Pontificia Universidad Católica de Chile
Instituto de Ingeniería Matemática y Computacional
IMT2116 - Taller de Matemáticas Aplicadas
Profesor Alejandro Cataldo

# Capstone Final Report

Aarhus University Project

December 26, 2024

Danilo Aballay, Catalina Alegría,
Valentín Conejeros, Vittorio Salvatore
and Cristóbal Silva

# Contents

# 1 Client

The client for this project is the SustainScapes research center, part of Aarhus University in Denmark. SustainScapes specializes in developing strategies for the sustainable and ecological use of land.

# 2 Context

Denmark is a small country, with approximately 65% of its territory dedicated to agriculture (World Bank, 2021). In 2022, Denmark enacted the Nature Restoration Act, which requires that by 2030, 30% of its territory must accommodate biodiversity and 20% of degraded natural areas must be restored.

Today, Denmark is 14% below its target of designating 30% of its territory to biodiversity. This 14% must be obtained exclusively through state acquisition or leasing of farmland. Our client, SustainScapes is currently developing mathematical models aimed at optimizing potential biodiversity in Denmark by selecting agricultural plots to transform.

The optimization is based on the potential for biodiversity, evaluated according to habitat suitability and expected species assemblages across eight nature types, organized into three transformation axes:

- Forest–Dry–Rich
- Forest–Dry–Poor
- Forest–Wet–Rich
- Forest–Wet–Poor
- Open–Dry–Rich
- Open–Dry–Poor
- Open–Wet–Rich
- Open–Wet–Poor

Additionally, the optimization problem includes an extra requirement of ensuring 250,000 hectares of new forests and 140,000 hectares of wetlands. Finally, as part of the problem formulation, the client introduced a bonus for contiguity of land parcels of the same type, which covers both preexisting and proposed natural areas.

# 3 Client's Challenge

The client's challenge lies in the fact that its current linear optimization algorithm is computationally inefficient, taking an excessively long time to converge to a solution. Initially, the linear optimization model converged to an optimal solution when applied to the entirety of Danish territory. However, once the bonus for the contiguity of pre-existing natural areas was added to the objective function, the algorithm runtime increased significantly.

Currently, when applied to all of Denmark, the algorithm fails to converge within two weeks of running time, thus hindering the progress of the client's research.

# 4 Objectives and Deliverables

The principal objective of this project is to develop an optimization model or algorithm that assigns land uses and maximizes biodiversity within a reasonable amount of time, defined by the client as less than a week. The goal is to optimize the resolution time of the model so that it can provide solutions for the entire Danish territory in the order of days. It is crucial that the model continues to account for all previously established parameters, particularly the contiguity of both restored and existing natural areas.

The solution should be sufficiently general to serve as a basis for potential adaptation and application in similar scenarios, such as in other European Union countries that pursue similar conservation targets.

The expected deliverables include the following.

- **Algorithmic Improvement Proposal: :** A technical document presenting the implementations developed to improve the speed of obtaining solutions.
- **Technical Documentation** A manual detailing the usage and configuration of the model for broader, more generalized implementations.
- **Code Repository:** A GitHub repository containing the source code, potentially implemented in Python, R, Jupyter Notebook, or another programming language.

The client has indicated flexibility regarding the format of these deliverables, as long as the problem is adequately addressed.

# 5 Client's Available Data

## 5.1 Database

The client possesses a geospatial database that segments the Danish territory into cells measuring $200 \times 200$ meters, amounting to hundreds of thousands of cells. This database is available in *GeoTIFF* and *.dat* formats. Each cell contains key information about the land:

- Current land use data, as described in Section 3
- Indicators of the biodiversity present on the land, facilitating the assessment of each area's biological richness
- Phylogenetic diversity for potential land uses
- Information on area contiguity

These data were obtained from public sources. Over the course of its work, the client has also developed scripts that manage and transform the data into various formats, create visual representations, and subdivide the data into smaller geographic sectors.

## 5.2 Current Model

The client developed the following optimization model:

$$\begin{aligned}\text{maximize:} \quad & \sum_{l \in \text{Landuses}, c \in \text{Cells}} \text{Biodiversity}[l, c] \\ & + \ \text{SpatialContiguityBonus} \times \sum_{(i,j) \in E, l \in \text{Landuses}} \text{Contiguity\_score}[l, i, j]\end{aligned}$$

Where

$$\begin{aligned}\text{Biodiversity}[l, c] &= \text{LanduseDecision}[l, c] \times \text{Richness}[l, c] \\ &\quad \times \text{PhyloDiversity}[l, c] \times \text{CanChange}[c] \\ \text{Contiguity\_score}[l, i, j] &= \text{Contiguity}[l, i, j] \times \text{CanChange}[i] \times \text{CanChange}[j] \\ &\quad + \text{Existingnature}[l, i] \times \text{LanduseDecision}[l, j] \times \text{CanChange}[j]\end{aligned}$$

Subject to

$$\sum_{l \in \text{Landuses}} \text{LanduseDecision}[l, c] \leq 1 \qquad \forall c \in \text{Cells} \quad (1)$$

$$\sum_{c \in \text{Cells}} \text{LanduseDecision}[l, c] \geq \text{MinLan} \qquad \forall l \in \text{Landuses} \setminus \{'Ag'\} \quad (2)$$

$$\text{LanduseDecision}['\text{Ag}', c] = 0 \qquad \forall c \in \text{Cells} \quad (3)$$

$$\sum_{c \in \text{Cells}, l \in \text{ForestLanduses}} \text{LanduseDecision}[l, c] \geq \text{MinFor} \quad (4)$$

$$\sum_{c \in \text{Cells}, l \in \text{WetLanduses}} \text{LanduseDecision}[l, c] \geq \text{MinWet} \quad (5)$$

$$\sum_{l \in \text{Landuses}, c \in \text{Cells}} \text{LanduseDecision}[l, c] \times \text{TransitionCost}[l, c] = b \quad (6)$$

$$\text{Contiguity}[l, i, j] \leq \text{LanduseDecision}[l, i] \qquad \forall l \in \text{Landuses}, (i, j) \in E \quad (7)$$

$$\text{Contiguity}[l, i, j] \leq \text{LanduseDecision}[l, j] \qquad \forall l \in \text{Landuses}, (i, j) \in E \quad (7)$$

$$\text{LanduseDecision}[l, i] + \text{LanduseDecision}[l, j] - 1 \leq \text{Contiguity}[l, i, j] \qquad \forall l \in \text{Landuses}, (i, j) \in E \quad (7)$$

In this formulation:

- $\text{LanduseDecision}[l, c]$ is a binary variable that specifies whether the cell $c$ is allocated to land use $l$.
- $\text{Contiguity}[l, i, j]$ is a binary variable that defines whether cells $i$ and $j$ are contiguous for the same land use $l$.

Because these decisions are binary, this optimization problem is a Mixed-Integer Linear Program (MILP).

**Summary of Constraints.**

- Constraint (1) ensures that only one land use is chosen per cell.
- Constraint (2) enforces a minimum total area for each land use (except agriculture).
- Constraint (3) ensures that no cell is assigned to agriculture.
- Constraints (4) and (5) guarantee minimum forest and wetland areas, respectively.
- Constraint (6) restricts the total transition costs to the available budget $b$.
- Constraint (7) defines the contiguity relationships, which is one of the most challenging aspects to scale.

Currently, this model is implemented in AMPL. In addition, the client has documents that contain further mathematical details.

# 6 Literature Review

To address the client's problem, a bibliographic review was conducted, focusing on similar large-scale mixed-integer problems, land planning, and districting. Discussions were also held with faculty members who specialize in these areas.

Two key considerations guided our search for solution approaches. First, the objective function assigns somewhat arbitrary values to biodiversity and contiguity benefits. Second, the problem is large in scale. Consequently, our primary aim was to find a method that can rapidly produce a sufficiently good (albeit not guaranteed to be optimal) feasible solution.

After extensive review, approaches such as genetic algorithms, column generation, and hierarchical decomposition were considered. Ultimately, the selected method was *Local Search*, a heuristic recommended by faculty with expertise in combinatorial optimization. Local Search explores neighboring solutions by making small changes to a current feasible solution in order to improve the objective function. The steps are:

1. Start with a feasible solution and calculate its objective value. This becomes the best solution found so far.
2. Identify neighboring solutions to the current solution by introducing small *mutations*.
3. Evaluate these solutions according to the objective function. If a better solution is found, update the current best solution.
4. Repeat until a maximum number of iterations is reached or until no further improvement is observed.

Below is a pseudocode snippet for a typical Local Search algorithm (adapted from V. Arya et al. 2004):

**Algorithm** `Local Search.`

1. $S \leftarrow$ an arbitrary feasible solution in $\mathcal{S}$.
2. While $\exists S' \in \mathcal{B}(S)$ such that $cost(S') < cost(S)$,
   do $S \leftarrow S'$.
3. return $S$.

Figure 1: A generic Local Search algorithm

The main benefit of choosing this algorithm is that it is capable of finding **feasible** solutions that progressively improve the objective function. Additionally, it is flexible, as it allows for control over how much time is dedicated to improving the solution. Finally, it is relatively easy to implement and understand. For these reasons, this method was chosen as the approach for the land-use optimization problem. Despite the large scale of the problem and the significant number of constraints, this method has the potential to adapt easily to these conditions and find solutions quickly.

The main drawback is that this method does not, on its own, demonstrate the optimality of the provided solution. However, the algorithm does ensure that the solutions delivered are equal to or better than the initial solution. Therefore, it suffices to ensure that the method chosen to select the initial solution provides a solution close to optimality.

# 7 Solution Approach

The main goal is to provide a land-use allocation—that is, a solution to the posed optimization problem—using heuristic approaches. In this respect, the intention is to find a solution that is as close to optimal as possible (i.e., to maximize the conservation index), but to do so more rapidly than by using an exact approach. Different initial solutions, found using the methods described below, will then be improved via various Local Search implementations. It is worth noting that the aim is not to prove solution optimality but to ensure that it is relatively close to optimality.

Thus, this work seeks to identify feasible solutions to the problem that are near optimal (ideally with a gap of less than 1%) in a significantly shorter time than would be required to find the exact optimal solution.

## 7.1 Feasible Initial Solution

To apply Local Search and seek an optimal solution, it is first necessary to obtain a feasible initial solution. Two methods were devised to generate feasible initial solutions, described bellow.

### 7.1.1 Incremental Subproblems

This method emerged from the structure of the database that the client provided. They supplied smaller subproblems, each covering certain areas of Denmark, with each subproblem growing in size while encompassing the previous (area) problem. Accordingly, the incremental subproblems method consists of exactly solving a portion of the problem, then using that solution as a constraint set when exactly solving the total problem.

In the following figure, one can observe a representation of this method. The total area for which a feasible initial solution is sought is shown in gray in panel (1). A subproblem (or smaller area) is chosen to be solved exactly; in the figure, this is the area in red (2). Once this subproblem is

solved, its solution is passed as a constraint to the total problem, which is represented in yellow (3). Consequently, the remaining part of the problem to be optimized (the area in red in panel (3)) is determined.

### 7.1.2 Incremental Subproblems

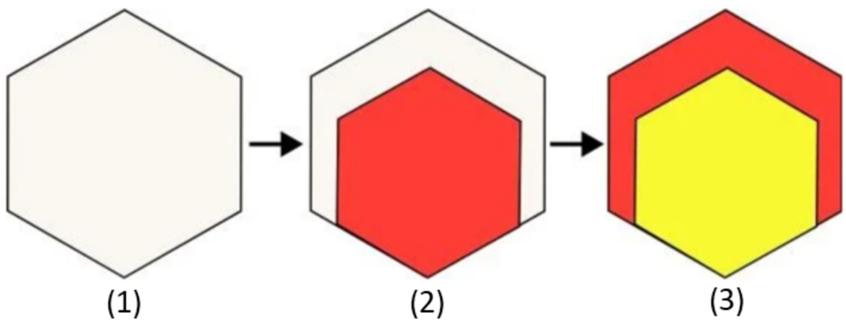



Figure 2: Representation of the incremental subproblems method

### 7.1.3 Complementary Subproblems

Continuing with the idea of splitting the area, to find a feasible initial solution for a certain region, a subproblem and its complement (i.e., the entire remaining area not part of the subproblem) are defined. Exact solutions are then computed for these two areas, and they are merged to form the solution for the entire area. Note that these two areas do not necessarily have to be of the same size.

In the following figure, a representation of the method is shown. The total area for which a feasible initial solution is sought is in gray in panel (1). The problem is split into two parts — the subproblem and its complement — as in panel (2). These two areas are solved separately (3), and the solutions are combined to form a solution for the entire area (4).

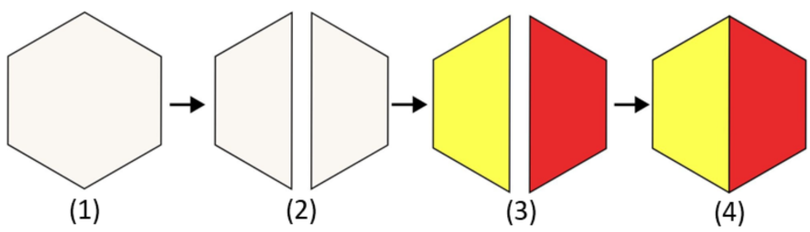



Figure 3: Representation of the complementary subproblems method

## 7.2 Local Search

We began with a very simple Local Search implementation to improve the initial solutions obtained with the previous methods. The expert faculty members suggested randomly selecting a (ratio) percentage of the cells and fixing the rest as constraints, so the solution is then re-optimized. This process is repeated as many times as necessary.

Below is a figure illustrating this first implementation. The entire area corresponds to the initial solution. A random 10% of the cells is designated as variables, while the remaining cells are fixed as constraints. The problem is then re-optimized, producing a solution that maintains or improves upon the original solution.

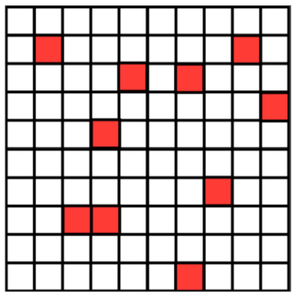


Figure 4: Representation of the first Local Search implementation

Then, the second Local Search implementation aims to select a larger neighborhood than simply individual cells to treat as variables in the optimization problem. Starting from a feasible initial solution, a Danish municipality is randomly chosen, and then a group of neighboring municipalities (a *cluster*) of a predefined size is selected around it. This size is defined by a ratio of the total number of cells, expressed as a percentage. If no more neighbors can be found (for instance, an island), another starting point is chosen, and neighboring municipalities are added to the *cluster* until the defined size is reached. Once this group is determined, the land uses outside the *cluster* are fixed (as constraints), leaving the cells within it as variables. This process is repeated until a specified amount of time elapses or a certain number of iterations has been reached.

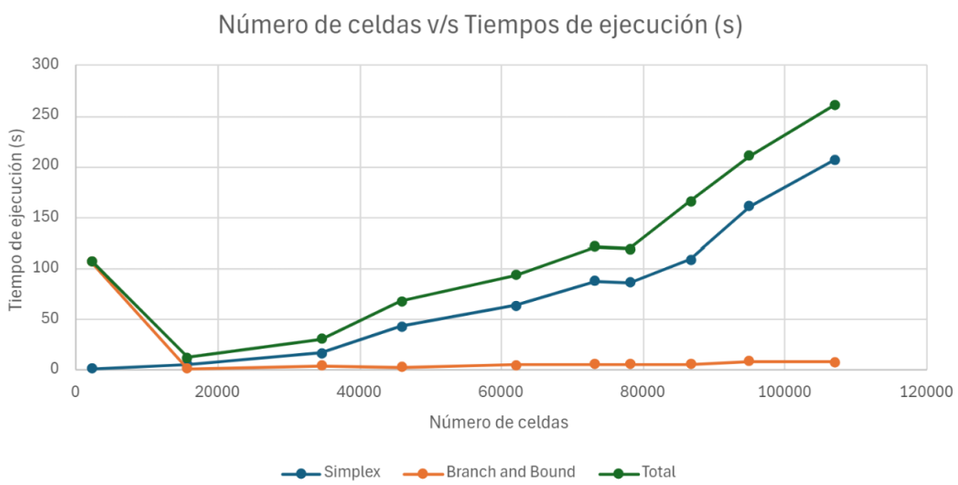
# 8 Results and Analysis

All optimization models were implemented in Python, specifically modeled and solved using `gurobipy`. It should be noted that the client built all of its models using AMPL. Additionally, for the initial tests, that is, the "small-scale cases" referenced below, a personal computer was used. For the tests involving medium-scale cases and the entirety of Denmark, all code was executed on the UC Engineering High-Performance Computing Cluster.

## 8.1 Exact Solutions

To assess how exact optimization solutions behave when solved with Gurobi, we ran a couple of tests measuring execution time and problem size. We also recorded how much of that time was dedicated to the Simplex stage versus the Branch and Bound stage.

In small cases, representing up to 10% of the total problem, we produced a graph of these times, shown below.

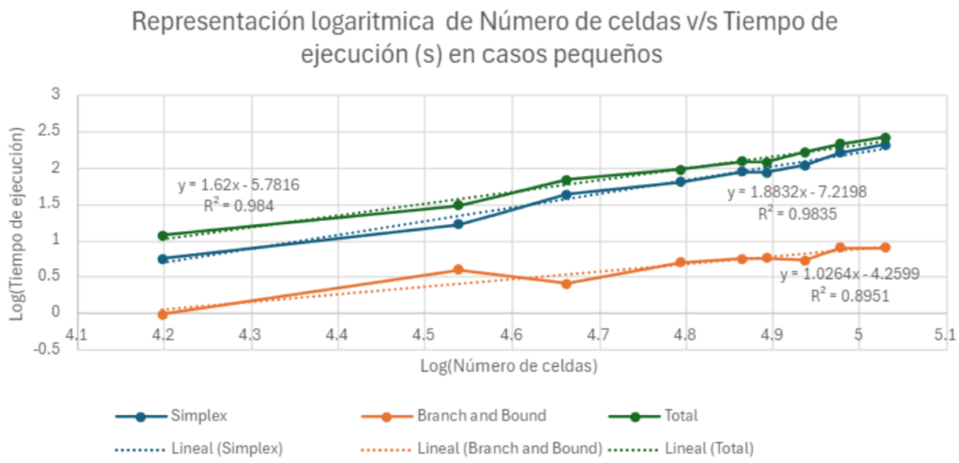



Graph 1: Number of cells in the problem vs. execution times in small-scale cases

It can be seen that in almost all cases, the Simplex stage takes longer than the Branch and Bound stage. In most cases, only one node is solved in the latter. The only case that deviates from these results is the one with the fewest cells; however, this may relate to the particular nature of that specific zone.

Below are two additional graphs showing the logarithmic representation of the number of cells versus execution time. By fitting a linear model to this graph, the slope indicates how the time

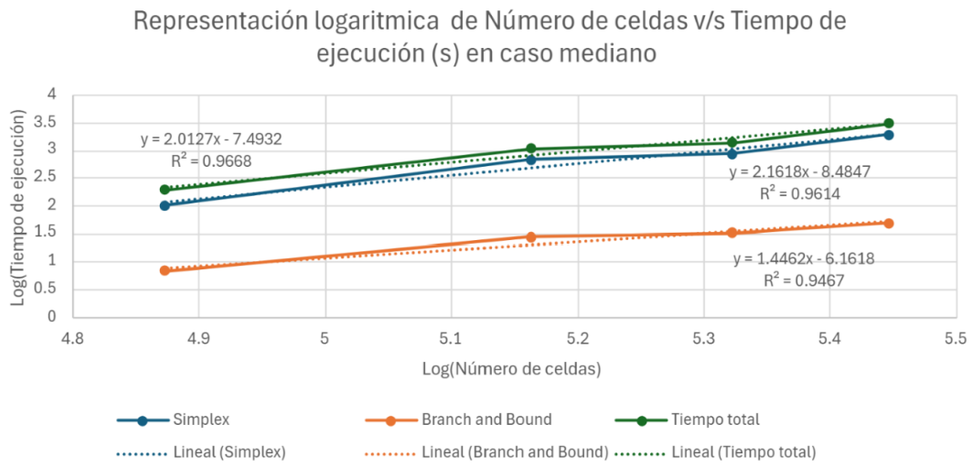
grows (linear if the slope is 1, quadratic if 2, and so on).




Graph 2: Logarithmic representation of the number of cells vs. execution time (in small-scale cases)

It can be observed that the Simplex stage execution time fits well to a linear model with an approximate slope of 1.88, implying that time increases almost quadratically as the number of cells grows. The Branch and Bound stage fits (less neatly) to a linear model with a slope of approximately 1.02, meaning that time grows in a linear manner, which matches our previous observations.

For these small cases, the number of cells ranged from about 2,000 to 100,000, i.e., less than 10% of the total problem.




Graph 3: Logarithmic representation of the number of cells vs. execution time (in medium-scale cases)

We created the same graph for medium-scale cases, ranging from about 70,000 to 300,000 cells, which corresponds to roughly 30% of the total problem (the entire territory of Denmark). It can be confirmed that Simplex time still grows nearly quadratically. In this scenario, the linear fit for the Branch and Bound stage has a slightly higher slope (1.44), meaning that the total execution time fits quadratic growth more accurately as the problem size increases.

Thus, for subsequent applications, as a reference point, we assume that when solving the optimization problem exactly, execution time grows quadratically with respect to problem size.

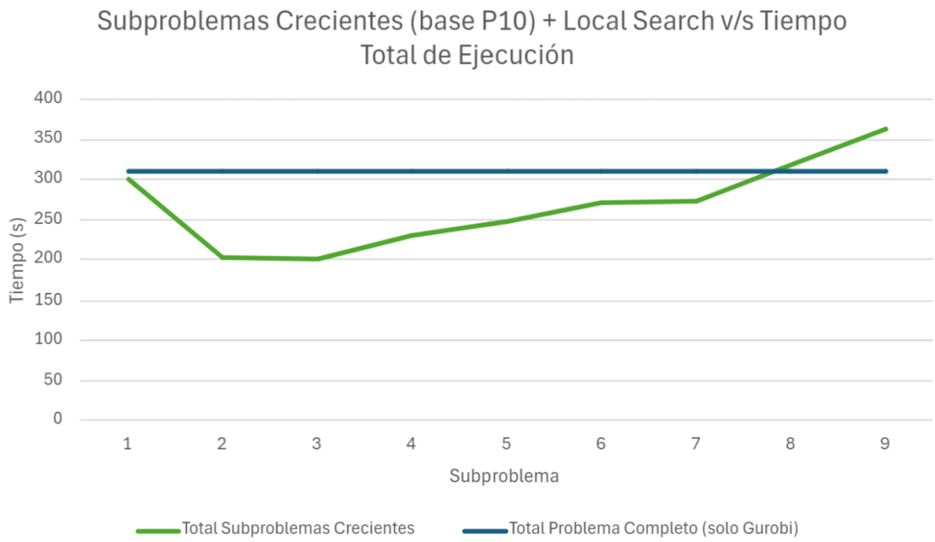
Although it was known that the client was unable to obtain a solution for all of Denmark in a reasonable time (under two weeks), we decided to run the exact optimization for the entire problem. The cluster produced an exact solution for Denmark in 20,342.95 seconds, or about 5.6 hours. This outcome provides concrete evidence of how "good" our chosen method for finding a feasible initial solution is, especially because optimality gap can now be calculated precisely.

## 8.2 Feasible Initial Solutions

To find feasible initial solutions, we began by testing the two methods explained above (incremental subproblems and complementary subproblems) on small-scale cases (the same regions solved exactly in the previous section).

In the database we used for the "small-scale cases," there are 10 areas, each of which is larger than the previous one. Problem 1 is the smallest, Problem 2 includes Problem 1 and is therefore bigger, and so forth, up to Problem 10.

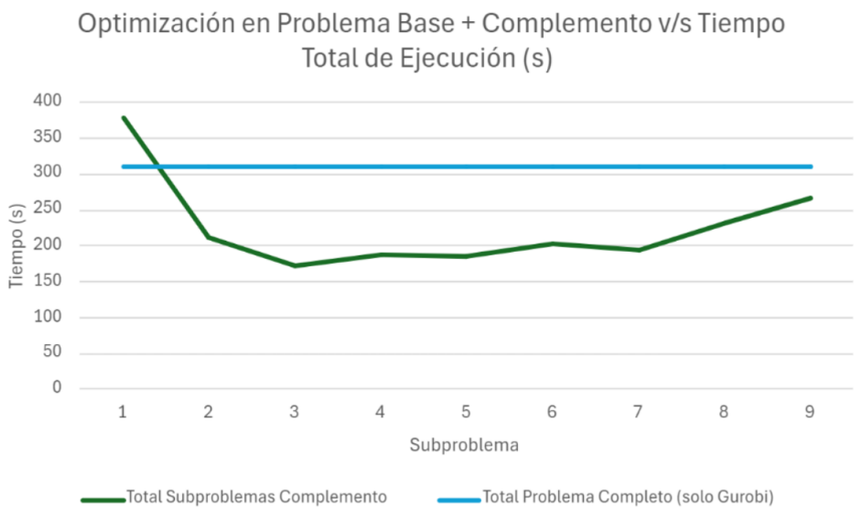
With that in mind, for the incremental subproblems method, we solved each problem exactly and then added that solution as a constraint to the largest problem (Problem 10). The following graph displays the execution time for this method (exact solution of the smaller problem + exact solution of the full problem with added constraints) compared to solving the largest problem exactly.




Graph 4: Total execution time for incremental subproblems

It is apparent that, overall, the heuristic method runs faster than the exact solution, except when the smaller subproblem to solve is too large or almost the same size as the entire problem.

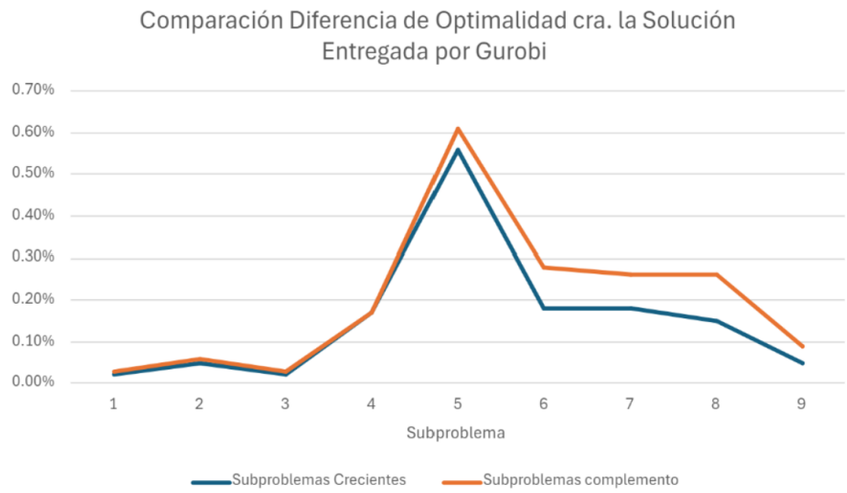
We produced the same type of graph using the same database for the complementary subproblems method. In this case, one subproblem and its complement are solved exactly, considering Problem 10 as the total problem.




Graph 5: Total execution time for complementary subproblems

Again, in most instances, the heuristic method is faster than the exact solution, except for Subproblem 1. Yet this difference can be explained by the earlier anomaly in which this particular problem area behaved differently.

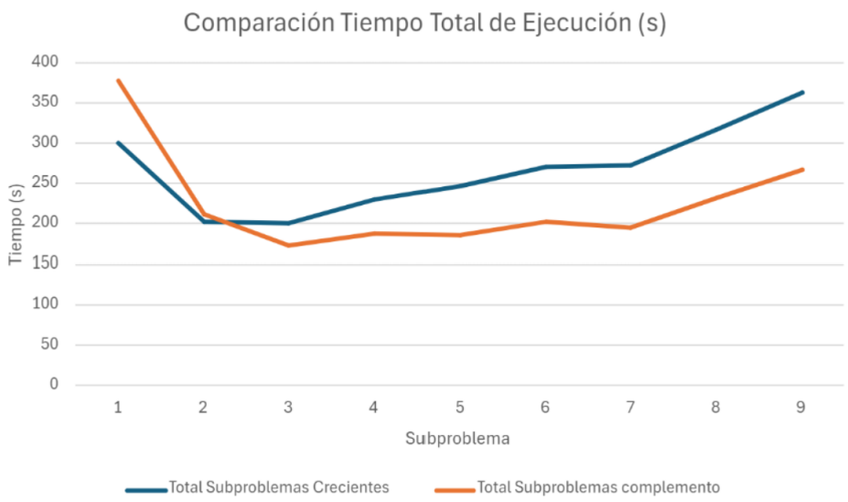
Below is a comparison of the two heuristic methods and the optimality gap of the solutions they produce relative to the exact solution of the problem.




Graph 6: Comparison of the optimality gap of the two heuristic methods

You can see that the incremental subproblems method is slightly better in terms of optimality compared to the other method. Even so, neither method yields an optimality gap greater than 0.61%. This means that they produce sufficiently good feasible initial solutions.

As a summary, the following figure compares the execution times of both methods.




Graph 7: Comparison of execution times for the two methods

It is clear here that the complementary subproblems method is faster in most cases than the other method.

Therefore, given that the difference in optimality between the two methods is not very large, but the complementary subproblems method is faster, we decided to use this method to find a feasible initial solution for the complete problem. It was also chosen because of its potential to parallelize processes, especially for obtaining exact solutions for the areas. This can lead to obtaining initial solutions quickly if the problem is divided into more than two sections. Although this method does not guarantee contiguity on the boundaries between areas, Local Search can help address such issues.

Below is the feasible initial solution obtained with the chosen method, complementary subproblems. The optimization problem was solved exactly for the four Danish regions: Midtjylland, Nordjylland, Sjaelland, and Syddanmark.

Table 1: Execution times for the feasible initial solution

| **Region** | Midtjylland | Nordjylland | Sjælland | Syddanmark | **Total** |
|---|---|---|---|---|---|
| **Time (s)** | 2454.90 | 1123.93 | 2180.60 | 2824.04 | 8583.47 |

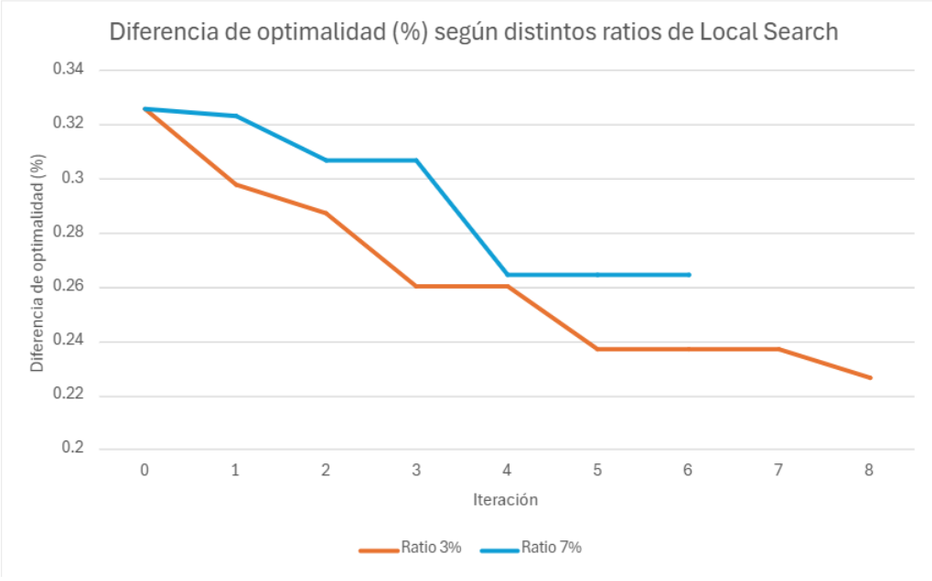
| **Description** | Solution Evaluation | Total Time | Denmark |
|---|---|---|---|
| **Time (s)** | 1279.21 | 9862.68 | 20342.95 |

It is noteworthy that the total time for obtaining the feasible solution by solving the regions sequentially (8583.47 s) is significantly less than that required for the exact solution of Denmark. In particular, it takes just under half the time to solve.

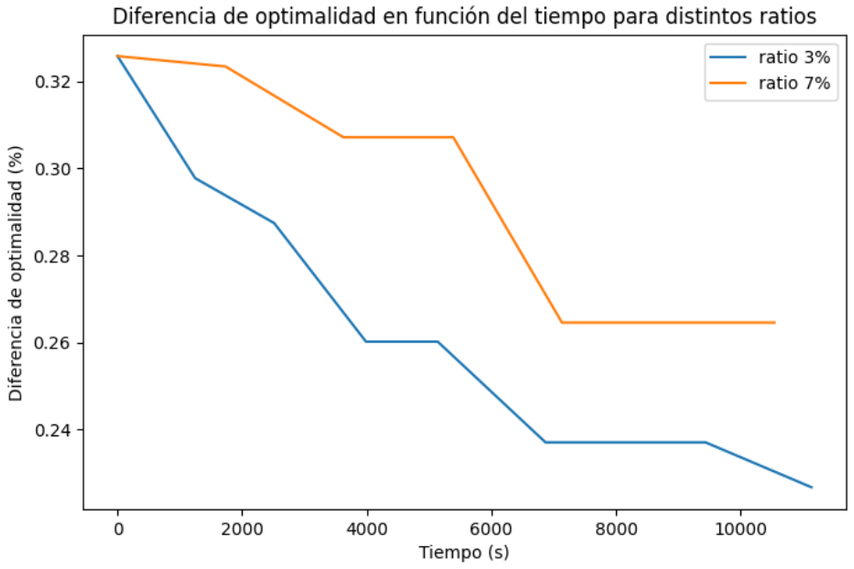
We also calculated the optimality gap of this solution compared to the exact solution, which is 0.33%. Although this gap is small, Local Search can be used to improve it.

## 8.3 Local Search

Two Local Search trials were carried out, with ratios of 3% and 7%. This means that a municipality is chosen, and then "neighboring" municipalities (as described previously) are selected until a *cluster* of more than 3% or 7% of the total problem is obtained. The following figure shows how the optimality gap evolves as the algorithm progresses.




Graph 8: Optimality gap (%) with different Local Search ratios




Graph 9: Optimality gap as a function of time for different ratios

It is clear that in both cases, Local Search successfully improved the initial solution and yielded a more optimal result.

Moreover, one can see that the Local Search implementation achieved a smaller optimality gap with a 3% ratio than with 7% (over similar time intervals). Also noticeable is that in some iterations, the optimality gap does not change, i.e., there is no alteration in the cells. This is especially apparent after the fourth or fifth iteration.

When the ratio is 3%, each iteration takes less time, allowing for more iterations within the same time period. Additionally, the performance difference when improving the initial solution is not only influenced by the parameter value; it also depends on which municipalities are chosen to

create the *cluster* in each iteration (since that selection is random). More extensive tests are therefore needed to determine the parameter's optimal value.

As a summary, the following table provides combined execution and import times.

Table 2: Summary of statistics for the Local Search implementation

| Metric | Ratio 3% | Ratio 7% |
|---|---|---|
| **Import Time (s)** | 1089.012 | 1129.209 |
| **Total Local Search Time (s)** | 11413.485 | 10766.138 |
| **Number of Iterations** | 8 | 6 |
| **Initial Optimality Gap** | 0.326 | 0.326 |
| **Final Optimality Gap** | 0.227 | 0.265 |

## 8.4 Parallelization

Two types of parallelization were used to improve the performance of the algorithms and methods implemented. First, a *Multiprocessing* strategy was employed to solve different subproblems in parallel. Secondly, *Multithreading* was used to enhance the solution process for each subproblem.

## 8.5 *Multiprocessing*

The initial solution process was parallelized under the multiprocessing strategy. Since it is based on solving disconnected and independent problems, our implementation solves them in parallel, leading to the total time for obtaining an initial solution being that of the subproblem that takes the longest, which is much smaller than the original problem. Notably, this implementation automates the process of detecting the number of computing cores and determining the maximum number of problems that can be solved simultaneously; it is also flexible regarding the number of subproblems and cores.

To obtain the same feasible initial solution described in Section 8.2, the four Danish regions were solved in parallel. The region that took the longest required 2573 seconds (approximately 43 minutes) to solve. This result coincides with our expectations based on solving the regions without parallelizing (Table 1). Thus, the initial solution was found three times faster than the non-parallelized algorithm. Moreover, the method developed finds a feasible solution in significantly less time than the original solution process (5.6 hours). Specifically, it amounts to 12.5% of the original time.

## 8.6 *Multithreading*

Gurobi itself takes advantage of parallelization under the *multithreading* paradigm by means of concurrent optimization, namely solving in parallel with *threads* using two methods: Simplex and Dual Simplex. Once one of them finds the optimal solution, the process halts. Unfortunately, the Simplex method does not utilize multiple *threads*, so its performance is not improved by *Multithreading*.

# 9 Conclusions

In conclusion, the two methods developed to find feasible initial solutions successfully reduce execution time. In particular, a feasible land-use allocation solution was found for all of Denmark through the complementary subproblems method. This solution is derived from solving the four Danish regions (Midtjylland, Nordjylland, Sjaelland, and Syddanmark) exactly. When these four areas are solved sequentially without parallelization, the total time to obtain the solution is less than half the time required for the exact solution for all of Denmark. Specifically, the non-parallelized method takes 2.38 hours, while the exact solution takes 5.6 hours. When the exact solution processes for the four areas are parallelized, it takes about 0.71 hours, or roughly 43 minutes. This means our method finds an initial solution in one-eighth the time it takes to solve the problem exactly, with an optimality gap of only 0.33%.

Although the solution obtained via the heuristic method does not guarantee contiguity criteria are met, the Local Search procedure was able to improve this aspect. In the best case, the optimality gap dropped to 0.22% within three hours (eight iterations).

The Local Search method allows users to customize the balance between the time spent improving the solution and the level of optimality. Thus, the algorithm can be configured to suit particular needs.

On the other hand, after five iterations, the optimality gap did not change significantly. This may be due to the small margin between the solution and the true optimum. If the feasible initial solution had been farther from the optimum, the optimality gap changes between Local Search iterations would probably have been larger.

In conclusion, through the heuristic methods developed in this work, a solution to the land-use problem in line with the new Biodiversity Law was found. Although the solution does not demonstrate optimality, the time to obtain it was reduced to 12.5% of the original, with only a 0.33% optimality gap compared to the exact solution. Furthermore, this (or other) feasible initial solutions can be improved by means of the Local Search implementation, which uses Danish municipalities as neighborhoods.

Finally, the methods and code implemented in this study provide considerable flexibility for modifying the algorithm. The initial solution search allows subdividing the problem into different numbers of subproblems and solving them efficiently. The Local Search implementation supports changing search parameters, in addition to selecting a desired balance between speed and optimality. Lastly, the methods presented are generalizable to other similar problems and are not limited to Denmark, enabling further research in this domain.

# 10 References

Gharaibeh, A. A., Ali, M. H., Abo-Hammour, Z. S., & Al Saaideh, M. (2021). Improving Genetic Algorithms for Optimal Land-Use Allocation. Journal of Urban Planning and Development, 147(4), 04021049. https://doi.org/10.1061/(ASCE)UP.1943-5444.0000744

Kaim, A., Cord, A. F., & Volk, M. (2018). A review of multi-criteria optimization techniques for agricultural land use allocation. Environmental Modelling & Software, 105, 79–93. https://doi.org/10.1016/j.envsoft.2018.03.031

Li, X., & Parrott, L. (2016). An improved Genetic Algorithm for spatial optimization of multi-objective and multi-site land use allocation. Computers, Environment and Urban Systems, 59, 184–194. https://doi.org/10.1016/j.compenvurbsys.2016.07.002

Liu, Y., Yuan, M., He, J., & Liu, Y. (2015). Regional land-use allocation with a spatially explicit genetic algorithm. Landscape and Ecological Engineering, 11(1), 209–219. https://doi.org/10.1007/s11355-014-0267-6

M. Chaieb, J. Jemai and K. Mellouli,"A four-decomposition strategies for hierarchically modeling combinatorial optimization problems: framework, conditions and relations," 2015 International Conference on High Performance Computing & Simulation (HPCS), Amsterdam, Netherlands, 2015, pp. 491-498, doi: 10.1109/HPCSim.2015.7237081.

Ministry of Environment of Denmark. (2022). Nature Conservation Act (No. 1392 of 2022). Retrieved from http://www.retsinformation.dk

Pablo Andrés Maya. (2008, December). Column Generation Algorithm: A revision from its application to the Student Assignation Proble. ResearchGate; Universidad de Antioquia. https://www.researchgate.net/publication/262463088_Column_Generation_Algorithm_A_revision_from_its_application_to_the_Student_Assignation_Proble

Porta, J., Parapar, J., Doallo, R., Rivera, F. F., Santé, I., & Crecente, R. (2013). High performance genetic algorithm for land use planning. Computers, Environment and Urban Systems, 37, 45–58. https://doi.org/10.1016/j.compenvurbsys.2012.05.003

Schwaab, J., Deb, K., Goodman, E., Lautenbach, S., van Strien, M. J., & Grêt-Regamey, A. (2018). Improving the performance of genetic algorithms for land-use allocation problems. International Journal of Geographical Information Science, 32(5), 907–930. https://doi.org/10.1080/13658816.2017.1419249

V. Arya, N. Garg, R. Khandekar, A. Meyerson, K. Munagala, V. Pandit, (2004): Local Search Heuristics for k-Median and Facility Location Problems, SIAM Journal of Computing 33(3).

World Bank. (2021). World Development Indicators. The World Bank. https://databank.worldbank.org/source/world-development-indicators

# 11 Annexes

## 11.1 Definition of the variables used in the optimization model

Sets and parameters:

- **Cells:** Set of agricultural cells.
- **Landuses:** Set of land uses.
- **ForestLanduses:** Subset of land uses that are forests.
- **WetLanduses:** Subset of land uses that are wetlands.
- **E:** Set of pairs of adjacent cells $(i, j)$ indicating potential contiguity.
- **Existingnature[Landuses, Cells]:** Indicates whether a cell currently has a specific nature type.
- **Richness[Landuses, Cells]:** Biodiversity value (species richness) for each cell and land use type.
- **PhyloDiversity[Landuses, Cells]:** Phylogenetic diversity for each land use and cell.
- **TransitionCost[Landuses, Cells]:** Cost of transforming an agricultural cell into a specific land use type.
- **CanChange[Cells]:** Indicates whether a cell can be changed (i.e., is agricultural).
- **b:** Budget constraint for transition costs.
- **MinFor:** Minimum required area for new forests.
- **MinWet:** Minimum required area for new wetlands.
- **MinLan:** Minimum required area for each land use except agriculture.
- **SpatialContiguityBonus:** Bonus for creating contiguous areas of the same land use.

Variables:

- **LanduseDecision[$l$ in Landuses, $c$ in Cells]:** Binary variable indicating whether land use $l$ is selected for cell $c$.
- **Contiguity[$l$ in Landuses, $(i, j)$ in $E$]:** Binary variable indicating whether cells $i$ and $j$ are contiguous for land use $l$.