Game Design for Problem Solving with Python (Chapters 1-3)

Release 0.11

Steve Tanimoto

March 28, 2017

CONTENTS

CHAPTER

ONE

INTRODUCTION

1.1 Motivation

1.1.1 Why solve problems?

"Life is a series of problems that we must try and solve: first one, and then the next, and the next, until at last we die." – Old Lady Grantham in Julian Fellowes' *Downton Abbey*.

Problem solving is a necessity of life. Our problems are not only our personal problems, but societal problems. They include global warming, weapons proliferation, population growth and sustainability of food supplies, information pollution, and the emergence of drug-resistant diseases. Problems also include engineering challenges, mathematics problems, architectural design commissions, and even simple puzzles and games that people solve for pleasure. But most of the problems we solve are done out of necessity. We have to eat, to live, to enable our successful futures, and to pursue our personal and societal goals.

1.1.2 What Is a Problem?

A problem is an identified need. It is typically stated as a goal to be achieved by performing some action, especially when the action consists of an unknown sequence of steps. In order to prepare an Indian-cuisine, I will have to decide on a menu, gather the needed ingredients, and perform a sequence of steps that will allow three or four separate culinary dishes to be properly made within a given period of time. My problem is to plan the menu, identifying the dishes, the preparation steps, and the sequencing so as to be able to achieve the goal of having a successful Indian meal prepared on time.

There are many kinds of problems, including logical puzzles, troubleshooting of automobiles, diagnosing diseases, solving murder mysteries, designing bridges, designing social policies, composing music, and writing novels. Some are simple, such as determining a winning strategy at Tic-Tac-Toe, and others are so difficult, even to clearly formulate, that they are called "wicked" problems, like figuring out how to arrest global warming.

Problems all have certain common features. They all have some sort of starting situation. In a mathematics problem, the values of certain variables are given. The solver starts with that information. In the troubleshooting of an automobile malfunction, one starts with a particular car or at least a description of what it does not do properly. Problems all have some notion of *goal* or *objective*. Through some kind of problem-solving process, the solver(s) should reach a goal. They should achieve an objective. Problems usually arise in some sort of "context" in which there are "resources" available to help solve the problem. Resources take many forms, but frequently consist of knowledge sources. There may also be tools, suggested steps of action, or other resources available.

1.1.3 What Does it Mean to Solve a Problem?

To solve a problem is to take a series of steps that produce a solution or that lead to a goal by an efficent path. For example, if the problem is to find one's way through a maze, then solving the problem means to find a path out of maze. Such a path might be found by exploring sequences of steps through the maze until one of the sequences succeeds in reaching the outside or designated location. We can imagine two kinds of solutions to the maze problem: one kind is any path that gets out of the maze; the other is a shortest path out of the maze. The first kind of solution is sometimes called a "satisficing" solution. The second kind is called an "optimal" solution.

Not all problems have solutions. A goal may be unreachable with the available resources. Problems may have social dimensions that require shared understandings that are difficult to achieve.

1.2 Games and Their Relation to Problem Solving

Some problems, such as puzzles, already resemble games, because they typically have well-defined rules and a well-defined goal. Puzzles such as the Towers of Hanoi and Rubik's Cube offer a form of amusement in which the player gets to try to use ingenuity and analytical thinking. The ability to see patterns among differing puzzle states and to devise plans for transforming an instance of one pattern to another is a type of skill that is helpful in solving puzzles and puzzles help people to develop these skills.

Games, such as military strategy games, survival in fictional worlds, and other simulations, can offer an opportunity to confront many problems but in an environment that is somehow more forgiving than reality. You may lose such a game, but when you die in the game, you don't die in real life. The risk can be lower, the problems may be more tractable, and the knowledge that it's only a game, make game play more inviting than dealing with a complex real-world problem

As we will see later in the book, games can provide a form of cognitive scaffolding that helps human players become engaged with highly complex real-world problems. So-called "wicked" problems are tough to deal with precisely because they generally lack the clear structures that games provide. When we provide those structures, people can deal with the problems more effectively.

1.3 Design

Design is the process of synthesizing some object, document, or policy that meets a need. It might be a new kind of kitchen gadget, or an architectural building, or a new piece of software. It also could be a work of art or music. The process of designing something is a type of problem solving. The techniques of problem solving that we cover this book can very often be applied to design problems. We will see this especially in Chapter 7.

1.4 The Study of Problem Solving: Recent History

While problem solving has been going on throughout human history and undoubtedly before that, too, attempts to study the general process of problem solving explicitly are relatively modern. A milestone in the study of problem solving was the publication of George Polya's book in 1945 entitled *How to Solve It: A new aspect of mathematical method*. Its key idea is that, although mathematicians often presented their work as if it were invented magically, there is a process to solving problems that can be discussed and developed. Polya described a variety of strategies and guidelines that students and others could use to avoid getting stuck when working to find solutions to challenges in mathematics.

When Polya wrote his book, computers had been invented, but they were not part of the environment of typical mathematicians, scientists, engineers, or other people with problems. Indeed, the advent of widely accessible computer changes things. Even before computers became available to ordinary people, they began to influence the ways scientists thought about problem solving. The work during the 1950s of Allen Newell, Herbert Simon, and others exemplifies this.

The study of problem solving is a "cross-cutting" subject. Polya presented it in the context of mathematics, a field in which most concepts have clear definitions, and where the criteria for solutions are well agreed-upon. The work of Newell, Simon, and others in artificial intelligence was centered in computer science but it bridged into cognitive science and psychology. More recent work on methodologies for solving difficult problems outside of mathematics has emphasized the needs of educational approaches (focusing on pedagogy for problem solving) as well as social planning and government policy. One of the most difficult problems today is the whole phenomenon and social challenge of global warming, sometimes referred to more generally as "climate change." When dealing with this kind of problem, the subjects of earth science, political science, and many others obviously come into play. It should be clear now why I describe problem solving as a cross-cutting subject. As a metaphor for this aspect of the subject, we can consider how a cross-cut saw, working at a right-angle to the grain of wood in a set of logs, can expose the insides of many logs at once, just as solving a difficult problem can take us into aspects of multiple subjects.

Figure 1.1: The study of problem solving is a cross-cutting subject, involving psychology, computer science, education, and other disciplines. This metaphor is a reminder that parts of many areas of knowledge typically must be brought together, not only to design systems that support problem solving, but for successful problem solving itself.

1.5 Some Computer Systems for Solving

Modern technologies such as telecommunications and computing are making new approaches to problem solving possible or practicable. The internet has given a big boost to "crowdsourcing" – an approach to getting work done by recruiting human talent through open calls for participation. Computer technology provides people not only with access to communications networks but access to modern writing tools such as word processors, drawing programs, and computer-aided design software. Collaboration is supported through email, blogs, shared virtual workspaces, and video and voice conference-call technologies, such as Skype.

A variety of interesting computer systems or services have been developed that address one aspect or another of problem-solving activity. Let's consider some of them briefly.

- Innocentive: This company offers a web-based service that helps companies who have engineering problems or other problems (usually of a technical nature) atttract solutions from a world-wide pool of inventors and problem solvers, professional or amateur. Thus it facilitates the crowdsourcing of solutions to industrial problems.
- Amazon Mechanical Turk: This web-based service, offered by Amazon.com, allows clients who need lots of little tasks performed online to recruit and remunerate human workers to perform those tasks. The tasks might or might not be part of solving a problem, and the tasks must be structured by the clients.
- Koios: By providing a content management system designed to promote communication among people who want to work on solving complex, often ill-structured problems, this website helps promote a culture of problem solving.
- Galaxy Zoo: In astronomy, many new space images are becoming available from the Hubble and other telescopes. There is more data than scientists can analyze. This website allows members of the public to become

"citizen scientists" and help classify new galaxies according to a set of types that a user can learn in a few minutes.

- FoldIt: Protein folding problems form a general class in which a given molecule must be reshaped to achieve a minimum-energy state or other important configuration. The FoldIt game allows previously unskilled users (players) to learn to fold proteins and sometimes actually solve important instances of the problem.

Each of these systems has admirable qualities that contribute to a stronger base of approaches and support for problem solving with technology. However, each falls short in one or another of three important criteria needed to effectively bring problem solving to a level appropriate for the challenging problems facing us today. These criteria first-rate support for (a) collaboration, (b) computation, and (c) generality of problem domain. To support collaboration means to empower teams of solvers to work together in solving a problem. Innocentive's model is to promote competition, not collaboration, among solvers. Koios is a content-management system that does a good job of hosting communication about problem solving, but it does not provide ways for computation to play roles in the solving process. Amazon Mechanical Turk is good at assigning many workers from the Internet "crowd" to perform useful tasks that might be part of a problem-solving effort, but its support for collaboration, computation and generality are relatively shallow. Its workers generally don't communicate with one another, and their collaboration is limited to whatever task decomposition has been arranged by the client. FoldIt is a computer program designed for a particular kind of problem: protein folding. It supports computation by providing FoldIt users with tools for exploring the foldings of proteins, but the program cannot be used for problems in, say, mathematics. Its support for collaboration is limited. Galaxy Zoo, like Mechanical Turk, gives small pieces of work to multiple human workers, managing a somewhat rigid collaboration process. It's limited to galaxy classification, though.

What each of these systems is missing is having all three of the key aspects supported together: computation + collaboration + generality. In the remainder of this chapter, I discuss an approach to building problem solving technology that is based on the idea that these three aspects must be considered fundamental. I'll begin with a brief excursion into the *computational* aspect, explaining what any student of artificial intelligence has already encountered, but which is foundational for any rigorous study of problem solving.

1.6 The Classical Theory

Any problem solving system for people needs a language of terms and concepts that the people can use in their collaborations, and that help them understand what they are doing and how computers can assist them. The "Classical Theory of Problem Solving" (which I'll subsequently abbreviate as CTPS) grew out of work by early researchers in artificial intelligence. These researchers sought to develop computer systems that could reason like people and solve problems like people. Many of the early efforts were directed at game playing (e.g., Tic-Tac-Toe, Checkers, Chess, Go, Othello), and some were directed at problem solving in logic, algebra, and geometry. A particularly notable system was called the General Problem Solver (GPS) and it was largely based on what is described here as CTPS.

In the Classical Theory of Problem solving, we formally represent a problem as a three-component mathematical object as follows:

$$P = (\sigma_0, \Phi, \Gamma)$$

Here σ_0 is the *initial state*, Φ is a set of *operators*, and Γ is a set of *goal states.*

The initial state represents the situation given, before any solving activity proceeds. The initial state for a Towers-of-Hanoi puzzle has all the disks piled up on the first of the three pegs. The initial state for a Rubik's Cube puzzle could be a random arrangement of the parts, from which it is desired to get all little faces of each particular color on its own side of the cube.

Each member of the set of operators is an "operator" $\phi \in \Phi$ which consists of the following parts: (a) a *precondition* function, (b) a *state-transformation* function, and (c) an optional *parameter list*. In addition, the operator may have a name and a description, but these are for the convenience of users in talking about or controlling operators in a system or collaborative content. In a Towers-of-Hanoi puzzle, one operator could be described as "Move the topmost disk from Peg 1 to Peg 2". Its name might be "Move one-to-two". This operator has a precondition function that takes a state as its argument and returns true or false, depending on whether it is legal to make that move from that state. It also has a state-transformation function that can be applied to a state *s* if the precondition yield true for *s*. Then the result of applying the state-transformation function is a new state *s'* which represents what we get after making the move.

If an operator has a parameter list, when we call it a *parameterized operator*. Its precondition and state-transformation functions take $n+1$ arguments, where the state *s* is the first argument, and the remaining *n* arguments correspond to the parameters in the list. As an example of a parameterized operator, we can consider one whose description is "Move a disk from one peg to another" and whose parameter list is (source_peg_number, destination_peg_number). The legality of such a move is again computed by the precondition function, but it requires not only the current state *s* but also the numbers of the source and destination pegs in order to produce a verdict. Similarly, the state-transformation function of this operator requires the numbers of the two pegs in question, in addition to the current state, in order to make the desired move.

When an operator ϕ is applied to a state *s*, either nothing happens, because the precondition is false, or some state *s'* is returned. It is possible in some problems that $s' = s$. However, usually $s' \neq s$.

By applying operators to the initial state, new states are thus obtained. If this process is or could be repeated indefinitely, a set of reachable state would be accumulated. This set of reachable states we call the *state space* for the problem *P*. We sometimes use the symbol Σ to represent the state space. Note that σ_0 is always included in Σ. The state space of any problem always contains at least one state – the initial state.

The set Γ of goal states is a subset of Σ for which whatever criteria associated with having a goal state for the problem are satisfied. Not all problems are solvable. A problem might have an empty set of goal states: $\Gamma = \emptyset$.

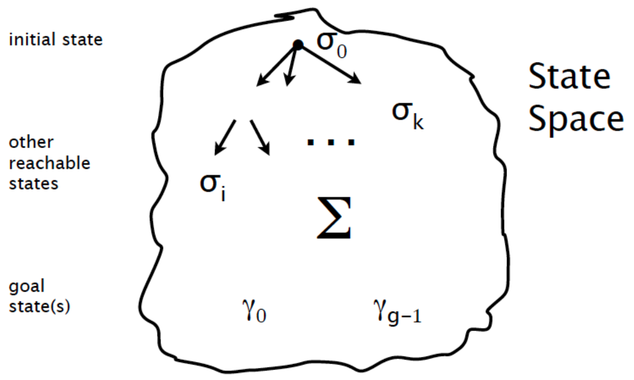


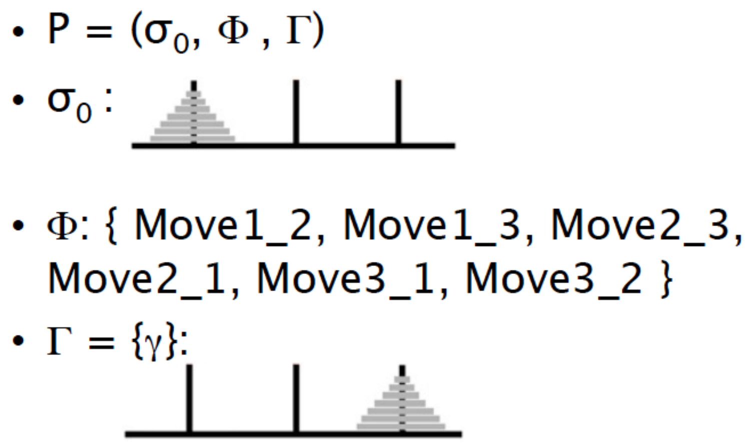
Figure 1.2: Diagrammatic illustration of the state space for a problem. Each state σ_i is derived from σ_0 by zero or more applications of operators in Φ. Some number, zero or more, of the σ_i may equal some $\gamma_g \in \Gamma$. The overall set of possible state is Σ, the state space. Hopefully, there is at least one goal state γ in Σ.

1.6.1 Towers of Hanoi as a CTPS Problem

Problem solving isn't just about simple, toy problems like puzzles, but about difficult, "wicked" problems with many aspects and complexities. We'll deal with both kinds of problems. However, we start with simple problems for the reason nicely stated by Judea Pearl in his book *Heuristics*.

> *The expository power of puzzles and games stems from their combined richness and simplicity. If we were to use examples taken from real-life problems, it would take more than a few pages just to lay the background...* – J. Pearl

Let's now give one complete example of a problem with its three components: initial state, operator set, and goal states. The Towers of Hanoi is a nice example, because it's easy to understand or already known, and we can therefore focus full attention on its representation. The figure below shows the three components of this problem.



- $P = (\sigma_0, \Phi, \Gamma)$
- σ_0 :
- Φ: { Move1_2, Move1_3, Move2_3, Move2_1, Move3_1, Move3_2 }
- $\Gamma = \{\gamma\}$:

Figure 1.3: The three components of the problem representation for Towers of Hanoi. This illustration uses little pictures of the puzzle to show the initial state and the (one) goal state.

As we'll see in the next chapter, full problem formulations will typically use program code to specify key problem components, and in addition, there will usefully be additional code to represent visualization methods so that each state of a problem space can be nicely displayed for users who are solving the problem or presenting a solution and how it was derived.

1.7 Introducing SOLUZION

While this book could have been written in a style that avoids the use of any particular software system, I believe that most readers would prefer to see more specific illustrations of problems, their formulations and solutions. Therefore, the book uses a particular system, called SOLUZION, for many of its examples.

SOLUZION is a specially-designed system that supports explorations in collaborative problem solving using the Classical Theory of Problem Solving (described earlier in this chapter). It is a fusion of several kinds of software technology: artificial intelligence, visualization, collaborative work, networking, and programming tools.

The purpose of SOLUZION is to support research and learning in the area of collaborative problem solving with computers. It might or might not be useful for solving specific problems, and that will depend on many factors – especially the degree to which the problem can benefit from the particular affordances that SOLUZION offers, and the extent to which a knowledgeable individual is able to effectively formulate the problem to exploit SOLUZION's affordances.

1.7.1 Solving with SOLUZION

Solving with SOLUZION is typically like finding the sequence of moves that takes you from the initial state of the Towers of Hanoi puzzle to the goal state. You do it by selecting an operator to apply to the current state, or by changing the current state to another state you have in your solving session. You may work alone or in a team. You usually work using an Internet browser such as Google Chrome, Mozilla Firefox, or Apple Safari.

Todo

Illustrate a solving session with SOLUZION, whenever it is finally ready.

1.7.2 Posing with SOLUZION

In order to solve a problem, it must already be formulated. If nobody has formulated it, then you might have to do it yourself. SOLUZION offers a means to formulate problems by performing a particular type of computer programming called problem posing for SOLUZION or problem coding for SOLUZION. The particular language in this programming is Python 3.

The details of problem formulation, including problem coding for SOLUZION are detailed in the next chapter.

1.8 Bibliographical Information

The essence of a problem, according to Thorndike (1898), is a situation where there is a goal that has to be achieved through an unknown sequence of actions. He came up with this definition in the context of his experiments with animals learning how to escape from specially constructed puzzle boxes.

George Polya's book "How to Solve It" (1945) was a milestone in the study of problem solving itself, focusing attention on what we might call the problem-solving metaproblem: how can be best solve problems, in general?

The Classical Theory of Problem Solving was developed by researchers in artificial intelligence during the 1950s, 60s, and 70s. The General Problem Solver (GPS) system of Newell, Shaw, and Simon embodied several ideas from the theory, including the notion of a space of states to be searched and the application of operators to obtain explicit representations of new states from existing ones. In their 1972 book, Newell and Simon used the term "problem space theory" to refer to these ideas. That term has gotten traction with the educational psychology community.

Solid introductions to state-space search can be found in Nilsson (1971) and Pearl (1984) or as part of a full course on artificial intelligence (e.g., Russell and Norvig 2010, or Tanimoto 1995)

The use of the classical theory as a structure for collaborative problem solving among humans using computers was demonstrated in the CoSolve system (Fan, 2013; Fan et al, 2012).

Other methodologies for human problem solving include TRIZ, for thinking about inventive solutions to engineering problems (Altshuler, 2005).

1.9 References

- Altshuller, Genrikh. (2005). *40 Principles: TRIZ Keys to Technical Innovation*. Worcester, MA: Technical Innovation Center.
- Fan, S. B. (2013). CoSolve: A Novel System for Engaging Users in Collaborative Problem-Solving. Ph.D. dissertation. Dept. of Computer Science and Engineering, Univ. of Washington, Seattle, WA 98195.
- Fan, S. B., Robision, T. S., and Tanimoto, S. L. (2012). CoSolve: A system for engaging users in computer-supported collaborative problem solving. Proc. of VL/HCC 2012: pp.205-212.
- Newell, A., and Simon, H. A. (1972). Human Problem Solving. Englewood Cliffs, NJ: Prentice-Hall.
- Newell, A.; Shaw, J.C.; Simon, H.A. (1959). Report on a general problem-solving program. Proceedings of the International Conference on Information Processing. pp. 256–264.
- Nilsson, N. J. (1971). Problem solving methods in Artificial Intelligence. (McGraw-Hill, Ed.) New York.
- Pearl, J. (1984). Heuristics: Intelligent Search Strategies for Computer Problem Solving. Reading, MA:Addison-Wesley.
- Polya, G. (1945). How to Solve It: A New Aspect of Mathematical Method. Princeton Univ. Press.
- Russell, S. and Norvig, P. (2010). *Artificial Intelligence: A Modern Approach, 3rd ed.* Pearson.

- Simon, H. (1996). The Sciences of the Artificial (3rd ed.). Cambridge MA: MIT Press.
- Thorndike, E., (1898). Animal intelligence: An experimental study of the associative process in animals. *The Psychological Review, Series of Monograph Supplements*, Vol.~2, No.~8, pp.551-553.

1.10 Exercises

1. Consider the maze shown in Figure 1.4.

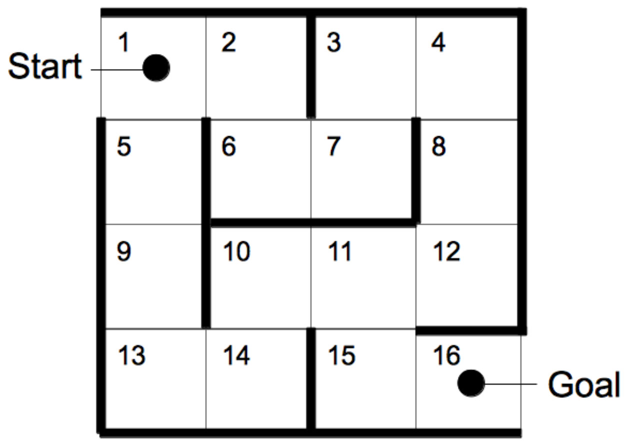


Figure 1.4: A maze with 16 cells, a start cell, and a goal cell. This problem has a nonunique solution.

   By a *simple path* in this maze we mean a sequence of cells from some cell A to some cell B, that moves from cell to cell one unit at a time without going through walls or repeating any cells. For example 10-11-12-8-4 is a simple path from cell 10 to cell 8. A solution to the maze is a simple path from cell 1 to cell 16. An optimal solution is a shortest solution path.

   1. Give an optimal solution.
   2. Give a satisficing, non-optimal path.
   3. Remove one wall (between two cells) so that there become exactly two optimal solutions. What are the numbers of the cells on each side of the wall being removed?

2. (a) Explain the similarity between solving a maze and solving a Towers-of-Hanoi problem.

   (b) Give an example of a satisficing, non-optimal solution for a Towers-of-Hanoi problem with 3 disks.

3. The Peg solitaire puzzle (English version) starts with a set of pegs laid out in the pattern of Fig. E2.

   A legal move involves moving a peg over an adjacent one (horizontally or vertically) to an empty hole and removing the peg that was jumped over. The goal is an empty board except for one peg in the center.

   1. Make up a code or data structure for representing a state of this puzzle, and show your coding of σ_0 the initial state. (You do not need to use a programming language, but thecoding should be textual.)
   2. Describe a typical operator for this problem.
   3. How many operators are there?
   4. What is your coding for the goal state?
   5. Does this problem have a unique solution?
   6. Are there any satisficing, non-optimal solutions?

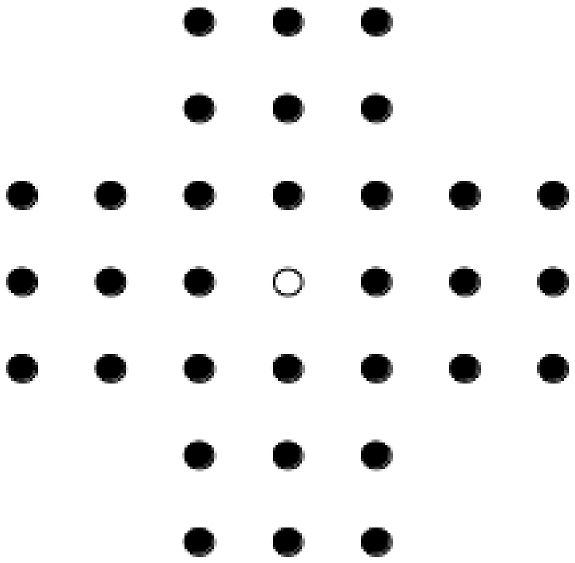


Figure 1.5: The board for Peg solitare at the initial state. The center hole is empty to start with, but each of the other holes contains a peg.

4. The game of Clue (known in the United Kingdom as Cluedo) involves two to six players who compete to solve a murder mystery. If you are not familiar with this game already, either read about it on Wikipedia or, better yet, play a game or two with someone who has the Clue game set. Then, describe the extent to which a player is "solving a problem" when playing the game. To what extent can you formulate this kind of problem solving in terms of the classical theory? What might correspond to each of the problem components $P = (\sigma_0, \Phi, \Gamma)$?
5. The game of Othello has standard rules that can be found online. However, for purposes of this exercise, let's assume that one player makes all the moves, alternating with black and white turns. Suggest a possible goal criterion for this activity. Suggest how this version of the game could be expressed as a problem of the form $P = (\sigma_0, \Phi, \Gamma)$.

CHAPTER TWO

PROBLEM FORMULATION

"The mere formulation of a problem is far more often essential than its solution, which may be merely a matter of mathematical or experimental skill." – Albert Einstein

Problem formulation has to be done before the solving process of applying operators to states can be started. However, we've postponed discussing the formulation process until after presenting the Classical Theory of Problem Solving in the previous chapter. Formulation can be easy or challenging, depending on the complexity of the problem and how well understood it happens to be.

2.1 The Basic Formulation Process

A problem begins with an identified need – some thing or condition that is desired but not currently present. For example, I'd like to have an Indian-cuisine dinner at home by dinnertime tonight. The formulation of the problem involves several elements that may be described as steps, resulting in a representation of the problem than can support the deliberate kind of solving process described in the previous chapter. These steps are the following.

1. Describing a need
2. Identifying resources
3. Restriction and simplification
4. Designing a state representation (i.e., a data structure for an abstract data type)
5. Designing a set of operators
6. Listing constraints on states and desiderata for solutions
7. If appropriate, providing for multiple roles within teams of solvers, designing a problem-space visualization, and/or designing any additional tools that solvers should have available when solving the problem.
8. Specifying in code the state representation, and operators.
9. Specifying in code any needed constraints, evaluation criteria, and/or goal criterion.
10. Specifying in code a state visualization method.
11. Specifying in code any additional computational functions and tools that should be provided to solvers to further empower them to address the problem. These may include methods for computational agents (e.g., parameter generators for] operators), or additional visualization methods directed at showing the problem space to the human solvers.

These eleven steps can be grouped into four phases: *preformulation*, *posing*, *essential coding* and *extended coding*. Preformulation comprises steps 1 and 2. Posing encompasses steps 3, 4, 5, 6, and 7. The essential coding phase consists of step 8 alone. Extended coding covers steps 8, 9, 10, and 11.

With the Classical Theory of Problem Solving, a problem *P* is a triple (σ_0, Φ, Γ). When formulating the problem, the most important parts, in order to get something that a computer system can do something with, are the initial state σ_0 and the set of operators Φ. In a system where humans will be deciding what operators to apply when, Γ is less important, because the humans can decide when they have reached a state that is good enough for a solution to the problem.

Once σ_0 and Φ have been specified, however, it is usually quite desirable to go on and specify Γ. This specification is typically done by constructing a predicate *goal(s)* that can compute, for any state s a boolean value that is True if $s \in \Gamma$ and False, otherwise.

2.1.1 Posing

Some of the formulation steps may be straightforward. For example, if we are formulating a problem that is already well understood, such as the Towers-of-Hanoi puzzle, then we can skip the preformulation phase entirely and begin with posing. We can also skip the restriction-and-simplification step, as this is not needed for puzzles, having essentially been done by the inventor of the puzzle. It is in the posing phase that we can decide what operators to have. For the Towers-of-Hanoi, there are at least a few alternatives for operator sets. Here's one.

Φ_a = { Move-from-1-to-2, Move-from-1-to-3, Move-from-2-to-1, Move-from-2-to-3, Move-from-3-to-1, Move-from-3-to-2 }.

Here is another.

Φ_b = { Select-peg-1, Select-peg-2, Select-peg-3 }.

With this set of operators, moving a disk from a source peg to a destination peg requires a sequence of two choices of pegs.

A third set of operators is this:

Φ_c = { Move-disk(source, destination) }.

This is a singleton set whose one operator has two parameters. A solver applying this operator must specify two arguments, each in the range 1 to 3.

Step 4 is the designing of a state representation. For a problem like the Towers-of-Hanoi, this means deciding on a form of memory organization to hold the information about what disks are on each peg at one moment in the solving process.

One form of represenation would be a list of three sublists, where the sublists contain integers that correspond to the disks. In Python, for example, such a data structure could be coded as follows:

```
[[5,4,3,2,1], [], []]
```

An alternative state representation is based on a dictionary structure, in which each peg gets a name-value entry. In Python, once again, this might appear as follows:

```
{'peg1':[5,4,3,2,1], 'peg2':[], 'peg3':[] }
```

Of primary concern, when designing the state representation, is what is and is not to be represented. For example, if a Tower-of-Hanoi puzzle contains a platform on which the pegs stand, the question might be raised as to whether the state representation should include information about the platform, e.g, `'platform-width':39`. We know this information to be irrelevant to the process of solving the puzzle. Thus we choose *not* to include it in the state representation.

Of secondary concern is the efficiency of the representation. Given that a computer will be working with possibly many states, we don't want to needlessly require large amounts of computer memory or large amounts of time for the manipulation of the states.

Yet there is still the very important concern, in addition to these first two critera, that a state representation be easily understandable to human users. The names of data-structure components should be in English (or other natural language) or based on natural language or terminology related to the problem, and they should be well-chosen or well made up. We'll illustrate that in our coding examples below.

2.2 About Our SOLUZION examples

The examples we consider below are specific enough that (a) all aspects are out in the open, and (b) a computer system can process them directly. We develop the formulations following the SOLUZION approach, again, for purposes of standardization (i.e., one set of conventions for all problems in this book), operationality (i.e., they can be run by the computer) and clarity (we can understand them easily).

Each example has the following parts:

```
METADATA
COMMON_CODE
COMMON_DATA
INITIAL_STATE
OPERATORS
GOAL_TEST
STATE_VIS
```

The METADATA section gives values to a variety of variables such as PROBLEM_NAME. These document what versions of software are involved, who the authors are, etc.

The INITIAL_STATE and OPERATORS are evidently the first two items in our CTPS triples (σ_0, Φ, Γ). The GOAL_TEST provides a function that takes a state as its argument and returns True if the state is an element of Γ.

COMMON_DATA holds the values of variables particular to the instance of the problem, and, in essence, common to all states. This data does not need be a part of any state representation, since it will not change from one state to another. For the Towers-of-Hanoi problem, we might have an instance where someone has decided that the number of disks will be 7. This then would be common data for that problem instance, and it could be represented by a dictionary such as `{'ndisks': 7}`.

In our examples, all the items, including the operators are expressed in Python 3.x code. However, the initial state, operators, and visualization may use functions and/or classes that are defined in the part called COMMON_CODE.

The STATE_VIS part of each example represents one or more methods for producing a display from a state. This portion of the formulation is typically platform-dependent (e.g., HTML5).

Note that our initial state could be expressed as a function of some of the common data. We'll see an example of that in the Towers-of-Hanoi details below.

2.3 Towers of Hanoi

We use the Towers-of-Hanoi puzzle (TOH) because so many people are already familiar with it that we can focus on certain details of the formulation right away, without having to first address the meanings of terms, solution criteria, etc. We've already used it in talking about posing. Here we continue the use of it, but now with an emphasis on the essential coding phase.

2.3.1 TOH Common Data

An instance of the Towers-of-Hanoi problem requires a particular number of disks. This could be decided by a problem-solving session initiator through a choice box. Within our problem formulation, we can represent the result as follows:

```
#<COMMON_DATA>
NDISKS = 5
#</COMMON_DATA>
```

2.3.2 TOH State representation

We'll use the dictionary representation suggested earlier for this problem. Here is a version of our initial state, in Python, for a 5-disk instance of the problem.

```
{'peg1':[5,4,3,2,1], 'peg2':[], 'peg3':[] }
```

Using this representation, We as posers and coders can explain to solvers, if they need to know (and they might, if we do not provide a suitable state-visualization method), that the disk numbers correspond to their relative sizes, with 1 for the smallest disk, and 5 the largest disk here.

According to their definition, dictionaries in Python can have their entries given in any order, without effect on the semantics. However, when coding, semantically equivalent expressions can be more or less clear to humans. The following state representation, for example, is less clear than either of the previous two examples of TOH state representations.

```
{'ZZZ':['a','b','c','d','e'], 'XXX':[], 'YYY':[] }
```

One remaining limitation of this representation, however, is that it ignores the possibility of puzzle instances having numbers of disks other than 5. In order to overcome this, we express the initial state as executable code that processes information from COMMON_DATA.

```
#<INITIAL_STATE>
INITIAL_STATE = {'peg1': list(range(NDISKS,0,-1)), 'peg2':[], 'peg3':[] }
#</INITIAL_STATE>
```

In this way, the initial state depends upon how many disks are involved in the problem. The Python `range(n)` normally creates a range object representing `[0, 1, \ldots, n-1]`. However, by using its three-argument form, we can get it to give us the elements we need in the decreasing order that we need.

2.3.3 TOH Operator representation

For any problem, the operators are typically more complicated to express than is the initial state. We can break down the representation of operators into several parts: standard functions, common code, and high-level operator representation.

An example of a standard function is `copy_state(s)` which always produces a "suitably deep" copy of a given state *s*. It is important that each state object be sufficiently separate from any other state object that mutating it (making a change) according to an operator's state-transformation function not cause any change to the original (parent) state. The job of the function `copy_state` is to copy all components of the given state *s* that need to be modified. For example, the three lists in our TOH state representation should be copied. (Note, however, that in principle, only the two lists that are affected by a particular disk movement really have to be copied, but having three different `copy_state` functions to support the different operators would complicate the formulation process. It would also make the resulting code more difficult to understand while having only a minor effect on reducing the memory requirements for states.)

Here is code for the high-level operator representations in our TOH formulation.

```
#<OPERATORS>
peg_combinations = [('peg'+str(a),'peg'+str(b)) for (a,b) in
                    [(1,2),(1,3),(2,1),(2,3),(3,1),(3,2)]]
OPERATORS = [Operator("Move disk from "+p+" to "+q,
                      lambda s, p1=p, q1=q: can_move(s,p1,q1),
                      lambda s, p1=p, q1=q: move(s,p1,q1) )
          for (p,q) in peg_combinations]
#</OPERATORS>
```

The above code works as follows. First a list of six pairs of of strings is constructed, using a Python list comprehension. The first of these string pairs is `('peg1','peg2')`. In a list of six operators is constructed using another list comprehension. Each operator is made by calling the constructor of the class using the name `Operator`. Each operator has its three parts: name, precondition, and state-transformation function. The name of the first operator is `"Move disk from peg1 to peg2"`. It's precondition tests whether it is legal, in the state `s` to move a disk from peg 1 to peg 2. Its state-transformation function makes that move, returning a new state. The lambda expression in the precondition has two keyword arguments `p1` and `p1`. These serve to bind values of `p` and `q` at the time the operators are created, and when the precondition is called, only one argument, the state, will be supplied in the call. (A function created this way in Python, using lambda arguments to bind the values of variables needed in the function body but where the values are coming from the environment outside of the body, is called a *closure* in programming-language parlance.)

The Common Code portion of the formulation is as follows.

```
#<COMMON_CODE>
def copy_state(s):
  # Performs an appropriately deep copy of a state,
  # for use by operators in creating new states.
  news = {}
  for peg in ['peg1', 'peg2', 'peg3']:
    news[peg]=s[peg][:]
  return news

def describe_state(state):
  # Produces a textual description of a state.
  # Might not be needed in normal operation with GUIs.
  txt = "\n"
  for peg in ['peg1','peg2','peg3']:
      txt += str(state[peg]) + "\n"
  return txt

def can_move(s,From,To):
  '''Tests whether it's legal to move a disk in state s
     from the From peg to the To peg.'''
  try:
   pf=s[From] # peg disk goes from
   pt=s[To]   # peg disk goes to
   if pf==[]: return False  # no disk to move.
   df=pf[-1]  # get topmost disk at From peg..
   if pt==[]: return True # no disk to worry about at To peg.
   dt=pt[-1]  # get topmost disk at To peg.
   if df<dt: return True # Disk is smaller than one it goes on.
   return False # Disk too big for one it goes on.
  except (Exception) as e:
   print(e)

def move(s,From,To):
  '''Assuming it's legal to make the move, this computes
     the new state resulting from moving the topmost disk
    from the From peg to the To peg.'''
  news = copy_state(s) # start with a deep copy.
  pf=s[From] # peg disk goes from.
  df=pf[-1]  # the disk to move.
  news[From]=pf[:-1] # remove it from its old peg.
  news[To]+=[df] # Put disk onto destination peg.
  return news # return new state

def goal_test(s):
  '''If the first two pegs are empty, then s is a goal state.'''
  return s['peg1']==[] and s['peg2']==[]

def goal_message(s):
  return "The Tower Transport is Triumphant!"

class Operator:
  def __init__(self, name, precond, state_transf):
    self.name = name
    self.precond = precond
    self.state_transf = state_transf

  def is_applicable(self, s):
    return self.precond(s)

  def apply(self, s):
    return self.state_transf(s)

#</COMMON_CODE>
```

2.3.4 TOH State Visualization

So far this formulation does not provide any method for visualization of the states of TOH. We can remedy this with the following.

```
#<STATE_VIS_FLASK_SOLUZION>
import svgwrite
from TowersOfHanoi import NDISKS

DEBUG = False

# The following control the layout of the TOH state displays.
MARGIN = 10
DISK_HEIGHT = 16
DISK_SEP = 4
DISK_MAX_WIDTH = 20 * (1 + NDISKS)
PEG_HEIGHT =  NDISKS*(DISK_HEIGHT + DISK_SEP) + MARGIN*3/2
PEG_RADIUS = 4
PEG_SEP = DISK_MAX_WIDTH + MARGIN
BOARD_HEIGHT = PEG_HEIGHT + 2*MARGIN
BOARD_WIDTH  = 3 * DISK_MAX_WIDTH + 4*MARGIN

DWG = None
def render_state(state):
    global DWG
    DWG = svgwrite.Drawing(filename = "test-svgwrite.svg",
      id = "state_svg",  # Must match the id in the html template.
      size = (str(BOARD_WIDTH)+"px", str(BOARD_HEIGHT)+"px"),
      debug=True)

    # draw background rectangle
    DWG.add(DWG.rect(insert = (0,0),
      size = (str(BOARD_WIDTH)+"px", str(BOARD_HEIGHT)+"px"),
      stroke_width = 2*PEG_RADIUS,
      fill = "rgb(192, 192, 210)")) # sky blue?

    # draw pegs
    ybottom = MARGIN + PEG_HEIGHT
    ytop = MARGIN
    for idx, peg in enumerate(['peg1','peg2','peg3']):
      xc = MARGIN + DISK_MAX_WIDTH / 2 + idx*PEG_SEP
      if DEBUG: print("xc = " + str(xc))
      DWG.add(DWG.line(start=(xc, ybottom),
          end=(xc, ytop),
          stroke="red",
          stroke_width=str(2*PEG_RADIUS)))

      # draw disks
      dsks = state[peg]
      pad=3
      for jdx, disk in enumerate(dsks):
        disk_radius = int((DISK_MAX_WIDTH / NDISKS) / 2)*disk
        y = (ybottom-MARGIN/2-pad-DISK_HEIGHT/2)\
           -jdx*(DISK_HEIGHT+DISK_SEP)
        DWG.add(DWG.line(start=(xc-disk_radius-pad, y),
          end=(xc+disk_radius+pad, y),
          stroke="green",stroke_width=str(DISK_HEIGHT+2*pad)))

        DWG.add(DWG.line(start=(xc-disk_radius, y),
          end=(xc+disk_radius, y),
          stroke="blue",stroke_width=str(DISK_HEIGHT)))

    # Draw a base
    base_y = BOARD_HEIGHT - 7*MARGIN/8
    DWG.add(DWG.line(start=(0, base_y),
      end=(BOARD_WIDTH, base_y),
      stroke="brown",stroke_width=MARGIN))

    return (DWG.tostring())
#</STATE_VIS_FLASK_SOLUZION>
```

The comment line with `<STATE_VIS_FLASK_SOLUZION>` delineates the beginning of the code for the visualization of states.

Within the problem-template source file, these comments tell a SOLUZION problem processor how to split up code for client-server environments, when needed. The Brython reference indicates that this code is applicable within a browser environment where SOLUZION is set up to use the Flask server and *svgwrite* module for Python to create web graphics. The visualization code above belongs in its own file “TowersOfHanoi_SVG_VIS_FOR_BRIFL.py” and is imported by the main problem file “TowersOfHanoi.py”.

Here is an example of a state rendering with the above code.

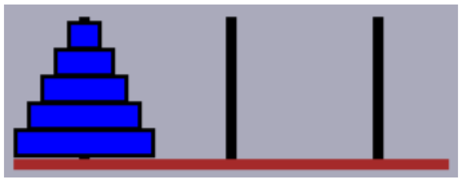


Figure 2.1: The initial state for Towers of Hanoi, similar to that rendered with the given state-visualization code.

2.4 Missionaries and Cannibals

Now let's consider another well-known problem, but different enough from TOH that we can begin to see a richer set of formulation approaches. (e.g., mechanically generating the operator set using lambda functions). We'll also use this problem to illustrate a problem-space visualization technique in the next chapter.

We describe the Missionaries and Cannibals ("M&C") problem as follows. Three missionaries and three cannibals are on the left bank of a river. They must all cross the river, and they have a boat. However, the boat's capacity is limited to three people at a time, and there are additional restrictions. The cannibals must never outnumber the missionaries either at the left bank, at the right bank, or in the boat, lest some missionaries be eaten. Also, the boat requires a missionary to steer it whenever crossing the river. (Note that there are other versions of this problem in which, say, a cannibal can steer the boat or the other restrictions are different.)

2.4.1 M&C Initial State

The initial state for Missionaries and Cannibals is illustrated in Fig. 2.2.

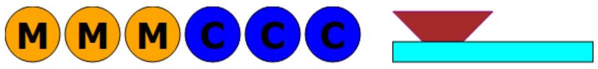


Figure 2.2: Missionaries and Cannibals: initial state.

A state can be represented by a pair of numbers *m* and *c* which give the numbers of missionaries and cannibals remaining on the left bank, respectively. (The numbers on the right bank are $3 - m$ and $3 - c$.) In addition the state should also indicate whether the boat is at the left or right bank of the river. The following state representation is slightly redundant in showing the numbers of missionaries and cannibals not only on the left side of the river but also on the right side.

```
#<INITIAL_STATE>
INITIAL_STATE = {'people':[[3, 0], [3, 0]], 'boat':LEFT }
#</INITIAL_STATE>
```

(Note that other state representations for M&C are certainly possible. One of the exercises in concerned with this issue.)

2.4.2 M&C Operator Representation

There are 5 operators. Each is of the form "Cross the river taking *x* missionaries* and *y* cannibals." The possible combinations (*x*,*y*) are these:

(1, 0), (1, 1), (2, 0), (2, 1), (3, 0)

We use a lambda expression to represent the various preconditions and another lambda expression to represent the various state-transformation functions.

```python
#<OPERATORS>
MC_combinations = [(1,0),(2,0),(3,0),(1,1),(2,1)]

OPERATORS = [Operator(
  "Cross the river with "+str(m)+" missionaries and "+str(c)+" cannibals",
  lambda s, m1=m, c1=c: can_move(s,m1,c1),
  lambda s, m1=m, c1=c: move(s,m1,c1) )
  for (m,c) in MC_combinations]
#</OPERATORS>
```

The class Operator is defined in the COMMON_CODE section, as are the functions copy_state, can_move, and move.

2.4.3 M&C Common Code

The first part of the COMMON_CODE for M&C are definitions of some constants that are used to help access parts of the state. For example, if *s* is a state, we can access the number of missionaries on the left bank of the river with the expression: *s['people'][M][LEFT]*.

```python
#<COMMON_CODE>
M=0  # array index to access missionary counts
C=1  # same idea for cannibals
LEFT=0 # same idea for left side of river
RIGHT=1 # etc.

def copy_state(s):
  news = {}
  news['people']=[[0,0],[0,0]]
  for i in range(2): news['people'][i]=s['people'][i][:]
  news['boat'] = s['boat']
  return news

def can_move(s,m,c):
  '''Tests whether it's legal to move the boat and take
     m missionaries and c cannibals.'''
  side = s['boat'] # Where the boat is.
  p = s['people']
  if m<1: return False # Need an M to steer boat.
  m_available = p[M][side]
  if m_available < m: return False # Can't take more m's than available
  c_available = p[C][side]
  if c_available < c: return False # Can't take more c's than available
  m_remaining = m_available - m
  c_remaining = c_available - c
  # Missionaries must not be outnumbered on either side:
  if m_remaining > 0 and m_remaining < c_remaining: return False
  m_at_arrival = p[M][1-side]+m
  c_at_arrival = p[C][1-side]+c
  if m_at_arrival > 0 and m_at_arrival < c_at_arrival: return False
  return True

def move(olds,m,c):
  '''Assuming it's legal to make the move, this computes
     the new state resulting from moving the boat carrying
     m missionaries and c cannibals.'''
  s = copy_state(olds) # start with a deep copy.
  side = s['boat']
  p = s['people']
  p[M][side] = p[M][side]-m     # Remove people from the current side.
  p[C][side] = p[C][side]-c
  p[M][1-side] = p[M][1-side]+m # Add them at the other side.
  p[C][1-side] = p[C][1-side]+c
  s['boat'] = 1-side            # Move the boat itself.
  return s
```

2.4.4 M&C Goal Test

The following predicate returns True if *s* is a goal state and False, otherwise.

```
def goal_test(s):
  '''If all Ms and Cs are on the right, then s is a goal state.'''
  p = s['people']
  return (p[M][RIGHT]==3 and p[C][RIGHT]==3)
```

2.4.5 M&C State Visualization

Code for state visualization using HTML5 graphics is the following.

```
import svgwrite
from Missionaries import M, C, LEFT, RIGHT

DEBUG = False
W=600; H=200
BOAT_LENGTH_FRAC = 0.2  # fraction of overall width W
BOAT_HEIGHT_FRAC = 0.2  # fraction of overall height H

def render_state(s):
  global W,H,BOAT_LENGTH_FRAC, DEBUG

  dwg = svgwrite.Drawing(filename = "test-svgwrite.svg",
    id = "state_svg",  # Must match the id in the html template.
    size = (str(W)+"px", str(H)+"px"),
    debug=True)

  # Background rectangle...
  dwg.add(dwg.rect(insert = (0,0),
    size = (str(W)+"px", str(H)+"px"),
    stroke_width = "1",
    stroke = "black",
    fill = "rgb(192, 150, 129)")) # tan

  # River in the middle (another rect.)
  dwg.add(dwg.rect(insert = (W*0.3,0),
    size = (str(W*0.4)+"px", str(H)+"px"),
    stroke_width = "1",
    stroke = "black",
    fill = "rgb(127, 150, 192)")) # turquoise

  # The boat
  boatX = 0.3*W
  boatY = H*(1-BOAT_HEIGHT_FRAC - 0.02)
  if (s['boat']): boatX=(0.7-BOAT_LENGTH_FRAC)*W
  dwg.add(dwg.rect(insert = (boatX,boatY),
    size = (str(W*BOAT_LENGTH_FRAC)+"px", str(H*BOAT_HEIGHT_FRAC)+"px"),
    stroke_width = "1",
    stroke = "black",
    fill = "rgb(192, 63, 63)")) # reddish
  dwg.add(dwg.text('B',
    insert = ((boatX+BOAT_LENGTH_FRAC*W/2),
              (boatY+BOAT_HEIGHT_FRAC*H/2)),
    text_anchor="middle",
    font_size="25",
    fill = "white"))

  # Missionaries
  Ms = s['people'][M]
  for i in range(Ms[LEFT]):
    draw_person(dwg, M, LEFT, i)
  for i in range(Ms[RIGHT]):
    draw_person(dwg, M, RIGHT, i)

  # Cannibals
  Cs = s['people'][C]
  for i in range(Cs[LEFT]):
    draw_person(dwg, C, LEFT, i)
  for i in range(Cs[RIGHT]):
    draw_person(dwg, C, RIGHT, i)

  if DEBUG:
    print(dwg.tostring())
    dwg.save()

  return (dwg.tostring())

def draw_person(dwg, M_or_C, left_or_right, i):
  "Represent a person as a colored rectangle."
  global W, H
  box_width = W*0.08
  box_height = H*0.3
  x = 0+W*0.01
  if left_or_right: x+=W*0.70
  x += i*W*0.1
  if M_or_C==M: color='green'; y=H*0.1
  else: color='violet'; y= H*0.6
  dwg.add(dwg.rect(insert = (x,y),
              size = (str(box_width)+"px", str(box_height)+"px"),
              stroke_width = "1",
              stroke = "black",
              fill = color))
  text = "M"
  if M_or_C==C: text = "C"
  dwg.add(dwg.text(text, insert = (x+box_width/2,y+box_height/2),
              text_anchor="middle",
              font_size="25",
              fill = "white"))
```

2.4.6 M&C Other Parts

Not shown in this example is the METADATA component. The COMMON_DATA component is empty. Note that although we might have put the global variables *M, C, LEFT,* and *RIGHT* in COMMON_DATA, they are more appropriate to COMMON_CODE, because they do not represent problem-instance-specific information. They are simply tools for performing state access.

2.5 The Circular Siding Shunting Problem

Our next example further enriches our experience in problem formulation, but without confronting any really tough issues. This puzzle is not as well-known as TOH or M&C, but still has a relatively constrained state space. But it is not as small a space as one might think.

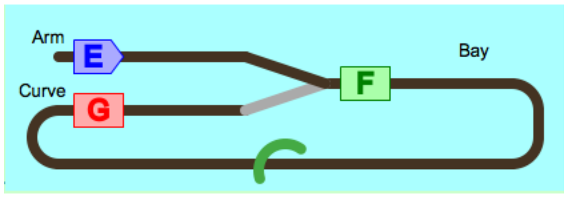


Figure 2.3: The Circular-Siding Shunting Puzzle.

In this puzzle, there is a railroad track that is roughly circular, with a switch going to a siding. There is a locomotive (the engine, designated 'E') and two carriages: a freight car ('F') and a gas car ('G'). Crossing over the track in the middle of the south side of the loop is a low bridge that only the engine can pass under. The track is organized into three zones: the Arm ('A') which is the siding at the north-west, the Bay ('B'), which is the section east of the switch, down to the bridge, and the Curve ('C'), which is the section of the loop to the west of the switch and south to the bridge.

At the start, the Engine is in the Arm, the Freight car is in the Bay, and the Gas car is in the Curve.

To solve the puzzle, you must find a sequence of operators that can be applied to exchange the locations of the Freight and Gas cars (i.e., exchange the contents of the Bay and Curve), and get the Engine back in the Arm. Here are the constraints. In each step, you can do one of four things: move the current train clockwise or counterclockwise one zone, change the switch setting (either to Arm or to Curve), couple a car that is in the same zone as the train (and adjacent to the train) to be part of the train, or uncouple a car from the train. The train is a chain of train carriages that includes the engine and which are all coupled together. As mentioned previously, the train may not pass under the low bridge unless the train consists of only the engine. Although the engine has an orientation (pointing clockwise on the track loop), it can move forward or backward, and it may connect to carriages at either end.

2.5.1 CSSP Initial State

We can store each state in the form of a dictionary with two entries: "sections" and "switchAtArm". The first entry tells what carriages are in each section of track and what couplings exist among the carriages. The second entry is True if the track switch points into the Arm and False if it points into the Curve.

Note that each track section is represented by a sublist of the sections entry, and each connected group of carriages within a section is represented by a sublist of the track section list. For example, if the Engine is coupled to the Freight car in the Bay, and the Gas car is in the Arm, with the switch pointing into the Curve, the state representation would be

```
{'sections': [[['G']], [['E','F']], []],
 'switchAtArm': False}
```

With this scheme, our initial state is given as follows:

```
#<INITIAL_STATE>
INITIAL_STATE = {'sections': [[['E']],
                              [['F']],
                              [['G']]],
                 'switchAtArm': True}
#</INITIAL_STATE>
```

2.5.2 CSSP Operator Representation

```
#<OPERATORS>
OPERATORS = [\
  Operator("GC - Go Clockwise to next section",
           lambda s: can_go(s, 'cw'),
           lambda s: go(s, 'cw')),
  Operator("GQ - Go Counter-clockwise to next section",
           lambda s: can_go(s, 'ccw'),
           lambda s: go(s, 'ccw')),
  Operator("SA - Switch into Arm",
           lambda s: can_switch(s, 'Arm'),
           lambda s: change_switch(s)),
  Operator("SA - Switch into Curve",
           lambda s: can_switch(s, 'Curve'),
           lambda s: change_switch(s)),
  Operator("CF - Couple forward",
           lambda s: can_couple(s, 'cw'),
           lambda s: couple(s, 'cw')),
  Operator("CB - Couple backward",
           lambda s: can_couple(s, 'ccw'),
           lambda s: couple(s, 'ccw')),
  Operator("UF - Uncouple forward",
           lambda s: can_uncouple(s, 'cw'),
           lambda s: uncouple(s, 'cw')),
  Operator("UB - Uncouple backward",
           lambda s: can_uncouple(s, 'ccw'),
           lambda s: uncouple(s, 'ccw'))]
#</OPERATORS>
```

Note that the operators are specified here with the aid of one class (Operator) and several helper functions. The operator class is a simple container for the three components: name, precondition, and state-transformation function and has the same definition as in the formulation for M&C.

The helper functions `can_go`, `can_switch`, `can_couple`, and `can_uncouple` do the work of checking that preconditions hold or not. The other helper functions do state transformations: `go`, `change_switch`, `couple`, and `uncouple`.

The state-visualization code is somewhat detailed, using HTML5 "canvas" graphics. A high-level sketch is shown below. The details can be found elsewhere.

2.5.3 CSSP State Visualization

```
#<STATE_VIS>
# (a sketch is given, to avoid too much detail)
set_up_canvas()
draw_track()
draw_track_zone_labels()
draw_switch(s.switchAtArm)
for zone in s.sections:
  draw_rolling_stock_in_zone(zone)
#</STATE_VIS>
```

Here is an intermediate state of the puzzle. In this state we can see the three carriages all coupled together in the Bay.

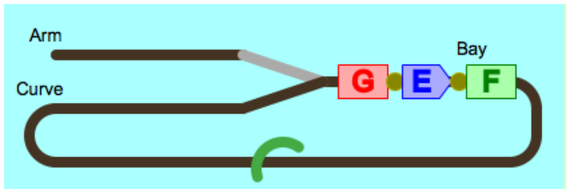


Figure 2.4: An intermediate state of the Circular-Siding Shunting Puzzle, where all three carriages are couples and in the Bay zone of the track.

2.6 A Design Problem

The problems we have considered so far in this chapter have been puzzles. A common characteristic of them has been the existence of well-defined goal states. Now let's consider a problem that has a much more elusive goal criterion. It exemplifies the large class of problems we call design problems. This is thus a preview of Chapter 4, which focuses on Design Problems.

Our problem here is to creating Mondrian-like, black-line, rectangle-division style graphics. It's an artistic problem, and the goal criteria are aesthetic and subjective.

The formulation we'll develop is one with a set of operators that add components. This style of operator will be common in formulations for composing things like graphics, text, music, or arbitrary objects.

2.6.1 Mondrian Initial State

Here a `MondRect` object represents a rectangle. The corner coordinates are values in the range [0.0, 1.0]. The default color of a rectangle is white, but red, blue, and yellow are also allowed. A state consists of a list called 'boxes' of these rectangles, plus an integer called 'selected' which indicates the index in the list of whichever rectangle is currently considered to be selected. This number must be in the range 0 to $n - 1$ where n is the number of rectangles.

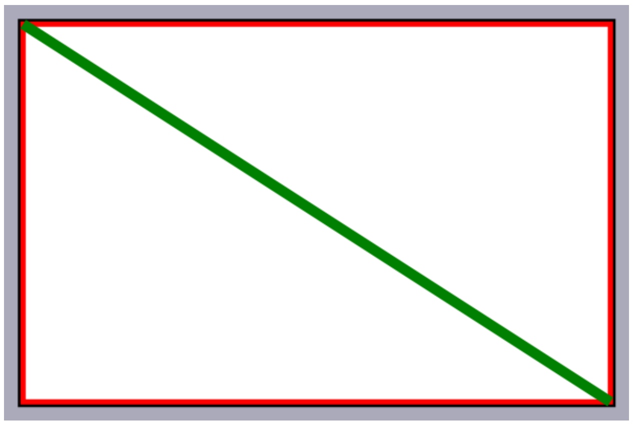


Figure 2.5: The initial state of a Mondrian design process. There is one rectangle. It is selected

The graphical representation of the initial state is shown in the figure. The visualization code provides a button that offers the solver the opportunity to turn the selected-box highlighting on or off. It's easier to work with it turned on, but the aesthetics of the current state can be better appreciated with highlighting off.

The highlighting puts a green rectangle on the currently selected box and a green diagonal line down the middle of it. The hightlighting on/off status is not considered here to be part of a state of the problem space but a viewing mode.

2.6.2 Mondrian Operator Representation

The operators for the Mondrian design problem are of four types: subdivision, selection, showing or hiding the selection, and coloring.

The first subdivision operator has the name "Divide with a horizontal line at fraction 0.25".

There are 10 of these subdivision operators, offering richness in possibilities without having a monstrously large branching factor. There are only two selection operators: "Select next box for alteration" and "Select previous box for alteration". These afford basic navigation within the list of rectangles, although if the list gets long, it will be tedious to move long distances through the list. (A more advanced alternative would allow specification of the selected box by clicking on it.)

There are four coloring operators. They permit changing the color of the currently selected box.

In total, this formulation has 18 operators. Their preconditions help limit the range of choices, but there is nothing to stop a solver from changing the selection endlessly or changing the color of a box endlessly. Subdivision is disallowed on a box that is smaller than a limit value in either its width or height.

```python
#<OPERATORS>
available_fractions = [0.25, 0.3333, 0.5, 0.6667, 0.75]

subdivision_operators =\
  [Operator("Divide with "+hv+" line at fraction "+str(f),
            lambda s, hv1=hv, f1=f: s['showing_selection'] and\
              selected_box_is_large_enough(s),
            lambda s, hv1=hv, f1=f: subdivide(s, hv1, f1))
   for hv in ['horizontal', 'vertical']
   for f in available_fractions]

selection_operators =\
  [Operator("Select next box for alteration",
            lambda s: s['showing_selection'] and\
              s['selected'] < len(s['boxes'])-1,
            lambda s: change_selection(s, 1)),
   Operator("Select previous box for alteration",
            lambda s: s['showing_selection'] and\
              s['selected'] > 0,
            lambda s: change_selection(s, -1))]

hide_or_show_selection_operators =\
  [Operator("Show which box is selected",
            lambda s: not s['showing_selection'],
            lambda s: set_showing_selection(s, True)),
   Operator("Hide which box is selected",
            lambda s: s['showing_selection'],
            lambda s: set_showing_selection(s, False))]

color_operators =\
  [Operator("Recolor the selected box to "+color,
            lambda s, c=color: s['showing_selection'] and\
              not c==s['boxes'][s['selected']].color,
            lambda s, c=color: recolor(s, c))
  for color in ['white', 'red', 'blue', 'yellow']]

OPERATORS = subdivision_operators + selection_operators +\
            color_operators + hide_or_show_selection_operators
```

#</OPERATORS>

2.6.3 Mondrian State Visualization

The essential quality of a Mondrian state display is the appearance of the "painting" that is currently represented. This graphic consists of one or more rectilinear boxes, with thick, black-line frames, and filled with one of the four colors white, red, yellow, or blue. The boxes, if there are more than one, are nested, having been created by subdivision.

An example of an intermediate state and a final composition are shown in the next two figures

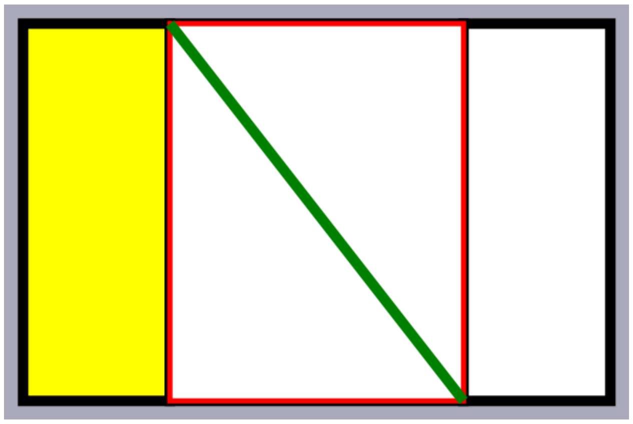


Figure 2.6: An intermediate state of a Mondrian design process.

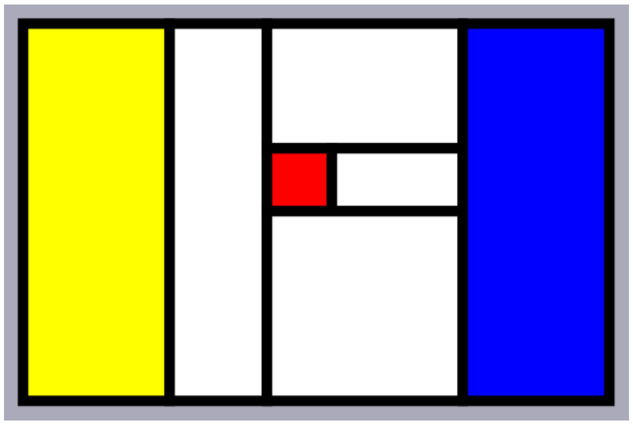


Figure 2.7: A final state of a Mondrian design process.

2.6.4 Aesthetic Criteria

The goal criteria for this aesthetic problem could be stated formally and reduced to code. Some of the criteria could be based on the numbers of elements of various kinds within the graphic. For example, we might have these:

- “There should be at least 5 and no more than 25 lines.”
- “There must be at least one block each of colors red, yellow, and blue.”
- “The number of vertical lines and the number of horizontal lines may differ by at most 3.”

However, we leave the subject of aesthetic evaluation for Chapter 7 which is on design problems.

2.7 Problem Posing As A Means to Explore a Topic

Some learning expeditions begin with a problem, and solving that problem serves as an impetus to learning about the fields to which the problem relates. If one’s goal is simply to learn about a field or subject, it may help to find a problem to drive the exploration. In mathematics, Brown and Walter explained how posing problems serves as a means to study the subject. The discuss where problems come from and how to synthesize problems that will be useful to solve.

For example, in mathematics, problems often arise out of curiosity, and the desire to understand. Brown and Walter offer a specific technique to turn an idea into a problem. They call their method “What if not?”. For example, starting with the idea of the Fibonnaci sequence as being generated by two initial numbers and an addition rule leads to questions such as What if the initial two numbers were not 0 and 1 but 10 and 7? Or What if the operation were to add not the last two numbers but the last three numbers. And then, What fraction does the ratio successive terms approach as we go further into the sequence? These problems seem more like factual questions than the challenge-style problems treated in this chapter. However, there is a fuzzy dividing line between these different sorts of problems.

There is a great potential for a rich interplay between problem posing and problem solving. We discuss this further in the next chapter, which is focused on a class of challenging problems (called “wicked” problems) for which formulation is especially problematical in its own right.

2.8 Conclusions and Discussion

Problem formulation is necessary when solving using CTPS. It’s got several challenges. Restriction and simplification is a big one, in the case of wicked problems. Designing representations, operator sets, visualization, and other solving “handles” is another.

In Chapter 5, we’ll address the challenge of designing good visualizations, both for the states themselves and for the more abstract aspects of solving such as the space of possible states for a problem. These visualization may be simple plots or they may be interactive computational objects that can be explored by clicking on components, examining substructures, or testing them under various conditions. The human experience of solving or helping to solve a problem is greatly affected by the design of these visualizations, for they can turn out to be the primary interface between the solvers and the rest of the system and team.

2.9 Bibliographical Information

Formulating a problem typically involves different knowledge and skills than does solving a well-structured problem. Work by Goel and Grafman (2000) suggested that solving well-structured problems tends to be more correlated with activity in the left hemisphere of the brain while working with ill-structure problems is associated with right-hemisphere activity.

The Lumsdaines (1995) use the term “problem definition” for what we are calling problem formulation. Their meaning is slightly different, however, because their purpose in a problem definition is to have an agreement between people on what the problem is. For us, a problem formulation is more than that, because it also serves as something a computer can understand.

In business contexts, methodologies for problem analysis have been developed, such as Kepner-Tregoe problem analysis (see Cosgrove, 2013).

Jackman and co-authors (2007) detail the process of formulating a problem as one of gradually working through a metaproblem space (MPS), in which the constructions of fragments of a problem representation correspond to moves through the MPS. This metaproblem space is in the mind of the student doing the formulating, although the student is probably unaware of it. The MPS can also be thought of a model for the realm of possible formulations for the problem, limited only by one's intuition about what is likely to be relevant or irrelevant to the problem.

Brown and Walter (2005) are concerned less with the formulation of a problem than with generating the question on which a problem is based. In mathematics, traditionally solving is taught but the process of coming up with problems is not, and their book provides ways to counter that tendency as well as arguments as to why problem posing is important.

The problems presented in this chapter represent a cross-section of relatively simple problems to formulate. The Towers of Hanoi is often used as an example in beginning programming courses, because there is an elegant solution approach based on functional recursion, and this scheme will work no matter how many disks are used in the problem. (However, the number of moves required is of the order of 2^n, where n is the number of disks. The Missionaries and Cannibals problem exemplifies simple puzzles with relatively small search spaces. The Circular Siding Shunting Problem is an example of a fairly wide category of problems known as railroad shunting problems. An online website giving a number of intriguing examples (Wyatt, 2017).

Many artists have developed well-known and somewhat predictable styles of painting. Piet Mondrian is known for geometrical arrangements of rectangles that are white or brightly colored with simple, primary colors, and having clear black lines separating the regions of the painting (Wikipedia on Piet Mondrian, 2017). Formulating a category of art such as the graphic designs possible with this chapter's Mondrian problem formulation is controversial with artists. Artists often work hard to develop a style, and if that style can be approximated by a computational scheme that might in any way compete with the artist, it can be perceived as a threat to the artist's trade or artistic identity. On the other hand, formulations such as that in this chapter can help art critics and students to understand and appreciate the artist's style, which more often that not, is much more sophisticated than can be captured by a simple problem formulation like the Mondrian formulation in this chapter.

Wicked problems, which we examine in depth in the next chapter, are usually not just ill-structured (as described by Jonasson, 2007) but hampered by social constraints.

2.10 References

- Brown, S. I. and Walter, M. I. (2005). *The Art of Problem Posing*, 3rd ed. Mahwah, NJ: Lawrence Erlbaum Associates.
- Cosgrove, I. (2013). Kepner-Tregoe Problem Analysis. In *Quality Matters* (blog). https://iancos.wordpress.com/2013/01/14/kepner-tregoe-problem-analysis/ Accessed March, 2017.
- Jackman, J., Ryan, S., Olafsson, S., and dark, V.~J. (2007). Metaproblem Spaces and Problem Structure. In Jonassen, D. (ed.) *Learning to Solve Complex Scientific Problems*. Lawrence Erlbaum and Assoc., pp.247-270.
- Jonassen, D. (ed.) (2007). *Learning to Solve Complex Scientific Problems*. Lawrence Erlbaum and Assoc.
- Lumsdaine, E., and Lumsdaine, M. (1995). Chapter 5: Problem Definition. In *Creative Problem Solving: Thinking Skills for a Changing World*. New York: McGraw-Hill.
- Wikipedia. (2017). Piet Mondrian. https://en.wikipedia.org/wiki/Piet_Mondrian Accessed March, 2017.
- Wymann, Adrian. (2017). The Model Railways Shunting Puzzles Website. http://http://www.wymann.info/ShuntingPuzzles/index.html Accessed March, 2017.

2.11 Exercises

1. Reformulate the Towers-of-Hanoi problem to use four pegs and N disks, where, as in the example formulation given, N is assigned a value in COMMON_DATA.
2. Reformulate the Towers-of-Hanoi problem to use M pegs and N disks, where both M and N will be assigned values in COMMON_DATA.
3. (a) Reformulate the Missionaries and Cannibals problem so that the boat may be steered by a cannibal without requiring a missionary in the boat.

   (b) What is the minimum number of moves required to solve this version of the problem?
4. (a) Reformulate the Circular Shunting problem such that the Gas car is permitted to pass under the bridge with the engine.

   (b) What is the minimum number of moves to solve the puzzle now?
5. Introduce one or more new operators into the Mondrian problem formulation.

   (a) a pair of operators to move the selection to the first and to the last box, respectively.

   (b) a pair of operators to split a box diagonally (resulting in two triangles). The triangles should be capable of being selected and/or colored, but not split further. One operator should use the upper-left to lower-right diagonal, and the other operator should use the upper-right to lower-left diagonal.

   (c) an operator to break up the selected box into 4 parallel, equal-width (or equal height) smaller boxes.
6. Examine several works of the artist Piet Mondrian by using a Google image search for Mondrian. Make a list of those paintings you find that use a similar style to the graphics that can be produced with the Mondrian problem formulation in this chapter. Compile statistics for the following features of the paintings you examine: NR: number of rectangles, FC: fraction of rectangles in a painting that are colored rather than white, AL: relative area of the largest rectangle in the image, AS: relative area of the smallest rectangle in the image, NH: number of horizontal black lines, NV: number of vertical black lines. Finally, make up a measure of "Mondrian beauty" that ranges from 0 to 10 based on these statistics, where 0 means "very unlike an actual Mondrian painting," and 10 means "very much like an actual Mondrian painting." The measure should be expressed as a formula involving the features listed above.

CHAPTER

THREE

WICKED PROBLEMS

"Ding, dong! The witch is dead. Which old witch? The wicked witch" – from the Wizard of Oz

Some problems are so broad, vague or otherwise unruly that we call them "wicked problems." An example is the problem of global warming and how to slow it. In this chapter we consider the various properties of wicked problems, and then we consider how to "tame" them so that we can begin to bring classical solving methods to bear on them.

3.1 Wicked Problems

Although any problem begins with an identified need or goal, some problems or so vaguely discussed that we can be hard-pressed to know where to start with them. For example, where does one begin to try to solve the problem of global warming? This is an example of a type of problem that we call "wicked."

While simple ("toy") problems such as the Towers-of-Hanoi are very important for developing theories, teaching them, and debugging software tools, they are not the real problems that need solving. Real problems are generally much more complex, and their parts may not be clear or well-defined. Some problems seem to have a number of impediments in the way of solving or even formulating them. These are known as "wicked" problems. We discuss them here for several reasons. First, it helps us to better understand the realm of problem solving to study many types of problems. Second, it helps us to clarify our problem-formulation methodology and better understand how to apply it, even when the problems seem to be intractable. Thirdly, in spite of the great difficulties and even seeming or real intractabilities of wicked problems, by formulating aspects of them we can better understand where the real difficulties lie.

We begin with a discussion of what makes a problem wicked.

3.1.1 The Ritter & Webber Criteria

During the 1960s, the development of the theory of problem solving for computers to be artificially intelligent led to some pushback in fields further away from computing. The idea that a simple theory and mechanical process could solve all kinds of problems, while appealing to some people, was considered silly, impractical and/or premature by others. In the field of policy planning, the idea that problems could be reduced to a simply formulated, mechanically-oriented representation, was considered objectionable. In a famous paper by Rittel and Webber, ten criteria for "wicked problems" were laid out, so that policy planners could avoid criticism for not applying the new theory. Whether or not the Classical Theory of Problem Solving has anything to offer in solving wicked problems, the ten criteria remain useful in analyzing challenging problems. The criteria are the following, and the wording is that of the authors.

1. "There is no definitive formulation of a wicked problem."
2. "Wicked problems have no stopping rule."
3. "Solutions to wicked problems are not true-or-false, but good-or-bad."
4. "There is no immediate and no ultimate test of a solution to a wicked problem."

5. "Every solution to a wicked problem is a "one-shot operation"; because there is no opportunity to learn by trial-and-error, every attempt counts significantly."
6. "Wicked problems do not have an enumerable (or an exhaustively describable) set of potential solutions, nor is there a well-described set of permissible operations that may be incorporated into the plan."
7. "Every wicked problem is essentially unique."
8. "Every wicked problem can be considered to be a symptom of another problem."
9. "The existence of a discrepancy representing a wicked problem can be explained in numerous ways. The choice of explanation determines the nature of the problem's resolution."
10. "The planner has no right to be wrong."

Although by this description, every wicked problem has all ten characteristics, in the rest of this book I will consider a problem to be wicked if, as presented, it seems to have characteristics 1 and 6. The other characteristics might or might not be present. There is an additional characteristic of many wicked problems that's not in the list above, and that is the existence of opposing stakeholders, some of whom may not agree with a formulation of the problem for financial, religious, political, or personal reasons. We'll address this aspect in our "case studies" chapter.

However, part of the purpose of this section is to

suggest that any apparently wicked problem can be "tamed" by a process of careful analysis and problem formulation. Taming does not imply "solvable" but that it can be at least formulated in a meaningful way with a clarification of its essential elements.

3.1.2 Examples of Wicked Problems

Climate change is a wicked problem. It's a problem because there is a perceived need for action to stop or limit global warming before way too much damage is done and further warming becomes unstoppable. It's wicked because, first, there is no definitive formulation of it; people and governments to not agree on a formulation, in spite of much work by organizations such as the Intergovernmental Panel on Climate Change, and second, there is no enumerable set of possible solutions or a well-defined set of operations that could be incorporated into a solution plan.

Other wicked problems include the proliferation of drug-resistant bacteria, weapons proliferation, a growing glut of unreliable information online, extinction of species, and human population growth.

Let's consider several examples of wicked problems and then treat one in detail: global warming.

3.1.3 Drug-Resistant Diseases

During the 20th century, many of the worst diseases from previous centuries were effectively stopped by the use of antibiotics. However, during the late 20th century and early 21st century, some of these diseases have been making a comeback, evolving with increased resistance to the antibiotics. If this process continues, the chances of major and deadly epidemics will continue to increase, and humanity may find itself one again having to contend with plagues and major scourges.

Until Louis Pasteur discovered that diseases were caused by the reproduction of microorganisms, and Fleming discovered how penicillin inhibited the growth of bacteria, diseases such as cholera, tuberculosis, and anthrax could cause widespread epidemics and human suffering. Later, various other antibiotics were developed that could kill the agents of many infectious diseases.

However, during the late 20th century, antibiotic use became so widespread, and often not-properly controlled, that conditions became ideal for the evolution of antibiotic-resistant bacteria. Today, antibiotics are widely used in agriculture, not necessarily to cure animal disease, but to try to prevent it. Doctors often prescribe antibiotics to patients just in case, even if the risk of bacterial infection is relatively small. When a course of antibiotics is not taken to completion, it can often be the case that it is then only the relatively antibiotic-resistant bacteria that are still alive, and then left to prosper after all the other competing bacteria have been nicely removed for them. The result is that new strains of disease agents are prospering, and modern medicine doesn't have any effective new drugs to fight them. And even if it did, the bacteria would just continually evolve to become resistant to those drugs, due to the way antibiotics have come to be used.

Further information about this problem can be found in *Breakout* (Lappe 1995) and at the websites of the Center for Disease Control and Drugs.com.

3.1.4 Urban Management

The management of a modern city such as Chicago, can be seen as a wicked problem. Because of the many dimensions of the problem, including the city's economy, crime statistics, education system, and logistics systems of transportation, there is no definitive formulation of the problem of fixing the city. Thus criterion 1 is met. Also, there is no clear set of actions that can be taken to address the problem. We can consider general actions, such as spending money to hire more police, but how much to spend is not clear, and there are countless alternatives in how to use increased funding to address the problem. Thus criterion 6 is met. Even if increased spending is considered an actionable option, the question will remain of where to get the funding – who would pay the taxes or fees, and different stakeholders may oppose one another when the funding issue comes to the fore. Thus the opposing stakeholders criterion applies.

The complexity of urban management has captured the imagination of game designers and players, particularly with the SimCity game series, started by Will Wright. The first release was in 1989, published by Maxis, currently part of Electronic Arts. The basic game offers the role of town mayor to the player, who manages the growth of the city by declaring zoning and allocating funds to projects, while a simulation of the city's operation is running in real-time.

3.1.5 Weapons Proliferation

As an example of another wicked problem ready for creative, serious formulation, we next describe the issue of increasing access to amplifiers of deadly force.

From bullying on the playground to nations skirmishing over where border lines should be drawn, people have historically dealt with violence and threats of violence by arming up. Whether it is to try to increase one's power or to allay fear of violence by others, a person, group or country may seek to acquire weapons, and in particular, weapons that are more powerful than believed to be held by the "other."

An arms race is a continuing process of acquiring more and more lethal weapons in efforts to have more than the other side. Even if an arms race ends without a battle, the result is a state in which a small mistake of communication, intention, understanding or human sanity, can have a high cost. For example, during and after a nuclear arms race, there is a persistent risk of complete destruction of modern civilization. Even after a small arms race, in which all families in a neighborhood buy guns out of fear of crime, there is a persistent risk of a prank, trick or quarrel causing manslaughter or murder.

It seems that no effective solution to this problem has yet been found. Meanwhile, weapons technology continues to be developed.

3.1.6 Population Growth and Sustainability

A host of wicked problems are wrapped up on the buzz-word "sustainability." What are people talking about? What are some problems of sustainability and how can they be formulated?

- Birth-Rate Pressure. The human population of planet Earth is not just at an all-time high, but over the last couple of centuries, the rate has been exponential. Besides the biological tendency of individuals in all species to reproduce, there are a number of factors that push humans to have more children. These include cultures that value large families, as well as religions that proscribe contraception or any artificial means of stopping pregnancies. The result is ever more births.

- "Developing Countries" and the politics of development. In efforts to achieve a form of equity between industrialized (or post-industrial) countries and countries that have not yet experienced such industry, the assumption is often that the so-called "developing" countries should strive to be like those which are already industrialized. This further puts pressure on both Earth population and its natural resources.

There are many separate manifestations of the population growth and world sustainability issue. These include the global fresh-water supply and how it is threatened, ocean acidification, and what it means for fish populations worldwide, rising sea levels and its consequences for coastal communities, air pollution and its effects on health, and the finiteness of fossil fuel sources.

3.1.7 The Information Glut and Information Pollution

Is it a problem that search-engine results do not always tell us what we want to know? Are the engines getting better? Are the results getting better? Results are often getting worse. Why? Does it actually matter? What can be done about it? Here are some questions and subissues related to information pollution and "false news."

Freedom of information is complicated. When is "speech" information and when is it not? And when is it libel, slander, or crying "fire!' What should be the limits of "free speech" when it is blatantly false? Who should judge what's allowed to be online? Is it OK to carry propaganda and blatantly false information on websites deceptively set up to look like legitimate news sites? Dealing with our increasingly polluted information environment is an emerging wicked problem, brought to the fore by the rise of the internet, and some of the ways some people are trying to use it.

3.2 A SOLUZION Formulation for the Climate Change Problem

3.2.1 The Problem

As a case study in the formulation of a wicked problem, we focus in on climate change.

The IPCC Report of 2013 stated,

> "Warming of the climate system is unequivocal, and since the 1950s, many of the observed changes are unprecedented over decades to millenia. The atmosphere and oceans have warmed, the amounts of snow and ice have diminished, sea level has risen, and the concentrations of greenhouse gases have increased."

The problem is to stop the rise of average temperatures before they go way too high. One of the difficulties is that world economic development tends to contribute to global warming, and yet sustainable development (i.e., without contributing to global warming) is difficult. Sustainable development is defined as follows:

> "Sustainable development: development that meets the needs of the present without compromising the ability of future generations to meet their own needs.
>
> —*Our Common Future* (1987 report of the World Commission on Environment and development, United Nations)

The biggest contributor to global warming is the accumulation of certain gases, such as carbon dioxide, in the atmosphere.

> "Continued emisisons of greenhouse gases will cause futhre warming and changes in all components of the climate system. Limiting climate change will require substantial and sustained reductions of greenhouse gas emissions."
>
> —also in the IPCC Report of 2013.

This is a problem that most people who think about it find to be daunting. It can overwhelm potential solvers with its complexity and the prospect of getting the cooperation of many people. Many of the people have short-term interests that are opposed to the intervention measures that seem to be required. In fact, some have taken the position that the problem doesn't exist, and they argue that nothing needs to be done. Climate change deniers may even go so far as to attack, verbally or otherwise, those who would solve the problem. There certainly are political and economic obstacles in the way of possible solutions.

Climate change seems to lack a definitive formulation, and the set of measures which might be invoked in a project to solve the problem does not seem to be clear. Thus climate change is a wicked problem.

3.2.2 Preformulation

The first phase of formulation is to describe the need that is to be satisfied by a solution, and also to identify the possible resources that might be available in further formulation and solution of the problem.

The need is to find a way, possibly with a combination of measures, that has a high likelihood of preventing the rapid rise of the average earth surface temperature. Possible resources include knowledge about global warming, actions of individuals, governments, non-governmental organizations, and companies, etc. The resources may include funding, expertise, programs of action, etc.

3.2.3 Restriction and Simplification

In order to formulate the problem, we must develop a representation and set of operators that correspond to understandable and meaningful problem-solving steps. Given the complexity of the problem, it is important to ask the question, “What essential aspects of the climate change problem might form parts of the states in a formulation according to the classical theory?”

3.2.4 Basic Physical Model

The earth’s atmosphere, oceans, land cover and land activity are all complex systems. Any detailed mathematical model of them will have large numbers of variables, perhaps thousands or millions, and numerous interaction formulas. However, the earth’s overall trend towards higher average temperature is something that is measurable and describable with amazing simplicity: a single valued function of time.

If we are going to try to find a comprehensible and justifiable formulation of the global warming problem, it makes sense to start out by looking for simple, understandable variables, and ones which we can relate to one another using trustworthy representations. Although people’s ideas and attitudes are a big part of the climate change problem, because the conceivable actions have both financial and human costs, we will turn first to the realm of the physical sciences for insight.

Fortunately, the field of physics has something solid to offer us. Nearly all the warming of the earth is due to the sun. The very small amounts of electromagnetic radiation (“EMR”), including light and heat, from other stars, does also contribute in principle, but is not signficant in the balance. Even if it were, our analysis would simply include another term for the combined effects of all other sources of EMR. The physics tells us that energy in a closed system is conserved. In the case of the sun and the earth, in equillibrium, the energy reaching the earth must be equal to the energy sent back out from the earth. Without an equillibrium, either the earth would get hotter and hotter or colder and colder, as it can neither store up an infinite amount of energy nor emit an infinite amount of energy. In general, if more energy is received from the sun, then more energy must be emitted by the earth. However, the average temperature of the earth will also rise. The phenomenon is is known as black-body radiation equilibrium. The amount of energy coming in from the sun can be represented with one mathematical expression and the amount of energy leaving the earth by another. The two must be equal, in an equillibrium.

This is the black-body radiation equilibrium equation.

$(1-a)S\pi r^2 = 4\pi r^2 \epsilon\sigma T^4$

The symbols in this equation have the following meanings:

- S: solar constant

- a: earth albedo
- ϵ: earth emissivity
- σ: Stefan-Boltzmann constant
- r: radius of the earth
- T: earth temperature, in degrees Kelvin

The solar constant S represents the amount of solar radiation (here called "irradiance," per unit area on earch is coming towards the earth.

Todo

verify the def'n of irradiance.

The earth albedo a is the fraction of irradiance that is reflected from the earth.

$0 \leq a \leq 1$

Any light that strikes the ocean will have some of it reflected back into the sky and some of it transmitted (after being refracted, etc.) and eventually turned into heat. The portion reflected is the albedo. The albedo depends on surface materials and atmospheric conditions. High albedo materials include snow and ice, whereas forest canopy and grass have low albedo.

The average value today is approximately $a = 0.3$.

The earth emissivity ϵ is the fraction of available radiation that escapes from the earth.

$0 \leq \epsilon \leq 1$

The average value today is approximately $\epsilon = 0.6$. Increasing concentrations of greenhouse gases in the atmosphere tend to lower ϵ, meaning that less of the earth's heat is emitted, leading to a buildup; the earth's average temperature rises.

It's interesting to consider the effect of clouds on the albedo and emissivity of the earth. Clouds tend to increase the albedo, thus leading to less warming during the day. However, clouds also tend to decrease the emissivity, because heat the would otherwise radiate from the earth is partially reflected back to the earth. This effect happens not only during the day, but at night as well. Cloudy weather thus tends to lower daytime temperatures but raise nighttime temperatures.

A particular type of clouds are the contrails that are created in the wakes of airplanes. After the September 11, 2001 events, nearly all flights over North America were suspended for security reasons. Without the normal contrails from the thousands of flights per day, temperatures over North America were higher than otherwise expected during the day and lower during the night. Thus an artificial activity such as flying lots of airplanes can have an almost immediate effect on the temperature of large areas of the earth's surface.

Let's go back to the black-body radiation equation and work with it, in order to determine how various factors affect the average temperature of the earth. If we solve the equation for T we get the following.

$T = \sqrt[4]{(1-a)S/4\epsilon\sigma}$

Notice that some of the terms in the original equation are gone. Those relating to the surface area of the earth have cancelled each other out. (The earth's radius r is gone, and π is gone.) The resulting version is remarkably simple. From this, given values for the albedo and emissivity of the earth, we can compute the earth's average temperature. With a bit of ordinary arithmetic and a square root of a square root (i.e., a fourth-root operation).

3.2.5 Designing a Set of Operators

By identifying some key phenomena (heating and cooling of the earth) and their relationships (according to the blackbody radiation equilibrium equation), we have taken an important step towards a formulation of the problem. An initial state for a problem formulation could be expressed in terms of values for albedo and emissivity in a given starting year, where those values will then change as a result of applying operators, one of which may simply represent the passage of time.

The wickedness of the global warming problem is not yet tamed, for we have not done much about this set of operators. An operator might represent the initiation of a government program, or an international project, to reduce automobile emissions. The real-world implementation of such a project might require overcoming financial and political impediments that are part of the wickedness of the problem. They are, essentially, the focus of fundamental disagreements among people in the relative costs and benefits of the projects. Thus a possible operator for the formulation might be praised as valuable by one stakeholder while simultaneously being ridiculed by another.

In order to tame the problem, we will "factor out" several wicked aspects from our formulation. The result will be something simpler than the real-world problem, but something possibly still useful as a focus of discussion and negotiation, as well as a means to gain a modicum of understanding of how certain factors may interrelate to have an effect on global warming. To factor out the disagreements, we will posit possible actions and estimate their costs and effects. This might ideally be done with great care, involving experts, polls, etc., but the reality of the result might still be too complicated to serve all the purposes of a problem formulation such as achieving a means to engage potential solvers. The following possible actions are a somewhat arbitrary collection, but the collection is small yet diverse, and yet tied in directly with the blackbody-radiation model we have chosen. Here are the possible actions.

- Invest $200 million in solar energy.
- Invest $80 million in reforestation of former rainforests.
- Invest $50 Million in reducing automobie emission of greenhouse gases.

These programs are certainly hypothetical and could be considered to a form of fiction. We might have to decide if the value of the truth-plus-fiction package we end up with is worth the effort of proposing and promoting it. Before proceeding, it's worth mentioning that today's widespread *gamification* of many activities, including education, is done for some of the same reasons that we accept some amount of fiction in our problem formulation: to better engage learners. Even though the mechanics and narrative of an education game may be fiction, there are real lessons embedded in the game that the designers and adopters deem important.

In order to accommodate such actions as the above three in our problem formulation and still have a model with some degree of integrity, we must introduce some additional state variables (other than albedo and emissivity), because we are dealing with a dynamical (time-dependent) system, where albedo and emissivity cannot instantly change. The first additional variable is of special importance: time. Each state of the world will have associated with a particular time. As discussed later, the times will be limited to a set of years that begins with a starting year and has years every five years after that up until an ending year. We'll use another two variables to represent the rate of change of albedo in the state and the rate of change of emissivity in the state. The operators will exert their effects on future states by incrementing or decrementing these variables. During a simulation step, these variables will be used to update the albedo and emissivity state variables. A final new state variable is "funds remaining." This represents the hypothetical budget of an organization that is empowered to do something about climate change. When investments are made, the funds remaining are decremented. When the simulation is driven forward for the next 5-year period, the funds remaining are incremented with amount that is constant from one cycle to the next, like passing "Go" in Monopoly. The budget aspect of the problem template is partly fiction (in its particulars of amounts, timing, constancy over time) and partly realism (Yes, money does constrain organizations dealing with wicked problems).

With that disclaimer and rationale for our approach to operator design, let us now work out the details of the above actions in terms of our physical model and the made-up financial constraints.

```
<INITIAL STATE>
INITIAL_STATE = CC_State(a=0.3, e=0.612, T=15,
   year=2020, funds=200, n=0)
</INITIAL STATE>
```

Solar Energy Investment

This will end to lessen the need to burn fossil fuels and thus reduce the rate at which greenhouse gases are accumulating in the atmosphere. The effect is to slow the rate at which emissivity is declining and thus to slow the rate at which global warming is progressing. If a large area of the earth were covered by solar energy panels, which have a low albedo, this would tend to lower the average albedo of the earth, causing more heat to accumulate and temperatures to rise. However, the areas of solar panels would have to be huge for this to be a serious concern.

```python
def f_Solar(s):
  print("Investing in Solar energy.")
  s2=copy_state(s)
  global SOLAR_COST
  s2.funds -= SOLAR_COST
  s2.emissivity_change_factor *= 1.003 # This reduces the rate of accumulation of GG
  s2.lastOpDesc = "Inv. in Solar Energy"
  return s2

def f_Reduce_Auto(s):
  print("Reducing Automobile emissions.")
  s2=copy_state(s)
  global REDUCE_AUTO_COST
  s2.funds -= REDUCE_AUTO_COST
  s2.emissivity_change_factor *= 1.001 # This also reduces the rate of accumulation of GG
  s2.lastOpDesc = "Red. Auto. Emissions"
  return s2

def f_Reforest(s):
  print("Investing in reforestation.")
  s2=copy_state(s)
  global REFOREST_COST
  s2.funds -= REFOREST_COST
  s2.emissivity_change_factor *= 1.001 # nice improvement in rate of e, but
                                       # albedo goes down.
  s2.albedo_change_factor *= 0.993 # forests have lower albedo than average.
  s2.lastOpDesc = "Inv. in reforestation"
  return s2

def f_Implement(s):
  print("Implementing selected policies.")
  s2=copy_state(s)
  # Next  update the albedo and emissivity. They must lie in the range [0.0 - 1.0].
  e_new = s2.e * s2.emissivity_change_factor  # Record effects of 5 more years
                                              # of greenhouse gas accum.
  s2.e = min(1.0, e_new)
  a_new = s2.a * s2.albedo_change_factor
  s2.a = min(1.0, a_new)

  s2.year += 5
  s2.updateT()
  global INCOME
  s2.funds += INCOME # Pass Go, collect $100M
  s2.lastOpDesc = "Implement"
  return s2
```

The operators for this formulation are defined as follows.

```
op1 = Operator("Reduce Automobile Emissions "+\
                 "(cost is "+str(REDUCE_AUTO_COST)+"M)",
               lambda s: s.funds>=REDUCE_AUTO_COST,
               f_Reduce_Auto)

op2 = Operator("Invest "+str(SOLAR_COST)+"M in Solar Generation",
               lambda s: s.funds>=SOLAR_COST,
               f_Solar)

op3 = Operator("Invest "+str(REFOREST_COST)+"M in Reforestation",
               lambda s: s.funds>=REFOREST_COST,
               f_Reforest)

op4 = Operator("Implement Selected Policies over 5 years",
               lambda s: True,
               f_Implement)
```

Reforestation

Trees are quite effective at scrubbing carbon dioxide from the atmosphere and "sequestering" its carbon in solid form. They thus tend to slow the accumulation of greenhouse gases and help to increase the emissivity of the earth, letting more heat escape to outer space.

Reducing Automobile Emissions

Automobilies powered by gasoline and diesel engines are a major contributor of greenhouse gases to the atmosphere. Ways to reduce emissions include conversion to hybrids (provided that the electricity needed by hybrids can itself be generated without contributing to greenhouse gases) and electric cars, reducing miles driven, increasing fuel efficiency, ensuring cleanest possible combustion, etc.

Simulating Time Passing

Another aspect of these three kinds of actions is the timeframes for them. How long does it take for a project to have a significant effect? Effects of actions to remediate the accumulation of greenhouse gases will be delayed, if noticeable at all, due to the huge buildup of greenhouse gases already present, and due to the many sources of additional greenhouse gases that cannot be stopped.

The kind of timeframe that is useful in an educational collaborative problem-sovling is one in which (a) the simulated results of a possible plan can be seen after only a few minutes or hours of problem solving, (b) the individual actions have results that, however small, are generally detectable. The typical scenarios discussed in articles on global warming suggest timeframes of one or two human generations: 25 to 50 or perhaps 100 years. If a typical simulation step represents the passing of 5 years, then a 100-year timeframe implies problem solving sessions with root-to-leaf paths involving up to 20 applications of an operator which drives the simulation forward by 5 years. Actual path lengths will be longer, because we will assume that initiating or continuing a project for another 5 years involves an operator application that is distinct from one representing the passing of time. Furthermore, an investment of, say, $50 million in reforestation, does not prohibit another investment of an equal amount at the same time, thus doubling the magnitude of the project during its 5-year period.

While the choice of a 5-year simulation cycle is arbitrary from a scientific modeling perspective, it is natural from a human perspective, as some governments, companies, and other organizations find it convenient to work according to 5-year project schedules. Five years is a time period that is relatively easy to imagine and to work with.

3.2.6 State Visualization

The state visualization for our formulation will meet several objectives:

1. be graphically suggestive of the temperature of the world.
2. present vital statistics of the earth, based on the physics model.
3. be small enough that many states, in a session tree, can be drawn on the screen at once.
5. be visually appealing.

The state visualization is a combination of an image, showing the earth, a frame colored in accordance with the temperature of the earth in that year, and with the following vital statistics shown in a textual caption: the year, the temperature (converted to degrees celsius) for the state, and the albedo and emissivity, and the current values of their rates of change. Also shown is the amount of money still available for investments.

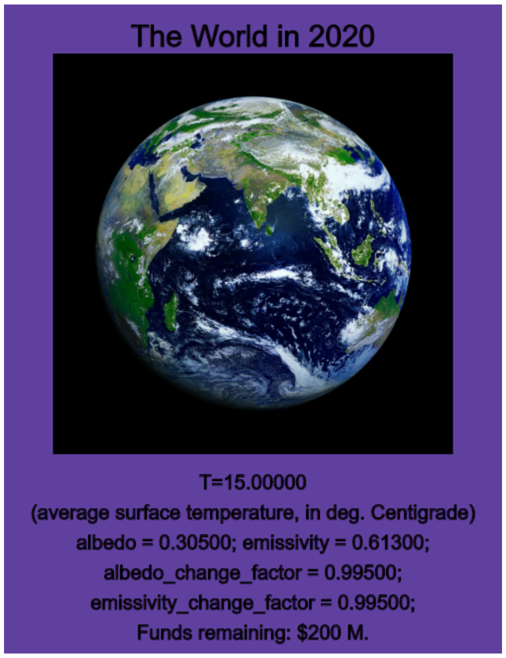

The World in 2020
T=15.00000
(average surface temperature, in deg. Centigrade)
albedo = 0.30500; emissivity = 0.61300;
albedo_change_factor = 0.99500;
emissivity_change_factor = 0.99500;
Funds remaining: $200 M.

Figure 3.1: Climate Conundrum state visualization.

3.2.7 Evaluation of the formulation

Here's a brief critique of the above formulation. Its depth of modeling is shallow, and we can describe it mathematically in as few as three formulas, one expressing the black-body radiation relationship, and the others updating the albedo and emissivity from the change factors. On the other hand, the model captures the macro phenomenon of the earth's average surface temperature using a mathematical formula supported by modern physics.

This formulation incorporates a mix of science and fiction, in the sense that the black-body radiation model mirrors scientific understanding, but the set of operators for changing parameters of the model are made-up by the author of the formulation. The operators have been designed to be relatively easy to understand by anyone playing the game, trying to make it through 50 years without the temperature climbing more than 3 degrees Centigrade. The operators are understandable because they are based on actions that seem understandable: investing money, and in activities we can imagine: building solar energy facilities, cutting down on automobile emissions, and planting new forests. Part of the fiction is that an organization exists that might be willing to make such investments. However, we can point to the

United Nations as an organization that could plausibly be in such a position, even if today's politics seem to make it unlikely to happen.

Does this formulation support critical thinking on the part of users or players? I'd argue that it might. It depends on how the formulation figures into a conversation or a study of the problem. A user who learns what the scientific part of the game is and what the fictional part of the game will be in a better position to discuss much more complex proposals than someone unfamiliar with any models or proposed actions for fixing the problem.

From a playability and learning perspective, this formulation has some notable features.

1. It is difficult to solve.
2. It can be difficult to understand what the effects of an operator are, because the direct, short-term effects are relatively subtle. This is like the real world!

A much deeper model and set of narratives are embodied in the professionally-produced game *Fate of the World* (Red Redemption 2011). This game also involves a combination of science and fiction, but offers many more variables, actions, and narrative outcomes.

3.3 Working with Multiple Formulations of a Wicked Problem

"There is no definitive formulation" is the first of Rittel and Webber's criteria for a wicked problem. An in-depth study of a wicked problem therefore really requires multiple formulations, with an analysis of their relative strengths and weaknesses. Let's now consider some of the issues involved in working with more than one formulation for the same problem.

3.3.1 Problem Posing As A Means to Explore a Topic

First, having multiple formulations allows us to explore the topic that the problem relates to. In traditional problem-solving contexts such as classes in mathematics, the problems are generally *given* rather than formulated by students. In the case of "word problems" in which there is a story around the problem, the student gets to make up the equations that represent the problem, but there is not usually much flexibility there. There is usually one "right" way to represent the problem.

In their book *The Art of Problem Posing," Stephen Brown and Marion Walter argued that when students actually formulate mathematics problems, asking questions and stating their own hypotheses, the learn mathematics in a very different way, and generally more thoroughly, than when they simply solve lots of given problems. They learn to see relationships among problems, and they learn about factors that contribute to the ease with which a problem can be solved. One of the big questions the authors address is where problems come from. In their context, mathematical problems often arise out of curiosity, and the desire to understand. A new idea for a problem can be constructed from any established mathematical statement. The approach is to ask, "What if not?" The possible consequences of supposing differently from a given statement can lead to many new ideas, some of which may have merit on their own.

There is a great potential for a rich interplay between problem posing and problem solving. We discuss this in detail in Chapter 9.

3.3.2 Ill-Formed vs Well-Formed Problems

There are several scales on which we can evaluate problem formulations. One is the extent to which they are "well-formed." David Jonassen makes the distinction between ill-formed and well-formed problems in connection with teaching how to solve complex scientific problems. An ill-formed problem is one whose formulation lacks various components such as a clearly defined state representation, initial state, a set of operators, or goal criteria.

3.3.3 Metaproblem Space

An important concept in formulation of wicked problems is the space of formulations itself, which we call a *metaproblem space*. The idea here is that each possible formulation of a problem can be considered to be an element, like a point, in this space. We can make an analogy between this metaproblem space and our familiar 3D Euclidean space, and so we can have a notion of distance that is great for two problem formulations that are very different, and that is small for a pair of very similar formulations.

Designing a formulation can be considered as moving through the metaproblem space through an iterative design process, arriving at a place in that space that corresponds to a well-formed formulation of the given problem. The path taken through the metaproblem space will typically include parts in which the number of state variables grows and possibly shrinks, and parts in which the operators undergo refinements.

In later chapters we will discuss (a) iterative design itself, as well as (b) evaluating formulations.

3.4 Fusing Formulation and Solving

We have separated formulation and solving to explicate our process, but many problems require the kind of reformulation that requires rapid iteration. So we consider ways to streamline the reformulation and re-solving process. Part of the process of evaluating a formulation is to try to solve the problem as formulated. If turns out to be impossible, that says something important about the formulation. If there are too many goal states, or it is too easy to get to one, that may mean the formulation needs to be made more realistic or detailed.

3.5 Conclusions and Discussion

Problem formulation is necessary when solving using CTPS. It's got several challenges. Restriction and simplification is a big one, in the case of wicked problems. Designing representations, operator sets, visualization, and other solving "handles" is another.

In the next chapter, we approach problem formulation from a different angle: as part of game. From the game perspective, a wicked problem is tamed according to the needs of the game, which may make it easier to deal with some of the complexities of the problem.

After that, in Chapter 5 we'll address the challenge of designing good visualizations, both for the states themselves and for the more abstract aspects of solving such as the space of possible states for a problem. These visualization may be simple plots or they may be interactive computational objects that can be explored by clicking on components, examining substructures, or testing them under various conditions. The human experience of solving or helping to solve a problem is greatly affected by the design of these visualizations, for they can turn out to be the primary interface between the solvers and the rest of the system and team.

3.6 Bibliographical Information

The bibliographic references for this chapter fall into several categories: (a) works on problem solving and problem formulation, (b) works about wicked problems in general, (c) works about specific problems such as global warming, and (d) works about technology that is related to problem formulation, solving, or to empower those who seek to solve difficult problems.

Formulating a problem typically involves different knowledge and skills than does solving a well-structured problem. Work by Goel and Grafman (2000) suggested that solving well-structured problems tends to be more correlated with activity in the left hemisphere of the brain while working with ill-structure problems is associated with right-hemisphere activity.

Jackman and co-authors (2007) detail the process of formulating a problem as one of gradually working through a metaproblem space (MPS), in which the constructions of fragments of a problem representation correspond to moves through the MPS. This metaproblem space is in the mind of the student doing the formulating, although the student is probably unaware of it. The MPS can also be thought of a model for the realm of possible formulations for the problem, limited only by one's intuition about what is likely to be relevant or irrelevant to the problem.

Brown and Walter (2005) are concerned less with the formulation of a problem than with generating the question on which a problem is based. In mathematics, traditionally solving is taught but the process of coming up with problems is not, and their book provides ways to counter that tendency as well as arguments as to why problem posing is important.

Wicked problems and their characteristics were the focus of the paper by Rittel and Webber. Wicked problems are usually not just ill-structured but hampered by social constraints.

Climate change (a.k.a. global warming) is addressed in detail in the reports of the Intergovernmental Panel on Climate Change. After the British Broadcasting Company sponsored the development on an online game called Climate Challenge, the company that developed it, Red Redemption, Ltd., went on to create a full-blown commercial game, Fate of the World. This game uses a computational model with hundreds of parameters, with important parts of the model based on the work of a climate scientist at Oxford University. An early review of Fate of the World was made by J. Arnott in the English newspaper, The Guardian in late 2010.

3.7 References

- Arnott, J. 2010. Fate of the World - review. The Guardian, 31 October 2010. http://http://www.theguardian.com/technology/2010/oct/31/fate-of-the-world-review.
- Brown, S. I. and Walter, M. I. (2005). The Art of Problem Posing, 3rd ed. Mahwah, NJ: Lawrence Erlbaum Associates.
- Drugs.com. (2017). Antibiotic resistance. https://www.drugs.com/article/antibiotic-resistance.html. (accessed March 2017).
- Goel, V., and Grafman, J. (2000). The role of the right prefrontal cortex in ill-structured planning. *Cognitive Neuropsychology*, Vol. 17, pp.415-436.
- Intergovernmental Panel on Climate Change. 2014. Fifth Assessment Report. http://http://www.ipcc.ch/.
- Jackman, J., Ryan, S., Olafsson, S., and dark, V.~J. (2007). Metaproblem Spaces and Problem Structure. In Jonassen, D. (ed.) *Learning to Solve Complex Scientific Problems*. Lawrence Erlbaum and Assoc., pp.247-270.
- Jonassen, D. (ed.) 2007. *Learning to Solve Complex Scientific Problems*. Lawrence Erlbaum and Assoc.
- Lappe, Marc. (1995). *Breakout: The Evolving Threat of Drug-Resistant Disease*. San Francisco, CA: Sierra Club.
- Red Redemption, Ltd. (2011). *Fate of the World*. Distributed by Steam.
- Rittel, H., and Webber, M. 1973. Dilemmas in a General Theory of Planning. *Policy Sciences*, Vol. 4, pp.155-169.
- Study.com. (2017). Weapons proliferation: Concerns and Actions. http://study.com/academy/lesson/weapons-proliferation-concerns-actions.html Accessed March 2017.

3.8 Exercises

1. America is experiencing an opioid overdose epidemic in 2017. Explain how this problem might meet each of the Rittel and Webber ten criteria for being a wicked problem. Are there also stakeholder issues that contribute

to the challenge?

2. Could the problem of protecting a personal computer from malware be considered to be a wicked problem? The second wicked-problem criterion of Rittel and Webber is that there is “no stopping rule.” Does this apply here? Why or why not?
3. Consider the problem of establishing a human colony on Mars. Should this be considered a wicked problem? In what respects might this be a wicked problem?
4. Consider the problem, faced by coastal communities, of rising ocean levels.
   (a) Suggest a possible state representation for an extremely simplified version of this problem. (For example, you can simplify it by dealing with a single community rather than all affected communities.)
   (b) Give the starting state and a goal state based on your representation.
   (c) suggest three or four operators that might be used in order to transform the starting state into the goal state through some sequence of operator applications.
5. As computer systems and robots gradually become more capable and more widespread, a number of jobs currently performed by humans may be lost.
   (a) Is job loss due to automation a wicked problem? Consider at least five of the Rittel and Webber criteria, and describe how the problem either meets or fails each criterion.
   (b) Suggest a state representation, an initial state, a goal state, and a set of operators that could be used to solve or at least to strategize about solving this problem.