1 2 9 0

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE
COIMBRA

Pedro Miguel Duque Rodrigues

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

Dissertation in the context of the Master in Informatics Engineering, Specialization in Intelligent Systems, advised by Professor Alexandre D. Jesus and Professor Carlos M. Fonseca, and presented to the Faculty of Sciences and Technology / Department of Informatics Engineering.

September 2023

The work presented in this thesis was carried out in the Algorithms and Optimization Laboratory of the Adaptive Computation group of the Centre for Informatics and Systems of the University of Coimbra and financially supported by the Foundation for Science and Technology under a scholarship with reference UIDP/00326/2020.

Abstract

Acknowledgments

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

"Begin at the beginning", the King said gravely, "and go on till you come to the end; then stop."

— Lewis Carroll

1.1 Motivation

Optimization problems are ubiquitous in real-world scenarios. For example, when considering the task of planning a road trip, we are quickly faced with several optimization challenges arising from a seemingly simple task, such as, finding the shortest or cheapest route, and efficiently packing luggage. When solving these problems the goal is typically to find "good" solutions in a time-efficient manner.

Tackling an optimization problem can typically be seen as a two-phase process. First, we start by understanding the problem and modeling its details. Then, we can apply, or develop, a solver to find one or more solutions taking into account the given model. This approach serves, for example, as the foundation of most linear optimization software packages which are widely used to solve real-world problems, *e.g.*, *Gurobi*, *CPLEX*, or *GLPK*. In particular, such packages expect a mathematical formulation (model) describing the problem as a linear objective function and a set of linear constraints, and then use one or more algorithms designed to solve such linear optimization problems. This clear separation of concerns leads to some advantages. Notably, practitioners who want to solve a particular problem can focus on developing the model for that problem and easily use existing solvers to find solutions without needing to implement state-of-the-art algorithms themselves. Meanwhile, solver developers can take advantage of existing problems to test and enhance their solvers' performance.

In this work, we are interested in tackling Combinatorial Optimization (CO) problems using a similar separation of concerns. Generally speaking, CO consists of finding an optimal solution, according to some objective function, from a discrete set of solutions. Several generic approaches have been developed to solve CO problems exactly, *i.e.*, to find an optimal solution. However, many CO problems are NP-Hard, meaning that, the time required to solve them via (problem-specific) exact approaches grows exponentially with the problem size, and consequently generic exact methods will also grow exponentially. In practice, this means that exact methods are often ineffective to solve "real-world" CO problems which have large problem sizes.

As a result, there has been a growing interest in the development of methods that can find "good" solutions for such problems. In this work, we focus on heuristic and meta-heuristic methods. Heuristic methods are search procedures, often problem-specific, that attempt to quickly solve a problem and provide a "rule of thumb" for attaining decent solutions, albeit without optimality guarantees. Meta-Heuristic (MH) methods employ several high-level strategies to construct and improve solutions. It is worth noting that such high-level strategies often depend on problem-specific details, *e.g.*, the neighborhood structure and search tree definition. However, meta-heuristics do not require knowledge about these problem-specific details and instead use the high-level strategies in a black-box fashion. As such, MH approaches are problem-independent and can be applied to a broad range of problems.

Given the nature of MH methods and the inherent diversity of problems, crafting universal MH solvers is a challenging task made harder due to the difficulty in separating the problem-specific details required by the high-level strategies from the MH problem-independent solving process. In fact, the abundance of MH optimization software that provides specific frameworks for implementing evolutionary, local or constructive search meta-heuristics for CO problems [12, 11, 22] and the lack of a unifying framework supporting all approaches can be regarded as a symptom of the difficulty of this endeavor. Still, it is worth remarking the works by Vieira [14] and Outeiro [32], which partially looked at the formalization of this objective.

The development of a unifying framework would standardize problem-solving approaches, facilitate the reuse of MH methods, and distinctly separate the tasks of problem modelling and solver development. Moreover, it would provide researchers and practitioners with a valuable tool to experimentally assess the performance of MH methods across a range of diverse problems.

Simultaneously, alongside the development and application of MH strategies to address CO problems, there exists a community interest in constructing a collection of benchmark optimization problems that hold both theoretical and practical significance [28]. The Google Hash Code competition problems, arguably, present themselves as suitable candidates.

The Hash Code programming competition, formerly hosted annually by Google, challenged teams of up to four members to solve intricate CO problems within a four-hour time frame using any tools, (online) resources, and programming languages of their choice. These problems often drew inspiration from real-world challenges, such as vehicle routing, task scheduling, and router placement. There is often some relation to classical problems found in CO literature, which may provide theoretical and practical insights on how to solve them but these seem to be harder to solve then their classical counterparts. Moreover, exact algorithms to solve these problems efficiently are not known, and up to our knowledge there is also no known heuristic or approximation method that prevails over other approaches.

Given the pertinence of these problems, and the wide range of challenges they present from both a theoretical and practical standpoint, they serve as interesting benchmarks for the evaluation of meta-heuristics offering ample research potential. Furthermore, they offer a suitable approach to assess the feasibility of the aforementioned unifying framework on more realistic and challenging problems beyond the ones commonly found in the literature.

1.2 Goals & Scope

The main goal of this work is the implementation and evaluation of meta-heuristic solution approaches for the Google Hash Code problems, using a principled approach that separates the modeling of the problems from the solvers.

In particular, we aim to expand upon the modelling approach for meta-heuristics that has been partially explored in previous research [14, 31, 32]. The objective is to solidify existing concepts while introducing additional functionality, both in conceptual understanding and practical application.

Furthermore, we aim to construct models for the Google Hash Code problems. These models will not only be described and discussed in this thesis but will also serve as illustrative examples documenting the modelling concepts. Furthermore, they will enable a critical evaluation of the merits and shortcomings of this principled approach in comparison to more ad-hoc and traditional methods of problem-solving.

Finally, the implementation of state-of-the-art meta-heuristic solvers is a vital component of our work as it will enable us to assess the performance and quality of solutions found for the models of the Google Hash Code problems as well as the feasibility of the modelling approach for meta-heuristic solver development.

In summary, the main research questions we outline for this thesis are:

R1. Can we formalize the existing ideas explored by previous work on the modelling framework [14, 31, 32] and produce a practical implementation, potentially contributing with new features?

R2. Can we implement general-purpose meta-heuristic solvers with respect to the principled modelling framework implementation?

R3. Can Google Hash Code problems be solved effectively using this modelling approach?

1.3 Contributions

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

C1. With the existing research on principled modelling for meta-heuristics [14, 31, 32], this document aims to consolidate and formalize a comprehensive specification. Our objective is to encapsulate all the concepts and developments made thus far. Additionally, we have created a practical Python implementation of this framework. In essence, both in the formalization and implementation, we endeavour to synthesize the existing ideas concerning modelling for constructive and local search techniques.

C2. We implemented several meta-heuristic solvers and utilities both for gathering the solutions and for testing the developed models. Given that these are general-purpose they can work with any model that is developed under the practical implementation of the framework we devised.

C3. We selected two Google Hash Code problems for which some models for each of the problems were developed that explore the different properties of the problems in an attempt to both obtain the best solutions possible. These models provide a practical example on how to model relatively complex problems and also allows us to think critically about the framework capabilities.

1.4 Software

The following software resulted from the development of this thesis and has been distributed under an open source license.

S1. Python Framework (TODO)

S2. Models and Experiments Code (TODO)

1.5 Outline

The remainder of thesis is structured as follows. In Chapter 2, we provide an overview of optimization concepts, meta-heuristics, and modelling in the context of meta-heuristics. Moving to Chapter 3, we analyze the Google Hash Code competition, focusing on the characteristics of the problems and their relation to existing CO literature. In Chapter 4, we discuss a modelling framework and its role in meta-heuristic development. Chapters 5 and 6 present detailed studies of the Hash Code problems "Optimize a Data Center" and "Book Scanning", with experimental results. Finally, Chapter 7 summarizes findings in this work and suggest future research directions.

Chapter 2

Background

"If I have seen further than others, it is by standing upon the shoulders of giants."

— Isaac Newton

This chapter presents a comprehensive literature review of optimization, metaheuristics and modelling. Additionally, it provides a background review of the state-of-the-art regarding the principled modelling approach. In particular, Section 2.1 describes fundamental CO concepts deemed relevant for better understanding this work. Section 2.2 discusses multiple well-known techniques for solving CO problems. Section 2.3 describes MH methods and presents an extensive review of MH solvers. Finally, Section 2.4 delves into the details of the modelling approach and describes the existing implementations.

2.1 Optimization Concepts

Optimization, as defined by Papadimitriou and Steiglitz [7], is the task concerning the search for an optimal configuration or set of parameters that maximizes or minimizes a given objective function. In other words, optimizing involves finding the best solution to a given problem among a set of feasible solutions. Formally, an optimization problem can be defined as follows:

Definition 2.1.1 (Optimization Problem [7]). *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:*

$$f: \mathcal{S} \longrightarrow \mathbb{R} \tag{2.1}$$

That is, each solution $s \in \mathcal{S}$, is assigned a real value representing its quality.

Definition 2.1.2 (Global Optimum [5, 7]). *Assuming, without loss of generality, an optimization problem with a maximizing objective function a global optimum $s^* \in \mathcal{S}$ is expressed by:*

$$\forall s \in \mathcal{S}: f(s^*) \geq f(s) \tag{2.2}$$

Since Google Hash Code problems [34] have a single-objective maximizing objective function, we will only consider maximization in this work. However, it is possible to reformulate problems with a minimizing objective function for maximization [13] using the identity:

$$\min f(s) = \max -f(s) \tag{2.3}$$

2.1.1 Combinatorial Optimization

Combinatorial Optimization (CO) problems are a subset of optimization problems characterized by a discrete solution space that typically involves different permutations, groupings, or orderings of objects that satisfy some problem-specific criteria [7, 14, 10, 21]. Thus, regarding the previous definition of an optimization problem, a CO can be formally defined as follows:

Definition 2.1.3 (Combinatorial Optimization Problem [7]). *A combinatorial optimization problem is an optimization problem (2.1.1) where the set $\mathcal{S}$ of feasible solutions is finite or countably infinite.*

Typical examples of CO problems include network flow, matching, scheduling, shortest path and decision problems. Notably, the Knapsack Problem (KP) [33, 21, 26] is a well-known example of a CO problem where the goal is to find the subset of items with the highest total profit that can fit in a knapsack without exceeding its maximum capacity.

Due to the combinatorial nature of CO problems, solutions are often defined in terms of a *ground set*.

Definition 2.1.4 (Ground Set [32, 26, 25]). *The ground set of a CO problem* *is a finite set of components $\mathcal{G} = \{c_1, c_2, \ldots, c_k\}$, such that every solution to the problem, feasible or not, can be defined as a subset of $\mathcal{G}$.*

For this work, it is also relevant to define the notion of empty, partial and complete solutions.

Definition 2.1.5 (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 $s = \emptyset$.*

Definition 2.1.6 (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 $s' \supseteq s$.*

Definition 2.1.7 (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 $s' \supset s$.*

It is worth noting that, according to our definition, a partial solution is not required to be feasible, unlike a complete solution.

To illustrate these concepts, let's consider the practical example of the KP. In this context, the ground set is the set of all the available items (components). As such, a feasible solution is one in which the sum of the weights of the items placed within the knapsack (select components) does not exceed its capacity. A partial solution is one where additional items (components) can be still be added to the current solution without exceeding the capacity of the knapsack. Note that, for the KP every partial solution is feasible. Finally, a complete solution is a feasible solution where no further items can be added due to capacity constraints.

In essence, since CO problems involve choosing a combination of components any algorithm that is able to enumerate all possible combinations can be used can be used to solve these problems. However, finding an optimal solution can be difficult, and exhaustive search strategies may not be able to solve many of these problems, which are often NP-Hard [21, 26]. In these cases, heuristic and MH methods present themselves as effective alternatives to be considered.

2.1.2 Bounds

In mathematics, the notion of bounds has its origins in set (order) theory. More precisely, upper and lower bounds are defined as the sets of *majorants* and *minorants* of a given parent set. Majorants are the elements greater or equal to the highest value within the parent set. Likewise, minorants are the elements smaller or equal to the smallest value in the parent set. Furthermore, an upper bound is regarded as *tight* or *strong* when no smaller value can serve as an upper bound, a concept known as the *supremum* or *least upper bound* of a set. Similarly, a lower bound is considered tight when no higher value can function as a lower bound, which is known as the *infimum* or *greatest lower bound* of a set.

This concept holds significance in CO and is frequently employed when aiming to optimize the objective value of (partial) solutions. In essence, the upper and lower bounds of a (partial) solution encompass the range of values for an objective function that are either greater/smaller than or equal to the optimal value achievable through the partial solution. The same logic applies to the concept of tight upper and lower bounds.

Formally speaking, the upper bound and lower bound [7, 32] of a (partial) solution can be defined as follows.

Definition 2.1.8 (Upper Bound). *An upper bound of a (partial) solution $s' \in 2^{\mathcal{G}}$ is a numeric value given by a function $\Phi_{ub}\colon 2^{\mathcal{G}} \longrightarrow \mathbb{R}$ such that:*

$$\forall s \in \mathcal{S} \land s \supseteq s'\colon f(s) \leq \Phi_{ub}(s') \tag{2.4}$$

Definition 2.1.9 (Lower Bound). *A lower bound of a (partial) solution $s' \in 2^{\mathcal{G}}$ is a numeric value given by a function $\Phi_{lb}\colon 2^{\mathcal{G}} \longrightarrow \mathbb{R}$ such that:*

$$\forall s \in \mathcal{S} \land s \supseteq s'\colon \Phi_{lb}(s') \leq f(s) \tag{2.5}$$

The usage of bounds is often common in CO algorithms, especially in exact approaches, as will be later detailed. Nonetheless, concerning MH methods, bounds can be helpful tools for guiding the solution construction process. The rationale is that, when faced with a choice between adding one of two components, considering the upper bound's value becomes valuable as opting for the component having a lesser negative impact on the bound's value preserves the solutions' future potential for improvement. Note that the tightness of a bound is a determining factor for keeping this potential.

Moreover, while the objective function holds significance in directing the optimization process, there are situations where evaluating the quality of a partial solution might not be possible. Additionally, the objective function might pose a bottleneck, obstructing the optimizer's ability to measure the impact of changes on solution quality. Nevertheless, the value for an upper bound can consistently be defined, thus emphasizing the importance of bounds in this context.

In essence, the upper bound value provides an optimistic look at the quality of a (partial) solution and its potential to improve. Conversely, the lower bound, provides a realistic perspective on the objective value of the solution. Moreover, for a complete solution, the upper bound value will always match the objective value.

2.1.3 Global and Local Optimization

With regard to the search of solutions for optimization problems, there are two primary strategies: Global Optimization (GO) and Local Optimization (LO).

GO involves the process of discovering the ultimate global optimum (2.1.2) for a given problem, regardless of where it might lie within the solution space. This search for the best solution is often called *exploration*. In contrast, LO concentrates on finding the most optimal solution among proximate that are close in some way, which is commonly referred to as *exploitation*. The concept of proximity is related to the definition of a neighborhood, which for a given solution is specified by a particular neighborhood structure defined as follows:

Definition 2.1.10 (Neighborhood Structure [7, 10]). *A neighborhood structure for an optimization problem is a mapping:*

$$\mathcal{N} : \mathcal{S} \longrightarrow 2^{\mathcal{S}} \tag{2.6}$$

Such that, a set of neighboring solutions $\mathcal{N}(\hat{s}) \subseteq \mathcal{S}$ is assigned for each solution $\hat{s} \in \mathcal{S}$. This referred to as the neighborhood of $\hat{s}$.

In general, the neighborhood structure refers to the set of rules that must be applied to a solution in order to generate all of its neighbors. Additionally, we can define a local optimal solution or just local optimum as follows:

Definition 2.1.11 (Local Optimum [5, 10, 13]). *Assuming maximization without loss of generality, a solution $\hat{s}$ is a local optimum with respect to a given neighborhood structure $\mathcal{N}(\hat{s})$ iff:*

$$\forall s \in \mathcal{N}(\hat{s}) : f(\hat{s}) \geq f(s) \tag{2.7}$$

Furthermore, $\hat{s}$ is a considered a strict a local optimum iff:

$$\forall s \in \mathcal{N}(\hat{s}) \setminus \{\hat{s}\} : f(\hat{s}) > f(s) \tag{2.8}$$

As an illustrative example, consider the objective function $f(s)$ shown in 2.1. With respect to the Definitions 2.1.2 and 2.1.11, the solution s^1 is a global optimum and s^2, s^3 are (strict) local optima.

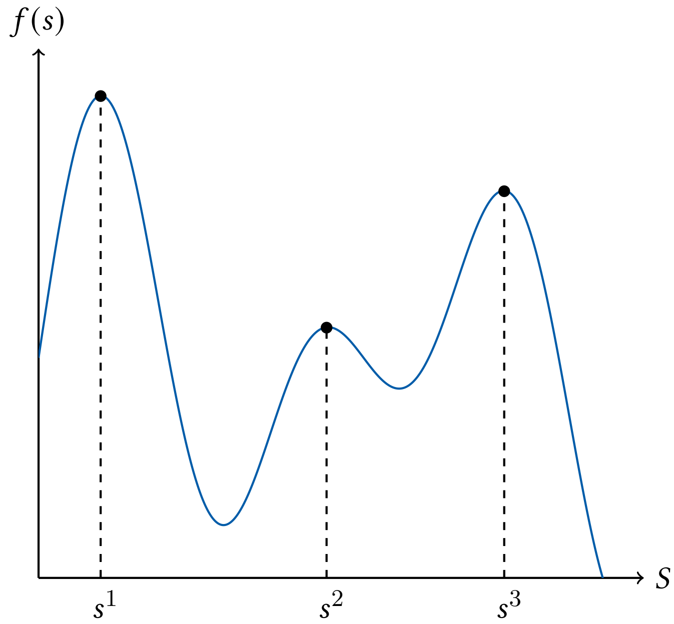

Figure 2.1: Global and Local Optima

In practice, the decision to use either a global or local optimization strategy is often influenced by factors such as the available time budget and the preferences of the decision maker. While GO aims to find the optimal solution to a problem, the search process may be time-consuming or, in some cases, computationally infeasible due to the size of the search space. On the other hand, LO, while lacking the optimality guarantees of, is able to quickly generate "good" solutions that may be acceptable to the decision maker. Nonetheless, the quality of the solutions may be poor due to the ruggedness of the objective function fitness landscape (many local optima). Ultimately, the performance of both methods is closely tied to problem-specific knowledge.

In the Google Hash Code competition, due to the time imposed by the competition setting, it is often not in the interest of the contestants to use global optimization methods, as they are unlikely to finish on more complex problem instances. Instead, a balance between global and local optimization (*exploration* and *exploitation*) is typically employed. To elaborate, the strategy typically involves *exploring* the search space via GO methods to find "good" starting solutions, that LO methods can further *exploit*.

2.1.4 Black-Box and Glass-Box Optimization

In the field of optimization, two settings are commonly recognized: Black-Box Optimization (BBO) and Glass-Box Optimization (GBO).

In BBO optimization settings there is no information about the landscape of the function being optimized, constraints defining the set of feasible solutions [30] or the objective function is too complex to be approached from an analytical perspective. As such, algorithms for achieving solutions for these problems do so only by interacting with the problem through the evaluation of potential candidate solutions [29, 17]. Meta-Heuristics, as will be later detailed are examples of methods that follow this approach for finding/improving solutions. By contrast, in GBO optimization, also known as *white box* optimization, there is a good understanding of the problem instance being optimized and the objective function properties [29]. Hence, the algorithms used may take advantage of more analytical properties of the problem since they are transparent to the solver.

For the purpose of clarification, with regard to the previously mentioned KP, a BBO strategy would entail the utilization of a search heuristic such as simulated annealing [24] to obtain solutions. This is because the algorithm only necessitates knowledge of how to evaluate the quality of solutions through the objective function and not any specific information about the function being optimized. Alternatively, if the problem were to be formulated as an Integer Linear Programing (ILP) [13, 7] problem, it would become amenable to a GBO approach, as the objective function would be accessible, and additional information about the problem could be inferred from it and provided to the algorithm.

In the context of the Hash Code competition, contestants typically engage the problems from a BBO perspective, as the underlying objective function of the is too complex to formalize. Additionally, the process of formalization can be time-consuming and, as a result, the usage of GBO methods post-formalization would not be justified, as they could turn out to be computationally slower. However, it is in many cases, possible to use GBO methods, *e.g.*, ILP to tackle subproblems that are more simple to formalize as more knowledgeable contestants have demonstrated in the past.

2.2 Optimization Strategies

Combinatorial optimization literature extensively documents a series of methods for solving multiple problems [7, 21, 24]. The approaches followed by these methods are diverse and typically defined by factors such as the time complexity, the quality and the strategy for finding solutions. Particularly, algorithms are often classified in the literature as exact, approximation, or heuristic based on the quality of solutions. Moreover, for each category exist procedures with varying complexities rely on different strategies to find solutions which, in essence, can be classified as constructive or local.

2.2.1 Exact, Approximation and Heuristic Methods

Exact methods are designed to find the optimal solution for a given problem. These typically involve an exhaustive enumeration and evaluation of solutions. However, in large problem instances, this may prove to be computationally infeasible. In the context of CO problems, two general exact algorithms are well-known and widely studied: *Branch and Bound* and *Dynamic Programming* [8, 26].

These algorithms operate by iteratively breaking down a problem into smaller, interconnected or standalone sub-problems. The solutions to these are then combined to form the final solution. However, each algorithm employs distinct techniques to enhance the exploration of the search space.

In Branch and Bound approaches, the strategy revolves around utilizing bounds to restrict the search space. Specifically, the upper bound facilitates pruning the search tree, effectively eliminating the need to explore solutions that are undoubtedly worse with respect to best solution found by the algorithm at a particular stage. Similarly, the lower bound guarantees that solutions of inferior quality are rejected during the search process. On the other hand, Dynamic Programming approaches leverage the optimal substructure property [26], thereby avoiding recomputing previous sub-problems.

Approximation methods are designed to find solutions that are provably guaranteed to be close to the optimal quality with respect to a given approximation factor. In fact, approximation methods are often able to solve problems in polynomial time and yield solutions of relatively high quality [23]. However, it is important to note that they require a mathematical proof of approximation that is specific to the problem at hand. Notably, a significant amount of research exists in this field concerning CO problems [1].

Heuristic methods work by finding solutions according to a general "rule of thumb", the quality of which can be verified through experimentation. These methods do not provide any guarantees of optimality, as they are derived from intuition and their effectiveness is closely tied to the characteristics of the problem at hand. Nevertheless, they are reliable means of finding solutions in difficult CO problems, typically yielding good solutions in a short time frame when compared to exact methods. A well-known class of algorithms that further extends these methods are meta-heuristics as will be further discussed and detailed in Section 2.3.

A MH is as a high-level heuristic method, as alluded by the word "meta", which describes a concept as an abstraction of other. In the literature, the definition of meta-heuristic varies across different sources, resulting in a lack of consensus on a formal description [6, 10, 26, 24]. Nonetheless, one commonly accepted definition, by Osman and Laporte [6], captures the essence of MH methods, which can be defined as iterative generation processes that *"guide and intelligently combine subordinate heuristics for exploring and exploiting solutions in the search space"*. Moreover, most, if not all heuristic and meta-heuristic approaches are defined in terms of constructive and local search approaches, which we describe next.

2.2.2 Constructive Search

Constructive Search (CS) is an approach for optimization where from an empty or partial solution for a given problem a feasible complete solution is constructed by iteratively adding components extracted from the ground set. The construction process is guided by a pre-established set of rules, which may be heuristic in nature or informed by other relevant information, *e.g.*, objective value and bounds. These rules determine what components from the ground set can be included in the solution at each iteration. The construction stops when no more components can be added to the solution, *i.e.*, the solution is complete. For clarification, a generic pseudocode for a CS procedure [25] is shown in Algorithm 2.1.

It's important to observe that a constructive search approach hinges on a strategy for selecting a component to add to a solution, represented in Algorithm 2.1 through the `SelectComponent` function. Moreover, various other problem-specific details come into play. These encompass activities such as enumerating components (C), creating an empty solution, adding a component to a solution, and evaluating its feasibility. Notably, these details directly influence the CS procedure's ability to construct good solutions. This highlights

Algorithm 2.1: Constructive Search Procedure

Input : Ground Set ($\mathcal{G}$)
Output: Solution (s)

$s \leftarrow \emptyset$
$C \leftarrow \{ c \in \mathcal{G} \setminus s \mid s \cup \{c\}$ `is feasible` $\}$
while $C \neq \emptyset$ **do**
  $c \leftarrow$ `SelectComponent`(C)
  $s \leftarrow s \cup \{c\}$
  $C \leftarrow \{ c \in \mathcal{G} \setminus s \mid s \cup \{c\}$ `is feasible` $\}$
end
return s

the importance of having a solid problem *model*, a concept we will delve into further in this thesis.

2.2.3 Local Search

Local Search (LS) approaches begin, with feasible solution to a given problem, and then make modifications by adding or removing components in order to improve it. These modifications, are typically localized and aim to exploit the solution's neighborhood, as defined by the neighborhood structure of the problem. The procedure concludes when no further alterations can enhance the solution's quality, ultimately leading to a local optimum. In the scope of this work, a transformation that can be applied to a solution within in the context of a LS procedure will be referred to as *local move*.

The primary objective of a LS process is to improve a solution in the direction of the local optimum. However, it is common in a LS approach to eventually worsen a solution in order to allow for further exploration of previously unseen regions of the search space. This action is commonly referred to as a *perturbation* [18]. Furthermore, LS is frequently applied in sequence to a constructive search phase, where a CS algorithm is used to construct a good initial solution, which is then further improved through a LS approach.

A generic pseudocode for a local search procedure is outlined in Algorithm 2.2. It's crucial to clarify that within the pseudocode, the `Step` function signifies the execution of a local move, while the `Perturb` function represents an optional action introducing a perturbation to the solution, if possible. Additionally, the enumeration of possible local moves ($\mathcal{M}$) and the selection of a specific local move to incorporate into the solution are abstracted through the `LocalMoves` and `SelectLocalMove` functions, respectively. Note that, as in CS, the problem-specific choices made regarding the implementation of each of these functions will significantly influence the effectiveness of the LS procedure.

Algorithm 2.2: Local Search Procedure

Input : Solution (s')
Output: Solution (s)

$s \leftarrow s'$
$\mathcal{M} \leftarrow$ `LocalMoves`(s)
while $\mathcal{M} \neq \emptyset$ **do**
    $m \leftarrow$ `SelectLocalMove`$(\mathcal{M})$
    $s \leftarrow$ `Step`(s, m)
    $s \leftarrow$ `Perturb`(s) ▷ `Optional`
    $\mathcal{M} \leftarrow$ `LocalMoves`(s)
end
return s

2.3 Meta-Heuristics

In the literature, a multitude of meta-heuristic algorithms have emerged over the years, exploring various ideas to guide the search process [6]. These encompass strategies that narrow the search space to promising regions, enhance solutions in a greedy manner, or utilize randomized and probabilistic techniques, some of which draw inspiration from natural phenomena like collective behavior, natural selection, and physical processes of materials.

However, the majority of state-of-the-art meta-heuristic algorithms can be described by a couple of distinctive traits [10] such as:

Search Strategy. This refers to the method used to construct a solution. It can be one of three main types: constructive (building step by step), local (making small improvements), or a composite approach that combines both strategies.

Memoization. This concept involves maintaining a record or archive of previously explored solutions. This record helps in identifying solutions that may be revisited or disregarded in subsequent stages of the optimization process.

Population vs. Trajectory. This pertains to the number of solutions being evolved during the construction phase. In *population methods*, multiple solutions are worked with at each iteration, while in *trajectory methods*, only a single solution is dealt with at a time.

In this section, we will offer a brief overview of select state-of-the-art MH algorithms, which encapsulate all the above properties and will be utilized and implemented in the context of this work.

2.3.1 Beam Search

Beam Search (BS) [3, 32]

Algorithm 2.3: Beam Search

Input : Beam Width (w), Objective Function f
Output: Solution (s^*)

$s^* \leftarrow \emptyset$
$\mathcal{B} \leftarrow \{s'\}$
while $\mathcal{B} \vee \emptyset$ $\wedge$ `stopping criteria not met` **do**
    $\mathcal{B}' \leftarrow \emptyset$
    foreach $s \in \mathcal{B}$ **do**
        $\mathcal{B}' \leftarrow \mathcal{B}' \cup$ `Branch`(s)
    end
    if $\mathcal{B}' \neq \emptyset$ **then**
        $\mathcal{S} \leftarrow \operatorname{argmax}_{w}\limits_{s \in \mathcal{B}'} f(s)$ ▷ `Select the “w” best solutions`
        foreach $s \in \mathcal{S}$ **do**
            if $f(s) > f(s^*)$ **then**
                $s^* \leftarrow s'$
            end
            $\mathcal{B} \leftarrow \mathcal{B} \cup \{s\}$
        end
    end
end
return s^*

2.3.2 Iterated Greedy

Iterated Greedy (IG) [27, 32]

Algorithm 2.4: Iterated Greedy

2.3.3 Greedy Randomized Adaptive Search Procedure

Greedy Randomized Adaptive Search Procedure (GRASP) [20, 32, 10]

2.3.4 Ant Colony Optimization

Ant Colony Optimization (ACO) [15, 32, 24, 10]

Algorithm 2.5: Greedy Randomized Adaptive Search Procedure

Input : Objective Function (f)
Output: Solution (s^*).

$s^* \leftarrow \emptyset$
while `stopping criteria not met` **do**
  $s \leftarrow$ `GreedyRandomizedConstruction()`
  $s \leftarrow$ `LocalSearch(`s`)`
  if $f(s) > f(s^*)$ **then**
    $s^* \leftarrow s$
  end
end
return s^*

Algorithm 2.6: Ant Colony Optimization

Input : Population ($\mathcal{S}$), Pheromone Update Rule ($\mathcal{R}$), Pheromone Values ($\vec{\tau}$), Evaporation Rate (α)
Output: Solution (s^*)

$s^* \leftarrow s'$
$\mathcal{S} \leftarrow \{\emptyset\}$
while `not stopping criteria met` **do**
  $\mathcal{S} \leftarrow$ `AntBasedSolutionConstruction(`$\mathcal{P}, \vec{\tau}$`)`
  $\mathcal{S} \leftarrow$ `LocalSearch(`$\mathcal{P}$`)` ▷ `Optional`
  $s \leftarrow \operatorname{argmax}_{s \in \mathcal{P}} f(s)$ ▷ `Select best solution`
  if $f(s) > f(s^*)$ **then**
    $s^* \leftarrow s$;
  end
  $\mathcal{S} \leftarrow$ `PheromoneUpdate(`$\mathcal{P}, \mathcal{R}, \vec{\tau}, \alpha$`)`
end
return s^*

2.3.5 Hill-Climbing

Hill Climbing (HC) [24, 14] is a simple stochastic, memory-less, LS trajectory algorithm that works by iteratively attempting to improve a starting solution through a sequence of incremental changes, *i.e.*, by selecting the solution in the neighborhood that yields the best increment with respect to the objective value. This process terminates when the best solution possible is found or another stopping criteria is met. Despite the simplicity of this method it is worth noting that, due to the inherent greedy choice of the best at each step this approach is susceptible to getting trapped in local optima. For illustration purposes the pseudocode of simple version of an HC method is shown in Algorithm 2.7.

Algorithm 2.7: Hill Climbing

Input : Solution (s'), Objective Function (f)
Output : Solution (s^*)

$s^* \leftarrow s'$
while `stopping criteria not met` **do**
    $s \leftarrow$ `Perturb(`s^*`)`
    if $f(s) > f(s^*)$ **then**
        $s^* \leftarrow s$
    end
end
return s^*

It's worth noting that the presented algorithm's effectiveness is rooted in the efficiency of a random process, which strives to improve solution quality through a sequence of perturbation attempts (`Perturb` function). Nevertheless, there exist two noteworthy variations that more thoroughly explore the solution's neighborhood and take steps in the direction of the most substantial improvement. These are commonly known as First Improvement (FI) and Best Improvement (BI). The latter is also known in the literature as *steepest ascent* HC [24].

In a FI approach, the first neighboring solution that improves the current solution's quality is retained. Conversely, in a BI scenario, the entire neighborhood of the current solution is examined, and the best neighboring solution is selected for the subsequent iteration.

2.3.6 Iterated Local Search

Iterated Local Search (ILS) [18, 24, 10]

Algorithm 2.8: Iterated Local Search

Input : Solution (s'), Objective Function (f)
Output: Solution (s^*).

```
s* ← s'
while stopping criteria not met do
    s ← LocalSearch(s*)
    s ← Perturb(s)
    if f(s) > f(s*) then
        s* ← s
    end
end
return s*
```

2.3.7 Simulated Annealing

Simulated Annealing (SA) [2, 19, 4]

Algorithm 2.9: Simulated Annealing

Input : Solution (s'), Initial Temperature (t_0), Cooling Rate (α), Objective Function (f)
Output: Solution (s^*)

```
t ← t0
s* ← s'
while stopping criteria not met do
    s ← Perturb(s*)
    δ ← f(s*) - f(s)
    if δ < 0 ∨ Random(0, 1) < e^(-δ/t) then
        s* ← s
    end
    t ← t · α
end
return s*
```

2.3.8 Tabu Search

Tabu Search (TS) [9, 16, 24]

2.3.9 Outline

2.4 Modelling

Algorithm 2.10: Tabu Search

Input : Solution (s'), Tabu Length (l_{max}), Objective Function (f)
Output : Solution (s^*)

$s^* \leftarrow s'$
$\mathcal{T} \leftarrow \{\emptyset\}$
while `stopping criteria not met` **do**
    $s \leftarrow \underset{s \in \mathcal{N}(s) \setminus \mathcal{T}}{\operatorname{argmax}} f(s)$ ▷ `Select best neighbor` $s \notin \mathcal{T}$
    if $f(s) > f(s^*)$ **then**
        $s^* \leftarrow s$
        $\mathcal{T} \leftarrow \mathcal{T} \cup \{s\}$
    end
    if $|\mathcal{T}| > l_{\max}$ **then**
        `RemoveOldest(`$\mathcal{T}$`)`
    end
end
return s^*

Chapter 3

Google Hash Code Competition

"understanding a question is half an answer"

— Socrates

This chapter presents an overview of the Google Hash Code competition. In Section 3.1, we provide a concise review of the competition, encompassing both its historical background and format. Subsequently, in Section 3.2, we delve into the problems presented to participants over the years, attempting to categorize them and establish connections with well-known combinatorial optimization problems described in the literature. Moving forward, **??** sheds light on the design of competition instances. Concluding this chapter, Section 3.4 offers remarks that highlight key aspects of the competition problems, deemed pertinent to this work.

3.1 History & Format

The Google Hash Code competition, organized by Google from 2014 to 2022, consisted of two main phases: a qualifying round and a final round. During this competition, teams of 2-4 skilled individuals were tasked with solving complex problems that mirrored real-world engineering challenges faced by Google's own engineers. The primary aim of the competition was to attract talented individuals to the company. In the qualifying round, participants worldwide engaged in a 4-hour problem-solving session. Subsequently, around 40-50 select teams advanced to the final round, which took place at a Google headquarters. Additionally, participants gathered at designated hubs globally during the qualifying round, fostering a competitive environment.

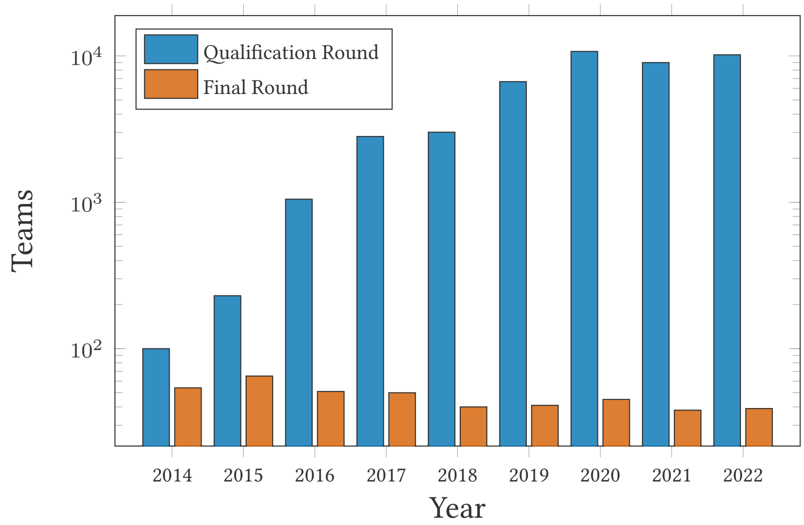

Figure 3.1: Google Hash Code Competition Attendance 2014-2022

In the early years of the competition, it was only open to teams from France. In the subsequent three years, it was open to teams from Europe, Africa, and the Middle East before becoming a worldwide competition. Therefore, it is expected that there will be an increase in the number of results available in the later years and more challenging problems due to the increase in competition. Furthermore, the number of participants kept growing throughout the years which highlights the importance of this event as illustrated in Figure 3.1

Unfortunately, this year Google decided to cease all its coding competitions including Hash Code. Nevertheless, the competition generated a diverse collection of attempts at solving the problems, resulting in a valuable wealth empirical data accessible to the community. As the official coding competition website is no longer accessible, Google created a repository containing all problem statements and instances distributed under an open source license [34]. However, it's worth noting that the scores achieved by participants are not integrated into this repository. Instead, they are documented across various blog posts and third-party websites [35].

It is worth noting, that due to the nature of the problems and format of the competition, participants frequently made use of heuristic and meta-heuristic strategies to solve the problems as best as possible in the alloted time. Moreover, the majority of competition problems are structurally different from one other, which makes it demanding to write general-purpose heuristic solvers that can be easily reused. Hence, it's a common practice for competitors to use solvers that are easily implementable or readily available online, given that internet access is permitted.

3.2 Problems

Its main objective is to provide the reader with a comprehensive understanding of the relevant details of the problems that drive connections with well-studied topics in combinatorial optimization literature instead of and exhaustively review the problem details, as the problem statements consulted in the coding [34].

3.2.1 Hash Code 2014

Street View Routing

In the context of constructing street view maps there is a need to collect imagery that is taken by specialized vehicles equipped for that purpose. This constitutes a challenging problem since given a fleet of cars which may only be available for a limited amount of time a route for each must be defined as to maximize the number of streets photographed. City streets are modelled as a graph where nodes are junctions and the edges are streets connecting said junctions. Moreover, streets are defined by three distinct properties: direction, length and cost that will take for the car to traverse the street.

The challenge consists of scheduling the routes for street view cars in the city, adhering to a pre-determined time budget. The goal is to optimize the solution by maximizing the sum of the lengths of the traversed streets, while minimizing the overall time expended in the process. The quality of the solution for this problem is evaluated by using the sum of the lengths of the streets as the primary criterion and the time spent as a tie-breaker.

The problem at hand bears a strong resemblance to a combination of the Vehicle Routing Problem and the Maximum Covering Problem. This is because the scheduling of routes for the fleet of cars must be done in a way that ensures that the combination of all sets of streets visited by each car encompasses the entire city, in the most time-efficient manner possible.

3.2.2 Hash Code 2015

Optimise a Data Center

The optimization of server placement problem is a concern that pertains to the design of data centers, as various factors must be taken into account to ensure optimal efficiency. In this context, the ‘optimizing servers” problem portrays a scenario in which contestants are in the position of designing a data center and seeking to determine the optimal distribution of servers. The data center is physically organized in rows of slots where servers can be placed. Hence, the challenge is to efficiently fill the available slots in a Google data center with servers of varying sizes and computing capacities, while also ensuring that each server is assigned to a specific resource pool.

Objectively, the goal is to assign multiple servers to available slots and resource pools in such a way as to maximize the guaranteed capacity for all resource pools. This metric serves as the criterion for evaluating solutions to this problem. The guaranteed capacity, in this context, refers to the lowest amount of computing power that will remain for a specific resource pool in the event of a failure of an arbitrary row of the data center. It is important to note that this objective function is considered a bottleneck, as small changes in a solution may not result in significant changes in the score, making the optimization process more difficult.

Notably, the problem of optimization the placement of servers in a data center can be thought of as a combination of a Multiple-Knapsack Problem and an assignment problem. This is because the servers must be placed within the constraints of the available space in the data center rows, and subsequently, they must be assigned to resource pools.

Loon

Project *Loon*, which was a research endeavor undertaken by Google, aimed at expanding internet coverage globally by utilizing high altitude balloons. The problem presented in this competition drew inspiration from this concept, requiring contestants to devise plans for position adjustments for a set of balloons, taking into consideration various environmental factors, particularly wind patterns, with the objective of ensuring optimal internet coverage in a designated region over a specific time frame.

The objective of this problem was to develop a sequence of actions, including ascent, descent, and maintaining altitude, for a set of balloons with the goal of maximizing a score. In this case, the score is calculated based on the aggregate coverage time of each location, represented as cells on a map of specified dimensions, at the conclusion of the available time budget.

In summary, this problem can be classified as both a simulation and a coverage and routing problem, based on the properties previously described. It is important to note that the simulation aspect of this problem has a direct impact on the calculation of the score, and is not solely limited to constraints on the available time budget for operations. Furthermore, this problem can be represented in a forest, where the vertices represent spatiotemporal coordinates (x, y, z, t), and the edges symbolize changes in altitude and lateral movement (wind) for a given balloon.

3.2.3 Hash Code 2016

Delivery

In today's world, with the widespread availability of internet, online shopping has become a prevalent activity. As a consequence, there is an ever-growing need for efficient delivery systems. This competition challenges participants to manage a fleet of drones, which are to be used as vehicles for the distribution of purchased goods. Given a map with delivery locations, a set of drones, each with a set of operations that can be performed (load, deliver, unload, wait), a number of warehouses, and a number of orders, the objective is to satisfy the orders in the shortest possible time, taking into consideration that the products to be delivered in an order may have product items stored in multiple different warehouses and therefore require separate pickups by drones.

In this problem, the simulation time $\mathcal{T}$ is given and the goal is to complete each order within that time frame. The score for each order is calculated as $\frac{(\mathcal{T}-t)}{\mathcal{T}} \times 100$, where t is the time at which the order is completed. The score ranges from 1 to 100, with higher scores indicating that the order was completed sooner. The overall score for the problem is the sum of the individual scores for all orders, and it is to be maximized.

In summary, this problem can be classified as a variant of the Vehicle Routing Problem, specifically as a Capacitated, Pickup and Delivery Time Windowed Multi-Depot Vehicle Routing Problem. This classification takes into account the pickup and delivery of items, the time window for delivery, the multiple routes and warehouses that each vehicle may need to visit in order to fulfill the orders.

Satellites

Terra Bella was a Google division responsible for managing and operating a constellation of satellites that collected and processed imagery for commercial purposes. Specifically, these satellites were tasked with capturing images in response to client requests.

The challenge presented to participants involves crafting schedules for individual satellites within the fleet. The goal is to secure image collections that match customer preferences. These collections are characterized by geographical coordinates on Earth and specific time windows for image capture. Each satellite, originating from unique latitude and longitude coordinates and possessing a certain velocity, possesses the ability to make minor positional adjustments along both axes to access potential photography sites. The problem's score is determined by aggregating the points earned through the successful completion of customer collections. In this context, completion signifies capturing all images for a given collection within the designated time frame.

In essence, this problem falls into the categories of both an assignment and a maximum covering problem. It involves not only covering the maximum number of images with the available satellites to complete collections, but also making decisions about which satellites will capture each photo. Additionally, the simulation aspect is crucial as it directly affects scoring; images not taken within the specified time frame won't contribute to the collection, potentially influencing its completion and the overall score.

3.2.4 Hash Code 2017

Streaming Videos

In the era of online streaming services like YouTube, effectively distributing content to users is crucial. This challenge focuses on optimizing video distribution across cache servers to minimize transmission delays and waiting times for users. Contestants must strategize video placement within servers while considering space limitations.

With a roster of videos, each assigned a specific size, an collection of cache servers with designated space, and an index of endpoints initiating multiple requests for various videos, this challenge entails determining an optimal video assignment within servers. The time saved for each request is measured as the difference between data center streaming time and cache server streaming time with minimal latency. The overall score is computed by summing the time saved for each request, multiplied by 1000, and then divided by the total request count. It's important to note that the problem description offers transmission latencies between different nodes.

In general, we categorize this problem as a combination of assignment and knapsack problems. Contestants are tasked not only with determining the allocation of videos to servers but also with accounting for capacity limitations on the number of videos per server. It's worth noting that the calculation of time saved for each request may encounter a bottleneck effect, which can pose challenges when optimizing the overall score.

Router Placement

Strategically optimizing the placement of Wi-Fi routers to achieve optimal signal coverage is a challenge encountered by many institutions and individual users. This issue becomes particularly prominent in larger and complex buildings. Furthermore, in such scenarios, the task may involve setting up a wired connection to establish internet connectivity from the source point, facilitating the strategic positioning of routers for maximum coverage.

The challenge tasked participants with optimizing the arrangement of routers and fibber wiring within a building's cell-based layout, along with a designated backbone connection point. The aim was to strategically position routers and devise an effective wiring configuration. The primary goal encompassed achieving optimal coverage while adhering to a predefined budget. The problem's score comprised two components: the count of cells covered by routers, multiplied by 1000, and the remaining budget. Notably, the scoring approach emphasized both extensive coverage and economical budget allocation.

In essence, this problem falls under the category of a maximum covering problem, as the central aim is to ensure the coverage of as many cells as possible. Furthermore, considering the budget limitations and wiring arrangement, we observe that this challenge shares similarities with the Steiner Tree Problem. This likeness arises from the possibility of determining the optimal cost of wiring placement based on the router locations, which may hold significance for the problem's resolution.

3.2.5 Hash Code 2018

Self-Driving Rides

Daily car commuting is a ubiquitous practice globally, involving trips to homes, schools, workplaces, and more. As a means of travel, cars remain a common choice, with ongoing efforts to enhance safety through the advancement of self-driving technology. In this challenge, contestants assume the role of managing a fleet of self-driving cars within a simulated setting. The goal is to ensure commuters reach their destinations securely and punctually.

With a fleet of cars at disposal and a roster of rides defined by their starting and ending intersections on a square grid representing the city, along with the earliest start time and the latest end time to ensure punctuality, the task is to allocate rides to vehicles. The aim is to maximize the number of completed rides before a predefined simulation time limit is reached. The scoring is determined by the summation of the individual ride scores. A ride's score is computed as the sum of a value proportional to the distance covered during the ride, augmented by a bonus if the ride commences precisely at its earliest allowed start time.

Generally, this problem can be categorized as an assignment and vehicle routing problem with time windows. This classification arises from the necessity to assign rides to cars within specific time constraints. Notably, the car's route is determined by the sequence of rides assigned to it. Moreover, this challenge falls under the simulation category, as it directly impacts the scoring mechanism and cannot be simplified or abstracted.

City Plan

With the world's population increasingly concentrating in urban areas, the demand for expanded city infrastructure is on the rise. This entails not only residential buildings but also the incorporation of essential public facilities and services to cater to the growing populace. This challenge mirrors a scenario where participants are tasked with planning a city's building layout, involving both the selection of building types and their strategic placement.

For this challenge, participants receive building projects with specific width and height dimensions, covering both residential and utility structures. The city is a square grid of cells. Overall, the goal is to create buildings from these plans, arranging them within the city to optimize space and create a balanced mix of structures. This minimizes residents' walking distance to reach essential services. The overall score is the sum individual residential building scores, calculated by multiplying the number of residents and the number of utility building types within walking distance of that building. Notably, the walking distance parameter is specific to each problem instance.

Essentially, this problem belongs to the category of packing problems. The core objective revolves around determining how to fit buildings within the city layout. Importantly, there is no predetermined limit on the number of buildings that can be constructed for each plan, granting contestants the flexibility to make choices accordingly.

3.2.6 Hash Code 2019

Photo Slideshow

Given the surge in digital photography and the vast number of images traversing the internet daily, this challenge delves into the interesting concept of crafting picture slideshows using the available photo pool.

In this scenario, participants were tasked with creating a slideshow composed of pictures, which could be oriented either vertically or horizontally in the slides. Notably, a slide could contain two photos if they were arranged vertically. Additionally, these photos could be tagged with multiple descriptors corresponding to their subjects. The scoring of this problem revolves around the slideshow's appeal, determined by a calculated value that depends on consecutive slide pairs. This value is computed as the minimum between the tags count of the first picture, the second picture in the sequence and the count of the common tags shared between the two images.

Overall, this challenge can be categorized as a scheduling problem, to be precise, a single-machine job scheduling problem. If we liken the jobs to photos, the goal is to sequence them to optimize a specific objective function in this context, the "appeal" factor. Additionally, the interactions between slides introduce elements resembling a grouping problem.

Compiling Google

Given Google's extensive codebase spanning billions of lines of code across numerous source code files, compiling these files on a single machine would be time-consuming. To address this, Google distributes the compilation process across multiple servers.

This challenge tasks participants with optimizing compilation time by strategically distributing source code files across available servers. Notably, the compilation of a single code file can depend on other files being compiled prior to it, involving dependencies. Given a certain number of available servers and specific deadlines for compilation targets, the problem's score is calculated by summing the scores for the completion of each compilation target. These scores are determined by a fixed value for meeting the deadline, with an additional bonus if the compilation is completed ahead of the expected time.

This problem can be categorized as a scheduling problem, as the primary objective involves distributing compilation tasks (jobs) among different machines while adhering to dependencies between files. In essence, this problem resembles a variation of the classical job-shop scheduling problem.

3.2.7 Hash Code 2020

Book Scanning

Google Books is project that aims to create a digital collection of many books by scanning them from libraries and publishers around the world. In this challenge, contestants are put in the position of managing the operation of setting up a scanning pipeline for millions of books.

Given a dataset describing libraries and available books, the objective of this challenge is to select books for scanning from each library within a specified global deadline. Each library has a distinct sign-up process duration before it can commence scanning, and only one library can be signed up at a time. Moreover, each library has a fixed scanning rate for books per day, and each scanned book contributes to the final score. The problem's goal is to maximize the overall score, which is calculated as the sum of the scores for unique books scanned within the given deadline.

This problem exhibits a combination of characteristics from classical scheduling, assignment, covering, and knapsack problems. It resembles a scheduling problem as the order in which libraries are signed up needs to be determined. It involves assignment, since libraries can share books, necessitating a decision on which libraries will scan each book. The covering aspect is apparent in the scoring mechanism, where the aim is to maximize the number of unique books scanned. Lastly, the problem also incorporates a knapsack-like element. While the time-related simulation factor exists, it can be abstracted into a knapsack scenario where the goal is to optimize the overall score by considering the number of books a library can scan until the deadline as its capacity.

Assembling Smartphones

Constructing smartphones is a intricate process that entails assembling a multitude of hardware components. This challenge delves into the concept of creating an automated assembly line for smartphones, employing robotic arms to streamline the manufacturing process.

Contestants are tasked with placing robotic arms within a workspace depicted as a rectangular cell grid. The objective is to optimize the arrangement of these arms to allow the execution of assigned tasks. Each task involves specific movements that a robotic arm must perform, essentially traversing a designated number of cells to accomplish the task. Notably, robotic arms cannot cross each other, necessitating precise task assignment and arm positioning to ensure unobstructed task execution for all arms. The problem's score is derived from the summation of scores obtained by successfully completing tasks within the constraints.

In summary, this challenge falls under the category of both assignment and scheduling problems since it encompasses the assignment of robotic arms to suitable positions and the scheduling of tasks across these arms to optimize the completion of tasks.

3.2.8 Hash Code 2021

Traffic Signaling

This challenge delves into the optimization of traffic light timers to enhance the travel experience in a city. While traffic lights inherently contribute to road safety, their built-in timers are important in regulating traffic flow. The focus here is to fine-tune these timers with the aim of optimizing overall travel time for all commuters within the city.

Contestants are presented with a city layout, complete with intersections housing traffic lights. The task is to strategically allocate time intervals to these traffic light timers, optimizing traffic flow to ensure the maximum number of car trips are successfully completed within a predefined simulation time limit. The problem's score is the cumulative sum of scores assigned to each completed trip. These scores comprise a fixed value for trip completion and a bonus proportional to how early the trip concludes relative to the simulation's time limit. While the challenge may seem complex due to its detailed rules and operational aspects, its core objective revolves around this fundamental optimization process.

In summary, this challenge can be categorized as a simulation problem. It's worth highlighting that this problem aligns closely with the Signal Timing problem in the literature of Control Optimization.

Software Engineering at Scale

This challenge addresses the complexity of managing Google's vast monolithic codebase, which has grown significantly alongside the expanding number of engineers. To overcome the hurdles of effective feature deployment, participants are tasked with creating a solution that optimally schedules feature implementation work among engineers.

In this challenge, there are three primary components to be considered: features, services, and binaries. Each feature may require certain services, which can be present in specific binaries. The main objective is to efficiently assign features to engineers, considering that their implementation might entail additional tasks such as service implementation, binary relocation, new binary creation, or waiting for a designated time. The challenge revolves around optimizing this workflow to minimize delays caused by multiple engineers working in the same service. The scoring is based on the sum of scores awarded for feature completion. Each completed feature's score is determined by the product of the number of users benefiting from it, as specified in the problem statement, and the number of days between the maximum day (also defined) and the day the feature was launched.

In essence, this challenge falls within the realm of classic scheduling problems. It involves assigning tasks (jobs) to engineers with the aim of optimizing a quantity influenced by the order in which each engineer performs their tasks and the interactions of tasks among multiple engineers.

3.2.9 Hash Code 2022

Mentorship and Teamwork

This challenge delves into the concept of a teamwork environment, where knowledge sharing among peers and collaborative efforts are central to task completion. In this challenge, participants are tasked with orchestrating a team comprising individuals with diverse backgrounds to successfully execute projects that demand a variety of skills.

The main goal is to efficiently assign a list of contributors, each possessing specific skills and the potential to improve them through project involvement, mentoring, or being mentored. The challenge involves allocating contributors to projects with skill requirements to ensure timely completion. Notably, contributors can participate in multiple projects concurrently. The key factor here is the order in which contributors develop or enhance their skills, a decision that significantly impacts the overall project completion process. The score in this challenge is the sum of project scores achieved by completing them before the defined overall deadline. A project's score comprises a fixed value for completion, minus penalty points if it surpasses the deadline but is still within a tolerance window. Projects exceeding the deadline or tolerance won't add to the score but will still contribute to workers' training.

In summary, this challenge shares similarities with a scheduling problem, as it involves assigning projects to contributors while considering the order in which they are completed to maximize the overall score achieved through project completion.

Santa Tracker

The *Google Santa Tracker* is a project that visualizes the route taken by the famous *Santa Claus* character during his December gift distribution to children globally. In this challenge, participants were tasked with optimizing the delivery route to enhance the efficiency of gift distribution.

The challenge scenario revolves around a 2D cell grid with no friction, symbolizing the world. Within this grid, children are located, and two types of items, carrots (providing speed boosts) and gifts, can be picked up by the cart. While the cart maintains its speed on the frictionless grid, the total weight affects the impact of carrot consumption. Thus, the main goal consists in devising a route that efficiently delivers the most gifts within the time constraints. The scoring metric for this problem involves summing the scores of successfully delivered items.

In summary, this challenge can be categorized as a type of Vehicle Routing Problem, specifically a Capacitated with Pick up and Delivery Vehicle Routing Problem. This is due to the presence of capacity constraints on the cart and the need to pick up and deliver items throughout the cart's journey.

3.2.10 Outline

In summary, this section provided an overview and description of the key aspects of the Hash Code problems. Furthermore, a categorization that links these problems to topics commonly found in combinatorial optimization literature was presented. The Table 3.1 shows a summary of the analysis conducted.

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 3.1: Categorization of Google Hash Code Problems

3.3 Instances

In the competition context, in combination with the problem statements, test case instances are provided to participants with the primary aim of providing a mechanism for scoring teams, thus quantitatively assessing the efficacy of their strategies. These instances are carefully generated to conform to the stipulated limits and constraints inherent to the challenge, as described in upon in the problem statement.

The initial instance, commonly denoted as the “example”, is routinely included within the problem statement for contestants’ reference. This instance is included in the problem statement for contestants’ reference, but is not solved optimally. Its purpose is to illustrate the input and output format for the instance and solution. However, the example is intentionally designed with small dimensions, making it approachable via exact brute force methodologies.

Subsequent instances are typically designed to push the boundaries of the problem. These instances are intentionally large and design to discourage exact methods and general heuristics, aiming to thoroughly examine various aspects of the problem and avoid that (non-exact) greedy approaches find an optimal solution. The ruggedness of their objective space introduces challenges for solvers, potentially rendering some of them ineffective or even unusable within the available time budget.

In the competition context, teams are allowed to provide unique solutions for each instance, thus becoming a common practice among participants to conduct thorough cross-instance analysis. This practice proves valuable in revealing patterns that can offer insights into tackling the challenge with greater efficiency. As such, participants have the flexibility to develop focused strategies for each instance. This can in fact be interesting for the study of general-purpose meta-heuristics, to understand whether they can achieve comparable results to instance-specific approaches.

Finally, given the articulated problem statements and the transparent instance generation process, participants can create customized test instances. This capability proves valuable for debugging purposes in a competition setting and further advocates these problems as interesting benchmarks for black-box optimization [28].

3.4 Concluding Remarks

In this chapter, we conducted a comprehensive exploration of the Google Hash Code competition, delving into its structure, problem descriptions, and instances. In particular, we established links between the challenges presented and well-known CO problems. Furthermore, we highlighted common techniques employed by participants, drawing from our own engagement over several years.

We consider this analysis to be an important step that not only facilitated a deeper comprehension of the challenges, but also guided our choice of two specific problems (Optimise a Data Center and Book Scanning) for detailed exploration in this study. The particular choice of these problems is mainly motivated by the range of combinatorial optimization topics covered, leaving only vehicle routing and simulation as subjects to address in future work.

Moreover, based on the conducted analysis, we once again emphasize the importance of these problems as promising candidates for BBO [28]. However, we believe that for this potential to be realized, it is essential to establish a repository containing the scores achieved across various problems and instances. Ideally, this repository should be accompanied by the corresponding source code for reproducibility purposes. From the standpoint of experimentally evaluating meta-heuristics for these problems, we consider it vital to generate a diverse set of instances, eventually generated through different methods.

Having understood the Google Hash Code problems, in the ensuing chapters, we will discuss our modelling approach to solve them, analyze the chosen problems, and conclude with a reflection on the work carried out.

Chapter 4

Principled Modelling Framework

Chapter 5

Optimize a Data Center Problem

Chapter 6

Book Scanning Problem

Chapter 7

Conclusion

Acronyms

ACO Ant Colony Optimization. ix, 17, 18

BBO Black-Box Optimization. 12, 36

BI Best Improvement. 19

BS Beam Search. ix, 17

CO Combinatorial Optimization. 2, 3, 5–9, 13, 14, 35

CS Constructive Search. ix, 14, 15

FI First Improvement. 19

GBO Glass-Box Optimization. 12

GO Global Optimization. 10–12

GRASP Greedy Randomized Adaptive Search Procedure. ix, 17, 18

HC Hill Climbing. ix, 19

IG Iterated Greedy. ix, 17

ILP Integer Linear Programing. 12

ILS Iterated Local Search. ix, 19, 20

KP Knapsack Problem. 7, 8, 12

LO Local Optimization. 10–12

LS Local Search. ix, 15, 16, 19

MH Meta-Heuristic. 2, 3, 6, 8, 9, 14, 16

SA Simulated Annealing. ix, 20

TS Tabu Search. ix, 20, 21

Bibliography

[1] David S. Johnson. "Approximation Algorithms for Combinatorial Problems". In: *Journal of Computer and System Sciences* 9.3 (Dec. 1, 1974), pp. 256–278. ISSN: 0022-0000. DOI: 10.1016/S0022-0000(74)80044-9.

[2] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. "Optimization by Simulated Annealing". In: *Science* 220.4598 (May 13, 1983), pp. 671–680. DOI: 10.1126/science.220.4598.671.

[3] PENG SI OW and THOMAS E. MORTON. "Filtered Beam Search in Scheduling†". In: *International Journal of Production Research* 26.1 (Jan. 1, 1988), pp. 35–62. ISSN: 0020-7543. DOI: 10.1080/00207548808947840.

[4] R. W. Eglese. "Simulated Annealing: A Tool for Operational Research". In: *European Journal of Operational Research* 46.3 (June 15, 1990), pp. 271–281. ISSN: 0377-2217. DOI: 10.1016/0377-2217(90)90001-R.

[5] J.-B. Hiriart-Urruty. "Conditions for Global Optimality". In: *Handbook of Global Optimization*. Ed. by Reiner Horst and Panos M. Pardalos. Nonconvex Optimization and Its Applications. Boston, MA: Springer US, 1995, pp. 1–26. ISBN: 978-1-4615-2025-2. DOI: 10.1007/978-1-4615-2025-2_1.

[6] Ibrahim H. Osman and Gilbert Laporte. "Metaheuristics: A Bibliography". In: *Annals of Operations Research* 63.5 (Oct. 1, 1996), pp. 511–623. ISSN: 1572-9338. DOI: 10.1007/BF02125421.

[7] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Courier Corporation, Jan. 1, 1998. 530 pp. ISBN: 978-0-486-40258-1.

[8] Jens Clausen. "Branch and Bound Algorithms - Principles and Examples." In: *Branch and Bound Algorithms - Principles and Examples.* (1999).

[9] Fred Glover and Manuel Laguna. *Tabu Search I*. Vol. 1. Jan. 1, 1999. ISBN: 978-0-7923-9965-0. DOI: 10.1287/ijoc.1.3.190.

[10] Christian Blum. "Metaheuristics in Combinatorial Optimization". In: *ACM Computing Surveys* 35.3 (2003).

[11] Luca Di Gaspero and Andrea Schaerf. "EasyLocal++: An Object-Oriented Framework for the Flexible Design of Local-Search Algorithms". In: *Software—Practice & Experience* 33.8 (July 10, 2003), pp. 733–765. ISSN: 0038-0644. DOI: 10.1002/spe.524.

[12] S. Cahon, N. Melab, and E.-G. Talbi. "ParadisEO: A Framework for the Reusable Design of Parallel and Distributed Metaheuristics". In: *Journal of Heuristics* 10.3 (May 2004), pp. 357–380. ISSN: 1381-1231. DOI: 10.1023/B:HEUR.0000026900.92269.ec.

[13] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer New York, 2006. ISBN: 978-0-387-30303-1. DOI: 10.1007/978-0-387-40065-5.

[14] Ana Vieira. "Uma plataforma para a avaliação experimental de meta-heurísticas". Doctoral Thesis. University of Algarve, Portugal, 2009.

[15] Marco Dorigo and Thomas Stützle. "Ant Colony Optimization: Overview and Recent Advances". In: *Handbook of Metaheuristics*. Ed. by Michel Gendreau and Jean-Yves Potvin. International Series in Operations Research & Management Science. Boston, MA: Springer US, 2010, pp. 227–263. ISBN: 978-1-4419-1665-5. DOI: 10.1007/978-1-4419-1665-5_8.

[16] Michel Gendreau and Jean-Yves Potvin. "Tabu Search". In: *Handbook of Metaheuristics*. Ed. by Michel Gendreau and Jean-Yves Potvin. International Series in Operations Research & Management Science. Boston, MA: Springer US, 2010, pp. 41–59. ISBN: 978-1-4419-1665-5. DOI: 10.1007/978-1-4419-1665-5_2.

[17] Walter J. Gutjahr. "Stochastic Search in Metaheuristics". In: *Handbook of Metaheuristics*. Ed. by Michel Gendreau and Jean-Yves Potvin. International Series in Operations Research & Management Science. Boston, MA: Springer US, 2010, pp. 573–597. ISBN: 978-1-4419-1665-5. DOI: 10.1007/978-1-4419-1665-5_19.

[18] Helena R. Lourenço, Olivier C. Martin, and Thomas Stützle. "Iterated Local Search: Framework and Applications". In: *Handbook of Metaheuristics*. Ed. by Michel Gendreau and Jean-Yves Potvin. International Series in Operations Research & Management Science. Boston, MA: Springer US, 2010, pp. 363–397. ISBN: 978-1-4419-1665-5. DOI: 10.1007/978-1-4419-1665-5_12.

[19] Alexander G. Nikolaev and Sheldon H. Jacobson. "Simulated Annealing". In: *Handbook of Metaheuristics*. Ed. by Michel Gendreau and Jean-Yves Potvin. International Series in Operations Research & Management Science. Boston, MA: Springer US, 2010, pp. 1–39. ISBN: 978-1-4419-1665-5. DOI: 10.1007/978-1-4419-1665-5_1.

[20] Mauricio G.C. Resende and Celso C. Ribeiro. "Greedy Randomized Adaptive Search Procedures: Advances, Hybridizations, and Applications". In: *Handbook of Metaheuristics*. Ed. by Michel Gendreau and Jean-Yves Potvin. International Series in Operations Research & Management Science. Boston, MA: Springer US, 2010, pp. 283–319. ISBN: 978-1-4419-1665-5. DOI: 10.1007/978-1-4419-1665-5_10.

[21] Xinjie Yu and Mitsuo Gen. "Combinatorial Optimization". In: *Introduction to Evolutionary Algorithms*. Ed. by Rajkumar Roy. Decision Engineering. London: Springer, 2010, pp. 263–324. ISBN: 978-1-84996-129-5. DOI: 10.1007/978-1-84996-129-5_7.

[22] Juan J. Durillo and Antonio J. Nebro. "jMetal: A Java Framework for Multi-Objective Optimization". In: *Advances in Engineering Software* 42.10 (Oct. 1, 2011), pp. 760–771. ISSN: 0965-9978. DOI: 10.1016/j.advengsoft.2011.05.014.

[23] David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms*. 1st ed. Cambridge University Press, Apr. 26, 2011. ISBN: 978-0-511-92173-5. DOI: 10.1017/CBO9780511921735.

[24] Sean Luke. *Essentials of Metaheuristics*. Second Edition. 2013.

[25] Rafael Martí, Mauricio G. C. Resende, and Celso C. Ribeiro. "Multi-Start Methods for Combinatorial Optimization". In: *European Journal of Operational Research* 226.1 (Apr. 1, 2013), p. 2. ISSN: 0377-2217. DOI: 10.1016/j.ejor.2012.10.012.

[26] P. Festa. "A Brief Introduction to Exact, Approximation, and Heuristic Algorithms for Solving Hard Combinatorial Optimization Problems". In: *2014 16th International Conference on Transparent Optical Networks (ICTON)*. 2014 16th International Conference on Transparent Optical Networks (ICTON). July 2014, pp. 1–20. DOI: 10.1109/ICTON.2014.6876285.

[27] Thomas Stützle and Rubén Ruiz. "Iterated Greedy". In: *Handbook of Heuristics*. Ed. by Rafael Martí, Panos M. Pardalos, and Mauricio G. C. Resende. Cham: Springer International Publishing, 2018, pp. 547–577. ISBN: 978-3-319-07124-4. DOI: 10.1007/978-3-319-07124-4_10.

[28] Thomas Bartz-Beielstein et al. *Benchmarking in Optimization: Best Practice and Open Issues*. July 7, 2020.

[29] Carola Doerr. "Complexity Theory for Discrete Black-Box Optimization Heuristics". In: *Theory of Evolutionary Computation: Recent Developments in Discrete Optimization*. Ed. by Benjamin Doerr and Frank Neumann. Natural Computing Series. Cham: Springer International Publishing,

2020, pp. 133–212. ISBN: 978-3-030-29414-4. DOI: 10.1007/978-3-030-29414-4_3.

[30] Stéphane Alarie et al. “Two Decades of Blackbox Optimization Applications”. In: *EURO Journal on Computational Optimization* 9 (Jan. 1, 2021), p. 100011. ISSN: 2192-4406. DOI: 10.1016/j.ejco.2021.100011.

[31] Carlos M. Fonseca. *Nasf4nio*. Oct. 27, 2021. URL: https://github.com/cmfonseca/nasf4nio (visited on 01/15/2023).

[32] Samuel Barroca do Outeiro. “An Application Programming Interface for Constructive Search”. Msc Thesis. University of Coimbra, Portugal, Nov. 9, 2021.

[33] Valentina Cacchiani et al. “Knapsack Problems — An Overview of Recent Advances. Part I: Single Knapsack Problems”. In: *Computers & Operations Research* 143 (July 2022), p. 105692. ISSN: 03050548. DOI: 10.1016/j.cor.2021.105692.

[34] Google LLC. *Coding-Competitions-Archive*. Aug. 19, 2023. URL: https://github.com/google/coding-competitions-archive (visited on 08/20/2023).

[35] Zibada. *Google Coding Competitions Archive*. 2023. URL: https://zibada.guru/gcj/#hc (visited on 08/20/2023).