Diese Arbeit wurde vorgelegt am Lehr- und Forschungsgebiet
Theorie der hybriden Systeme

Optimierung von Zielpunktstrategien für Heliostaten in solarthermischen Kraftwerken

Optimization of Aiming Strategies for Heliostats in Solar-Thermal Power Plants

Bachelorarbeit
Informatik

Januar 2018

Vorgelegt von Presented by	Georg Wicke Nizzaallee 9, 52072 Aachen Matrikelnummer: 333047 georg.wicke@rwth-aachen.de
Erstprüferin First examiner	Prof. Dr. Erika Ábrahám Lehr- und Forschungsgebiet Theorie der hybriden Systeme RWTH Aachen University
Zweitprüferin Second examiner	Jun.-Prof. Dr. Christina Büsing Lehrstuhl II für Mathematik RWTH Aachen University
Korefferent Co-supervisor	Dr. rer. nat. Pascal Richter Lehr- und Forschungsgebiet Kontinuierliche Optimierung RWTH Aachen University

Eigenständigkeitserklärung

Hiermit versichere ich, dass ich diese Bachelorarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Die Stellen meiner Arbeit, die dem Wortlaut oder dem Sinn nach anderen Werken entnommen sind, habe ich in jedem Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht. Dasselbe gilt sinngemäß für Tabellen und Abbildungen. Diese Arbeit hat in dieser oder einer ähnlichen Form noch nicht im Rahmen einer anderen Prüfung vorgelegen.

Aachen, im Januar 2018

Georg Wicke

Contents

List of Figures

List of Tables

1. Introduction

Image source: https://commons.wikimedia.org/wiki/File:PS10_desde_PS20.jpg

Figure 1: The PS10 plant in Spain, which is used in the example calculations.

Solar tower power plants are facilities that produce electricity from solar radiation.[2] The radiation is collected with mirrors, called heliostats, and concentrated onto a receiver mounted on a tower[3]. On the receiver, the incoming rays are heating a medium (e.g. water, molten salt, air), which is used to transport the energy to systems that make use of it and, finally, generate electricity. Increasing the efficiency of solar power plants may lead to higher adoption of this technology, providing clean energy at a low price in a scalable way.

The receiver system is a complex piece of hardware that makes use of the incoming radiation in an efficient way. Excess heat or thermal stress can damage the receiver - the manufacturer has to make sure that this doesn't happen. For this purpose, it is necessary to aim the heliostats properly. If they are all aiming onto the centre, the receiver may not withstand the literal power of a thousand suns and may get damaged. This thesis presents multiple improvements for aiming strategies in solar-thermal power plants, with the goal of meeting a light distribution that is energetically optimal while not endangering people or property.

1.1. Preliminaries

In solar tower power plants, energy is collected by heliostats and directed to a receiver mounted on a central tower (see Figure 2). This device heats a medium to transfer the heat to a power generator (e.g. a turbine) or a heat storage, where the energy can be stored for later use (e.g. at night).

The heliostats are equipped with precise motors to aim their light onto a defined spot on the receiver. As the sun direction changes over the course of a day, the heliostats have to track that movement. As the motors are, although precise, not perfect, the casted image[5] may be off-centre. This is known as the tracking error.

[2] Another frequently used term is Concentrating Solar Power, or CSP for short, which also includes parabolic mirrors and similar systems.

[3] Sometimes multiple towers are used with a single central steam turbine.

[4] Figure is joint work with Hannah Arndt.

[5] The casted image is sometimes called the flux profile of the heliostat in literature.

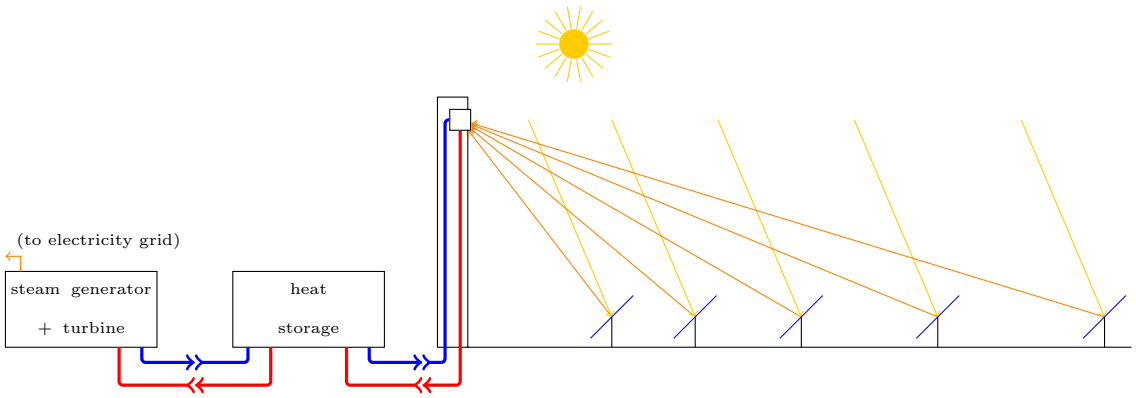


Figure 2: Rough illustration of a solar tower plant.[4]

Other factors, such as optical and sun shape errors, are adding up to an image that can be approximated by a normal distribution. One algorithm that uses this approximation to generate the heliostat image is HFLCAL. We will present an adapted version of it in Section 2.3.

The images casted by the heliostats are summing up to a total flux distribution on the receiver's surface. A metric often used when discussing efficiency of this flux distribution is the spillage loss, which is the (relative) difference of the received power and the power that would be received if all heliostats were targeting the centre of the receiver. The spillage losses increase when more heliostats are targeting points near the edge of the receiver.

Note that the power that reaches the receiver is not the same as the power that can be made use of. An energetically ideal power distribution on the receiver can be determined with thermal simulations. The first optimization model, presented in Section 3.2, assumes that such an ideal distribution is available and optimizes towards reaching it.

The flux distribution may also be relevant for safety reasons: if too many heliostats are targeting a single point, the material may melt or catch fire. And if the temperature gradient on the receiver is high, thermal stress is created, which lowers the life span of the receiver. These constraints can be encoded in a maximum distribution, which is dependant on the total power reaching the receiver. In Section 3.3, an optimization model based on the assumption that such distributions are available is presented.

Aiming strategies that can be computed in real-time are quite an advantage, as partial cloud coverage can quickly and drastically change the amount of radiation that individual heliostats are sending. Pre-computed strategies can only work on cloud-free days.

For the data, the properties of the PS10 plant in Andalusia, Spain are used for the field layout and tower properties. For both the ideal and maximum distributions, data was taken from a DLR publication [6, p. 10] instead, as such data was not available for this plant.

An important limitation for the performance of the presented algorithms is the number of heliostats involved. The PS10 has 624 of them, but bigger plants can

have multiple thousand mirrors. At the time of writing, the biggest plant in operation – the Ivanpah plant in California – has 170 000 heliostats distributed over three fields.

Note that the used data does not describe an existing plant: It combines the receiver model from one with the field layout and tower properties of another. However, it demonstrates the feasibility, performance and potential outcomes of the described algorithms.

1.2. Related Work

A review from 2014 [10] gives a good overview over existing techniques for aimpoint strategy optimization. It does also mention a lack of publications for commercially used strategies, which doesn't seem to have changed since then. It becomes apparent that most approaches only cover the security, but not the optimization aspects of the problem. This applies to most of the works presented in this subsection.

A. Grobler [9] presents two ways of determining aiming strategies and combines them. The goal here is to minimize stress, which means minimizing temperature gradients. This leads to very even distributions. The techniques used are a genetic algorithm and a tabu search. The work focuses on rather small, experimental fields, such as the Helio40 and the Helio100 systems[6] – the 624-heliostat PS10 plant is already considered big by its standards. However, it does also provide significant contributions to the Gaussian flux approximation methods, which can be used to generate receiver images.

The company Abengoa Solar, which operates several such plants, targets a flux distribution that is flat on the vertical central 75 % of the cylindrical receiver and drops to 20 % at the edges, which they claim to have determined to be a good distribution via a spreadsheet [11, p. 116]. Five vertical aim levels are defined to which the heliostats are randomly assigned according to an unspecified weight function, then they are shifted in an again unspecified way so that they meet the objective. Although called a "challenge", a detailed description of the algorithm is missing.

For the GemaSolar plant, two basic strategies are proposed [2, pp. 67-71] to lower peak flux and temperature gradients compared to targeting a single point. Under the first proposal, closer (resp. more distant) heliostats target the lower (resp. higher) end of the receiver[7]. The second proposal takes into account that closer heliostats have a much smaller image than those that are far away from the tower. Therefore, close heliostats should target the edges, while more distant heliostats target the centre. This limits the spillage losses while meeting the primary objectives. These strategies seem to be rather primitive and leave a lot of room for optimization.

Until 1998 the retargeting of the heliostats required a lot of manual control: heliostats that overheated certain portions of the receiver could automatically be removed from operation, but not be re-activated. For the PSA CESA-1 plant a closed-loop automated

[6] With 20 and 100 heliostats respectively.

[7] The Gemasolar plant has a cylindrical receiver. This means that it's sufficient to only move the aimpoints along the vertical axis to illuminate the entire receiver.

control system was developed [7] to reduce the manpower required to operate such a plant. It uses thermocouples integrated into the receiver for measurements and five aimpoints for targeting. Heliostats can be moved from one aimpoint to another, and the aimpoints themselves can be moved within their zone. Whenever certain thresholds are exceeded, the heliostats or aimpoints are adjusted to homogenize the temperatures on the receiver surface. The work mainly focuses on reducing workload for the operator and is not really concerned with efficiency.

For the Jülich Solar Tower, operated by the DLR[8], an extensive optimization system has been built, which also includes aiming strategy optimizations [3]. The project is generally aimed at doing whole-system optimization, and thus uses the system's energy output as its quality metric. Technologically, the mapping of heliostats to aimpoints is formulated as a directed graph, and a genetic ant colony algorithm is used to minimize the length of a path through this graph, which is constructed such that this length corresponds to the system's output. It is inspired by real-world ant colonies, where ants are guided by other ant's pheromones when finding a path, and use their other senses to locate the resources when they're close. An ant produces pheromones proportional to the quality of its path. More ants following the same way lead to a higher chance of an individual ant to follow that way. For each path, every visited node corresponds to a heliostat targeting a specific aimpoint, and the path length corresponds to the quality of the solution. This approach seems promising, but is rather slow, so that it can't be used as a real-time approach. At the same time, it does not always converge to the global optimum, so that it's not as suitable for a reference solution either.

For the french THEMIS solar tower, two local search strategies have been evaluated [14]: hill-climbing and tabu search, where a modification of the latter proved most efficient. The goal of this optimization was, again, to minimize thermal stress on the receiver, while not getting too inefficient. However, the efficiency is only measured by the spillage losses, which are restricted by a hard limit. As such, the system is not making any cheap trade-offs where e.g. much less stress could lead to an only slightly reduced efficiency.

A new approach [4] works with a more conventional genetic algorithm, where the mapping of heliostats to aimpoints is encoded in a binary matrix. The fitness is given by the standard deviation of the flux density distribution, crossover is done by selecting matrix columns, and mutation is done by randomizing individual columns. The complications introduced by encoding aimpoints as rows in the matrix does not seem to have any advantage over simply using a single number per aimpoint. Apart from this, the general approach of using a genetic algorithm to tackle this problem is adapted in this work.

A recent paper [1] is the basis for one part of the thesis: the authors describe a method to optimize the aim point strategy with Mixed-Integer Programming (MIP). Their goal is to maximize the received energy, with constraints that limit stress and set maximum energies. This work extends on the general layout of the MIP formulation, giving more realistic objectives and constraints.

[8]Deutsches Zentrum für Luft- und Raumfahrt, German Aerospace Centre

1.3. Structure of this Work

This thesis is structured as follows: At first, the general framework is explained in Section 2. After that, two optimization models are proposed in Section 3. For these models, multiple solving approaches are presented in Section 4, which are then evaluated in Section 5. Finally, in Section 6, the results are summarized and some future work is given.

There are multiple areas of research that this work depends on, which aren't covered here. These areas are:

- Simulation and/or measurement of heliostat images.
  This thesis assumes that data about the heliostat images is available. For all simulations and calculations, an adapted version of HFLCAL is used (see Section 2.3). As all profiles can be precomputed once, the speed of the optimizations presented here is independent from the speed of the profile calculation (e.g. expensive raytracing).

- Determination of ideal and maximum flux densities on the receiver (the "model"). It is assumed that a proper thermal simulation is available that returns those ideal profiles. These computations depend on the available radiation and therefore have to be redone for every new scenario (e.g. different time of day or year, different cloud coverage, ...).[9]

Whether the approaches were successful or not can't really be said, as there are no proper requirements that can be met or missed. Neither are realistic parameters available, nor is there a hard time limit for the solutions (although it reportedly exists, and it is somewhere in the area of seconds to minutes) or a limit on accuracy for the approximations. What can however be said is that a genetic algorithm without further adaptions does not converge in an acceptable time frame to anything near the global optimum, and that the MIP does not scale well but can be approximated, yielding scalable results.

[9] There is the possibility to precompute these receiver models for multiple fixed numbers of total radiation in the field. This would introduce slight rounding errors in most cases, as there is probably no model available for the exact incidence at a given moment, but would allow to reuse those computations for multiple sun positions or cloud coverage scenarios with similar total radiation.

2. Optical Model

2.1. General Setup

The general problem consists of the assignment of a set of heliostats H, to a set of aimpoints A on the receiver. We measure the power that reaches the receiver surface at a set of measure points M. Essentially, we're searching for a function $H \to A$ that violates no safety constraints, resembles a given flux density distribution as close as possible, and has minimum spillage losses.

In reality, there are infinitely many possible aimpoints, as the receiver surface is continuous. However, the heliostats can't be aligned beyond a certain exactness. Therefore, we can discretize the surface of the receiver. One of the aimpoints is somewhere off, functioning as a standby location to enable deactivating certain heliostats. Any heliostat pointing on it won't be shining any light on the receiver. The other points will be aligned in a rectangular grid.

The same holds true for the set of measure points: as the light distribution curve of a heliostat shouldn't have jumps, we can afford to discretize the receiver surface and only measure the light distribution in certain points. Usually, it makes sense to measure at least on all aimpoints, as there will be local maxima there. The measure points will, as well, be aligned in a rectangular grid.

We first formalize the images as a function $Q \in H \times A \times M \to \mathbb{R}_+$. This function returns the power that is being received in measure point m when heliostat h points to aimpoint a_h.

We then define the local flux density on a point m on the surface of the receiver as the sum of the flux densities contributed by all heliostats, and the total received power as the sum of all local flux densities.

$$Q^m_{\text{local}} = \sum_{h \in H} Q(h, a_h, m) \quad \forall m \in M \tag{1}$$

$$Q_{\text{total}} = \sum_{m \in M} Q^m_{\text{local}} \tag{2}$$

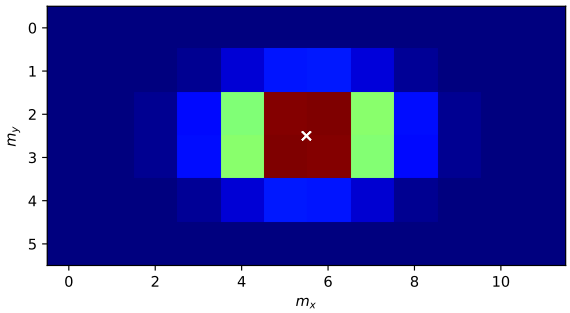


Figure 3: An example image, i. e. an evaluation of $Q(h, a, m)$ for fixed a and h, with 12×6 measure points. The white mark in the centre is the aim point.

Note that the last definition has to incorporate the represented area of each measure point if they are not evenly distributed over the receiver surface. However, as we're evenly distributing the measure points in this work, we can skip this.

We also define spillage losses as

$$s = 1 - \frac{Q_{\text{total}}}{Q_{\text{total}}^{\text{max}}} \tag{3}$$

where $Q_{\text{total}}^{\text{max}}$ is the theoretical upper bound for the total power.

This upper bound is obtained by evaluating Q_{total} for the scenario that all heliostats are targeting the upper right or upper left corner of the receiver (depending on the sun's position), as this configuration yields maximal power due to slightly different heliostat alignment (as we will see in Section 2.3 later). To make sure that the radiation doesn't miss the receiver, the receiver is extended by its own size in all directions while retaining the same measure point density.

In the following, we will first present the notation that will be used in the rest of the chapter, then we will introduce our light model.

2.2. Notation

We define $\text{lerp}(a, b, x)$ to be the result of a linear interpolation from a to b by an amount x, i.e.

$$\text{lerp}(a, b, x) = a \cdot (1 - x) + b \cdot x \tag{4}$$

We also define its inversion:

$$\text{lerp}^{-1}(a, b, x) = \frac{x - a}{b - a} \tag{5}$$

We can use both $\mathbb{R}$ and $\mathbb{R}^n$ as domains for this operation. In the latter case, all operations apply pointwise.

When using (euclidean) vectors, $\overrightarrow{a} \cdot \overrightarrow{b}$ denotes the dot product. The normalization function $\text{normalize}(\overrightarrow{a})$ is defined as

$$\text{normalize}(\overrightarrow{a}) = \frac{\overrightarrow{a}}{|\overrightarrow{a}|} \tag{6}$$

2.3. Image Generation Algorithm

The function $Q \in H \times A \times M \to \mathbb{R}_+$ that was used previously (see Equation 1) creates the images: Assuming that heliostat $h \in H$ points to the aimpoint $a \in A$, what is the flux density in the point $m \in M$? We shorten the notation to $Q(m)$ here by assuming a fixed h and a.

There are currently three methods that can be used to generate these images:

1. Raytracing.
   This method is well-known in computer graphics to produce physically accurate results on the one hand and to be computationally expensive on the other. For this reason, several approximations have been considered in existing literature and this technique has not been used here.

2. Using cone optics. [8]
   This quite new technique generates almost exact results (it barely relies on approximations) by not tracing individual rays, but calculating the light of an entire facet that shines onto a given point on the receiver at once[10]. With the help of precomputation, this can be done in constant time for a fixed radially symmetrical sun shape. Due to the increased implementation effort – the algorithm works best when implemented on a GPU –, this method was not used here.

3. HFLCAL and similar methods. [15]
   These are formulas that are approximating the flux density purely analytically. Only a few terms have to be evaluated, which makes them both simple to implement and fast to compute. For these reasons, they were applied here. A further source [5] claims HFLCAL to be of superior quality when compared to other methods, which, together with its widespread application in most of the papers discussed in Section 1.2, has lead to the decision to use this technique.

It has been widely discussed, both in theory and through experimental verification [9], that most errors affecting the heliostat image are statistically independent from each other as well as not too strongly correlated between the heliostats. They add up to a Circular Gaussian Distribution profile due to the central limit theorem. This fact is used by the HFLCAL algorithm: It models an image as a Circular Gaussian Distribution with its centre in m. The width of this distribution is dependent on a range of errors created by the sun shape, tracking, optical errors etc. All these errors are summarized into a single value σ_{total}. Additionally, the width of the distribution scales linearly with the distance of the heliostat to the tower d_h. To model the increase in width at high incidence angles, the width is additionally modified by a $1/\sqrt{cos\phi}$, where ϕ is the incidence angle. This resizes the distribution such that a given quantile covers an appropriately larger area. The complete formula is as follows:

$$\begin{aligned} Q(x,y) &= \frac{P}{2\pi\sigma_{\text{effective}}} \cdot \exp\left(-\frac{x^2+y^2}{2\sigma_{\text{effective}}}\right) \\ \text{with} \quad \sigma_{\text{effective}} &= \frac{d_h \cdot \sigma_{\text{total}}}{\sqrt{\cos\phi}} \end{aligned} \tag{7}$$

where

[10] To be exact, only the light of a triangle belonging to a certain triangular decomposition of the triangle is computed at once. However, the number of triangles generated by such a decomposition is in $O(k)$, where k is the number of corners of the facet polygon. The algorithm assumes flat facets with a polygonal edge. The facets may have a focusing/parabolic effect, but this effect is only considered in the distortion of the image and not in the geometry of the facets themselves.

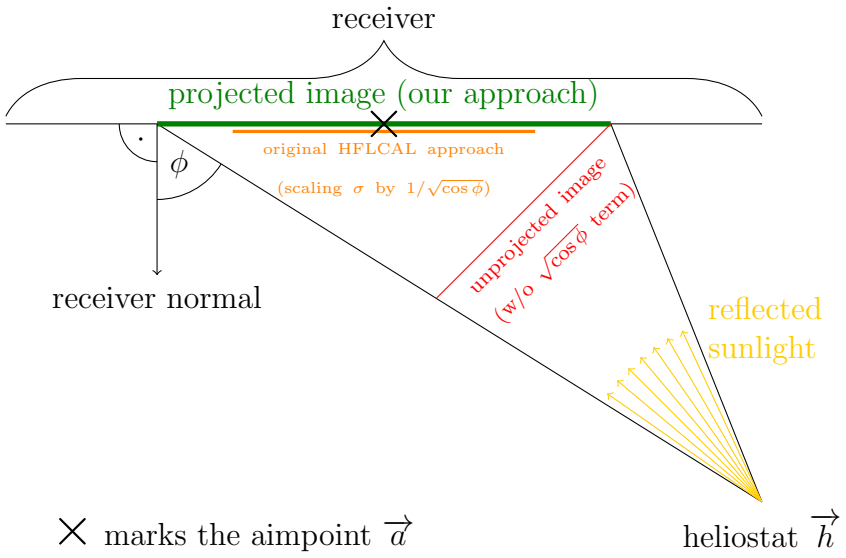


Figure 4: The adapted HFLCAL method. The coloured/labelled lines represent the width of a fixed quantile of the CGD in one dimension.

- P is the beam power, i.e. the total amount of power that is reflected by the heliostat,
- (x, y) is the position of the measure point relative to the aim point (i.e. $\overrightarrow{m} - \overrightarrow{a}$).
- $Q(x, y)$ is the flux density in that point radiated by the current heliostat.

In the HFLCAL model, the incidence angle effect is also considered to be a statistical error, which simply scales the radius of the circular distribution evenly (by scaling σ). This creates slightly larger circles (for the quantile borders) at high incidence angles. But the real outcome of projecting a circle onto a plane under an angle is not a scaled circle, but an ellipsis. The fact that this error is not uncorrelated among heliostats and therefore may add up has been noted in the past [12, 13]. This makes it necessary to be more exact about that effect in this work.

Contrary to available literature, we simply project the undistorted HFLCAL image onto the receiver instead of approximating the effect of incidence angles by using two σ values for both directions or similar complications.

This projection is done by a slight change to the formula, where the approximation is removed and replaced by a more accurate term:

$$Q(\overrightarrow{m}) = \cos\phi \cdot \frac{P}{2\pi\sigma_{\text{effective}}} \cdot \exp\left(-\frac{\text{dist}(\overrightarrow{m}, \overrightarrow{h} \rightarrow \overrightarrow{a})}{2\sigma_{\text{effective}}}\right) \tag{8}$$
$$\text{with} \quad \sigma_{\text{effective}} = d_h \cdot \sigma_{\text{total}}$$

where

- $\overrightarrow{m}$ is the measure point m as a position vector

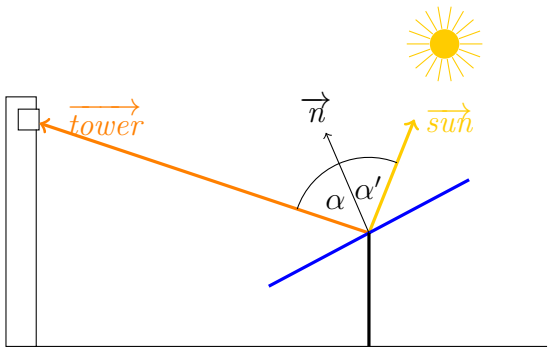


Figure 5: How the heliostat is aligned ($\alpha = \alpha'$). The vectors $\overrightarrow{tower}$, $\vec{n}$ and $\overrightarrow{sun}$ are, unlike displayed here, unit vectors in the calculations.

- $\text{dist}(\vec{m}, \vec{h} \rightarrow \vec{a})$ is the (smallest) distance between $\vec{m}$ and the ray from the heliostat position to its aimpoint.
- $Q(\vec{m})$ is the flux density in m radiated by this heliostat.

See also Figure 4 for an illustration of the parameters.

Note the term $\cos\phi$ at the beginning, which is crucial for energy conservation as the light is now distributed over a larger area. This gets clear when looking at Figure 4: the red line is scaled to the size of the green one - if the integral over the function (i.e. the total radiation of the heliostat) should keep constant, the function has to be multiplied by a stretching factor, which is the ratio between the length of the red and green lines. This ratio is $\cos\phi$.

To calculate the beam power, we have to consider the sun strength, the atmospheric absorption, the mirror surface, and the turning angle of the mirror. The first three of these effects are constant for the entire field (when assuming equal mirror surfaces), and are therefore incorporated into a single factor P_{raw} here. The last angle, however, is different for each mirror in the field, and has to be considered in more detail.

The two turning angles of a heliostat can be written as a single normal vector $\vec{n}$. The vector $\overrightarrow{tower}$ shall denote the direction of the tower as seen from the heliostat (to be exact: the direction of the aim point), and $\overrightarrow{sun}$ shall be the direction of the sun. As incoming and outgoing angles are equal (see Figure 5), the receiver normal is

$$\vec{n} = \text{normalize}(\frac{\overrightarrow{tower} + \overrightarrow{sun}}{2}) = \text{normalize}(\overrightarrow{tower} + \overrightarrow{sun}) \tag{9}$$

and the beam power is

$$P = \vec{n} \cdot \overrightarrow{tower} \cdot P_{\text{raw}} \tag{10}$$

This approach keeps the calculation fast while being simple and graphically intuitive, and eliminates the most obvious flaws of HFLCAL.

3. Optimization Problem

3.1. Notation

In the following, tuples over all measure points will be used, which we will call distributions. This concept has already been used: Q_{local} for example is such a distribution. They take values from $\mathbb{R}^{|M|}$. When stating $d \in \mathbb{R}^{|M|}$ or $d \in \mathbb{R}_+^{|M|}$, we indicate explicitly that d is such a distribution. Let d be an example distribution.

When writing d^m, the element in the distribution assigned to the measure point $m \in M$ is meant. Distributions can be written as

$$(d^{m_1}, \ldots, d^{m_n}) \tag{11}$$

For example, when having three measure points, we can have a distribution $(3, 6, 9)$, which means that the value assigned to the first measure point is 3, the value of the second one is 6 and the value for the third one is 9.

When taking a norm of such a distribution, it is interpreted as a vector, i.e.

$$\|d\|_r = \sqrt[r]{\sum_{m \in M} |d^m|^r} \tag{12}$$

$$\|d\|_\infty = \max_{m \in M} |d^m| \tag{13}$$

For example, the distribution $(3, 6, 9)$ from above would have a 1-norm of 18 ($= |3| + |6| + |9|$) and a ∞-norm of 9 ($= \max(|3|, |6|, |9|)$).

All operators that are not defined otherwise shall mean point-wise application. E.g.

$$(1, 2, 3) \cdot (4, 5, 6) = (1 \cdot 4, 2 \cdot 5, 3 \cdot 6) = (4, 10, 18) \tag{14}$$

and

$$\frac{1}{(2, 3, 4)} = (\frac{1}{2}, \frac{1}{3}, \frac{1}{4}) \tag{15}$$

3.2. Ideal Flux Map Model

In this model, we want to reach an ideal flux density Q^m_{target} for each point m. For this purpose, we define the distances between the desired and actual distributions, which we call ε:

$$Q_{\text{local}} - Q_{\text{target}} = \varepsilon \tag{16}$$

We want to minimize all $|\varepsilon^m|$.

Minimizing these values requires some additional definitions that contain several pitfalls. This is discussed in the following.

For combining the individual (local) epsilons into a single variable to optimize, we will use a norm. Usually, one would use a quadratic norm for summing up errors, but

as quadratic constraints aren't particularly compatible with linear MIPs, we will use a weighted sum of the 1- and ∞-norm[11]:

$$\varepsilon_{\text{total}} = \text{lerp}(\frac{\|k \cdot \varepsilon\|_1}{|M|}, \|k \cdot \varepsilon\|_\infty, w_\varepsilon) \tag{17}$$

with $k \in \mathbb{R}_+^{|M|}$ as weighting factors, allowing to weigh the ε-values of individual points differently (see later), and $w_\varepsilon \in [0, 1]$ as a factor to weigh the 1- and ∞-norm.

This combination of 1- and ∞-norm is used because the ∞-norm alone would leave room for Pareto optimizations: for example, $\varepsilon_1 = (-2, -1, 1, 2)$ and $\varepsilon_2 = (-2, 0, 0, 2)$ have the same ∞-norm (2), while the latter is in fact preferable and should therefore score better. Similar examples can be constructed for the 1-norm: a distribution $(1, 1, -1, -1)$ would be preferable over $(0, 0, 2, -2)$, where both would have a 1-norm of 4.

The factor k in the last equation allows us to weight individual epsilon values, depending on the sensitivity of the thermal model in m. For now, we will assume that a lower Q^m_{target} means more sensitivity, i.e. that we're interested in the *relative* distribution of power:

$$k = \frac{1}{Q_{\text{target}}} \tag{18}$$

There is still a subtle problem with this definition: The Q_{target} is scaled for a specific amount of spillage losses – usually 0 %. This means that Q_{target} would be an ideal distribution if the entire available power would hit the target. However, this is mutually exclusive with the property that we're trying to achieve with this constraint: a distribution where all heliostats are targeting the centre of the receiver is most likely never equivalent to the thermodynamically ideal distribution.

A first thought may now be: When we're trying to reach a certain fixed distribution, which can physically not be reached, the optimum value would be a value that both matches the distribution and uses as much power as possible. This would, of course, never reach the desired distribution, but would at least come very close to it, and therefore minimize the spillage losses on the way. However, using an unreachable (i.e. wrongly-normalized) Q_{target} creates certain artefacts. In the remainder of this section, we will first explain what the possible artefacts are, then propose a solution, and then come back and compare the solution with this first idea.

Assume that we have an example Q_{target} distribution

$$Q_{\text{target}} = (90, 90, 90) \tag{19}$$

and two Q_{local} distributions

$$Q_{\text{local},1} = (72, 72, 90) \qquad Q_{\text{local},2} = (72, 81, 81) \tag{20}$$

[11] To get comparable results, the 1-norm is divided by $|M|$, essentially resulting in the average ($\bar{\varepsilon}$).

The latter is better, as it distributes the power more evenly, which is what the image describes. However, when looking at the ε-values, we're getting the following results:

$$\begin{aligned}
&\text{Using the } \infty\text{-norm:} \\
&\|Q_{\text{target}} - Q_{\text{local},1}\|_\infty = \max(18, 18, 0) &&= 18 \\
&\|Q_{\text{target}} - Q_{\text{local},2}\|_\infty = \max(18, 9, 9) &&= 18 \\
&\text{Using the average:} \\
&\overline{|Q_{\text{target}} - Q_{\text{local},1}|} = \overline{\{18, 18, 0\}} &&= 12 \\
&\overline{|Q_{\text{target}} - Q_{\text{local},2}|} = \overline{\{18, 9, 9\}} &&= 12
\end{aligned} \tag{21}$$

Using Equation 16, these solutions are equally good, on both metrics. But actually only $72 + 72 + 90 = 72 + 81 + 18 = 234$ units are to be distributed in either system, which means that a more fitting target distribution would be $Q'_{\text{target}} = Q_{\text{target}} \cdot \frac{234}{270} = (78, 78, 78)$. In that case, we would get

$$\begin{aligned}
&\text{Using the } \infty\text{-norm:} \\
&\|Q'_{\text{target}} - Q_{\text{local},1}\|_\infty = \max(6, 6, 12) &&= 12 \\
&\|Q'_{\text{target}} - Q_{\text{local},2}\|_\infty = \max(6, 3, 3) &&= 6 \\
&\text{Using the average:} \\
&\overline{|Q'_{\text{target}} - Q_{\text{local},1}|} = \overline{\{6, 6, 12\}} &&= 8 \\
&\overline{|Q'_{\text{target}} - Q_{\text{local},2}|} = \overline{\{6, 3, 3\}} &&= 4
\end{aligned} \tag{22}$$

And now the second solution is clearly better, just as expected.

Now, this is a rather realistic example, and the solution may be highly non-optimal when we're not considering this effect. This requires us to integrate the corrected Q_{target} into the calculation of ε (compare to Equation 16):

$$Q_{\text{local}} - (1 - s) \cdot Q_{\text{target}} = \varepsilon \tag{23}$$

However, if we minimize $\varepsilon_{\text{total}}$ now, we get an optimal solution that has $Q_{\text{local}} = 0$: this configuration has 100 % spillage losses, which makes $\varepsilon = 0$. Therefore, we will optimize for the weighted sum of spillage losses and epsilon, using a weighting parameter $w_s \in 0, 1$:

$$\text{Minimize } 2 \cdot \text{lerp}(s, \varepsilon_{\text{total}}, w_s) \tag{24}$$

(The factor of 2 is just there for easier reasoning about the results, as it simplifies the formula to $s + \varepsilon_{\text{total}}$ for $w_s = 0.5$.)

An alternative would be to introduce an additional constraint that limits the spillage losses, but experimentation with the MIP revealed that this leads to the maximum spillage losses being almost exactly reached and never fall significantly below that value. This goes against the goal to maximize the energy output.

A second alternative would be to run the optimization without the correction, inspect the spillage losses, and use them to scale Q_{target} accordingly, then run the optimization again. However, it's not known whether this results in anything near the "true" optimum, or whether this process stabilizes at all. In any case, the optimization would take multiple times longer. For all of these reasons, we're not considering these alternatives.

The only unknown parameter for our system is now w_s.

With $w_s = 0.5$ (and a k as defined in Equation 18), we will get a very similar result to the approach of using an unscaled Q_{target}, where the spillage losses and the deviation are handled at once, except that it is corrected for the effect described above. This value can also be refined by experimenting with the thermal simulation.

If we would assume that the epsilon already contains the spillage losses, and that we can only optimize for the epsilon, ignoring the above correction, then we're effectively just weighting s and $\varepsilon_{\text{total}}$ equivalently, except that there will be errors introduced by not applying the correction. Therefore, this approach is definitely preferable to the second idea stated above.

We will refer to this model as the "first model" for short.

3.3. Maximum Flux Map Model

While ideal flux densities are not available yet, more information from the producers of the receivers revealed that strict maximum flux densities are. Multiple images for maximum allowed flux densities (Q_{max}) can be calculated for different values of Q_{total}. These can be approximated by linear interpolation.

This leads us to the definition of a second model: If we know upper and lower bounds for the spillage losses, we can calculate two maximum images for those values of total received powers, and use them to interpolate maximum images for arbitrary values of Q_{total}. The maxima in turn can be formulated as a constraint. When we add the spillage losses as the objective, we get a model that is entirely linear and thus suited for a MIP solver.

Formally, we interpolate using the total energy, which we normalize between 0 and 1 for convenience. We call this normalized value r, where 0 corresponds to 100 % spillage losses and 1 corresponds to 0 % spillage losses. The values between which we interpolate between shall be called Q_0 and Q_1, which should be the supporting points at 0 and 1 respectively.

$$r = \frac{Q_{\text{total}}}{Q_{\text{total}}^{\text{max}}} = 1 - s \tag{25}$$

$$Q_{\text{max}}^m = \text{lerp}(Q_0^m, Q_1^m, r) \quad \forall m \in M \tag{26}$$

Then we add the constraint

$$\frac{Q_{\text{local}}^m}{Q_{\text{total}}^{\text{max}}} \leq Q_{\text{max}}^m \quad \forall m \in M \tag{27}$$

and set the objective to

$$\text{Minimize } s \tag{28}$$

For our example calculations, we're using a single receiver model Q_{base} which is normalized such that $\sum_{m \in M} Q^m_{\text{base}} = 1$ and multiply it with factors f_0 and f_1 to get to Q_0 and Q_1.

$$\begin{aligned} Q_0 &= f_0 \cdot Q_{\text{base}} \\ Q_1 &= f_1 \cdot Q_{\text{base}} \end{aligned} \tag{29}$$

This simplifies our formula to:

$$Q_{\max} = \text{lerp}(f_0, f_1, r) \cdot Q_{\text{base}} \tag{30}$$

We will refer to this model as the "second model" for short.

3.4. Summarized Problem Description

In summary, we have two optimization problems. Both of them share some constraints:

(Shared constraints:)

$$\begin{aligned} Q^m_{\text{local}} &= \sum_{h \in H} Q(h, a_h, m) \quad \forall m \in M \\ Q_{\text{total}} &= \sum_{m \in M} Q^m_{\text{local}} && \text{(see Eqn. 1)} \\ s &= 1 - \frac{Q_{\text{total}}}{Q^{\max}_{\text{total}}} && \text{(see Eqn. 3)} \end{aligned} \tag{31}$$

The first model is then summarized as:

Minimize

$$2 \cdot w_s \cdot s + 2 \cdot (1 - w_s) \cdot \varepsilon_{\text{total}} \qquad \text{(see Eqn. 24)}$$

Subject to:

(Shared constraints) (32)

$$Q^m_{\text{local}} - (1 - s) \cdot Q^m_{\text{target}} = \varepsilon^m \quad \forall m \in M \qquad \text{(see Eqn. 23)}$$

$$\varepsilon_{\text{total}} = w_\varepsilon \cdot \frac{\|k \cdot \varepsilon\|_1}{|M|} + (1 - w_\varepsilon) \cdot \|k \cdot \varepsilon\|_\infty \qquad \text{(see Eqn. 17)}$$

with default weights of

$$k^m = \frac{1}{Q^m_{\text{target}}} \quad \forall m \in M \qquad \text{(see Eqn. 18)} \tag{33}$$

$$w_\varepsilon = 0.5 \tag{34}$$

$$w_s = 0.5 \tag{35}$$

The second model is summarized as

$$
\begin{aligned}
&\text{Minimize} \\
&\qquad s && \text{(see Eqn. 28)} \\
&\text{Subject to:} \\
&\qquad \text{(Shared constraints)} \\
&\qquad Q_{\max} = \operatorname{lerp}(Q_0, Q_1, r) && \text{(see Eqn. 26)} \\
&\qquad \quad \textit{simplified: } Q_{\max} = \operatorname{lerp}(f_0, f_1, r) \cdot Q_{\text{base}} && \text{(see Eqn. 30)} \\
&\qquad \frac{Q^m_{\text{local}}}{Q^{\max}_{\text{total}}} \leq Q^m_{\max} \quad \forall m \in M && \text{(see Eqn. 27)}
\end{aligned}
\tag{36}
$$

We will use default parameters of $f_0 = 0.3$ and $f_1 = 1.2$, chosen arbitrarily.

4. Solving

4.1. MIP

The MIP is based on a construct where we have boolean variables $b_{h,a}$ for all $a \in A$, $h \in H$. These variables are set to 1 if the heliostat h targets the aimpoint a, otherwise they are set to 0. Obviously, one heliostat can only point to at most one aimpoint, so the following constraints are introduced:

$$\sum_{a \in A} b_{h,a} \leq 1 \qquad \forall h \in H \tag{37}$$

We can now introduce continuous variables for the local flux density in certain spots, which are defined via constraints.

$$Q^m_{\text{local}} = \sum_{h \in H} \sum_{a \in A} Q(h, a, m) \cdot b_{h,a} \qquad \forall m \in M \tag{38}$$

All the other constraints – from both models – can be used directly as they are formulated in Section 3.

Note that the constraint from Equation (38) causes the size of the MIP (in number of linear terms) to be in $O(|H \times A \times M|)$. As the receiver surface area is probably linearly dependent on the number of heliostats in the field, the size of the MIP is in approximately $O(|H|^3)$. Due to this high complexity, we will approximate and try other methods later.

The idea of this formulation was taken from [1].

4.2. Grouping

A natural simplification of the MIP is the unification of heliostats with very similar images. For this purpose, the set of heliostats H is divided into a partitioning P and each set $H' \in P$ will be represented by a single "virtual" heliostat $v(H')$.

For a cluster of heliostats $H' \subset H$, all boolean variables $b_{h,a}$ (with $h \in H'$) for a single aimpoint a can be replaced by only one variable $z_{H',a}$. The limitation that a heliostat can only point onto a single aimpoint can be relaxed – a cluster of heliostats may point onto as many aimpoints as it contains heliostats:

$$\sum_{a \in A} z_{H',a} \leq |H'| \qquad \forall\ H' \in P \tag{39}$$

The idea behind this approach is to improve the solver performance in multiple ways:

First of all, it simply reduces the number of variables. This both decreases the time spent in Simplex and the time required to load the problem file.

Then it also decreases the size of the search space. Heliostats that have a very similar image anyway are not differentiated any more by which heliostat points on what exactly, so this decision is removed from the solver.

Finally, this reformulation may create a MIP that is more suited for the solver by having less binary variables and more linearity to use. Subtrees could potentially be cut off quicker by the branch-and-bound algorithm, and there's a possibility that a bigger part of the search space doesn't have to be touched. There are more valid integer solutions in the vicinity of the linear relaxation optimum, which may mean that some of them are being reached faster.

The questions that arise here are:

1. What is $Q(v(H'), a, m)$ for a given group, i.e. what image does the virtual heliostat project onto the receiver surface?

2. Which metric should be used to evaluate how good a grouping H' is?

3. Which algorithm should be used to optimize on that metric?

All of these questions are interconnected.

The first approach, for demonstration purposes, was to simply put the heliostats into equally sized groups of similar distance and calculate the arithmetic mean position for the representative heliostat. The HFLCAL model then provided values for $Q(v(H'), a, m)$. This of course makes only limited sense, but served as a demonstrator that this indeed improves the solver performance drastically.

For the more structured approaches, we need to first define a metric. When grouping up heliostats, we want to minimize the error that is created. To be exact, we want to minimize the maximum error that occurs in Q_{local} by replacing a heliostat $h \in H'$ by its representative $v(H')$:

$$\text{Error bound on } Q_{\text{local}} = \min_{P \in PP(H)} \max_{m \in M} \sum_{H' \in P} \sum_{h \in H'} \max_{a \in A} |Q(h, a, m) - Q(v(H'), a, m)| \quad (40)$$

where $PP(H)$ is the set of all partitions of H. Knowing this error bound also allows us to make sure that Q_{max} is never exceeded in the second mode: When minimizing the maximum error that occurs, we can give an upper bound for the total error in each Q_{local}. As Q_{total} is simply the sum over Q_{local}, its relative error is bound by the maximum relative error on Q_{local}. This, in turn, allows us to calculate the maximum error on s and on Q_{max}. We can then use these numbers to scale Q_{max} down appropriately, so that its never exceeded even with the errors that we're introducing.

For the algorithm, a k-means seemed suitable for this task. It works as follows:

1. Choose k heliostats. Each of them is the representative of one cluster.

2. Assign every heliostat to the cluster it has the lowest error with (i.e. where $\max_{a \in A} \max_{m \in M} |Q(h, a, m) - Q(v, a, m)|$ is minimal, where v is the representative for the cluster as it was in the last iteration).

3. Update the representative heliostat of each cluster, i.e. update Q(v, a, m) (see below).

4. Repeat the second and third steps for n iterations.

Finally, the Q function was defined as the average of the individual heliostats Q values:

$$Q(v(H'), a, m) = \frac{\sum_{h \in H'} Q(h, a, m)}{|H'|} \tag{41}$$

This definition, however, resulted in the algorithm converging after very few (<5) iterations with all heliostats within one cluster and all other clusters being empty. This might be due to the nature of the central limit theorem, where many individual uncorrelated distributions eventually converge into a Gaussian distribution. When creating a single "average" distribution by summing up all profiles and dividing the values at the individual points by the number of profiles, there seems to be an average that is very close to all of the distributions.

As an alternative, the method from the first approach was reused: the arithmetic mean position of the heliostats of the group is taken, and its position is used for the representative heliostat, of which the flux density is calculated via HFLCAL. This worked better, the results of this approach can be seen in Figure 6.

4.3. Genetic Algorithm

As the MIP generation didn't seem to provide more room for optimization without approximation (see Section 5), other options were investigated. The choice fell on a Genetic Algorithm (GA), as some literature [4] claimed it to work well and it seemed easy to implement. Furthermore, by implementing MIP and GA methods side-by-side, a proper comparison between them can be made. Finally, the results can give a rough idea on how well similar search methods that consider the problem a black box like local search, simulated annealing etc. may work.

For the genetic algorithm, we use the objective as the fitness function of an alignment, alignments from $H \to A$ are the individuals. Other than this, we need to define recombination and mutation operators, and a simple method to generate starting solutions.

The starting solutions have all heliostats pointing onto the standby aimpoint, i.e. the spillage losses are 1. This solution is copied for the entire population. We don't really care about duplicate solutions as these are mostly vanishing very quickly due to mutations.

The selection is being done in two ways: first, a small part of the population with the highest objective value is always copied to the following iteration. The ratio between the number of elite individuals and the total population is a parameter to the optimization, which we call `elitismRate`.

In a second selection step, individuals are selected using the roulette wheel selection, i.e. the probability of each individual to survive is proportional to its score. As we're trying to minimize the objective, we use the inverse as the score:

$$\text{Score} = \frac{1}{\text{ObjVal}} \tag{42}$$

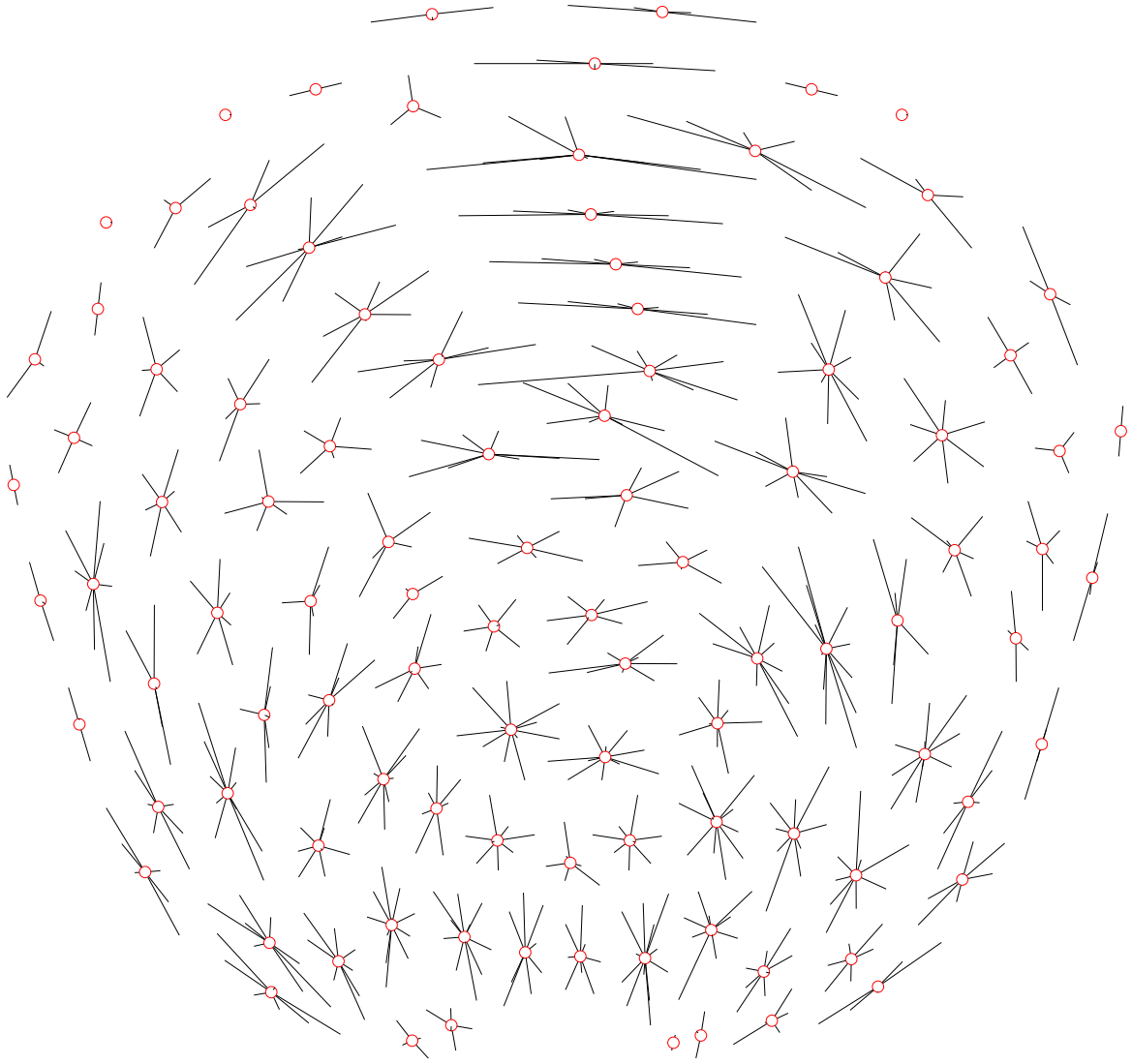


Figure 6: The heliostats as they are grouped by the k-means algorithm. The representatives are red, the original heliostats are on the tips of the lines. 5 iterations, 100 groups.

Parameter	Value
`populationSize`	100
`selectionRate`	0.5
`mutationRate`	0.005
`swapRate`	0.005
`elitismRate`	0.01
`iterations`	100
Model used	First model

Figure 7: Parameters used for performance measurements.

The ratio between the number of individuals being selected using this method and the total population size is called `selectionRate` and is also a parameter to the optimization.

The rest of the population is then filled up by recombination. This is done by selecting the aimpoint for each heliostat of the child randomly from one of its parents. In detail, the algorithm to generate a new solution is as follows:

1. Select two parents randomly. Both of the parents must be part of the group that was selected through either of the methods described above, i.e. both of the parents must survive.

2. For each heliostat, choose one of the two parents. In the new solution, the aimpoint of that heliostat is the same as the aimpoint of the selected parent of that heliostat.

After recombination, all non-elite individuals (i.e. those that weren't selected through elitism) are up for mutation. This is done by repeatedly selecting one of those individuals, selecting one heliostat, and assigning that heliostat to a random different aimpoint. The average ratio of mutated heliostat-aimpoint assignments per heliostat and individual is called the `mutationRate` and is a further parameter to the algorithm.

Another mutation operator was added which swaps the aimpoints of two heliostats within one individual. The parameter for the rate of swaps per heliostat per individual is called the `swapRate`.

Generally, constraint violations in the second model are reflected by increasing the objective massively. As the spillage losses can be at most 1, the value that is added per violation is 1, which means that every solution (with the second model) that violates constraints has an objective greater than or equal to 1. Inversely, whenever a solution has a reported objective smaller than 1, it does not violate any constraint.

The idea for using a genetic algorithm for this kind of problem was taken from [4].

Unlike the MIP approach, which is a black-box performance-wise due to the usage of the closed-source solver, we have a lot of influence on the performance of the GA. The evaluation of the model, as expected, proved to be the most expensive step in the GA. This will be subject to two optimizations in the following subsections.

As a baseline, the parameters as shown in Figure 7 were used. In this case, the first model was taken, but this has barely any influence on the overall performance, as the

biggest part of the calculation is the determination of Q_{local}. This setup needs 28.3 s to compute. When setting the iteration count to 0, we get to about 0.5 s, which is our initialization overhead and can't be reduced by tweaking this algorithm.

4.3.1. Caching the Image

A first and simple optimization was to apply memorization onto the task of computing the individual local energies. As hashtables had too much overhead to produce significant speedup, the memory locations were calculated directly. This was done by assigning consecutive indices to each heliostat. Aimpoints and measurepoints were indexed similarly. This allowed to compute a gap-free index:

$$\begin{aligned} \text{index}(h,a) &= \text{index}(h) \cdot |A| + \text{index}(a) \\ \text{index}(h,a,m) &= \text{index}(h,a) \cdot |M| + \text{index}(m) \end{aligned} \tag{43}$$

In the following, two chunks of memory were allocated: one for an array of booleans that indicated whether the image for an alignment $(h,a) \in H \times A$ has already been computed, the other one for the images of those alignments. When the image for an alignment is requested, it is first looked up in the boolean array whether this image has already been computed, and if it hasn't, it computes this image. In the end, it returns a pointer to the first element in this image.

For the same setup as above, this optimization results in a computation time of 7.4 s, creating a 3.8x speedup. With lower mutation rates and higher iteration counts, even higher speedups can be achieved.

4.3.2. The Local Flux Tree

Being an $O(|H \times M|)$ operation, the computation of local energies is the most expensive step while evaluating a solution, with all other steps only making up for O(|H| + |M|) operations.

We assume that we have a lot of very similar solutions, as the "survival of the fittest" method of the GA is very likely to let many descendants of a very successful individual survive, which then again recombine their already-similar genomes. This means that most of the time, the same numbers are added up to calculate Q_{local}.

However, when computing Q_{local} by sequentially adding the flux contributions of the individual heliostats, memorization of partial sums doesn't help much: on average, half of the additions are becoming invalidated (see Figure 8). One approach would be to subtract the old value from the sum and add the new one, but this may lead to diverging results due to floating-point inaccuracies that are getting bigger with each step.

Instead, we are organizing the additions in a balanced tree (see Figure 9). Each leaf contains the image of one heliostat with the alignment of the last computed solution. Each node contains the sum of the images of its children. When one of the leaves now changes their value, only additions on the path to the root have to be redone. When multiple leaves change, there is a chance that they invalidate the same nodes.

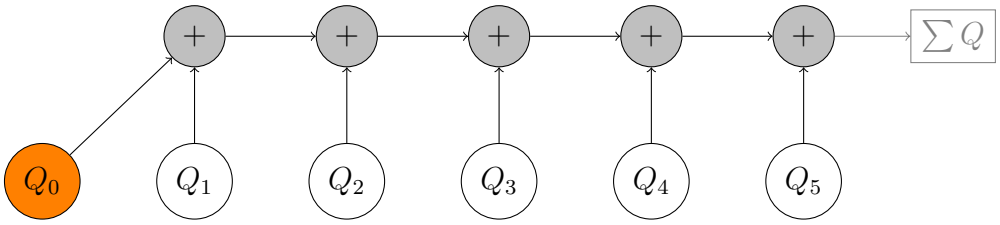


Figure 8: The dependency graph for sequential addition. An edge indicates that the result of the outgoing node influences the result of the incoming node. If Q_0 changes, all gray nodes have to be recalculated.

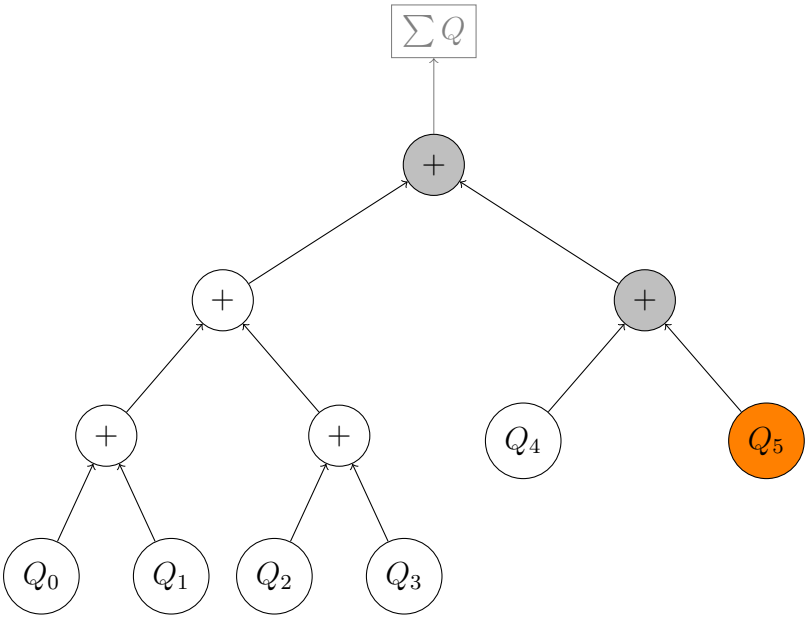


Figure 9: The dependency graph for tree addition. If any single Q_h changes, only the path to the root is invalidated, the other calculations can be reused.

In terms of complexity, the worst case doesn't get worse, and the best and average case improve in comparison to a sequential system. In all cases, the number of recalculated nodes is in $O(\log |H|)$, or $O(|M| \cdot \log |H|)$ for the time complexity, as the recalculation of each node requires $\Theta(|M|)$ operations. When all leaves change their values, the entire tree has to be recalculated - this is the worst case.[12] The tree construction just makes use of associativity, and when applying associativity laws, the total number of operations is conserved. This means that the complexity for recalculating the entire tree is in $\Theta(|H|)$.

In comparison, memorizing partial sums of the sequential addition have a best case time complexity when one image changes of only one addition having to be redone, resulting in an $\Theta(1)$ operation. The worst case, on the other hand, invalidates the entire chain of additions, resulting in a $\Theta(|H|)$ operation. As the invalidation of each image has the same probability in the GA, the average is that half of the nodes are invalidated, which is proportional to $|H|$, resulting in an average case time complexity of $\Theta(|H|)$ for a single change. For an arbitrary number of n changes, the situation is similar, with $\Theta(n)$ being the best case and average and worst case staying the same.

This results in the average case time complexity for updating the tree for an arbitrary but fixed number of changed leaves never being higher than the average case of recalculating a sequentially memorized result.

When being fed our test setup, this optimization - together with the caching above - takes 3.4 s to compute, creating another 2.2x speedup versus the previous optimization, or an 8.3x speedup in total.

[12] This does not only apply if all leaves change, half of them is enough if they are distributed in an unfortunate way.

5. Test Cases

5.1. General

For all test cases, the parameters as shown in Table 1 were used. Model-specific default values are as stated in Section 3.4. For the receiver model, i.e. the distribution used as both Q_{target} and Q_{base}, the data shown in Figure 10 was used.

Summarized results can be found in Table 2.

5.2. MIP

For testing, it was assumed that all heliostats have $P_{\text{raw}} = 1$ for the beam power. 12×6 aimpoints and measurepoints were used, distributed in a grid-like pattern over the surface of the receiver. All 624 heliostats of the PS10 plant were used. The optimization ran on Gurobi 7.5.1 on a 2013 laptop (see Appendix A.1).

The MIP generation took about 5.3 s for both scenarios. The resulting problem files, 62.3 MB each[13], took another 3 s to be read by the solver.

For the first model, the first solution was found within 50 s. It already had an objective value with a gap of only 2.37 % (ObjVal = 14.24 %, see Figure 11). After 1000 s, the solver was aborted - the last solution, found at 721 s, only reduced that gap to 1.35% (ObjVal = 14.09 %, see Figure 12). The best bound didn't change at all, it was constantly at about 13.90 %.

In the second model, we're setting $Q_0 = 0.3$ and $Q_1 = 1.2$. Again, the solver is run for 1000 s. The first solution is found after 27 s, the last one at 980 s (see Figure 13). The best bound didn't change this time either, it was constantly at about 4.24 %.

Note however that for the second model, these timings are extremely dependant on the parameter choice. For example, when changing Q_1 to 10 the solver only needs 6 s to do its job, while a setup with $Q_0 = 0$ and $Q1 = 1.3$ doesn't find any (non-trivial) solution in a reasonable time (tried for 1000 s).

[13] In the `.lp` file format.

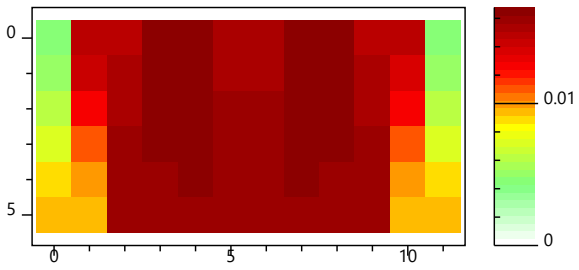


Figure 10: The receiver model used - this is the distribution that should be archived. The scale is relative, i.e. 0.01 means 1 % of the total energy reaching the receiver should be collected in this area.

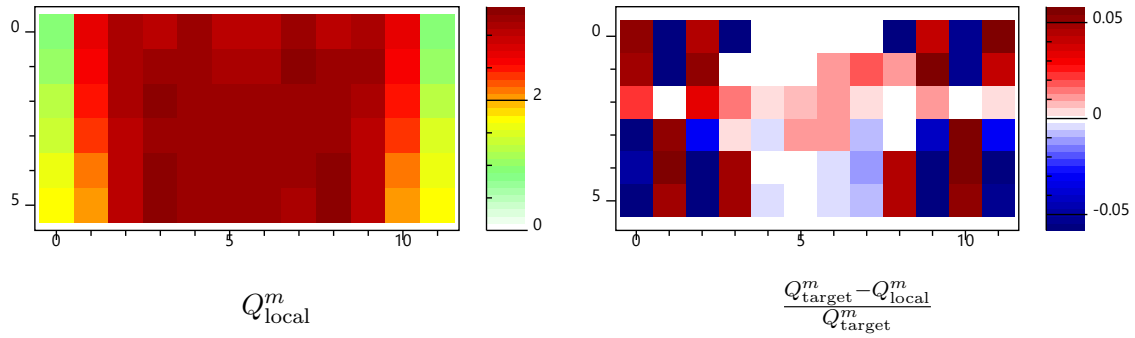
Parameter	Value
Total variance	4.525 mrad
Tower height	110 m
Aimpoints	12×6
Measurepoints	12×6
Receiver size	13.78 m × 12.00 m
Field layout	from PS10
Receiver model	See Figure 10
Receiver surface normal	South

Table 1: The parameters used for the test cases.

Method	Ideal flux map model		Maximum flux map model		
	Obj.	Time	Obj.	Time	Max.Violations
MIP	14.24 %	59 s [1]	5.49 %	27 s [1]	0 %
Grouping	15.60 %	7 s [2]	6.38%	15 s [2]	0.66 %
GA	27.52 %	17 s	41.33 %	15 s	0 %
Lower bound (from MIP)	13.09 %	-	4.24 %	-	-

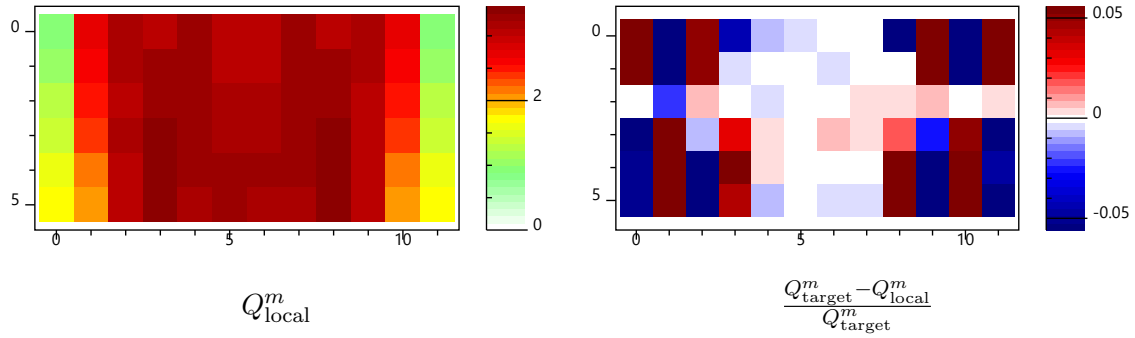
Remarks:
[1] Includes time for MIP generation and loading.
[2] Includes time for grouping, MIP generation and loading.

Table 2: The results. For the MIP, the time and objective to the first solution are shown. For the GA, the same parameters as above are used.



$s = 9.90\%$ $\|\varepsilon\|_\infty = 5.83\%$ $\|\varepsilon\|_1 = 2.85\%$ ObjVal = 14.24%

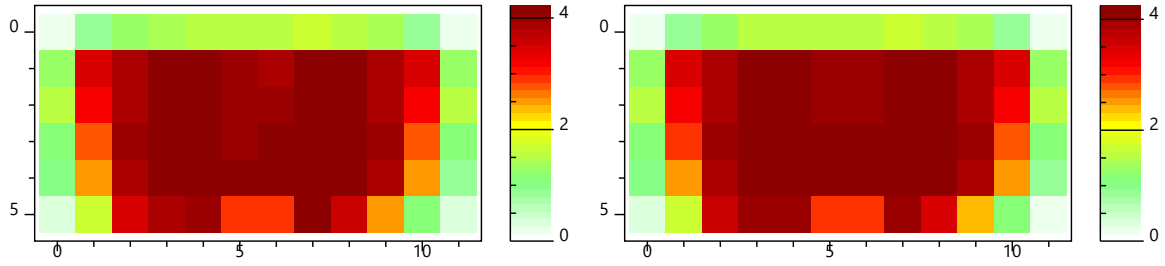
Figure 11: The first solution that was found for the first model using the MIP solver.



Q^m_{local} $\qquad\qquad$ $\frac{Q^m_{\text{target}} - Q^m_{\text{local}}}{Q^m_{\text{target}}}$

$s = 9.91\%$ $\qquad$ $\|\varepsilon\|_\infty = 5.58\%$ $\qquad$ $\|\varepsilon\|_1 = 2.78\%$ $\qquad$ ObjVal $= 14.09\%$

Figure 12: The last solution that was found for the first model using the MIP solver.



$s = 5.49\ \%$ $\qquad\qquad$ $s = 4.87\ \%$

Figure 13: The first and last solution found for the second model by the MIP solver.

This means that solving the problem in real-time with an MIP might be a viable strategy if enough computing power is employed. However, it does also mean that adapting within seconds (due to cloud coverage) is currently not possible.

Although the single-core performance of newer CPUs, which is crucial for solving MIPs, stagnated over the recent years, the speed of the available solvers still increases exponentially, which may lead to the conclusion that it is only a matter of time until these problems can be solved in an acceptable time. However, due to the large MIP size, generating and loading of the MIP creates a lower bound. And since many plants have much more than 624 heliostats, this approach is probably not applicable for the time being.

5.3. Grouping

The first area of investigation regarding this approach was the overhead introduced by the k-means itself. We want to find out:

1. How many iterations are required?
2. How long do they take?
3. How big is the error introduced?

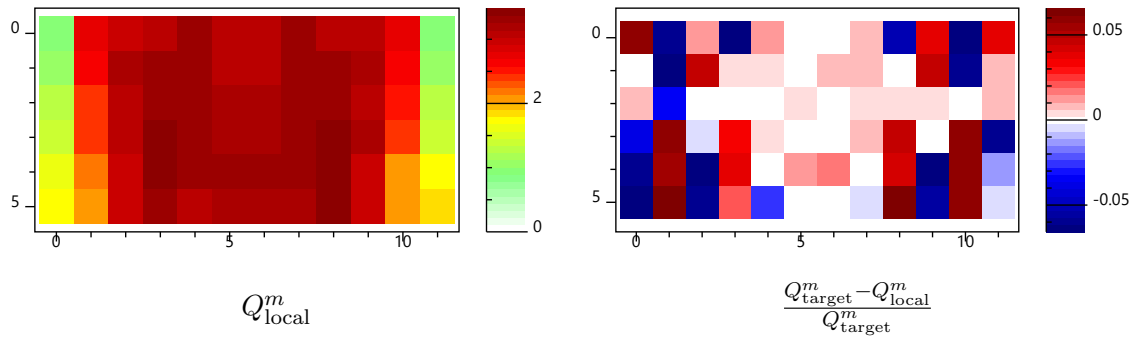
To answer the first question, we can visualize the individual steps of the algorithm. It shows some interesting behaviour: Instead of converging to a fixed point, it first shifts some of the groups to the front, where smaller groups appear. After that, the group in the back doesn't seem to stabilize, but is instead repeatedly replaced by one of the groups from the side. This may indicate that the method of taking the average position and recalculating the flux from there might not be the ideal approach.

The complexity of the k-means in its current state is in $O(k{\cdot}n)$, where k is the number of groups and n is the number of iterations. A setup with 50 groups and 5 iterations takes about 4.8 s to compute. The total time for grouping and MIP generation is at 5.3 s. This is about the same as needed for MIP generation without grouping, as the resulting LP file is much smaller.

The error introduced is quite bad in the worst case. Equation (40) gives a worst-case error of 2.42 on Q_{local} on any measure point (compare with the solutions above, e.g. Figures 11 and 12). This means that the approach of scaling up $Q_{\max}$ in the second model is not viable.

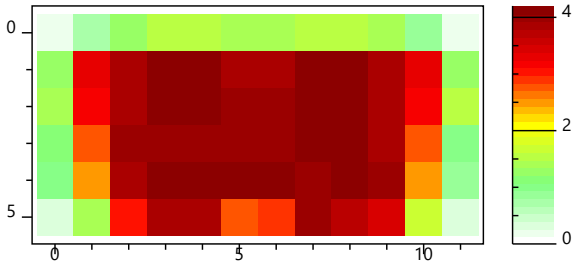
The MIPs generated are much faster at being solved. For the first model, the grouping from Figure 6 takes only 2 s to get to a solution, down from 50 s without grouping (see Figure 14). It is, however, noticeable that the objective bound has changed slightly - it is now at 13.75 % (compare to 13.90 % from the last section). The situation is less extreme for the second model, with 10 s for the first solution - down from 27 s (see Figure 15). Apart from the grouping, the same parameters as in the last section have been used.

As the theoretical error introduced by this approximation is very high, it was investigated how big the error is in practice. For this purpose, the last solutions generated



$s = 10.06\%$ $\|\varepsilon\|_\infty = 6.57\%$ $\|\varepsilon\|_1 = 2.63\%$ ObjVal $= 14.66\%$

Figure 14: The first solution that was found for by the MIP solver the first model with the grouping from Figure 6.



$s = 6.16\ \%$

Figure 15: The first solution that was found by the MIP solver for the second model with the grouping from Figure 6.

by the solver were taken and the aimpoints of heliostat groups were randomly assigned to the original heliostats. When evaluating this solution, it became apparent that, for the first model, the objective value was 15.60 %, up from the 14.66 % obtained without grouping (see above). The second model had 7 violations of its $Q_{\max}$-constraints, the biggest of which was 0.66 % (relative to the local $Q_{\max}$). Additionally, the spillage losses are 6.38 %, up from 6.06 %[14]. This means that the errors that occur in practice are very small. In the second model, f_{Q0} and f_{Q1} can just slightly be scaled down before given to the optimizer, and still return a result that is very close to the optimum. Depending on the exact requirements, the number of groups could be scaled up to reduce the error even further, at the cost of more computation time.

Finally, a small instance was created where it was simply assumed that the existing heliostats are already representative heliostats for groups of 16, creating a field of almost 10'000 heliostats. The solver showed no significant performance difference to the original field. This shows that the grouping approach scales very well, but it does

[14]This number is slightly different from the one found depicted in Fig. 15, because the randomized grouping returned a slightly different result.

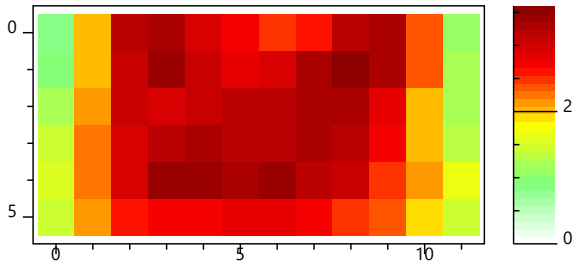


Figure 16: The resulting flux map for the solution generated by the genetic algorithm for the first model.

also indicates that the speculative claim that this reformulation is faster due to the different nature of the search space from the last subsection is actually wrong. If it was true, it would be expected that bigger groups actually make the solver take less time.

5.4. Genetic Algorithm

The following remarks are valid for both models, but we will use the first model for demonstration purposes.

After experimenting a bit, it became apparent that there is a tradeoff between fast convergence to a bad value at the beginning and slow convergence, but to a better value long-term (see Figure 17). To overcome this issue, we run the genetic algorithm multiple times, each instance having different parameters. This way, we can use the high mutation rate in the beginning to get into "the right direction", and then move the limit later.

Figures 18 and 19 show the convergence for both models. Notice how the limits that are reached are much higher than the values obtained from the MIP. Lowering the mutation rate further does not improve convergence at this point, as it results in an average of less than one mutation per solution per iteration.

The swaps did not lead to a faster convergence or better goal, as toggling them showed. When looking at the returned solutions (see Figure 16), it becomes apparent why this may be the case. Consider the bulk of superfluous light in the bottom left and the missing flux in the top center. It seems like there is potential for a pareto improvement there by moving some of the heliostats over. However, the algorithm is not aware of the fact that the swaps are a very fine-tuning operation, while the actual problem is that low-hanging fruits are not gathered.

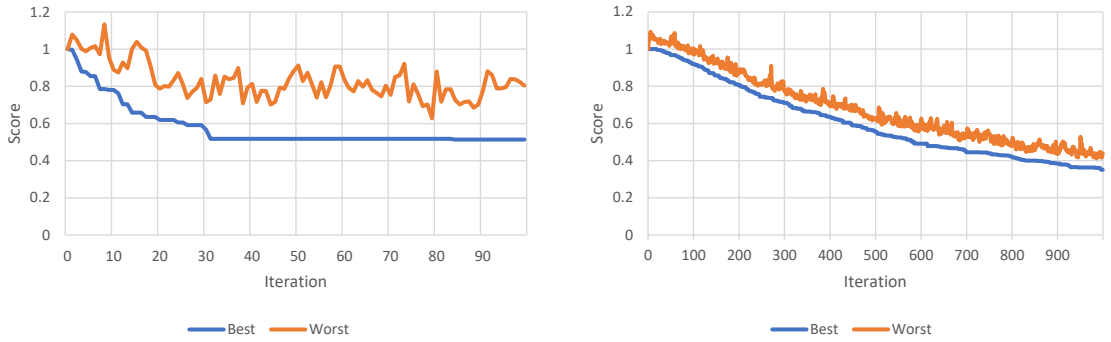
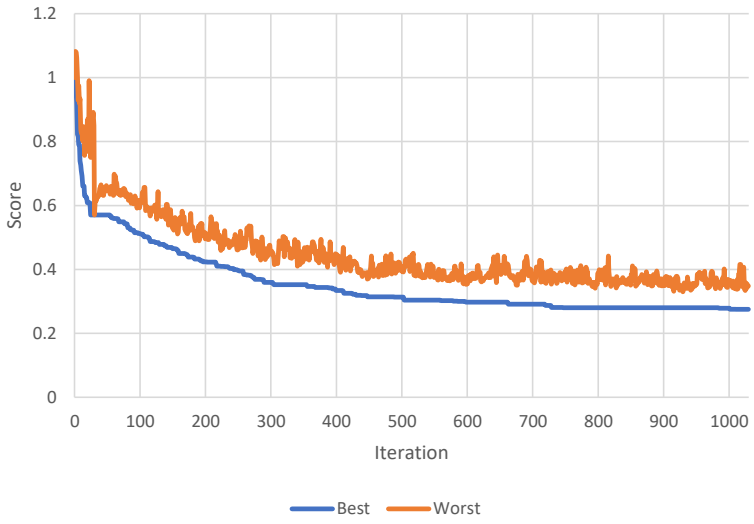


Figure 17: Convergence of the GA with two different sets of parameters. On the left, 100 iterations at 10 individuals in each population, 10 % mutation, 1.7 s required. On the right, 1000 iterations at 100 individuals in each population, with 0.1 % mutation and 0.1 % swaps, 13.7 s required.

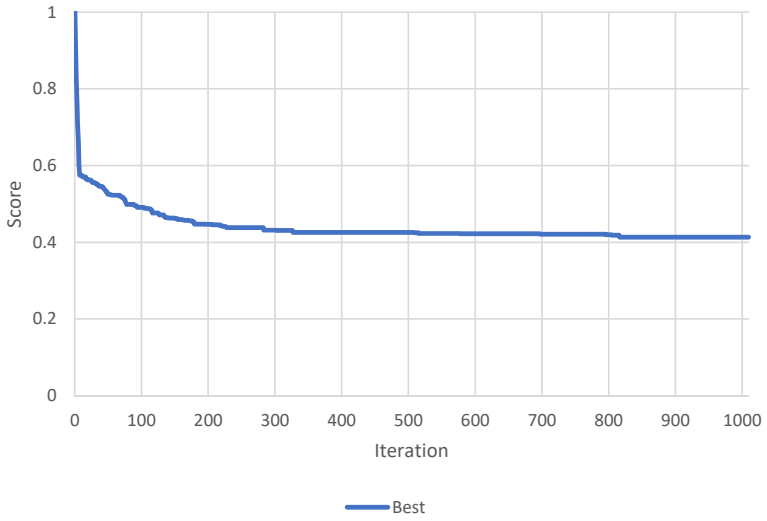


Parameter	Value
`populationSize`	10
`selectionRate`	0.5
`mutationRate`	0.1
`swapRate`	0.1
`elitismRate`	0.1
`iterations`	30

↓

Parameter	Value
`populationSize`	100
`selectionRate`	0.5
`mutationRate`	0.001
`swapRate`	0.001
`elitismRate`	0.01
`iterations`	1000

Figure 18: Convergence of the genetic algorithm applied to the first model. The parameters for the two instances of the genetic algorithm are on the right. The total running time was 16.9 s.



Parameter	Value
`populationSize`	10
`selectionRate`	0.5
`mutationRate`	0.1
`swapRate`	0.1
`elitismRate`	0.1
`iterations`	10

↓

Parameter	Value
`populationSize`	100
`selectionRate`	0.5
`mutationRate`	0.001
`swapRate`	0.001
`elitismRate`	0.01
`iterations`	1000

Figure 19: Convergence of the genetic algorithm applied to the second model. The "worst" line was omitted, as it would rescale the vertical axis too much. After a brief period at the start, the worst solution score fluctuates chaotically between 1.5 and 16. The total running time was 15.0 s.

6. Conclusion and Outlook

In summary, the Aiming Problem was tackled by the development of two models. A new light model, derived from HFLCAL, was added. A MIP formulation was presented and evaluated, and an approximation was made by grouping similar heliostats together. Finally, a Genetic Algorithm was implemented.

The models have the big flaw of not being validated in any way. There hasn't even been any direct contact to the responsible manufacturers, and no real-world or simulation data was available for testing. In fact, the receiver model has a different aspect ratio than the assumed receiver, and is actually for a cylindrical receiver and a 360 field, while the site in question has a flat receiver and a south field.

The models could be further extended to cover other restrictions of such plants. For example, many real-world receivers have a ceramic protective edge to protect the tower from excess heat and radiation. Although these covers can take a bigger amount of thermal energy than the materials in the receiver before they break, the latter is being cooled by the relatively cold medium that transports the heat away. This means that the flux density on these areas has additional constraints. Also, covering local minima in between the aimpoints with measurepoints may be interesting for stress computation.

The HFLCAL adaption might be useful not only when working on the Aiming Problem, but for all kinds of problems affiliated with solar power towers, e.g. field layout or performance prediction. It fills a gap between oversimplifying and overcomplicated, compute-heavy light models.

Using MIP to solve the Aiming Problem does not seem promising when trying to get exact results, especially with plants bigger than the 624-heliostat PS10. However, the approach to group heliostats together is very scalable and may pose as a solution. Although the theoretical error bounds are quite high, there is not much error in practice.

When further investigating the path of grouping heliostats together, another optimization problem appears: the assignment of heliostats to the aimpoints of their group. This is considered future work.

A Genetic Algorithm, on the other hand, does not seem suitable for these kind of problems in its raw form. Although the solutions can be generated much faster than by the MIP solver, their quality is very poor. There may be potential to improve upon these results by modifying the algorithm.

One such modification could be non-uniform probability distributions for the mutations of heliostats: when the light distribution is altered in areas where it's already good, it's very likely that the only result is wasted computation time. If it's less likely to touch these heliostats, convergence to higher values may be achieved. This may work well with modifications to the local flux tree, which could have less depth for heliostats that are regularly moved. Then the decision on which heliostats to move to which aimpoint with which probability poses much room for optimization. As arbitrary amounts of data can be generated on which decisions are good and which aren't, unsupervised learning could be used for this purpose. In particular, neural networks

would be feasible: the inputs could be differences of the local flux, the image of one heliostat, and ideal/maximum flux densities, while the output is a probability to move that heliostat.

Generally, a path that hasn't been investigated so far is parallelism. All of the computations (with the exceptions of the black-box MIP solver) are running sequentially right now. However, many areas are open to being parallelized. These are:

- The MIP generation. Many parts of the MIP are independent of each other (e.g. the constraints defining the local energy) and could be generated at the same time. Only the composition of these parts needs to be done sequentially.
- The grouping. In each iteration, every heliostat's image must be compared with every group to assign it optimally. This can be done in parallel for all heliostats at once, however requires some synchronization for the gathering of the results.
- The genetic algorithm. In each iteration, many individuals are evaluated. All of this evaluation can be done in parallel (when using seperate local energy trees and caches for each thread), only the selection steps need to be done sequentially.

When considering that processors with 20 or more (logical) cores are becoming more and more common - with individual core speeds stagnating -, these avenues should be definitely considered if there is a need for more performance.

A. Interface Documentation

A.1. Project Layout

The code was mainly written in C++, with a frontend written in C♯ to interactively display the solver progress and graphics generation. For solving, Gurobi 7.5 was used. GCG was also briefly tried, but quickly exhausted more than 8 GB of memory, even for simple MIPs.

Most calculations were run on the author's machine (Intel Core i7-3630QM @ 2.4 GHz; 8 GB RAM @ 1600 MHz).

The C++ code was embedded into the SolarTower project[15], although there are barely any dependencies to the rest of that project. It could be easily departed from it.

A.2. Building and Running

In its current state, the project builds with CMake on Linux. When using Windows to build the project, it is recommended to use the Linux Subsystem for Windows[16]. Also, the use of Ninja[17] instead of make can considerably speed up builds, especially consecutive ones.

The design of the program is such that it takes in commands via JSON. The specification of these commands is outlined in Section A.3. A JSON command file `file.json` can be executed by starting the program with the parameter `--commandFile=file.json`. By default, the output is written to `../../output.txt`, but an alternative output file `otherFile.txt` can be specified by using the command line parameter `--output=otherFile.txt`. Alternatively, the program can be started with the `--server` flag, which opens a server on port 12456 that listens for incoming TCP connections. When a connection is established, it reads the JSON code from the TCP stream, evaluates it, and writes the result back into the stream. This mechanism is used by the frontend application, which assumes that a command server is running at localhost.

A.3. JSON Interface

The JSON interface is documented through a JSON Schema file, which can be found under `data/AimingStrategyRequestSchema.json`.

B. Code Documentation

This section gives a brief overview over the main C++ codebase to assist reading the code. It is not intended to cover all details, and not the entire public interface, either.

[15] `https://git.rwth-aachen.de/Computational-Renewable-Energy/Solar-Tower.git`
[16] `https://msdn.microsoft.com/en-us/commandline/wsl/about`
[17] `https://ninja-build.org/`, run CMake with `-G Ninja`

```
std::unique_ptr<ILP> ilp = std::make_unique<ILP>();
auto a = ilp->addVariable("a", CONTINUOUS_POSITIVE);
auto b = ilp->addVariable("b", INTEGER);
auto c = ilp->addVariable("c", BINARY);

ilp->objective = 3 * a + 5 * b + 22.3 * c;
ilp->addConstraint("someConstraint", 2 * a + 5 < b);
ilp->addConstraint("someOtherConstraint", 1.34 * c == 5);

std::cout << ilp->toString() << std::endl;
```

Listing 1: Intuitive operator usage

B.1. ILP Framework

To generate the ILPs, a simple framework was written. This section gives a brief overview of this framework.

The class structure is laid out as follows:

- The main class, `ILP`, contains one objective of type `LinearTerm`, a list of `Constraints`, and a list of `Variables`. It can be serialized into an LP file.
- A `Constraint` represents an inequality, and has two `LinearTerms` for the left and right side, as well as a `ConstraintType`, which is an enum representing its sense (`<`, `>`, `<=`, `>=`, `=`)
- A `LinearTerm` contains a list of `AtomicLinearTerms`.
- An `AtomicLinearTerm` contains a `Variable` and a `double`. If the variable is set to an empty value, the term is interpreted as a constant. Otherwise, it represents a product of a constant and a variable.
- A `Variable` has a name (a `std::string`) and a type (an enum `VariableType` that can take the values `Binary`, `Integer` and `Continuous`).

By overloading the operators, constraints can be generated in a very intuitive fashion, as showed in Listening 1.

B.1.1. Remarks Related to the LP File Format

Note that the generated LP files are normalized to be compatible with Gurobi. There does not seem to be a standard definition of the `.lp` file format, but the documentation of various solvers in combination with additional experiments suggest that purely linear constraints will be interpreted correctly when they conform to the regular expression[18]

[18] These are standard Perl Regular Expressions, as used in most RegEx libraries. `\w` stands for an alphanumeric or underscore character, `\d` stands for a digit. See e.g. `http://perldoc.perl.org/perlre.html` for a complete documentation.

```
(number?  varname)+ sense number
```

where

`number` is `(\d+(\.\d+)?)`,
`varname` is `[a-zA-Z]\w*`, and
`sense` is `<|<=|=|>=|>`.

This means that individual summands may be moved to the correct side of the inequation, and single constant on the right hand side is a sum of all constant summands in the inequation.

To specify which values a variable can take (e.g. whether its values must be integer, binary etc.), the variables are listed in variable lists at the end, which are prefixed with the type that the variables in that list should have. However, to express that a variable can hold any real number (that it is "continuous"), they mustn't be appearing in any of those lists. In fact, this is the only way to create continuous variables.

B.2. Solar Tower Representation

B.2.1. SolarTowerSite

Provides access to all information about the plant that could be relevant during optimization. These include instances of all classes explained in the remainder of this section.

B.2.2. ReceiverModel

Provides the receiver model, i.e. the image that is used for Q_{target} and Q_{max}.

B.2.3. BeamEnergyProvider

Provides methods to determine the image that is casted by an individual heliostat.

This class is abstract. It is implemented by `HflcalBeamEnergyProvider`, which provides the HFLCAL-based method for flux approximation that was presented in Section 2.3. Other implementations could be added for other types of providers.

B.2.4. Receiver

Represents the receiver and contains information about its dimensions and resolution.

Also provides aimpoints and measure points, which are provided as `ReceiverSurfacePointCollection` instances. These structures can directly be iterated, but also provide methods to count the total number of points and convert 2D indices to real-world positions.

B.2.5. Evaluator

Represents a model. Provides the ability to evaluate a solution and to generate a MIP for an instance.

This class is abstract, it is implemented once for each of the two models.

This superclass also supervises the use of the optimization structures introduced in the Sections 4.3.1 and 4.3.2.

B.3. Solver Interface

B.3.1. JsonCommandProcessor

Processes the JSON commands. These commands can come in via TCP, in which case this class is called from the `CommandServer` class, or from the program itself, via `AimingStrategyMain.cpp`.

B.3.2. GroupingAlgorithm

Represents a function that partitions a set of heliostats into groups.

This class is abstract. It is e.g. implemented by `KMeansGroupingAlgorithm`, which groups heliostats via a k-Means algorithm, and `DoNothingGroupingAlgorithm`, which simply returns the same heliostats again.

B.3.3. SolvingAlgorithm

Represents a method to generate an optimized assignment of heliostats to aimpoints.

This class is abstract. It is implemented by `GeneticAlgorithm` and `GeneticIterativeAlgorithm`, where the latter is a variant of the former that allows different parameter sets to be executed one after another.

Note that there is no ILP solver; no direct ILP interface was integrated to avoid introducing further dependencies. To solve via ILP, generate an LP file and feed it to the solver of your choice.

B.3.4. SimpleSolution

Represents an assignment of heliostats to aimpoints.

Note that this class only supports an $n : 1$ mapping; it allows assigning each heliostat group to one aimpoint only, hence the name.

B.4. Utility classes

These classes were introduced to minimize the external dependencies that the program requires.

B.4.1. Array2D

This class represents a simple way of accessing two-dimensional data. It's especially useful when working with receiver models.

B.4.2. Vector2<T>

This class represents a classical euclidean vector and provides common operations such as vector addition, vector scaling, length calculation, JSON deserialization and conversions.

C. Nomenclature

C.1. Symbols used throughout the document

Symbol	Domain	Description
H	-	The set of heliostats.
h	H	Used to refer to a single heliostat.
A	-	The set of aim points.
a	A	Used to refer to a single aimpoint.
M	-	The set of measure points.
m	M	Used to refer to a single measure point.
Q	$H \times A \times M \to \mathbb{R}_+$	A function determining the power that a heliostat h contributes to a measure point m when targeting an aimpoint a.
Q_{local}	$\mathbb{R}_+^M$	The local power measured in each measure point on the receiver surface.
Q_{total}	$\mathbb{R}_+$	The total power arriving at the receiver surface.
$Q_{\text{total}}^{\text{max}}$	$\mathbb{R}_+$	The maximum power that could arrive at the receiver surface.
s	$[0, 1]$	The spillage losses, i.e. the ratio of radiation that misses the receiver. A value of 0 means that no light misses the receiver, a value of 1 means that no light hits the receiver.
lerp	$D \times D \times [0, 1] \to D$	Linearly interpolates between two values. D can be any vector space, e.g. $\mathbb{R}$ or $\mathbb{R}^M$. If the third parameter is 0, only the first value contributes to the result, if it is 1, only the second value contributes.

C.2. Symbols used in the context of the image generation algorithm

Symbol	Domain	Description
normalize	$\mathbb{R}^3 \to \{\vec{x} \in \mathbb{R}^3 : \lvert\vec{x}\rvert = 1\}$	Normalizes a vector, i.e. converts a vector to another vector with the same direction and length of 1.
P	$\mathbb{R}_+$	The total power reflected by a heliostat.

Symbol	Domain	Description
ϕ	$[-\frac{\pi}{2}, \frac{\pi}{2}]$	The angle between the normal of the receiver surface and the direction of the incoming light.
$\overrightarrow{n}$	$\{\overrightarrow{n} \in \mathbb{R}^3 : \lvert\overrightarrow{n}\rvert = 1\}$	The normal of the heliostat surface.
$\overrightarrow{t}$	$\{\overrightarrow{t} \in \mathbb{R}^3 : \lvert\overrightarrow{t}\rvert = 1\}$	The direction of the tower, as seen from the heliostat. (To be exact: the direction of the aim point.)
$\overrightarrow{s}$	$\{\overrightarrow{s} \in \mathbb{R}^3 : \lvert\overrightarrow{s}\rvert = 1\}$	The direction of the sun.
P_{raw}	$\mathbb{R}_+$	The power reflected by a heliostat if the sun was exactly behind the tower, i. e. if its effective surface area was equal to its actual surface area.

C.3. Symbols used in the context of the Ideal Flux Map Model

Symbol	Domain	Description
Q_{target}	$\mathbb{R}_+^M$	The ideal amount of power for each point on the surface.
ε	$\mathbb{R}^M$	The difference between the ideal and the actual amount of power on each point on the surface.
$\varepsilon_{\text{total}}$	$\mathbb{R}_+$	A measure of the deviation of the actual and ideal power distribution.
k	$\mathbb{R}_+^M$	Weighting factors for the calculation of $\varepsilon_{\text{total}}$ for each individual point on the surface.
w_ε	$[0, 1]$	A weighting factor that weighs the 1- and ∞-norm. If it is zero, only the 1-norm contributes to $\varepsilon_{\text{total}}$. If it is one, only the ∞-norm contributes to it.
w_s	$[0, 1]$	A weighting factor for the contribution of s and $\varepsilon_{\text{total}}$ to the objective. 0 means that only $\varepsilon_{\text{total}}$ is optimized for, 1 means that only s is optimized for.

C.4. Symbols used in the context of the Maximum Flux Map Model

Symbol	Domain	Description
r	$[0, 1]$	The ratio of light that hits the receiver. This is simply the inverse of the spillage losses ($r = 1 - s$), so a value of 0 means that no light hits the receiver and a value of 1 means that no light misses the receiver.
Q_{max}	$\mathbb{R}_+^M$	The maximum power that should arrive at each point of the receiver surface.
Q_0	$\mathbb{R}_+^m$	The value for Q_{max} if $r = 0$.
Q_1	$\mathbb{R}_+^m$	The value for Q_{max} if $r = 1$.
Q_{base}	$\mathbb{R}_+^M$	A distribution that is used for example calculations.
f_0	$\mathbb{R}_+$	A pre-factor for the distribution to get to Q_1.
f_1	$\mathbb{R}_+$	A pre-factor for the distribution to get to Q_2.

References

[1] T. Ashley, E. Carrizosa, and E. Fernández-Cara. Optimisation of aiming strategies in solar power tower plants. *Energy*, 137:285–291, 2017. doi: 10.1016/j.energy.2017.06.163.

[2] G. Augsburger. *Thermo-economic optimisation of large solar tower power plants*. PhD thesis, STI, Lausanne, 2013.

[3] B. Belhomme, R. Pitz-Paal, and P. Schwarzbözl. Optimization of heliostat aim point selection for central receiver systems based on the ant colony optimization metaheuristic. *Journal of solar energy engineering*, 136(1):011005, 2014.

[4] S. M. Besarati, D. Y. Goswami, and E. K. Stefanakos. Optimal heliostat aiming strategy for uniform distribution of heat flux on the receiver of a solar power tower plant. *Energy Conversion and Management*, 84:234–243, 2014. doi: 10.1016/j.enconman.2014.04.030. URL `http://www.sciencedirect.com/science/article/pii/S0196890414003343`.

[5] F. J. Collado. One-point fitting of the flux density produced by a heliostat. *Solar energy*, 84(4):673–684, 2010.

[6] R. Flesch. Optimization of heliostat field - receiver interaction. Workshop, July 2017. URL `http://www.solar.rwth-aachen.de/publications/workshop/Presentation_Flesch.pdf`.

[7] F. J. García-Martín, M. Berenguel, A. Valverde, and E. F. Camacho. Heuristic knowledge-based heliostat field control for the optimization of the temperature distribution in a volumetric receiver. *Solar Energy*, 66(5):355–369, August 1999. doi: 10.1016/S0038-092X(99)00024-9.

[8] V. Grigoriev and C. Corsi. Unified algorithm of cone optics to compute solar flux on central receiver. In *AIP Conference Proceedings*, volume 1850, page 030021. AIP Publishing, 2017.

[9] A. Grobler. Aiming strategies for small central receiver systems. Master's thesis, Stellenbosch: Stellenbosch University, 2015. URL `http://scholar.sun.ac.za/bitstream/handle/10019.1/97051/grobler_aiming_2015.pdf?sequence=1`.

[10] A. Grobler and P. Gauché. A review of aiming strategies for central receivers. In *Second Southern African Solar Energy Conference*, 2014.

[11] B. D. Kelly. *Advanced Thermal Storage for Central Receivers with Supercritical Coolants*. Abengoa Solar Inc., Jun 2010. doi: 10.2172/981926. URL `http://www.osti.gov/scitech/servlets/purl/981926`.

[12] W. A. Landman and P. Gauché. Analysis of canting strategies using the hflcal model. *Proceedings of SASEC*, pages 1–7, 2014.

[13] W. A. Landman, A. Grobler, P. Gauché, and F. Dinter. Incidence angle effects on circular gaussian flux density distributions for heliostat imaging. *Solar Energy*, 126:156–167, 2016.

[14] A. Salomé, F. Chhel, G. Flamant, A. Ferrière, and F. Thiery. Control of the flux distribution on a solar tower receiver using an optimized aiming point strategy: Application to themis solar tower. *Solar Energy*, 94:352–366, 2013. doi: 10.1016/j.solener.2013.02.025. URL `http://www.sciencedirect.com/science/article/pii/S0038092X1300090X`.

[15] P. Schwarzbözl, R. Pitz-Paal, and M. Schmitz. Visual HFLCAL - a software tool for layout and optimisation of heliostat fields. In *SolarPACES 2009*, September 2009. URL `http://elib.dlr.de/60308/`.