0, was developed by Thomas Bortfeld[Bor97]. Bortfeld's equation uses water as homogeneous media to approximate dose delivery:

$$D_{H_2O}(z) = \Phi_0 \frac{e^{-(R_0-z)^2/4\sigma^2}\sigma^{0.565}}{1+0.012\cdot R_0} \cdot \left[11.26\sigma^{-1}\mathscr{D}_{-0.565}(-\frac{R_0-z}{\sigma}) + (0.157+\frac{11.26\epsilon}{R_0})\cdot\mathscr{D}_{-1.565}(-\frac{R_0-z}{\sigma})\right] \tag{3.8}$$

with $z :=$ residual depth, $\Phi_0 :=$ inital proton fluence, $R_0 :=$ target depth,
$\sigma :=$ standard deviation of the Gaussian distribution function,
$\mathscr{D}_v :=$ the parabolic cylinder function[ZJ96],
$\epsilon :=$ Fraction of primary fluence contributing to the "tail" of the energy spectrum.

The approximation was confirmed by Rasouli et al.[Ras20] with <2% deviation (only occuring at >200 MeV), but with the potential of being mediated by p and α values of the underlying Bragg-Kleeman relationship. The table 2.4 gives a selection of

possible candidates. Ulmer et al. have found that proton fluence, i.e. the amount of protons that pass through a unit volume per time frequency unit, scale with depth in homogeneous media (like water)[UM12]. Consequently, they also described that the required initial energy scales with the depth as well in order to uphold proton fluence with increasing depth. Since the possibility of irradiation reactions (deposing dose) is proportional to the fluence, or more general the interaction cross-section (the possibility that protons interact with the material they travel through) for protons scales with fluence.
Considering this and observing the convergence behaviour of $D_{H_2O}(z)$:

$$\lim_{z\to\infty} D_{H_2O}(z) = 0 \tag{3.9}$$

$$\lim_{z\to R_0} D_{H_2O}(z) = 1 \text{ (max dose)} \tag{3.10}$$

$$\lim_{z\to 0} D_{H_2O}(z) = min(D_{H_2O}(z)), \forall z < R_0 \tag{3.11}$$

It can be determined that the dose distributed along the way (up to R_0)is scaling with the water-equivalent path length (WEPL) and shortly before R_0 it begins to scale exponentially, if fluence is constant. Also considering the general form of the stopping power function up to R_0 (Bragg-Peak) (section 2.2, figure 2.4) is strictly increasing, supports this argument. For the purpose of suggesting "optimal" beam angles for proton therapy, which will have to be evaluated by an expert and evaluated in the final fluence matrix calculations, the beam model used for ray tracing here can be the translated WEPL.

3.5 Optimizing Paths

This part of the system essentially utilizes the "Teaching-Learning-Based Optimization" (TLBO) algorithm by Kern et al.[KLV20]. Using the taxonomy by Stork et al. in section 2.4.2, TLBO is population-based metaheuristic using a "Teacher (current best solution candidate) teaches a class room of students (other solution candidates)"-metaphor to iteratively attempt to mutate the population to converge towards an optimum.
The basic underlying problem that has been identified to optimize path lengths can be expressed as a "0/1-Multidimensional Knapsack Problem" (0/1-MKP) (section 2.4.1), utilizing TLBO to solve. The original description by Kern et al. is completely CPU-bound and, to the best of the author's knowledge, only one well-described method to utilize the GPU for parallelization exists[Ric+19]. GPU-utilizing parallel versions of the repair operator and the evaluation operator will be presented as well.

By definition of the 0/1-MKP, the problem lies within NP and thus no "efficient" algorithm exists to solve it, including TLBO and its variants. Consider again the TLBO algorithm already presented before in section 2.4.2:

Algorithm 6: The TLBO variation as described by Kern et al.[KLV20].

1 $g = 0$;
2 X = initialize population($size_{pop}$);
3 evaluate(X);
4 **repeat**
5 X = sort(X);
  // from minimum objective function score to maximum
6 **for** $i = 1$ **to** $size_{pop}$ **do**
    // Teacher Phase
7 t_f = round to nearest integer(1 + rand(0, 1));
8 $x_{mean} = x_{\lfloor \frac{|X|}{2} \rfloor} \in X$;
9 $x_{teacher} = x_1 \in X$;
10 $x_{i,new} = x_i + rand(0,1) \cdot (x_{teacher} - (t_f \cdot x_{mean}))$;// $x_i \in X$
11 evaluate(X);
    // If the objective function score of $x_{i,new}$ is better than the score of x_i
12 **if** $x_{i,new} > x_i$ **then**
13 $x_i = x_{i,new}$;
    // LearnerPhase
14 $ii = rand(1, size_{pop})\{ii \neq i\}$;
15 **if** $x_i > x_{ii}$ **then**
16 $x_{i,new} = x_i + rand(0,1) \cdot (x_{ii} - x_i)$;
17 **else**
18 $x_{i,new} = x_i + rand(0,1) \cdot (x_i - x_{ii})$;
19 evaluate(X);
20 **if** $x_{i,new} > x_i$ **then**
21 $x_i = x_{i,new}$;
22 g = g + 1;
23 **until** $g \neq generations_{max}$

The evaluation operation is parallelized, as well as the repair operation, following immediately (but has sparsely been documented by Kern et al.) after a transformation operation (lines 10, 16 and 18 in 6). As with many other metaheuristics, it involves non-deterministic components. Here in the form of random generation of initial solution candidates as well as in the transformation operations, which utilize a biased randomness as well.
Note that one immediate performance and memory optimization being applied is to

use sparse vectors to represent a solution candidate. Solution candidates are vectors of $\mathbb{B}^n$, with usually many more 0-components than 1-components. This seems to hold especially true for proton radiation therapy as research of the current literature only yielded applied number of beams, i.e. the number of 1-components, within the single-digit range (< 10). For large number of variables/items, i.e. a large n, it is therefore sensible to only consider the 1-components for computation. For this, the sparse vector type of the venerable Eigen library[GJ+10] is used.
The first step is to initialize the solution population, which also only has been documented as a suggestion of possible vague high-level steps, most of which had to be reconsidered for this thesis.

3.5.1 Initial Population Generation

The purpose of the first step of TLBO is to generate unique, feasible solutions according to a given population size. The method devised for this thesis utilizes factoriadics (number system with base factorial) to quickly take a randomly generated unique decimal number and using it as a "rank", i.e. an index corresponding to a set of combinations of decimal numbers. The core idea is to use the isomorphism between factoriadics and sets of numerical permutations and combinations and to compute, from the given "rank" (the randomly generated number) the combination of numbers by translating the "rank" into a factorial number, which in turn encodes a corresponding to numerical combination as described by Genitrini et al.[GP21]. This step will be explained in greater detail later.

The algorithm developed in this thesis to generate the initial population is described in 7:

Algorithm 7: Initialize population algorithm. Generates unique, feasible solutions according to a given population size.

1 $num_{perms} = nCr(num_{vars}, size_{sample})$;
2 $size_{pop}$ = ternary_op($(num_{perms} \leq size_{pop})$, num_{perms}, $size_{pop}$);
3
4 **solution_population** $\subseteq \mathbb{R} \times \mathbb{B}^n$;
5
6 seed = rand(0, $num_{perms} - 1$) $\in \mathbb{N}$;
7 b = $nCr(num_{vars} - 1, size_{sample} - 1) \cdot num_{vars}$;
8 **for** $i = 0$ **to** $size_{pop}$ **do**
9 $idx_{combination}$ = permute(i, num_{perms}, seed);
10 $\overrightarrow{solution}$ = unrank(num_{vars}, $size_{sample}$, i, b) $\in \mathbb{B}^n$;
11
12 **if** $\overrightarrow{solution}$ *is valid* **then**
13 score = calc_objective_score($\overrightarrow{v_{profits}}$, $\overrightarrow{solution}$);
14 **solution_population** = **solution_population** $\bigcup \{(score, \overrightarrow{solution})\}$;
15 return **solution_population**;

The first two lines are ensuring that the given population size $size_pop$ can actually yield the number of requested unique solutions. This is done by computing the binomial coefficient to get the amount of possible combinations of r elements out of a set of n elements:
As mentioned in the beginning, a numerical combination essentially represents a solution candidate $\overrightarrow{solution} = (s_1, s_2, ..., s_n) \in \mathbb{B}^n$, encoded as a sparse vector. This is done by generating the combination $c = (c_1, c_2, ..., c_n) \in \mathbb{N}^n$ and taking the first

k elements $c_1, ..., c_k$, which are, according to the combination, different for each combination. Remember that permutations do have constraints regarding the order, so permutations which have the first k elements as "1, 2, 3", "1, 3, 2", "2, 1, 3", ... are all considered different, whereas they are considered to be the same combination. These first k elements are then interpreted as indices of non-zero (i.e. 1) components in the solution: $\overrightarrow{solution} = (..., s_{c_1} = 1, ..., s_{c_2} = 1, ...s_{c_k} = 1, ...)$.
To compute the binomial coefficient (also known as "n choose r", nCr), the well-known multiplicative formula has been chosen[GKP89, Chapter 5]. It is a fast ($O(k)$) and simple-to-implement method. As the initialization of the population is done once during the whole pipeline and nCr is also only called once in the beginning of the algorithm, further optimization of this step is not a priority and $O(k)$ is deemed as fast enough.

solution_population $\subseteq \mathbb{R} \times \mathbb{B}^n$ holds the set of generated unique and feasible solution candidates. A solution candidate is a tuple consisting of the objective score of a $\overrightarrow{solution}$ and the solution itself: $\{(score, \overrightarrow{solution})\}$. Within the for-loop (line 8 and onward), the aforementioned "rank" of a combination ($idx_{combination}$), corresponding to a combination by considering the isomorphic mapping between factoriadics and combinations, is first randomly generated. This is done by using the jittered multi-sampling method described by Kensler[Ken13], which, for a given seed, generates in O(1) essentially a table on-the-fly of (pseudo-)random, unique decimal numbers within a given range. The method itself was initially devised to improve path tracing sampling for Pixar's internal renderer. A problem with path tracing is to find a good (pseudo-)random sampling scheme for the bouncing rays without any apparent bias due to clustering or emerging meta-patterns (which would indicate a non-random behaviour of the sampling). Kensler reports that his method provides a good-enough random spread of numbers for a given seed. The implementation of Kensler's O(1) method can be found in the source code, it is called "permute".

The solution is then checked whether it is valid/feasible by using the evaluation operation, parallelized using the GPU. This will be discussed in the section following this one. If the solution is valid, the objective score is calculated, which is essentially the inner product of the profits (consider the 0/1-MKP!) vector $\overrightarrow{v_{profits}}$ and $\overrightarrow{solution}$. This is a common operation provided by Eigen. Lastly, the valid solution is added to **solution_population**. The for-loop repeats until the given (or possible) solution population size or a maximum amount of "tries" has been reached.

In the context of combinatorial mathematics, unranking a permutation (and combination) describes an algorithm that, for a given integer $u \in \mathbb{N}$, will give out a permutation sorted according to a specified scheme (usually lexicographical), so u denotes the "rank" or index of that permutation/combination according to the sorting scheme[PB81]. Commonly, for combinations, this was done using the com-

binatorial number system[PB81][GP21]. Genitrini et al. have described a method using factoriadics to unrank combinations, providing a straight forward and fast algorithm (similar to nCr, in $O(k)$).
The core ideas will be introduced. Different to representing a decimal value in another number system with a different base (like base 2 or 8, etc.), the factoriadics do not have a fixed radix, i.e. not a fixed "distance" from one digit to the next after it.

Definition: Factoriadic:
Let $u, n \in \mathbb{N} : (n-1)! \leq u < n!$. Then there exists a unique sequence of integers $(f_l) = (f_0, f_1, ..., f_{n-1})$ with $0 \leq f_l \leq l$ such that

$$u = f_0 \cdot 0! + f_1 \cdot 1! + ... + f_{n-1} \cdot (n-1)! = \sum_{i=0}^{n-1} f_i \cdot i! \tag{3.12}$$

where the finite sequence $(f_0, f_1, ..., f_{n-1})$ is called factoriadic of u.

Definition: Permutation:
Let $n \in \mathbb{N}$. A permutation of size n is an ordering of the elements of the set $P = 0, 1, ..., n-1$.

Genitrini et al. report and prove that there exists an isomorphism between integers and permutations, specifically with regards to sets of permutations of size n. Furthermore, that isomorphism provides a lexicographical order on the sets of permutations[GP21]. The unoptimized, common method to obtain a permutation from a decimal number, i.e. "unranking", is simple:

1. Establish a set of integers $S = \{0, 1, ..., n-1\}$
2. From the factoriadic of u, read from right to left.
3. The current element f_i read from the factoriadic corresponds to the index of the element in S
4. The value of the element in S at index f_i is extracted ($S = s_{f_i} \setminus S$)and inserted in the permutation $P = P \bigcup s_{f_i}$.

5. Repeat from step 2 until the factoriadic has been fully read, i.e. leftmost element f_0 has been processed.

An example might be helpful: Let $u = 1986$, the corresponding factoriadic is $(0, 0, 0, 3, 2, 4, 2)$. Applying the method:
$S_0 = \{0, 1, 2, 3, 4, 5, 6\}$, $P = \{\}$
Read: 2 $\Rightarrow s_2 \in S_0 = 2$, extract from S, add to P: $S_1 = \{0, 1, 3, 4, 5, 6\}$, $P = \{2\}$
Read: 4 $\Rightarrow s_4 \in S_1 = 5$, $S_2 = \{0, 1, 3, 4, 6\}$, $P = \{2, 5\}$
Read: 2 $\Rightarrow s_2 \in S_2 = 3$, $S_3 = \{0, 1, 4, 6\}$, $P = \{2, 5, 3\}$
Read: 3 $\Rightarrow s_3 \in S_3 = 6$, $S_4 = \{0, 1, 4\}$, $P = \{2, 5, 3, 6\}$
Read: 0 $\Rightarrow s_0 \in S_4 = 0$, $S_5 = \{1, 4\}$, $P = \{2, 5, 3, 6, 0\}$
Read: 0 $\Rightarrow s_0 \in S_5 = 1$, $S_6 = \{4\}$, $P = \{2, 5, 3, 6, 0, 1\}$
Read: 0 $\Rightarrow s_0 \in S_6 = 4$, $S_7 = \{\}$, $P = \{2, 5, 3, 6, 0, 1, 4\}$

Genitrini et al. then define an isomorphism between k-combinations of sets with n elements and a subset of permutations of n elements. Note that permutations are essentially combinations with more restrictions: Permutations also consider the order of elements as a distinctive factor, e.g. let set $S = \{a, b, c, d\}$. There exist 2-combinations: ab, ac, ad, bc, bd, cd. And 2-permutations of S would be: ab, ba, ac, ca, ad, da, bd, db, cd, dc. They then transform the "rank" u (for a combination!) to a corresponding rank u' for a permutation, using u' to build a permutation according to the scheme presented (in an optimized way), corresponding to a combination. In essence, permutations are used as an "in-between" domain between the domain (combination rank u) and the codomain (the combination), isomorphic to both.
An optimized version of the unranking algorithm is described by Genitrini et al. (Algorithm 6 in their paper[GP21]) and utilized in this thesis with a slight modification: Since a solution vector is a sparse vector and only k-combinations of a set of integers of size n are considered (i.e. k corresponds to the number of non-zeros in the solution candidate), the value of s_{f_i} is taken as index of a non-zero element in the solution candidate. I.e. each s_{f_i} represents an index to a non-zero (1-component) in the solution candidate. And only the first k s_{f_i} are considered, akin to the number of non-zero elements allowed/required.

3.5.1.1 Complexity

Looking at algorithm 7, the worst-case complexity can be determined in a straightforward manner: nCr is done twice before the loop, requiring $2 \cdot T(k) \in O(k)$. The loop itself is bounded by the number of solutions in the population, $size_{pop}$. Kern et al. recommend a size around 30 in their paper[KLV20] and argue that increasing the population size after a certain point does not yield significantly better converging

results. Therefore an argument can be made that there is a "usefulness"-bound for a population size, i.e. $log(size_{pop})$. However, it will not be used as an argument here and instead $size_{pop}$ will be considered naively.
Within the loop, "permute" is called, requiring $O(1)$ and can be omitted. Then "unrank" is called, requiring $O(k)$.
Evaluation of the solution candidate is parallelized using the GPU and depends on its capabilities. A more thorough discussion about the worst-case complexity of the evaluation operation is done in the next section. For the sake of argument, if the GPU is only capable of processing one constraint at a time (with m constraints in total) and the solution candidate has n variables/items, it would require $T(n \cdot m) \in O(n^2)$, being ultimately no different than using a CPU sequentially. Allowing for more grids/kernels p_k to process constraints and more threads p_t per kernel:

$$T(\frac{n}{p_t} \cdot \frac{m}{p_k}) \Rightarrow \lim_{(p_t,p_k)\to(n,m)} (T(\frac{n}{p_t} \cdot \frac{m}{p_k})) = T(1 \cdot 1) \in O(1). \tag{3.13}$$

If the solution is valid, the objective score is calculated, requiring $T(num_{vars}) \in O(n)$. If evaluation is done sequentially without using the GPU, one loop requires $T(\frac{n}{p_t} \cdot \frac{m}{p_k}) + T(n + k)) \in O(n^2)$. As such, the whole loop requires $size_{pop} \cdot (T(\frac{n}{p_t} \cdot \frac{m}{p_k}) + T(n + k)) \in O(n^3)$. Adding the nCk from the beginning ($O(k)$) to this does not impact the significant terms and can be omitted. Considering the GPU, the worst-case complexity is

$$\lim_{(p_t,p_k)\to(n,m)} (log(size_{pop}) \cdot T(\frac{n}{p_t} \cdot \frac{m}{p_k}) + T(n + k)) = size_{pop} \cdot (T(1 \cdot 1) + T(n + k)) \in O(n^2) \tag{3.14}$$

Even if it is not the best, the fact that the population is initialized once during the whole pipeline in the beginning of TLBO diminishes the overall impact.

3.5.2 Evaluation Operator

One of the principal operations used in TLBO is the evaluation of a solution. It occurs not only in the initial population generation but also after each transformation operation in TLBO and therefore possible optimizations of this operation have merit. Since at its most basic, the evaluation is a generalized matrix-vector product with a generalized reduction of the resulting vector, it fits the typical use cases for parallelization on the GPU. This is also the core mechanism with which to compute/solve the 0/1-MKP. Unfortunately, to the best of the author's knowledge, no libraries exist providing such functionality implemented on the GPU supporting heterogeneous matrices as well as different function applications for each row, which is required to process each constraint accordingly. As such, such functionality

has been developed and implemented for the system presented in this thesis with supporting data structures.

Indeed, the actual operation for evaluating a solution is structurally equivalent to the "constraint part" of the 0/1-MKP, as it has been discussed in section 2.4.1. On the left below, the 0/1-MKP can be seen and on the right the evaluation operation that specifically targets to solve the constraints.

$$
\begin{aligned}
&\text{maximize } \sum_{j=1}^{n} p_j x_j \\
&\text{subject to } \{ \quad \sum_{j=1}^{n} a_{i1} x_j \square_1 b_1, \\
&\qquad\qquad \sum_{j=1}^{n} a_{i2} x_j \square_2 b_2, \\
&\qquad\qquad \ldots, \\
&\qquad\qquad \sum_{j=1}^{n} a_{ij} x_j \square_i b_i, \\
&\qquad\qquad \ldots, \\
&\qquad\qquad \sum_{j=1}^{n} a_{mj} x_j \square_m b_m \quad \}
\end{aligned}
\qquad
\bigwedge_{i=0}^{m} f_{rel_i} \left(\left[\sum_{j=0}^{n} f_{ci}(a_{ij}, s_i) \cdot x_j \right], b_i \right)
$$

where
$\square_i \in \{<, \leq, =, \geq, >\}$, $x_j \in \{0, 1\}$,
$i = 1, ..., m$ and $j = 1, ..., n$.

where
$f_{rel_i} : \mathbb{R} \times \mathbb{R} \to \mathbb{B}$.
$f_{ci} : T_i \times T_i \to \mathbb{R}$.
$a_{ij} \in \mathbf{A}^{m \times n}$ (element of type T_i of constraint matrix.)
$m := |\Gamma|$ (number of constraints given).
$n := dim(\mathbb{B}^n)$ (number of items/elements in a solution candidate.)
s_i (supplemental data to support dynamic/dependent constraints.)
$x_j \in \mathbb{B}^n$ (the selection vector, i.e. the solution candidate vector.)
$b_i \in \mathbb{R}^m$ (the boundaries).

With a binary relation $f_{rel_i} \equiv \square_i$ and a binary function f_{ci} according to constraint i, $\leq i \leq m$. The latter function serving the purpose of allowing dynamic constraints, i.e. constraints that are dependent on the form of the solution candidate or other dynamic conditions. As of now, this is primarily needed for minimum distance constraints between beams, i.e. between the positions of the 1-components in the solution

candidate x_j. f_{rel_i} and f_{c_i} are implemented as host and device functions and stored on constant device memory, accessible via function tables; an index is used to call the specific function, i.e. f_c_table[i](a_{ij}, s_i) and f_rel_table[i]($\sum_{j=0}^{n} f_{ci}(a_{ij}, s_i), b_i$). The general form of a constraint is:

$$(\text{"}ConstraintName\text{"}, T_i, f_{rel_i}, f_{c_i}) \tag{3.15}$$

With "Constraint Name" being a string, denoting the name of the constraint. T_i being the data type of the corresponding row of the constraint matrix, holding the respective a_{ij}. The constraint matrix $\mathbf{A}^{m \times n}$ is implemented as a row-wise heterogeneous matrix, where each row holds one type T_i of data according to its constraint. More precisely, $\mathbf{A}^{m \times n}$ can be seen as the direct sum of row-vectors of size n:

$$\mathbf{A}^{m \times n} = \overrightarrow{r_1} \in T_1^n \oplus \overrightarrow{r_2} \in T_2^n \oplus ... \oplus \overrightarrow{r_m} \in T_m^n \tag{3.16}$$

The purpose of using a heterogeneous matrix is to reduce cache misses: The underlying type to store data is (cuda::)std::byte, data is stored and read linearly. Offset sizes (in byte) for how many bytes a row requires are stored as well as respective row indices. The trade-off being made here is to gain more speed but forgoing type-safety.

Figure 3.8 visualizes the process of evaluating a solution candidate as it has been implemented. For the purpose of parallelization, CUDA-streams were used to launch grids concurrently on the device (the gray areas). The number of streams is v with $1 \leq v \leq m$, where m denotes the number of constraints or "rows" in the constraint matrix. Each stream is scheduled with a number of rows/constraints that it processes sequentially. Each stream is scheduled with a number of rows/constraints that it processes sequentially; from at least one row/constraint per stream to possibly more rows/constraints per stream, depending on how many streams and memory the GPU provides. E.g. if there are 8 constraints in total and the device can only hold 2 and therefore can only run 2 parallel streams, then each stream receives 4 rows to process. In the case of odd numbers of streams, those "remaining" rows are given to the streams in a round-robin scheme. Note here that the streams are asynchronous: They are launched and the host thread is free to launch other streams or do something else until a barrier (e.g. a cudaDeviceSynchronize()) is reached.
Each stream/grid is processing the same tasks: First computing an offset idx_v which serves as a global index to the constraint matrix rows and the global device buffer buffer_d, which is accessible to every stream and each stream works on a specific region according to its offset.

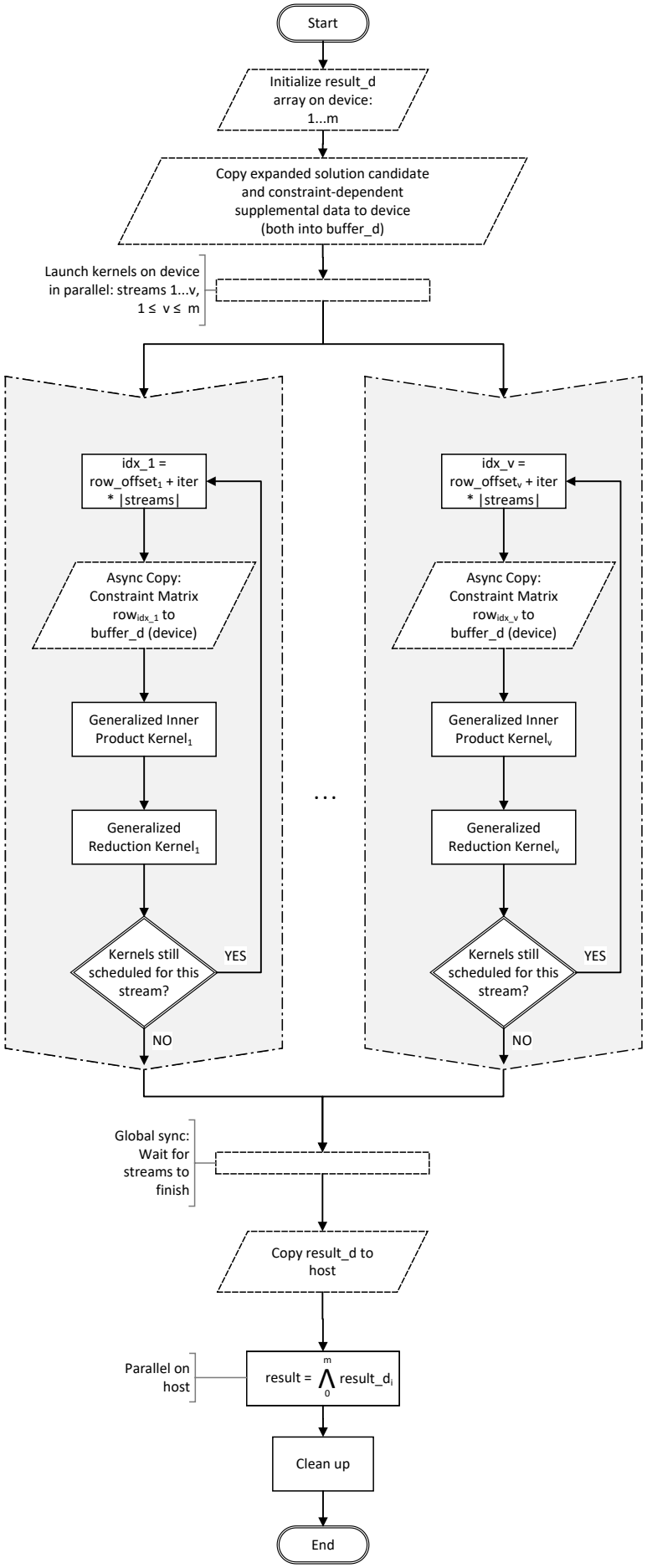


Fig. 3.8: Visualization of the evaluation operation and its parallelization scheme. Note that the gray areas denote a stream. The number of streams is v with $1 \leq v \leq m$, where m denotes the number of constraints or "rows" in the constraint matrix. Each stream is scheduled with a number of rows/constraints that it processes sequentially. The number of rows a stream receives is determined by the number of total constraints and the number of constraints fitting on the device.

That region within buffer_d is used to asynchronously copy one to several rows from the host constraint matrix to the device. A section of buffer_d at the beginning of the buffer is restricted as it holds the solution vector $\overrightarrow{x}$; which was expanded from a sparse to a "full" vector since, to the best of the author's knowledge, implementations of sparse vectors for GPU-side usage don't exist, typically being implemented using hash-tables. It also holds result_d, i.e. the results of each f_{rel_i}, which are later reduced on the host in parallel to one final result; determining whether the constraint is valid or not. After a stream has copied the required data to the device, a "generalized inner product kernel" is launched, computing $\sum_{j=0}^{n} f_{ci}(a_{ij}, s_i)$, via the usage of the aforementioned function tables. As the kernel is a heavily modified dot-product kernel, a final step is required to reduce all the remaining partial results. This is done using the "generalized reduction kernel", computing f_{rel_i} also via a function table. Lastly, the result of the stream is stored in a corresponding index at result_d and if more tasks are scheduled on that stream, work begins anew. The host waits until all streams have finished before reducing result_d.

For performance reasons, the block configuration and grid configuration required to launch the streams are computed once in the beginning of TLBO and kept; they are not re-computed every time an evaluation operation is launched. buffer_d was similarly declared once in the beginning so memory allocation and deallocation don't have to be done repeatedly. The size of buffer_d tries to fit in as much of the constraints as possible while also considering memory requirements for the solution candidate $\overrightarrow{x}$, result_d and a "safety"-margin for temporary data on the device.

3.5.2.1 Complexity

As briefly mentioned in the discussion of the complexity for the generation of the inital population (section 3.5.1.1), the worst-case runtime complexity is dependent on the capabilities of the device (GPU). Before the streams are launched however, data is initialized and copied onto the device according to Figure 3.8. Specific timings on cudaMemcpy are not public, it is a reasonable assumption that it requires T(m) for initializing result_d, T(n) for copying the solution vector onto the device and T(n) for copying supplemental data of the current solution candidate: $T(m) + 2 \cdot T(n) \in O(n)$.
The more interesting part comes right after: The concurrent streams processing scheduled tasks on the device. The kernel and stream configurations have already been determined in the beginning of TLBO and need not be computed again. Calculating the respective offset idx_v is done in $T(1)$, copying a row to the corresponding spot within buffer_d is done in $T(n)$. The "generalized inner product kernel" and the following "generalized reduction kernel" compute $\sum_{j=0}^{n} f_{ci}(a_{ij}, s_i)$.
The "generalized inner product kernel" still possesses the general structure of the

common implementation of the cuda dot-product kernel (found in the official nVidia CUDA examples[nVi22b]), detailed in algorithm 8:

Algorithm 8: Outline of the generalized dot product kernel, which performs a partial reduction operation as well.

```
1  ...
2  tid = threadIdx.x + blockDim.x · blockIdx.x;
3  result_tmp = 0;
4  cache_idx = threadIdx.x;
5  ...
6  while tid < n do
7      result_tmp = f_c_table[i](a_ij, s_i);
8      tid += blockDim.x · gridDim.x;
9  ...
10 i = blockDim.x/2;
11 ...
12 while i ≠ 0 do
13     if cache_idx < i then
14         cache[cache_idx] += cache[cache_idx + i];
15     __syncthreads();
16     i /= 2;
17 ...
```

The number of iterations in first while-loop at line 6 are determined by the block and grid layouts: $T(\frac{n}{\text{blockDim.x}\cdot\text{gridDim.x}}$. For one grid with one thread it would require $T(n)$, increasing the threads per block and blocks (i.e. grid size) would converge to $T(1)$. The second while-loop at line 12 reduces the step size of i by 2 at each iteration, therefore $T(log_2(\text{blockDim.x})$. The dominant term here is the first while-loop with $T(n)$.

The "generalized reduction kernel" utilizes one while-loop with the same conditions as the the one at line 6 for the "generalized inner product kernel", thus: $T(n)$. The stream then would require in total $2 \cdot T(n)$. The threads per block and number of blocks per grid determine the number of total threads influencing the dominant term:

$$2 \cdot T(\frac{n}{p_t}) \tag{3.17}$$

with p_t denoting the number of threads in total for the grid assigned to the stream. As multiple constraints/rows can be scheduled to a stream:

$$2 \cdot T(m \cdot \frac{n}{p_t}) \tag{3.18}$$

with m being the number of constraints. As several concurrent streams can be launched:

$$2 \cdot T(\frac{m}{m_k} \cdot \frac{n}{p_t}) \tag{3.19}$$

with p_k being the number of concurrent grids/streams launched. Considering the time required for the operations before the grid launches (copying and initializing data), i.e. $T(m) + T(n) \Rightarrow T(m) + T(n) + T(\frac{m}{p_k} \cdot \frac{n}{p_t})$ can be omitted as the stream-complexity is the dominant term here. The following observations regarding the worst-case complexity can be made now:

$$\lim_{(p_t,p_k)\to(1,1)}(2 \cdot T(\frac{n}{p_t} \cdot \frac{m}{p_k})) = 2 \cdot T(\frac{n}{1} \cdot \frac{m}{1}) = 2 \cdot T(n \cdot m) \in O(n^2) \tag{3.20}$$

with one thread and one stream launched, essentially emulating as if the CPU would compute it sequentially.

$$\lim_{(p_t,p_k)\to(n,1)}(2 \cdot T(\frac{n}{p_t} \cdot \frac{m}{p_k})) = 2 \cdot T(\frac{n}{n} \cdot \frac{m}{1}) = 2 \cdot T(m) \in O(n) \tag{3.21}$$

with up to n threads and one stream launched.

$$\lim_{(p_t,p_k)\to(n,m)}(2 \cdot T(\frac{n}{p_t} \cdot \frac{m}{p_k})) = 2 \cdot T(\frac{n}{n} \cdot \frac{m}{m}) = 2 \cdot T(1 \cdot 1) \in O(1) \tag{3.22}$$

with up to n threads and up to m streams launched. The work of a stream here has already been considered in calculation of the complexity. The algorithm as a whole can be considered work-efficient, as $O(1) < O(n) < O(n^2)$, where $O(n^2)$ would be the worst-case complexity for a sequential processing of the algorithm, whereas $O(1)$ and $O(n)$ already consider multiple threads and streams.
The depth D of stream can be easily deduced from the previous observations: The "generalized inner product kernel" has a depth $D_{prod}(n) = T(n)+T(log_2(\text{blockDim.x}))$, as the first while-loop iterates up to size times at maximum, depending on the block and grid layouts. The second while-loop iterates $log_2(\text{blockDim.x})$ times. The "generalized reduction kernel" has a depth $D_{red}(n) = T(n)$, as it utilizes one while-loop with similar conditions to the first while-loop in the "generalized inner product kernel". Therefore the depth for a stream $D_{stream}(n) = D_{prod}(n) + D_{red}(n) = 2 \cdot n + log_2(\text{blockDim.x})$.

3.5.3 Repair Operator

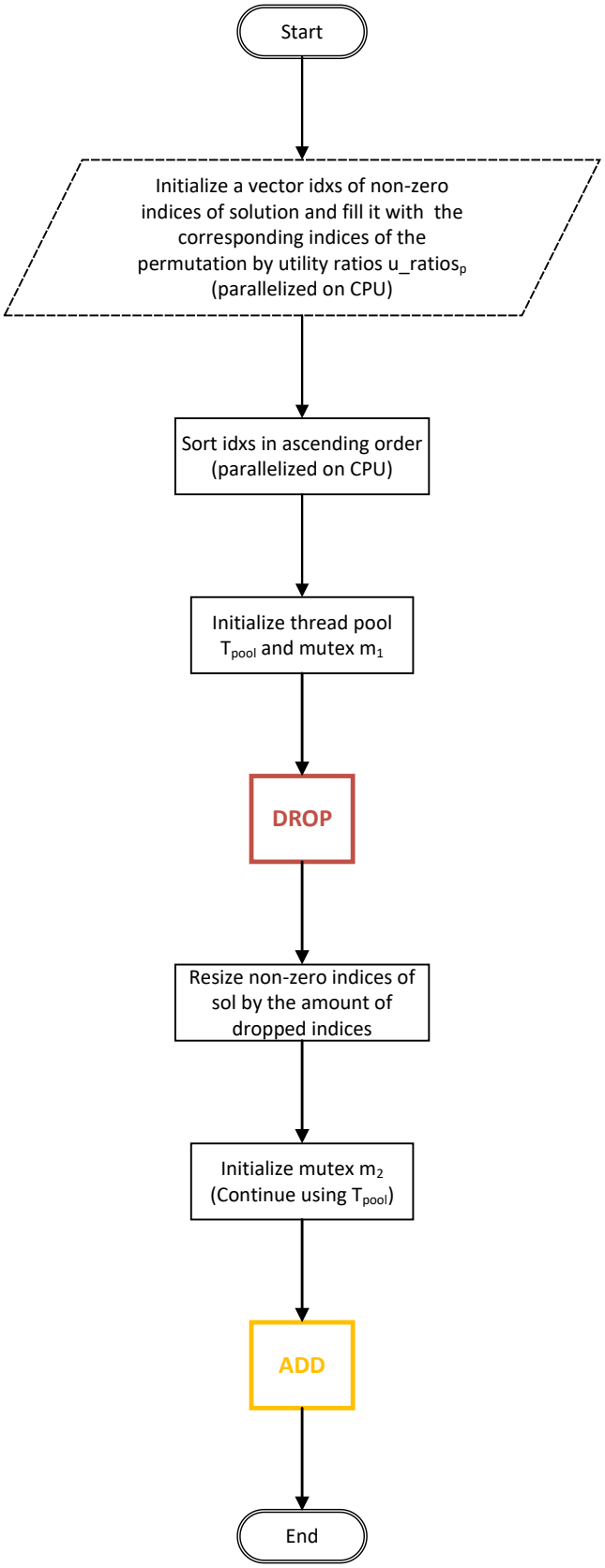


Fig. 3.9: High-level visualization of the main steps repair operation. Note that the gray areas denote a thread. The number of threads is determined by the size of the thread pool T_{pool}. Note that both the drop and add operations launch threads, which launch their own evaluation operation on the GPU. This is discussed in more detail in the specific sections of these operations.

Originally, the TLBO-version described by Rao et al.[RSV11] does not include a repair operation and instead discards invalid solution candidates and generates new ones until one is valid. Kern et al.[KLV20] have incorporated the repair operator, originally described by Chu et al. in their description of Genetic Algorithms[CB98], to attempt to repair an invalid solution candidate after a transformation. This chapter describes an approach to implement it (as only a vague, abstract description has been given by Kern et al.) and parallelize it, building on the evaluation operator discussed in the previous section.

Figure 3.9 is a visualization of the principal steps of the algorithm developed in this thesis to implement the repair operator. The first step is to initialize a vector $idxs$ of non-zero indices of the current solution candidate and mutating these indices during initialization by overwriting them to with the corresponding permutation indices in the utility ratios permutation vector u_ratios_p. Then these indices are sorted in ascending order. This will be helpful during the drop operation.

3.5.3.1 Utility Ratios

Kern et al. describe the purpose of the utility ratios, as derived from Chu et al.'s work, as essentially an order of "impact" of each variable/item in a solution candidate. It is defined as

$$\overrightarrow{u\_ratios} = (u_1, u_2, ..., u_j, ..., u_n) \in \mathbb{R}^n \text{ with} \qquad (3.23)$$
$$u_j = \frac{p_j}{\sum(W_i \cdot a_{ij})}$$

where p_j is the j-th element of the profits vector and a_{ij} the corresponding element of the constraint matrix $A^{m \times n}$ (see 0/1-MKP as discussed in section 2.4.1 and section 3.5.2 respectively). Kern et al. describes the "W_i-values as surrogate multipliers, essentially being weights for the respective row of constraints. These are typically obtained via solving a linear programming relaxation of the MKP but Kern et al. have devised a quicker scheme: They compute the W_i by taking the best (i.e. best objective score value) solution candidate x_{best} from the inital population and sort its constraints (the constraint matrix rows $1 \leq i \leq m$) by tightness. According to Dyer et al.[Dye+17], an inequality constraint is tight at a certain point j if the point lies on the hyperplane, i.e. $a_{ij} \cdot x_j = b_i$. However, Kern et al. relax it by looking for "least slack", i.e. $min(|(a_{ij} \cdot x_j - b_i|)$ for all j, $1 \leq j \leq n$ and constraint i.
For the the implementation in this thesis then, tightness t_i for a constraint matrix

row i is the absolute value of subtracting the evaluated constraint value from the corresponding boundary value b_i:

$$(t_1, t_2, ..., t_i, ...t_m) \in \mathbb{R}^m \text{ with} \tag{3.24}$$
$$t_i = \left| \left[\sum_{j=0}^{n} f_{ci}(a_{ij}, s_i) \cdot x_{bestj} \right] - b_i \right|$$

Using these tightness values to sort the constraint matrix rows in descending order (tightest constraints are at the top of the list). Then the top half of the sorted constraint matrix rows have their W_i set to 1, the bottom half have their W_i set to 0. The i correspond to the respective constraint matrix row indices.
The next step then is to sort the copied solution candidate by the u_ratios vector/list, which serves as a permutation vector to sort sol_c by. This is implemented using a customized sorting iterator for permutation-based sorting applied to std::sort in a parallelized manner on the host.
Algorithms 9, 10 and 11 document the computation of the surrogate multipliers, the computation of the utility ratios and the utilization of the utility ratios as a permutation to sort by respectively. These listings will also be referred to during the discussion of the complexities. **Note that the computation of the surrogate multipliers as the utility ratios is only done once in TLBO after the initial population has been generated.** and thus, there is no multiplicative speedup to be gained and therefore it was deemed to opt for the easier implementation and less memory allocation operations/movement.
An additional algorithm, almost equivalent to algorithm 11, also exists to compute the inverse permutation vector. The only difference there is that the custom sorting operation is reversed and the already computed permutation vector by algorithm 11 is used as a basis, thus not requiring to call the other algorithms again. This algorithm is not listed here but discussed in the complexity analysis. The inverse permutation vector as well as the permutation vector are required in the "drop operation".

Algorithm 9: Algorithm to compute the surrogate multipliers.

thread_pool $T_{pool}(num_{threads})$;

// Vector of pairs. First component holds the tightness value, second component holds the constraint index according to the constraint matrix. A

tightnesses[$A.num_{rows}$] $\subset \mathbb{R} \times \mathbb{N}$;

// Submit tasks to the pool. Each task is a callback to evaluation function, specifically for one row; i.e. for one constraint. The amount of constraints is equal to the number of rows to the constraint matrix and we will save the computed tightness of each constraint in the tightnesses vector.

for *(i = 0, iter = A.row[0])* **to** *($A.num_{rows}$, A.row[m - 1])* **do**

    T_{pool}.push_task(λ(i) : $\mathbb{N} \rightarrow \mathbb{R} \times \mathbb{N}$ {

    // compute tightness using a version of the evaluate operator that, instead of a boolean result, computes the value as presented in equation 3.24 with respect to the best solution sol_{best} of the initial population. Done on the GPU in parallel but for just one row. I.e. one thread in the thread pool launches one stream, essentially. Result is then inserted into tightnesses at position i.

    });

// pool-wide barrier

T_{pool}.wait_for_tasks();

// sort and get the median solution and map every value > median = 1, else = 0.

std::sort(std::execution::par, tightnesses.begin(), tightnesses.end(), ...);

$t_{median} \in \mathbb{R}$ = tightnesses[$\frac{|\text{tightnesses}|}{2}$].first;

$\overrightarrow{results}$[|tightnesses|] $\in \mathbb{R}^m$;

foreach $t_i \in$ *tightnesses* **do**

    $\overrightarrow{results}$[elem.second] = ternary_op(t_i.first $\geq t_{median}$), 1, 0);

return $\overrightarrow{results}$;

Algorithm 10: Algorithm to use the computed surrogate multipliers to compute the utility ratios.

```
1 thread_pool T_pool(num_threads);
2
3 u_ratios ∈ ℝ^n;
4 for i = 0 to n - 1 do
5     T_pool.push_task( λ(i) : ℕ → ℝ {
      // Essentially computes in parallel on the host the equation 3.23.  Each
         surrogate multiplier computed before serves as a weight for a_ij ∈ A^{m×n}
         for row i respectively.  The computed u_j are stored in u_ratios[i].
6     });
  // pool-wide barrier
7 T_pool.wait_for_tasks();
8
9 return u_ratios;
```

Algorithm 11: Algorithm to sort a solution candidate according to the computed utility ratios. It uses a custom sort iterator, permuting (sorting) an input range according to a permutation vector

```
1 u_ratios ∈ ℝ^n = utility_ratios(p, sol_best, W, A^{m×n}, ...);
2
  // Permutation initialized as {0, 1, ..., n − 1}
3 P ⊆ ℕ^n = {0, 1, ..., n − 1};
  // Custom sort iterators that sort P, which is a set of {0, 1, ..., n - 1},
     such that the elements are now sorted according to utility_ratios, making P
     now a permutation that can be used in the repair operation to sort the
     elments of given candidate solution by using P as indices.
4 std::sort(std::execution::par, custom_begin(P, u_ratios),
    custom_end(P, u_ratios), ...);
5
6 return P;
```

3.5.3.2 Drop

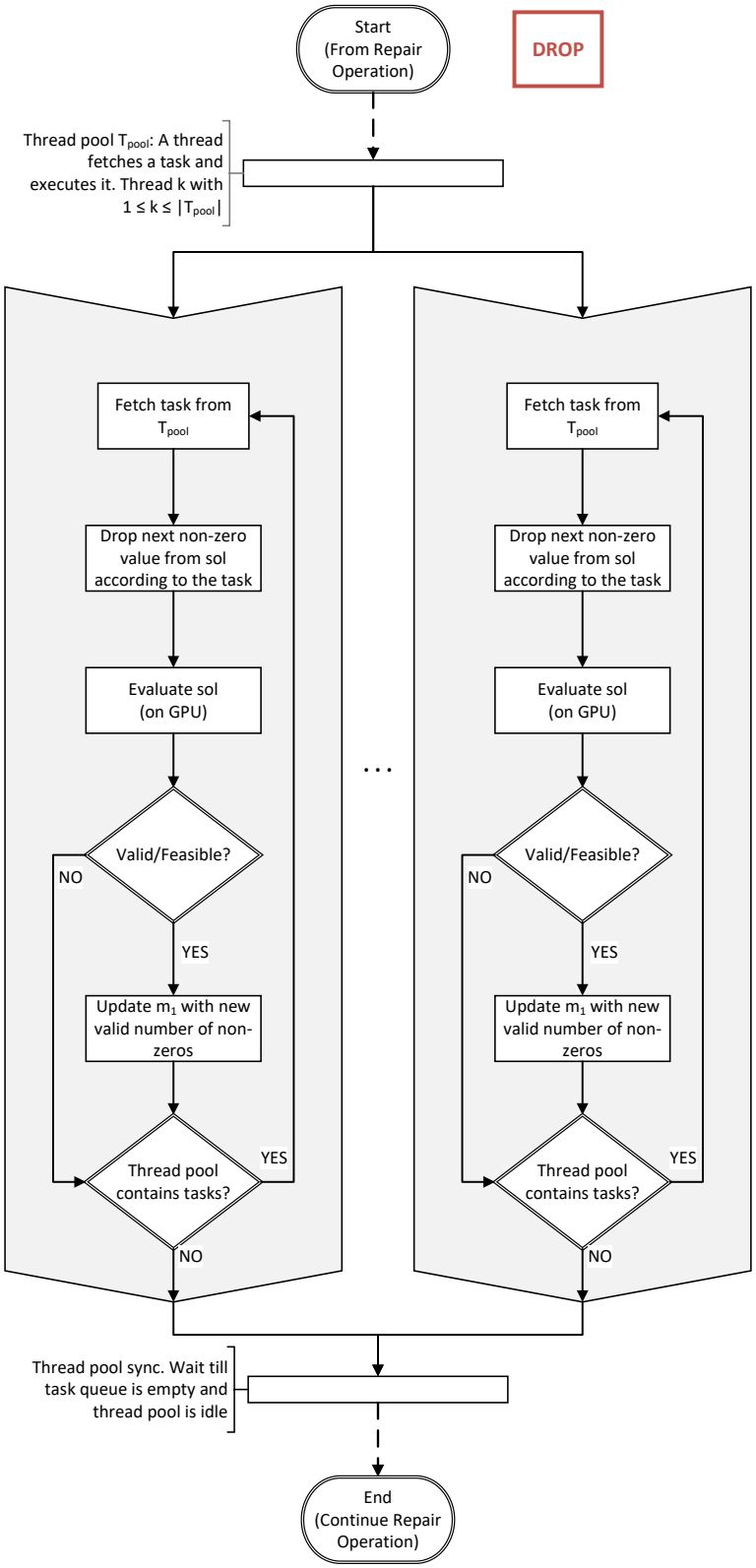


Fig. 3.10: Visualization of the drop operation. Note that the gray areas denote a host-thread. There are up to $|T_{pool}|$ threads and the number of tasks submitted to the thread pool T_{pool} is equal to the amount of non-zero indices of the current solution candidate sol. The evaluation operation in each thread is done on the GPU as described in section 3.5.2.

After $idxs$ has been initialized according to u_ratios_p, a thread pool T_{pool} is created using the maximum amount of threads the CPU can run earnestly in parallel alongside a mutex-object m, serving to guard a specific data field keeping track of the current state of non-zeros in the (about to be mutated) solution candidate sol.
Tasks are then pushed into the pool, each task is marked as gray area in Figure 3.9 and are run in parallel by the amount of threads $1 \leq k \leq |T_{pool}|$ available. Each task encompasses what Kern et al. call a "drop operation"[KLV20]. If a solution is unfeasible after a transformation operation within TLBO, the repair operator attempts to make it feasible again. The drop operation reads the variables/items of sol from right to left and drops a non-zero element, i.e. sets a non-zero value to 0, evaluates the solution with the dropped element and repeats until sol becomes feasible again.
The size of the thread pool depends on how many concurrent grids can be run on the device/GPU, i.e. how many streams can be launched as each thread launches its own evaluation operation which has one stream and 1 to m rows per stream, akin to the evaluation operation. The only difference is that the number of streams is determined from the "outside", by the threads. GPUs capable of supporting compute capability 7 or higher support the launch of several streams from several host threads.
The amount of tasks pushed into the pool depends on the number non-zero elements of sol. A thread fetches a task from the pool and drops the next 1-component assigned to it. Each task is assigned an index of a non-zero element where it acts as if all the other non-zero elements after it were already set to 0. E.g.:

$$sol = (1_0, 0_1, 0_2, 1_3, 1_4, 0_5, 1_6, 0_7, ..., 0_{n-1}) \in \mathbb{B}^n \tag{3.25}$$

non-zero elements of sol: 4 $\Rightarrow |T_{pool}| = 4$

Therefore:

$$thread_1(sol) \Rightarrow sol_{thread_1} = (1_0, 0_1, 0_2, 1_3, 1_4, 0_5, 0_6, 0_7, ..., 0_{n-1})$$
$$thread_2(sol) \Rightarrow sol_{thread_2} = (1_0, 0_1, 0_2, 1_3, 0_4, 0_5, 0_6, 0_7, ..., 0_{n-1})$$
$$thread_3(sol) \Rightarrow sol_{thread_3} = (1_0, 0_1, 0_2, 0_3, 0_4, 0_5, 0_6, 0_7, ..., 0_{n-1})$$
$$thread_4(sol) \Rightarrow sol_{thread_4} = (0_0, 0_1, 0_2, 0_3, 0_4, 0_5, 0_6, 0_7, ..., 0_{n-1})$$

After the evaluation operation, if the solution is still not feasible the tasks concludes and the thread fetches a new task from the pool. If the solution is feasible, the thread writes into a global index, readable and writable by all threads, its index of the dropped 1-component. However, the global index is guarded by mutex m_1 to avoid data races and additionally also takes care that the maximum amount of 1-components being kept that makes it feasible. I.e. only the minimal necessary amount of 1-components are attempted to be dropped (otherwise it would just drop all non-zero components).

3.5.3.3 Add

After dropping 1-components, the solution candidate sol is feasible again. The add operation intended by Kern et al. tries to optimize the objective score of the solution candidate by adding 1-components from left to right while keeping it feasible. The evaluation operation is parallelized in familiar fashion on the GPU as described in its section.

The parallelization scheme of the add operation can be seen in figure 3.11. Each gray area denotes a host-thread, i.e. a task. Each task copies sol locally, denoted by sol_c in the figure. Structurally, a task is similar to a task in the drop operation but instead of dropping a non-zero component, a non-zero component is added from left to right. Each task is grouped in a stride. There is only one stride with a certain stride.width (the number of threads) and sol is being iterated through using the stride-width, so essentially it is semantically not different from how the drop operation is processing a solution candidate (a number of thread processes sol) but a stride gives the "movement" of a thread an additional structure: It is required that the order of tasks a thread processes is kept. A stride is therefore used as a barrier before moving on to the next tasks, i.e. the next number of indices in sol to consider, and ensures that each thread k receives task $iteration \cdot stride\_width + k$.

Copying sol to a thread's specific sol_k is done using the inverse permutation vector u_ratios_{ip}, constructed from the utility ratios. I.e. the indices $i \in 0, ..., |sol|$ are translated to the real positions/indices in sol using u_ratios_{ip}. This is necessary to adhere to the "utility" that the utility ratio permutation vectors give to each position of a solution candidate while still being able to just iterate through the solution candidate elements. After a non-zero component has been added, sol_c is being evaluated on the device. If feasible, a global variable, holding the current smallest position index of the non-zero element to be added, is updated by a task. This global variable is guarded by mutex m_2. Furthermore, the global variable is only updated if it is the current smallest of the whole stride, i.e. if several tasks of the same stride attempt to update it, the smallest index will prevails. After a stride of tasks has finished, the position stored in the global variable is then used to add a non-zero element to sol at that corresponding position using u_ratios_{ip}.

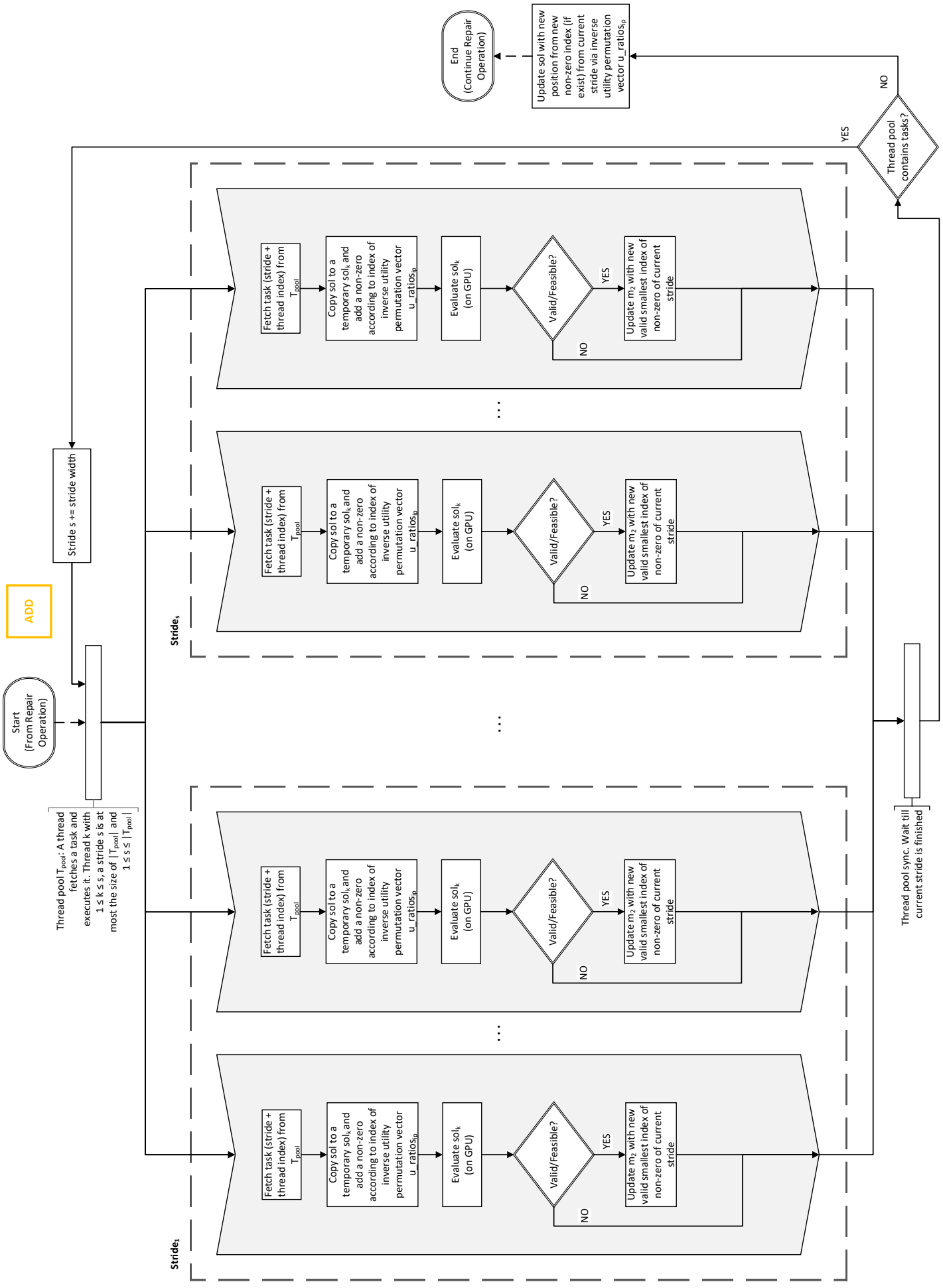


Fig. 3.11: Visualization of the add operation. Gray areas denote a thread, dashed boxes denote a stride.

3.5.3.4 Complexity

The first step of the repair operation is to initialize a vector $idxs$ of non-zero indices of a solution candidate sol permuted by an already-computed permutation vector u_ratios_p in parallel on the CPU: $O(\frac{n}{p_c})$, with p_c threads. Then, $idxs$ is sorted using std::sort in parallel as well. The C++ standard guarantees a std::sort time of $O(n \cdot log(n))$ (as of C++14 and newer), no official word is given on how parallelization might affect the time and therefore only the time guaranteed is considered. Thus, the time worst-case time before the drop operation is $\frac{n}{p_c} + O(n \cdot log(n)) = O(\frac{n}{p_c} + n \cdot log(n)) \Rightarrow O(n \cdot log(n))$.
The drop operation is done by assigning tasks to a thread pool. The size of the pool equals the number of non-zero elements of sol, here denoted as γ_1. A task drops a zero-value according to the index assigned to the task (done in $O(1)$) and checks feasibility via the application of the evaluation operation. However, the "twist" here is that one host-thread launches a stream. Each thread therefore launches a stream for its own version (local copy) of sol with different dropped non-zero elements; therefore γ_1 single-stream evaluation operations are launched and processed in parallel: $\gamma_1 \cdot T(\frac{m}{1} \cdot \frac{n}{p_t}) = T(\gamma_1 \cdot m \cdot \frac{n}{p_t})$, with p_t denoting the number of threads (per stream on the GPU), m the number of constraints or constraint matrix rows.
The add operation is similar in its processing as the drop operation, including the usage of the evaluation operation. Even through the traversal of sol is done using a stride, the amount of threads remains the same. Each thread makes a local copy of sol: $O(n)$. Each thread then adds a non-zero component according to the task index and inverse permutation vector and incurs a single-stream evaluation operation: $T(\frac{m}{1} \cdot \frac{n}{p_t}) = T(m \cdot \frac{n}{p_t})$, with p_t denoting the number of threads (per stream on the GPU), m the number of constraints or constraint matrix rows. The traversal of sol requires at worst $\frac{n}{p_k}$ iterations, with p_k being the number of threads, i.e. the stride-width. As such, the add operation requires $\frac{n}{p_k} \cdot T(m \cdot \frac{n}{p_t}) = T(\frac{n}{p_k} \cdot m \cdot \frac{n}{p_t})$.
Thus, the whole timing is

$$O(n \cdot log(n)) + T(\gamma \cdot m \cdot \frac{n}{p_t}) + T(\frac{n}{p_k} \cdot m \cdot \frac{n}{p_t}) \tag{3.26}$$

As γ might be, at worst (but unrealistically), approaching n, the term for the drop operation can be encompassed by $T(n \cdot m \cdot \frac{n}{p_t})$. That term for grows faster individually than $O(n \cdot log(n))$, i.e.:

$$O(n \cdot log(n)) < T(n \cdot m \cdot \frac{n}{p_t}) \in O(n^2 \cdot \frac{n}{p_t}) \tag{3.27}$$

The term for the add operation requires more careful observation: Letting the stride-width p_k be as if it were sequentially executed, i.e. $p_k = 1$, it converges similarly to the drop operation:

$$O(n \cdot log(n)) < \lim_{p_k \to 1} T(\frac{n}{1} \cdot m \cdot \frac{n}{p_t}) \in O(n^2 \cdot \frac{n}{p_t}) \tag{3.28}$$

And letting p_k grow up to n, the term that grows faster is now dependent on p_t:

$$O(n \cdot log(n)) \overset{?}{<} \lim_{p_k \to n} T(\frac{n}{n} \cdot m \cdot \frac{n}{p_t}) \in O(m \cdot \frac{n}{p_t}) \tag{3.29}$$

Thus, with consideration of p_t:

$$O(n \cdot log(n)) < \lim_{(p_t,p_k) \to (1,1)} T(\frac{n}{1} \cdot m \cdot \frac{n}{1}) \in O(n^3) \tag{3.30}$$

$$O(n \cdot log(n)) > \lim_{(p_t,p_k) \to (n,n)} T(\frac{n}{n} \cdot m \cdot \frac{n}{n}) \in O(n) \tag{3.31}$$

It can be observed that if parallel capabilities are utilized, the add operation falls within $O(n)$, meaning that the dominant term then is $O(n \cdot log(n))$, whereas for sequential observation it is at worst $O(n^3)$. And since $O(n^3) \geq O(n^2 \cdot \frac{n}{p_t})$ (from drop, eq. 3.27), the worst-case complexity for sequential processing is within $O(n^3)$. When parallel capabilities are considered, it is within $O(n^2)$, where drop is the dominant term (letting p_t there grow up to n): $O(n^2) > O(n \cdot log(n))$.
That being said, if observed more realistically, i.e. with $\gamma < n$, it would be within $O(n \cdot log(n))$. Work complexity has been considered here and is similar to the complexity observations of sequential computation as discussed for the evaluation operation.
Depth complexity $D(n)$ for drop equals the depth required to process one row, which is similar to the evaluation operation: $D_{stream}(n) = D_{prod}(n) + D_{red}(n) = 2 \cdot n + log_2(\text{blockDim.x})$. This is the same for the evaluation operation within the add operation.

With respect to the computation of the utility ratios permutation vector being done once at the beginning of TLBO, algorithms 9, 10 and 11 document the computational steps: The computation of the surrogate multipliers W_i, i.e. algorithm 9, consists of primarily a for-loop pushing m tasks, with m being the amount of constraints or constraint matrix rows of $A^{m \times n}$, to a thread pool, followed by a std::sort (done in parallel on the CPU) and last for-each-loop. The first for-loop at line 5 launches essentially $|T_{pool}|$ amount of streams on the device, each stream computing evaluating one row. Using the complexities discussed for the evaluation operation: $T(\frac{m}{|T_{pool}|} \cdot \frac{n}{p_t})$. The C++14 standard guarantees at least $O(n \cdot log(n))$ for std::sort (non-parallel

version) and the last for-each-loop at line 15 requires $T(|\text{tightnesses}|) \in O(m)$. Thus,

$$T(\frac{m}{|T_{pool}| \cdot \frac{n}{p_t}}) + O(n \cdot log(n)) + O(m) \tag{3.32}$$

$$\Rightarrow \lim_{(p_t, T_{pool}) \to (1,1)} T(\frac{m}{|T_{pool}|} \cdot \frac{n}{p_t}) + O(n \cdot log(n)) + O(m) \in O(n^2) \tag{3.33}$$

As the term for the evaluation operation is dominant for sequential operations. For scaling up parallel capabilities:

$$\lim_{(p_t, T_{pool}) \to (n,m)} T(\frac{m}{|T_{pool}|} \cdot \frac{n}{p_t}) + O(n \cdot log(n)) + O(m) \in O(n \cdot log(n)) \tag{3.34}$$

Algorithm 10 documents the computation for the utility ratios vector. Computationally, only the for-loop at line 4 is of interest: It pushed n tasks onto a thread pool of size $|T_{pool}|$, each thread essentially computes equation 3.23 on the host. Therefore:

$$T(\frac{m}{|T_{pool}|} \cdot n) \tag{3.35}$$

$$\Rightarrow \lim_{|T_{pool}| \to 1} T(\frac{m}{|T_{pool}|} \cdot n) \in O(n^2) \tag{3.36}$$

$$\lim_{|T_{pool}| \to m} T(\frac{m}{|T_{pool}|} \cdot n) \in O(n) \tag{3.37}$$

Algorithm 11 documents the computation for the permutation vector with which to sort the invalid/"broken" solution candidate. It calls algorithm 10, initializes the permutation $\mathbf{P}$ (done in $O(n)$) and calls std::sort (also in parallel on the host) with a guaranteed $O(n \cdot log(n))$. Therefore it is either bound by $O(n^2)$ due to the utility ratios function called, if the utility ratios function is computed sequentially. Otherwise the dominant bound is $O(n \cdot log(n))$ within $O(n) + O(n) + O(n \cdot log(n))$, with the first $O(n)$ bounding a parallelized utility ratios execution.
Additionally, the creation of the inverse permutation vector used by the repair operator is also called once and is structurally identical to algorithm 11 but works using the already computed permutation vector to created the inverse permutation vector and as such, not requiring algorithm 10. Therefore it requires only the initialization time done in $O(n)$ and time required by std::sort, also in parallel on the host) with a guaranteed $O(n \cdot log(n))$. Therefore it is bound by $O(n \cdot log(n))$.
Each of the four algorithms is called once during the computation and thus the timing of the whole depends on sequential or parallel execution observations: It is either bound by $O(n^2)$ or $O(n \cdot log(n))$ respectively.

3.5.4 Result And Initial Energy Of The Ray/Beam

After TLBO has finished, the sorted list of solution candidates, sorted from worst to best is returned. Depending on how many solution candidates should be considered, they are taken from the list. Each solution candidate might contain multiple beams, i.e. multiple 1-components in $\overrightarrow{x}$. Each 1-component denotes a beam angle from the isocenter to the respective cell the 1-component inhabits. This beam angle is determined via spherical coordinates (ρ, θ, ϕ). ρ represents the radius/distance from the cell center of the corresponding 1-component and the isocenter, θ represents the azimuthal angle and ϕ represents the polar angle. Calculated by the common formulas:

$$\rho = \sqrt{c_{j_x}^2 + c_{j_y}^2 + c_{j_z}^2} \tag{3.38}$$

$$\theta = atan2(c_{j_y} + c_{j_x}) \tag{3.39}$$

$$\phi = cos^{-1}\frac{c_{j_z}}{\rho} \tag{3.40}$$

With $(c_{j_x}, c_{j_y}, c_{j_z}) \in \mathbb{R}^3$ denoting the euclidean coordinates of the cell center corresponding to the index of j of x_j.
In addition, the initial energy E_0 required for the beam/ray to reach the isocenter is computed. The beam model used here for the application domain of proton radiation therapy also considers the target dose with which the beam should hit the isocenter as discussed more thoroughly in section 3.4.2 and section 2.2.2. The cell index of a cell corresponding to a 1-component x_j maps to the cell index of the corresponding cell p_j in the profits vector $\overrightarrow{p}$ holding the WEPL.

To consider the two main approaches (phenomenological, i.e. Bragg-Kleeman[NZ15], and approximation of relativistic, i.e. Bethe-Bloch[NZ15]) to model proton traversal and proton energy radiation/dissipation during travel through a medium (here a homogeneous medium: water), two methods have been considered to calculate E_0. The phenomenological Bragg-Kleeman model as reported by Newhauser et al.[NZ15] and already discussed in in section 2.2.2:

$$R - x = \alpha E^P \Leftrightarrow E(x) = \left(\frac{R-x}{\alpha}\right)^{\frac{1}{P}} \tag{3.41}$$

The other model considers the relativistic, more accurate Bethe-Bloch equation, which has also been officially adopted as the "gold standard" by the ICRU(ICRU Reports 50[Lan+93] and 62[Lan+99]) to measure against and often computed using Monte Carlo methods. However, as already discussed in the previous sections,

the model is impossible to compute directly. Ulmer et al.[UM10] provide an approximation of the the Bethe-Bloch equation. They approximated the integration operation by using a 4th-degree polynomial based on the power series expansion:

$$\sum_{k=0}^{\infty} a_n (x-c)^n \tag{3.42}$$

Providing a higher accuracy while still being computationally cheap: Computed in O(1). They fitted $\lambda_k \in \mathbf{\Lambda}$ and $c_k \in \mathbf{C}$ empirically to fit their function as closely as possible to the official ICRU curve for the proton ranges up to 300MeV with a deviation from the ICRU curve of 0.0013MeV:

$$\begin{gathered}
\mathbf{\Lambda} = \{10.256410256410255, 0.800006400051200, 0.175435518675111, \\
0.099501497497537, 0.009369626519191\} \\
\mathbf{C} = \{96.63872, 25.0472, 8.80754, 4.19001, 9.2732\} \\
E_0 = (wepl - z) \cdot \sum_{k=1}^{5} c_k \cdot e^{-\lambda_k \cdot (wepl - z)} \\
\text{with } c_k \in \mathbf{C}, \lambda_k \in \mathbf{\Lambda}
\end{gathered} \tag{3.43}$$

z denoting the residual range (for the purposes here it is 0).

For each 1-component of a solution candidate, a list of 4-tuples is returned:

$$[(E_{0a}, \rho_a, \theta_a, \phi_a)], a = 1, ..., \#\text{non-zero elements of solution candidate} \tag{3.44}$$

4 Evaluation And Discussion

This chapter presents and discusses the quality of the system presented in the previous chapter and if the goals stated in the beginning were achieved and to what degree. A main source of radiation therapy data sets is the Imaging Cancer Archive (ICA)[Pri+17], a database with numerous data sets of CT scans and additional treatment data for various treatment modalities. A major difficulty during the creation of this thesis has been finding suitable data sets in the application domain of proton therapy: DICOM structure variations and/or missing/inconsistent meta information and incomparable environments on which they were obtained/generated were an unpleasant factor. Furthermore, purposely missing information due to being generated by implementations of algorithms and methods within proprietary systems (often in cooperation with device manufacturers) were also encountered.

Luckily, researchers in the domain identified these problems as well and efforts were made to build "standardized" or easily comparable data sets specifically for the purpose of evaluating treatment planning systems. One of the more famous ones, which shall also be used in the evaluation here, is the CORT data set[Cra+14], as well as the Alderson-Phantom[SVS08]. More detailed discussions on these data sets can be found in section 4.2.

As the application domain for the system presented in the previous chapter is proton radiation therapy, it makes sense to utilize the domain's methodology of evaluating the quality of a treatment plan, of which the beam angles are a crucial factor. This is commonly done using dose-volume histograms (DVH)[KG14], which allow for a qualitative analysis and comparison of treatment plans and will be explained in the next section.
Naturally, timings will be presented for the critical parts of the system: The generation of the cost map (i.e. ray tracing) as well as the optimization (TLBO).

4.1 Dose-Volume Histograms (DVH)

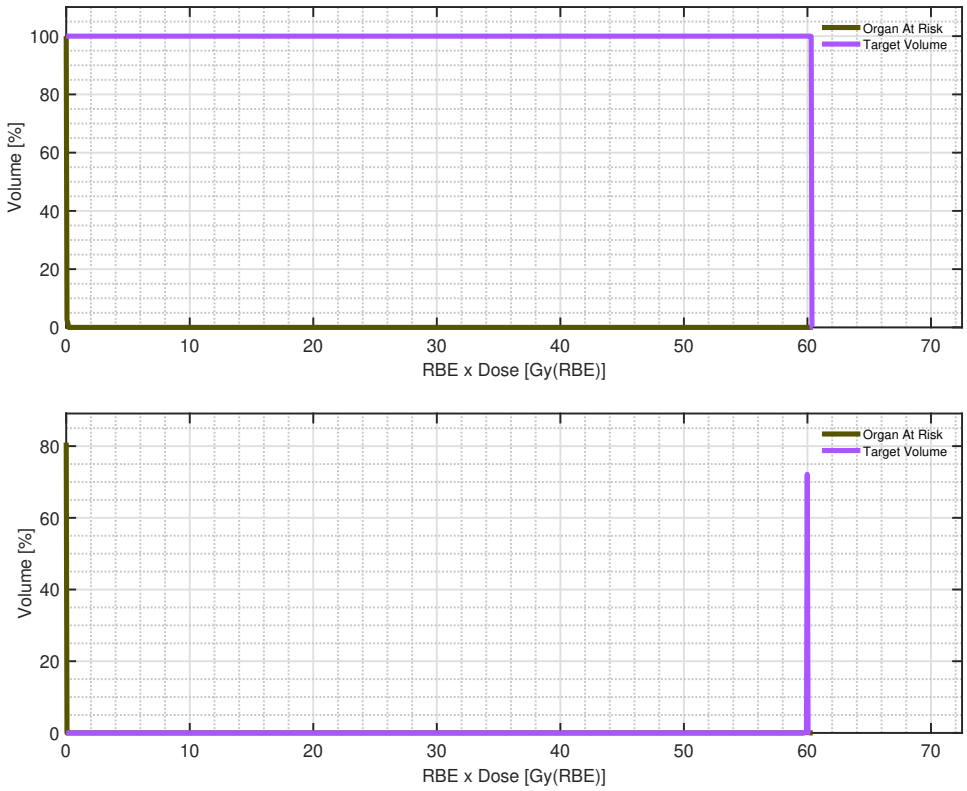


Fig. 4.1: Top: A theoretical ideal cumulative DVH (cDVH).
Bottom: A theoretical ideal differential DVH (dDVH).

Dose-Volume histograms (DVH) are the common tool in radiation treatment planning to assess the quality of a plan. A DVH represents a summarization of 3D dose distributions as a 2D graph and relates the amount of dose received by which amount of voxels of a volume. Two types can usually be encountered in related literature: The cumulative DVH (cDVH) and the differential DVH (dDVH). The independent variable is expressed by the domain, being expressed as absolute or relative value in Gy (Gray). The dependent variable is expressed by the codomain and denotes the absolute or relative value of the volume receiving a certain amount of dose. Note that this morphism is essentially a "binning", i.e. discretization, of the domain (a histogram)[KG14]. This thesis expresses the volume receiving the doses relatively (in %).

A dDVH, visualizes the amounts variation of dose (dose bin) a volume is receiving, i.e. a spectrum of dose distribution for a volume. E.g. the bottom dDVH in figure 4.1 shows that for the target volume, a bit above 70% of voxels of that volume receive 60Gy. The rest of the volume voxels receive close to 0% (but it sums up to 100% of

volume voxels considered). Whereas for the organ at risk, it can be seen that 80% of the volume voxels receive close to 0Gy.
Generally, the narrower the peak in a dDVH, the more homogeneous the dose distribution in a volume is. The wider the peak, the more spread-out the dose distribution in a volume is. That is, voxels of a volume receive different doses. Ideally, it is desired for the voxels of the target volume to receive the same dose, so that the target volume is being destroyed a homogeneous manner. Thus reducing the number of fractions, beams and maybe sessions, even possibly simplifying the treatment plan itself.

A cDVH can be calculated from a dDVH by integration of the dDVH from a dose bin D to ∞, for every dose bin. The cDVH visualizes the amount of cumulative dose (or higher) a volume receives. E.g. in the top cDVH in figure 4.1, the target volume graph shows that 100% of the volume receives up to 60Gy, whereas 100% of the organ at risk receives up to close to 0Gy, the rest of the graph is close to 0% for up to 60Gy.
Organs at risk in cDVH should essentially strive to mimic the curve of a very sharp/steep $log(\frac{1}{x})$, whereas the graphs for the target volumes should strive to mimic the curve of $-e^{x^c} + 100$, with $c \in \mathbb{N}$, c being very large.

4.2 Evaluation

OS	Microsoft Windows 10, Version 10.0.19042, Build 19042
CPU	AMD Ryzen 7 5800X @3800 Mhz
RAM	32 GB
GPU	nVidia GeForce RTX 3080

Tab. 4.1: Test system specifications.

For the purpose of evaluating the system presented in the previous chapter, the CORT data set[Cra+14] has been chosen, being the result of one of the more recent efforts to produce a purpose-built data set for evaluating treatment planning systems. In addition, the Alderson-Phantom[Pri+17] will be utilized as well. These data sets also come with volume/organ segmentations/contours as well as constraint/penalties and further additional information to use as treatment planning parameters.
Due to the nature and approach of the system developed in this thesis, the constraints utilized are interpreted from a more geometrical/physical point of view. Medical constraints were considered via the usage of a penalty map; a voxel grid with the same dimensions and resolutions of the CT of the respective data set but holding clinical penalty values for each voxel, i.e. voxels of an organ at risk have a higher penalty than non-OARs as well as target volumes. Note that this approach is not clinically

verified, only a suggestion with the application area of proton therapy radiation. A radiologist and/or oncologist are required to review and consider the beam angle suggestions given by the system. Further note that contouring/segmentation of the OARSs and PTVs do have an impact on DVH evaluation as well as they might differ (slightly) for the same patient for different setups, e.g. patient positions, devices used, oncologists generating the treatment constraints, etc. Table 4.2 documents the general parameters of each scenario used.

	Head And Neck	**Prostate**	**TG119**
CT Dimensions	$161 \times 161 \times 67$	$183 \times 183 \times 90$	$167 \times 167 \times 129$
Voxel Dimensions	$3 \times 3 \times 5$(mm)	$3 \times 3 \times 3$(mm)	$3 \times 3 \times 2.5$(mm)
# CT/Cuboid Voxels	1736707	3014010	3597681
# CuboidSurface Voxels	94990	132858	141950
	Alderson-Phantom	**Box-Phantom**	**Liver**
CT Dimensions	$155 \times 155 \times 170$	$160 \times 160 \times 160$	$217 \times 217 \times 168$
Voxel Dimensions	$2 \times 2 \times 2$(mm)	$3 \times 3 \times 3$(mm)	$3 \times 3 \times 2.5$(mm)
# CT/Cuboid Voxels	4084250	4096000	7910952
# CuboidSurface Voxels	153450	153600	240002

Tab. 4.2: Overview of data set scenarios. The scenarios are ordered by the number of CT voxels, ascending (left to right).

4.2.1 Cost Map

Figure 4.2 shows average timings (in milliseconds) for each scenario for the cost map generation/ray tracing, using the slightly modified method by Xiao et al.[Xia+12], discussed in section 3.4. Each time measurement is an average over 10 runs. Each run consists of ray from the voxel of interest (the isocenter) towards each surface voxel of a cuboid face. This is done in parallel for each side as far as the GPU capabilities allow. For the GPU used here for testing purposes (refer to table 4.1), full parallel processing was always achieved, without the need for queuing tasks due to memory limitations. Timings include the copying of data to and from the device, as well as the initialization of the CUDA-surfaces and additional data structures.

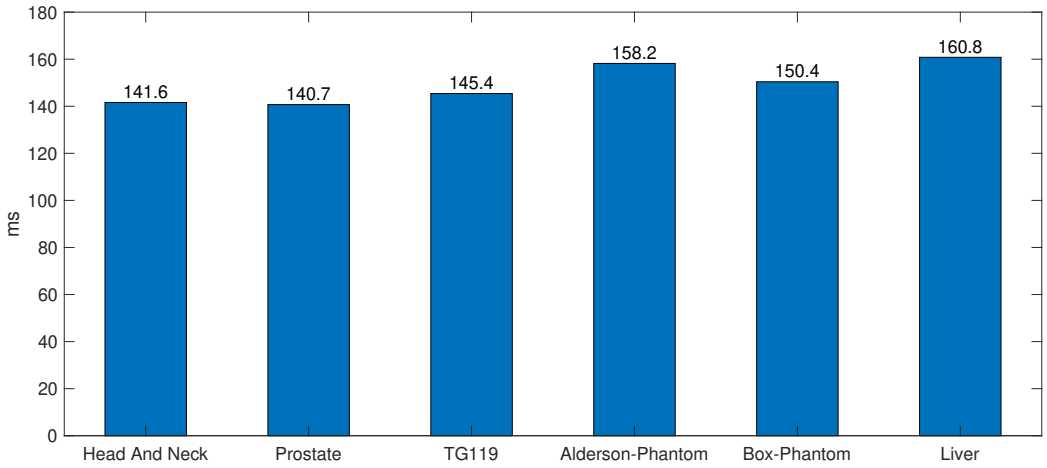


Fig. 4.2: Timings of the CostMap-step (Ray Tracing), discussed in section 3.4, via the slightly modified method of Xiao et al[Xia+12]. All 6 faces of a cuboid are done in parallel for a data set. Timings in milliseconds, average of 10 runs for each scenario.

4.2.2 TLBO

The evaluation of the TLBO consists of two observation areas: Qualitative evaluation of the results and quantitative evaluation of the time required. Since TLBO attempts to solve the 0/1-multidimensional knapsack problem (0/1-MKP), introduced in section 2.4.1, which lies within NP, it seems to be the most useful to present the timings for each scenario as a 3-tuple of (min, max, average) time required for an iteration of the main loop, i.e. the "most outer" loop of TLBO (see algorithm 6). These values are taken from 100 iterations. For the qualitative results, the computed beam angles will be used.

Those computed beam angles are then applied as the beam angles used in the treatment planning system matRad[Wie+17], calculating the fluence matrix and final dose distributions for a scenario. The results for each scenario are presented here using a dDVH and cDVH respectively, as well as visualizations of dose distributions as heat maps for each of the principal axes in medical physics (axial, sagittal, coronal).

However, not only are the number of voxels of a cuboid surface object affecting the time required, but also the number of beams and constraints. Moreover, the constraint to ensure a minimum distance between beams, namely C_2 (eq. 4.2), (so that they don't fall all into one position, which could happen since TLBO doesn't check for identical solutions but rather just tries to converge to an optimum) is what can be considered to be a "dynamic constraint" or "dependent constraint" as it relies on the number of beams. Internally, this is done by reading the number of

beams from the boundary condition of the boundary set B to the supplemental data buffer and then being read from there when C_2 (or another dependent constraint) is selected by a kernel for computation. As such, the independent constraints need to be fed into the system first.
A listing of used constraints for the evaluation can be found below. Note that C_1 and C_2 are always used. The notions of "1 Constraint", "2 Constraints" and "3 Constraints" found in the visualizations that follow and tables 4.3 and 4.4 are equivalent to the number of penalty-constraints used, i.e. "1 Constraint" $\equiv \{C_1, C_2, C_3\}$, "2 Constraints" $\equiv \{C_1, C_2, C_3, C_4\}$ and "3 Constraints" $\equiv \{C_1, C_2, C_3, C_5\}$.
A note on the nature of the constraints: The data of each constraints is also a cuboid surface object of the same size and dimensions as the "profit" (i.e. WEPL) data. Each voxel element v_{c_i} of a constraint cuboid surface c_i can be any valid supported data type or even a vector over a field, as in C_2. The penalty values of C_3 are taken from the clinical constraint penalties given for each scenario found in the data set.

$$C_1 = (uint64, \text{"MAX\_BEAMS"}, \leq, f_{max\_beams}) \tag{4.1}$$
$$C_2 = (double3, \text{"MIN\_DIST\_SPHERICAL"}, \geq, f_{min\_dist\_spherical}) \tag{4.2}$$
$$C_3 = (double, \text{"PENALTY"}, \leq, f_{penalty}) \tag{4.3}$$
$$C_4 = (double, \text{"PENALTY"}, \leq, f_{penalty}) \tag{4.4}$$
$$C_5 = (double, \text{"PENALTY"}, \leq, f_{penalty}) \tag{4.5}$$

The structure of the constraints are akin to equation 3.15 as discussed in section 3.5.2.

$$\mathbf{F_c} = \{f_{max\_beams},\ f_{min\_dist\_spherical},\ f_{penalty_1},\ f_{penalty_2},\ f_{penalty_3}\} \tag{4.6}$$
$$\mathbf{F_{rel}} = \{\leq, \geq, \leq, \leq, \leq\} \tag{4.7}$$
$$\mathbf{B} = \{b_1,\ b_2,\ b_3,\ b_4,\ b_5\} \tag{4.8}$$

1 to 3 beams are used together with (2 +) 1 to 3 constraints. The beam number of 3 to 5 appears to be common in related literature, so up to 3 will be used here. The reasoning for the amount of constraints is that $\{C_1, C_2, C_3\}$ appear to be the required constraints to have at least somewhat potentially useful results. The additional penalties, C_4 and C_5, are just using the data from C_3 but pronounce it a bit more, adding/subtracting penalty values according to the importance given by the values of C_3, without changing clinical semantics introduced by the penalty values for voxels defined in C_3. They rather just introduce more computational

complexity.
Lastly, the timings for the initial population generation were all taken over 50 runs, collected as a 3-tuple (min, max, average) of time required to finish the population generation. Generally, a population size of 32 has been chosen as this is the around the number of what Kern et al.[KLV20] recommend.
A discussion of the results follows in section 4.3.

4.2.2.1 Initial Population Generation Timings

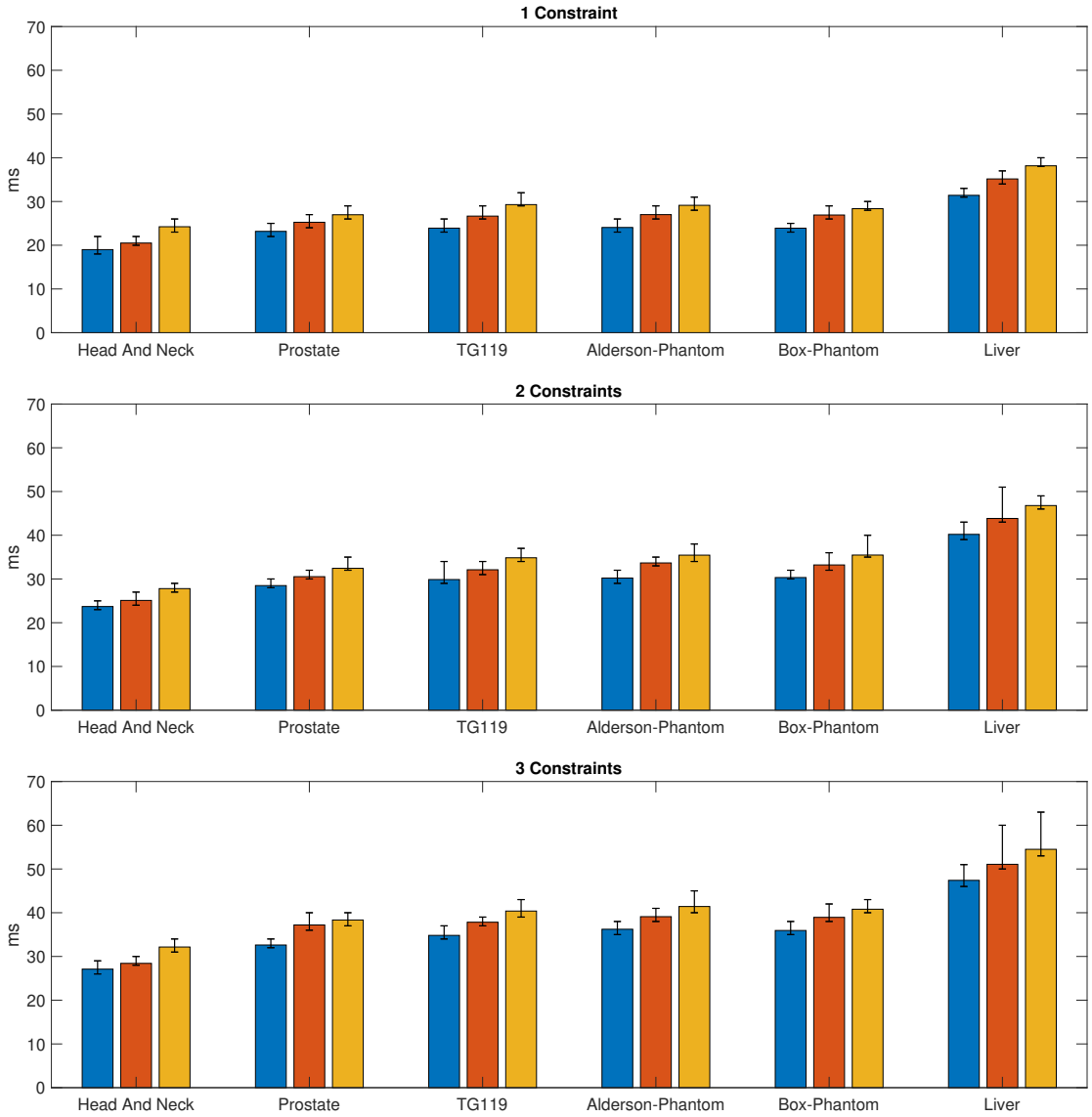


Fig. 4.3: Initial population generation timings visualized for easier comparison. The error bars denote the most and least amount of time required for a for the generation. The average time required is denoted as a bar. Blue bar denotes 1 Beam, red bar 2 beams, yellow bar 3 beams. Timings were taken from 50 runs for each condition (number of constraints, number of beams). See table 4.3 for a more detailed reference.

# Constraints	# Beams/Rays	Head And Neck			Prostate			TG119		
		min	max	avg	min	max	avg	min	max	avg
1	1	18	22	19.0	22	25	23.17	23	26	23.91
	2	20	22	20.50	24	27	25.25	26	29	26.66
	3	23	26	24.23	26	29	26.97	29	32	29.29
2	1	23	25	23.70	28	30	28.49	29	34	29.87
	2	24	27	25.10	30	32	30.53	31	34	32.11
	3	27	29	27.80	32	35	32.44	34	37	34.85
3	1	26	29	27.10	32	34	32.62	34	37	34.82
	2	28	30	28.42	36	40	37.20	37	39	37.83
	3	31	34	32.16	37	40	38.34	39	43	40.38

# Constraints	# Beams/Rays	Alderson-Phantom			Box-Phantom			Liver		
		min	max	avg	min	max	avg	min	max	avg
1	1	23	26	24.06	23	25	23.90	31	33	31.41
	2	26	29	27.0	26	29	26.91	34	37	35.16
	3	28	31	29.13	28	30	28.36	38	40	38.16
2	1	29	32	30.21	30	32	30.33	39	43	40.20
	2	33	35	33.67	32	36	33.21	43	51	43.84
	3	34	38	35.45	35	40	35.47	46	49	46.79
3	1	35	38	36.22	35	38	35.94	46	51	47.40
	2	38	41	39.10	38	42	38.94	50	60	51.07
	3	40	45	41.42	40	43	40.80	53	63	54.50

Tab. 4.3: Initial population generation timings of 50 runs: min | max | avg. (in milliseconds). Min denotes the minimum time required of a run out of 50, max denotes the maximum time required of a run out of 50 and avg. denotes the average time of run out of 50. The data sets are ordered by the number of CT voxels, ascending (left to right).

4.2.2.2 Optimization (Main Loop) Timings

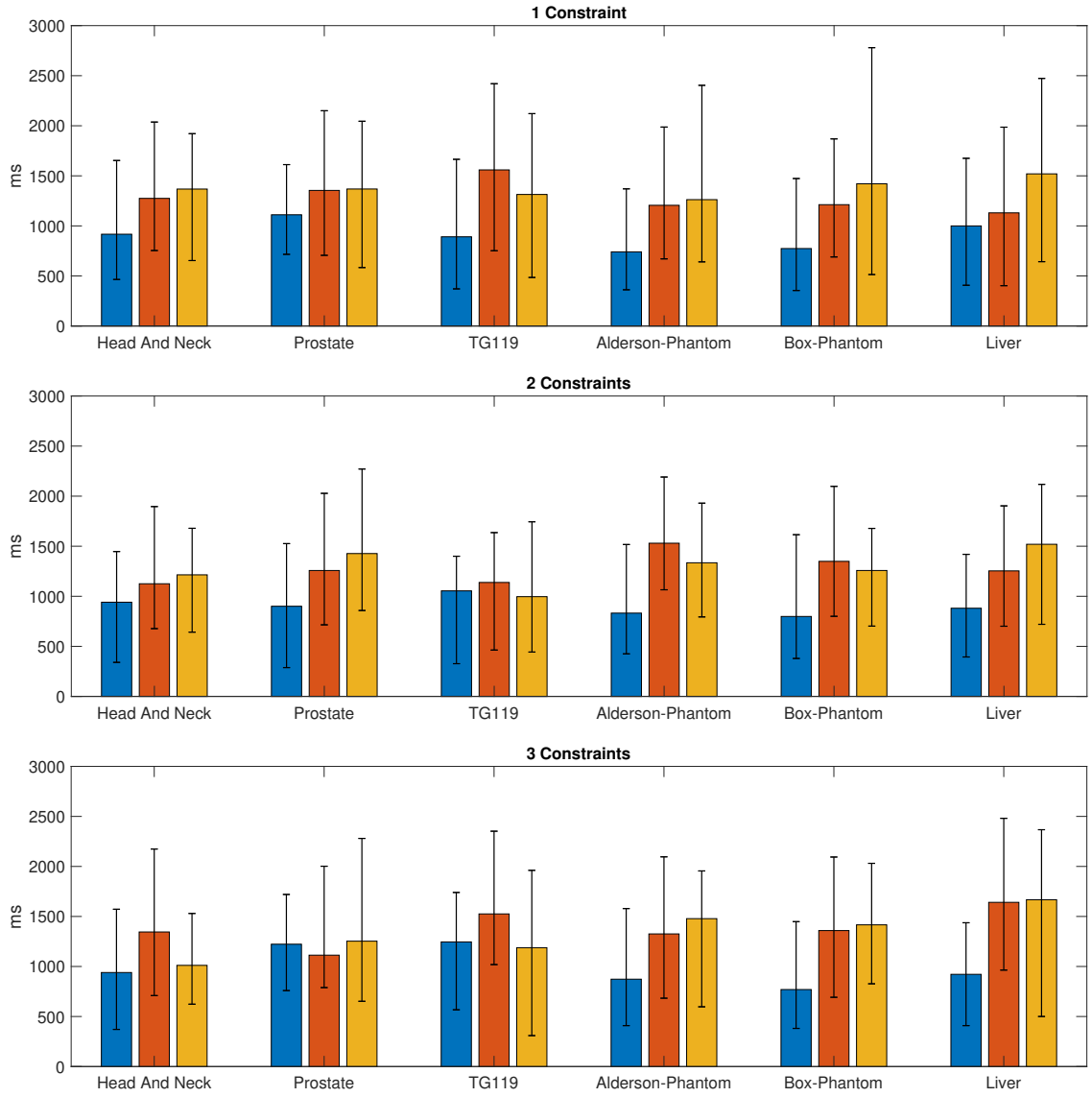


Fig. 4.4: TLBO timings visualized for easier comparison. The error bars denote the most and least amount of time required for a TLBO main loop iteration. The average time required for a loop is denoted as a bar. Blue bar denotes 1 Beam, red bar 2 beams, yellow bar 3 beams. Timings were taken from 50 iterations for each condition (number of constraints, number of beams). See table 4.4 for a more detailed reference.

# Constraints	# Beams/Rays	Head And Neck			Prostate			TG119		
		min	max	avg	min	max	avg	min	max	avg
1	1	465	1653	917.06	716	1612	1111.03	371	1665	891.23
	2	745	2037	1274.86	706	2150	1354.50	752	2420	1559.36
	3	654	1921	1368	584	2045	1368.53	485	2121	1314
2	1	340	1447	940.90	288	1526	900.50	327	1400	1054.20
	2	677	1896	1125.20	715	2028	1257.63	463	1636	1138.20
	3	642	1679	1214.30	858	2271	1426.76	443	1744	996.43
3	1	371	1572	939.73	759	1720	1222.66	567	1740	1245.55
	2	708	2174	1345.03	789	2002	1113.16	1019	2352	1525.41
	3	623	1530	1011.22	652	2280	1253.03	309	1961	1187.71

# Constraints	# Beams/Rays	Alderson-Phantom			Box-Phantom			Liver		
		min	max	avg	min	max	avg	min	max	avg
1	1	362	1370	740.80	355	1473	773.83	407	1675	999.90
	2	671	1987	1205.70	690	1869	1212.06	403	1986	1130.26
	3	641	2404	1262.33	514	2780	1420.53	643	2473	1519.70
2	1	425	1517	831.93	379	1615	798.63	394	1419	880.63
	2	1065	2191	1530.13	800	2098	1348.93	701	1903	1253.63
	3	794	1930	1334.63	702	1678	1257.70	720	2118	1519.46
3	1	409	1579	873.06	381	1448	769.60	409	1437	921.66
	2	683	2096	1325.50	692	2094	1358.97	964	2481	1641.15
	3	597	1955	1478.12	827	2031	1416.29	500	2367	1667.11

Tab. 4.4: TLBO timings of 50 main loop iterations: min | max | avg. (in milliseconds). Min denotes the minimum time required of an iteration out of 50, max denotes the maximum time required of a loop out of 50 and avg. denotes the average time of a loop out of 50. The data sets are ordered by the number of CT voxels, ascending (left to right).

4.2.2.3 Optimization (TLBO) Results

In the following pages, qualitative results for the "top" returned 3 beams are shown for each scenario. The top shows the differential DVH and the cumulative DVH. The middle and bottom shows axial, sagittal and coronal views of the scenario with the beams applied and corresponding dose distributions as heat maps. matRad is then given these three beams for each scenario to compute the best possible dosimetry to hit the tumor with to fullfill the given radiation dose target (also found in the meta data of the data sets), including the amount of Gray to hit the target with.

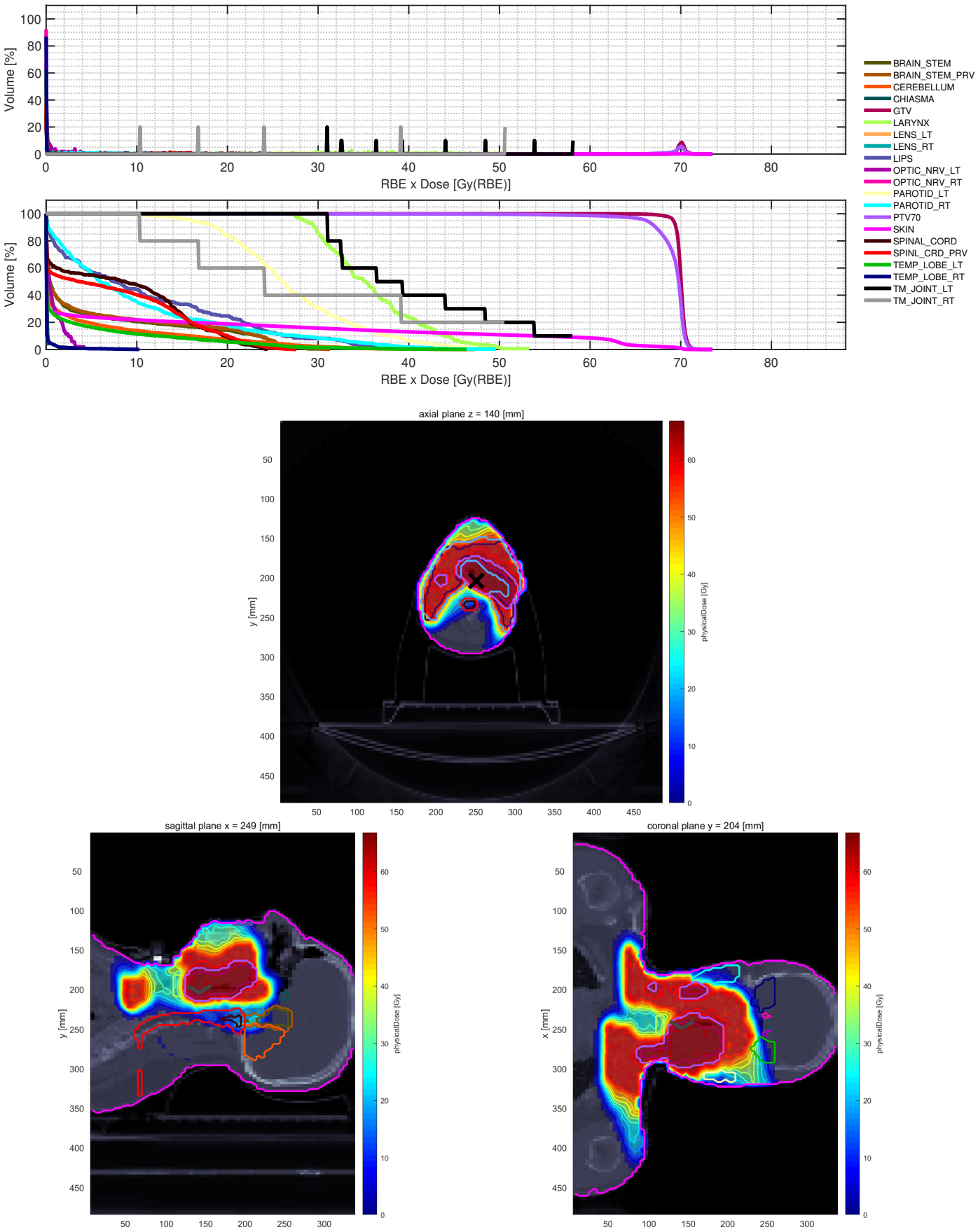


Fig. 4.5: "Head And Neck" results. Top: dDVH, middle: cDVH, bottom: (top to bottom right) axial view, sagittal view, coronal view. 3 Beams, (gantry angle, couch angle): (-73°, 157°), (-63°, 180°), (-33°, -154°).

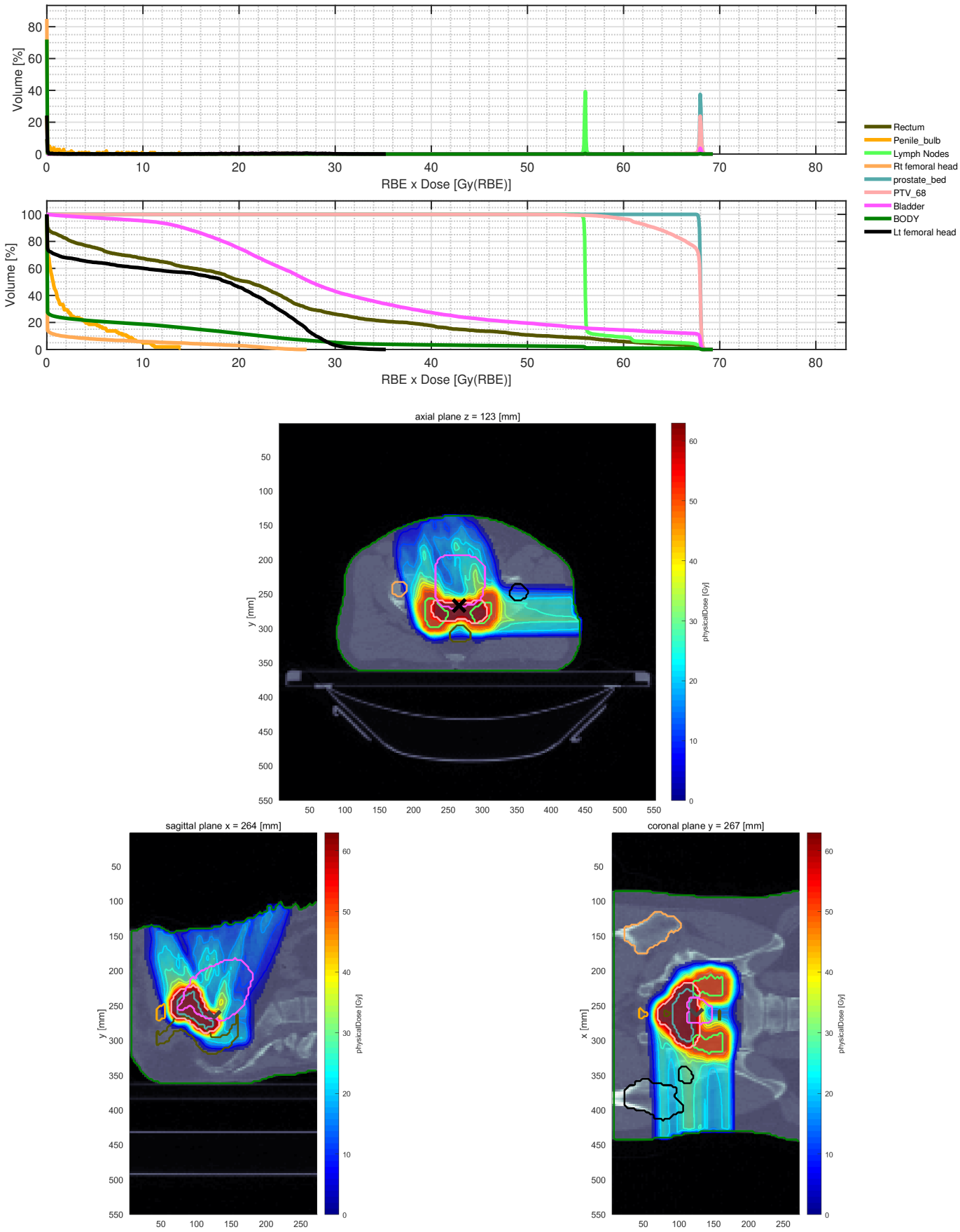


Fig. 4.6: "Prostate" results. Top: dDVH, middle: cDVH, bottom: (top to bottom right) axial view, sagittal view, coronal view. 3 Beams, (gantry angle, couch angle): (-30°, -57°), (-21°, 61°), (-90°, -180°).

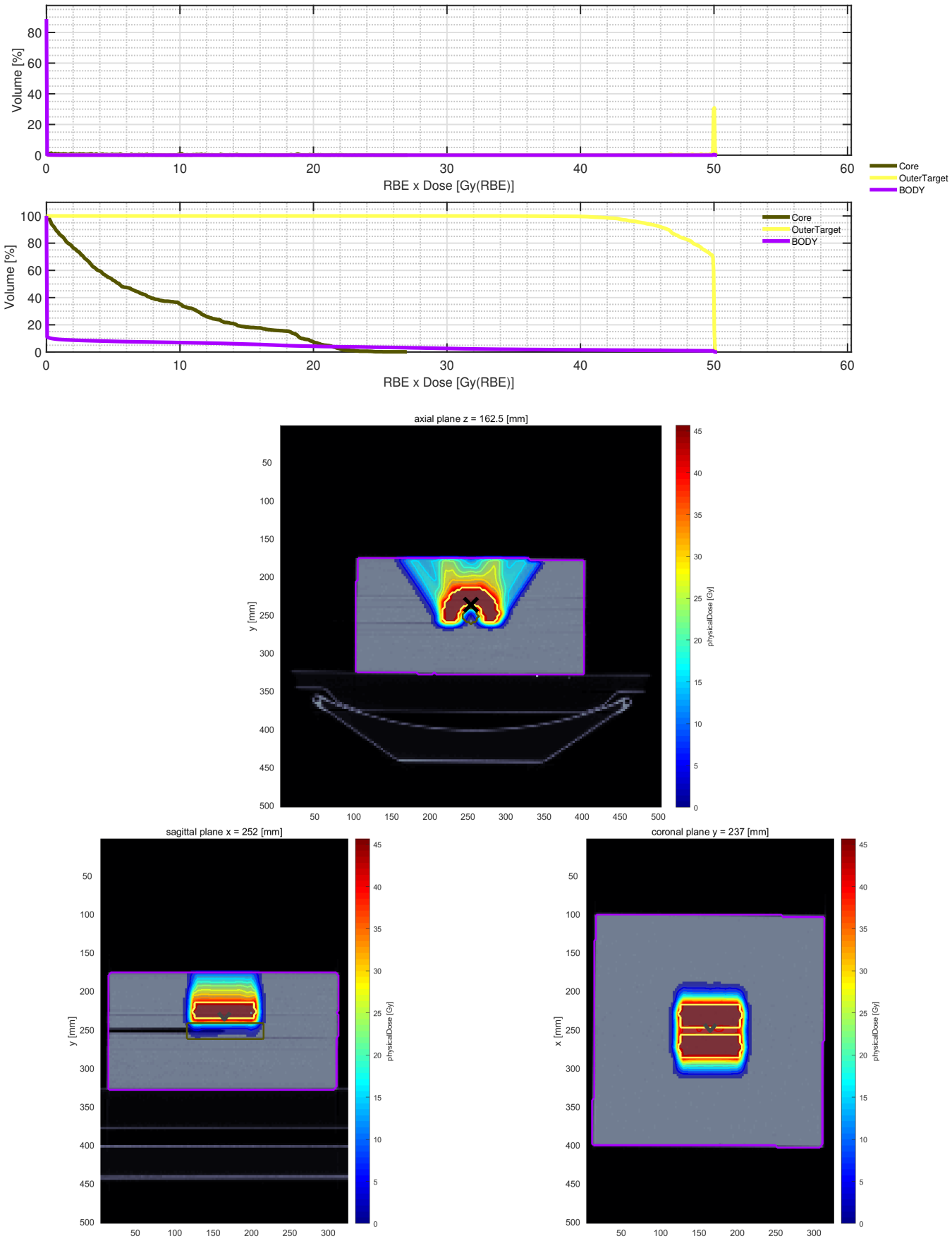


Fig. 4.7: "TG119" results. Top: dDVH, middle: cDVH, bottom: (top to bottom right) axial view, sagittal view, coronal view. 3 Beams, (gantry angle, couch angle): (-35°, 0°), (0°, 0°), (35°, 0°).

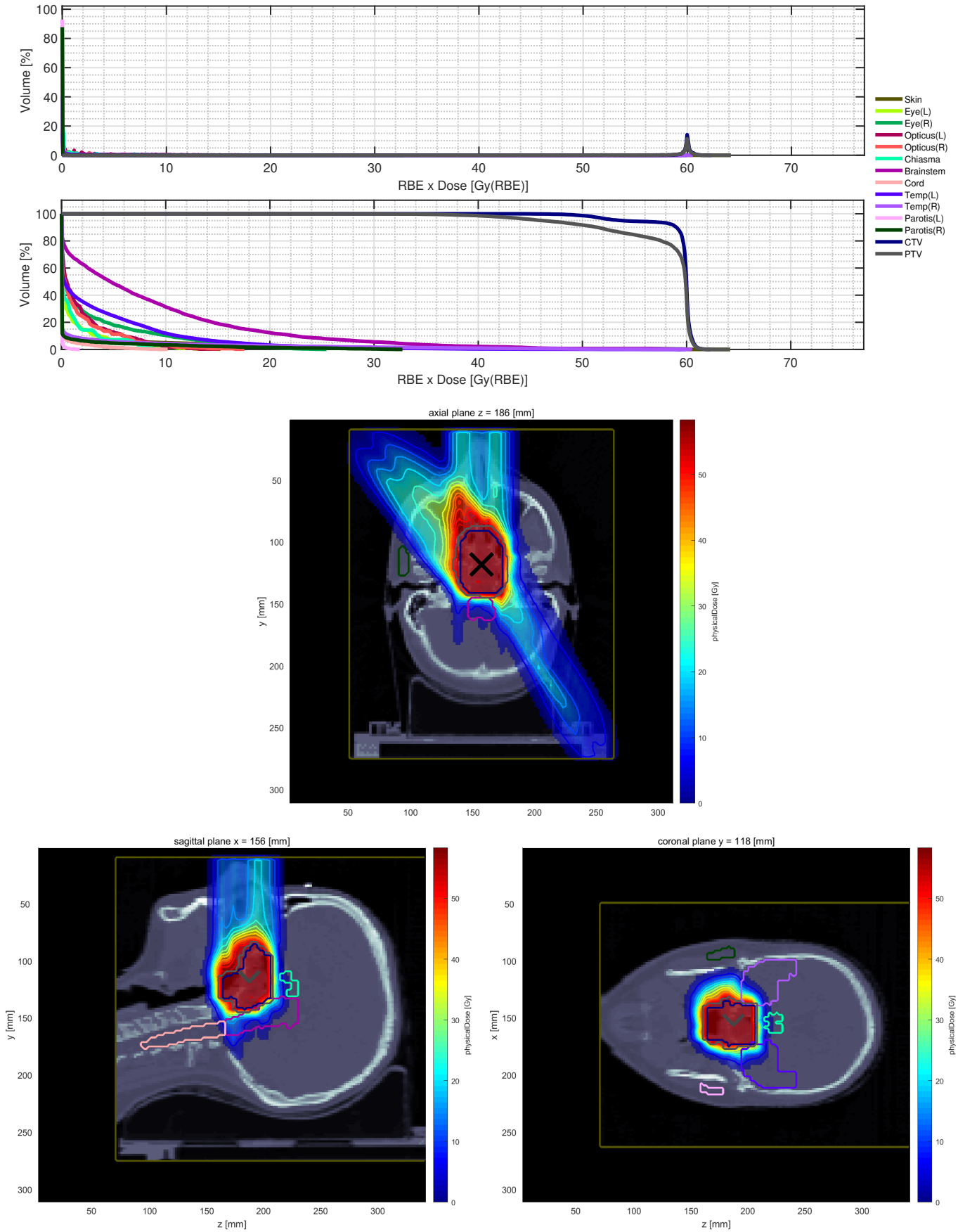


Fig. 4.8: "Alderson-Phantom" results. Top: dDVH, middle: cDVH, bottom: (top to bottom right) axial view, sagittal view, coronal view. 3 Beams, (gantry angle, couch angle): (0°, -15°), (-45°, -16°), (-214°, -17°).

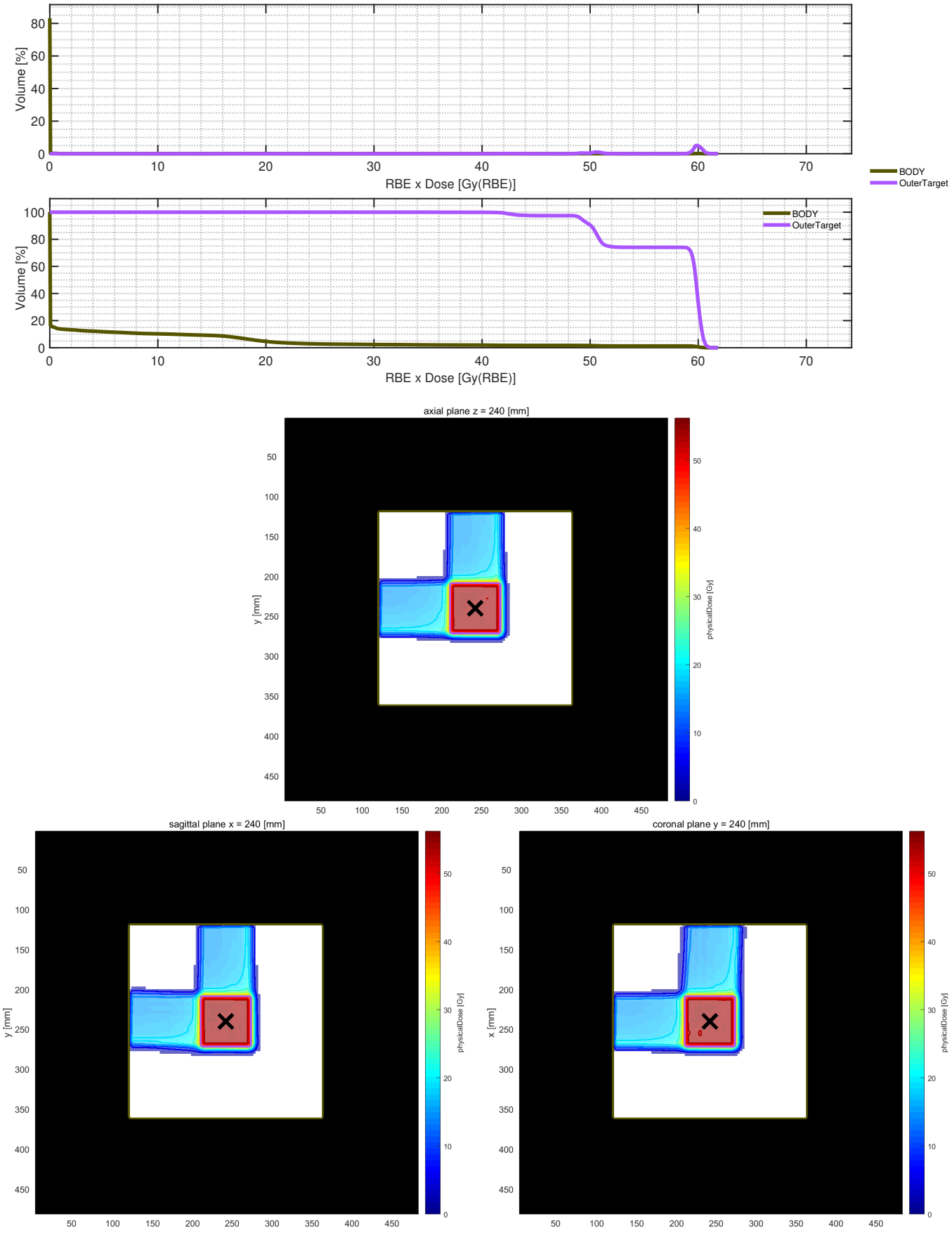


Fig. 4.9: "Box-Phantom" results. Top: dDVH, middle: cDVH, bottom: (top to bottom right) axial view, sagittal view, coronal view. 3 Beams, (gantry angle, couch angle): (0°, 0°), (90°, 182°), (90°, 90°).

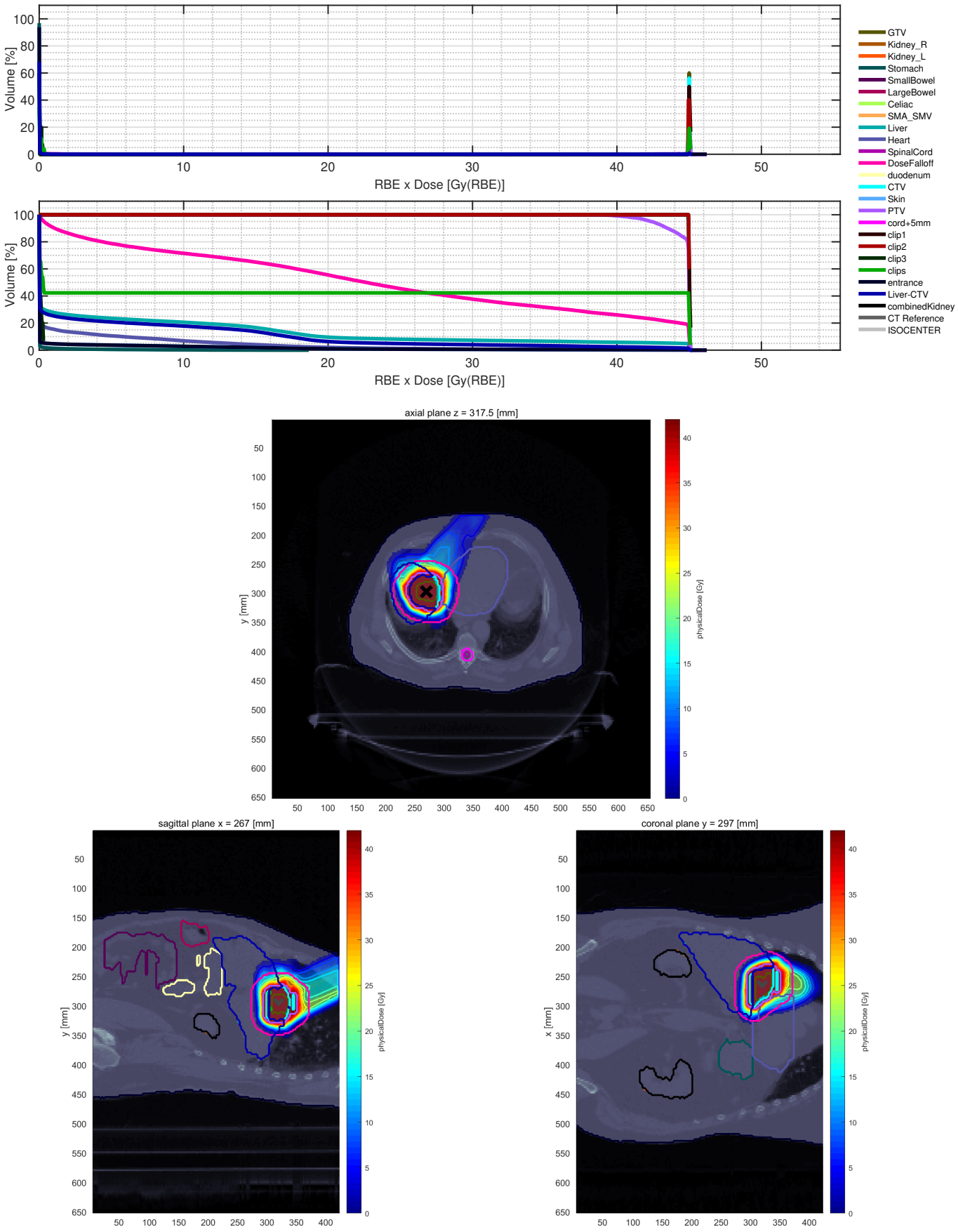


Fig. 4.10: "Liver" results. Top: dDVH, middle: cDVH, bottom: (top to bottom right) axial view, sagittal view, coronal view. 3 Beams, (gantry angle, couch angle): (46°, -138°), (46°, 20°), (64°, -90°).

4.3 Discussion

The fist component of the system to undergo evaluation is the cost map generation. Figure 4.2 lists the timings over 10 runs for each scenario. All 6 faces of the cuboid are processed in parallel, tracing paths from the voxel of interest to the respective surface voxel. Generally, the time required scales as expected: The more voxels, the more time required. An outlier is the Alderson-Phantom, requiring an average time of 158.2ms, being very close to the Liver scenario with 160.8, despite having almost less than double the voxels. The Box-Phantom has just slightly more voxels than the Alderson-Phantom but requires 7.8ms less. It is notable however, that the Alderson-Phantom has the smallest resolution with 2mm for each voxel dimension, whereas all the other data sets generally have around 3 for most of their voxel dimensions. That being said, the overall timings are actually not bad at all and certainly can be deemed acceptable, especially with respect to the optimization step (TBLO) of the whole system. An additional note: The utilization of a CUDA-surface certainly affected the time required in a positive way, as it measurably reduced cache misses. The alternative would have been to transpose/rotate the cube before a kernel call for a cube face with voxel indices not aligned for linear memory access patterns, which would have required more time in total.

The optimization step is the "main course" of the system, especially with the overall time required. Due to the core of the problem being a 0/1-MKP and thus being solved during the optimization step by applying the TLBO metaheuristic, there is a non-deterministic nature regarding the timings and the quality of results, i.e. the overall converging behaviour, which are not always guaranteed to converge towards the global optimum. As such, there is a variance to the solutions and timings reported.
This also holds true for the first substep of TLBO; the generation of the initial population. This substep already utilizes (biased) randomness to generate candidates to be rejected or accepted into the population. Figure 4.3 and table 4.3 give an overview of minimum, maximum and average time required over 50 runs of the population generation for each scenario and each condition. Generally, it scales as expected: The more voxels and constraints needed to be processed, the longer the time required. Liver being especially time-consuming due to its number of voxels. This also translates to the number of voxels needed to be considered for each constraint. Depending on the GPU, there is a threshold of how much a kernel can process before it needs to queue data that is "too large to fit", causing additional memory movement and allocations. However, for each scenario and each condition presented, this did not occur with the GPU used for testing (see table 4.1). The system also reported the biggest variance of timings for the Liver scenario, with the curious observation that for the condition of 2 beams having the most variance

reported. The difference between the Alderson-Phantom and Box-Phantom is slightly diminished here in comparison to the cost map generation but still observable. It does seem that the lower the voxel resolution is, the more costly related floating point operations become; but this is just an educated guess and might warrant further observation. Overall, if a graph would be fitted over each similar condition for each scenario respectively, it would resemble a monotonic increase, adhering to the overall expectations.

After the initial population generation, the main loop, i.e. the optimization of the population, is to be observed and evaluated next. As figure 4.4 and table 4.4 report, The overall time required lies within the thousands of milliseconds, an order of magnitude of difference to the cost map generation before and even two orders of magnitude of difference to the initial population generation. The cause of this is not only the evaluation of feasibility of a solution candidate itself for each of the inner loop iteration, but also primarily due to the repair operation, which itself also checks for feasibility (although parallelized). Looking at the initial population generation, which rejects unfeasible solutions outright and just generates a new one, it might be faster to just use this scheme for the optimization step as well. However, this would heavily affect the overall converging behaviour negatively, i.e. it would make it unstable as solution population itself would be unstable and previously rejected candidates could occur again and again. This is why the repair operation was deemed necessary by Kern et al.[KLV20] and even Chu et al.[CB98]; time is sacrificed for solution quality.
Figure 4.4 shows a big variance of timings for each scenario and each condition, mainly caused by the add operator of the repair operation rejecting added 1-components. Interestingly, for the condition of 1 constraint and 3 beams, the Box-Phantom, not the Liver or Alderson-Phantom, reports the most variance of time required, whereas it is more "in line" with expectations for the other conditions. It can also be observed that no monotonic increase or decrease can be fitted nicely over the whole data, due to the bigger variance and spread of time. This is most likely caused by the increased reliance on randomness: Each transform operation utilizes a (biased) random factor of 0 or 1 for multiplication during the combination of two solutions and the transform operation is called upon multiple times. It is observable however, especially with table 4.4, that the number of beams, i.e. the number of 1-components in a solution, appears to have a more dominant effect than the number constraints. That being said, for "dependent constraints", like the minimum spherical distance, the number of beams directly affects the computational time required: It scales with linearly with the number of beams $(O(n))$.

Depending on the constraints, beams and number of main loop iterations, it can be minutes before solutions are computed and optimized. For real-time purposes this is obviously unusable but literature discussing treatment planning systems

and components, single-digit to lower double-digit amount of minutes are "not a bad" ballpark to be in. This is especially true for a system supposed to give initial suggestions or estimations to a domain expert for further consideration.

With regards to the quality of the system's output, the figures 4.5 to 4.10 are giving an overview of a run of each scenario for 3 beams and the constraints C_1, C_2, C_3 (see section 4.2.2). Looking at each respective cumulative DVH, it can be seen that the GTVs, CTVs and PTVs are hit with the maximum dose, unfortunately not always with a sharp fall-off as desired but still within workable margins. Each of the chosen beam angles seem, to the author's untrained eye and with respect to the OARs, sensible. For the "Head And Neck"-scenario, it can be seen that the suggested beam positions also distribute dose of up to 60Gy to OARs. Looking at the differential DVH however, it can be assumed that less than 30% of voxels of the OARs are hit non-desired does, i.e. they are partially hit by the beam due to its tendency to have a Gaussian spread, especially as the beam approaches the Bragg-peak depth.
The "Prostate"-scenario has an equally unfortunate "casualty": The lymph nodes (green graphs). The prostate bed seems unavoidable to be hit in the process as it essentially surrounds the tumor itself. Here, the careful positioning of the Bragg-peak appears to be the key and at least serves as a reminder that it is a factor to consider.
The "TG119"-scenario, "Alderson-Phantom"-scenario and the "Box-Phantom"-scenario are, from the author's untrained eye, results that could be considered for a plan without a lot of "fine-tuning" by a domain expert.
The results for the "Liver"-scenario also show a desired dose distribution rate for the target volumes. However, the "clip"-curves/objects denote clinical supporting structures that are situated within or around the tumor and are also unavoidable to hit, unless maybe a "thinner" beam is chosen (with higher proton fluence). To the author, the purpose of these structures is not exactly clear and the documentation by Craft et al [Cra+14] did not help to alleviate this. This might very well be due to the author's inexperience in the field of medical physics.
Overall, these results can very well be used to help to inform decisions made by the planners, oncologists and other related experts for a possible treatment therapy, especially with the timings in mind.

4.3.1 Limitations And Future Work

Most usability, performance-affecting and computational limitations can be found withing the optimization step itself. While TBLO has been chosen, among other reasons more thoroughly discussed in section 2.4.2 and 3, to alleviate the need for parameter tuning, as common with other metaheuristics, it is still not free from possible improvements. As already discussed before, the overall time required for TLBO lies within the thousands of milliseconds, an order of magnitude difference to

the cost map generation and even two orders of magnitude difference to the initial population generation. And both the cost map and the population generation are only executed once and not in a loop. So it seems that future optimization work should focus on this part in particular.
One of the main reasons for this is that the repair operation is called multiple times within one inner loop iteration, especially after a transformation operation occurred. So it might be worthwhile to investigate a transformation operation that does not (or slightly less frequently) "damage" solution candidates, i.e. making them not unfeasible. A more "stable" transform operation could possibly even eliminate the need for a repair operation altogether. The main caveat here of course is to also preserve the converging behaviour at the same time.

Another, more usability-focused, concern is that constraints need to be added (implemented) "by hand" and the system needs to be recompiled. This is due to two reasons. The first is that functors (pure and impure) defined in Matlab are not exportable to external DLLs, unless a specific interface/API has been developed and implemented to do so, which is outside the scope of this system. Secondly, there currently is no way to copy functors, especially impure ones, to the device from the host without added complexity, impacting the overall performance. Consequently, the user needs to be aware of what types of constraint functions the system currently supports and possibly needs to add custom constraints in code and recompile the system.
Unfortunately, other current metaheuristics, as listed in the taxonomy by Stork et al. (figure 2.11), do not sidestep or eliminate the overall basic structure of implementation as the problem (the 0/1-MKP) remains the same. Maybe with more GPU-Host interoperability efforts, like unified memory, less host-side code is required and more GPU-side computations can be made, eliminating the "memory interoperability abyss" between host and device. In this regard, the unification of GPU and CPU in one system-on-a-chip, like the Apple M1 or the APUs by AMD, may allow exactly for this.

Additionally, due to the generality of the system presented, it does not deliver results that should be used prima facie in the application domain. One of the reasons are the abstractions to the beam model made and the discretizations made in favor of performance. On a personal level, the idea to use the parallelized evaluation operation (with the constraints) together with a Virtual Reality application (VR), allowing an oncologist and/or radiologist to place beams in 3D and evaluate them in real-time is something worth to explore. It would combine the expert domain knowledge with the evaluation capabilities of the system. For the purpose of a more readily applicable evaluation, the resolution, constraint complexity and overall scope can be increased. The overall structure of computation would not change. This could allow for real-time beam evaluation in VR for a treatment or teaching scenario.

The overall precision/accuracy of the results might be alleviated by a better penalty constraint map, maybe even a custom-made map for this system in mind.

The system can also possibly be adapted for volumetric modulated arc therapy (VMAT), wherein support points on a sphere around the isocenter of a PTV are used as "visiting nodes" of a path, essentially forming a travelling salesman problem of finding the shortest hamiltonian path. The path is then traversed by a beam (or multiple beams) while it shoots. The beam angles suggested here could be used as these "visiting nodes". That being said, the more popular area of research for treatment planning and beam angle optimization approaches seem to be fully dependent on the usage/utilization of convolutional neural networks, as reported in section 1.2. Which attempt to directly compute beam angles (and even other treatment parameters) using a CT (and other meta information) as input. Unfortunately, the training data sets are limited and do not cover all possible variations and abnormal situations and even the simple task of optaining a set that does not overfit a network is not trivial. Also discussed in section 1.2, a combination of CNNs and expert systems, i.e. systems that try to SAT-solve incomplete predicates using description logics over a knowledge base, seem to be another possible approach. That being said, this system is also applicable to other areas of application, which is the main disadvantage of trained neural networks and expert systems: It is obvious that the more specialized a tool for a problem is, the better it generally solves given problem but the worse it is applicable to other, maybe similar, problems. And one of the goals stated was to maintain a certain generality of application.

Bibliography

[AMA19] S. Almufti, R. Marqas, and V. Ashqi. “Taxonomy of bio-inspired optimization algorithms”. In: *Journal Of Advanced Computer Science & Technology* 8.2 (2019), p. 23 (cit. on p. 7).

[App68] A. Appel. “Some techniques for shading machine renderings of solids”. In: *AFIPS ’68 Proceedings of the April 30–May 2, 1968, Spring Joint Computer Conference*. 1968, pp. 37–45 (cit. on pp. 1, 6, 10).

[AW87] J. Amanatides and A. Woo. “A Fast Voxel Traversal Algorithm for Ray Tracing”. In: *Proceedings of Eurographics 1987*. Eurographics Association, Aug. 1987 (cit. on pp. xi, 6, 11–13, 47, 51).

[BB11] M. Balonov and A. Bouville. “Radiation exposures due to the Chernobyl accident”. In: (2011) (cit. on p. 18).

[BBT20] T.-D. Bradley, T. Bryson, and J. Terilla. *Topology: A Categorical Approach*. MIT Press, 2020 (cit. on p. 44).

[Ber+13] D. Bertsimas, V. Cacchiani, D. Craft, and O. Nohadani. “A hybrid approach to beam angle optimization in intensity-modulated radiation therapy”. In: *Computers & Operations Research* 40.9 (2013), pp. 2187–2197 (cit. on p. 5).

[Ber+93] M. Berger, M. Inokuti, H. Andersen, et al. “Report 49”. In: *Journal of the International Commission on Radiation Units and Measurements* 2 (1993), NP–NP (cit. on p. 24).

[BJB18] L. Burigo, O. Jäkel, and M. Bangert. “matRad-An open-source treatment planning toolkit for educational purposes”. In: *MEDICAL PHYSICS* 6.1 (2018) (cit. on p. 31).

[Bor97] T. Bortfeld. “An analytical approximation of the Bragg curve for therapeutic proton beams”. In: *Medical physics* 24.12 (1997), pp. 2024–2033 (cit. on pp. 23, 24, 52).

[BS93] T. Bortfeld and W. Schlegel. “Optimization of beam orientations in radiation therapy: some theoretical considerations”. In: *Physics in Medicine & Biology* 38.2 (1993), p. 291 (cit. on p. 4).

[Cac+22] V. Cacchiani, M. Iori, A. Locatelli, and S. Martello. “Knapsack problems — An overview of recent advances. Part I: Single knapsack problems”. In: *Computers & Operations Research* 143 (2022), p. 105692 (cit. on p. 35).

[Can15] B. Cantanzaro. *Variant*. `https://github.com/bryancatanzaro/variant`. 2015 (cit. on p. 42).

[Can17] Canon Medical Systems. *Aquilion Lightning 80*. Technical Brochure. Online accessible via https://us.medical.canon/products/computed-tomography/aquilion-lightning-80/. 2017 (cit. on pp. 28, 45).

[CB98] P. C. Chu and J. E. Beasley. "A genetic algorithm for the multidimensional knapsack problem". In: *Journal of heuristics* 4.1 (1998), pp. 63–86 (cit. on pp. 7, 38, 68, 99).

[Cra+14] D. Craft, M. Bangert, T. Long, D. Papp, and J. Unkelbach. "Shared data for intensity modulated radiation therapy (IMRT) optimization research: the CORT dataset". In: *GigaScience* 3.1 (2014), pp. 2047–217X (cit. on pp. 81, 83, 100).

[DBC03] J. O. Deasy, A. I. Blanco, and V. H. Clark. "CERR: a computational environment for radiotherapy research". In: *Medical physics* 30.5 (2003), pp. 979–985 (cit. on p. 31).

[Don+19] X. Dong, Y. Lei, T. Wang, et al. "Automatic multiorgan segmentation in thorax CT images using U-net-GAN". In: *Medical physics* 46.5 (2019), pp. 2157–2168 (cit. on p. 28).

[Dye+17] M. Dyer, B. Gärtner, N. Megiddo, and E. Welzl. "Linear Programming". In: *Handbook of Discrete and Computational Geometry, 3rd Edition*. Ed. by J. E. Goodman, J. O'Rourke, and C. D. Toth. Boca Raton, Florida, United States of America: Chapman and Hall/CRC Press, 2017. Chap. 49, pp. 1291–1310 (cit. on pp. 7, 68).

[FH12] P. F. Felzenszwalb and D. P. Huttenlocher. "Distance transforms of sampled functions". In: *Theory of computing* 8.1 (2012), pp. 415–428 (cit. on p. 7).

[GJ+10] G. Guennebaud, B. Jacob, et al. *Eigen v3*. http://eigen.tuxfamily.org. 2010 (cit. on pp. 42, 55).

[GKP89] R. L. Graham, D. E. Knuth, and O. Patashnik. *Concrete Mathematics: A Foundation for Computer Science*. Boston, MA, USA: Addison-Wesley, 1989 (cit. on p. 57).

[GP21] A. Genitrini and M. Pépin. "Lexicographic unranking of combinations revisited". In: *Algorithms* 14.3 (2021), p. 97 (cit. on pp. 56, 58, 59).

[Gra20] J. Graetz. "High performance volume ray casting: A branchless generalized Joseph projector". In: *arXiv preprint arXiv:1609.00958* (2020) (cit. on pp. 7, 52).

[Gu+18] W. Gu, D. O'Connor, D. Nguyen, et al. "Integrated beam orientation and scanning-spot optimization in intensity-modulated proton therapy for brain and unilateral head and neck tumors". In: *Medical physics* 45.4 (2018), pp. 1338–1350 (cit. on p. 5).

[Gu+19] W. Gu, R. Neph, D. Ruan, et al. "Robust beam orientation optimization for intensity-modulated proton therapy". In: *Medical physics* 46.8 (2019), pp. 3356–3370 (cit. on pp. 5, 18).

[Gu+20] W. Gu, D. O'Connor, D. Ruan, et al. "Fraction-variant beam orientation optimization for intensity-modulated proton therapy". In: *Medical Physics* 47.9 (2020), pp. 3826–3834 (cit. on p. 5).

[GWP17] D. R. Grimes, D. R. Warren, and M. Partridge. “An approximate analytical solution of the Bethe equation for charged particles in the radiotherapeutic energy range”. In: *Scientific reports* 7.1 (2017), pp. 1–12 (cit. on p. 23).

[Hei+14] H. Heinrich, P. Ziegenhein, C. Kamerling, H. Froening, and U. Oelfke. “GPU-accelerated ray-tracing for real-time treatment planning”. In: *Journal of Physics: Conference Series*. Vol. 489. 1. IOP Publishing, 2014, p. 012050 (cit. on p. 6).

[HLY19] C. Huang, Y. Li, and X. Yao. “A survey of automatic parameter tuning methods for metaheuristics”. In: *IEEE transactions on evolutionary computation* 24.2 (2019), pp. 201–216 (cit. on p. 8).

[Hug+13] J. F. Hughes, A. van Dam, M. McGuire, et al. *Computer graphics: principles and practice (3rd ed.)* Boston, MA, USA: Addison-Wesley Professional, 2013, p. 1264 (cit. on pp. 1, 10).

[HW12] Y. Hung and W. Wang. “Accelerating parallel particle swarm optimization via GPU”. In: *Optimization Methods and Software* 27.1 (2012), pp. 33–51 (cit. on p. 7).

[Jak+10] W. Jakob, A. Arbree, J. T. Moon, K. Bala, and S. Marschner. “A radiative transfer framework for rendering materials with anisotropic structure”. In: *SIGGRAPH ’10: ACM SIGGRAPH 2010 papers*. ACM New York, NY, USA, 2010, pp. 1–13 (cit. on p. 17).

[JKH19] P. Jackson, T. Kron, and N. Hardcastle. “A future of automated image contouring with machine learning in radiation therapy”. In: *Journal of Medical Radiation Sciences* 66.4 (2019), pp. 223–225 (cit. on p. 29).

[JMP18] B. Jones, S. McMahon, and K. Prise. “The radiobiology of proton therapy: challenges and opportunities around relative biological effectiveness”. In: *Clinical Oncology* 30.5 (2018), pp. 285–292 (cit. on p. 6).

[Jos82] P. M. Joseph. “An Improved Algorithm for Reprojecting Rays through Pixel Images”. In: *IEEE Transactions on Medical Imaging* 1.3 (1982), pp. 192–196 (cit. on pp. 6, 52).

[Kai+19] A. Kaiser, J. G. Eley, N. E. Onyeuku, et al. “Proton therapy delivery and its clinical application in select solid tumor malignancies”. In: *JoVE (Journal of Visualized Experiments)* 144 (2019), e58372 (cit. on p. 20).

[Kaj86] J. T. Kajiya. “The rendering equation”. In: *SIGGRAPH ’86: Proceedings of the 13th annual conference on Computer graphics and interactive techniques*. Association for Computing Machinery (ACM), 1986, pp. 143–150 (cit. on pp. 9, 16).

[Kel+19] A. Keller, T. Viitanen, C. Barré-Brisebois, C. Schied, and M. McGuire. “Are we done with ray tracing?” In: *SIGGRAPH Courses*. 2019, pp. 3–1 (cit. on p. 6).

[Ken13] A. Kensler. “Correlated multi-jittered sampling”. In: *Pixar Technical Memo* 13-01 (2013), pp. 1–8 (cit. on p. 57).

[KG14] F. M. Khan and J. P. Gibbons. *The physics of radiation therapy*. Lippincott Williams & Wilkins, 2014 (cit. on pp. 14–16, 18, 19, 23, 27–29, 31, 32, 81, 82).

[Kim+20] J. Kim, Y.-K. Park, G. Sharp, P. Busse, and B. Winey. “Beam angle optimization using angular dependency of range variation assessed via water equivalent path length (WEPL) calculation for head and neck proton therapy”. In: *Physica Medica* 69 (2020), pp. 19–27 (cit. on p. 5).

[KLV20] Z. Kern, Y. Lu, and F. Vasko. “An OR practitioner’s solution approach to the multidimensional knapsack problem”. In: *International Journal of Industrial Engineering Computations* 11.1 (2020), pp. 73–82 (cit. on pp. xi, 3, 8, 37, 38, 53, 54, 59, 68, 73, 87, 99).

[KM19] B. Krayer and S. Müller. “Generating signed distance fields on the GPU with ray maps”. In: *The Visual Computer* 35.6 (2019), pp. 961–971 (cit. on p. 7).

[KWB15] J. B. Kiely, B. White, and S. Both. “A beam angle optimization technique for proton pencil beam scanning treatment planning of lower pelvis targets”. In: *World Congress on Medical Physics and Biomedical Engineering, June 7-12, 2015, Toronto, Canada*. Springer. 2015, pp. 479–482 (cit. on p. 4).

[Laa+18] S. Laabadi, M. Naimi, H. El Amri, B. Achchab, et al. “The 0/1 multidimensional knapsack problem and its variants: a survey of practical models and heuristic approaches”. In: *American Journal of Operations Research* 8.05 (2018), pp. 395–439 (cit. on pp. 35, 36).

[Lan+93] T. Landberg, J. Chavaudra, J. Dobbs, et al. “Report 50”. In: *Journal of the International Commission on Radiation Units and Measurements* 1 (1993), NP–NP (cit. on pp. 29, 79).

[Lan+99] T. Landberg, J. Chavaudra, J. Dobbs, et al. “Report 62”. In: *Journal of the International Commission on Radiation Units and Measurements* 1 (1999), NP–NP (cit. on pp. 29, 79).

[LCM14] G. Lim, W. Cao, and R. Mohan. “Recent advances in intensity modulated proton therapy treatment planning optimization”. In: *Proceedings of the 15th Asia Pacific Industrial Engineering and Management Systems Conference*. 2014, pp. 1520–1525 (cit. on p. 4).

[Li+20] L. Li, X. Zhao, W. Lu, and S. Tan. “Deep learning for variational multimodality tumor segmentation in PET/CT”. In: *Neurocomputing* 392 (2020), pp. 277–295 (cit. on p. 28).

[Lüh+18] A. Lühr, C. von Neubeck, M. Krause, and E. G. Troost. “Relative biological effectiveness in proton beam therapy–Current knowledge and future challenges”. In: *Clinical and translational radiation oncology* 9 (2018), p. 35 (cit. on p. 6).

[Lus+18] T. Lustberg, J. van Soest, M. Gooding, et al. “Clinical evaluation of atlas and deep learning based automatic contouring for lung cancer”. In: *Radiotherapy and Oncology* 126.2 (2018), pp. 312–317 (cit. on p. 28).

[Maj+18] A. Majercik, C. Crassin, P. Shirley, and M. McGuire. “A ray-box intersection algorithm and efficient dynamic voxel rendering”. In: *Journal of Computer Graphics Techniques* 7.3 (2018), pp. 66–81 (cit. on p. 51).

[MAK03] M. Mokbel, W. Aref, and I. Kamel. “Analysis of Multi-Dimensional Space-Filling Curves”. In: *GeoInformatica* 7 (Sept. 2003), pp. 179–209 (cit. on p. 48).

[Mar+19] D. Martinez, M. Rahmani, C. Burbadge, and C. Hoehr. “A practical solution of the Bethe equation in the energy range applicable to radiotherapy and radionuclide production”. In: *Scientific reports* 9.1 (2019), pp. 1–9 (cit. on pp. 22, 23).

[MAT21] MATLAB. *version 9.11.0 (R2021b)*. Natick, Massachusetts: The MathWorks Inc., 2021 (cit. on pp. 3, 43).

[Mat96] G. B. Mathews. “On the Partition of Numbers”. In: *Proceedings of the London Mathematical Society* s1-28.1 (1896), pp. 486–490 (cit. on p. 35).

[MD19] A. K. Mandal and S. Dehuri. “A Survey on Ant Colony Optimization for Solving Some of the Selected NP-Hard Problem”. In: *International Conference on Biologically Inspired Techniques in Many-Criteria Decision Making*. Springer. 2019, pp. 85–100 (cit. on p. 8).

[Mei+21] D. Meister, S. Ogaki, C. Benthin, et al. “A Survey on Bounding Volume Hierarchies for Ray Tracing”. In: *Computer Graphics Forum* 40.2 (2021), pp. 683–712 (cit. on p. 6).

[Nat17] National Electrical Manufacturers Association (NEMA). *DICOM strategic document - Revision 2017-03-08*. `https://dicom.nema.org/dicom/geninfo/Strategy.pdf`. 2017 (cit. on p. 43).

[Nic+08] J. Nickolls, I. Buck, M. Garland, and K. Skadron. “Scalable parallel programming with CUDA”. In: *Queue* 6.2 (2008), pp. 40–53 (cit. on p. 3).

[nVi22b] nVidia. *Github Repository: CUDA C++ Samples*. `https://github.com/NVIDIA/cuda-samples`. 2022 (cit. on p. 65).

[NZ15] W. D. Newhauser and R. Zhang. “The physics of proton therapy”. In: *Physics in Medicine & Biology* 60.8 (2015), R155 (cit. on pp. 2, 18, 20–22, 24–26, 79).

[Pag17] H. Paganetti. *Proton beam therapy*. IOP Publishing Bristol, 2017 (cit. on pp. 4, 6, 22).

[PB81] G. Pâolya and E. F. Beckenbach. *Applied Combinatorial Mathematics*. USA: Krieger Publishing Co., Inc., 1981 (cit. on pp. 57, 58).

[Pet+18] H. E. S. Pettersen, M. Chaar, I. Meric, et al. “Accuracy of parameterized proton range models; a comparison”. In: *Radiation Physics and Chemistry* 144 (2018), pp. 295–297 (cit. on pp. 23, 24).

[Pin+12] C. Pinter, A. Lasso, A. Wang, D. Jaffray, and G. Fichtinger. “SlicerRT: radiation therapy research toolkit for 3D Slicer”. In: *Medical physics* 39.10 (2012), pp. 6332–6338 (cit. on p. 31).

[PJH16] M. Pharr, W. Jakob, and G. Humphreys. *Physically based rendering: From theory to implementation*. Morgan Kaufmann, 2016 (cit. on pp. 10, 11, 16, 17).

[Pri+17] F. Prior, K. Smith, A. Sharma, et al. “The public cancer radiology imaging collections of The Cancer Imaging Archive”. In: *Scientific data* 4.1 (2017), pp. 1–7 (cit. on pp. 81, 83).

[Pur+05] T. J. Purcell, I. Buck, W. R. Mark, and P. Hanrahan. “Ray tracing on programmable graphics hardware”. In: *ACM SIGGRAPH 2005 Courses*. 2005, 268–es (cit. on p. 6).

[Ras20] F. S. Rasouli. “Analytical range evaluation for therapeutic protons in stack of materials”. In: *Radiation Physics and Engineering* 1.2 (2020), pp. 35–39 (cit. on p. 52).

[Ric+19] H. Rico-Garcia, J.-L. Sanchez-Romero, A. Jimeno-Morenilla, et al. “Comparison of high performance parallel implementations of TLBO and JAYA optimization methods on manycore GPU”. In: *IEEE Access* 7 (2019), pp. 133822–133831 (cit. on pp. 3, 8, 53).

[RSV11] R. V. Rao, V. J. Savsani, and D. Vakharia. “Teaching–learning-based optimization: A novel method for constrained mechanical design optimization problems”. In: *Computer-Aided Design* 43.3 (2011), pp. 303–315 (cit. on pp. 8, 37, 68).

[Sán+14] D. Sánchez-Parcerisa, M. Kondrla, A. Shaindlin, and A. Carabe. “FoCa: a modular treatment planning system for proton radiotherapy with research and educational purposes”. In: *Physics in Medicine & Biology* 59.23 (2014), p. 7341 (cit. on p. 31).

[Sar+06] D. Sarrut, V. Boldea, S. Miguet, and C. Ginestet. “Simulation of four-dimensional CT images from deformable registration between inhale and exhale breath-hold CT scans”. In: *Medical physics* 33.3 (2006), pp. 605–617 (cit. on p. 28).

[SEB20] J. Stork, A. E. Eiben, and T. Bartz-Beielstein. “A new taxonomy of global optimization algorithms”. In: *Natural Computing* (2020), pp. 1–24 (cit. on pp. xi, 34, 37).

[SG13] K. Sörensen and F. Glover. “Metaheuristics”. In: *Encyclopedia of operations research and management science* 62 (2013), pp. 960–970 (cit. on pp. 36, 37).

[Sho21] B. Shoshany. “A C++ 17 Thread Pool for High-Performance Scientific Computing”. In: *arXiv preprint arXiv:2105.00613* (2021) (cit. on p. 42).

[Sid85] R. L. Siddon. “Fast calculation of the exact radiological path for a three-dimensional CT array”. In: *Medical physics* 12.2 (1985), pp. 252–255 (cit. on p. 6).

[SS15] G. Schrack and L. Stocco. “Generation of Spatial Orders and Space-Filling Curves”. In: *IEEE Transactions on Image Processing* 24.6 (2015), pp. 1791–1800 (cit. on p. 49).

[SVS08] L. Struelens, F. Vanhavere, and K. Smans. “Experimental validation of Monte Carlo calculations with a voxelized Rando–Alderson phantom: a study on influence parameters”. In: *Physics in Medicine & Biology* 53.20 (2008), p. 5831 (cit. on p. 81).

[Taa+20] V. T. Taasti, L. Hong, J. S. Shim, J. O. Deasy, and M. Zarepisheh. “Automating proton treatment planning with beam angle selection using Bayesian optimization”. In: *Medical Physics* (2020) (cit. on p. 5).

[Tan21] T. “Tanki” Zhang. “Handling Translucency with Real-Time Ray Tracing”. In: *Ray Tracing Gems II: Next Generation Real-Time Rendering with DXR, Vulkan, and OptiX*. Ed. by A. Marrs, P. Shirley, and I. Wald. Berkeley, CA: Apress, 2021, pp. 127–138 (cit. on p. 7).

[Tho06] S. J. Thomas. “Margins for treatment planning of proton therapy”. In: *Physics in Medicine & Biology* 51.6 (2006), p. 1491 (cit. on p. 30).

[Tia+18] X. Tian, K. Liu, Y. Hou, J. Cheng, and J. Zhang. “The evolution of proton beam therapy: Current and future status”. In: *Molecular and clinical oncology* 8.1 (2018), pp. 15–21 (cit. on p. 31).

[UM10] W. Ulmer and E. Matsinos. “Theoretical methods for the calculation of Bragg curves and 3D distributions of proton beams”. In: *The European Physical Journal Special Topics* 190.1 (2010), pp. 1–81 (cit. on pp. 23, 80).

[UM12] W. Ulmer and E. Matsinos. “A calculation method of nuclear cross-sections of proton beams by the collective model and the extended nuclear-shell theory with applications to radiotherapy and technical problems”. In: *Journal of Nuclear and Particle Physics* 2.3 (2012), p. 42 (cit. on p. 53).

[WC08] I. B. Weinstein and K. Case. “The History of Cancer Research: Introducing an AACR Centennial Series”. In: *Cancer Research* 68.17 (2008), pp. 6861–6862 (cit. on p. 2).

[Wie+17] H.-P. Wieser, E. Cisternas, N. Wahl, et al. “Development of the open-source dose calculation and optimization toolkit matRad”. In: *Medical physics* 44.6 (2017), pp. 2556–2568 (cit. on pp. 3, 31, 43, 85).

[Woh+17] P. Wohlfahrt, C. Möhler, V. Hietschold, et al. “Clinical implementation of dual-energy CT for proton treatment planning on pseudo-monoenergetic CT scans”. In: *International Journal of Radiation Oncology* Biology* Physics* 97.2 (2017), pp. 427–434 (cit. on pp. 28, 29).

[Xia+12] K. Xiao, D. Z. Chen, X. S. Hu, and B. Zhou. “Efficient implementation of the 3D-DDA ray traversal algorithm on GPU and its application in radiation dose calculation”. In: *Medical Physics* 39.12 (2012), pp. 7619–7625 (cit. on pp. 6, 13, 47, 50, 84, 85).

[Zie15] A. Zietman. “The Practicing Clinician’s Perspective On Proton Beam Therapy”. In: *Principles and Practice of Proton Beam Therapy*. Ed. by I. J. Das, H. Paganetti, American Association of Physicists in Medicine, et al. Medical Physics Publishing, 2015. Chap. 2 (cit. on p. 2).

[ZJ96] S. Zhang and J.-M. Jin. *Computation of special functions*. Wiley, 1996 (cit. on p. 52).

[ZZB10] J. F. Ziegler, M. D. Ziegler, and J. P. Biersack. “SRIM–The stopping and range of ions in matter (2010)”. In: *Nuclear Instruments and Methods in Physics Research Section B: Beam Interactions with Materials and Atoms* 268.11-12 (2010), pp. 1818–1823 (cit. on p. 24).

Websites

[Epi20a] Epic Games. *Unreal Engine 4 Documentation: Distance Field Ambient Occlusion*. 2020. URL: https://docs.unrealengine.com/en-US/BuildingWorlds/LightingAndShadows/DistanceFieldAmbientOcclusion/index.html (visited on Mar. 22, 2022) (cit. on p. 7).

[Epi20b] Epic Games. *Unreal Engine 4 Documentation: Distance Field Soft Shadows*. 2020. URL: https://docs.unrealengine.com/en-US/BuildingWorlds/LightingAndShadows/RayTracedDistanceFieldShadowing/index.html (visited on Mar. 22, 2022) (cit. on p. 7).

[Glo18] Global Change Data Lab. *Our World In Data*. 2018. URL: https://ourworldindata.org/cancer#cancer-over-the-long-run (visited on Mar. 3, 2022) (cit. on p. 2).

[ITN13] ITN - Imaging Technology News. *An Introduction to Current Radiation Therapy Treatment Planning Systems*. 2013. URL: https://www.itnonline.com/article/introduction-current-radiation-therapy-treatment-planning-systems (visited on Feb. 7, 2022) (cit. on p. 31).

[NAS14] NASA HEASARC - NASA High Energy Astrophysics Science Archive Research Center. *What are the Energy Range Definitions for the Various Types of Electromagnetic Radiation?* 2014. URL: https://heasarc.gsfc.nasa.gov/docs/heasarc/headates/spectrum.html (visited on Feb. 13, 2022) (cit. on p. 18).

[NIS20] NIST - National Institute of Standards and Technology. *Electron volt*. 2020. URL: https://physics.nist.gov/cgi-bin/cuu/Value?evj%7Csearch_for=electron+volt (visited on Feb. 22, 2022) (cit. on p. 18).

[nVi22a] nVidia. *CUDA C++ Programming Guide*. 2022. URL: https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html (visited on Mar. 22, 2022) (cit. on pp. 42, 47, 48).

[Wal22] J. Walton. *GPU Benchmarks and Hierarchy 2022: Graphics Cards Ranked*. 2022. URL: https://www.tomshardware.com/reviews/gpu-hierarchy,4388.html (visited on Mar. 22, 2022) (cit. on p. 45).

[WHO20] WHO - World Health Organization. *Fact Sheet: Cancer*. 2020. URL: https://www.who.int/news-room/fact-sheets/detail/cancer (visited on Mar. 3, 2022) (cit. on p. 1).