1 Introduction to search and optimization

This chapter covers

- What are search and optimization?
- Why care about search and optimization?
- Going from "toy problems" to real-world solutions
- Defining an optimization problem
- Introducing well-structured problems and ill-structured problems
- Search algorithms and the search dilemma

Optimization is deeply embedded in nature and in the systems and technologies we build. Nature is a remarkable testament to the ubiquity and prevalence of optimization. Take, for instance, the foraging behaviors of social insects like ants and honeybees. They have developed their own unique optimization methods, from navigating the shortest path to an existing food source to discovering new food sources in an unknown external environment. Honeybee colonies focus their foraging efforts on only the most profitable patches. They cooperatively build their honeycombs with hexagonal structures for efficient use of space (the maximum number of cells that can be built in a given area), material efficiency (using less beeswax), structural strength, and the optimal angle to prevent honey from spilling

out of the cells. Similarly, birds exhibit optimization behavior during their annual migrations. They undertake long voyages from their breeding grounds to their winter homes, and their migration routes have been optimized over generations to conserve energy. These routes account for factors such as prevailing wind patterns, food availability, and safety from predators. These examples underscore how nature intuitively applies optimization strategies for survival and growth, offering us lessons that can be translated into algorithmic problem-solving.

Optimization is also a regular aspect of our daily lives, often so seamlessly integrated that we barely notice its constant influence. As human beings, we strive to optimize our everyday lives. Consider the simple act of planning your day. We instinctively order or group tasks or errands in a way that minimizes travel time or maximizes our free time. We navigate the challenge of grocery shopping within a budget, trying to get the most value out of every dollar spent. We create workout routines aiming for the maximum fitness benefits within our limited time. Even at home, we optimize our energy usage to keep our utility bills in check.

Likewise, corporations maximize profits by increasing efficiency and eliminating waste. For example, logistics giants like FedEx, UPS, and Amazon spend millions of dollars each year researching new ways to trim the cost of delivering packages. Telecommunications agencies seek to determine the optimal placement of crucial infrastructure, like cellular towers, to service the maximum number of customers while investing in the minimum level of equipment. Similarly, transportation network companies like Uber, Lyft, and DiDi route drivers efficiently during passenger trips and direct drivers to ride-hailing hotspots during idle periods to minimize passenger wait time. As urbanization intensifies worldwide, local emergency services depend on efficient dispatching and routing platforms to select and route the appropriate vehicles, equipment, and personnel to respond to incidents across increasingly complex metropolitan road networks. Airlines need to solve several optimization problems, such as flight planning, fleet assignment, crew scheduling, aircraft routing, and aircraft maintenance planning. Healthcare systems also handle optimization problems such as hospital resource planning, emergency procedure management, patient admission scheduling, surgery scheduling, and pandemic containment. Industry 4.0, a major customer of optimization technology, deals with complex optimization problems such as smart scheduling and rescheduling, assembly line balancing, supply-chain optimization, and operational efficiency. Smart cities deal with large-scale optimization problems such as stationary asset optimal assignments, mobile asset deployment, energy optimization, water control, pollution reduction, waste management, and bushfire containment.

These examples show how ubiquitous and important optimization is as a way to maximize operational efficiency in different domains. In this book, we'll dive into the exciting world of optimization algorithms. We'll unravel how these algorithms can be used to tackle complex continuous and discrete problems in different domains.

1.1 Why care about search and optimization?

Search is the systematic examination of states, starting from an initial state and ending (hopefully) at the goal state. Optimization techniques are in reality search methods, where the goal is to find an optimal or a near-optimal state within the feasible search space. This feasible search space is a subset of the optimization problem space where all the problem's constraints are satisfied. It's hard to come up with a single industry that doesn't already use some form of search or optimization methods, software, or algorithms. It's highly likely that in your workplace or industry, you deal with optimization daily, though you may not be aware of it. While search and optimization are ubiquitous in almost all industries, using complicated algorithms to optimize processes may not always be practical. For example, consider a small pizzeria that offers food delivery to its local customers. Let's assume that the restaurant processes around ten deliveries on an average weeknight. While efficiency-improving strategies (such as avoiding left turns in right-driving countries or right turns in left-driving countries, avoiding major intersections, avoiding school zones during drop-off and pick-up times, avoiding bridges during lift times, and favoring downhill roads) may theoretically shorten delivery times and reduce costs, the scale of the problem is so tiny that implementing these kinds of changes may not lead to any noticeable effect.

In larger-scale problems, such as fleet assignment and dispatching, multicriteria stochastic vehicle routing, resource allocation, and crew scheduling, applying search and optimization techniques to a problem must be a qualified decision. Some firms or companies may not benefit from excessive process changes due to a lack of expertise or resources to implement those changes. There may also be concerns about a potential lack of follow-through from stakeholders. Implementing these changes could also cost more than the savings obtained through the optimization process. Later in this book, we will see how these costs can be accounted for when developing search and optimization algorithms.

This book will take most anyone from never having solved search and optimization problems to being a well-rounded search and optimization practitioner, able to select, implement, and adapt the right solver for the right problem. It doesn't assume any prior knowledge of search and optimization and only an intermediate knowledge of data structures and Python. For managers or professionals involved in high-level technological decisions at their workplace, these skills can be critical in understanding software-based approaches, their opportunities, and their limitations when discussing process improvement. In contrast, IT professionals will find these skills directly applicable when considering options for developing or selecting new software suites and technologies for in-house use. The following section describes the methodology we will follow throughout this book.

1.2 Going from toy problems to the real world

When discussing algorithms, many books and references present them as formal definitions and then apply them to so-called "toy problems." These trivial problems are helpful because they often deal with smaller datasets and search spaces while being solvable by hand iteration. This book follows a similar approach but takes it one step further by presenting real-world data implementations. Whenever possible, resources such as datasets and values are used to illustrate the direct applicability and practical drawbacks of the algorithms discussed. Initially, the scaled-down toy problems will help you appreciate the step-by-step operations involved in the various algorithms. Later, the Python implementations will teach you how to use multiple datasets and Python libraries to address the increased complexity and scope of real-world problems.

As illustrated in figure 1.1, the source of inspiration for each search or optimization algorithm is identified, and then the algorithm pseudocode, algorithm parameters, and heuristics/solution strategies used are presented. The algorithm's pros and cons and adaptation methods are then described. This book contains many examples that will allow you to carry out iterations by hand on a scaled-down version of the problem and fully understand how each algorithm works. It also includes many programming exercises in a special problem-solution-discussion format so you can see how a scaled-up version of the problem previously solved by hand can be solved using Python. Through programming, you can optimally tune the algorithm and study its performance and scalability.

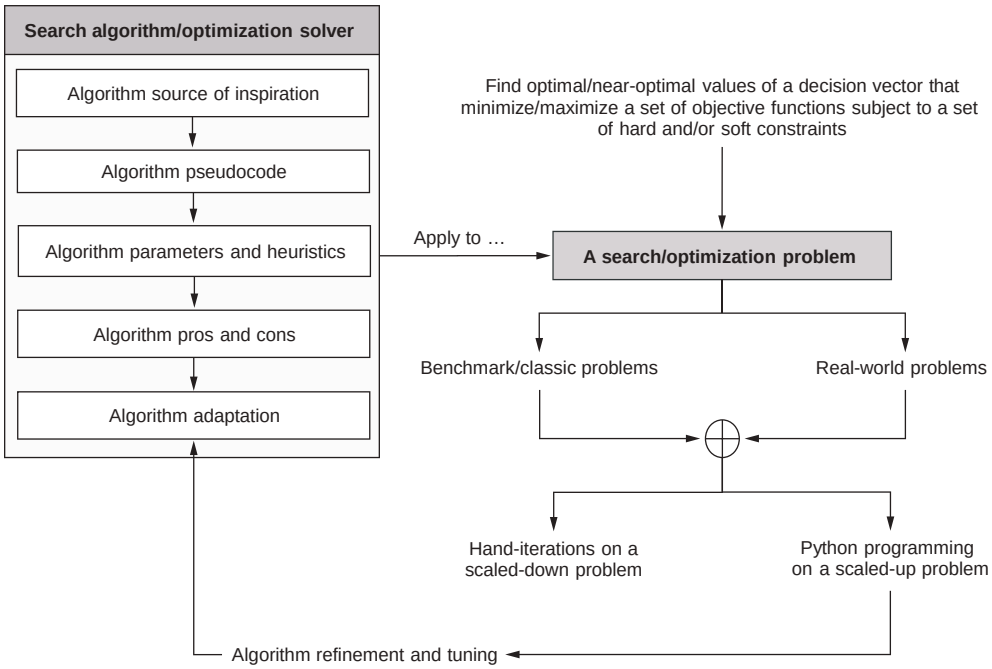


Figure 1.1 The book's methodology—each algorithm will be introduced following a pattern that goes from explanation to application.

Throughout this book, several classic and real-world optimization problems will be considered to show you how to use the search and optimization algorithms discussed in the book. Figure 1.2 shows examples of these optimization/search problems.

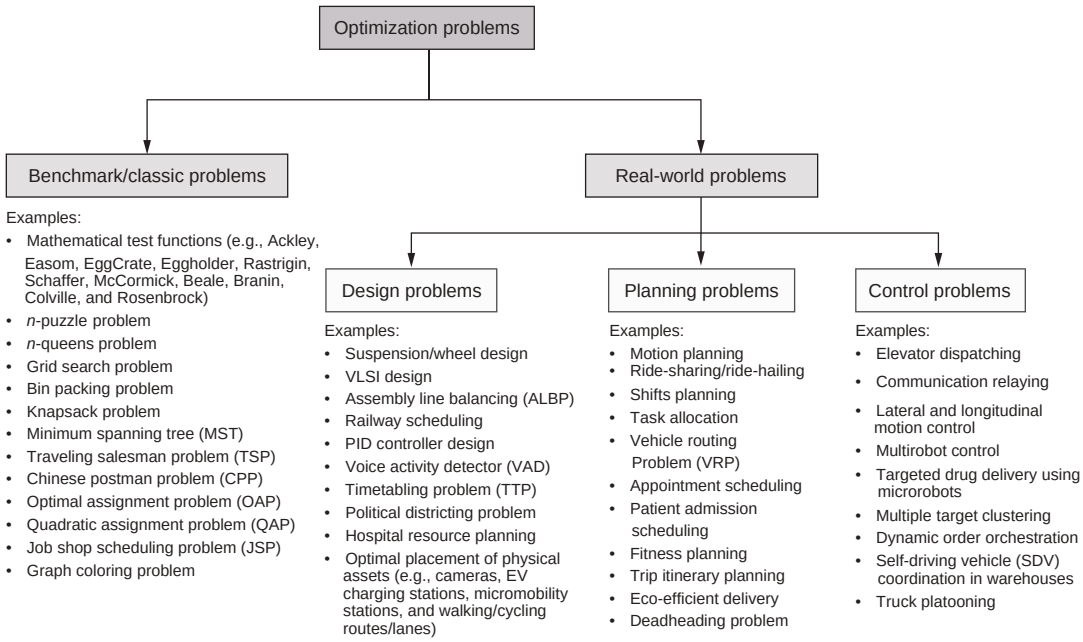


Figure 1.2 Examples of classic and real-world optimization problems

Real-world design problems, or strategic functions, can be used in situations when time is not as important as the quality of the solution and users are willing to wait (sometimes even a few days) to get optimal solutions. Planning problems, or tactical functions, need to be solved in a time span from a few seconds to a few minutes. Control problems, or operational functions, need to be solved repetitively and quickly, in a time span from a few milliseconds to a few seconds. To find a solution in such a short period of time, optimality is usually traded in for speed gains. In the next chapter, we'll discuss these problem types more thoroughly.

I highly recommend that you first perform the hand iterations for the examples following each algorithm and then try to recreate the Python implementations yourself. Feel free to play around with the parameters and problem scale in the code; one of the advantages of running optimization algorithms through software is the ability to tune for optimality.

1.3 Basic ingredients of optimization problems

Optimization refers to the practice of finding the "best" solutions to a given problem, where "best" usually means satisfactory or acceptable, possibly subject to a given set of constraints. The solutions can be classified into *feasible*, *optimal*, and *near-optimal* solutions:

- *Feasible solutions* are solutions that satisfy all the given constraints.
- *Optimal solutions* are both feasible and provide the best objective function value.
- *Near-optimal solutions* are feasible solutions that provide a superior objective function value but are not necessarily the best.

Assuming we have a minimization problem, where the goal is to find the values of a decision variable that minimize a certain objective function, a search space may combine multiple global minima, strong local minima, and weak local minima, as illustrated in figure 1.3:

- A *global optimum* (or a *global minimum* in the case of minimization problems) is the best of a set of candidate solutions (i.e., the lowest point of the entire feasible search space). Mathematically, if $f(x)$ is the objective function, a point x* is the global minimum if, for all x in the domain of f, $f(x^*) \leq f(\mathrm{x})$.
- A *strong local minimum* is a point where the function's value is less than (or equal to) the values of the function in a neighborhood around that point but is higher than the global minimum. Mathematically, a point x* is a strong local minimum if there is a neighborhood N of x^* such that $f(x^*) < f(\mathrm{x})$ for all x in N with $x \neq x^*$.
- A *weak local minimum* is a point where the function's value is less than or equal to the function's values at neighboring points, but there are sequences of points converging to this point for which the function's values strictly decrease. Mathematically, a point x^* is a weak local minimum if there is a neighborhood N of x^* such that $f(x^*) \leq f(x)$ for all x in N.

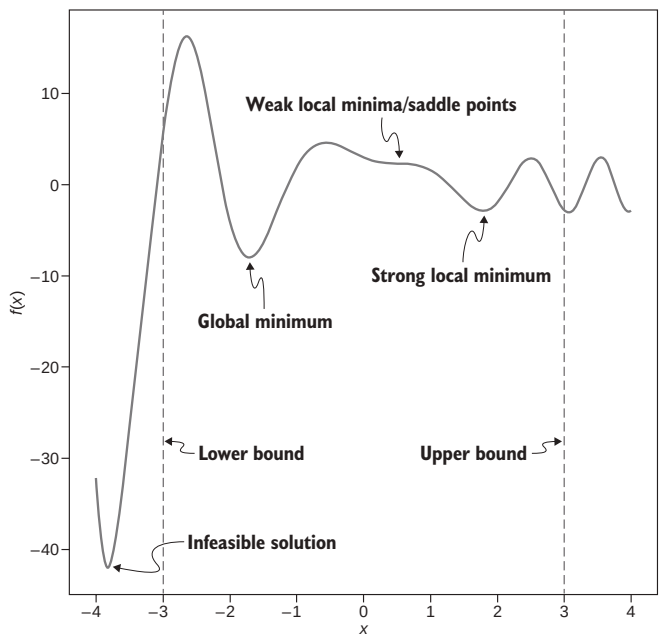


Figure 1.3 Feasible solutions fall within the constraints of the problem. A feasible search space may display a combination of global, strong local, and weak local minima.

These optimum seeking methods, also known as *optimization techniques*, are generally studied as a part of operations research (OR). OR, also referred to as *decision* or *management science*, is a field that originated at the beginning of World War II due to the urgent need to assign scarce resources in military operations. It is a branch of mathematics that applies advanced scientific analytical methods to decision-making and management problems to find the best or optimal solutions.

Optimization problems can generally be stated as follows. Find X which optimizes f, subject to a possible set of equality and inequality constraints:

$$g_i(X) = 0, i = 1, 2, \ldots, m$$
$$h_j(X) \leq 0, j = 1, 2, \ldots, p \quad \textbf{1.1}$$

where

- $X = (x_1, x_2, \ldots, x_n)^T$ is the vector representing the decision variables
- $f(X) = (f_1(X), f_2(X), \ldots, f_M(X))$ is the vector of objectives to be optimized
- $g_i(X)$ is a set of equality constraints
- $h_j(X)$ is a set of inequality constraints

The following subsections describe three main components of optimization problems: decision variables, objective functions, and constraints.

1.3.1 Decision variables

Decision variables represent a set of unknowns or variables that affect the objective function's value. These are the variables that define the possible solutions to an optimization problem. If X represents the unknowns, also referred to as the independent variables, then $f(X)$ quantifies the quality of the candidate solution or feasible solution.

For example, assume that an event organizer is planning a conference on search and optimization algorithms. The organizer plans to pay a for fixed costs (the venue rental, security, and guest speaking fees) and b for variable costs (pamphlets, lanyards, ID badges, and a catered lunch), which depend on the number of participants. Based on past conferences, the organizer predicts that demand for tickets will be as follows:

$$Q = 5000 - 20x \quad \textbf{1.2}$$

where x is the ticket price and Q is the expected number of tickets to be sold. Thus, the company expects the following scenarios:

- If the company charges nothing ($x = 0$), they will give away 5,000 tickets for free.
- If the ticket price is $x = \$250$, the company will get no attendees, and the expected number of tickets will be 0.
- If the ticket price is $x < \$250$, the company will sell some number of tickets $0 \leq Q \leq 5{,}000$.

The profit $f(x)$ that the event organizer can expect to earn can be calculated as follows:

$$Profit = Revenue - Costs \quad \textbf{1.3}$$

where $Revenue = Qx$ and $Costs = a + Qb$. Altogether, the profit (or objective) function looks like this:

$$\begin{aligned} f(x) = Revenue - Costs &= Qx - (a + Qb) \\ &= -20x^2 + (5000 + 20b)x - 5000b - a \end{aligned} \quad \textbf{1.4}$$

In this problem, the predefined parameters include fixed costs, a, and variable costs, b. There is a single decision variable, x, which is the price of the ticket where $x_{LB} \leq x \leq x_{UB}$. The ticket price's lower bound x_{LB} and upper bound x_{UB} are considered boundary constraints. Solving this optimization problem focuses on finding the best value of x that maximizes the profit $f(x)$.

1.3.2 Objective functions

An objective function $f(x)$, also known as the criterion, merit function, utility function, cost function, stands for the quantity to be optimized. Without a loss of generality, optimization can be interpreted as the minimization of a value, since the maximization of a primal function $f(x)$ can be just the minimization of a dual problem generated after applying mathematical operations on $f(x)$. This means that if the primal function is a minimization problem, then the dual problem is a maximization problem (and vice versa). According to this duality aspect of optimization problems, a solution x^*, which is the minimum for the primal minimization problem, is also, at the same time, the maximum for the dual maximization problem, as illustrated in figure 1.4.

Moreover, simple mathematical operations like addition, subtraction, multiplication, and division do not change the value of the optimal point. For example, multiplying or dividing $f(x)$ by a positive constant or adding or subtracting a positive constant to or from $f(x)$ does not change the optimal value of the decision variable, as illustrated in figure 1.4.

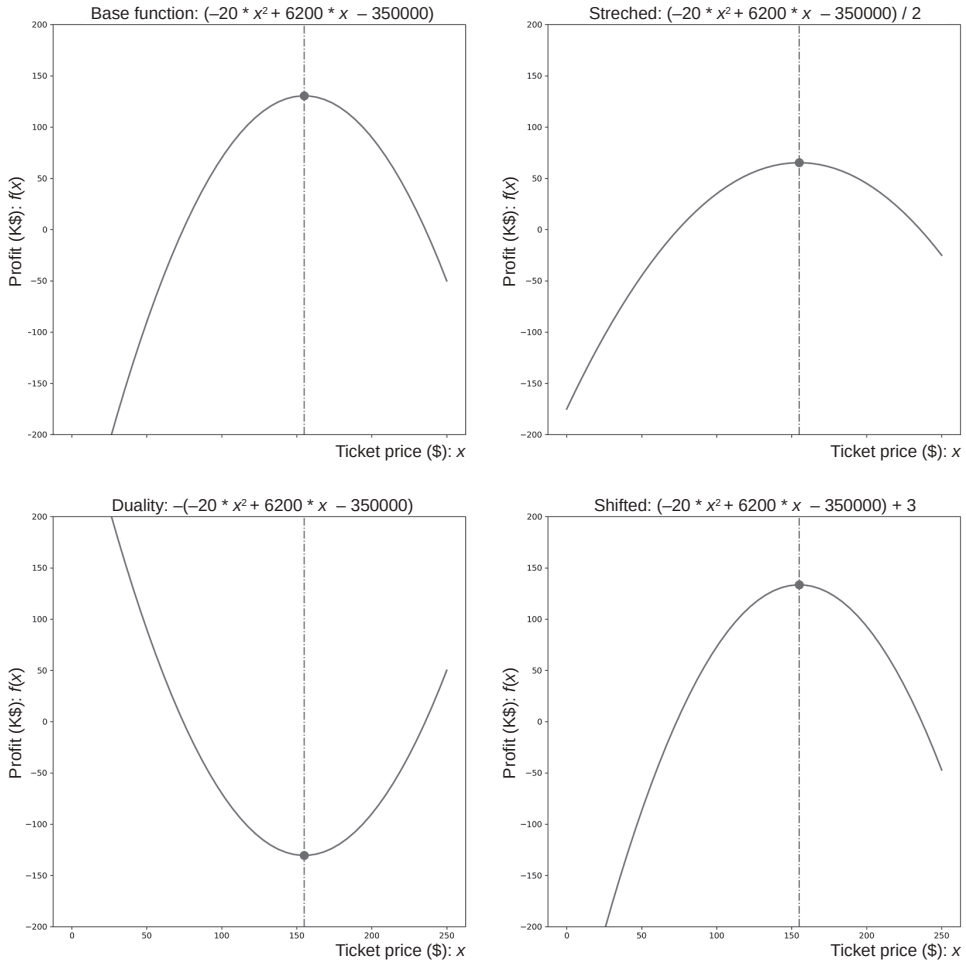


Figure 1.4 Duality principle and mathematical operations on an optimization problem

In the earlier ticket pricing problem, assume that $a = 50{,}000$, $b = 60$, $x_{LB} = 0$, and $x_{UB} = 250$. Using these values, we have a profit function: $f(x) = -20x^2 + 6{,}200x - 350{,}000$. Following a derivative-based approach, we can simply derive the function to find its maximum: $df/dx = -40x + 6{,}200 = 0$ or $40x = 6{,}200$. Thus, the optimal ticket price is \$155, which yields a net profit of \$130,500, as shown in figure 1.5.

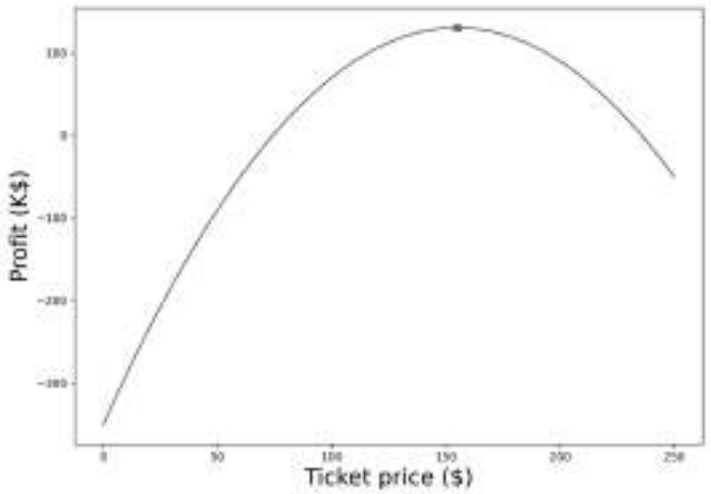


Figure 1.5 Ticket pricing problem—the optimal pricing that maximizes profit is \$155 per ticket.

In the ticket pricing problem, we have a single objective function to be optimized, which is the profit. In this case, the problem is called a *mono-objective optimization problem*. An optimization problem involving multiple objective functions is known as a *multi-objective optimization problem*. For example, assume that we want to design an electric vehicle (EV). This design problem's objective functions can be minimizing acceleration time and maximizing Environmental Protection Agency (EPA) driving range. The acceleration time is the time in seconds the EV takes to accelerate from 0 to 60 mph. The EPA driving range is the approximate number of miles that a vehicle can travel in combined city and highway driving (using a mix of 55% highway and 45% city driving) before needing to be recharged, according to the EPA's testing methodology. Decision variables can include the size of the wheels, the power of the electric motor, and the battery's capacity. A bigger battery is needed to extend the driving range of the EV, which adds extra weight, and therefore the acceleration time increases. In this example, the two objectives are in conflict, as we need to minimize acceleration time and maximize the EPA range, as shown in figure 1.6.

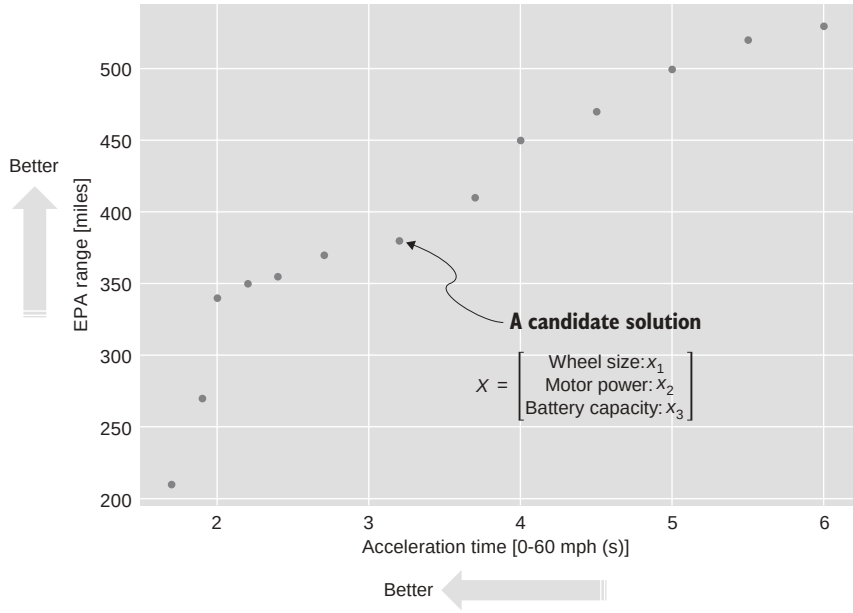


Figure 1.6 Electric vehicle design problem for maximizing EPA range and minimizing acceleration time

This multi-objective optimization problem can be handled using a preference-based multi-objective optimization procedure or by using a Pareto optimization approach. In the former approach, the duality principle is applied first to transform all the conflicting objectives for maximization (e.g., maximizing the EPA range and the inverse of the acceleration time) or for minimization (e.g., minimizing the acceleration time and the inverse of the EPA range). Then we combine these multiple objectives into an overall objective function by using a relative preference vector or a weighting scheme to scalarize the multiple objectives. For example, you may give more weight to EPA range over acceleration time. However, finding this preference vector or the weights is subjective, and sometimes it's not straightforward. The Pareto optimization approach relies on finding multiple trade-off optimal solutions and choosing one using higher-level information. This procedure tries to find the best trade-off by reducing the number of alternatives to an optimal set of nondominated solutions known as the Pareto frontier, which can be used to take strategic decisions in multi-objective space. Multi-objective optimization is discussed in chapter 8.

Constraint-satisfaction problems (CSPs) do not define an explicit objective function. Instead, the goal is to find a solution that satisfies a given set of constraints. The *n*-queen problem is an example of a CSP. In this problem, the aim is to put n queens on an n x n board with no two queens on the same row, column, or diagonal. The 4 x 4 queen CSP problem has two optimal solutions. Neither of these two optimal solutions is inherently or objectively better than the other. The only requirement of the problem is to satisfy the given constraints.

1.3.3 Constraints

Constrained optimization problems have a set of equality and/or inequality constraints $g_i(X)$, $l_j(X)$ that restrict the values assigned to the decision variables. In addition, most problems have a set of boundary constraints, which define the domain of values for each variable. Furthermore, constraints can be hard (must be satisfied) or soft (desirable to satisfy). Consider the following examples from a school timetabling problem:

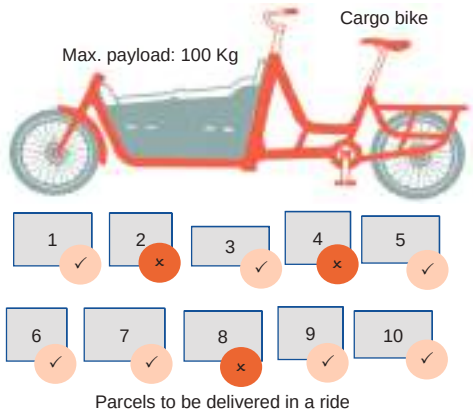
- Not having multiple lectures in the same room at the same time is a *hard constraint.*
- Not having a teacher give multiple lectures at the same time is also a *hard constraint.*
- Guaranteeing a minimum of three teaching days for every teacher may be a *soft constraint.*
- Locating back-to-back lectures in nearby rooms may be a *soft constraint.*
- Avoiding scheduling very early or very late lectures may also be a *soft constraint.*

As another example of hard and soft constraints, navigation apps such as Google Maps, Apple Maps, Waze, or HERE WeGo may allow users to set preferences for routing:

- Avoiding ferries, toll roads, and highways would be *hard constraints.*
- Avoiding busy intersections, highways during rush hour, or school zones during drop-off and pick-up times might be *soft constraints.*

Soft constraints can be modeled by incorporating a reward/penalty function as part of the objective function. The function can reward solutions that satisfy the soft constraints and penalize those that do not.

As an example, assume that there are 10 parcels to be loaded in the cargo bike in figure 1.7.



Item	Weight (kg)	Profit ($)	Efficiency ($/kg)
1	14.6	14.54	1.00
2	20	15.26	0.76
3	8.5	5.8	0.68
4	10	12.12	1.21
5	13	8.2	0.63
6	9.6	7.4	0.77
7	4.9	10.3	2.10
8	16.5	13.5	0.82
9	8.77	6.6	0.75
10	7.8	20.9	2.68

Figure 1.7 The cargo bike loading problem is an example of a problem with a soft constraint. While the weight of the packages can exceed the bike's capacity, a penalty will be applied when the bike is overweight.

Each parcel has its own weight, profit, and efficiency value (profit per kg). The goal is to select the parcels to be loaded in such a way that the profit function f_1 is maximized and the weight function f_2 is minimized. This is a classic example of a combinatorial problem:

$$f_1 = \sum_{i=0}^{n} E_i \tag{1.5}$$

where n is the total number of packages and E_i is the efficiency of package i

$$f_2 = \left| \sum_{i=0}^{n} w_i - C \right|, 50 \text{ is added } iff \left[\sum_{i=0}^{n} w_i > C \right] \tag{1.6}$$

where w_i is the weight of package i and C is the maximum capacity of the bike. A penalty of 50 is added if and only if the total weight of the added parcels exceeds the maximum capacity.

Soft constraints can also be used to make the search algorithm more adaptive. For example, the severity of the penalty can be dynamically changed as the algorithm progresses, imposing less strict penalties at first to encourage exploration, but imposing more severe penalties near the end to generate a result largely bound by the constraint.

1.4 Well-structured problems vs. ill-structured problems

We can classify optimization problems based on their structure and the procedure that exists (or doesn't exist) for solving them. The following subsections introduce well-structured and ill-structured problems.

1.4.1 Well-structured problems

In "The Structure of Ill Structured Problems," Herbert Simon outlines six key characteristics of well-structured problems (WSPs) [1]. These include the presence of a clear criterion for testing proposed solutions, the existence of a problem space capable of representing the initial problem state and potential solutions, and the representation of attainable and considerable state changes within the problem space. Moreover, any knowledge acquired by the problem solver can be represented within these spaces, and if the problem involves interacting with the external world, the state changes reflect the laws governing the real world. Simon emphasizes that these conditions hold strongly, implying that the processes require feasible computation and that information necessary for problem-solving is effectively available without excessive search efforts.

Assume that we are planning a robotic pick-and-place task in an inspection system. In this scenario, the robot waits until receiving a signal from a presence sensor, which indicates the existence of a defective workpiece over the conveyer belt. The robot stops the conveyer belt, picks up the defective piece, and deposits it in a waste box. Then the robot reactivates the movement of the conveyer belt. After this operation, the robot returns to its initial position and the cycle repeats. As illustrated in figure 1.8, this problem has the following well-structured components:

- *Feasible states*—The position and speed of the robot arm and its orientation and status (open or closed and orientation) of its end-effector (gripper)
- *Operator (successor)*—Robot arm motion control command to move from one point to another following a certain singularity-free trajectory (positions or joint angles in space and motion speed) and end-effector control (orientation and open or closed)
- *Goal*—Pick and place a defective workpiece regardless of its orientation
- *Solution/path*—Optimal sequence through state space for the fastest pick-and-place operation
- *Stopping criteria*—Defective workpiece is picked from the conveyer belt and placed in the waste box, and the robot returns to its home position
- *Evaluation criteria*—Pick-and-place duration and/or the success rate of the pick-and-place process

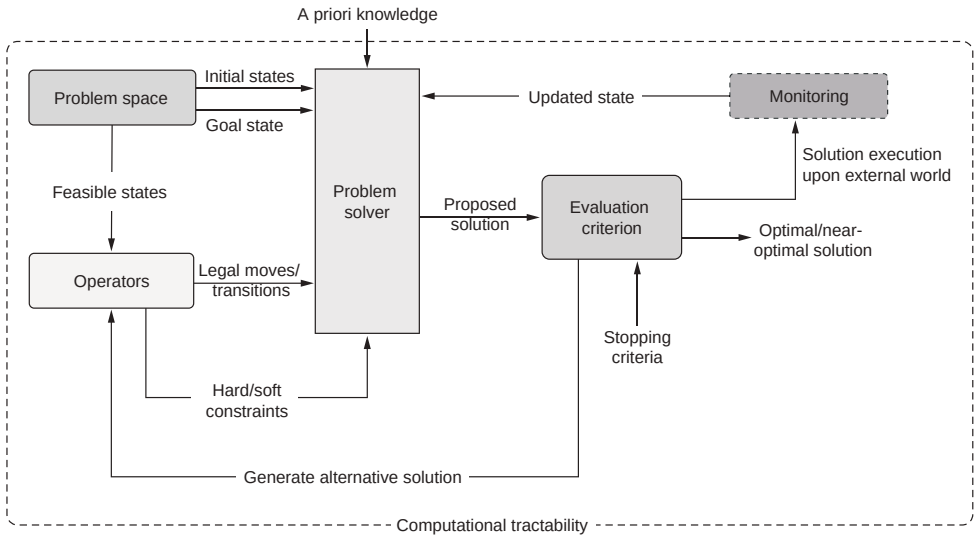


Figure 1.8 A WPS features a defined problem space, operators for allowable moves, clear evaluation criteria, and computational tractability.

As you can see, the work environment is highly structured, static, and fully observable. The problem can be mathematically modeled, and an optimal pick-and-place plan can be generated and executed with a high level of certainty. This pick-and-place problem can be considered a WSP.

1.4.2 Ill-structured problems

Ill-structured problems (ISPs) are complex discrete or continuous problems without algorithmic solutions or general problem solvers. ISPs are characterized by one or more of these characteristics:

- A problem space with different views of the problems, unclear goals, multimodality, and a dynamic nature
- A lack of exact mathematical models or a lack of well-proven algorithmic solutions
- Solutions that are contradictory, consequences that are difficult to predict, and risk that is difficult or impossible to calculate, resulting in a lack of clear evaluation criteria
- Considerable data imperfection in terms of uncertainty, partial observability, vagueness, incomplete information, ambiguity, or unpredictability that makes monitoring the execution of the solutions difficult and sometimes impossible
- Computational intractability

Assume that we need to find the optimal dispatching of four elevators to serve users between 10 floors, as illustrated in figure 1.9. This is a classic example of a problem too large to solve using traditional means.

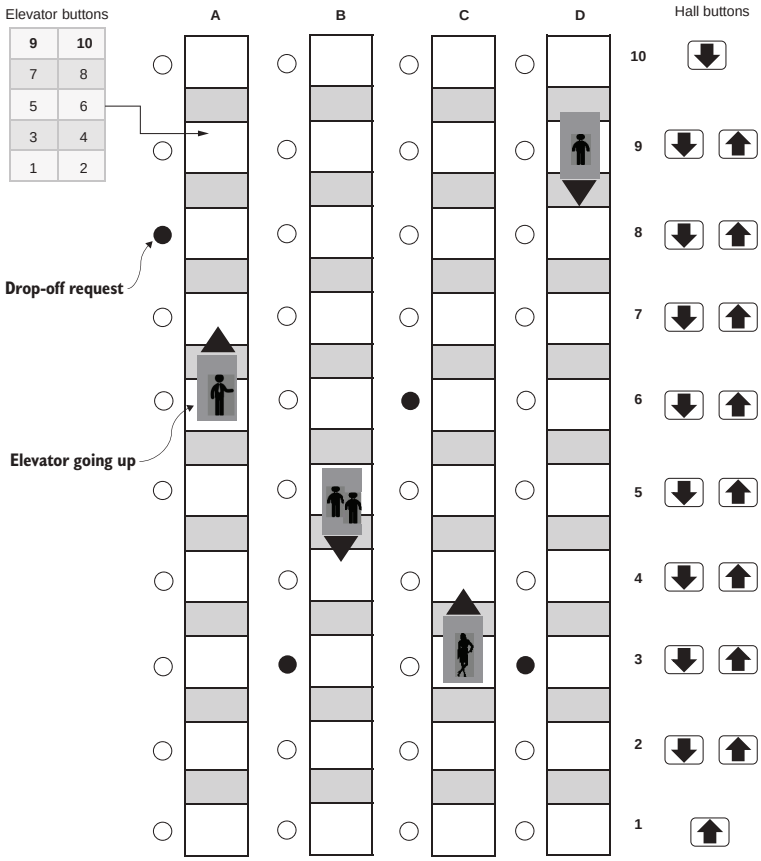


Figure 1.9 Elevator dispatching problem—with four elevator cars and 10 floors, this problem has ~10^{21} possible states.

The following objective functions can be considered in this optimal dispatching problem:

- Minimizing the average waiting time—how long the user waits before getting on an elevator
- Minimizing the average system time—how long the user waits before being dropped off at the destination floor
- Minimizing the percentage of users whose waiting time exceeds 60 seconds
- Ensuring fairness in serving all the users of the elevators

This optimal dispatching problem is an example of an ISP, as the problem space has a dynamic nature and partial observability; it is impossible to predict the user calls and destinations. Defining an optimum is almost impossible, as it can immediately change after a decision has been made based on the known situation (such as if a new request comes in for a move in the opposite direction). Moreover, the search space is huge due to the extremely high number of possible states, taking into consideration different elevator positions, elevator buttons, and hall call buttons:

- *Elevator position*—Each elevator can be on one of 10 floors. Therefore, for each elevator, there are 10 different possible states. Since there are four elevators, the number of combinations for elevator positions is 10^4.
- *Elevator buttons*—Each elevator has 10 buttons that can be either on (pressed) or off (not pressed). Therefore, for one elevator, there are 2^{10} different possible states. Since there are four elevators, the number of combinations for elevator buttons is 2^{40}.
- *Hall call buttons*—There are 18 hall call buttons (up and down buttons at each floor, except the first and the last floor) that can be either on or off. Therefore, the number of combinations for hall call buttons is 2^{18}.

Assuming that every combination of button presses is valid (i.e., ignoring the physical or logical limitations of an elevator system, such as not allowing both the up and down hall call buttons on the same floor to be pressed at the same time), the total number of states can be calculated as follows: number of possible states = 10^4 (elevator positions) * 2^{40} (elevator buttons) * 2^{18} (hall call buttons) = 2.88 x 10^{21} different states. The total number of states is more than the number of stars in the universe!

1.4.3 WSP, but ISP in practice

The traveling salesman problem (TSP) is an example of a problem that may be well-structured in principle, but in practice becomes ill-structured. This is because of the impractical amount of computational power required to solve the problem in real time.

Assume that a traveling salesman is assigned to make sales calls to a list of n cities. The salesman would like to visit all these cities in the minimum amount of time, as salespeople are generally paid by commission rather than hourly. Furthermore, the tour of the cities may be asymmetric; the time it takes to go from city A to city B may not be the same as the reverse due to infrastructure, traffic patterns, and one-way streets. For example, with 13 cities to visit, the problem may initially seem trivial. However, upon closer examination, the search space for this TSP results in 13! = 6,227,020,800 different possible routes to be examined in the case of using naive algorithms! Fortunately, dynamic programming algorithms enable reduced complexity, as we will see in the next chapter.

This book largely focuses on ISPs, and on WSPs that are ISPs in practice, for a few reasons:

- WSPs tend to have well-known solving algorithms that often provide trivial, step-by-step procedures. As such, very efficient and well-known solutions often exist for these kinds of problems. Moreover, several WSPs can be solved using derivative-based generic solvers.
- The amount of computational power needed to solve WSPs is often negligible, or very manageable at worst. Especially with the continued improvement of consumer-grade computers, not to mention the vast resources available through cloud computing and distributed processing, we often do not have to settle for near-optimal WSP solutions resulting from computational bottlenecks.
- Most problems in the real world are ISPs, as the problem scope, state, and environment are dynamic and sometimes partially observable with certain degrees of uncertainties. Solutions or algorithms for ISPs, therefore, have much more applicability to real-world scenarios, and there is a greater incentive to find solutions to these problems.

Most of the algorithms explored in this book are derivative-free and stochastic; they use randomness in their parameters and decision processes. These algorithms are often well suited to solving ISPs, as the randomness of their initial states and operators allows the algorithms to escape local minima and find optimal or near-optimal solutions. In contrast, deterministic algorithms use well-defined and procedural paths to reach solutions and generally are not well suited for ISPs, as they either cannot work in unknown search spaces or are unable to return solutions in a reasonable amount of time. Moreover, most of the algorithms covered in this book are black-box solvers that deal with the optimization problem as a black box. This black box provides, for certain decision variable values, the corresponding values of the objective functions and constraint functions. Importantly, this approach eliminates the need to consider various properties of the objective and constraint functions, such as nonlinearity, differentiability, nonconvexity, monotonicity, discontinuities, or even stochastic noise.

1.5 Search algorithms and the search dilemma

The goal of any optimization method is to assign values to decision variables so that the objective function is optimized. To achieve this, optimization algorithms search the solution space for candidate solutions. Constraints are simply limitations on specific regions in the search space. Thus, all optimization techniques are, in reality, just search methods, where the goal is to find feasible solutions to satisfy constraints and maximize (or minimize) the objective functions. We'll define "search" as the systematic examination of feasible states, starting from the initial state, and ending (hopefully) at the goal state. However, while we explore the feasible search space, we may find a few reasonably good neighboring solutions, and the question is whether we should exploit this region or keep exploring, looking for better solutions in other regions of the feasible search space.

Exploration (or *diversification*) is the process of investigating new regions in the feasible search space with the hope of finding other promising solutions. On the other hand, *exploitation* (or *intensification*) is the process of directing the search agent to focus on an attractive region of the search space where good solutions have already been found.

This exploration–exploitation dilemma is one of the most important problems in search and optimization, and in life as well. We apply exploration–exploitation tactics in our lives. When we move to a new city, we start by exploring different stores and restaurants and then focus on shortlisted options around us. During a midlife crisis, some middle-aged individuals feel bored in their daily routine and lifestyle without satisfactory accomplishments, and they tend to take explorative actions. The US immigration system tries to avoid exploiting specific segments of applicants (e.g., family, skilled workers, refugees, and asylees) and enables more diversity through a computer-generated lottery. In social insects like honeybees, foraging for food sources is performed by two different worker groups, foragers and scouts (5–25% of the foragers). Forager bees focus on a specific food source while scouts are novelty seekers who keep scouting around for rich nectar. In search and optimization, the exploration–exploitation dilemma represents the trade-off between exploring new unvisited states or solutions in the search space and exploiting the elite solutions found in a certain neighborhood in the search space (figure 1.10).

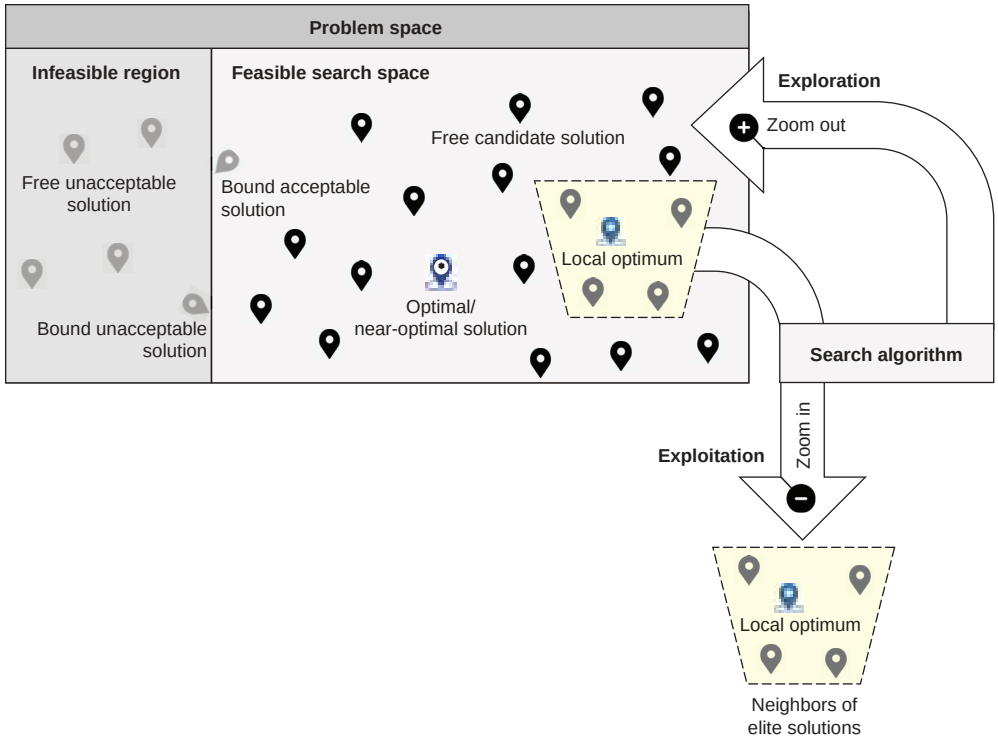


Figure 1.10 Search dilemma—there is always a trade-off between branching out to new areas of the search space or focusing on an area with known good or elite solutions.

Local search algorithms are exploitative algorithms that can be easily trapped in local optima if the search landscape is multimodal. On the other extreme, random search algorithms keep exploring the search space with a high chance of reaching global optima at the cost of an impractical search time. Generally speaking, explorative algorithms can find global optima at the cost of processing time, while exploitative algorithms risk getting stuck at local minima.

Summary

- Optimization is ubiquitous and pervasive in numerous areas of life, industry, and research.
- Decision variables, objective functions, and constraints are the main ingredients of optimization problems. Decision variables are the inputs that you have control over and that affect the objective function's value. An objective function is the function that needs to be optimized, either minimized or maximized. Constraints are the limitations or restrictions that the solution must satisfy.
- Optimization is a search process for finding the "best" solutions to a problem, providing the best objective function values, and possibly subject to a given set of hard (must be satisfied) and soft (desirable to satisfy) constraints.
- Ill-structured problems are complex discrete or continuous problems without exact mathematical models and/or algorithmic solutions or general problem solvers. They usually have dynamic and/or partially observable large search spaces that cannot be handled by classic optimization methods.
- In many real-life applications, quickly finding a near-optimal solution is better than spending a large amount of time searching for an optimal solution.
- Two key concepts you'll see frequently in future chapters are the exploration (or diversification) and exploitation (or intensification) search dilemmas. Achieving a trade-off between exploration and exploitation will allow the algorithm to find optimal or near-optimal solutions without getting trapped in local optima in an attractive region of the search space and without spending a large amount of time.