Principled Modelling Of The Google Hash Code Problems For Meta-Heuristics

Intermediate Defense

Pedro Rodrigues

Alexandre B. Jesus
Carlos M. Fonseca

January 2023

University of Coimbra
Department of Informatics Engineering

Outline

Introduction

Motivation

This work is related to both theoretical and practical aspects that arise from the analysis of real-world problems.

The motivation behind this work relates to the answer to three key questions:

- Why The Google Hash Code Problems ?
- Why (Principled) Modelling ?
- Why Meta-Heuristics ?

Why The Google Hash Code Problems ?

The Hash Code programming competition is a yearly event that invites people from around the world to solve a challenging problem.

The contestants participate in teams of 2 to 4 elements and have about 4 hours to come up with a solution.

The challenges:

- Are complex problems inspired by engineering challenges.
- Resemble real-world scenarios.
- Are well formulated and can be solved to some extent in a short amount of time (4 hours).

Why (Principled) Modelling ?

Modelling 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.

A good model encodes information about:

- The particular problem instance parameters.
- The space of possible solutions.
- How solutions can be evaluated.

Why Meta-Heuristics ?

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:

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

The “Big Picture”

Figure 1: Modelling and Problem-Solving

Background

Optimization

Optimization Problem

An optimization problem is a collection $\mathcal{I}$ of instances, typically generated in a similar manner. An instance ι of an optimization problem consists of a pair $(\mathcal{S}, f)$, where $\mathcal{S}$ is a set containing all feasible solutions, and f is an objective (cost) function such that:

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

Combinatorial Optimization Problem

An instance ι of a combinatorial optimization problem is an instance of an optimization problem where the set $\mathcal{S}$ of feasible solutions is finite.

Global Optimization

Global Optimization is the process of identifying a global optimal solution to a given optimization problem.

Global Optimal Solution

Assuming, without loss of generality, the maximization of an objective function $f(s)$, a global optimal solution s^* is expressed by:

$$\forall s \in \mathcal{S} : f(s^*) \geq f(s)$$

Local Optimization

Local Optimization is the process of finding a solution that is optimal among those that are close to it in some sense.

(Strictly) Local Optimal Solution

Assuming maximization without loss of generality, a solution $\hat{s}$ is locally optimal with respect to a given neighborhood structure $\mathcal{N}$ iff:

$$\forall s \in \mathcal{N}(\hat{s}) : f(\hat{s}) \geq f(s)$$

Furthermore, $\hat{s}$ is considered strictly locally optimal iff:

$$\forall s \in \mathcal{N}(\hat{s}) \setminus \{\hat{s}\} : f(\hat{s}) > f(s)$$

Example — Global and Local Optimal Solutions

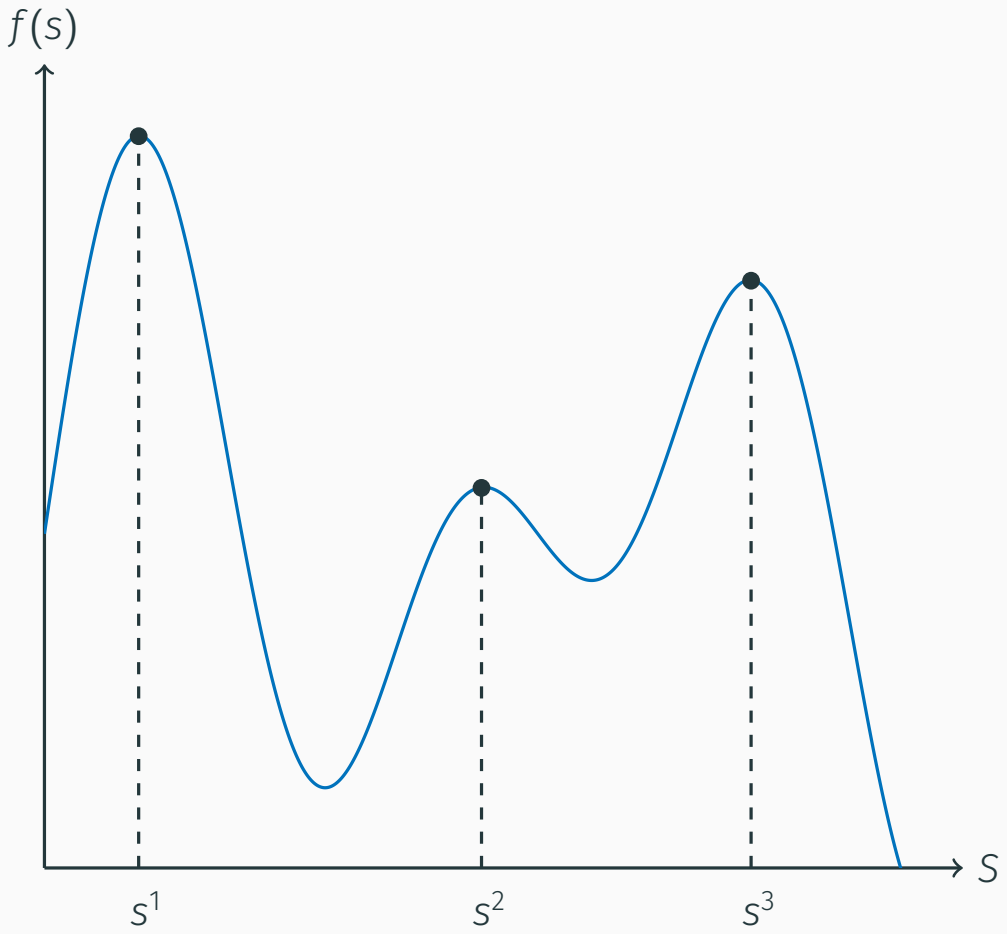

Figure 2: Global and Local Optimal Solutions

Glass-Box and Black-Box Optimization

In Glass-Box optimization, a problem is approached with an understanding of the instance and objective function structure, enabling the optimizer to manipulate that structure directly.

In Black-Box optimization, to find optimal solutions, optimizers evaluate generated solutions (usually taking into account the result of this evaluation as a way to guide the progress of the algorithm) without knowledge of the underlying characteristics of the problem

Exact, Approximation and Heuristic Methods

Three general approaches exist for tackling optimization problems:

- Exact methods
  - Able to find one or all global optimal solutions.
  - Can take a very large amount of time.
- Approximation Methods
  - Able to find solutions with an approximation quality guarantee.
  - Problem-specific and often hard to come up with for unseen (non-trivial) problems.
- Heuristic methods
  - Can often generate “good” solutions but offer no guarantee on their quality.
  - Quick to develop and implement.

Meta-Heuristics

In general, most meta-heuristics can be characterized by the following key properties:

Search Strategy Constructive and Local Search.

Memory Keep track of previous states.

State Size Population *vs* Single-State.

The following meta-heuristics, analyzed in the context of this work, incorporate these aspects:

- *Hill-Climbing*
- *Iterated Local Search*
- *Tabu Search*
- *Greedy Randomized Adaptive Search Procedures*
- *Ant Colony Optimization*

Constructive Search

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

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

Ground Set

The ground set $\mathcal{G}$ is a finite set of elements that represents all possible components that may integrate a solution.

$$\mathcal{G}: \{c_1, c_2, c_3, \ldots, c_i\}$$

Thus, a feasible solution $s \in S$ is a subset of the ground set i.e. $s \subseteq 2^{\mathcal{G}}$.

Local Search

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

- Start with a feasible solution to the problem.
- Perturb the solution by exploring candidate solutions in the neighborhood.
- Repeat the process until the solution cannot be further improved.

This method is typically applied after a constructive search phase.

Bounds

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

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

Upper Bound

An upper bound of a solution $s^p \in 2^{\mathcal{G}}$ is a numeric value given by a function Φ_{ub} such that:

$$\forall s \in \mathcal{S} \land s \supseteq s^p : f(s) \leq \Phi_{\text{ub}}(s^p)$$

Modelling

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

- Instance Parameters
- Decision Space
  - Solution Definition
  - Component Definition
- Construction Rules
- Evaluation
  - Objective Function
  - Bounds

Modelling Frameworks

In the field of optimization, most software does not separate the modeling for meta-heuristics from the implementation of solvers.

However, some frameworks that integrate these concepts have been developed, such as:

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)

Preliminary Work

The Google Hash Code Problems

To better understand the Google Hash Code problems (Google, 2014) a comprehensive analysis of a few problems was conducted.

The main goal was to understand the key properties and similarities with known literature problems.

Problem	Categories				
	Assignment	Knapsack	Coverage	Vehicle Routing	Simulation
Street View Routing			✓	✓	
Optimize a Data Center	✓	✓			
Loon			✓	✓	✓
Delivery				✓	

Table 1: Categorization of Google Hash Code Problems

Modelling “Optimize a Data Center”

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 are logically assigned to pools contributing with their (computing) capacity.

Example — Data Center Layout

Server	Size	Capacity
0	3	2
1	2	5
2	3	10
3	2	3

Table 2: Server Properties

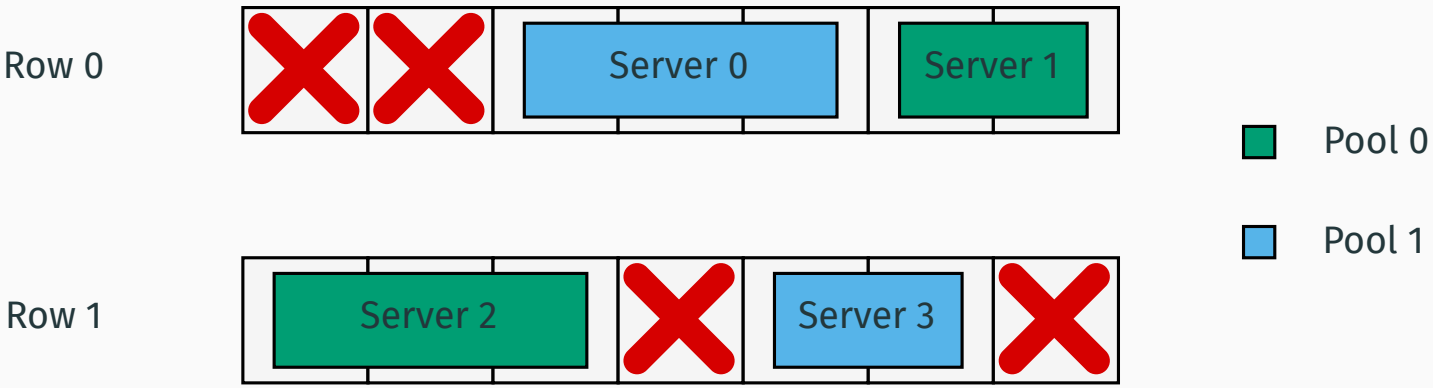

Figure 3: Example Data Center Layout

Objective

Guaranteed Capacity

Given a resource pool ($p \in \mathcal{P}$), the guaranteed capacity gc_p is a measure of the remaining computing capacity available in the event that at most one arbitrary row ($r \in \mathcal{R}$) of the data center becomes offline.

$$gc_p = \min_{r \in \mathcal{R}} \left(\sum_{s \in p} c_s - \sum_{s \in p \wedge s \in r} c_s \right)$$

Objective Function

The objective function to be maximized is:

$$f(s) = \min_{p \in \mathcal{P}} \left(gc_p \right)$$

Example — Solution Evaluation

Pool	Row 0	Row 1	Guaranteed Capacity	Score
0	5	10	5	2
1	2	3	2	

Table 3: Guaranteed Capacities and Score

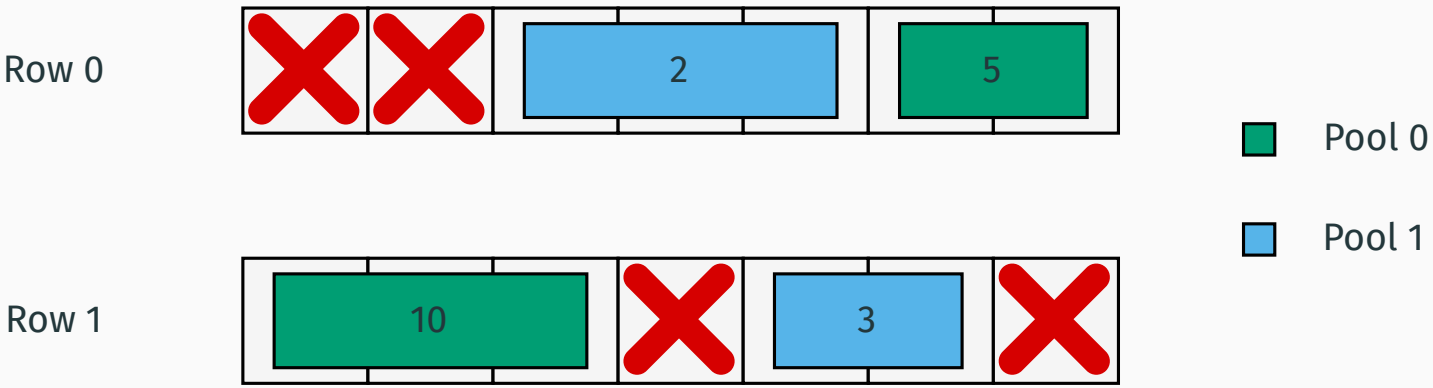

Figure 4: Example Data Center Layout (Capacities Only)

Instance Parameters

The problem statement guarantees that the generated instances have the following properties:

- $1 \leq \mathcal{R} \leq 1000$: Number of rows in the data center.
- $1 \leq \mathcal{S} \leq 1000$: Number of slots in a row.
- $0 \leq \mathcal{U} \leq \mathcal{R} \times \mathcal{S}$: Number of unavailable slots.
- $1 \leq \mathcal{P} \leq 1000$: Number of resource pools to be created.
- $1 \leq \mathcal{M} \leq \mathcal{R} \times \mathcal{S}$: Number of servers to be allocated.

Decision Space

A solution is a set of assignments of servers to rows and pools that does not violate the problem's constraints.

A solution, either *empty*, *partial*, or *complete* is always *feasible* provided that it does not violate the problem's constraints.

- **Empty Solution**: All pools and servers are unassigned.
- **Partial Solution**: Some servers are available and there is space for placement.
- **Complete Solution**: There is no space available for server placement or all the servers have already been assigned.

A component is defined as a tuple containing the server, segment, and pool.

- A "segment" in the context of this problem is a contiguous sequence of slots that may exist in a given row.

Construction Rules

For constructing a solution the following rules must be considered:

- A server can be placed in any segment that can hold its size.
- A server can be assigned to any resource pool available.

A possible heuristic construction abiding by these rules undergoes the following steps:

1. Select an available **server** with the best capacity-to-size ratio.
2. Select the **pool** with the lowest guaranteed capacity.
3. Select the **row** with the least capacity for this pool that has space available.
4. Select any **segment** in that row that can hold the server.
5. Assign the *server* to the selected *segment* and *pool*.

Upper Bound

The construction of the upper bound is made in two steps:

1. **Row-Wise Bound**
   Let,
   - $\Theta_{\mathcal{R}\setminus r}$: Denote the remaining empty space in all the rows in the data center but the row r.
   - $\sum_{\Theta_{\mathcal{R}\setminus r}}$: Denote the maximum sum of the capacities of the available servers that can be fractionally placed into $\Theta_{\mathcal{R}\setminus r}$ w.r.t to the ratio between the capacity and the size of the server.

   Then, the row-wise upper bound is expressed by:

$$\Phi^{r}_{ub} = \frac{\sum_{\Theta_{\mathcal{R}\setminus r}}}{\mathcal{P}}$$

2. **Upper bound**
   The upper bound for a given solution $s \in \mathcal{S}$ can then be calculated as:

$$\Phi_{ub}(s) = \min_{r \in \mathcal{R}} \Phi^{r}_{ub}$$

Upper Bound

Some remarks must be made on the "Row-Wise Bound" Φ^r_{ub} calculation:

- Discarding a row involves subtracting the capacities of servers in the respective resource pools.
- If $\Phi^r_{ub} > gc_p$ for any pool $p \in \mathcal{P}$, the following correction is applied:
  1. Remove relevant pools and servers.
  2. Recompute Φ^r_{ub} with a reduced server set and number of pools.
  3. Repeat this process until no further correction is required.

Results

We were able to achieve a score of 386 points (maximum known is 407) on the instance that was available, which places us at 25^{th} place (out of 230) in the classification table.

This was achieved with a simple constructive search approach, and without considering yet local search.

Results

Algorithm 1: Narrow Guided Heuristic Construction

Input: Problem Instance ($\mathcal{P}$), Limit (N).
Output: Best solution (s^*).

```
begin
    s* ← ∅
    while True do
        updated ← False
        s' ← s*
        c* ← ∅
        for i = 0 to N do
            c' ← HeuristicMoveWOR(s*, ADD)
            ApplyMove(c', s', ADD)
            if Φ_ub(s') > Φ_ub(s*) then
                c* ← c'
                updated ← True
            end
            ApplyMove(c', s', REMOVE)
        end
        if ¬ updated then
            return s*;
        end
        ApplyMove(c*, s*, ADD)
    end
end
```

Work Plan

Work Plan

The table below outlines the tasks to be executed in the upcoming semester, based on the objectives for this work:

Task	Description
HC#01	Modelling “Optimize a Data Center”
HC-AS	Hash Code Problem Analysis and Selection
HC#02	Modelling Hash Code Problem #2
HC#03	Modelling Hash Code Problem #3
MH	Meta-Heuristic Implementation
EA	Experimental Analysis
TW	Thesis Writing

Table 4: Task List

Timeline

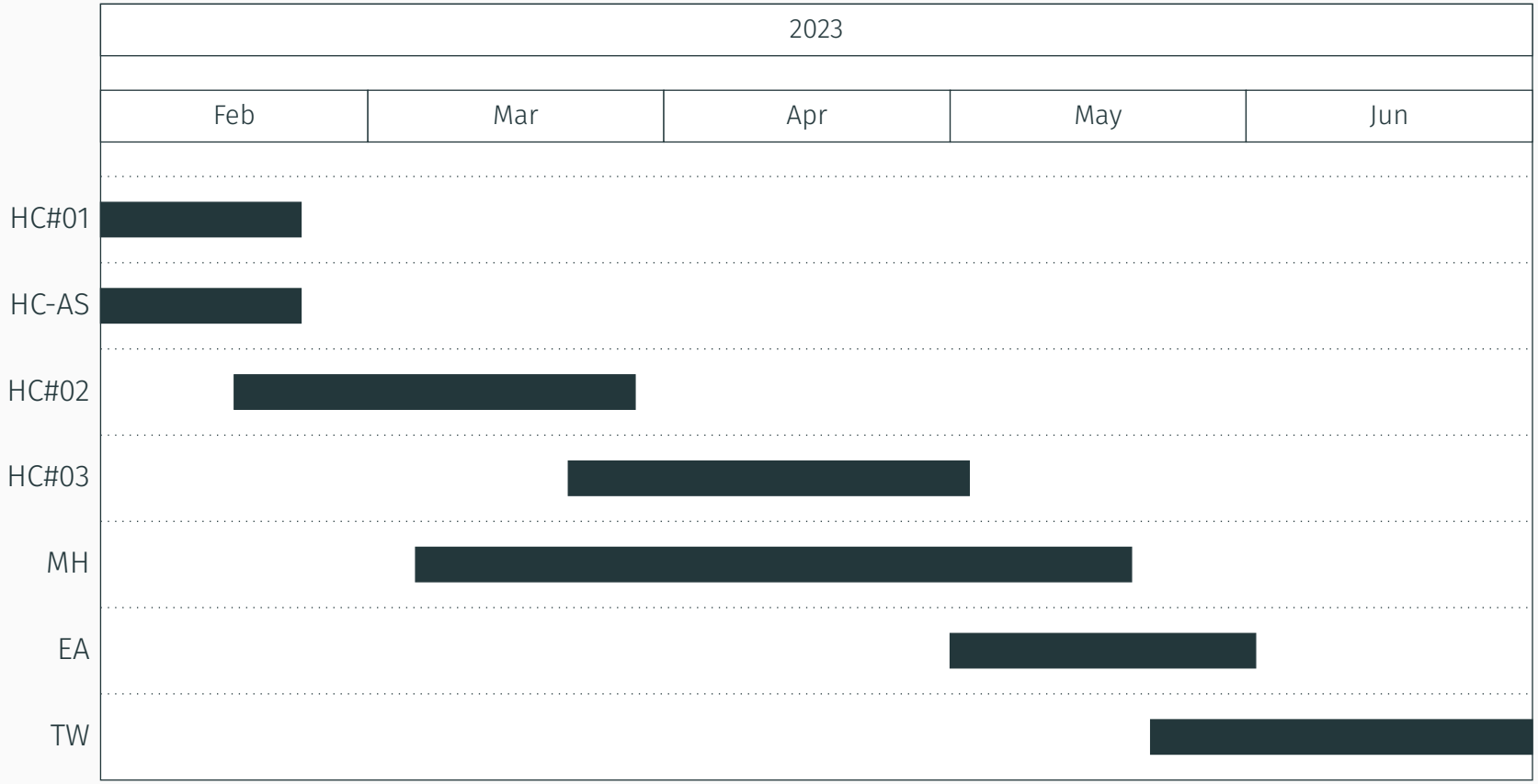

Figure 5: Project Timeline

Questions?

References

References i

- Fonseca, Carlos M. (Oct. 27, 2021). *Nasf4nio*. URL: https://github.com/cmfonseca/nasf4nio (visited on 01/15/2023).
- Google, HashCode (2014). *Hash Code - Google's Coding Competitions*. Coding Competitions. URL: https://codingcompetitions.withgoogle.com/hashcode/archive (visited on 01/10/2023).
- 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.

References ii

 Vieira, Ana (2009). “Uma plataforma para a avaliação experimental de meta-heurísticas”. Doctoral Thesis. University of Algarve, Portugal.