UCL Université catholique de Louvain

École polytechnique de Louvain (EPL)

epl

Identification of the optimal glass type for buildings

Dissertation presented by
Simon-Pierre CORDONNIER

for obtaining the Master's degree in
Mathematical Engineering

Supervisor(s)
François GLINEUR, Tanguy TIMMERMANS

Reader(s)
Pierre-Antoine ABSIL

Academic year 2017-2018

Abstract

AGC Glass Europe has a tool named 'Building Energy Efficiency Calculator' (BEE) that permits to determine the impact of the glass on a building in term of thermal insulation, natural light and solar protection.
The goal of this master thesis is to develop an optimization algorithm in order to determine the optimal glass for a specific building.
First, an analysis of the glass domain and a study of a database of glasses were done to create a convex hull containing the entire glazing catalog. We have developed derivative-free black-box optimization algorithms as there is no analytic formulation of the objective function. The objective function value is a result from a simulation generated by EnergyPlus, a full-year energy simulation for buildings. The methods are deterministic algorithm such as Nelder-Mead, evolution strategies as Covariance Matrix Adaptation (CMA-ES), Particle Swarm Optimization (PSO), Differential Evolution (DE) and surrogate model as Regression model, Radial Basis Function Network (RBF), Kriging model. An improvement of algorithms has been made with the projection on the database, in particular on the convex hull, by quadratic programming. The study has two kinds of problem : either the case of one glass for the building or the case of multi-glasses.
The user of this tool can add some constraints to the problem : with constraints related to variables, with aesthetic constraints as the color of the glass and the price constraint. This tool will allow to give easier advises for employees for a type of glass for a building. Before, their only tool is to determine the glazing according to their experience and to verify, through a simulation 'EnergyPlus', the energetic impact of the building. At present, with the optimization tool, they will directly be able to ask for a specific problem which glass is optimal.

Acknowledgements

I would like to take this opportunity to thank all the people who have contributed to this master thesis.

I would first like to thank my thesis supervisor François Glineur for accepting the subject of this master thesis, steering me in the right direction with constructive comments and suggestions.

I would like to thank Tanguy Timmermans for giving me the subject and offering me an internship at AGC Glass Europe. Many thanks for his presence and relevant advice along the internship and the year.

I would also like to thank Pierre-Antoine Absil for accepting and taking the time to read this master thesis.

Finally, I would like to thank my parents, my friends and particularly my girlfriend for their support and encouragements during this final master year.

Contents

Table of Symbols

U	Thermal insulation
g	Solar factor
LT	Light Transmission
$T_{\min}$	minimal Temperature
$T_{\max}$	maximal Temperature
α	Parameter of scalarization
β	New value of the parameter of scalarization α
E_r	Energy of reference (double glazing)
IT_r	thermal discomfort of reference (double glazing)
$\mathbb{R}_{>0}$	Strictly positive real numbers
$\mathcal{N}(m, C)$	multivariate normal distribution with mean m and covariance matrix C
p_σ	Evolution path for standard deviation σ
p_C	Evolution path for the covariance matrix C
MLE	Log-likelihood function for a Gaussian process
$E[\cdot]$	Expectation value
L^*	Lightness
a^*	Red/green opponent colors
b^*	Yellow/blue opponent colors
γ	Coefficient for the impact of the price on the optimization

Table of Acronyms

AGC	Asahi Glass Company
BEE	Building Energy Efficiency Calculator
IDF	Intermediate Data Format
EPW	EnergyPlus weather data format
N-M	Nelder-Mead method
CMA-ES	Covariance Matrix Adaptation - Evolution Strategies
PSO	Particle Swarm Optimization
DE	Differential Evolution
LHS	Latin hypercube sample designs
Quadratic	Quadratic Regression model
RBF	Radial Basis Functions
KKT	Karush-Kuhn-Tucker
CIE	Commission Internationale de l'Éclairage

Chapter 1

Introduction

Providing the right window for the right application is the best way to provide customer satisfaction in terms of energy consumption and comfortable interior environment.
This is why AGC offers a wide range of glazing solutions to best suit the needs of the customer in terms of thermal insulation, available natural light and solar protection.

To help the customer for selecting the product made for him, AGC developed a web-based application named AGC Building Energy Efficiency calculator (BEE). It allows the customer to run comparative analysis between different glazing solutions and evaluate the benefits of each of them. This application uses the well-known open source software, EnergyPlus to run energy calculations. It needs to run the program to determine the performance of the selected products.

Suggesting the right window for the right application in order to optimize the energy consumed and the comfortable interior environment would be a really new application that will allow AGC to best provide the most suitable glazing for the consumer.

This purpose of this master thesis is to provide algorithms in order to determine the optimal glass for the building of the customer. The most suitable method for this problem are derivative-free black-box optimization algorithms. Throughout this thesis, we will expose different algorithms that have been tested and compared on different applications. We will also explain how those algorithms have been best adapted to the problem in order to have better results.

At first, we will focus on the case where there is only one type of glass for the whole building. It means that for each face of the building, we have a unique glass. Afterwards, we extend the problem to the case where each face of the building has its own glass. Thereby, the number of variables increases and makes the problem more difficult.

Finally, the customer can have additional specific requirements on the product that he wants. He/she can add constraints related to variables of the glass. The customer wants a specific color for its faces of his building. This person may also want to not spend too much money. He suggests a price to the software for which he is ready to spend. All his requests are formulated as constraints to the problem of optimization.

The thesis is organized as follows.
In chapter 2, we will introduce the main characteristics of the glass and how we can

differentiate the different existing types of glass. We will present the main problem and the impact of this master thesis for AGC Glass Europe. We will explain how we can formulate the problem as an optimization problem.
In chapter 3, we will discuss derivative-free black-box methods and their comparison.
In chapter 4, the application of the problem is discussed.
In chapter 5, the additional constraints are introduced and the ways to add them to the problems are exposed.
Finally, a conclusion about the techniques developed in this thesis and possible future work will be drawn.

Chapter 2

Context

2.1 Definition of the domain

AGC Glass Europe is an European company that produces and transforms flat glass for different fields such as building, industrial and automotive.

In our case, it is the field of the construction that is related to this master thesis. So the glass is used to protect from cold, wind, heat, noise, etc.

The goal of this chapter is to understand the characteristics of the glass and how we can differentiate the different existing types of glass.

The glass can be defined by its transmission of solar radiancy. The spectral radiancy is the radiance of the surface per unit wavelength. The transmission of solar radiancy through a glass can be different for each glass. Then a glass is differentiated on how the solar radiancy passes through. In the following figure 2.1, the solar radiancy is represented.

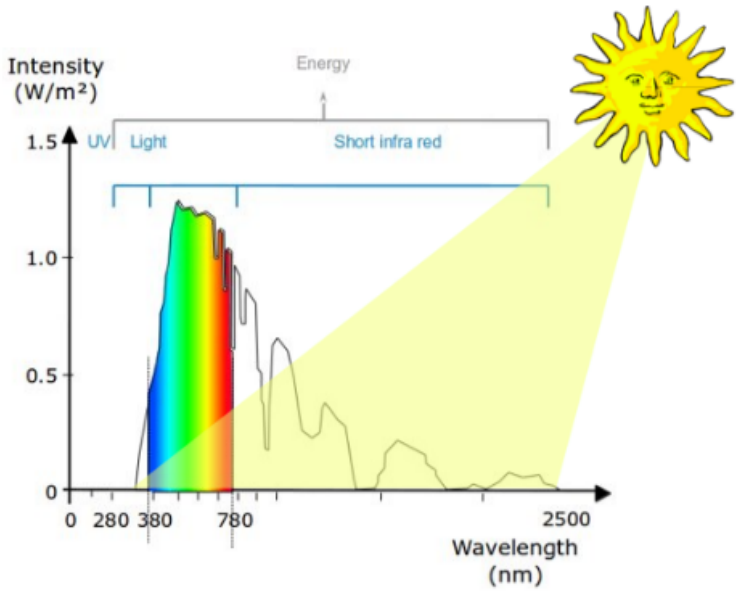


Figure 2.1: Solar radiancy - from [11]

But this way of representation of a glass is difficult to use because the glass is defined by the spectrum that composes it. For this, we have to know for each wavelength

the value of its intensity. The solar spectrum ranges from 280 to 2500 nm. In the following table, we report for each type of wave, the emitted energy by those waves. All following data come from [11].

Wave type	Wavelength (nm)	Energy (%)
UV	250 - 380	5
Light	380-780	43
Short infrared	780-2500	52

Using this to define a glass will lead to a lot of variables.
Fortunately, it is possible to redefine or summarize each glass according to 3 parameters : U thermal insulation, LT light transmission and g solar factor.

First, the thermal insulation U is the amount of heat transferred due to temperature difference. For instance, suppose it is cold outside while inside the building, heater works and the inside temperature is high. A transfer of heat between the building and outside will happen. The construction will lose heat through the window. Then, U is defined as this amount of lost heat. U-value is defined as $W/(m^2 \cdot K)$. In practice, U-value is between 0.4 and 6 $W/(m^2 \cdot K)$. The value of 6 is the case when it is only a single glazing. It is possible to reduce its value to 0.4 in the case of triple glazing with added coatings and gas as air, argon or krypton.

The light is emitted by the sun. When the light comes to the glass, one part is transmitted and another is reflected (cfr. figure 2.2).

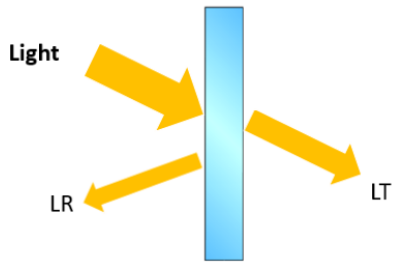


Figure 2.2: Light transmission - from [11]

The transmitted light is defined by LT.

$$LT = \frac{\text{transmitted light}}{\text{emitted light}}$$

Then, this value is between 0 and 1. Then, we already have the following two linear constraints:

$$0 \leq LT \leq 1$$

In the same way, the solar factor g is defined as the sun heat transmitted through the glass (cfr. figure 2.3).

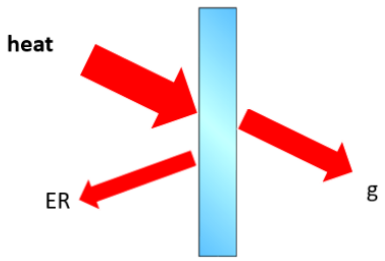


Figure 2.3: Solar factor - from [11]

Here, the solar factor is considered as the ratio between transmitted heat and emitted heat.

$$g = \frac{\text{transmitted heat}}{\text{emitted heat}}$$

This value is also between 0 and 1. We have another linear constraint between both parameters g and LT. It is the selectivity defined as the ratio between LT and g. It's a constraint that has been obtained by AGC.

$$\text{selectivity} = \frac{LT}{g} \leq 2.3$$

We have to know that the higher the selectivity is, the more performing is the glass in cutting more solar heat than visible light.

The goal is to define a convex domain for which it will be easy to find the optimal glass. It exists a database containing a set of glass products typically available on the market today. So we want a domain containing all those glasses. The size of this database is huge : the size furnished is around 506 000 glasses. In fact, many other glasses may exist. The objective of using a convex hull is to to take in account a very important constraint, the search of the optimal glass should stay inside a specific domain of solutions, the smallest domain containing glasses of the database.

In fact, those linear constraints are not sufficient and too large in order to define all available glasses in the database. The catalog doesn't cover the domain defined by those linear constraints as represented in the figure 2.4.

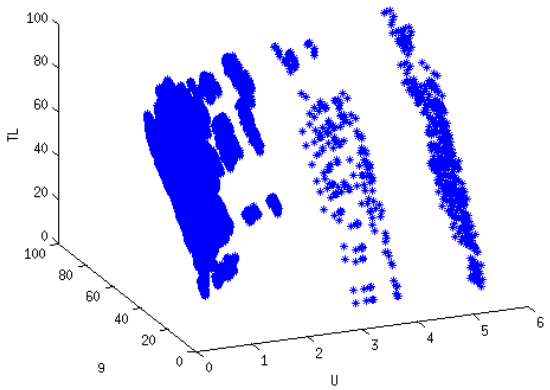


Figure 2.4: Database of glasses

The first conclusion that we can make on this domain is that it could be partitioned in small groups presenting a lot of holes.

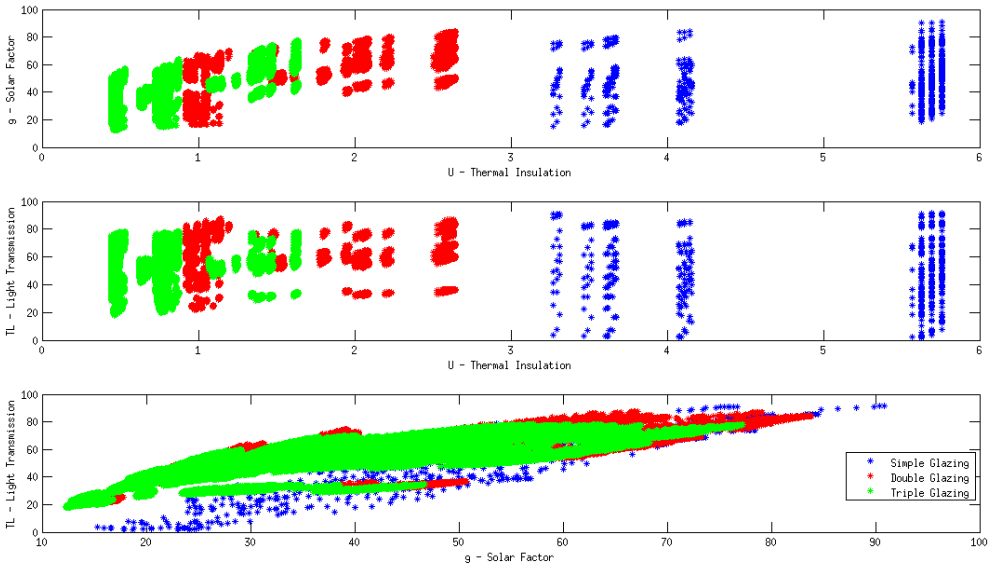


Figure 2.5: Database of glasses

The first solution is to divide this domain into 3 groups :

Group 1	$U \leq 3$
Group 2	$3 < U < 4.5$
Group 3	$U \geq 4.5$

Note that those 3 groups are not the 3 different types of glazing : single, double and triple glazing. The group 2 and 3 are only for single glazing and group 1 includes double and triple glazing.

The domain of each group can be represented as its smaller convex hull containing all the glass catalog. Then a MATLAB function `K = convhulln(X)` is used, returning the indices `K` of the points `X` that make up the facets of the convex hull of `X`.

A convex hull of a set of points is the smallest convex set that contains all the points.

Afterwards, a MATLAB function [22] has been written by Michael Kleder that converts a set of points to the set of inequality constraints which most tightly contain the points.

$$Ax \leq b$$

with

$$x = \begin{pmatrix} U \\ g \\ LT \end{pmatrix}, \; A \in \mathbb{R}^{n \times n} \text{ and } b \in \mathbb{R}^n$$

This function creates constraints to bound the convex hull of the given points. The algorithm used by M. Kleder is the following :

Algorithm 1 `vert2con`

Input:
V
Output:
A, b
$k \leftarrow$ `covhulln`
$c \leftarrow$ `mean`$(V(k,:))$ % Compute centroids of the convex hull
$V \leftarrow V -$ `repmat`$(c, [$`size`$(V,1)1])$; % Centralize data around origin
$N \leftarrow$ `size`$(k,1)$
$j \leftarrow 0$
for $i \in \{1,..,N\}$ **do**
  $F \leftarrow V(k(i,:),:)$ % Points `X` that make up the facet i of the convex hull
  if F is full rank **then**
    $j \leftarrow j+1$
    $A(j,:) \leftarrow F \backslash$`ones`$(3,1)$
  end if
end for
$b \leftarrow$ `ones`$($`size`$(A,1),1)$
$b \leftarrow b + A * c'$
Eliminate duplicate constraints

Then, after this algorithm, we obtain the tightly convex hull containing all the points. In the following figure 2.6, we show how the 3 different convex hulls, that we have obtained, look like.

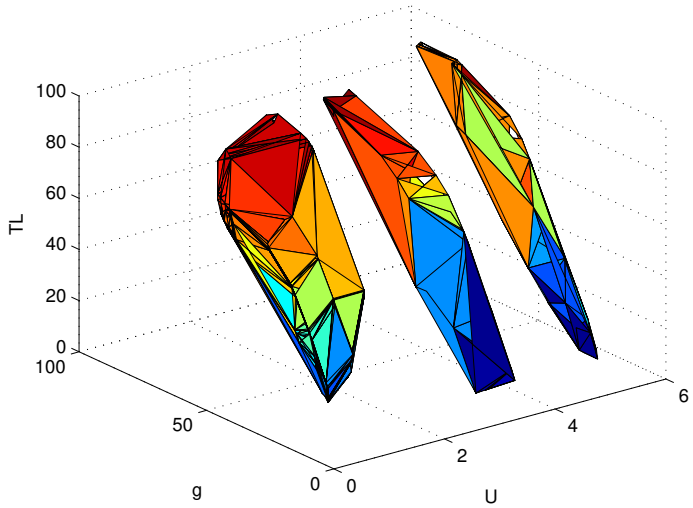


Figure 2.6: Convex hull containing all the AGC glasses

Note that the three convex hulls contain respectively 396, 71 and 52 facets.

2.2 Definition of the problem

First, we will introduce the main problem and the impact of this master thesis for AGC Glass Europe. This future tool will be very useful for the company because it will simplify the work of employees. Indeed, the tool will make it easier for employees to recommend a type of glass for a building. Before, their only tool is to determine the glazing according to their experience and to verify, through a simulation 'EnergyPlus', the energetic impact of the building. Next, with the optimization tool, they will directly be able to ask for a specific problem which glass is optimal.

AGC Glass Europe has a tool that permits to determine the impact of the glass in term of thermal insulation, natural light and solar protection. The tool runs comparative analysis between different glazing solutions and evaluate the advantages of each of them. This tool is named AGC Building Energy Efficiency Calculator (BEE) and uses an open-source simulation EnergyPlus.

EnergyPlus computes the energy to provide to a building thanks to an IDF file describing the physical characteristics of the building and to an EPW file that details the meteorologic conditions. In the AGC BEE Calculator, the user can define the parameters of the building and select the glazing he wants analyze. EnergyPlus will run a full-year simulation using weather data. At the end of simulation, the user will obtain the consumed energy and the impact on thermal comfort.

Different criteria are available in order to describe at best the building inside BEE and are transcribed inside an IDF file :

- Type of building: Choice between hospital, office, school and residential. This criterion defines the characteristics on the occupancy of the building. Indeed, the occupancy hours of a school are not the same than hours for a residential complex. A school is occupied all the day between 8 and 16 o'clock and is empty during the night. On the other hand, an apartment is empty the day and needs to be heated early the morning and the evening.

- Construction material: this option plays also a major role in the simulation. A light material as the wood will be less insulating than an heavy material as concrete.

- Size and orientation of building: Its length, width, number of floors and orientation has a direct impact on the consumption of building.

- Glazed proportion of each facade: More this surface is large, more the glazing has an impact on the energy computation of the simulation.

Note that the orientation is defined as following : 0° if the specific face 1 is facing North, 90° if face 1 is facing East, 180° if face 1 is facing South and 270° if face 1 is facing West.

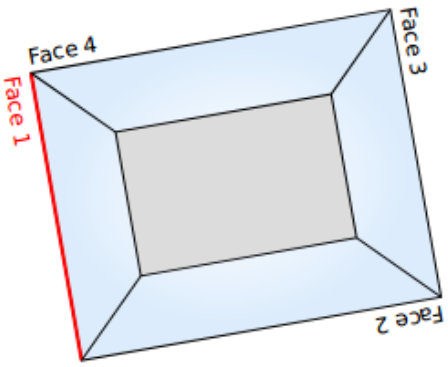


Figure 2.7: Example of rectangular building created by AGC BEE Calculator

In fact, this algorithm will be very useful for AGC because it will simplify the work of the employees. Indeed, the tool will permit easily to users to advise a type of glazing for a specific building. Thanks to the optimization tool, the user will be able to directly ask to the software for a specific problem what the optimal glass is. If the user is a beginner in the energetic impact of the glass on a given building, the tool will directly advice an 'optimal' glass. Otherwise, if the user is an expert in the energetic impact of the glass, this tool will be able perhaps to suggest a glass he would not have thought.

The subject of this master thesis is to establish an algorithm that permit to estimate the glazing for which the following objective function f is the lowest in terms of energy and comfort.

The objective function f of this optimization problem (2.1) is defined as the weighted sum between the total energy provided and the thermal discomfort in the building. The total energy is defined by the energy furnished by the electricity for the light, heating and cooling in one year.
In the other hand, the thermal comfort is computed when the temperature is considered to be in a user-defined range: in other words, the building is in a thermal comfort when the temperature of each side of the building is between $T_{\min}$ and $T_{\max}$. The objective function counts days when the building is not in thermal comfort. For example, the user can choose his own interval : $T_{\min} = 19^oC$ and $T_{\max} = 23^oC$.

$$f(x) = \alpha \text{ Energy} + (1 - \alpha) \text{ Thermal Discomfort} \tag{2.1}$$

Where α is a parameter that the user can choose. His choice depends on its preference to minimize the energy or thermal discomfort. Then, the parameter is introduced as $0 \leq \alpha \leq 1$:

- if $\alpha = 0$, thermal discomfort is fully minimized and energy has no impact.
- if $\alpha = 0.5$, thermal discomfort and energy are both minimized.
- if $\alpha = 1$, energy is fully minimized and the thermal discomfort has no impact on the optimization.

As the energy and thermal discomfort have not the same scale, we need to recompute another α in order to have energy and thermal discomfort in the same scale, called β.

In order to compute β, we use as reference, the case of the simple double glazing. For this reference case, the EnergyPlus simulation computes the values of energy E_r and thermal discomfort IT_r. We define $\beta_{0.5}$ as being β when energy and thermal discomfort are equally minimized ($\alpha = 0.5$). For this case, we are faced with this equation:

$$\beta_{0.5}\ E_r = (1 - \beta_{0.5})\ IT_r$$

Then, we solve this equation in order to determine the value of b.

$$\beta_{0.5} = \frac{E_r}{E_r + IT_r}$$

In order to estimate the value of β for any α, we use a second order polynomial interpolation going through three points $\{(0,0), (0.5, \frac{E_r}{E_r+IT_r}), (1,1)\}$. This is represented on the figure 2.8.

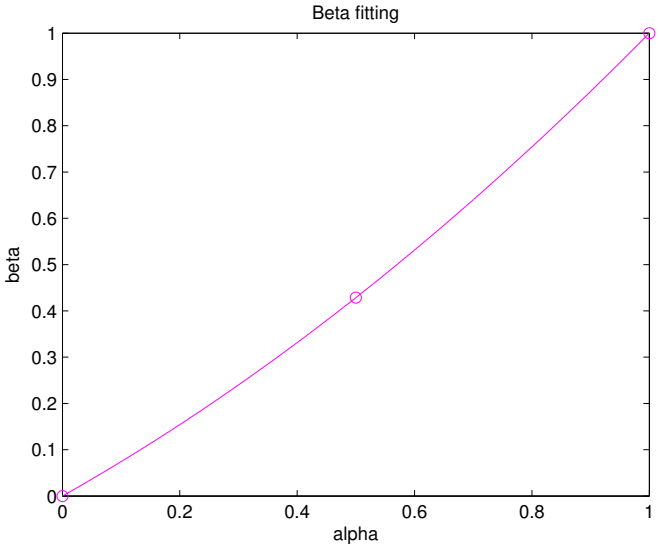


Figure 2.8: Example of second order polynomial interpolation in order to fit β

At present, we can easily determine the value of β for any α by replacing α in the second order polynomial interpolation. Then, we introduce a pre-processing for the optimization where we compute the value of β for the specific building and for the specific parameter α that the user has chosen. The objective function can be redefined according this new parameter β as following :

$$f(x) = \beta \text{ Energy} + (1 - \beta) \text{ Thermal Discomfort}$$

2.3 Hill Climbing algorithm

Before starting to analyze complex derivative-free black-box algorithms, a simple discrete hill climbing algorithm has been assessed. The method is an optimization technique that starts with an arbitrary solution of the problem and attempts to find incrementally a local optimal solution. At each iteration, it consists on finding the closest nodes from the current node. All these successors are compared and

the weakest node is chosen. If the algorithm doesn't find a weaker node, the local minimum is found.

Algorithm 2 Hill Climbing

```
currentNode is the double glazing : U = 2.631, g = 79.77 and LT = 82
while true do
    data = neighbors of currentNode
    newNode = argmin(f(data))
    if f(newNode) < f(CurrentNode) then
        currentNode = newNode
    else
        return currentNode
    end if
end while
```

In order to determine the neighbors of the current node, we use the k-nearest neighbors according to the euclidean distance. The value of k was fixed to 7 for the test but can be modified. This way, different results will be obtained.

Hill Climbing has been tested on a simple problem. It is a square building of 1000 m^2 with the following characteristics :

- Length : $40m$
- Width : $25m$
- One floor
- Weather : Brussels
- 60 % of glazing on each facade
- Material of construction : heavy
- $\alpha = 0.5$, then $\beta = 0.3574$

On this example, the algorithm doesn't seem to converge to a performing glass. Indeed, the method stops on this node : $U = 2.6037, g = 77.4416$ and $LT = 80.5366$. The value of the objective function is 66.9483. The value of U is too high for a case in Belgium, demonstrating that the simple Hill Climbing is not efficient enough. Indeed, in Belgium, the buildings need thick windows because the winters are cold. It is why we need to investigate more complex derivative-free black-box optimization algorithms. These methods will be introduced in the following chapter. We will see that the value of the objective function for this problem can attend less than 49.

Chapter 3

Derivative-free black box Optimization

There is no analytic formulation for the objective function because the value of the objective function is the value obtained due to a simulation.

$$\min_{U,g,LT} f(U, g, LT) : \mathbb{R}^3 \to \mathbb{R}$$

In the same way, it is not possible to compute the exact gradient of the objective function.

We note that we can compute its approximate gradient by finite differences. To compute its approximation is expensive, because we have to compute the values of the objective function at different points. If we use a second order finite difference, we have for each direction,

$$\begin{aligned}
\frac{\partial f}{\partial U}(U^*, g^*, LT^*) &= \frac{f(U^* + h, g^*, LT^*) - f(U^* - h, g^*, LT^*)}{2h} \\
\frac{\partial f}{\partial g}(U^*, g^*, LT^*) &= \frac{f(U^*, g^* + h, LT^*) - f(U^*, g^* - h, LT^*)}{2h} \\
\frac{\partial f}{\partial LT}(U^*, g^*, LT^*) &= \frac{f(U^*, g^*, LT^* + h) - f(U^*, g^*, LT^* - h)}{2h}
\end{aligned}$$

Then, if the problem has 3 different variables, we have to compute the value of the objective function at 6 different points in order to compute the gradient at (U^*, g^*, LT^*). As in our case the value of an objective function is the value of a simulation, we have to launch 6 simulations.

So, the first stage is the implementation of derivative-free black box algorithms in order to solve the problem. Afterwards, we improve and test them in order to choose the most appropriate algorithm for this type of problem because there is no theory to guarantee performance. We sort those algorithms in some classes as deterministic, evolution strategies, surrogate models or trust region model. Note that at this stage, we don't take into account the constraints. We develop each algorithm with its advantages and drawbacks. In the next chapter, we present an improvement that permits to consider the convex hull.

3.1 Deterministic Algorithm

Deterministic algorithms are gradient-free methods to solve problems that are difficult to solve using gradient-based methods. Deterministic methods are heuristic methods that search for the local optimum.

3.1.1 Nelder-Mead method

Nelder-Mead, also called simplex method, performs a search on an n-dimensional space, in our case it is in a 3-d space as we have 3 variables U, g and LT. It is an heuristic method that computes the local optimum on a non-linear simplex, we don't have to mix with the simplex used in linear optimization that is completely different.

Algorithm 3 `Nelder-Mead`

Input:
α, γ, ρ and σ
Output:
x_1
We choose $n+1$ initial points (n is the dimension of variables) and we compute the values of the function at this points and sort according to the function value
$N \leftarrow$ number of iterations
for $i \in \{1, .., N\}$ **do**
  Compute x_0 the mid-point of the $n+1$ points.
  Compute the reflection-expansion-contraction points :
  $x_r = x_0 + \alpha(x_0 - x_{n+1})$
  $x_e = x_0 + \gamma(x_r - x_0)$
  $x_c = x_0 + \rho(x_{n+1} - x_0)$
  Compute the function values of points x_r, x_0, x_e, x_c
  if $f(x_1) \leq f(x_r) \leq f(x_n)$ **then**
    if $f(x_e) < f(x_1)$ **then**
      Expansion, $x_{n+1} \leftarrow x_e$
    else
      Reflection, $x_{n+1} \leftarrow x_r$
    end if
  else if $f(x_r) < f(x_{n+1})$ and $f(x_c) \leq f(x_r)$ **then**
    Contraction outside, $x_{n+1} \leftarrow x_c$
  else if $f(x_c) < f(x_{n+1})$ **then**
    Contraction inside, $x_{n+1} \leftarrow x_c$
  else
    Shrink the domain, $\forall i \in \{1, ...n+1\}$, $x_i = x_1 + \sigma(x_i - x_1)$
  end if
end for

A simplex is a structure composed by $n+1$ points that are not in the same hyperplane. In our case we are in a 3-dimensional space, then we face a simplex that is equal to a tetrahedron (cfr. figure 3.1).

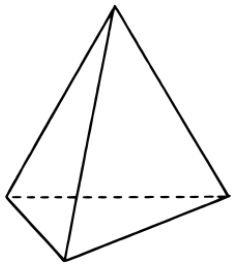


Figure 3.1: 3-dimensional tetrahedron

The initialization of Nelder-Mead begins with an initial simplex. At each iteration, the simplex is modified using four operations : reflection, expansion, outside contraction, inside contraction and shrinking. All these operations are briefly detailed in the algorithm 3. The output of this algorithm is the optimum x^* defined as x_1. the best vertex of the last iterated simplex. In order to take advantage of parallel computing of objective function values. We may compute function values for the points x_r, x_0, x_e, x_c at the same time.

In the literature [4], the main strength of this method is that it requires no derivative to be computed and the objective function has not to be smooth. The weakness is that it is not very efficient for the cases when we have a lot of variables. Fortunately, in our case, we have only 3 variables, U, g and LT for the case of an unique type of glass for buildings.

3.2 Evolution Strategies

Evolution Strategies or Genetic algorithms are methods inspired by the process of natural evolution of organisms. They consist in initializing a population of candidate solutions. For each element of the sample, we have its objective function value. At each iteration, genetic algorithms are based on three essential components : survival of the sample (Selection), reproduction of this sample (Crossover) and finally, recombination of this sample (Mutation).

At this phase, we don't take into account the constraint, so we have the following unconstrained problem.

$$\min_{U,g,LT} f(U, g, LT)$$

As we have a population, it is adapted for parallel computing of objective function values of the sample. The main advantage of this kind of algorithm is the use of population. Then, it is possible to explore simultaneously several regions of the domain in order to find good local minimum. This way, it is more probable to find the global optimum and to not be stuck in a local optimum

3.2.1 Covariance Matrix Adaptation (CMA-ES)

Covariance Matrix Adaptation (CMA-ES) is a well-known evolution strategy algorithm. It uses concepts as eigendecomposition of a positive definite matrix, multivariate normal distribution, covariance matrices.

The eigendecomposition of a matrix C must be positive semi-definite and is defined as

$$C = BD^2B^T$$

where

- B is an orthogonal matrix ($BB^T = B^TB = I$) such that the columns of B forms an orthogonal basis of eigenvectors of C
- $D^2 = \text{diag}(\lambda_1^2, \lambda_2^2, ..., \lambda_n^2)$ is a diagonal matrix with eigenvalues of C as diagonal elements, $\lambda_i > 0$

A multivariate normal distribution $\mathcal{N}(m, C)$ has an unimodal density where the top of the distribution is the mean $m \in \mathbb{R}^n$. The covariance matrix $C \in \mathbb{R}^{n \times n}$ is symmetric and positive definite. The covariance matrix can be represented as an ellipsoid shown in figure 3.2. It corresponds to the case where the number of dimensions is 2.

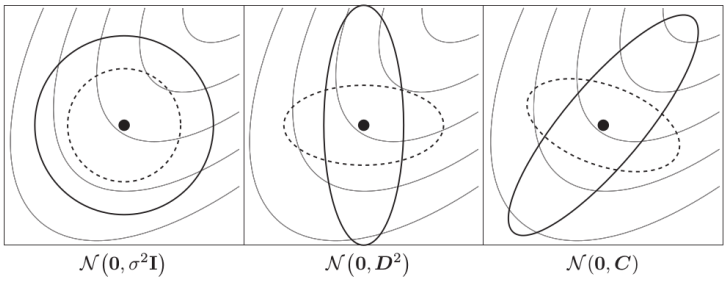


Figure 3.2: Ellipsoids, three case : $\sigma \in \mathbb{R}_{>0}$ (left), D is a diagonal matrix (middle) and C is a positive definite full covariance matrix (right). Thin lines represent possible objective function contour lines. Image from Nikolaus Hansen (Inria, France) [19]

The principal axes of the ellipsoid correspond to the eigenvectors of C. Also note that the length of the axes depends on the eigenvalues. As the eigendecomposition is denoted as $C = BD^2B^T$,

- If $D = \sigma I$ where $\sigma \in \mathbb{R}_{>0}$ and I the identity matrix, the ellipsoid corresponds to a perfect circle with radius σ^2. (left figure 3.2)
- If $B = I$, then $C = D^2$, a diagonal matrix, an ellipse is obtained with length of the first axis σ_1^2 and second axis σ_2^2. (middle figure 3.2)
- If B is not the identity, the distribution of $\mathcal{N}(m, C)$ corresponds to the right figure 3.2.

Note that the multivariate normal distribution $\mathcal{N}(m, C)$ can be written as the following

$$\mathcal{N}(m, C) \sim m + B\mathcal{N}(0, D)$$

where $C = BD^2B^T$.

Now, we will develop how the algorithm uses those concepts in order to find the optimum.

The initialization of a population of new search points by a multivariate normal distribution for the generation $g+1$ is :

$$x_k^{(g+1)} \sim m^{(g)} + \sigma^{(g)}\mathcal{N}(0, C^{(g)}) \text{ for } k = 1, ..., \lambda$$

where $C^{(g)}$ is the covariance matrix at generation g, $x_k^{(g+1)}$ is the k-th search point from generation $g+1$, $m^{(g)}$ and $\sigma^{(g)}$ are respectively the mean value and the standard deviation of the population at generation g and λ is the population size.

The algorithm computes in parallel the function values for the population of new search points and sorts the search points according to function values.

It updates the mean $m^{(g+1)}$ of the search distribution which is a weighted average of μ selected points from the sample at generation $g+1$.

$$m^{(g+1)} = m^{(g)} + \sum_{i+1}^{\mu} w_i(x_{i:\lambda}^{(g+1)} - m^{(g)})$$

where $\mu \leq \lambda$. Note that $\mu_{\text{eff}} = \frac{\left(\sum_{i=1}^{\mu} |w_i|\right)^2}{\sum_{i=1}^{\mu} w_i^2}$.

It updates the evolution paths p_σ and p_c for generation $g+1$.

$$p_\sigma^{(g+1)} = (1-c_\sigma)p_\sigma^{(g)} + \sqrt{c_\sigma(2-c_\sigma)\mu_{\text{eff}}}\left(C^{(g)}\right)^{-1/2}\left(\frac{m^{(g+1)} - m^{(g)}}{\sigma^{(g)}}\right)$$

$$p_c^{(g+1)} = (1-c_c)p_c^{(g)} + \sqrt{c_c(2-c_c)\mu_{\text{eff}}}\left(\frac{m^{(g+1)} - m^{(g)}}{\sigma^{(g)}}\right)$$

where $1/c_c$ and $1/c_\sigma$ are backward time horizons of respectively evolution paths p_c and p_σ. A time horizon is between $\sqrt{n}$ and n the dimension of variables.

It updates the Covariance Matrix and σ.

$$C^{(g+1)} = (1 - c_1 - c_\mu \sum_{i=1}^{\lambda} w_i)c^{(g)} + c_1(p_c^{(g+1)})^T + c_\mu \sum_{i=1}^{\lambda} w_i y_{i:\lambda}^{(g+1)}(y_{i:\lambda}^{(g+1)})^T$$

$$\sigma^{(g+1)} = \sigma^{(g)} \exp\left(\frac{c_\sigma}{d_\sigma}\left(\frac{\|p_\sigma^{(g+1)}\|}{E\|\mathcal{N}(0,I)\|} - 1\right)\right)$$

where c_1 and c_μ are learning rates, $y_{i:\lambda}^{(g+1)} = (x_{i:\lambda}^{(g+1)} - m^{(g)})/\sigma^{(g)}$ and d_σ is a damping parameter.

In order to have more information about how updating formulas of evolution paths, σ and covariance matrix are obtained, see those following articles [19] [5] [29] [28]. The following algorithm is a summary of how covariance matrix adaptation is implemented.

Algorithm 4 `CMA-ES`

Input:
MaxIter
Output:
$x*$
Setting of different parameters as λ, σ, w_i, μ, c_1, c_μ and C
for $g \in \{1, .., \text{MaxIter}\}$ **do**
Generate population : $x_k^{(g+1)} \leftarrow m^{(g)} + \sigma^{(g)}\mathcal{N}(0, C^{(g)})$ for $k = 1, ..., \lambda$ (Selection)
Compute function values of this sample
Recombination of the mean value m (Mutation and Crossover)
Updating of evolution paths, σ and covariance matrix C
end for
$x^* \leftarrow$ best obtained point

3.2.2 Particle Swarm Optimization (PSO)

Particle Swarm Optimization has been modelled as a biological and social behaviour. It is inspired of the movement of birds in the sky. It consists at each iteration to evaluate the function values of your particles, to compare the function values to the previous best and to the best of the neighbors and to modify the particles in order to imitate the best particles. So, we consider that each point in the domain has a position and a velocity. At each iteration, we update velocities and positions according to other particles of the sample. For a particle i in the sample,

$$v_i \leftarrow \omega v_i + c_1 r_1 (x_{\text{current best}} - x_i) + c_2 r_2 (x_{\text{global best}} - x_i)$$
$$x_i \leftarrow x_i + v_i$$

where ω, c_1 and c_2 are constants and r_1 and r_2 are random. Then, at each iteration, we update $x_{\text{current best}}$, the best particle of the current sample and $x_{\text{global best}}$, the best particle since the first iteration according to its objective function values.

Algorithm 5 `PSO`

Input:
MaxIter, $n, c_1, c_2, \omega_{\max}, \omega_{\min}$
Output:
$x*$
Initial position and velocity of a population : x_j and v_j for $j \in \{0, ..., n\}$
for $i \in \{1, .., \text{MaxIter}\}$ **do**
Updating of $\omega = \omega_{\max} - \frac{(\omega_{\max} - \omega_{\min})i}{\text{MaxIter}}$
Updating of the velocity $v_{i+1} = \omega v_i + c_1 r_1 (x_{\text{current best}} - x_i) + c_2 r_2 (x_{\text{global best}} - x_i)$
Updating of the position $x_{i+1} = x_i + v_i$
Updating $x_{\text{current best}}$ and $x_{\text{global best}}$
end for
$x^* \leftarrow$ best obtained point

In the same way as CMA-ES, PSO takes advantages of samples to compute the objective values of different points in parallel. More information about this algorithm

can be found in articles [8], [21], [27].

3.2.3 Differential Evolution (DE)

Differential Evolution consists in picking at each iteration three particles randomly. Computing the new position of all particles depends on this 3 selected particles. The algorithm 6 explains in details each iteration and how the population evolves. More details is available in this following article [15].

Algorithm 6 DE

Input:
MaxIter, n (number of particles)
Output:
$x*$
$c_1 \in [0, 1]$ (advice $c_1 \leftarrow 0.9$)
$F \in [0, 2]$
for $k \in \{1, .., \text{MaxIter}\}$ **do**
  for $i \in \{1, .., n\}$ **do**
    Pick three particles randomly in the population : p_1, p_2 and p_3
    Pick R a random index between 1 and N
    Where N is the dimension of the variables
    for j each variable of the particle i **do**
      Pick r_j an uniform random number
      if $r_j < c_1$ or $j == R$ **then**
        $y_j = (p_1)_j + F((p_2)_j - (p_3)_j)$
      end if
    end for
    if $f(y) < f$(particle i) **then**
      Replace the particle i by the new particle y
    end if
  end for
end for
$x^* \leftarrow$ best obtained point

3.3 Surrogate Model

Surrogate models consist in approximating the black-box function by an algebraic functions. It is easy to compute the value of this approximated function. Hence, we optimize this algebraic function in order to obtain the best point. More information about surrogate models is available in following references [7], [23], [17], [34], [18], [31], [35], [24], [9].

The main steps of the surrogate models are :

Algorithm 7 Surrogate Models

Initializing sample datas and build an initial surrogate model
Training the model in order to have a more accurate model :
Test a new sample
if Good convergence (|approximating function - real function| ≈ 0) or maximum number of iterations attended **then**
    Stop the training
else
    Updating the sample datas
    Building a new surrogate model
end if

To determine the initial sampling, we use a Latin hypercube sample designs (LHS). Latin hypercube sample designs consists in creating n points : we divide each axis into n equals intervals and we compute a matrix where there is only one sample in each rows and each columns. (function : `lhsdesign`)

In order to cover properly the domain, we use the criterion `maximin` : $\max_{|S|=n} \min_{x,y \in S} d(x,y)$. In the following figure 3.3 and 3.4, we show this criterion can improve the partition of the initial sample.

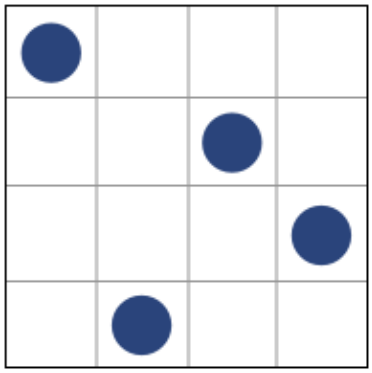


Figure 3.3: Classic LHS

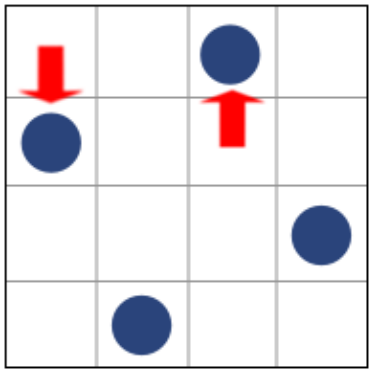


Figure 3.4: Improved LHS

3.3.1 Quadratic regression

Quadratic regression creates a second order model that approximates the objective function. This approximation is found by regression. In this case, we suppose that the objective function is convex and has only one optimum.

We define the predictor $\hat{y}$ of the true objective function y as a quadratic approximation such that :

$$\hat{y}(x) = \beta_0 + \sum_{i=1}^{m} \beta_i x_i + \sum_{i=1}^{m} \sum_{j \geq i}^{m} \beta_{i,j} x_i x_j$$

The coefficients β_0, β_i and $\beta_{i,j}$ are unknows. Then, the least square estimators of β is,

$$\beta = (U^T U)^{-1} U^T y_s$$

where

$$U = \begin{bmatrix} 1 & x_1^{(1)} & \cdots & x_m^{(1)} & x_1^{(1)}x_2^{(1)} & \cdots & x_{m-1}^{(1)}x_m^{(1)} & (x_1^{(1)})^2 & \cdots & (x_m^{(1)})^2 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(n)} & \cdots & x_m^{(n)} & x_1^{(n)}x_2^{(n)} & \cdots & x_{m-1}^{(n)}x_m^{(n)} & (x_1^{(n)})^2 & \cdots & (x_m^{(n)})^2 \end{bmatrix}$$

In order to train the model, we compute the minimum value of the approximated function by Evolution Optimization Algorithm several times such that the search region is modified.

$$\mathcal{D}_n := \mathcal{D} \bigcup_{i=1}^{n} B(x_i, r_i)$$

where the ball $B(x, r) = [x - r, x + r]$ and x_i are the previous minimum points. In the case of testing, the computation of the minimum will be launch 7 times, the computer for the tests has 8 thread processes, the computation of objective function values of those minima could be launched in parallel.

At the end of the iteration, new points are added in the database and the next iteration will compute the new model.

3.3.2 RBF network

A Radial Basis Functions network will approximate the objective function by interpolation such that for a set of points $\{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$

$$\hat{y}(x) = \sum_{i=1}^{n} \lambda_i \phi(\|x - x_i\|) + p(x)$$

where p is a first degree polynomial $p(x) = b^T x + a$, ϕ is the radial cubic function ($\phi(x) = x^3$) and λ are unknown parameters.
The interpolate condition system is for $y = (y_1, ..., y_n)$ and $\Phi = [\phi(\|x_i - x_j\|)]_{1 \leq i,j \leq n}$

$$\Phi\lambda + p = y$$

To determine the parameters $\lambda_1, \lambda_2, ..., \lambda_n$, a, b_1, ..., b_d, we solve the following equation system,

$$\begin{bmatrix} \Phi & P \\ P^T & 0 \end{bmatrix} \begin{bmatrix} \lambda \\ c \end{bmatrix} = \begin{bmatrix} y \\ 0 \end{bmatrix}$$

Where

$$P = \begin{bmatrix} x_1^T & 1 \\ \cdots & \cdots \\ x_n^T & 1 \end{bmatrix}, \; c = \begin{bmatrix} b_1 \\ \vdots \\ b_d \\ a \end{bmatrix}$$

The model learns as this following way : we compute the minimum value of the approximated function by Evolution Optimization Algorithm several times such that the search region is modified

$$\mathcal{D}_n := \mathcal{D} \bigcup_{i=1}^{n} B(x_i, r_i)$$

where the ball $B(x, r) = [x - r, x + r]$ and x_i are the previous minimum points. In our case, for the same reason as Quadratic regression, the number of times that the minimum is computed is 7 times.

In the literature [24], another way to learn the model is to adapt the "bumpiness of functions" but this part wasn't implemented.

3.3.3 Kriging Model

The Kriging model is also an interpolation method that approximate the objective function. In theory, the model will take advantage of statistics in order to approximate better.

The Kriging predictor $\hat{y}(x)$ for a set of points $\{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$ is given by

$$\hat{y}(x) = \beta_0 + r^T(x)R^{-1}(y - \beta_0 1)$$

Where,

$$\begin{aligned} \beta_0 &= (1^T R^{-1} 1)^{-1} 1^T R^{-1} y \\ R &= [Corr(x_i, x_j)]_{1 \leq i,j \leq n} \\ r(x) &= [Corr(x_i, x)]_{1 \leq i \leq n} \\ Corr(x, x') &= \exp\left[-\sum_{k=1}^{d} \theta_k |x_k - x'_k|^{p_k} \right] \text{ with } 0 \leq p_k \leq 2 \end{aligned}$$

We have to determine the parameters θ_k and p_k of the correlation function for each k between 1 and d,

- $\theta_k > 0$ determines how the correlation drops off in the k-th direction
- $0 < p_k \leq 2$ determines the smoothness of the function

To determine estimators of the parameters, this following log-likelihood function for a Gaussian process has to be maximal,

$$MLE(\theta, p) = -n \ln(\sigma^2(\theta)) - ln|R(\theta)|$$

where

$$\sigma^2(\theta, p) = \frac{1}{n}(y - \beta_0 1)^T R^{-1}(y - \beta_0 1)$$

To maximize, an evolution optimization strategies is used.
The improvement function is defined at point x such that $I(x) = \max[y_{\min} - Y(x), 0]$ where $y_{\min}$ is the current best value.
The expected improvement is defined as

$$\begin{aligned} E[I(x)] &:= E[\max[y_{\min} - Y(x), 0]] \\ &= [y_{\min} - \hat{y}(x)]\Phi\left(\frac{y_{\min} - \hat{y}(x)}{\hat{s}(x)}\right) + \hat{\sigma}(x)\phi\left(\frac{y_{\min} - \hat{y}(x)}{\hat{s}(x)}\right) \end{aligned}$$

where Φ and ϕ are respectively the normal cumulative distribution and probability density functions.

The expected improvement calls the standard deviation function $\hat{s}(x)$ defined by,

$$\hat{s}^2(x) = \sigma^2 \cdot [1.0 - r^T(x)R^{-1}r + (r^T(x)R^{-1}1 - 1)^2/(1^T R^{-1}1)]$$

and $\sigma^2 = \frac{1}{n}(y - \beta_0 1)^T R^{-1}(y - \beta_0 1)$
To improve the kriging model, at each model, the maximum of the expected improvement is computed by evolution optimization strategies. This point is added to the data sample and afterwards a new model is recomputed.

3.3.4 Trust Region Model based

Trust region Model based method is a method that interpolates a second order model in a local region. At each iteration, it updates the search region in order that the new region is made near the minimum of the interpolation.

A quadratic model m is derived from a region $\mathcal{B}_k$ around the point x_k.

$$\mathcal{B}_k = \{x \in \Omega : \|x - x_k\| \leq \Delta_k\}$$

The next point is defined as the minimum value of the quadratic model in $\mathcal{B}_k$.

$$x_{k+1} = \text{argmin}\{m(x_k + s) : x_k + s \in \mathcal{B}_k\}$$

(x_k, Δ_k) is updated at each iteration based on ρ_k that indicates how well the model predicts the function decrease.

$$\rho_k = \frac{f(x_k) - f(x_k + s)}{m(x_k) - m(x_k + s_k)}$$

The quadratic model is determined by a quadratic interpolation.

The quadratic model needs at least $\frac{(n+1)(n+2)}{2}$ points to obtain the exact quadratic model. (n is the dimension)

$$\begin{bmatrix} 1 & x_1 & \cdots & x_n & x_1x_2 & \cdots & x_{n-1}x_n & (x_1)^2 & \cdots & (x_m)^2 \end{bmatrix}$$

In the literature [40] [37], Michael J. D. Powell proposed that keeping only $2 \cdot n + 1$ points is relevant in terms of efficiency and number of function evaluations. The terms $x_i x_j$ are then deleted for all $1 \leq i, j \leq n$ and $i \neq j$.

Chapter 4

Numerical experiments and analysis

4.1 Case of one glass for buildings

In this section, we will compare the results of each algorithm. At the end of this analysis, we will determine which algorithm is the most adapted to this following problem. We face the case where we have to choose one type of glazing for all the facades of the building. At the end of the simulation, the software proposes to the user the best glazing. It is the first step before the next section where we focus on the case where the algorithm proposes 4 glasses for the building, one for each facade.

4.1.1 Comparison of derivative-free black-box algorithms

In order to test the derivative-free black box algorithm, tests with suitable parameters are conducted. Parameters have been chosen for each algorithm. Then for this case, algorithms are compared according to a small classical problem. It is a one-floor square building of 1000 m^2 with all those following characteristics, the same as hill climbing:

Length (m)	Width (m)	# floors	Weather	% Glazing on each facade	Material construction
40	25	1	Brussels	60	Heavy

This analyze focuses only on the convex hull with double and triple glazing.

A way to take into account the constraints is needed, hence a penalty function inside the objective function is added. So, the idea is to penalize the value of the objective function when the algorithm finds a point outside the domain. For the constrained problem,

$$\min f(x) \text{ subject to } Ax - b \leq 0$$

where $A \in \mathbb{R}^{m \times d}$ and $b \in \mathbb{R}^{m \times 1}$, we can solve the similar problem

$$\min \hat{f}(x) = f(x) + \sigma \sum_{i=1}^{m} g(a_i^T x - b_i)$$

Where $g(y) = (\max(0, y))^2$.

Note that σ is the penalty coefficient. In the first phase of test, we used $\sigma = 10$. This coefficient penalizes a lot when the algorithm goes out the domain.

For deterministic algorithms such as Nelder-Mead, the number of iterations is set to 30. For this specific algorithm, an initial point has to be set, in the center of the specific convex hull of research : $U = 0.6W/(m^2 \cdot K)$, $g = 35\%$ and $LT = 60\%$.

For evolution strategies, the number of iterations is 30 for CMA-ES and PSO because their population for the 3 dimensional case are respectively 7 and 10. In the other way, as different evolution needs a population size of 15; its number of iterations is only 10. This size of population is computed by the algorithm. This number of iterations permits to keep the same total of objective function evaluations.

For surrogate models, initial parameters for the optimization are different for each algorithm. For the quadratic regression and RBF model, the optimization simulation is launched with the following parameters : number of iterations = 15 and initial sampling = 20. In the other way, Kriging model has been set with more iterations : 40. Finally, trust-region model has those following parameters : 30 iterations, initial point in the center of the specific convex hull of research : $U = 0.6W/(m^2 \cdot K)$, $g = 35\%$ and $LT = 60\%$. We set the number of needed points for obtaining each sub model at $2 \times \text{dimension} + 1 = 7$ (as we search in a 3 dimensional space).

For most algorithms (N-M, CMA-ES, PSO, DE, Quadratic Regression), we converge to the following triplet :

U	g	TL	Thermal Incomfort	Energy	Function Value
0.465	14.006	28.238	60.413	42.517	48.9878

In the other way, RBF and Trust Region methods obtain unfortunately to another point with a higher objective function value. So, it may possible that the algorithms converge badly only on this example but as those algorithms fail on a simple problem, we can conclude that both algorithms can be abandoned for further analyses. For Kriging, the minimum of its final model doesn't converge to the same point as other algorithms. So, either the algorithm has to be improved, or the algorithm is not appropriated for this problem. So, it is preferable to abandon further analyses on this model.

In the following table, we have the CPU time used by each algorithm. The number of function evaluation is not the same but it is the time before that the algorithm finds a solution.

Methods	time (s)
Nelder-Mead	1542.6
CMA-ES	1672.3
PSO	2449.1
DE	2549.2
Quadratic	1183.6
RBF Model	1242.8
Kriging Model	1437.5
Trust-Region	2649.3

We observe that the time performed for each algorithm is not the same. It is interesting to remark that the time of optimization is smaller for surrogate models. For evolution strategies, the execution time is better for CMA-ES than PSO or DE with a difference of at least 800 seconds. Nelder-Mead is also fast in comparison of evolution strategies.

Now, all algorithms, with penalty when the algorithm goes out the domain, are compared on their convergence on a graph (figure 4.1). In this case, the value of α is 0.5 and then β is equal to 0.3574.

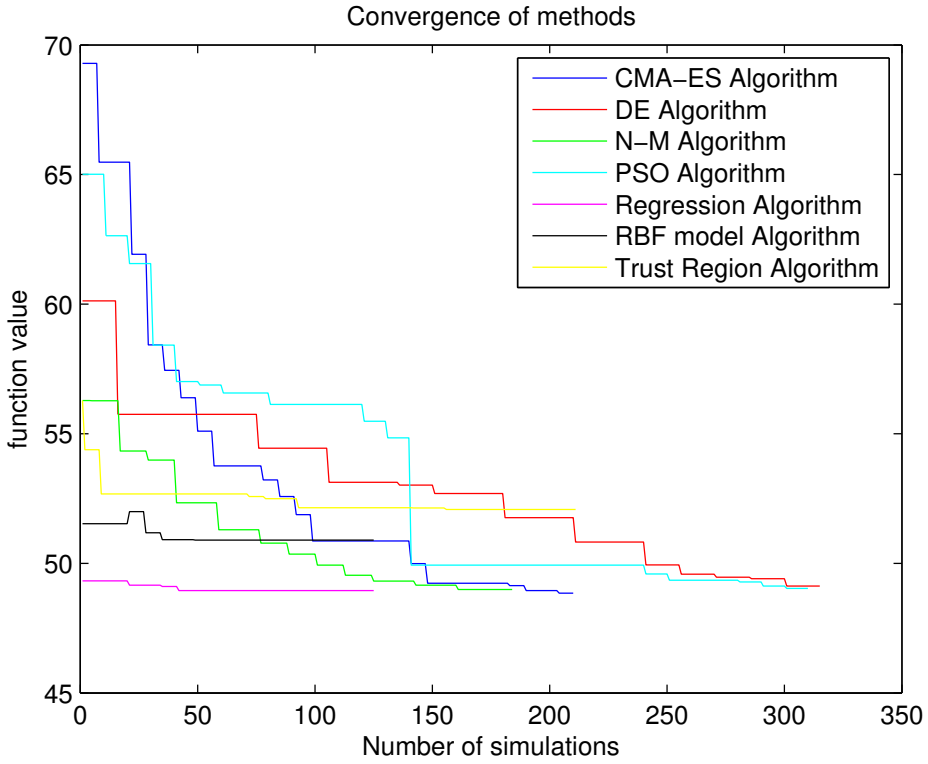


Figure 4.1: Convergence of algorithms

We can conclude at the end of this first analysis that all algorithms don't converge at the same speed. As observed above, Trust Region and RBF models seems to converge to a common point, not the optimal point (the assumption was confirmed when we look at the converged point). The 3 faster algorithms for this problem are Nelder-Mead, CMA-ES and Regression algorithm. And they also converge to the better optimal point. So, the rest of the analysis will be continued with only those 3 algorithms. It is actually a good sample of all algorithms tested before. Indeed, Nelder-Mead (N-M) represents deterministic algorithms. CMA-ES is an algorithm like evolution strategies. Finally, Regresion model is the most basic surrogate model. Note that hill climbing is not performing (its objection function value is more than 65 for this problem), then it is very interesting to have studied derivative-free black-box optimization algorithms.

4.1.2 Improvements of algorithms : Projection on database

As explained in the chapter 2, to use a convex hull allows to to take in account a very important constraint, the search of the optimal glass should stay inside a specific domain of solutions, the smallest domain containing glasses of the database. As we are in a case of constrained algorithms, a first way was to add a penalty function inside the objective function. The main drawback of the penalty function is that the algorithm allows searching outside of the domain. It is inefficient to lose precious time to compute useless points that won't be acceptable being out of the convex hull. So we need to impose the algorithm to stay inside the convex hull. So, the main improvement of previous algorithms is to project at each iteration the variable on the convex hull. It is useful in order that the variable stays inside the convex hull when the optimization algorithm is executed. This projection can be written as the following optimization problem :

$$\min_{U,g,TL} \|(\bar{U}, \bar{g}, \bar{TL}) - (U, g, TL)\|$$

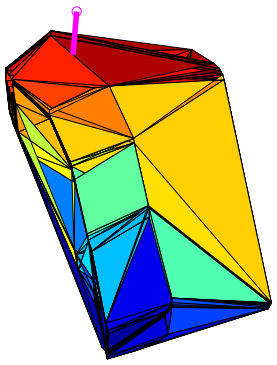


Figure 4.2: Projection on the convex hull. The point in magenta is outside the convex hull; the point is projected on the convex hull. The new value of the point is that one obtained after projection.

The objective function of this sub-optimization problem can be rewritten as the following if we set $x = [U, g, TL]^T$.

$$\begin{aligned} \|x - \bar{x}\|^2 &= (x - \bar{x})^T I (x - \bar{x}) \\ &= x^T I x - 2\bar{x}^T x + \bar{x}^T \bar{x} \end{aligned}$$

Then, we have

$$\min_x x^T I x - 2\bar{x}^T x + \bar{x}^T \bar{x}$$

We face to a quadratic programming problem. It is easy to compute its solution through MATLAB thanks to the function `quadprog`. Then, we use this function with $H = I$ and $f = -\bar{x}$.

With this improvement, we are sure that the algorithm take into account the constraints and that the search stay well inside the convex hull. So for each algorithm, we modify its implementation such that, for each search, if the points go outside of the convex hull, we project the points on the convex hull.

We will observe the influence of the projection on the algorithms. For those tests, it was applied on a new weather environment : Las Vegas. So, we compare old and new algorithms for the 3 retained methods : CMA-ES, N-M and Regression model. The main difference it that at each iteration, for obtained points that are out of convex hull, a projection on the database is applied. The old algorithm used the penalty function.

For covariance matrix adaptation (CMA-ES), the difference between both methods on the convergence is obtained on the figure 4.3. We observe easily that the new algorithm is more efficient. It is due of the fact that for this master thesis problem, the optimal point is very often on the boundary of the convex hull. Then, the projection is a way to reach more rapidly to this point.

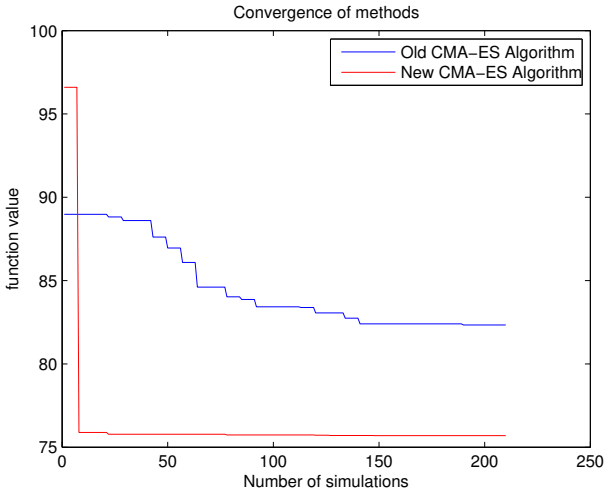


Figure 4.3: Comparison between both CMA-ES algorithms : *old* with penalty function and *new* with projection on the database

On the figure 4.4, we see the difference between both Nelder-Mead methods (old with penalty function and new with projection on the database). In this case, the algorithm with projection is worse than the algorithm with penalty function. It is due to the fact that the algorithm used a simplex (tetrahedron in 3 dimensional space). The $n + 1$ vertices of this tetrahedron has to be on different planes. On the new algorithm with the adding of the projection, the position of vertices are modified when one of them is out of the convex hull. During the execution of the optimization (cfr. figure 4.4), projections occurred at the 6th iteration. At this moment, the new algorithm improves its search. But when the algorithm had made this improvement, one vertex degenerated on the same plane as the 3 others vertices. Then, the consequence is that as the $n + 1$ vertices are not in the same plane, the algorithm is no more able to continue its search.

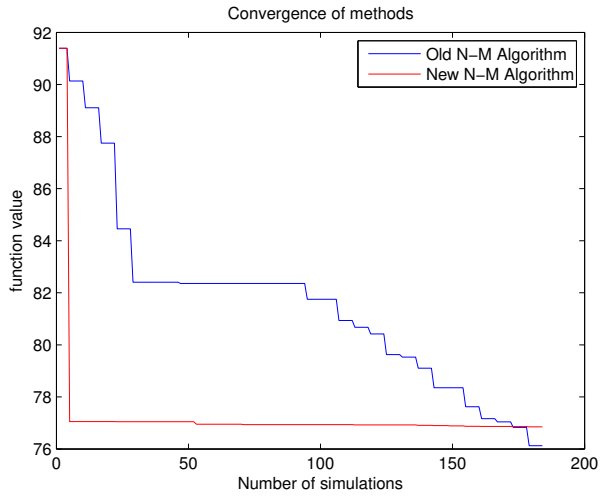


Figure 4.4: Comparison between both Nelder-Mead algorithms : *old* with penalty function and *new* with projection on the database

Finally, we see on figure 4.5 the convergence of both regression methods, with the same conclusion as for CMA-ES, the new algorithm converges more rapidly.

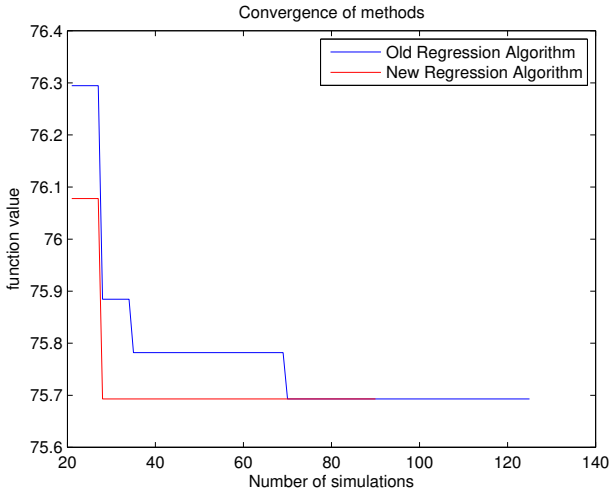


Figure 4.5: Comparison between both regression model algorithms : *old* with penalty function and *new* with projection on the database

Then, the conclusion is that we will compare only two algorithm CMA-ES and Regression for the rest of the analysis. As Nelder-Mead is no more efficient with the projection, this algorithm is abandoned. On the figure 4.6, it is the comparison between those three algorithms. This highlights that CMA-ES and Regression are more powerful than Nelder-Mead.

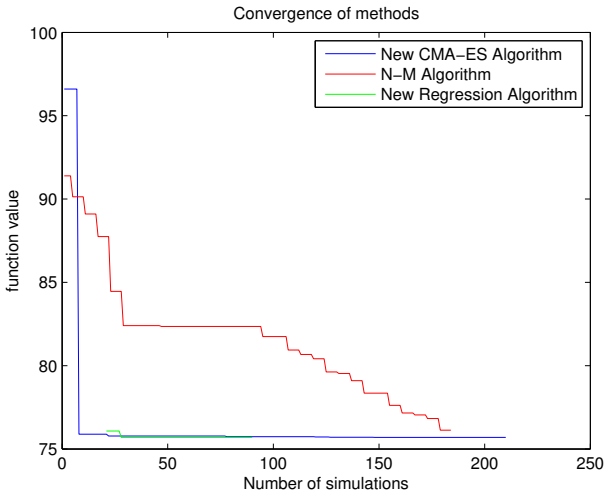


Figure 4.6: Comparison between algorithms with projection on the database

4.1.3 Tool to measure the optimality of final iterates

In order to measure the optimality of final iterates in each algorithm, we use a very well-known concept in optimization : Karush-Kuhn-Tucker (KKT) conditions. It is a way to verify that a solution of a constrained problem is optimum. A first way is to verify if the gradient is equal to zero for each component. But in many cases, the optimum is on the boundary of the domain. This way, we need this conditions because it is possible that its gradient is here not equal to zero. Then, the KKT conditions appear.

Karush-Kuhn-Tucker (KKT) conditions

If x^* is a local optimal solution for the problem

$$\min_{x \in \mathbb{R}^n} f(x) \text{ such that } c_i(x) \leq 0 \text{ for } i \in \mathcal{I}$$

such that the set of gradients of active constraints ($c_i(x^*) = 0$) with $A(x^*)$ are the indices of active constraints in $c_i(x^*)$

$$\{\nabla c_i(x^*)\}_{i \in A(x^*)} \text{ is linearly independent}$$

then it exists a Lagrange multiplier λ^* such that

$$\nabla f(x^*) = -\sum_{i \in A(x^*)} \lambda_i \nabla c_i(x^*) \text{ and } \lambda_i^* \geq 0 \text{ for all } i \in \mathcal{I}$$

Those conditions are adapted to our precise problem. First, $f(x)$ is a derivative black-box function, then in order to compute the gradient of the objective function, finite difference formulas are used. Second order finite difference (with error in $\mathcal{O}(h^2)$) was chosen because it's a good compromise between precision and minimum number of objective function evaluation. Then, for each component, we use this following formula :

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h}$$

Second, the constraints are linear $Ax \leq b$, let c_i and active constraint

$$c_i(x) = a_i x - b_i$$

It gradient is easily obtained as $\nabla c_i(x) = a_i$.

Then, we have the following KKT conditions

$$\nabla f(x^*) = -\sum_{i \in A(x^*)} \lambda_i a_i \text{ and } \lambda_i^* \geq 0 \text{ for all } i \in \mathcal{I}$$

We can rewrite this problem as an optimization problem such that

$$\min_{\lambda_i} \left\| \nabla f(x^*) + \sum_{i \in A(x^*)} \lambda_i a_i \right\| \text{ and } \lambda_i^* \geq 0 \text{ for all } i \in \mathcal{I} \tag{4.1}$$

Now the value of this norm is called as the optimality measure. Smaller is its value, more optimal is the point x^* according to KKT conditions. In order to solve this optimization problem, the MATLAB `quadprog` is used because it is considered as solving a quadratic programming problem.

4.1.4 Choice of the algorithm for the case of one glass

At present, thanks to this new tool to measure the optimality of our algorithms, we will compare both algorithms. In order to measure the optimality more accurately, four different complex problems are created. In the following table, all the characteristics about each building are reported.

Problem	Type	Construction material	Length (m)	Width (m)	Number of floors
1	Hospital	Light (Wood)	80	70	5
2	School	Heavy	60	25	3
3	Residential	Medium (Brick)	70	80	5
4	Office	Light (Wood)	30	30	5
Problem	Orientation (°)	Glazing facade 1 (%)	Glazing facade 2 (%)	Glazing facade 3 (%)	Glazing facade 4 (%)
1	30	90	40	70	50
2	320	10	80	40	60
3	120	80	50	50	70
4	100	40	50	60	80

On the following figures, each problem is presented according to its convergence and its KKT conditions. On the left figure, we show the traditional convergence while on the right figure, the function value at each iteration is sketched with the value

`opt` of the KKT conditon :

$$\texttt{opt} = \left\| f(x^*) + \sum_{i \in A(x^*)} \lambda_i a_i \right\|_2$$

The value of λ is computed after solving the optimization problem described in 4.1. When this value `opt` is near to zero, it seems that the point is considered as optimal in the sense of Karush-Kuhn-Tucker.

Note that we will never sure about this optimality as the gradient is computed by finite difference. But it stays a good tool to measure if the point is a local optimum. Then in order to compare those two algorithms, the algorithm that converges rapidly to an efficient local minimum will be chosen. The final choice will be empirical and so it is possible that this algorithm doesn't react the same way for all problems. On the following figures, we face to a new graph, the comparison between the function value and the value of optimality computed thanks to KKT. When the value of the objective function decreases, we can verify that the objective point improves its optimality in term of KKT. Hence, if the value of optimality increases while the value of the objective function decreases, it means that we switch from local minimum to another. Then, we can continue to optimize in order to obtain a better optimal point with a better optimality value.

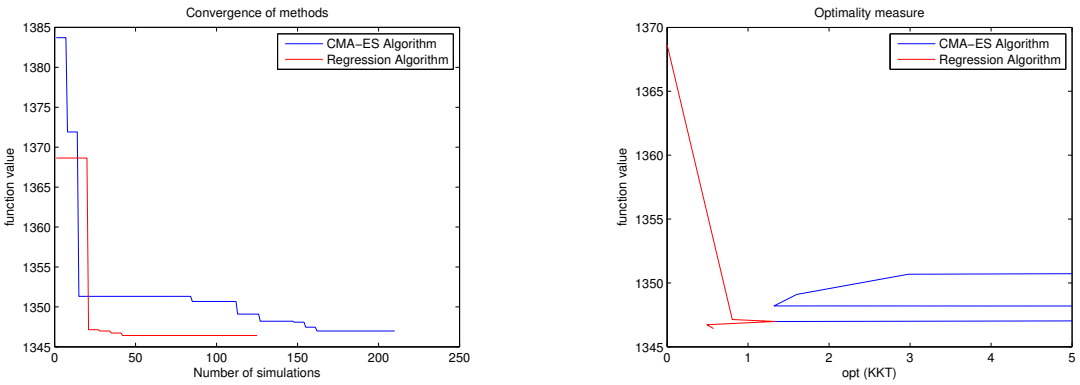


Figure 4.7: Convergence and KKT conditions for problem 1

The figure 4.7 represents the problem when the building is an hospital. In this situation, the regression algorithm converges more rapidly than CMA-ES. When we look at the KKT conditions, we observe that CMA-ES seems to change often from local minimum to another. Indeed, several times, the value `opt` explodes while the objective function value decreases.

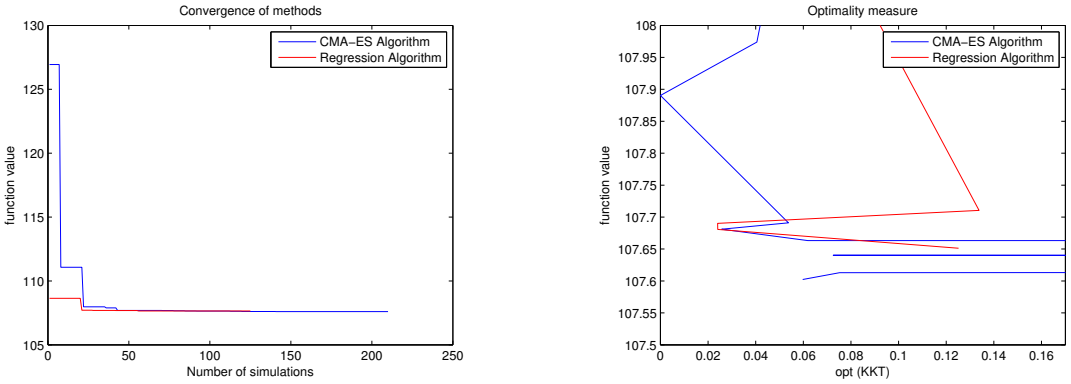


Figure 4.8: Convergence and KKT conditions for problem 2

In this second situation (cfr. figure 4.8), the convergence seems to be very similar for CMA-ES and Regression. On the other hand, CMA-ES finds a better point with lower objective function value. As previous, CMA-ES changes often from local minimum to another.

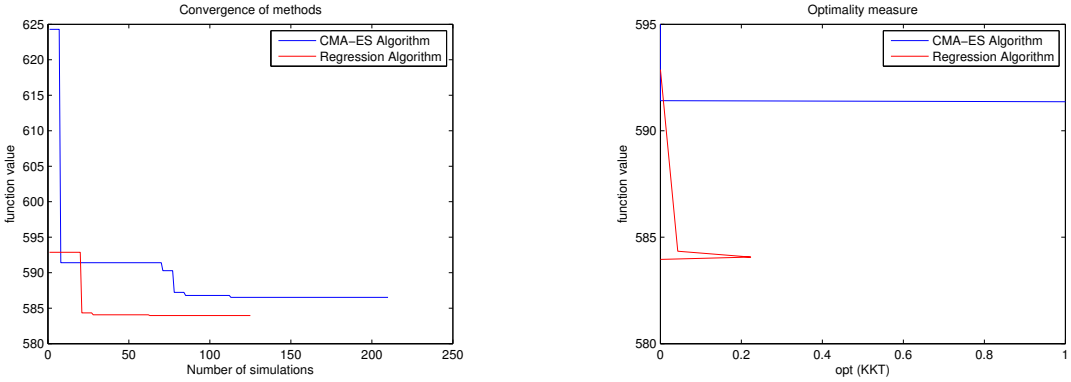


Figure 4.9: Convergence and KKT conditions for problem 3

In this third situation, the convergence is more efficient for the regression model. It finds directly the local minimum while CMA-ES arrives to a less performant glass.

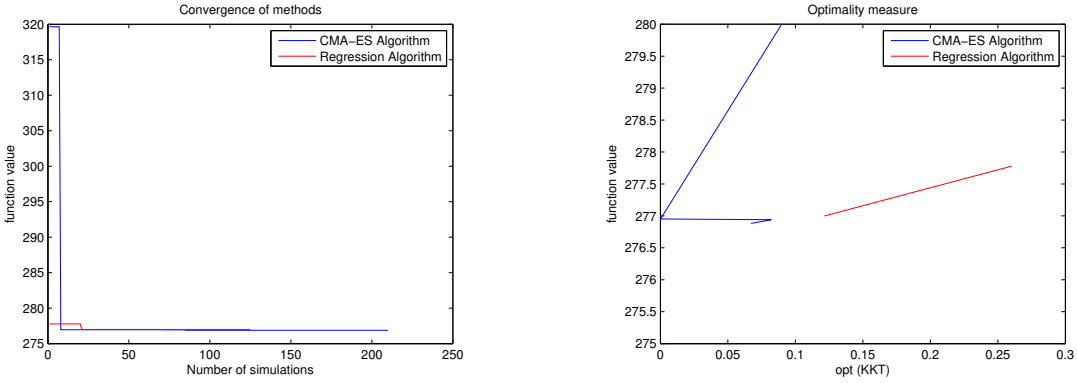


Figure 4.10: Convergence and KKT conditions for problem 4

In this final case, CMA-ES is more efficient than Regression model because it finds directly a perfect optimal point with its value `opt` equal to 0. But it changes to

another point because the algorithm has found another point with lower objective value. Regression model seems to converge to this value but at the end of the simulation, this one hasn't yet found this point.

After those different analyses and comparisons on those different algorithms, we can observe that Regression model seems to be more efficient in order to find rapidly a locally optimal glass. In the sense of KKT conditions, Regression model seems to stay often on the same minimum local and to converge to the best point. Then, in order to solve this problem of one glass, we decide to choose the algorithm **Regression model**.

4.1.5 Determination of optimal glass in the glass database

The different methods of optimization converge most of the time to a glass that is not in the set of glass products typically available on the market today. Indeed, as explained in the section about the definition of the glass domain, the convex hull is not completely filled by all the glasses, there are a lot of holes in the domain. The database is considered as a discrete set containing all the glass. On the other side, the convex hull is a continuous set containing all the glass from database. Then, in order to be sure that the proposed glass really exists, it is necessary to propose a glass that is really in the database of glasses.

The direct method is to use the method of k-nearest neighbors in order to determine the real glasses that are the nearest from the obtained glass by black-box optimization. The method looks in the database which glasses are the nearest according to its values U, g and LT. The number of glasses is determined by the parameter k. The method uses the euclidean distance in order to compute the distance between the glass and the one in the database. Note that variables are normed before using the euclidean distance in order that each variable is inside the same range. Indeed, the values of U are not in the same range as g or LT.

$$d(\mathrm{G}, \mathrm{G}_{\mathrm{database}}) = \sqrt{(U_{\mathrm{G}} - U_{\mathrm{G}_{\mathrm{database}}})^2 + (g_{\mathrm{G}} - g_{\mathrm{G}_{\mathrm{database}}})^2 + (LT_{\mathrm{G}} - LT_{\mathrm{G}_{\mathrm{database}}})^2} \tag{4.2}$$

Afterwards, the glasses from the database are compared between them. The best glass is kept. k can be chosen by the user. The EnergyPlus computation for all glasses are launched in parallel. Then, the chosen value for this parameter is the number of threads that the computer can do at the same time. In the case of the following tests, the number of threads is 7. Finally, among the 7 glasses, the glass with the smallest objective function value is chosen.

In order to evaluate the loss on the value of the objective function when we pass from continuous to discrete, that is to select the nearest glass in the database instead of the optimal glass found on the whole convex hull, the following formula is used :

$$\text{loss} = \frac{f(\text{optimal glass}) - f(\text{nearest glass})}{\text{range}} \tag{4.3}$$

Where the range is defined as the possible interval containing all objective function values obtained during the optimization

$$\text{range} = \max(\text{function values obtained}) - \min(\text{function values obtained})$$

In order to verify empirically that the selection of the glass in the database using k-nearest neighbors method is efficient, we compute the loss values for the four different complex problems previously introduced. The third column of the following tabular is the indice of the neighbor that has the smallest value of the objective function.

Problem	Loss (%)	Nth neighbors
1	5.12	5
2	0.24	2
3	0	1
4	1.64	3

According to those obtained values, it is easy to remark that there is not too much loss (below 5 %). 5 % is not big because this value is what we decrease compared to the range of possible values of objective function obtained during the optimization. For the third case, the value of the loss is null, it means that the value of the objective function is the same for the optimal glass and for the nearest neighbor. It doesn't mean that the optimal point matches exactly on a glass of the database. In this specific case, the glasses are not the same but are very close, probably to the fact that the optimization is located on a plateau.
Moreover, it is not always the nearest neighbor that has the smallest objective function value. In view of the values obtained empirically, the neighbors that are further than the optimal glass, have sometimes a better objective function value than the nearest.

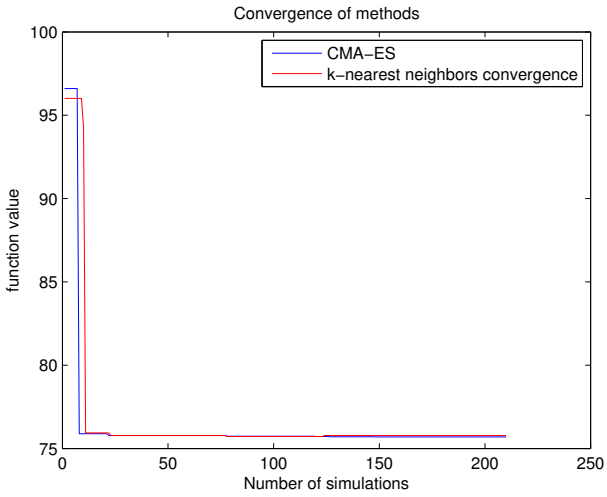


Figure 4.11: Comparison of the convergence between CMA-ES and k-nearest neighbors applied at each iteration on CMA-ES

In order to show that there is not a big loss between the solution obtained by CMA-ES and the resulting real glass obtained by k-nearest neighbors method, we make a comparison of the convergence between CMA-ES and k-nearest neighbors applied at each iteration on CMA-ES. The figure 4.11 represents the convergence of the case where it is the classic simple building of 1000 m^2 at the specific weather environment, Las Vegas. On this figure, the convergence of CMA-ES is compared with the one of the same CMA-ES such that at each iteration, the k-nearest neighbors method

is applied. At the end of the convergence, CMA-ES changes of optimum while k-nearest neighbors remains on the same point. Even if the algorithm tries to improve the value of its optimum, the final glass will remain the same.

4.2 Case of multiple glasses for buildings

The case of one glass for a whole building can be extended to a circumstance where there is more than only one type of glass used. Then, in this chapter, three cases are studied : when there are 2, 3 or 4 different glasses for the building. In order to solve this, we decide to allocate for each facade a glass.

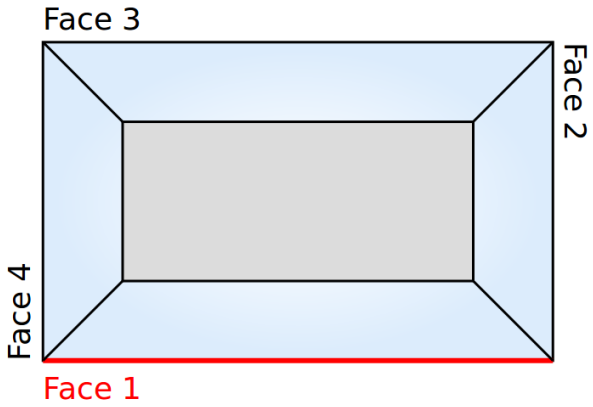


Figure 4.12: Building

The number of variables is computed according to :

$$\#\text{ variables } = 3 \times (\#\text{ faces})$$

For each case, the partition of glasses among facades are different. In the table 4.1, here is an explanation of how the variables are partitioned. The notation 'i:j' means that the variables between i and j are assigned to the facade.

# glasses	# variables	North	South	East	West
1	3	1:3	1:3	1:3	1:3
2	6	1:3	4:6	1:3	4:6
3	9	1:3	4:6	1:3	7:9
4	12	1:3	4:6	7:9	10:12

Table 4.1: Partition of glasses among facades

In order to analyze the case of multiple glasses, we redefine the classic simple problem, a square building of 1000 m^2 with all those following characteristics :

Length (m)	Width (m)	# floors	Weather	% Glazing on each facade	Material construction
40	25	1	Brussels	60	Heavy

The main adaptations for algorithms were made so that they can adapt to the number of glasses for the building. The projection on the database has been modified so that each triplet (U_i, g_i, LT_i) is projected properly on the 3-dimensional convex hull. So, for each case (2, 3 or 4 glasses), there are either 2, 3 or 4 projections at each iteration.

Hence, we launch optimizations for this example and we compare the convergence of CMA-ES and Regression method according to the number of different glasses for the building. The figures 4.13 and 4.14 represent respectively the cases with 2 and 3 glasses.

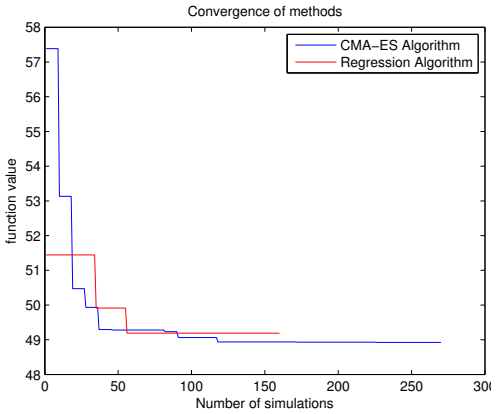


Figure 4.13: Convergence for 2 glasses

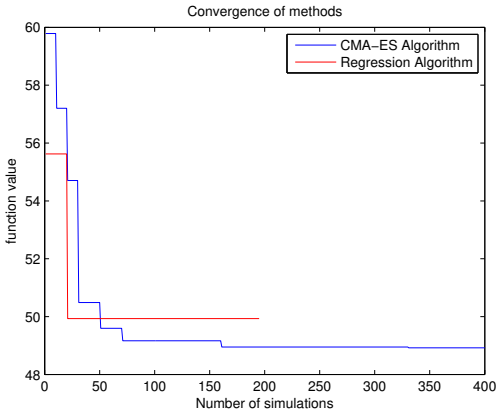


Figure 4.14: Convergence for 3 glasses

Surprisingly, regression method don't converge to the optimal glasses. It is unexpected because for one glass, regression method is the best method. This phenomenon also occurs for other problems. Either it is a mistake during implementation, or it is that regression is only efficient for problems with few variables. Then, as the performances for one glass between regression method and CMA-ES seem to be quite close and as CMA-ES is more powerful for multi glass, for the rest of the analysis, we will continue only with CMA-ES as being the main optimization method.

For those cases, KKT conditions can still be used. Indeed, when there are 4 different glasses, we have $x \in \mathbb{R}^{12}$ but the variable can also be written as

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \in \mathbb{R}^3 \times \mathbb{R}^3 \times \mathbb{R}^3 \times \mathbb{R}^3$$

Where x_i is each glass individually. Then, for each x_i, KKT conditions can be used separately. Indeed, for each glass x_i, it needs to satisfy

$$Ax_i \leq b$$

As the linear constraints are independent for each x_i, we can used KKT separately for each glass. With the same reasoning, we can conclude that the projection on the database can be performed separately for each glass.

In the case of the previous problem, the optimization converges constantly to the

same type of glass for each facade. It converges to those following glasses :

	U	g	LT
x_1^*	0.4638	15.29	30.0632
x_2^*	0.4637	13.3441	23.8725
x_3^*	0.4772	13.0896	24.0428
x_4^*	0.4709	13.2774	24.691

Then, as for this case, even if the first glass is not the same as the next three glasses, it converges for all facades to a similar glass. We need a problem for which the efficiency of the multi glass is proved. It means that we need to find a problem for which the glasses for each facade are different.

A new complex problem is defined and has the following characteristics :

Type	Construction material	Length (m)	Width (m)	Number of floors
Residential	Heavy (Concrete)	40	40	3
Orientation (°)	Glazing facade 1	Glazing facade 2	Glazing facade 3	Glazing facade 4
180	10	25	30	15

The weather of Saint-Petersbourg in Russia is selected. In this special case, as the proportions of glazing on each facade are very different, we should obtain different glasses for each facade. For this particular example, the following optimal point is obtained :

x_1^*	0.9458	29.0286	60.7003
x_2^*	1.0573	41.6673	75.7525
x_3^*	1.0630	29.9660	63.8691
x_4^*	1.0536	41.5309	75.7003

We observe that the algorithm proposes two types of glazing for the building : one for the facades north and east and another for the facades south and west. More g and LT are asked for facade south and west. South and west are more exposed by the sun in this part of the world.

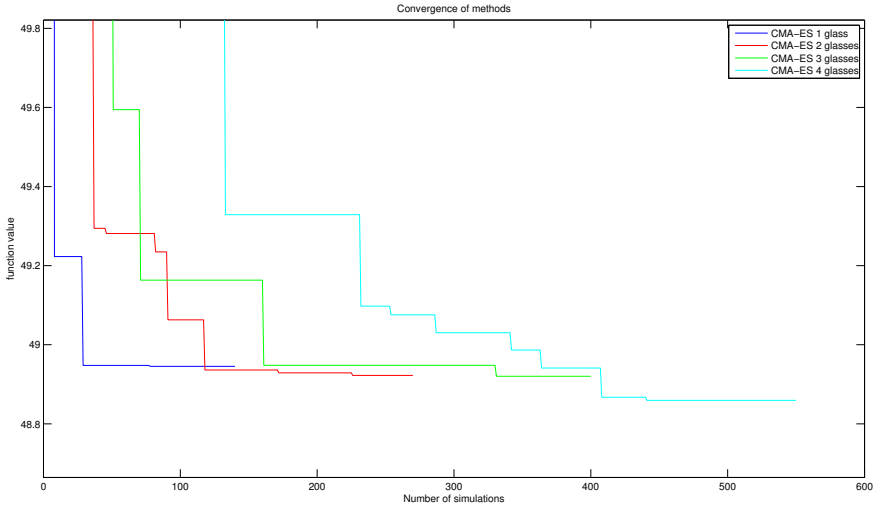


Figure 4.15: Comparison of convergence according to the number of glasses for the building

Figure 4.15 represents the comparison of the convergence of CMA-ES according to the number of glasses for the simple building in Belgium. Note that the number of iterations increases when the number of glasses also increases. Indeed, it is more difficult for the algorithm to search the optimum if the dimension of the problem increases. The figure confirms the assumption, the algorithm has more difficult to decrease the value of its objective function when the number of glasses is great. We also remark that to increase the number of glasses allows to obtain better results. Indeed, the optimal value of the objective function decreases according to the number of glasses.

Number of glasses	Gains compared to the case of one glass (%)
2	0.088
3	0.096258
4	0.33

We compute the gain in terms of the value of the objective function thanks to the formula (4.4), we remark that in this previous table, this gain is not so significant. Indeed, the gain doesn't exceed 1 %.

$$\text{gain} = \frac{f(\text{optimal point for 1 glass}) - f(\text{optimal point for n glasses})}{\text{range}} \tag{4.4}$$

Where n is the number of glasses and the range is defined as the possible interval containing all objective function values obtained during the optimization

$$\text{range} = \max(\text{function values obtained}) - \min(\text{function values obtained})$$

Chapter 5

Additional constraints for the problem

The classical method 'CMA-ES' works very well on the full domain, on the convex hull containing the set of all possible glasses. But it is probable that the user may want that the domain be more restricted. In this case, we can add additional constraints. So, in this chapter, three types of constraints will be discussed. First, constraints that are directly related to variables of the glass : U, g and LT. The second is related to different types of color for the glass and how it can influence the search of the optimal glass. The third type of constraints introduced in this master thesis is the price.

5.1 Constraints related to variables

The user can be restrictive about the domain of research. Then, he may require additional constraints linked to the main variables U, g and LT. These constraints have to be necessarily convex in order that the new domain remains convex. Then, the easiest way is to use linear constraints.

These constraints have the following form :

$$\begin{bmatrix} a & b & c \end{bmatrix} \begin{bmatrix} U \\ g \\ LT \end{bmatrix} \leq d$$

where a, b, c and d are parameters determined by the user.

The first method is to directly add the constraints after those that define the convex hull. Then, as the convex hull has the same form as the linear constraints, to append the new constraints is directly done.

A second way, more efficient, is to redefine the initial convex hull according to those linear constraints. In this case, the method goes through the database and only selects glasses that satisfy the constraints. Then, at the end of this procedure, only relevant glasses are kept and we redefine a new convex hull with these restrictive database. For this, the algorithm `vert2con` is reused in order to create the convex hull.

The advantage of this second method is to be sure to obtain the most restricted

domain. Indeed, as explained in the second chapter, the domain has a lot of holes. When we add a linear constraint to the convex hull, it is very likely that this constraint cuts the domain at the location where there is a hole. Thanks to the second method, the user is sure to obtain the smallest domain containing all the database that satisfies his constraints. This new domain is also adapted for the final projection on existing glasses, when the method of k-nearest neighbors is applied.

On the figure 5.1, an example is presented. The case where the user asks to restrict the domain according to this following simple constraint :

$$U \leq 0.8$$

We can remark that this new domain is very different from the initial convex hull on the figure 2.6.

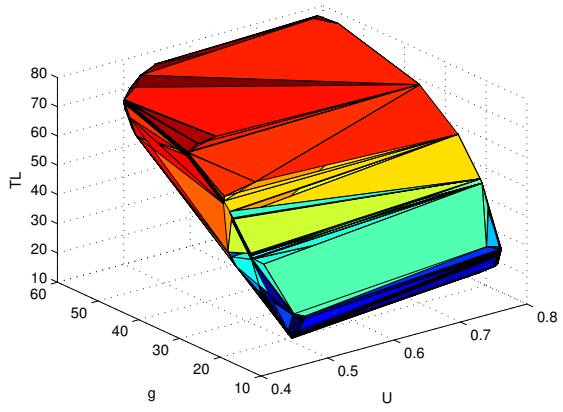


Figure 5.1: New convex hull after adding constraint $U \leq 0.8$

We can also analyze the case where there is more than one linear constraint. For instance, in the case of these following linear constraints,

$$U \geq 0.8$$
$$g \leq 30$$
$$LT \leq 40$$

we obtain the convex hull from figure 5.2.

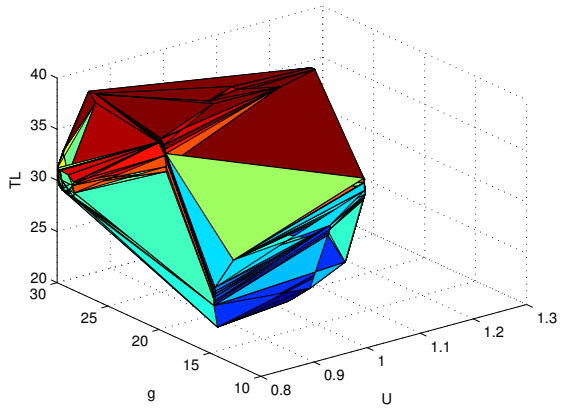


Figure 5.2: New convex hull after adding constraint $U \geq 0.8$, $g \leq 30$ and $LT \leq 40$

The second method is also relevant for non linear constraints. Indeed, as the algorithm selects in the database glasses that satisfy the constraints, other convex constraints than linear are also applicable to the problem. For example, if the user asks to find a glass which is similar to a specific glass chosen beforehand, it is possible to redefine a convex hull around this glass as an ellipsoid constraint.

$$\frac{(U-a)^2}{r_1^2} + \frac{(g-b)^2}{r_2^2} + \frac{(LT-c)^2}{r_3^2} \leq 1$$

where a, b, c, r_1, r_2 and r_3 are parameters defined by the user. $[a, b, c]$ is the center of the ellipsoid and $[r_1, r_2, r_3]$ are respective radius for each variable. This way, a glass that is in a certain radius of the initial glass, will be suggested.

For instance, if the user asks the following type of glass, U centered at 0.6 with radius 0.2, g centered at 30 with radius 10 and finally LT centered at 40 with radius 20, the convex hull from figure 5.3 is :

$$\frac{(U-0.6)^2}{0.2^2} + \frac{(g-30)^2}{10^2} + \frac{(LT-40)^2}{20^2} \leq 1$$

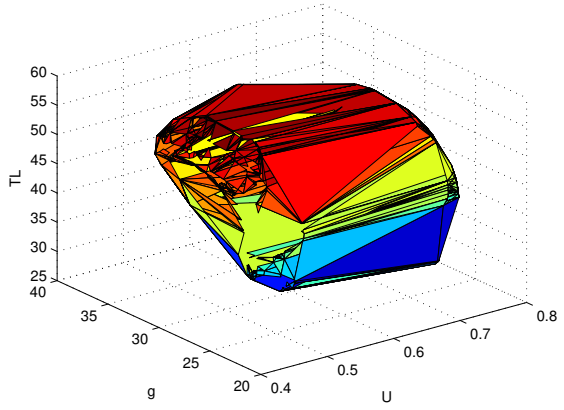


Figure 5.3: New convex hull after adding ellipsoid constraint

We can remark that the resulting convex hull does not really look like an ellipsoid anymore. It is due of the fact that the algorithm rebuilds the smallest convex hull containing all the database points that satisfy to these constraints.

5.2 Aesthetic constraints : color of the glass

The way used by AGC to define the color of a glass is the $L^*a^*b^*$ CIE system (Commission Internationale de l'Éclairage). This color space describes mathematically all perceivable color in three dimensions : L^* represents the lightness, darkest black at $L^* = 0$, and the brightest white at $L^* = 100$, while a^* and b^* denote the color components green-red and blue-yellow. The red/green opponent colors are represented along the a^* axis, with green at negative a^* values and red at positive a^* values. The yellow/blue opponent colors are represented along the b^* axis, with blue at negative b^* values and yellow at positive b^* values. The CIE system is represented on figure 5.4.

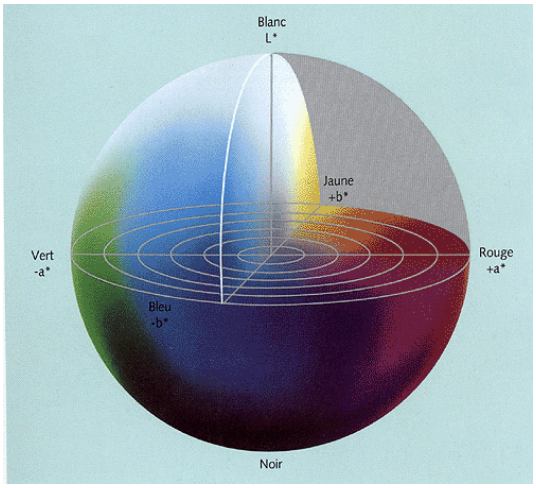


Figure 5.4: System CIE L^*, a^*, b^*

The $L^*a^*b^*$ CIE system used by AGC can be computed from the traditional CIE XYZ system via the following simple formulas :

$$L^* = 116 \left(\frac{Y}{Y_n}\right)^{\frac{1}{3}} - 16$$

$$a^* = 500 \left(\left(\frac{X}{X_n}\right)^{\frac{1}{3}} - \left(\frac{Y}{Y_n}\right)^{\frac{1}{3}}\right)$$

$$a^* = 200 \left(\left(\frac{Y}{Y_n}\right)^{\frac{1}{3}} - \left(\frac{Z}{Z_n}\right)^{\frac{1}{3}}\right)$$

where X_n, Y_n and Z_n are XYZ values of the reference white point.

The algorithm to redefine the new convex hull is quite equivalent as the one introduced for the case where the constraints have a direct link with variables of the glass. The method goes through the database and only select glasses that satisfy the color constraint. The user selects the color of his glass that the user wants for his

building. He encodes color parameters according to the CIE system : for the front and back of the glass, he defines L^*, a^* and b^*. During its research in the database, the algorithm selects all glasses that are inside the following hypercube,

$$\begin{aligned}
L^*_{\text{user,front}} - \Delta L \leq L^*_{\text{glass,front}} \leq L^*_{\text{user,front}} + \Delta L \\
L^*_{\text{user,back}} - \Delta L \leq L^*_{\text{glass,back}} \leq L^*_{\text{user,back}} + \Delta L \\
a^*_{\text{user,front}} - \Delta a \leq a^*_{\text{glass,front}} \leq a^*_{\text{user,front}} + \Delta a \\
a^*_{\text{user,back}} - \Delta a \leq a^*_{\text{glass,back}} \leq a^*_{\text{user,back}} + \Delta a \\
b^*_{\text{user,front}} - \Delta b \leq b^*_{\text{glass,front}} \leq b^*_{\text{user,front}} + \Delta b \\
b^*_{\text{user,back}} - \Delta b \leq b^*_{\text{glass,back}} \leq b^*_{\text{user,back}} + \Delta b
\end{aligned}$$

A particular case is studied in order to analyze the efficiency of the method. For instance, the user choose these following parameters

$L^*_{\text{user,front}}$	$a^*_{\text{user,front}}$	$b^*_{\text{user,front}}$	$L^*_{\text{user,back}}$	$a^*_{\text{user,back}}$	$b^*_{\text{user,back}}$	ΔL	Δa	Δb
21	-2.5	-1.6	18	-1.9	-5.7	5	1	1

Figure 5.5 shows the obtained convex hull. We remark that this convex hull strongly restricts the domain but still remains large. The convex hull also includes glasses that are not the same color as the one requested by the user. We could believe that the main disadvantage of this method is that the optimization method can converge to a glass that doesn't correspond to the demand of the user, it could be that the color obtained after optimization is not the right one. Indeed after optimization, when the procedure is at the post-processing, k-nearest neighbors method is applied and is looking at the nearest glass of the optimal point in the database. This database has been restrained, it is the domain containing glasses with the right color.

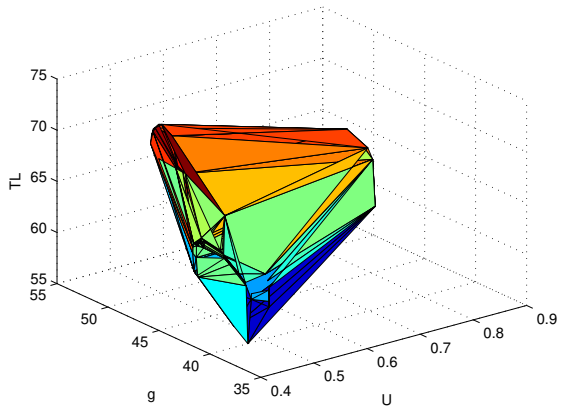


Figure 5.5: New convex hull after adding aesthetic constraint

But in fact, after optimization, the algorithm converges to a glass that has these particular color. Because the algorithm determines an optimal glass in the database thanks to k-nearest neighbors method. The algorithm suggests the best neighbors that are inside the range of color.

5.3 Price Constraint

Price is the other main criterion that can influence the choice between glasses. Suppose that the price p is known for all glass in the database. Indeed, a user has a budget and it is difficult for him to exceed his target. The system of recommendation for the optimal glass has to propose a glass that stays inside the budget of the user. Then, the user encodes in the tool his target t that corresponds to his budget for a glass. This target will have a direct impact in the objective function and also in the search of the optimal glass. The objective function can be written as

$$f(x) = \alpha \text{ Energy} + (1-\alpha) \text{ Thermal Discomfort} + \gamma[p-t]_+ \tag{5.1}$$

where p is the price of the glass and $[x]_+$ is considered as the positive function :

$$[x]_+ = \begin{cases} 0 \text{ if } x < 0 \\ x \text{ if } x \geq 0 \end{cases}$$

The parameter γ is a parameter that we will increase in order to determine the impact of the price on the search of the optimum. The effect of using the positive function is that when $p \leq t$, there is no change on the objective function. In the other hand, if $p > t$, the value of the objective function is penalized.

In this section, three types of methods are described : the first is based on determining the value of the price such that its value is interpolated according to other glasses. The second way is to consider the price as a fourth variable and the optimization is applied on a 4-dimensional space. The third way is to remove too expensive glasses from the database. It means that if the price of a glass p is greater than the target t fixed by the user, this glass is not taken into account and a new convex hull is computed.

Method 1 : Price is determined by interpolation

The price p is defined as a function of variables U, g and LT.

$$p(U, g, LT)$$

This price is determined by interpolation. In order to interpolate, the rule of k-nearest neighbors is used. The method looks in the database which glasses are the nearest according to its values U, g and LT. The number of glasses is determined by the parameter k. The method uses the euclidean distance in order to compute the distance between the glass and the one in the database.

$$d(\mathrm{G}, \mathrm{G}_{\mathrm{database}}) = \sqrt{(U_{\mathrm{G}} - U_{\mathrm{G}_{\mathrm{database}}})^2 + (g_{\mathrm{G}} - g_{\mathrm{G}_{\mathrm{database}}})^2 + (LT_{\mathrm{G}} - LT_{\mathrm{G}_{\mathrm{database}}})^2}$$

When the k-nearest neighbors are obtained, the mean of prices between glasses from database is computed.

We set k to 4. The database of glasses is in dimension 3 : U, g and LT. We want that it computes the mean of prices between glasses that surround the point. Then, we hope that the point is inside a tetrahedron formed by 4 glasses in the database as presented in the figure 5.6.

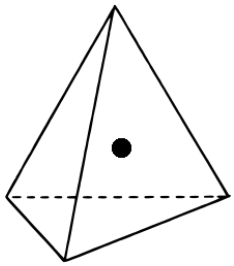


Figure 5.6: Point inside of the tetrahedron formed by 4 glasses in the database

In fact, it is not always the case, the point can be outside this tetrahedron formed by its 4 nearest neighbors. In order to illustrate this assumption, consider the 2-dimensional case represented on the figure 5.7, where the red point has 3-nearest neighbors, the dotted points of green. The red point is outside of the triangle formed by its 3 dotted point in green.

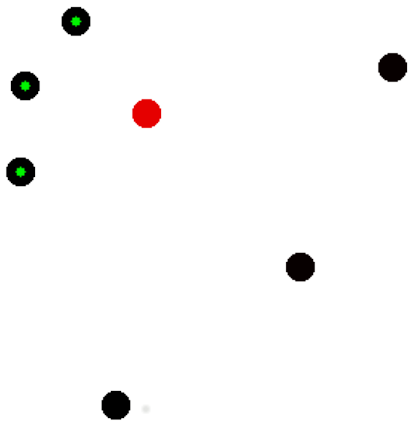


Figure 5.7: Point red is outside of the tetrahedron formed by its 3-nearest neighbors (dotted in green)

We can conclude that to chose the nearest neighbors is not always the best strategy to interpolate the price.

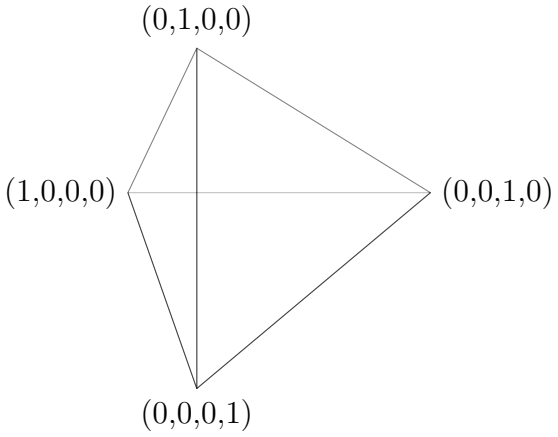
An example is set in order to analyze the efficiency of this algorithm. It is a basic example, a one floor square building of 1000 m^2 in Belgium. Then, a certain target is set at 25 and the value of γ is fixed to 10 in order that the price as a great importance in the search of the optimum. Then, the optimization is launched and the value of the optimum is 53.2366 and converges to the point

U	g	LT	price
0.9831	22.0573	40.154	22.4250

Here, the method CMA-ES with interpolated objective price converge properly to a point that is appropriate to the budget of the user. After projecting on the database with k-nearest neighbors method, the glass stay inside the budget of the user, the method interpolates correctly in this case. The loss in energy and comfort is not negligible compared to the optimal glass obtained before, which is around 25 %

according to the formula (4.3). It is due of the fact that the target is set too low before having convincing results in terms of energy and comfort.

Another method in order to improve the interpolation is to use the barycentric coordinate system. The new coordinates will be used to weight the mean of different prices. The Barycentric coordinate system is a coordinate system in which the location of vertices of a simplex is the basis of the system. In our case, as it is a 3-dimensional space, the simplex is a tetrahedron.



Coordinates of a point p in this new system are computed according to its position related to vertices. When the coordinates are not negative, it means that the point p lies inside the simplex. Coordinates can also extend outside the simplex, where one or more coordinates become negative. The values of coordinates are restricted with one condition :

$$\sum a_i = 1 \text{ where } a_i \text{ are the coordinates}$$

This allows to have unique coordinates for each point p. As a point p depends linearly on each vertex v_i of the simplex, we have

$$\sum_{i=1}^{4} a_i v_i = p \text{ where } v_i = \begin{pmatrix} U_i \\ g_i \\ LT_i \end{pmatrix} \text{ and } p = \begin{pmatrix} U_p \\ g_p \\ LT_p \end{pmatrix}$$

Hence, the barycentric coordinates can be solved as the solution of the linear system

$$\begin{pmatrix} U_1 & U_2 & U_3 & U_4 \\ g_1 & g_2 & g_3 & g_4 \\ LT_1 & LT_2 & LT_3 & LT_4 \\ 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{pmatrix} = \begin{pmatrix} U_p \\ g_p \\ LT_p \\ 1 \end{pmatrix}$$

The method to approximate the price for a glass consists on computing the k-nearest neighbors of the glass with $k = 4$. Afterwards, k-nearest neighbors are considered as the vertices of the simplex (tetrahedron) and the barycentric coordinates for the glass are computed. According to this new coordinates, the approximation of the price is the weighted mean of prices of k-nearest neighbors.
The drawback of this method is that in practice, the point p is often outside the tetrahedron. Then, the interpolation approximates the price. For instance, in the case of a one floor square building of 1000 m^2 in Belgium, it converges to $U = 0.4438$

$g = 21.6894$ and $LT = 47.1768$, its k-nearest neighbors are then computed int the following table.

nearest neighbor	U	g	LT	price
1	0.4438	21.9350	46.3931	37.0000
2	0.4706	22.1554	46.6837	33.0000
3	0.4646	22.3837	46.8791	37.4000
4	0.4650	22.4231	46.9712	37.2000

Table 5.1: k-nearest neighbors of $U = 0.4438$ $g = 21.6894$ and $LT = 47.1768$

The loss in energy and comfort is then very low compared to the optimal glass, $\sim 8\%$. But at the post processing, when the algorithm suggests a glass in the database, it advises a glass that is much more expensive than the target. The interpolation is very inaccurate predictor. For these reasons, this method is not reliable.

Another method to interpolate the price is to use the Shepard interpolation, also called inverse distance weighting. The general form of finding an interpolated value of the price p based on N samples p_i is :

$$p(x) = \begin{cases} \frac{\sum_{i=1}^{N} w_i(x) p_i}{\sum_{i=1}^{N} w_i(x)} \text{ if } d(x, x_i) \neq 0 \text{ for all } i \\ p_i \text{ if } d(x, x_i) = 0 \text{ for some } i \end{cases}$$

Where

$$w_i(x) = \frac{1}{d(x, x_i)^p}$$

The Euclidean distance is used and p is a parameter fixed at 5. No deeper analysis about the value of the parameter p has been made but article [12] provides an analysis on how to determine an acceptable value for the parameter. This interpolation method should be more accurate than the first method that uses the unweighted mean between k-nearest neighbors in order to interpolate the price. On the one floor square building of 1000 m^2 in Belgium, both algorithms are compared on the figure 5.8. The loss in terms of energy and comfort is smaller than the mean of prices of k-nearest neighbors, 18 % against 25 % according to the formula (4.3).

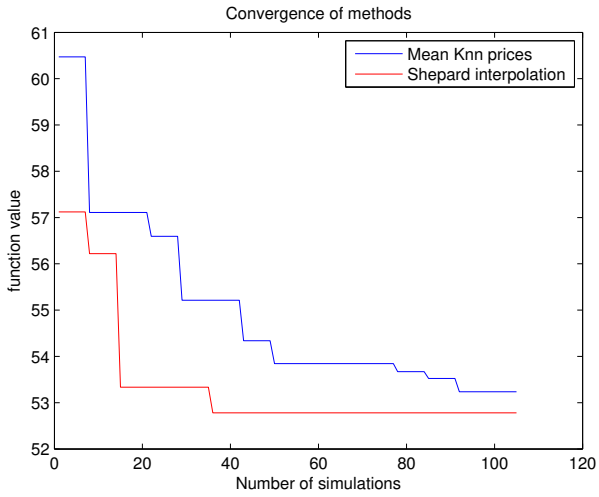


Figure 5.8: Convergence of both interpolation methods : mean of prices of k-nearest neighbors and Shepard interpolation

As expected, the Shepard interpolation is more accurate than the first interpolation method. The Shepard interpolation is set as method to interpolate the price.

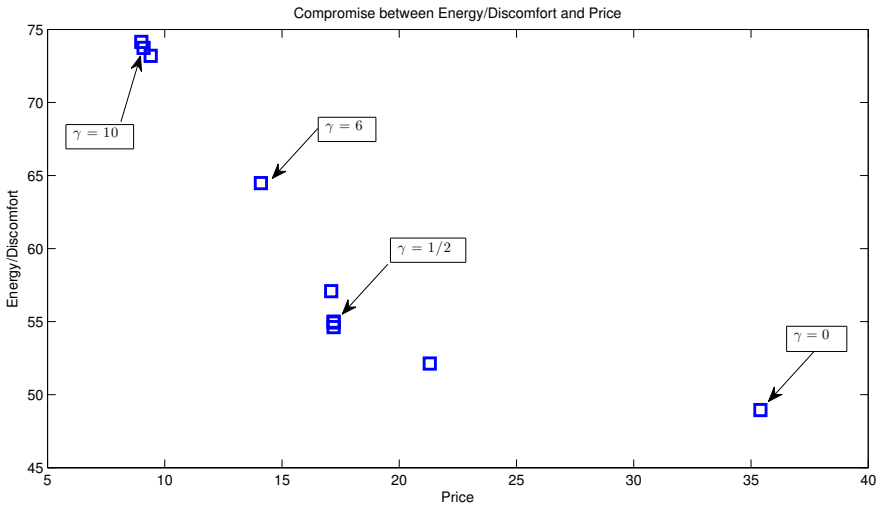


Figure 5.9: Compromise between Energy/Discomfort and Price for Sheppard interpolation; note that the target is not applied here, the price penalizes fully the objective function

On the figure 5.9, different optimization simulations were launched on the same problem with different value of γ, the parameter that determines the impact of the price and penalizes the objective function if the price is too high. Note that the target is not applied here, the price penalizes fully the objective function. The goal is to analyze the impact of the price on the search for the optimal glass. If γ is equal to zero, only the energy and the discomfort is to optimize. Then, the algorithm converges to a glass that is expensive but with the lowest energy consumption and maximal comfort. If γ increases, a new glass is rapidly attained with a lower price and where the impact of the energy and discomfort is not too much impacted.

When $0.5 \leq \gamma \leq 2$, the glasses obtained seems to be in the same category, CMA-ES algorithm tries to obtain a glass with not too much loss of energy and discomfort and with not too much price. When $\gamma > 5$, the algorithm only optimizes the price neglecting the energy consumed and the comfort.

Then, if the user wants the best glass for its building in terms of comfort and energy consumed, he has to buy a very expensive glass. Otherwise, if the customer is ready to make some concessions on the energy and the comfort, he can obtain a glass that is half the price with a limited loss of energy and comfort. Finally, if the user focuses only on the price, he will be directly impacted on the energy consumed and the comfort in his building.

Method 2 : Price is considered as an additional parameter

The second way to solve this problem with a price constraint is to consider the price p as a fourth variable. In this way, we have not only 3 variables U, g and LT but we add a fourth variable p.

Then, in order to take into account this fourth variable in the search of the minimum, we create a convex hull with those 4 variables thanks to the database of AGC. We have now a 4-d convex hull. It is no more possible to represent this space into a figure.

We launch CMA-ES algorithm with this 4 variables and the new convex hull. The objective function remains the same

$$f(x) = \alpha \text{ Energy} + (1 - \alpha) \text{ Thermal Discomfort} + \gamma[p - t]_+$$

Now, we will analyze the efficiency of this algorithm and compare with the Shepard method. It is the same example as the first example, a one floor square building of 1000 m^2 in Belgium with a target at 25 and the value of γ at 10. Then, the optimization is launched and the value of the optimum is 51.5325 and converge to the point

U	g	LT	price
0.8776	16.0785	25.2643	22.2753

Then, the method converge properly to a point that is appropriate for the budget of the user and is more efficient than Shepard interpolation method. On the following figure 5.10, the convergence of both methods is displayed. On this example, the loss in terms of energy and comfort is smaller than Sheppard interpolation, 10 % against 19 % according to the formula (4.3).

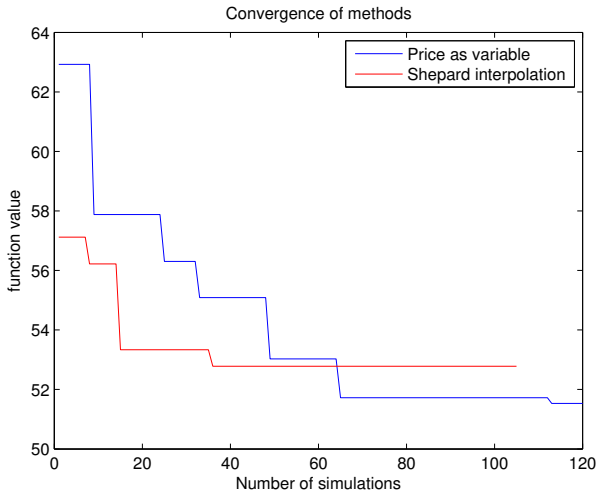


Figure 5.10: Convergence of both methods : price as fourth variable and Shepard interpolation

nearest neighbor	U	g	LT	price
1	0.8522	15.5685	26.2494	25.6000
2	0.8536	15.7012	26.4813	25.5000
3	0.8536	15.6036	26.4823	25.5000
4	0.8536	15.5808	26.4809	25.5000

Table 5.2: k-nearest neighbors of 0.8776, $g = 16.0785$, $LT = 25.2643$ and $price = 22.2753$

At the post processing stage, when the algorithm suggests a glass in the database, it advises a glass that is a little bit more expensive than the target. As the overtaking is not so excessive, it is not problematic that the algorithm suggest a glass that exceeds a little the target.

In the same way as Shepard interpolation, on figure 5.11 different optimization simulations were launched on the same problem with different value of γ, the parameter that determines the impact of the price and penalizes the objective function if the price is too high. Similar conclusions are therefore obtained. If γ is equal to zero, only the energy and the discomfort is to optimize. Then, the algorithm converges to a glass that is expensive but with the lowest energy consumption and maximal comfort. When $0.5 \leq \gamma \leq 2$, CMA-ES algorithm obtains a good compromise between energy, discomfort and price. And finally, when $\gamma > 5$, the algorithm only optimizes the price neglecting the energy consumed and the comfort.

So the conclusion obtained before remains valid. It's better for the customer to make some concessions in terms of energy and comfort because in this case he can obtain a glass that is half the price.

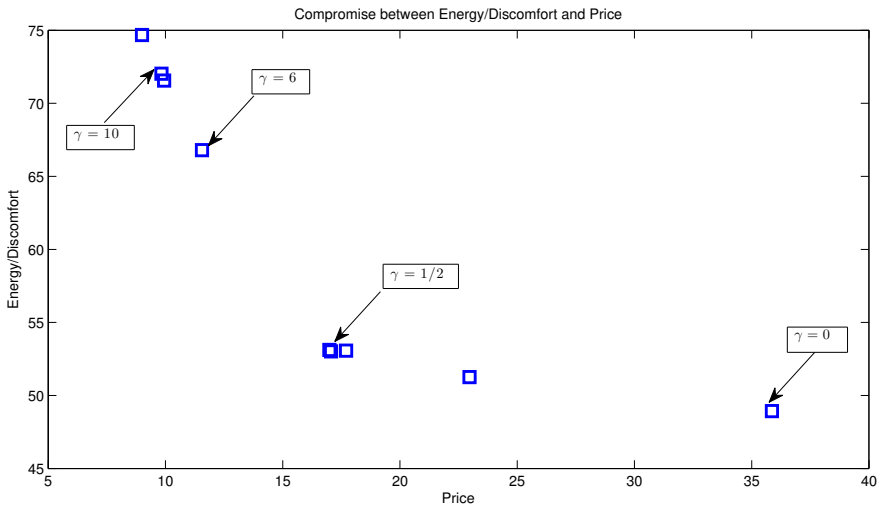


Figure 5.11: Compromise between Energy and Price for method 2; note that the target is not applied here, the price penalizes fully the objective function

Figure 5.12 represents the comparison of the compromise between the energy and the price for Shepard interpolation and for the method when the price is considered as an additional parameter. We can remark that both methods evolve in the same way.

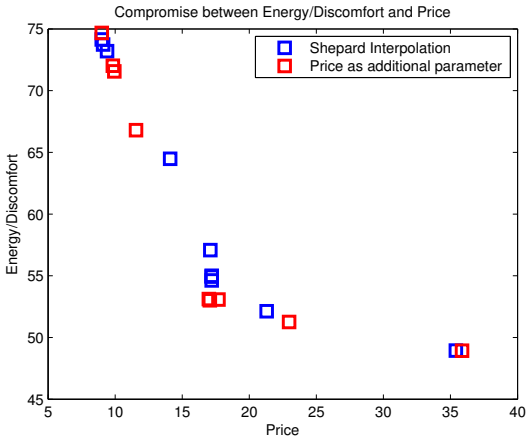


Figure 5.12: Comparison of the compromise between Energy and Price for Shepard interpolation and for method 2

Method 3 : Remove too expensive glasses from the database

The third way to solve this problem with a price constraint is to remove too expensive glasses from the database. It means that if the price of a glass p is greater than the target t fixed by the user, this glass is not taken into account.

Then, in this case, the method is the following, it goes through the database and only select glasses with a price p smaller than the target t. Then, at the end of this procedure, only relevant glasses are kept and we redefine a new convex hull with these restrictive database.

For instance, if the target is fixed at 25, figure 5.13 represents the resulting convex hull. This convex hull only contains glasses with price p smaller than 25. Note that, for simplicity, only the convex hull with $U \leq 3$ is represented on the figure.

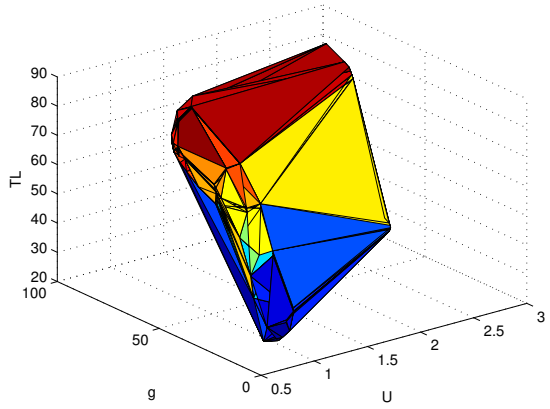


Figure 5.13: New convex hull after restricting the domain with only glasses cheaper than 25

The objective function remains the same as in the initial. The value of the objective function is not penalized because as the convex hull has been redefined, it is not possible that $p > t$.

$$f(x) = \alpha \text{ Energy} + (1 - \alpha) \text{ Thermal Discomfort}$$

After optimization and post-processing with k-nearest neighbors method, the method converges to the following glass. Note that the price is exactly the target t, 25.

U	g	LT	price
0.86096	15.9311	27.6752	25

Figure 5.14 represents the comparison between this method and the second method when the price is considered as a variable. Surprisingly, this method converge to a better glass than the second method. It is remarkable because the glass obtained here is not penalized in the case of the second method; this precise glass should be theoretically attained by the second method. It would seem that the algorithm did not explore this particular region. So we understand why restricting the domain has a great advantage : the algorithm explore a smaller space and the probability to attain the optimal glass increases.

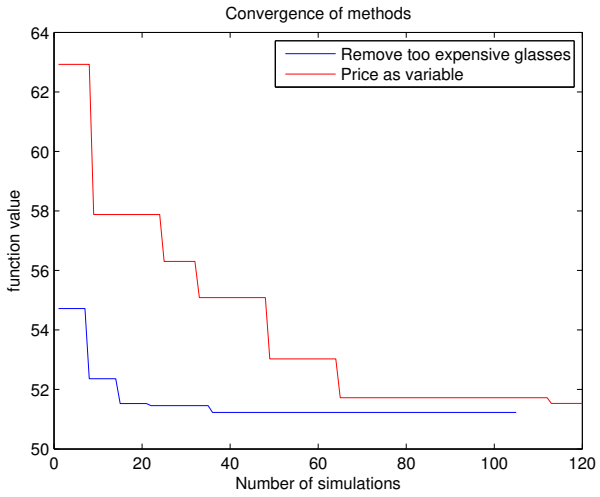


Figure 5.14: Convergence of both methods : Remove too expensive glasses and price as fourth variable

At the end of this chapter, we can conclude that adding the price to the optimization brings new interesting results. The experiments seem to conclude that restricting the domain by removing glasses with higher price is the best solution to include the price in the optimization.
We conclude also that for methods 1 and 2, penalizing too much the price can have irrelevant results where the energy consumed and the comfort are no longer taken into account. In the same way, if the user doesn't care about the price and only focuses on the energy consumed, he can end up with a glass that is very expensive. The solution here is to find the best compromise between energy consumed, comfort and price. In this way, the user can rapidly obtain a cheaper glass than the optimal glass in terms of energy and comfort, half the price and without an important loss of energy and comfort.

Chapter 6

Conclusion and possible way-ahead

Summary

Chapter 2 shortly explained how a study of a set of glass products typically available on the market today was done to create a convex hull containing the entire glazing catalog. It described how the optimization problem has been formulated. Afterwards, simple discrete hill climbing algorithm has been implemented in order to look at how the algorithm behaves on this problem. It concludes that we need to call new types of method : Derivative-free black-box optimization algorithms.

The chapter 3 exposes existing derivative-free black-box optimization methods that exhibit some of the wanted properties. It compared the algorithm for each method and stated theoretical advantages and drawbacks. Three kinds of algorithm are introduced : deterministic algorithms, evolution strategies and finally surrogate model.

In Chapter 4, the case of one glass for the building is first presented. A comparison of performance of each derivative-free black box optimization algorithm is presented on simple problems. Three algorithms are retained : a deterministic algorithm, Nelder-Mead, an evolution strategy, CMA-ES and finally a surrogate model, the quadratic regression.
At this stage, the algorithms are improved by the projection on the database. This projection is also an optimization problem that consists to project at each iteration the variable on the convex hull. The algorithms are compared according to whether or not they use the projection. Finally, for the conclusion for CMA-ES and Regression method is that the new algorithm with projection converges more rapidly. On the other hand, for Nelder-Mead, the algorithm with projection is worse than the classical algorithm. Finally, it was easy to remark that CMA-ES and Regression are more powerful than Nelder-Mead.
In order to measure the optimality of CMA-ES and Nelder-Mead, a very well-known concept in optimization is used : Karush-Kuhn-Tucker (KKT) conditions. In the sense of KKT conditions, Regression model seems to converge to the best point. Then, in order to solve this problem of one glass, the Regression algorithm is chosen. Unfortunately, as expected, the different methods of optimization converge most of the time to a glass that is not in the set of glasses. Then, in order to be sure that

the proposed glass really exists, it is necessary to propose a glass which is part of the catalog of glasses. The method used with success is k-nearest neighbors. It determines the real glasses that are the nearest from the optimal glass obtained by black-box optimization.
The case of one glass for a whole building is extended to a circumstance where there are more than only one glass. Then, in that chapter, three cases are studied : when there are 2, 3 or 4 different glasses for the building. The convergence of CMA-ES and Regression method are compared according to the number of different glasses for the building. Surprisingly, the Regression method doesn't converge to the optimal glasses. It is unexpected because, for one glass, it is the best method. As CMA-ES is more powerful for multi glass, CMA-ES is therefore considered as the main optimization method for the rest of the analysis.

Chapter 5 adds additional constraints to the problem. Three types of constraints are discussed. First, constraints that are directly related to variables of the glass : U, g and LT. Second, the different types of color for the glass will be presented and how it can influence the search of the optimal glass. Third, the price is the last type of constraints introduced in this master thesis. Three types of methods to take the price into account are proposed : the price is determined by interpolation. The price can be considered as an additional parameter p in addition to U, g and LT. Or if the price of a glass is greater than the target fixed by the user, this glass is not taken into account and a new convex hull is computed. The selected method is the third one.

Future work

Our study finishes by giving a way-ahead idea to expand the present results. The main idea of this improvement is to use derivative algorithm instead of derivative-free black-box optimization algorithms. As EnergyPlus is open-source, the source code is accessible and is written in C++. As showed in the figure 6.1, the code is divided into three Managers : Surface Heat Balance, Air Heat Balance and Building Systems Simulation. Each manager has its modules that is concerned about a specific calculation. The source code is represented as sub-functions that communicate with each other. The idea here is to propose a contribution to EnergyPlus. For each sub-functions, the gradient is computed according to what the function produces. If another function calls another one, the gradient is naturally computed according to the following formula.

$$(f(g(x)))' = f'(g(x))g'(x)$$

In the same way, when two sub-functions are multiplied between them, the gradient is computed thanks to this following formula.

$$(f(x)g(x))' = f'(x)g(x) + f(x)g'(x)$$

As EnergyPlus is made up of a lot of functions, it would need a lot of functions to compute their gradients. This work will require a lot of resources. It would be a new approach in order to optimize EnergyPlus. Derivative based optimization algorithms may be used, providing a more reliable approach because those methods

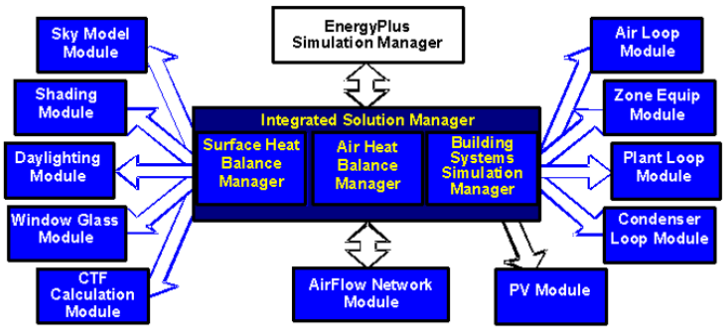


Figure 6.1: EnergyPlus Program Schematic

guarantee the local optimality of the obtained glass.

Conclusion

This master thesis is offering a response to the request of AGC. An optimization algorithm with MATLAB is developed in order to determine the optimal glass for a specific building. The tool uses the improved algorithm CMA-ES that optimizes the consumed energy and the comfort in the building through a simulation, EnergyPlus. The user can add directly constraints related to variables. The color selected by the user will restrict the domain of research. He can also set a target for the price and the algorithm will take in account this additional constraint.

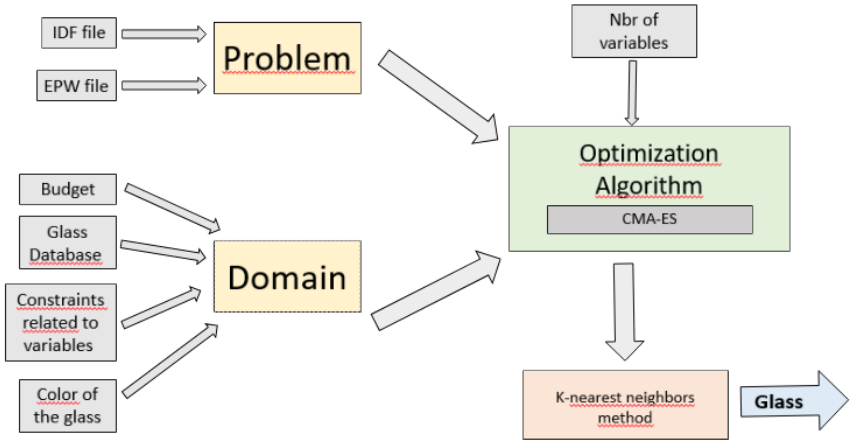


Figure 6.2: Resume of the program

To get a cleaner and more pleasant future, let's work together to suggest the right window for the right application in order to optimize the energy consumed and the comfortable interior environment.

Bibliography

[1] Gosavi Abhijit. *Simulation-Based Optimization: Parametric Optimization Techniques and Reinforcement Learning*. Norwell, MA, USA: Kluwer Academic Publishers, 2003. ISBN: 1402074549.

[2] Mark A. Abramson, Charles Audet, and John Dennis. "Generalized Pattern Search Algorithms : unconstrained and constrained cases". In: *IMA workshop – Optimization in simulation based models*. 2003.

[3] Muñoz Acosta et al. "Algorithm selection for black-box continuous optimization problems: A survey on methods and challenges". In: 317 (May 2015), pp. 224–245.

[4] Jason E. Hicken & Juan J. Alonso. *Introduction to Multidisciplinary Design Optimization*. 2012, pp. 133–161.

[5] Asma Atamna. "Analysis of Randomized Adaptive Algorithms for Black-Box Continuous Constrained Optimization". PhD thesis. Université Paris-Saclay, 2017.

[6] Anne Auger et al. "Experimental Comparisons of Derivative Free Optimization Algorithms". In: *CoRR* abs/1005.5631 (2010).

[7] Thomas Bartz-Beielstein. *A survey of model-based methods for global optimization*. SPOTSeven Lab, TH Köln, Gummersbach, Germany.

[8] Brian Birge. *A Particle SwarmOptimization (PSO) Primer*.

[9] Eric Brochu, Mike Cora, and Nando de Freitas. *A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning*. 2009.

[10] Kurt Bryan and Yosi Shibberu. *Penalty Functions and Constrained Optimization*.

[11] Messe Cedric. *Glass, Light, Energy, Colors and Standards*. AGC Glass Europe. 2017.

[12] Ion Cozac. "Shepard method - from approximation to interpolation". In: *Studia Univ. "BABE-BOLYAI", Mathematica* XLVIII (2003).

[13] P.G. Ellis et al. "Automated Multivariate Optimization Tool for Energy Analysis". In: *IBPSA SimBuild 2006 Conference Cambridge, Massachusetts* (2006).

[14] U.S. Department of Energy. *EnergyPlus Version 8.7 Documentation Engineering Reference*. 2016.

[15] Kelly Fleetwood. *An Introduction to Differential Evolution*.

[16] Glineur François. *INMA2471 - Méthodes et Modèles d'Optimisation*. Université Catholique de Louvain, École polytechnique de Louvain.

[17] Raphael T. Haftka, Diane Villanueva, and Anirban Chaudhuri. "Parallel surrogate-assisted global optimization with expensive functions – a survey". In: *Structural and Multidisciplinary Optimization* 54.1 (July 2016), pp. 3–13.

[18] Zhong-Hua Han and Ke-Shi Zhang. *Surrogate-Based Optimization*.

[19] Nikolaus Hansen. "The CMA Evolution Strategy: A Tutorial". In: *CoRR* abs/1604.00772 (2016).

[20] Thomas Hemker. "Derivative Free Surrogate Optimization for Mixed-Integer Nonlinear Black Box Problems in Engineering". PhD thesis. 2008.

[21] Michael Herrmann. *Particle Swarm Optimisation (PSO)*. 2011.

[22] Michael Kleder. *VERT2CON - vertices to constraints*. `https://nl.mathworks.com/matlabcentral/fileexchange/7895-vert2con-vertices-to-constraints?s_tid=prof_contriblnk`. 2005.

[23] Slawomir Koziel and Xin-She Yang. *Computational Optimization, Methods and Algorithms*. Springer-Verlag Berlin Heidelberg, 2011, pp. 33–59.

[24] Vu Khac Ky et al. *Surrogate-based methods for black-box optimization*. 2015.

[25] Driessen L. *Simulation-based optimization for product and process design*. 2006.

[26] Hoël Langouët. "Optimisation sans dérivées sous contraintes : deux applications industrielles en ingénierie de réservoir et en calibration des moteurs". PhD thesis. 2011.

[27] Aleksandar Lazinica. *Particle Swarm Optimization*. 2009.

[28] Ilya Loshchilov. *LM-CMA for Large Scale Black-box Optimization*. Oct. 2015.

[29] Ilya Loshchilov, Marc Schoenauer, and Michèle Sebag. *Comparison-Based Optimizers Need Comparison-Based Surrogates*. INRIA Saclay - Île-de-France, Université Paris-Sud.

[30] Wetter Michael and Wright Jonathan. "A comparison of deterministic and probabilistic optimization algorithms for nonsmooth simulation-based optimization". In: 39 (Aug. 2004), pp. 989–999.

[31] Hossein Mohammadi. "Kriging-based black-box global optimization: analysis and new algorithms". PhD thesis. École Nationale Supérieure des Mines de Saint-Étienne, 2016.

[32] Anh-Tuan Nguyen, Sigrid Reiter, and Philippe Rigo. "A review on simulation-based optimization methods applied to building performance analysis". In: *Applied Energy 113* (2014), pp. 1043–1058.

[33] Luis Miguel Rios & Nikolaos V. Sahinidis. *Derivative-free optimization: A review of algorithms and comparison of software implementations*. 2004.

[34] S. Sakata. "An efficient algorithm for Kriging approximation and optimization with large-scale sampling data". In: *Computer Methods in Applied Mechanics and Engineering* 193 (Jan. 2004), pp. 385–404.

[35] Timothy Simpson et al. "Kriging Models for Global Approximation in Simulation-Based Multidisciplinary Design Optimization". In: 39 (Dec. 2001), pp. 2233–2241.

[36] Timmermans Tanguy. *AGC Building Energy Efficiency User manual*. AGC Glass Europe. 2016.

[37] Anke Troltzsch. "An active–set trust-region method for bound-constrained nonlinear optimization without derivatives applied to noisy aerodynamic design problems". PhD thesis. Institut National Polytechnique de Toulouse, 2011.

[38] Julian (Jialiang) Wang et al. *Simulation-based optimization of window properties based on existing products*. 2016.

[39] Michael Wetter and Jonathan Wright. "Comparison of a generalized pattern search and a genetic algorithm optimization method". In: *Eighth International IBPSA Conference*. 2003.

[40] Stefan Martin Wild. *Derivative-free optimization algorithms for computationally expensive functions*. 2009.

[41] Hong Zhu and David B. Bogy. "DIRECT Algorithm and Its Application to Slider Air-Bearing Surface Optimization". In: *IEEE TRANSACTIONS ON MAGNETICS, VOL. 38*. 2002.

Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve **www.uclouvain.be/epl**