

Principled Modeling of the Google Hash Code Problems for Meta-Heuristics

Pedro Rodrigues

Alexandre D. Jesus

Carlos M. Fonseca

September 2023

University of Coimbra

Department of Informatics Engineering

Outline

Introduction

Background

Principled Modeling Framework

Google Hash Code Competition

Optimize a Data Center Problem

Book Scanning Problem

Conclusion & Future Work

Introduction

Combinatorial Optimization problems that often emerge in real-world scenarios:

- Are **large** and **complex**.
- **Cannot be solved** by **exact** methods in a **reasonable amount of time**.

The **Google Hash Code** programming competition invited people to solve challenging problems.

- **Complex problems** inspired by engineering challenges.
- Inspired by **real-world** scenarios.
- **Well formulated** and can be solved to some extent in a short amount of time.

Meta-Heuristics are methods that “guide and intelligently combine subordinate heuristics for exploring and exploiting solutions in the search space” (Osman and Laporte, 1996).

These methods have interesting properties:

- Can often **find “good” solutions** for hard problems **quickly**.
- **General-purpose**, i.e. they can be applied to many problems.

There is a community interest in studying them and making them more accessible in practice.

Motivation

Modeling is the process of creating a simplified representation or approximation of a real-world system, process, or phenomenon.

If this process is done in a principled way, it can be standardized in order to provide:

- A **structured approach** to problem-solving.
- A clear **separation** between **problems** and **solvers**.



Figure 1: Principled Modeling Framework

The main research questions we outline for this work are:

1. Can existing ideas on modeling frameworks (Fonseca, 2021; Outeiro, 2021; Vieira, 2009) be formalized into a more practical and complete implementation?
2. Can general-purpose meta-heuristic solvers be developed through the principled modelling framework implementation?
3. Can Google Hash Code problems be solved effectively using this modelling approach?

The main contributions of this work are related to the aforementioned research questions, as follows:

1. A practical Python implementation of the principled modelling framework was created.
2. Several meta-heuristic solvers and utilities were developed on top of the principled modelling framework implementation.
3. Several models were developed for two of the Google Hash Code problems.

Background

Optimization Problem

An optimization problem is a tuple (\mathcal{S}, f) , where \mathcal{S} is a set containing all feasible solutions, and f is an objective (cost) function, with a mapping such that:

$$f: \mathcal{S} \rightarrow \mathbb{R}$$

Combinatorial Optimization Problem

A combinatorial optimization problem is an optimization problem where the set \mathcal{S} of feasible solutions is finite.

In this work we assume without loss of generality a maximizing objective function.

Ground Set

The ground set of a CO problem is a finite set of components $\mathcal{G} = \{c_1, c_2, \dots, c_k\}$, such that every solution to the problem, feasible or not, can be defined as a subset of \mathcal{G} .

Empty Solution

A solution $s \in 2^{\mathcal{G}}$, where $2^{\mathcal{G}}$ denotes the powerset of \mathcal{G} , is said to be an empty solution if $s = \emptyset$.

Partial Solution

A solution $s \in 2^{\mathcal{G}}$ is said to be a partial solution if there is a feasible solution $s' \in \mathcal{S}$ such that $s' \supseteq s$.

Complete Solution

A feasible solution $s \in \mathcal{S}$ is said to be a complete solution if there is no feasible solution $s' \in \mathcal{S}$ such that $s' \supset s$.

The usage of **bounds**, particularly the *upper bound* in a maximization setting, is beneficial as it aids in:

- The evaluation of infeasible solutions.
- The measurement of a candidate solution's potential.

Upper Bound

An upper bound of a (partial) solution $s \in 2^{\mathcal{G}}$ is any numeric value given by a function $\Phi_{\text{ub}}: 2^{\mathcal{G}} \rightarrow \mathbb{R}$ such that:

$$\forall s' \in \mathcal{S} \wedge s' \supseteq s: f(s') \leq \Phi_{\text{ub}}(s)$$

Constructive Search is a procedure for optimization that operates as follows:

- Initiate the process with an empty or partial solution
- Add a component from the ground set to the solution.
- Repeat the process until no more (feasible) components are available.

Algorithm 1: Constructive Search Procedure

Input : Ground Set (\mathcal{G})

Output: Solution (s)

$s \leftarrow \emptyset$

$\mathcal{C} \leftarrow \{ c \in \mathcal{G} \setminus s \mid s \cup \{c\} \text{ is feasible} \}$

while $\mathcal{C} \neq \emptyset$ **do**

$c \leftarrow \text{SelectComponent}(\mathcal{C})$

$s \leftarrow s \cup \{c\}$

$\mathcal{C} \leftarrow \{ c \in \mathcal{G} \setminus s \mid s \cup \{c\} \text{ is feasible} \}$

end

return s

Local Search

Local Search is a procedure for optimization that operates as follows:

- Start with a feasible solution to the problem.
- Apply local moves and perturbations to the solution, thus exploring candidate solutions in the neighborhood.
- Repeat the process until the solution cannot be further improved.

Algorithm 2: Local Search Procedure

Input : Solution (s)

Output: Solution (s)

$\mathcal{M} \leftarrow \text{LocalMoves}(s)$

while $\mathcal{M} \neq \emptyset$ **do**

$m \leftarrow \text{SelectLocalMove}(\mathcal{M})$

$s \leftarrow \text{Step}(s, m)$

$s \leftarrow \text{Perturb}(s)$ ▷ Optional

$\mathcal{M} \leftarrow \text{LocalMoves}(s)$

end

return s

The following *meta-heuristics*, were analyzed and implemented in the context of this work.

- *Beam Search*
- *Iterated Greedy*
- *Greedy Randomized Adaptive Search Procedure*
- *Ant Colony Optimization*
- *Hill-Climbing*
- *Iterated Local Search*
- *Simulated Annealing*
- *Tabu Search*

Principled Modeling Framework

The following frameworks served as **references** for our implementation.

POF Python Optimization Framework (Vieira, 2009)

nasf4nio Not Another Software Framework for Nature-Inspired Optimization (Fonseca, 2021)

nasf4nio-cs Not Another Software Framework for Nature-Inspired Optimization — Constructive Search (Outeiro, 2021)

In the context of **modeling** for *meta-heuristics*, the following aspects are considered relevant:

- *Problem Instance*
- *Solution*
- *Construction Rules*
- *Objective Function & Bounds*
- *Combinatorial Structure (Components)*
- *Neighborhood Structure (Local Moves)*

The following **data structures** constitute the foundation of the framework:

Problem Holds immutable data that fully characterizes the problem instance.

Solution Characterizes a solution for the problem and serves as the mutable state that the solver can modify during the optimization process.

Component Characterizes any component within the ground set of a given problem.

LocalMove Characterizes any local move that can be applied to a solution

Notably, these are implemented as **Python classes**.

The following **operations** are provided in an interface between the model and the solvers.

- Problem
 - Create Empty Solution (`empty_solution`)
- Solution
 - Enumerate Components (`add_moves`, `heuristic_add_move`, ...)
 - Enumerate Local Moves (`local_moves`, ...)
 - Apply Component (`add`, `remove`)
 - Apply Local Move (`step`)
 - Perturb Solution (`perturb`)
 - Inspect Solution (`objective`, `upper_bound`)
- Component
 - Get Identifier (`id`)

These operations encompass a variety of **methods** present in each of the aforementioned classes.

Solvers call upon the operations defined in the specification in order to find solutions.

Heuristic Construction Solver Implementation

```
def heuristic_construction(problem: Problem) → Solution:
    solution = problem.empty_solution()
    c = solution.heuristic_add_move()
    while c is not None:
        solution.add(c)
        c = solution.heuristic_add_move()
    return solution
```

Google Hash Code Competition

Attendance

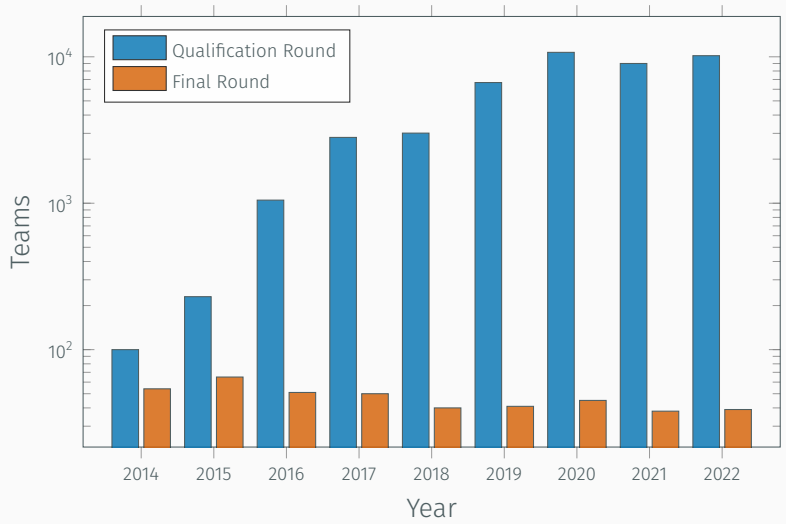


Figure 2: Google Hash Code Competition Attendance 2014-2022

Problems

To better understand the Google Hash Code problems a comprehensive analysis was conducted to understand the key properties and similarities with well-known literature problems.

Problem	Categories						
	Assignment	Knapsack	Coverage	Vehicle Routing	Simulation	Scheduling	Packing
Street View Routing			✓	✓			
Optimize a Data Center	✓	✓					
Loon			✓	✓	✓		
Delivery				✓			
Satellites	✓		✓		✓		
Streaming Videos	✓	✓					
Router Placement			✓				
Self-Driving Rides	✓			✓	✓		
City Plan							✓
Photo Slideshow						✓	
Compiling Google						✓	
Book Scanning	✓	✓	✓			✓	
Assembling Smartphones	✓					✓	
Traffic Signaling					✓		
Software Engineering at Scale						✓	
Mentorship and Teamwork						✓	
Santa Tracker				✓			

Table 1: Categorization of Google Hash Code Problems

Optimize a Data Center Problem

This problem entails optimizing the placement of servers in a data center.

- The data center is modeled as a series of rows, each containing several slots in which servers can be placed.
- Certain slots may be unavailable due to other installations within the data center.
- Servers must be logically assigned to pools contributing with their (computing) capacity.

Example — Data Center

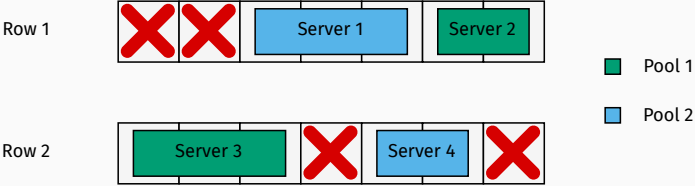


Figure 3: Example Data Center Layout

Server	Size	Capacity
1	3	2
2	2	5
3	3	10
4	2	3

Table 2: Server Properties

Guaranteed Capacity

The guaranteed capacity is a measure of the remaining computing capacity available for a given pool in the event that at most one arbitrary row of the data center becomes inoperable.

$$gc_p(x) = \sum_{m=1}^{\mathcal{M}} \sum_{r=1}^{\mathcal{R}} \sum_{i \in \mathcal{I}^r} c_m \cdot x_{m,p,i} - \max_{r=1}^{\mathcal{R}} \sum_{m=1}^{\mathcal{M}} \sum_{i \in \mathcal{I}^r} c_m \cdot x_{m,p,i}$$

Where,

- \mathcal{M} is the number of servers.
- \mathcal{R} is the number of rows.
- \mathcal{I}^r is a set containing the segment IDs for a given row r .
- c_m is the computing capacity of the server m .
- $x_{m,p,i}$ is a binary variable indicating whether the server, m , is assigned (1) or not (0) to pool, p , and segment i .

Example — Guaranteed Capacity

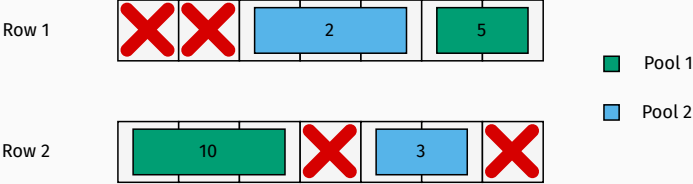


Figure 4: Example Data Center Layout (Capacities Only)

Pool	Row 1	Row 2	Guaranteed Capacity
1	5	10	5
2	2	3	2

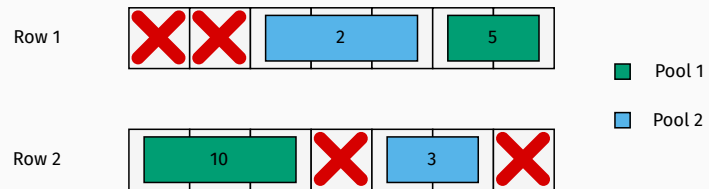
Table 3: Guaranteed Capacities

$$\begin{aligned} \max f(x) &= \min_{p=1}^{\mathcal{P}} g_{C_p}(x) \\ \text{s.t. } \sum_{p=1}^{\mathcal{P}} \sum_{r=1}^{\mathcal{R}} \sum_{i \in \mathcal{I}^r} x_{m,p,i} &\leq 1 \quad \forall m = 1, \dots, \mathcal{M} \\ \sum_{m=1}^{\mathcal{M}} \sum_{p=1}^{\mathcal{P}} \ell_m \cdot x_{m,p,i} &\leq \mathcal{L}_i \quad \forall i = 1, \dots, \mathcal{I} \\ x &\in \{0, 1\}^{\mathcal{M} \times \mathcal{P} \times \mathcal{R}} \end{aligned}$$

Where,

- \mathcal{P} is the number of pools
- \mathcal{I} is the number of segments
- \mathcal{L}_i is the size of segment i
- ℓ_m is the size of the server m .

Example — Objective



Pool	Row 1	Row 2	Guaranteed Capacity	$f(x)$
1	5	10	5	2
2	2	3	2	

Table 4: Guaranteed Capacities & Objective Value

The **problem** is characterized by the set of available servers, along with their respective sizes and capacities, the number of resource pools to be created, and the set of segments.

A **solution** is characterized by a collection of server assignments to pools and segments.

A **component** is characterized by a tuple containing a server a pool and a segment or just a server (indicating that is forbidden from being assigned).

Three component enumeration strategies were considered:

- **Standard:** Enumerate all feasible combinations of assignments of servers to pools and segments.
- **Sequential:** For one server at a time enumerate all feasible assignments to pools and segments.
- **Heuristic:** Select an order for the servers pools and segments and enumerate all possible assignments following that order.

Notably, the component associated with the action of forbidding a server is always feasible in any enumeration.

The first upper bound devised for this problem is calculated in through the following steps:

1. **Relax Segments Constraint:** Remove one row and treat the total available slots in all other rows as a knapsack.
2. **Optimize Computing Capacity:** Find the best computing capacity achievable by placing servers in this knapsack.
3. **Estimate Guaranteed Capacity:** Divide the capacity by the total number of pools for an optimistic guaranteed capacity estimate.
4. **Apply Correction:** If this estimate exceeds the capacity already assigned to any pool.
5. **Iterate and Determine Upper Bound:** Repeat these steps for all remaining rows, and the upper bound is the minimum value for the guaranteed capacity obtained through this process.

Modeling — Upper Bound

The second upper bound is a slight variation of the previous bound where instead of selecting the minimum value in each iteration, a **vector containing the minimum value for each pool** is kept.

The **upper bound** value is given by this **vector sorted in increasing order**.

Notably, the objective function must be updated where a sorted vector containing all the guaranteed capacities serves as the objective value.

Pool	Row 1	Row 2	Guaranteed Capacity	$f(x)$
1	5	10	5	(2, 5)
2	2	3	2	

Table 5: Guaranteed Capacities & Objective Value

The **local moves** considered for this problem are as follows:

1. **Assigning a server** to a pool and segment if there is an available one.
2. **Removing a server** from a segment.
3. **Changing the segment** to which a particular server is allocated.
4. **Changing the pool** to which a server is allocated, moving it to a different pool.
5. **Swapping the pools** of two assigned servers.
6. **Swapping the segments** of two assigned servers.

We were able to achieve a objective value of 386 on the single instance that was available through a **simple constructive meta-heuristic**.

Then an **Iterated Local Search** meta-heuristic was applied to further exploit this solution. The objective value obtained through this process was 410, which would have placed us in **1st place** on the competition leaderboard.

Notably, the best known objective value is 408.

Book Scanning Problem

This problem entails the setup of scanning pipeline for books.

- There are libraries housing various books. Before any scanning can commence, each library needs to register for the scanning process. Once registered, each library is allowed to scan a certain number of books daily, until global deadline.
- Only one library can undergo the sign-up process at any given time.
- Each book, when scanned, contributes a specific score.

Example — Book Scanning

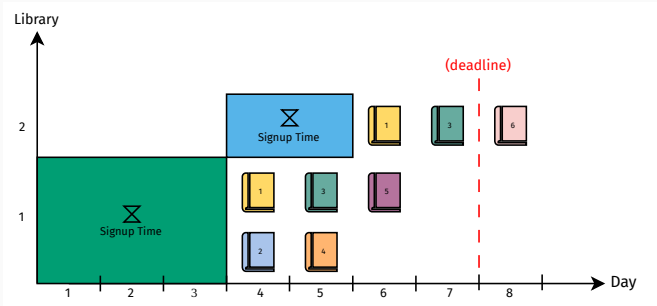


Figure 5: Book Scanning Example

Book	1	2	3	4	5	6
Score	3	1	5	4	7	1

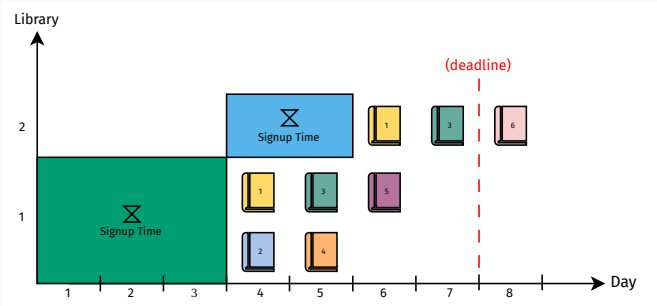
Table 6: Book Scores

$$\begin{aligned} \max f(x) &= \sum_{b=1}^{\mathcal{B}} s_b \cdot \min \left(\sum_{i=1}^{\mathcal{I}} x_{b, \phi_i^{\mathcal{I}}}, 1 \right) \\ \text{s.t. } \sum_{b=1}^{\mathcal{B}} x_{b, \phi_i^{\mathcal{I}}} &\leq r_i \cdot \left(\mathcal{D} - \sum_{k=1}^i t_{\phi_k^{\mathcal{I}}} \right) \quad \forall i = 1, \dots, \mathcal{I} \\ \sum_{i=1}^{\mathcal{I}} t_{\phi_i^{\mathcal{I}}} &\leq \mathcal{D} \end{aligned}$$

Where,

- \mathcal{B} is the number of books.
- \mathcal{D} is the global deadline.
- \mathcal{I} and $\phi^{\mathcal{I}}$ are the number and order of signed-up libraries.
- t_i and r_i are the sign-up time and book scanning rate of library i .
- s_b is the score of book b .
- $x_{b,i}$ is binary variable indicating if a book, b , is assigned (1) or not (0) to library i .

Example — Objective



Book	1	2	3	4	5	6
Score	3	1	5	4	7	1

$$f(x) = 3 + 1 + 5 + 4 + 7 = 20$$

Table 7: Book Scores & Objective Value

The **problem** is characterized by the set of all libraries available, along with their sign-up times, book shipping rate, and list of books that can be shipped, the scores for all the books, and the deadline.

A **solution** is characterized by a collection of assignments of books to libraries and the order in which libraries are scanned.

A **component** is characterized by a tuple containing a book and a library or a tuple containing two libraries (denoting the sign-up order).

Two component enumeration strategies were considered:

- **Standard:** Enumerate all the books that can be scanned by all signed-up libraries as well as all libraries that can be signed-up until the deadline.
- **Sequential:** Enumerate only the books of the last signed-up library and all the libraries that can still be signed-up.

The upper bounds were devised for this problem are as follows:

1. **Individual Library Knapsacks:** In this approach, we treat the number of books each library can scan before the deadline as an individual knapsack. The bound value for each library is calculated as the sum of scores from the best books that are yet to be scanned. The global upper bound is then determined by summing up the bound values for each library.
2. **Combined Knapsack for All Libraries:** Consider a single knapsack representing the total number of books that can be scanned by all libraries combined until the deadline.

The **local moves** considered for this problem are:

1. **Adding a book** to the set of books to be scanned by a given library.
2. **Removing a book** from the set of books that were going to be scanned by a library.
3. **Swapping books between libraries.**
4. **Removing a book** from the list of books to be scanned by a library and **adding that book to another library**. If possible, **replace the removed book** in the first library **with another one** that is available there.

Modeling — Two-Phase Approach

Use a meta-heuristic to choose an order for the libraries and solve the book assignment problem optimally.

Notably, the assignment problem can be modeled as bipartite graph and solved with any **min cost max-flow** solver.

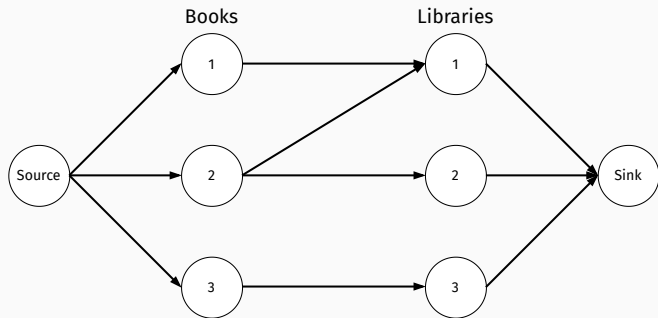


Figure 6: Assignment Problem Modeled as a Bipartite Graph

After the book assignment the solution can be further improved through the following **local moves**:

1. **Reverse the sign-up order** between two libraries.
2. **Change the positions of two libraries** in the order adjusting the sign-up times of every library in between.
3. **Select one library to remove and add another library** that is not currently considered in that position, if possible.

The best objective values for the five instance of this problem were obtained using the **two-phase** approach, as illustrated in the table below:

Instance	Objective Value	Best Known Objective Value
“Read On”	5 822 900	5 822 900
“Incunabula”	5 689 822	5 690 888
“Tough Choices”	5 028 530	5 107 113
“So many books”	5 208 455	5 237 345
“Libraries of the world”	5 328 034	5 348 248

Table 8: Book Scanning Best Results

Remarkably, this objective value would have placed us in **33rd** place (out of 10716 teams) on the competition leaderboard.

Conclusion & Future Work

In this work we were able to:

- Implement a modelling framework for meta-heuristics.
- Implement several meta-heuristic solvers.
- Develop models two Google Hash Code problems that achieve competitive results.





The following topics can be considered as interesting future work directions:

- Development of more Google Hash Code problem models.
- Implementation of more Meta-Heuristic solvers.
- Experimental evaluation of Meta-Heuristics.

Questions?

1. Thank the audience for being awake.

References

-  Fonseca, Carlos M. (Oct. 27, 2021). *Nasf4nio*. URL: <https://github.com/cmfonseca/nasf4nio> (visited on 01/15/2023).
-  Osman, Ibrahim H. and Gilbert Laporte (Oct. 1, 1996). “Metaheuristics: A Bibliography”. In: *Annals of Operations Research* 63.5, pp. 511–623. ISSN: 1572-9338. DOI: 10.1007/BF02125421.
-  Outeiro, Samuel Barroca do (Nov. 9, 2021). “An Application Programming Interface for Constructive Search”. Msc Thesis. University of Coimbra, Portugal.
-  Vieira, Ana (2009). “Uma plataforma para a avaliação experimental de meta-heurísticas”. Doctoral Thesis. University of Algarve, Portugal.