

1. Why Are Some Problems Difficult to Solve?

Our problems are man-made;
therefore they may be solved by man.

John F. Kennedy, speech, June 10, 1963

At the risk of starting with a tautology, real-world problems are difficult to solve, and they are difficult for several reasons:

- The number of possible solutions in the *search space* is so large as to forbid an exhaustive search for the best answer.
- The problem is so complicated that just to facilitate any answer at all, we have to use such simplified models of the problem that any result is essentially useless.
- The *evaluation function* that describes the quality of any proposed solution is noisy or varies with time, thereby requiring not just a single solution but an entire series of solutions.
- The possible solutions are so heavily constrained that constructing even one feasible answer is difficult, let alone searching for an optimum solution.
- The person solving the problem is inadequately prepared or imagines some psychological barrier that prevents them from discovering a solution.

Naturally, this list could be extended to include many other possible obstacles. For example, we could include noise associated with our observations and measurements, uncertainty about given information, and the difficulties posed by problems that have multiple and possibly conflicting objectives (which may require a set of solutions rather than a single solution). The above list, however, is sufficient for now. Each of these are problems in their own right. To solve a problem, we have to understand the problem, so let's discuss each of these issues in turn and identify their inherent details.

1.1 The size of the search space

One of the elementary problems in logic is the Boolean satisfiability problem (SAT). The task is to make a compound statement of Boolean variables evaluate to TRUE. For example, consider the following problem of 100 variables given in conjunctive normal form:

$$F(\mathbf{x}) = (x_{17} \vee \bar{x}_{37} \vee x_{73}) \wedge (\bar{x}_{11} \vee \bar{x}_{56}) \wedge \dots \wedge (x_2 \vee x_{43} \vee \bar{x}_{77} \vee \bar{x}_{89} \vee \bar{x}_{97}).$$

The challenge is to find the truth assignment for each variable x_i , for all $i = 1, \dots, 100$ such that $F(\mathbf{x}) = \text{TRUE}$. We can use 1 and 0 as synonyms for TRUE and FALSE, and note that \bar{x}_i here is the negation of x_i (i.e., if x_i were TRUE or 1, then \bar{x}_i would be FALSE or 0).

Regardless of the problem being posed, it's always useful to consider the space of possible solutions. Here, any binary string of length 100 constitutes a potential solution to the problem. We have two choices for each variable, and taken over 100 variables, this generates 2^{100} possibilities. Thus the size of the search space \mathcal{S} is $|\mathcal{S}| = 2^{100} \approx 10^{30}$. This is a huge number! Trying out all of these alternatives is out of the question. If we had a computer that could test 1000 strings per second and could have started using this computer at the beginning of time itself, 15 billion years ago right at the Big Bang, we'd have examined fewer than one percent of all the possibilities by now!

What's more, the choice of which evaluation function to use isn't very clear. What we'd like is for the evaluation function to give us some guidance on the quality of the proposed solution. Solutions that are closer to the right answer should yield better evaluations than those that are farther away. But here, all we have to operate on is $F(\mathbf{x})$ which can either evaluate to TRUE or FALSE. If we try out a string \mathbf{x} and $F(\mathbf{x})$ returns TRUE then we're done: that's the answer. But what if $F(\mathbf{x})$ returns FALSE? Then what? Furthermore, almost every possible string of 0s and 1s that we could try would likely evaluate to FALSE, so how could we distinguish between "better" and "worse" potential solutions? If we were using an enumerative search we wouldn't care because we'd simply proceed through each possibility until we found something interesting. But if we want the evaluation function to help us find the best solutions faster than enumeration, we need more than just "right" or "wrong." The way we could accomplish that for the SAT problem isn't clear immediately.

Some problems seem easier than SAT problems because they suggest a possible evaluation function naturally. Even so, the size of the search space can still be enormous. For example, consider a traveling salesman problem (TSP). Conceptually, it's very simple: the traveling salesman must visit every city in his territory exactly once and then return home covering the shortest distance. Some closely related problems require slightly different criteria, such as finding a tour of the cities that yields minimum traveling time, or minimum fuel cost, or a number of other possibilities, but the idea is the same. Given the cost of traveling between each pair of cities, how should the salesman plan his itinerary to achieve a minimum cost tour?

Figure 1.1 illustrates a simple symmetric 20-city TSP where the distance between each pair of cities i and j is the same in either direction. That is, $\text{dist}(i, j) = \text{dist}(j, i)$. The actual distances aren't marked on the figure, but we could assume this is the case. Alternatively, we could face an asymmetric TSP where $\text{dist}(i, j) \neq \text{dist}(j, i)$ for some i and j . These two classes of TSP present different obstacles for finding minimum cost paths.

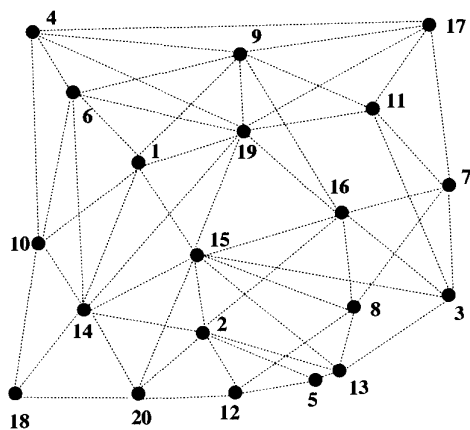


Fig. 1.1. A sample TSP. Textbook TSPs usually allow paths from every city to every other cities, but real-world problems don't always afford such opportunities.

So then, what is the search space for the TSP? One possibility would be to view it as the set of permutations of n cities. Any permutation of n cities yields an ordered list that defines the sequence of cities to be visited, starting at the salesman's home base, and continuing to the last location before returning home. The optimum solution is a permutation that yields the minimum cost tour. Note that tours such as:

2 - ... - 6 - 15 - 3 - 11 - 19 - 17,
 15 - 3 - 11 - 19 - 17 - 2 - ... - 6,
 3 - 11 - 19 - 17 - 2 - ... - 6 - 15, etc.

are identical because the circuit that each one generates is exactly the same regardless of the starting city, and there are n tours like that for any n -city TSP. It's easy to see then that every tour can be represented in $2n$ different ways (for a symmetrical TSP). And since there are $n!$ ways to permute n numbers, the size of the search space is then $|\mathcal{S}| = n!/(2n) = (n-1)!/2$.

Again, this is a huge number! For any $n > 6$, the number of possible solutions to the TSP with n cities is larger than the number of possible solutions to the SAT problem with n variables. Furthermore, the difference between the sizes of these two search spaces increases very quickly with increasing n . For $n = 6$, there are $5!/2 = 60$ different solutions to the TSP and $2^6 = 64$ solutions to a SAT. But for $n = 7$ these numbers are 360 and 128, respectively.

To see the maddening rate of growth of $(n-1)!/2$, consider the following numbers:

- A 10-city TSP has about 181,000 possible solutions.

- A 20-city TSP has about 10,000,000,000,000 possible solutions.
- A 50-city TSP has about 100,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000 possible solutions.

There are only 1,000,000,000,000,000,000,000,000 liters of water on the planet, so a 50-city TSP has an unimaginably large search space. Literally, it's so large that as humans, we simply can't conceive of sets with this many elements.

Even though the TSP has an incredibly large search space, the evaluation function that we might use to assess the quality of any particular tour of cities is much more straightforward than what we saw for the SAT. Here, we can refer to a table that would indicate all of the distances between each pair of cities, and after n addition operations we could calculate the distance of any candidate tour and use this to evaluate its merit. For example, the cost of the tour

$$15 - 3 - 11 - 19 - 17 - 2 - \dots - 6$$

is

$$\text{cost} = \text{dist}(15, 3) + \text{dist}(3, 11) + \text{dist}(11, 19) + \dots + \text{dist}(6, 15).$$

We might hope that this, more natural, evaluation function would give us an edge in finding useful solutions to the TSP despite the size of the search space.

Let's consider a third example — a particular nonlinear programming problem (NLP). It's a difficult problem that has been studied in the scientific literature and no traditional optimization method has given a satisfactory result. The problem [254] is to maximize the function:¹

$$G2(\mathbf{x}) = \left| \frac{\sum_{i=1}^n \cos^4(x_i) - 2 \prod_{i=1}^n \cos^2(x_i)}{\sqrt{\sum_{i=1}^n i x_i^2}} \right|,$$

subject to

$$\prod_{i=1}^n x_i \geq 0.75, \quad \sum_{i=1}^n x_i \leq 7.5n, \quad \text{and bounds } 0 \leq x_i \leq 10 \text{ for } 1 \leq i \leq n.$$

The function $G2$ is nonlinear and its global maximum is unknown, but it lies somewhere near the origin. The optimization problem poses one nonlinear constraint and one linear constraint (the latter one is inactive near the origin).

What's the size of the search space now? In a sense, it depends on the dimensionality of the problem — the number of variables. When treated as a purely mathematical problem, with n variables, each dimension can contain an infinity of possible values, so we have an infinitely large space — maybe even several degrees of infinity. But on a computer, everything is digital and finite, so if we were going to implement some sort of algorithm to find the optimum of $G2$, we'd have to consider the available computing precision. If our precision guaranteed six decimal places, each variable could then take on

¹We're preserving the notation $G2$ used for this function in [319]. It's the second function in an 11-function testbed for nonlinear programming problems.

10,000,000 different values. Thus, the size of the search space would be $|\mathcal{S}| = 10,000,000^n = 10^{7n}$. That number is much much larger than the number of solutions for the TSP. Even for $n = 50$ there are 10^{350} solutions to the NLP with only six decimal places of precision. Most computers could give twice this precision.

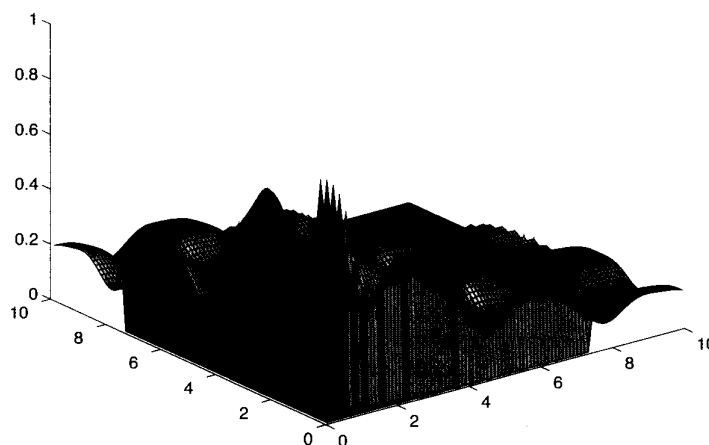


Fig. 1.2. The graph of function $G2$ for $n = 2$. Infeasible solutions were assigned a value of zero.

What about the evaluation function? How can we measure the quality of alternative solutions? One idea would be to use the function $G2$ itself as an evaluation function. Those solutions that yield higher values of $G2$ would be judged as being better than those that yield lower values. But there are some difficulties in this regard because, as illustrated in figure 1.2, there are infeasible regions, and all of the infeasible points were assigned a value of zero. The interesting boundary between feasible and infeasible regions is defined by the equation $\prod_{i=1}^n x_i = 0.75$, and the optimal solution lies on (or close to) this boundary. Searching boundaries of a feasible part of the search space isn't easy. It requires specialized operators that are tailored for just this purpose, on just this problem. This presents an additional level of difficulty that we didn't see in the SAT or TSP (although for a TSP that didn't allow transit between all possible pairs of cities, some permutations would also be infeasible). Even without this additional wrinkle, it's evident that problems that seem simple at first can offer significant challenges simply because of the number of alternative solutions. The means for devising ways to assess these solutions isn't always clear.

1.2 Modeling the problem

Every time we solve a problem we must realize that we are in reality only finding the solution to a *model* of the problem. All models are a simplification of the real world, otherwise they would be as complex and unwieldy as the natural setting itself. The process of problem solving consists of two separate general steps: (1) creating a model of the problem, and (2) using that model to generate a solution:

Problem \Rightarrow Model \Rightarrow Solution.

The “solution” is only a solution in terms of the model. If our model has a high degree of fidelity, we can have more confidence that our solution will be meaningful. In contrast, if the model has too many unfulfilled assumptions and rough approximations, the solution may be meaningless, or worse.

The SAT, TSP, and NLP are three canonical forms of models that can be applied in many different settings. For example, suppose a factory produces cars in various colors where there are n colors altogether. The task is to find an optimum production schedule that will minimize the total cost of painting the cars. Note, however, that each machine involved in the production line has to be switched from one color to another between jobs, and the cost of such a switch (called a *changeover*) depends on the two colors involved and their order. The cost of switching from *yellow* to *black* might be 30 units. This might be measured in dollars, minutes, or by some other reasonable standard. The cost of switching back from *black* to *yellow* might be 80 units.² The cost of going from *yellow* to *green* might be 35 units, and so forth. To minimize costs, we have to find a sequence of jobs that meets all of the production requirements for the number of cars of each color, in a timely fashion, while still keeping the costs of the operation as low as possible. This might be viewed in terms of a TSP, where each city is now a job that corresponds to painting a certain car with a particular color and the distance between cities corresponds to the cost of changing jobs. Here, the TSP would be asymmetric.

Consider the following scenario that illustrates how simplifications are inherent to modeling. Suppose a company has n warehouses that store paper supplies in reams. These supplies are to be delivered to k distribution centers. The warehouses and distribution centers can be viewed as *sources* and *destinations*, respectively. Every possible delivery route between a warehouse i and a distribution center j has a measurable transportation cost, which is determined by a function f_{ij} . The shape of this function depends on a variety of factors including the distance between the warehouse and the distribution center, the quality of the road, the traffic density, the number of required stops, the average speed limit, and so forth. For example, the transportation cost function between warehouse 2 and distribution center 3 might be defined as:

²Note the asymmetry here: the costs of switching between two colors need not be the same. Switching from *yellow* to *black* isn't usually as expensive as the obverse switch.

$$f_{23}(x) = \begin{cases} 0 & \text{if } x = 0 \\ 4 + 3.33x & \text{if } 0 < x \leq 3 \\ 19.5 & \text{if } 3 < x \leq 6 \\ 0.5 + 10\sqrt{x} & \text{if } 6 < x, \end{cases}$$

where x refers to the quantity of supplies transported from 2 to 3 (figure 1.3 displays the graph of the function f_{23}).³

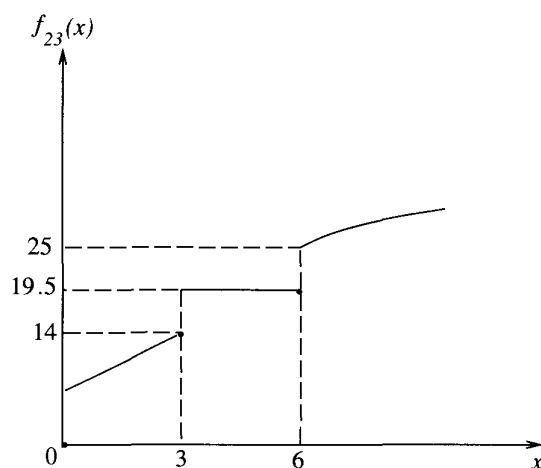


Fig. 1.3. An example transportation cost function for a given source and a given destination

Even though this function looks a bit unconventional, the justification for using it in the model of our transportation problem might be quite straightforward. If there's no delivery, the cost is zero, of course. If up to three reams of paper are transported we can use a special shipping container. This incurs an overhead cost of four units for the container and an additional cost of 3.33 units per ream. So the cost for this case increases linearly. However, if we transport more than three reams but not more than six, we can use a special wire mesh box. In this case, the cost is a flat 19.5 units, regardless of the number of reams being shipped. Finally, if we are shipping more than six reams we have to use a large reinforced crate, with a total transportation cost that depends on the number of reams being shipped and grows as a square root of that quantity plus a small overhead of 0.5 units.

Given these preliminaries, we can construct a model of the problem:

$$\text{minimize } \sum_{i=1}^n \sum_{j=1}^k f_{ij}(x_{ij}),$$

subject to

³Note that discontinuities are quite typical for most *real* transportation cost functions.

$$\begin{aligned} \sum_{j=1}^k x_{ij} &\leq \text{sour}(i), \text{ for } i = 1, 2, \dots, n, \\ \sum_{i=1}^n x_{ij} &\geq \text{dest}(j), \text{ for } j = 1, 2, \dots, k, \\ x_{ij} &\geq 0, \text{ for } i = 1, 2, \dots, n \text{ and } j = 1, 2, \dots, k, \end{aligned}$$

where *sour* is the source and *dest* is the destination. The constraints of the problem define a feasible solution: no transport from any warehouse exceeds the number of available reams in that warehouse, and the total transport to any distribution center must satisfy its demand (i.e., the total transport is at least equal to the number of ordered reams).

It might just be that this cost function describes the real-world situation faithfully (i.e., exactly), neglecting the costs for the other aspects we mentioned earlier such as the traffic density between the source and destination and so forth. And we might be able to construct similar exact functions that describe the costs of transporting the reams of paper from every warehouse to every distribution center. Still, such a precise model of the problem might be of only limited utility because these functions are too complex for many traditional optimization algorithms. For starters, they are discontinuous, and discontinuities present severe problems. The results that we would obtain after using some gradient-based methods on these functions would likely be quite poor [320]. Thus, we cannot derive a solution based on this model, so the model — as perfect as it is — is useless for deciding what to do!

What options do we have? There are at least two ways to proceed:

1. We can try to simplify the model so that traditional optimizers might return better answers.
2. We can keep the model as it is, and use a nontraditional approach to find a near-optimum solution.

The first idea is quite tempting. For example, we can *approximate* the function f_{23} as follows:

$$f'_{23}(x) = 2.66x + 8.25,$$

where x denotes the number of reams transported from 2 to 3 (figure 1.4 displays the graph of the approximate function f'_{23} together with the original f_{23}).

In this case, we simplified the transportation cost function f_{23} , and we can perform similar simplifications for the other functions. Note that if all of the f'_{ij} s were linear, we'd obtain a linear model of the problem that can be solved precisely by a linear programming method. But note that this exact solution would then be a solution for the simplified model and not for the real problem!

The second option is to leave the precise model as it is — with all of its discontinuities — and use a nontraditional method (e.g., simulated annealing or an evolutionary algorithm) to find a near-optimal solution. A large volume of experimental evidence shows that this latter approach can often be used to practical advantage.

Let's rephrase this discussion. There are two possible approaches. The first uses an approximate model, $Model_a$, of a problem, and then finds the precise solution $Solution_p(Model_a)$ for this approximate model:

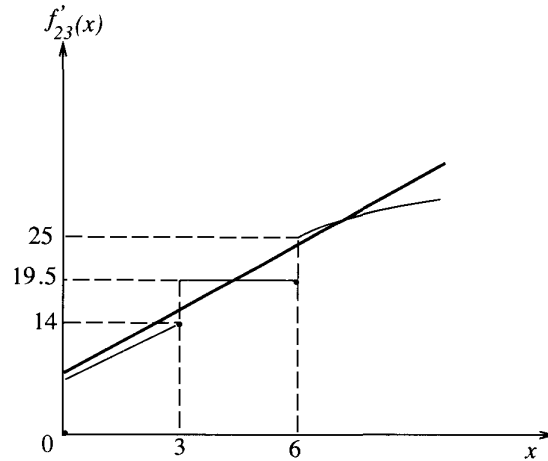


Fig. 1.4. An approximation of the transportation cost function (*bold line*) for a given source and a given destination

$$\text{Problem} \Rightarrow \text{Model}_a \Rightarrow \text{Solution}_p(\text{Model}_a).$$

The second approach uses a precise model, Model_p , of the problem, and then finds an approximate solution $\text{Solution}_a(\text{Model}_p)$ for this precise model:

$$\text{Problem} \Rightarrow \text{Model}_p \Rightarrow \text{Solution}_a(\text{Model}_p).$$

Of these two approaches, the latter one is often superior; i.e., $\text{Solution}_a(\text{Model}_p)$ is better than $\text{Solution}_p(\text{Model}_a)$ as a solution of the original problem.

But either way, this is the second source of difficulties we face in problem solving: it's difficult to obtain a precise solution to a problem because we either have to approximate a model or approximate the solution.

1.3 Change over time

As if the above concerns weren't enough, real-world problems often present another set of difficulties: they change. They change before you model them, they change while you are deriving a solution, and they change after you execute your solution. Let's look at some of the sources of trouble.

Think back to the traveling salesman problem from figure 1.1. Suppose you are the salesman and you are leaving your home at city 1 on your way to city 6. You've driven this route before and you know the distance between the two towns is, say, 20 miles. But do you know how long it will take you to travel between the cities? Not precisely. You might figure that typically you can average 30 miles per hour and so on average this trip will take 40 minutes. But

what is the probability that 40 minutes will be the exact travel time today? That's very unlikely.

The travel time depends on many factors. For example, you might be lucky and make all of the green traffic lights along the way. Or you might be unlucky and not only hit the red lights but also get stuck behind a slow-moving truck. Worse, you might get a flat tire, which would add a significant amount of time to your trip. All of these possibilities, and an unimaginable number of other outcomes including the weather, road conditions, traffic accidents, emergency vehicles requiring you to pull aside, a train on tracks crossing your path, and so forth, can be described under the heading of *noise* or *randomness*. You can't predict whether or not any of these events will happen before you leave. All you might be able to know, or estimate, is the likelihood of each of the anticipated events and the associated consequences to your travel time. But you must acknowledge that you'll never be able to account for every possibility.

Suppose you decided to simply calculate the expected travel time based on the probabilities of the known possible events and the effect they have on your travel time. Let's simplify things a little. Say there are only two possibilities for your trip: (1) everything goes fine and you can travel to city 6 in 40 minutes, or (2) you get stuck behind a slow vehicle and it takes you 60 minutes. Furthermore, let's say that these two events are equally likely. So the expected time for the trip is 50 minutes. But note that neither of the two possibilities above takes 50 minutes. If you use 50 minutes as the approximate time you ensure using a value that *will never happen in reality*. You can be 100 percent certain that your value will be wrong. It might be easy to imagine that if you used a series of these incorrect approximations to determine your travel time to each of the cities in your route, compounding your errors between each city as you go, that your final estimated time might be very different from any of the possible times that it would really take. Any decision you make based on this *average* time fails to take into account the *variability* of the time, and this can be much more important in effecting good decisions.

Random chance isn't the only source of change in real-world problems. Sometimes there are purely deterministic troubles as well. You know, for example, that travel at rush hour will be more time consuming in each city than travel at midnight. You might not know exactly how much more time you'll spend — that's a random event — but you know there is a bias that rush hour will stall your progress in traffic. That bias is a regular, predictable pattern, and you need to consider it or else your model might not correspond sufficiently to reality, and your solutions with the model won't be useful. In the worst case, a particular route between two cities might be available only during certain times of day, and not available otherwise (this often happens in cities where city planners limit your options for turning into side streets at rush hour in order to increase traffic flow). Acting as if the unavailable route were still an option would lead to an infeasible solution, and that means no solution at all.

It's also important to be sure that the model reflects current knowledge about the problem. It might be that road improvements or a new freeway system

between two of the cities on your list now allow for more rapid transit. If you fail to update your model to account for this change, you'll be deriving a solution to a problem that no longer exists.

The situation, however, is even more complex than this. The above vagaries might occur as a function of environmental changes or uncontrollable events, but none of them were conspiring against you. Unfortunately, in the real world, it's often the case that other people are trying to defeat your solution, and this requires you to continually update your model and anticipate other people's actions.

For example, suppose you are the owner of a major supermarket chain. You need to decide where to place a new store, so you calculate the cost of construction in each possible location, the demographics of the neighboring areas, the existing competition, etc., formulate an evaluation function, which would likely result in a nonlinear programming problem, and set out to decide where the best place for the store is. But there's more to the problem: as you are deciding where to put your new store, your competition is anticipating your choice of location, and they have their own new store to construct. They are actively trying to figure out how best to place their new store so as to minimize your success. If you only solve the problem of the best position for your store given the current conditions, you are treating the situation as if it were a one-player game. The real-world problem often changes while you are deriving a solution, and sometimes it changes in ways that are designed to make your life difficult.

1.4 Constraints

Unfortunately, things are often even worse than we've made it out so far because real-world problems don't offer you all of the possibilities that you might like to have. Almost all practical problems pose constraints, and if you violate the constraints you can't implement your solution. Think back to the NLP from section 1.1. There, we had a case where it wasn't enough to find the maximum of the function $G2$, we had to ensure that the solution we proposed was in the feasible region bounded by the product and summation constraints (see figure 1.2). Now at first you might think constraining problems like this would make life easier — after all, we have a smaller search space to worry about and therefore fewer possibilities to consider. That's true, but remember that to *search* for improved solutions we have to be able to move from one solution to the next. We need operators that will act on feasible solutions and hopefully in turn generate new feasible solutions that are an improvement over what we've already found. It's here where the geometry of the search space gets tricky.

For example, suppose we're faced with the problem of making a timetable for all of the classes at a college in one semester. Think about what this entails. First, we have to make a list of all the courses that will be offered. Next, we need a list of all the students assigned to each class, and let's not forget the

professor assigned to each class too. Third, we need a list of available classrooms, noting the size and other facilities that each offers (e.g., a white board, a video projector, laboratory equipment, and so forth). So then, what are we trying to accomplish? There are three hard constraints:

- Each class must be assigned to an available room that has enough seats for every assigned student and has the requisite facilities for the type of instruction (e.g., a chemistry lab must have beakers, Bunsen burners, the appropriate chemicals, safeguards, etc.).
- Students who are enrolled in more than one class can't have their classes held at the same time on the same day.
- Professors can't be assigned to teach courses that overlap in time.

We said those are the *hard constraints*. By that, we mean these are the things that absolutely must be satisfied in order to have a feasible solution. Moreover, with what we've presented so far, any assignment that meets the constraints would solve our problem. So this means the task is quite similar to the SAT problem: we have to find an assignment of classes (as compared with Boolean variables) such that an overall evaluation function returns a value of TRUE. Anything that violates the constraints means our evaluation function returns a value of FALSE. But this alone doesn't give us sufficient information to guide the search for a feasible solution.

We might be able to employ some strategy that could provide this additional information. For example, we might judge the quality of the solution not just by whether or not it satisfies the constraints, but for those assignments that fail to meet the constraints, we could tally the number of times that the constraints are violated (e.g., each time a student is assigned to two classes that meet at the same time we increase the tally). This would give us a quantitative measure of how poor our infeasible solutions were, and it might be useful in guiding us toward successively better solutions, minimizing the number of constraint violations. We could apply different operators for reassigning courses to classrooms, professors to courses, and so forth, and over time we'd hope to generate a solution that met the available constraints.

But then there are the *soft constraints*, the things we hope to accomplish but aren't mandatory. These include:

- Courses that meet twice a week should preferably be assigned to Mondays and Wednesdays or Tuesdays and Thursdays. Having these courses meet on consecutive days or with two or more days inbetween is not desired.
- Courses that meet three times per week should preferably be assigned to Mondays, Wednesdays, and Fridays. Other assignments are not desired.
- Course times should be assigned so that students don't have to take final exams for multiple courses without any breaks in between (final exam times are typically based on the time for the course).

- If undergraduate prerequisite courses are scheduled for the same day as their counterpart graduate courses, they should preferably be given earlier than the graduate course (this facilitates learning foundational material prior to advanced material in the same day).
- If more than one room satisfies the requirements for a course and is available at the designated time, the course should be assigned to the room with the capacity that is closest to the class size (this means that large auditoriums aren't used for small classes, thus enhancing student participation).

Certainly we could imagine many more such soft constraints. Any assignment that meets the hard constraints is feasible, but not necessarily optimal in light of the soft constraints. Here is where the problem gets sticky. First, we have to quantify each of the soft constraints into mathematical terms so that we can evaluate any two candidate assignments and decide that one is better than the other. Next, we have to be able to modify one feasible solution and, hopefully, generate another feasible solution that better meets the soft constraints.

Let's take the first issue: each soft constraint has to be quantified. Considering the first soft constraint, we could say that for each case where a solution is feasible, we could count up the number of times twice-a-week courses become separated by two or more days, or are placed on consecutive days, and use this as a penalty term. The lower the term, the better the solution. In fact, we could employ a similar approach to each of the soft constraints. But what would we do when we're through? We'd still need an overall method for considering the degree of violation of each of these constraints. That is, we'd have to determine the answers to questions such as: which is worse, scheduling five students to have back-to-back final exams, or scheduling back-to-back classrooms at opposite ends of the campus? Each of these possible trade-offs would have to be considered and quantified in some evaluation function, which poses quite a challenge!

Of course it's worse than that because even after all of these soft constraints have been quantified, we are still left with the problem of searching for the best assignment: the solution that is both feasible and minimizes our evaluation function for the soft constraints. Suppose we have found a feasible solution, but it doesn't do very well with regard to the soft constraints. Say we apply some variation operators to this solution and we significantly improve the situation with respect to the soft constraints, but in so doing, we generate a solution that violates one hard constraint. Now what? We might choose to discard the solution since it's infeasible, or we might see if we can repair it to generate a feasible solution that still handles the soft constraints well. Either way, this is typically a difficult chore. It would be even better to devise variation operators that never corrupt a feasible solution into an infeasible solution while still searching vigorously over the space of feasible solutions to find those that best handle the soft constraints. That's a nice aspiration, but it's often not much more than

wishful thinking. Effectively handling real-world constrained problems is one of the most challenging tasks we face.

1.5 The problem of proving things

Although it seems a bit strange, in our quest to solve problems, we sometimes make matters more difficult than they have to be. Having worked with students at many universities and in many countries, we've experienced the phenomenon that if you ask someone to *find* some solution to a problem, they'll typically find this much easier than if you had asked them to *prove* something about the solution, even when the two tasks are exactly the same mathematically.

As an example, think about any mathematical problem with just a moderate degree of difficulty. The task in the problem should be to find a particular value, whether it's the height of a building, the speed of a car, or the time to complete your homework. Regardless, the task should be to find the value of x . For example, you'll take four hours to fill a pool using a large pipe. You'll take six hours if you use a small pipe. How long would it take if you used both pipes? If the problem is formulated in terms of "find the value x " — such as, find the amount of time required to fill the pool using both pipes — this is a fairly easy task. But, for some reason, if we change the task and instead ask: prove that the amount of time required to fill the pool using both pipes is less than a , fewer students will manage this problem despite the fact that it's not the slightest bit more difficult. If you can find the time required to fill the pool, and if it's smaller than a constant a , then the proof is completed. To verify this for yourself, take the values given above for the times required to fill the pool using either pipe alone and test it out on your friends. Ask some to find the time required when using both pipes, and ask others to prove that the time required is less than 2 hours and 25 minutes.⁴

We believe that the reason for this aversion to proving things is that most people simply aren't experienced in proving things and don't know how to begin. Generalizing on this observation, many problems are apparently difficult simply because of the difficulty encountered when facing the question: "How should I start?"

Here's an example to help you exercise your ability to frame problems and get started on their solution:

Prove that any polyhedron must have at least two faces with the same number of edges.

No doubt, the first inclination when reading this is to think "Do I have to?" Yes, you do. So let's get started.

First, it's helpful to remind ourselves of some basic ideas about proving things. One way to prove something is simply to show that the conclusion

⁴Actually, the required time is 2 hours and 24 minutes.

follows directly from the given knowledge. For example, suppose that $a > b$ and $b > c$, then prove that $a > c$. You simply cite the transitive law and you're done. Another way to prove something is to consider the contrapositive. Recall that "if p then q " is logically equivalent to "if not q then not p ." Sometimes it might be easier to prove this relationship. Still another way to prove something is to assume that the condition you are trying to prove is false and then show that this is impossible. This is called *proof by contradiction*. Let's try that here.

Restating our problem, then, we need a proof that shows it's impossible to construct a polyhedron such that all of the faces have a different number of edges. The problem seems difficult because there aren't that many general theorems about polyhedrons. There is a famous one — Euler's theorem — which states that the following formula holds for every polyhedron:

$$v + f = e + 2,$$

where v , e , and f represent the number of vertices, edges, and faces of the polyhedron, respectively. But just how we might actually use this theorem in proving that all of the faces must have a different number of edges isn't very clear.

The initial step is the hardest part of the puzzle. Indeed, here the problem doesn't provide us with any convenient starting point. There's no number in the problem, nothing for us to factorize, divide by two, multiply by six, or anything else. In situations like this, it's often advantageous to introduce such a number ourselves: Let's consider a polyhedron with f faces. Now we have something, a variable, to work with.

We have to prove something about the edges of faces for this polyhedron. Each face has a number of edges. To say something about these numbers, it would be nice at least to know the range of values that we might expect. So, let's ask the question: what is the minimum number of edges that a face may have? The answer is three, and in that case the face is a triangle. This minimum number is independent of the total number f of faces of the polyhedron.

And what is the maximum number of edges a face may have? Well, each edge belongs to precisely two faces, so if we have a face with six edges, we know that the face is part of a polyhedron with at least seven faces (the current face plus six new faces, one for each edge; see figure 1.5). Thus, any face of a polyhedron that has f faces in total can't have more than $f - 1$ edges, since a new face originates from each edge.

This concludes the proof.

If this fact surprises you then take note that you have lost sight of what you are trying to prove. Go back and remind yourself of the problem. The reason the proof is complete is because if there are f faces in total, and if each face must have a number of edges between 3 and $f - 1$, then some repetitions in the number of edges must occur across the f faces. There are more faces than possibilities for the number of edges, so you'll have to use some of these numbers more than once. Therefore, at least two faces will have the same number of edges.

The keys to solving this problem were to invent a starting point to pursue and not lose sight of the goal.

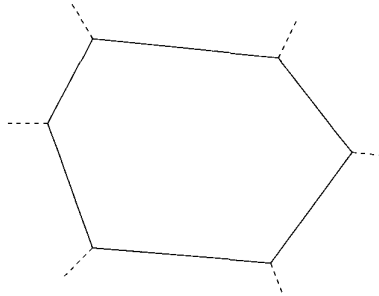


Fig. 1.5. A single face of a polyhedron with six edges

1.6 Your chance for glory

Now that you're primed for solving a problem, here's your chance to *prove* you can do it. This problem was offered to us by Peter Ross of the University of Edinburgh.

Mr. Smith and his wife invited four other couples for a party. When everyone arrived, some of the people in the room shook hands with some of the others. Of course, nobody shook hands with their spouse and nobody shook hands with the same person twice.

After that, Mr. Smith asked everyone how many times they shook someone's hand. He received different answers from everybody.

How many times did Mrs. Smith shake someone's hand?

We encourage you to put the book aside and try to frame this problem for yourself. Try to think of a starting point, maybe a graphical image that describes the problem, and see if you can follow it through to a successful conclusion.

The reason that this problem is a challenge is that once again there's no obvious starting point, but it's really quite simple to develop a graphical model for the problem (see figure 1.6).

This is a good model because we can clearly see Mr. Smith (shaded circle) and the remaining nine people (including his wife). Now, what information do we have? (Remember, think about all the given information.) The only information provided in this problem is that Mr. Smith asked every person in the room how many times they shook someone's hand, and that all of the answers he received were different.

What answers did he get then? Well, the minimum number of handshakes is zero: it's possible that some antisocial person didn't shake anyone's hand at all. But what is the maximum number of handshakes for a single person? It is more difficult to think of *asking* this question than it is to answer it. The maximum number of handshakes for a person is eight, as there are ten people

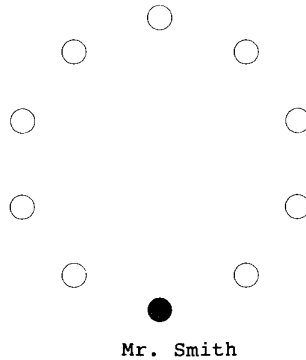


Fig. 1.6. Mr. Smith and the other people

in the room, and a person can't shake hands with himself or herself nor with his or her spouse.

Let's collect the facts we have now. Mr. Smith asked the question to nine people in the room and all of the answers were different. Furthermore, each answer was a number between zero and eight. Therefore, the answers he received were zero, one, two, three, four, five, six, seven, and eight (not necessarily in that order, of course!). So, we can update our model accordingly (figure 1.7).

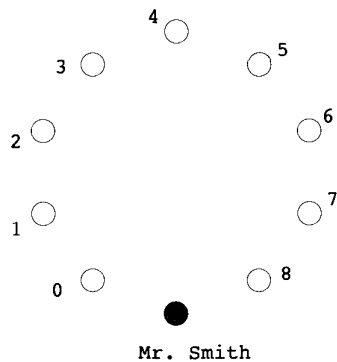


Fig. 1.7. Mr. Smith and the other people. The number of handshakes is indicated for each person (except Mr. Smith).

What's the next step, then? What can we infer? It seems, not much. We've already taken advantage of the fact that all of the answers were different, and we now know all of these answers. But how many times did Mrs. Smith shake hands? And which of the people in figure 1.7 is Mrs. Smith, anyway? If we only knew who has shaken hands with whom, then everything would be easier.

But, can't we? Let's try to draw all of the handshakes that were exchanged. The person 8 (we'll name everyone except Mr. Smith by the number of handshakes they exchanged) shook hands eight times, i.e., with everyone else in the room except himself or herself and his or her spouse. We can do two things based on this observation. We can draw all of the handshakes made by person 8 (see figure 1.8) and we can also conclude that the spouse of person 8 is person 0.

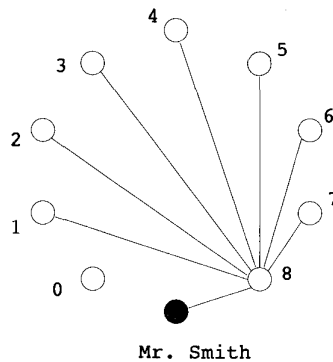


Fig. 1.8. Mr. Smith and the other people. Each handshake from person 8 is indicated.

Did the proverbial light bulb just turn on? We can turn our attention to person 7. This person exchanged seven handshakes, i.e., with everyone in the room except himself or herself, his or her spouse, and the spouse of person 8. Again, we can add all of the handshakes made by person 7 to our model (see figure 1.9) and we can also conclude that the spouse of person 7 is person 1.

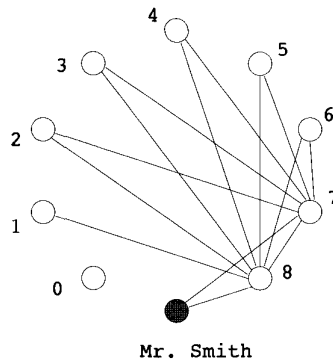


Fig. 1.9. Mr. Smith and the other people. All of the handshakes from persons 8 and 7 are indicated.

We can repeat this reasoning for persons 6 and 5. After adding the lines that correspond to the handshakes they exchanged we find an interesting graph shown in figure 1.10. The immediate conclusions are:

- The spouse of person 6 is person 2.
- The spouse of person 5 is person 3.

Using these conclusions, we know that the spouse of Mr. Smith is person 4. Therefore, Mrs. Smith exchanged four handshakes.

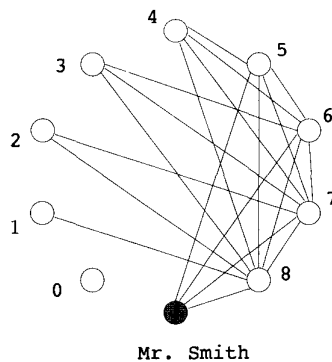


Fig. 1.10. Mr. Smith and the other people. All of the handshakes are indicated.

How did you do?

1.7 Summary

Problem solving is difficult for several reasons:

- Complex problems often pose an enormous number of possible solutions.
- To get any sort of solution at all, we often have to introduce simplifications that make the problem tractable. As a result, the solutions that we generate may not be very valuable.
- The conditions of the problem change over time and might even involve other people who want you to fail.
- Real-world problems often have constraints that require special operations to generate feasible solutions.

Furthermore, problem solving is often more difficult than it needs to be because we are threatened by the idea of *proving* things.

- Don't let the word *prove* intimidate you.
- Think about different ways to prove the solution. Sometimes it's useful to assume that what you are trying to prove is false, and then show that the false condition is impossible.
- Be ready to take a first step even if there doesn't seem to be anything in the problem for you to step on. Invent something, a variable, that describes a facet of the problem and see if that helps. Don't worry if you have to start over a few times before you finally find a path that takes you to the answer.

One way to facilitate taking the first step is to understand the search space: what are the variables? What are their possible values? What are the constraints? Most of all:

- Don't lose sight of the goal!

If you forget what you are trying to prove, you may as well watch television because your success rate will be the same either way: zero! Stay focused on the end result. Always ask yourself if what you are doing facilitates getting to where you want to go. You may not know the answer with certainty, but keep asking yourself this anyway. At least, you'll more quickly identify those paths that don't lead to the goal and you'll minimize the time spent on dead ends. At most, you'll readily identify the right path to choose and be on your way!