This article was downloaded by: [216.73.216.247] On: 14 November 2026, At: 09:15
Publisher: Institute for Operations Research and the Management Sciences (INFORMS)
INFORMS is located in Maryland, USA



INFORMS TutORials in Operations Research

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

Teaching Dynamic Programming and Duality Insights Using Games, Interdiction, and Robust Optimization

J. Cole Smith

To cite this entry: J. Cole Smith. Teaching Dynamic Programming and Duality Insights Using Games, Interdiction, and Robust Optimization. *In* INFORMS TutORials in Operations Research. Published online: 14 Oct 2014; 127-150.
https://doi.org/10.1287/educ.1120.0098

Full terms and conditions of use: https://pubsonline.informs.org/Publications/Librarians-Portal/PubsOnLine-Terms-and-Conditions

This article may be used only for the purposes of research, teaching, and/or private study. Commercial use or systematic downloading (by robots or other automatic processes) is prohibited without explicit Publisher approval, unless otherwise noted. For more information, contact permissions@informs.org.

The Publisher does not warrant or guarantee the article's accuracy, completeness, merchantability, fitness for a particular purpose, or non-infringement. Descriptions of, or references to, products or publications, or inclusion of an advertisement in this article, neither constitutes nor implies a guarantee, endorsement, or support of claims made of that product, publication, or service.

Copyright © 2012, INFORMS

Please scroll down for article—it is on subsequent pages

With 12,500 members from nearly 90 countries, INFORMS is the largest international association of operations research (O.R.) and analytics professionals and students. INFORMS provides unique networking and learning opportunities for individual professionals, and organizations of all types and sizes, to better understand and use O.R. and analytics tools and methods to transform strategic visions and achieve better outcomes.
For more information on INFORMS, its publications, membership, or meetings visit http://www.informs.org

TUTORIALS IN
OPERATIONS RESEARCH
INFORMS 2012

© 2012 INFORMS | ISBN 978-0-9843378-3-5
http://dx.doi.org/10.1287/educ.1120.0098

Teaching Dynamic Programming and Duality Insights Using Games, Interdiction, and Robust Optimization

J. Cole Smith
Department of Industrial and Systems Engineering, University of Florida,
Gainesville, Florida 32611, cole@ise.ufl.edu

Abstract This tutorial discusses two aspects in operations research foundations that have often proven formidable to beginning students: dynamic programming and duality. Although mathematically inclined students may immediately pick up on these very elegant and powerful concepts, the intuition behind these ideas is often unusual and confusing to beginners. This tutorial begins by discussing an effective method for teaching dynamic programming concepts to undergraduate students in an entertaining and accessible manner. This background permits an application-oriented discussion of duality, which can in turn be used to teach emerging topics in interdiction and robust optimization. These topics allow one to touch upon some of the nuances of linear programming duality concepts, such as left- and right-hand shadow prices. The integrated manner in which these topics are presented may allow students to learn the concepts in a more coherent manner and better understand the scope and limitations of modern mathematical programming approaches.

Keywords dynamic programming; duality; interdiction; robust optimization; shortest path

1. Introduction

In discussing optimization concepts with students who recently completed a beginner's course on deterministic operations research (OR), two concepts that consistently rank among the most daunting seem to be duality and (if covered) dynamic programming (DP). Both, in their own ways, tend to regard ideas that are counterintuitive and confusing for students. This tutorial chapter serves two purposes. The first is for instructors to consider additional ideas to embed within their courses, or new ways of presenting existing material. This chapter suggests ideas for making the core ideas behind these concepts natural to the students *before* presenting them in detail. Integrating these concepts allow students to gain a deeper understanding of the material. The second purpose of this tutorial is to serve as an introduction to interdiction and robust optimization for an audience having only a superficial understanding of the topics covered here. Hence, those readers who are not yet comfortable with one or both of the topics may use this chapter to augment their understanding of the topics covered herein.

Virtually all introductory OR courses—with good reason—begin by modeling and solving problems in which all variables are determined simultaneously. Hence, when DP is introduced and taught as a recursive procedure, students are essentially asked to envision a wholly different algorithm. In addition, this approach is often either (a) taught in a completely disparate manner from the preceding OR materials or (b) skipped altogether to avoid such a dramatic shift in the focus of a class. As for duality theory, modern OR textbooks do a wonderful job in explaining economic interpretations of dual variables, for instance. However,

Downloaded from informs.org by [216.73.216.247] on 14 November 2026, at 09:15 . For personal use only, all rights reserved.

the student is often focused on solving a mathematical optimization problem for several months, looking for (primal) variable values and costs, before he or she is confronted with this strange inverse problem that "complements" the original problem. This abrupt change tends to cause a struggle in class. Moreover, the full power of duality theory in solving practical problems is often concealed from students.

This chapter focuses on strategies for improving students' intuition for key DP and duality concepts, and using those concepts naturally to expand course content beyond that traditionally taught in introductory OR classes. We begin in §2 by discussing games that can be cast as dynamic programs, which lend themselves particularly well to teaching core DP concepts. Those concepts can then be extended to cover more complex optimization problems, including those that arise in contemporary research problems. In §3, we cover interdiction problems and modern approaches for formulating interdiction problems, especially via the use of duality. Particular emphasis is paid to interpreting the final interdiction model that is obtained in practical terms, thus enhancing a student's knowledge of duality interpretations and usages in real-world optimization challenges. A similar approach is taken in §4 in the study of robust optimization models. This tutorial closes in §5 with a summary and recommendations for further reading on this topic.

2. Dynamic Programming Games and Challenges

Some segments of class are naturally motivated toward solving production and lot-sizing problems, and others may naturally gravitate toward the solution of binary knapsack problems, making them both very natural illustrations of dynamic programming algorithms. However, a greater percentage of students seems to pay close attention to strategy games, especially those taking place between a class and an instructor taking great pleasure in defeating the class in these games. Section 2.1 focuses on these games and illustrates how they can be used in a "pre-DP" module to motivate recursive algorithms. Section 2.2 examines the study of Dijkstra's algorithm (Ahuja et al. [1], Dijkstra [18]) via dynamic programming and draws a relationship between duals obtained from alternative linear programming formulations of the problem and those labels gleaned from corresponding Dijkstra's implementations. Finally, §2.3 presents a challenge that illustrates the need to judiciously select state variables for dynamic programs, when the states (or stages) are not obvious, as they are in many textbook applications.

2.1. Games

An excellent, and very simple, example of these games is described as follows. A set of 21 sticks exists in a pile, and two players take turns removing sticks from the pile. Each player must take one, two, or three sticks from the pile in his or her turn. Whoever takes the last stick wins the game. (The game can be played starting with any number of sticks, though the eventual winner may change depending on this starting value. Also, the lessons remain the same if the rules dictate that whoever takes the last stick loses the game.) This example can found, for instance, in Winston [35], and is a special case of the famous game of Nim.

The vast majority of students will attack the problem *forward*, much as one might when playing a game like chess. A novice player may think of what happens when the first player (leader) takes one, two, or three sticks: The follower will respond by taking one, two, or three sticks. One may then need to assess the leader's possible reactions, but, already, 27 $(= 3^3)$ variations of this game are being considered. The situation is rendered hopeless at this point, and the student makes a decision based on heuristic criteria. (If the game is presented late in a lecture, the students may choose three sticks to expedite the conclusion of the game!) However, the game becomes very natural when played *backward*. With one, two, or three sticks remaining, it is obvious that the game can be won by simply taking all remaining sticks. But if a player faces a situation with four sticks remaining, regardless of how one got

Downloaded from informs.org by [216.73.216.247] on 14 November 2026, at 09:15 . For personal use only, all rights reserved.

to this point, they cannot win (against a rational, optimally behaving opponent): Any move made puts the opponent in a winning position. Hence, seeing only four sticks remaining is equivalent to losing the game.

From this point, consider what happens if five sticks remain. The player has the option of leaving the opponent with four sticks (which is a losing position), three sticks (a winning position), or two sticks (a winning position). Hence, it is possible to win with five sticks remaining by taking one stick and putting the opponent in a losing position. At this point, students implicitly understand what the "state" is: A snapshot of the game (or more generally, of the optimization problem under study) that contains all relevant information needed to make subsequent decisions. The only state variable required for this game is the number of remaining sticks. By the same logic as above, if there are six or seven sticks remaining, the player can win, and if there are eight sticks remaining, the player loses against a rational opponent. This pattern can then be extracted to see that losing states are exactly those that contain a multiple of four remaining sticks.

This game has limitations in that (a) its objective function is binary (win or lose), (b) one does not explicitly see stages, (c) determining the state variable is rather obvious, and (d) the optimal policy (reduce the opponent to having a multiple of four remaining sticks) is simple. However, it serves the purpose of illustrating the usefulness of attacking problems backward and gets the student thinking about recursion as a method to solve optimization problems. The following variations allow the students to expand on these basic DP notions.

2.1.1. Multiple Heaps (Nim). Now, consider the symmetric game, but with a few modifications. One, there exist several heaps of sticks. For instance, five sticks may be in one heap, four sticks in another heap, and three sticks in another heap. (We refer to this game as $(5,4,3)$.) Each player selects at least one stick in every turn and can choose to remove as many sticks from one heap as they would like. It is not possible, though, to remove sticks from multiple heaps. One game could proceed with player A removing all three sticks from heap 3, followed by player B removing three sticks from heap 2. The game can now be described as $(5,1,0)$. Player A would then want to take four sticks from the first heap: player B is forced to take a single stick from either heap, and player A wins by taking the final stick.

The primary lesson in this game is that multiple states are required to properly execute the dynamic program. A second concept introduced here places a premium on working backward in a disciplined manner. (Additionally, it is useful to track the states in a compact manner. There is no need, for example, to maintain six states for each combination of $(3,2,1)$; one can simply insist that the states that describe the game are listed in nonincreasing order of sticks in a heap.) Hence, the state of a game is the number of sticks present in each heap. Initially, we can start with a trivial losing state of $(0,0,0)$, from which it is clear that $(k,0,0)$ is a winning state, for each possible value $k=1,\ldots,5$. One can work backward and determine that, e.g., $(1,1,0)$ is a losing state (the only legal move puts the opponent into a winning state). It is then possible to determine several other winning states, which are those states from which it is possible to force the opponent into the losing state of $(1,1,0)$. By this logic, $(1,1,1)$ and $(k,1,0)$ are both winning states, for all $k=2,\ldots,5$.

The key point at this phase is that a dynamic program must proceed in a manner where some state (a_1,a_2,a_3) is examined such that every possible move from this state puts the opponent in a state that has been previously examined by the algorithm (and such that we know whether or not those states are winning or losing states). Several such sequences follow this pattern. Perhaps the simplest method examines all states starting with $(a_1,a_2,a_3)=(0,0,0)$. The next state examined is (a_1+1,a_2,a_3), unless $a_1=5$. In that case, if $a_2<4$, then the next state examined is (a_2+1,a_2+1,a_3), and if $a_2=4$, then the next state examined is (a_3+1,a_3+1,a_3+1). This pattern continues until state $(5,4,3)$ is examined. Most states are winning states, and so it suffices to list only the losing states. Table 1 lists all nontrivial losing states.

Downloaded from informs.org by [216.73.216.247] on 14 November 2026, at 09:15 . For personal use only, all rights reserved.

Table 1. Losing states in the $(5,4,3)$ Nim game.

$(1,1,0)$	$(2,2,0)$
$(3,3,0)$	$(4,4,0)$
$(3,2,1)$	$(5,4,1)$

A similar example can be used in the case for which the player who takes the last stick loses; all of the states in Table 1 remain losing states except for $(1,1,0)$; states $(1,0,0)$ and $(1,1,1)$ become losing states instead. In either game, the player going first can win by first removing two sticks from the last heap.

2.1.2. Asymmetric Players. Consider now the same one-heap game as discussed originally, starting with n sticks in a single heap with the player taking the last stick the winner. In this variant, player A must take between one and k_{A} sticks, and player B must take between one and k_{B} sticks, where $k_{\mathrm{A}} < k_{\mathrm{B}}$. Clearly, player A is at a disadvantage, but to what extent? For what values of n does player A lose no matter which player goes first? Again, the concept being tested in this game is one of state variables. As would be clear when the students attempt to formulate a strategy for this game, it is important to know not only how many sticks are remaining, but whose turn it is in the game.

To illustrate, it is useful to define a win/loss function, which is a precursor to dynamic programming value functions and the recursions commonly used to define them. Let $f_i(s)$ equal W if player i can win on his or her turn when s sticks remain, and L otherwise, when the player would lose to a rational opponent. Initially, for $i = \mathrm{A}, \mathrm{B}$, it is clear that $f_i(s) = W$ for $s = 1, \ldots, k_i$. A good question to ask next, before proceeding to work out an example, is the following: If $k_{\mathrm{B}} = 3$ and $s = 12$, then how should player B make its next decision? (Which states should player B consult to determine how many sticks he or she should take?) The answer would be that player B examines $f_{\mathrm{A}}(11), f_{\mathrm{A}}(10), f_{\mathrm{A}}(9)$, noting that a choice of one, two, or three sticks leaves its opponent in those states, respectively. Player B takes $k^\star$ sticks, where $f_{\mathrm{A}}(12-k^*) = L$, if such a $k^\star$ exists. In that case, $f_{\mathrm{B}}(12) = W$ (player B can indeed win). If no $k^\star$ exists, then $f_{\mathrm{B}}(12) = L$. The point should now be made that, after initialization, states $f_{\mathrm{A}}(s)$ and $f_{\mathrm{B}}(s)$ are made in increasing order of s (with memory of what decisions should be made in each "winning" state).

Interestingly, a pattern emerges quickly in this game. Because $k_{\mathrm{A}} < k_{\mathrm{B}}$, $f_{\mathrm{A}}(k_{\mathrm{A}}+1)$ is determined to be a losing state for A, because $f_{\mathrm{B}}(s) = W$ for $s = 1, \ldots, k_{\mathrm{B}}$ (i.e., player B would win regardless of player A's action). Also, we must have $f_{\mathrm{B}}(k_{\mathrm{A}}+1) = W$ at initialization, because $k_{\mathrm{B}} \geq k_{\mathrm{A}}+1$ implies that B can take all $k_{\mathrm{A}}+1$ sticks to win the game. Now consider $f_{\mathrm{A}}(k_{\mathrm{A}}+2)$. All states enumerated thus far for player B have been winning states, and so player A would lose regardless of action; hence, $f_{\mathrm{A}}(k_{\mathrm{A}}+2) = L$. But $f_{\mathrm{B}}(k_{\mathrm{A}}+2) = W$ necessarily, because even if $k_{\mathrm{B}} < k_{\mathrm{A}}+2$, taking one stick leaves player A with $k_{\mathrm{A}}+1$ sticks, forcing player A to lose (because $f_{\mathrm{A}}(k_{\mathrm{A}}+1) = L$) and player B to win. At this point, the instructor may use induction (if the students will not protest too stringently) or simply work out a few more stages of this game to show that $f_{\mathrm{A}}(k_{\mathrm{A}}+t) = L$ and $f_{\mathrm{B}}(k_{\mathrm{A}}+t) = W$ for any positive integer t: Player A always puts player B in a winning state (all states are winning states for B), and player B can always put player A in a losing state by taking a single stick, until it has no more than k_{B} sticks remaining, at which point player B takes all remaining sticks.

2.1.3. More than Two Asymmetric Players. A fun and complex variant of the previous game involves several players, each of whom is allowed to take a different number of sticks. When a particular player cannot win, he or she favors another player and puts that player in a position to win. For instance, suppose that there are three players starting with 12 sticks (in one heap). Player A can take one or two sticks; player B can take one, two, or three sticks; and player C can take one, two, three, or four sticks in each turn. If player A

Downloaded from informs.org by [216.73.216.247] on 14 November 2026, at 09:15 . For personal use only, all rights reserved.

Downloaded from informs.org by [216.73.216.247] on 14 November 2026, at 09:15 . For personal use only, all rights reserved.

TABLE 2. Values for $f_i(s)$ in the three-player asymmetric sticks game, with the number of sticks to be taken given in parentheses.

Sticks (s)	Player A	Player B	Player C
1	A (1)	B (1)	C (1)
2	A (2)	B (2)	C (2)
3	B (1)	B (3)	C (3)
4	B (1)	C (1)	C (4)
5	B (2)	C (1)	A (3)
6	C (1)	A (1)	A (4)
7	A (1)	A (1)	C (1)
8	A (1)	A (2)	C (2)
9	A (1)	A (3)	C (3)
10	A (1)	C (1)	C (4)
11	A (2)	C (1)	A (1)
12	C (1)	A (1)	A (1)

cannot win, he or she attempts to put player B in a position in which they could win (if possible). If player B (or player C) cannot win, he or she attempts to put player A in a position in which they could win (if possible). An interesting question for player A is the following: Does the advantage that players B and C favor player A over each other (if they cannot win themselves) overcome the disadvantage of being able to remove only one or two sticks at each turn?

Similar to the two-player asymmetric game, the state space includes the player and the number of sticks remaining. However, in this case, the value function $f_i(s)$ needs to reflect which player will win the game, rather than simply whether or not the player i will win. Here, we simply let $f_i(s)$ equal the letter (the identity) of the player who will win the game. For instance, $f_{\mathrm{A}}(s) = \mathrm{A}$ for $s = 1, 2$, $f_{\mathrm{B}}(s) = \mathrm{B}$ for $s = 1, 2, 3$, and $f_{\mathrm{C}}(s) = \mathrm{C}$ for $s = 1, 2, 3, 4$. The remaining value functions for each player/state combination are evaluated in increasing order of s. For $s = 3, 4$, player A must let player B win, and for $s = 4$, player B must let player C win. When $s = 5$, player A chooses who will win between players B and C, and takes two sticks to let player B win. Player B must let player C win when $s = 5$. Player C chooses to let player A win instead of player B when $s = 5$ by taking three (or four) sticks.

Interestingly, the implication is that $f_{\mathrm{B}}(6) = \mathrm{A}$ (player B prefers players A to C), and therefore player A can win when he or she has seven sticks remaining: Take one stick, and leave player B with no way to win the game. Player B cannot win with six sticks remaining, but can take one stick to ensure that player C cannot win. Player C, choosing between players A and B, takes three (or four) sticks to allow player A to win. A summary of this game is given in Table 2. Educationally, this game forces students to remain disciplined when computing each value function for each state. In computing $f_i(s)$, they must examine all actions that the next player (say, j) may make and force player j to make the game end as well as possible for player i. (This game repeats states 6–11 in states 12–17, and every six states thereafter.) Note that player B cannot win if the game starts with six or more sticks.

2.2. Shortest Path by DP/Duality

Although the sticks games in §2.1 teach the concepts of recursion and state variables, students are unlikely to have a satisfactory understanding of value functions that represent costs, or recursive functions that define them. These ideas can be easily embedded in the solution of shortest path problems having nonnegative costs. The notation for the shortest path problem is given as follows. Let $N = \{1, \ldots, n\}$ be the set of nodes, where node 1 serves as the origin and node n serves as the destination. Let A be the set of arcs in the problem, where c_{ij} is the cost of using arc $(i, j) \in A$ in the path. The *forward star* of node i, FS(i), is

the set of arcs that leave node i, i.e., $\mathrm{FS}(i) = \{j \in N\colon (i,j) \in A\}$. The *reverse star* of node i is the set of nodes that enter node i, i.e., $\mathrm{RS}(i) = \{h \in N\colon (h,i) \in A\}$ for all $i \in N$. Defining $x_{ij} = 1$ if the path uses arc $(i,j) \in A$, and 0 otherwise, the mathematical programming formulation that we use is given by

$$\min \quad \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{1a}$$

$$\text{s.t.} \quad \sum_{j\in \mathrm{FS}(i)} x_{ij} - \sum_{h\in \mathrm{RS}(i)} x_{hi} = b_i \quad \forall\, i \in N, \tag{1b}$$

$$\mathbf{x} \geq \mathbf{0}, \tag{1c}$$

where $b_1 = 1$, $b_n = -1$, and $b_i = 0$ for all other $i \in N\backslash\{1,n\}$. (Note that all vectors are in bold font.) Constraints (1b) in particular state that the net flow out of node i (flow out of node i minus flow in to node i) must equal 1 when $i = 1$, -1 when $i = n$, and 0 for all other nodes.

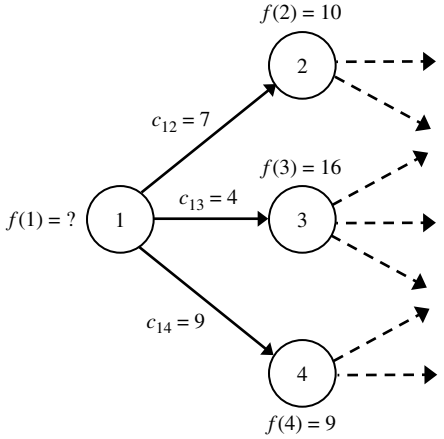
One can either proceed with a forward or backward recursion. Assume for now that backward recursion is being used, and denote $f(i)$ as the shortest path cost from node i to the destination node. An effective way of introducing the concept is to present a situation similar to that shown in Figure 1. Here, the goal is to compute $f(1)$. If $f(2)$, $f(3)$, and $f(4)$ are already known (and we assume that $\mathrm{FS}(1) = \{2,3,4\}$), then $f(1) = \min\{c_{12} + f(2), c_{13} + f(3), c_{14} + f(4)\}$. In the example of Figure 1, we have that $f(1) = 17$ and that any shortest path starting at node 1 traverses through node 2. The concept is that even with the rest of the network hidden from view, if $f(i)$ is known for $i = 2,3,4$, determining which way to go from node 1 is straightforward.

The emphasis in this section is less on the instruction of Dijkstra's algorithm and more on its use in presenting the value function recursion that is so common in dynamic programming. After having established that $f(n) = 0$ and

$$f(i) = \min_{j\in \mathrm{FS}(i)} \{c_{ij} + f(j)\} \quad \forall\, i \in N\backslash\{n\}, \tag{2}$$

in general, the only remaining challenge is to determine the order in which the $f(i)$ can be determined. One suggestion is to present acyclic networks first, which do not require the use of permanent and temporary labels and which follow the simple dynamic programming rubric given in the sticks games in §2.1; that is, one can order the nodes (again using the backward dynamic programming approach) in a list L in an order such that for every $(i,j) \in A$ we have that j appears in L before i.

FIGURE 1. Shortest path recursion example.



Downloaded from informs.org by [216.73.216.247] on 14 November 2026, at 09:15 . For personal use only, all rights reserved.

Alternatively, the same procedure can be run in a forward manner, in which $f(i)$ is the shortest path cost from node 1 to node i, for $i \in N$. Denoting the duals in this case as $f'(i)$, (2) is revised as

$$f'(i) = \min_{h \in \mathrm{RS}(i)} \{c_{hi} + f'(h)\} \quad \forall\, i \in N \backslash \{1\}, \tag{3}$$

with $f'(1) = 0$. Each $f'(i)$ would then be computed by examining the nodes in a list L', ordered such that for every $(i, j) \in A$, i appears in L before j.

Remark 1. As a side note, reintroducing cyclic networks (still with all nonnegative costs) at this point tends to force students to think about how the dynamic program would have to be adjusted, because no *topologically ordered* list L as described above will now exist. After struggling with this issue for a few minutes, the insight offered in Dijkstra's algorithm that the smallest "temporary" label must be correct tends to make a bigger impact on the students. For classes that are inclined to delve into simple complexity issues, it is possible to introduce why the shortest path problem is slightly easier to solve on acyclic graphs than on cyclic graphs at this point.

A more interesting note, though, connects the optimal labels used in the shortest path algorithm to sets of optimal dual variable values for (1). Associate dual variables π_i with the flow-balance constraints (1b). The primal formulation (again assuming nonnegative costs) in fact has one too many constraints, because the flow-balance constraints are linearly dependent. The constraint that is removed affects the optimal dual variable values; how and why this happens is an enlightening lesson for students, and is also important to gaining insights to the interdiction and robust optimization models discussed in §§3 and 4.

First, suppose that the last flow-balance (for $i = n$) constraint is removed. The dual to (1) is given by

$$\max \ \ \pi_1 \tag{4a}$$

$$\text{s.t.} \ \ \pi_i - \pi_j \le c_{ij} \quad \forall\, (i, j) \in A, \tag{4b}$$

where the π variables are unrestricted. Writing (4b), for each $i \in N$, as

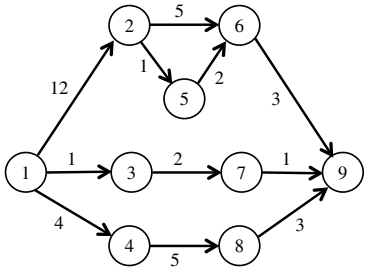
$$\pi_i \le c_{ij} + \pi_j \quad \forall\, j \in \mathrm{FS}(i), \tag{5}$$

along with the observation that at optimality π_i will take its largest possible value, establishes the equivalency of $f(i)$ and π_i, and of (2) and (5). This establishes that the dual variable values match exactly the labels from the DP (or Dijkstra's) algorithm traditionally given for the shortest path problem. A similar exercise demonstrates that labels obtained from forward recursion match those obtained by taking the dual of (1) in which the flow-balance constraint is omitted, and in which the variables are negated so that its dual seeks to maximize π_n subject to $\pi_j - \pi_i \le c_{ij}$, for all $(i, j) \in A$.

However, the shortest path problem affords the opportunity to reinforce the concept of dual interpretations, and to demonstrate the difference between left-hand shadow prices (LHSPs) and right-hand shadow prices (RHSPs). Suppose that we continue to assume that the last flow-balance constraint is removed from the problem, and recall that a *right-hand* shadow price for a constraint gives the rate of change in the optimal objective function value when the right-hand side of the constraint increases. If one continues to treat the duals π_i as the shortest path cost from node i to node n, then there exists a very natural interpretation of π_i as the right-hand shadow price for flow-balance constraint i. Initially, that $b_1 = 1$ and $b_n = -1$ indicates that one entity exists at node 1, which seeks to terminate at node n. If the right-hand side of constraint $i \in N \backslash \{1\}$ is increased by some $\Delta > 0$ (say, $\Delta = 1$ for simplicity, although in fact any positive Δ is valid in this case), then there now exists an additional "traveler" at node i. The key concept is that the absence of the flow-balance constraint for node n implies that this additional traveler must terminate at node n. Hence, the problem

Downloaded from informs.org by [216.73.216.247] on 14 November 2026, at 09:15 . For personal use only, all rights reserved.

FIGURE 2. Varying shadow prices for shortest path problem.



i	LHSP(i)	RHSP(i)
1	4	4
2	−8	6
3	3	3
4	0	8
5	−9	5
6	−11	3
7	1	1
8	−5	3
9	0	0

is the same as before, but one must additionally transport some traveler from node i to node n, which is done at a cost of $\pi_i = f(i)$.

A natural question then follows: What if one seeks the left-hand shadow price of a constraint? Do these left-hand shadow prices correspond to an optimal dual solution to the shortest path problem? In this case, it is instructive to follow the previous analysis in reverse order. First, what would the left-hand shadow price imply about the change in the model? If the right-hand side of constraint i *decreases* by one, then this implies that we have simultaneously changed b_n to equal 0, as implied by the summation of flow-balance constraints over $i = 1, \ldots, n-1$. As a result, decreasing the right-hand side of constraint i by one effectively revises the destination node of the problem from node n to node i. Thus, the left-hand shadow price for constraint i is simply given by the shortest path cost from node 1 to node n, minus the shortest path cost from node 1 to node i. It is then possible for the students to verify that (a) the left-hand shadow prices can be used as optimal values for π, and (b) for each node i on a shortest path from node 1 to node n, the right-hand shadow price for node i's flow-balance constraint matches the left-hand shadow price for node i.

Figure 2 provides an example of the left-hand and right-hand shadow prices (denoted LHSP and RHSP, respectively). Note that a more advanced class might additionally ask the question of whether or not a mixture of left-hand and right-hand shadow prices could exist in a dual optimal solution. The answer is yes. For instance,

$$\boldsymbol{\pi} = (4, -8, 3, 0, 5, 3, 1, 3, 0)$$

is an optimal dual solution. However, setting $\boldsymbol{\pi}$ equal to *any* combination of left-hand and right-hand shadow prices does not necessarily yield a dual feasible solution. One such infeasible solution would be given by setting $\pi_4 = 8$ (its right-hand shadow price) and $\pi_8 = -5$ (its left-hand shadow price), because the dual constraint $\pi_4 \leq 5 + \pi_8$ would be violated. For the example given in Figure 2, there are 12 optimal extreme point dual solutions, which is less than the 32 combinations of right-hand and left-hand shadow prices for nodes 2, 4, 5, 6, and 8.

2.3. DP for Network Interdiction

In the previous examples, students will be challenged to understand recursion functions and to incorporate as many state variables in their recursion as necessary. These lessons lead in naturally to the more traditional DP examples for solving combinatorial problems in pseudopolynomial time, such as knapsack and capacitated lot-sizing problems (CLSPs). (Demonstrating the DP recursion for the capacitated lot-sizing problem will also help to augment the interdiction discussion in §3.2 (see Remark 3).) Although most DP modules stop at this point, it is useful to demonstrate one further example that forces students to think carefully about what state variables one might use in a more complex DP.

The example presented in this subsection can motivated as a quarantine problem. Consider an undirected graph $G(N, E)$, where $N = \{1, \ldots, n\}$, and examine the goal of containing

Downloaded from informs.org by [216.73.216.247] on 14 November 2026, at 09:15 . For personal use only, all rights reserved.

an outbreak of some kind (e.g., an actual contagion, a computer virus, or the detection of some intruder that may seek to escape capture in a large network). Assigning a checkpoint to a node stops the spread of contamination through that node, and hence one can equate placing a checkpoint on that node to deleting the node itself.

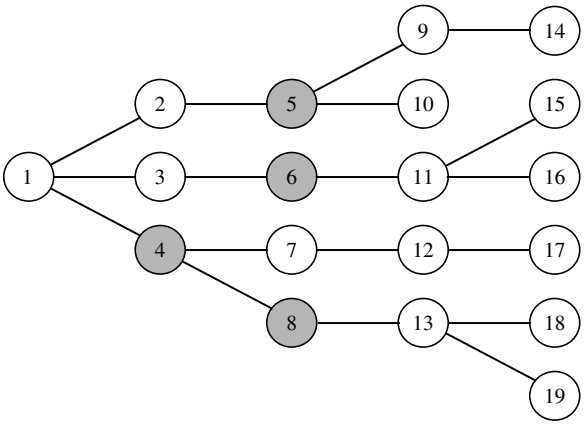
The problem at hand is to determine the fewest number of checkpoints to place on nodes (i.e., the fewest number of nodes to delete) so that any contamination is limited to no more than some τ nodes. It is assumed that a deleted node cannot be contaminated, and that a contamination spreads along graph edges until blocked by a deleted node. Therefore, we require that after all nodes are deleted, the largest remaining component in the graph is no more than τ nodes. (A *component* is a subgraph for which every pair of nodes in the subgraph is connected by a path consisting of edges in E.) For students interested in further reading on this problem, see Shen and Smith [25, 26] for a full treatment of this and several related problems. In a more general scheme, this node deletion problem belongs to a broader study of critical node problems (Grötschel et al. [21], Shier [27]). See also the work of Albert et al. [2], who examine robustness of complex graphs.

An example instance of this problem is given in Figure 3, where the goal is to delete the fewest number of nodes so that the largest possible contamination is no more than $\tau = 3$. The optimal solution depicted in that figure contains four deleted nodes. In general, the node deletion problem is strongly NP-hard (Yannakakis [37]), but certain special subgraphs can be solved in polynomial time via dynamic programming. This subsection focuses on the solution to this node deletion problem on tree graphs (connected graphs having no cycles), such as the one depicted in Figure 3.

The difficulty in prescribing an optimal DP on trees is that there is no obvious starting point—no "beginning" or "end" to the tree. But if pressed, most students will realize that the algorithm needs to proceed either from the root node to the leaf nodes of the tree, or vice versa. To build intuition, students may see that although they have no sense of whether nodes at or near the root should be deleted, they never need to delete a leaf node (assuming that $\tau \geq 1$). Hence, the "bottom" of the tree, near the leaf nodes, is a natural starting point.

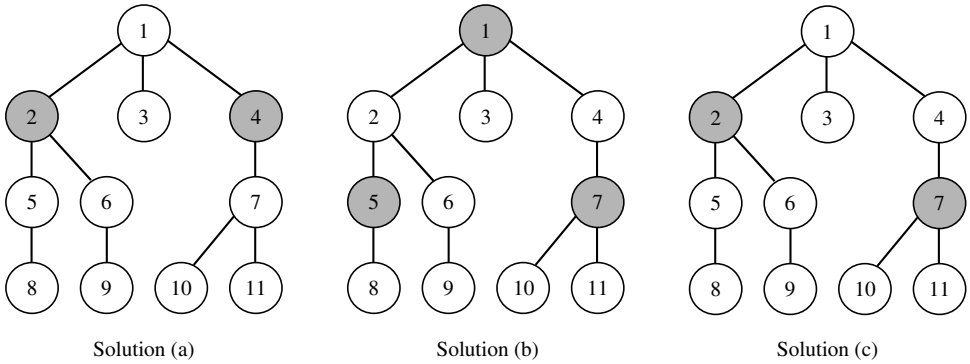
The overall paradigm is therefore to determine "promising solutions" to subtrees rooted at certain nodes and then merge those subtrees up the tree. A promising solution is one that might be a part of some optimal solution to the entire problem. All other solutions either cannot be a part of an optimal solution or, if they are, could equivalently be substituted by a promising solution already enumerated. The process terminates at the root node of the entire tree, at which point an optimal solution can be recovered. For instance, in Figure 3, one might enumerate all promising solutions to the subtree rooted at node 7 (i.e., the tree induced by nodes 7, 12, and 17) and all promising solutions to the subtree rooted at node 8

FIGURE 3. Example node deletion instance with $\tau = 3$, where deleted nodes are shown in gray.



Downloaded from informs.org by [216.73.216.247] on 14 November 2026, at 09:15 . For personal use only, all rights reserved.

FIGURE 4. Three subtree solutions to the node deletion problem with $\tau = 3$, where nodes to be deleted are shaded gray.

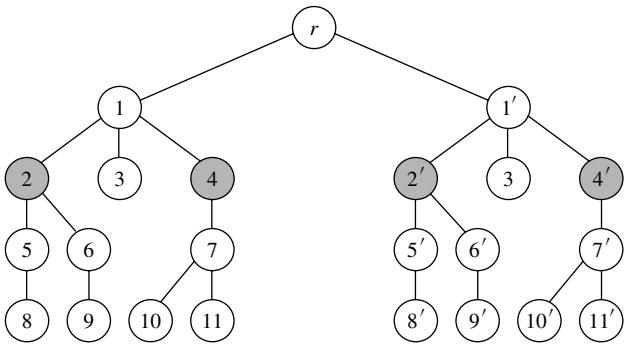


(i.e., the tree induced by nodes 8, 13, 18, and 19). Those solutions could then be merged along with node 4, which serves as the parent node to 7 and 8. As a result, all promising solutions to the subtree rooted at node 4 would now be obtained.

A key step therefore lies in determining what consists of a promising solution for a subtree. To aid in this discussion, consider the three solutions presented in Figure 4 for some subtree rooted at node 1, where node 1 is a child of some other node in the tree (i.e., node 1 is not the root of the entire tree). It is convenient to define the *open set* of a subtree as the set of nodes belonging to the same component as the subtree's root node; e.g., the open set of solution (a) is $\{1,3\}$, that of solution (b) is $\varnothing$, and that of solution (c) is $\{1,3,4\}$. All three solutions in Figure 4 are feasible, because the largest component remaining after the deletion of gray nodes is of size three in all solutions. In fact, if nodes 1–11 compose the *entire* tree's node set, then solution (a) is optimal: It has an objective of two, and there is no solution that deletes only a single node without leaving a component of size larger than three. Otherwise, solution (a) is simply one promising solution, but may not be part of an optimal solution to the node deletion problem.

The fact that deleting nodes 2 and 4 is possibly not optimal may not be obvious to students tackling this problem, and so to motivate why this might be the case, consider what would happen if two copies of this network exist (one having nodes $1,\ldots,11$, and the other having nodes $1',\ldots,11'$), joined by a root node r. Figure 5 illustrates this graph. Suppose that r is not deleted, and that both copies of the network use the solution in which nodes 2 and 4 ($2'$ and $4'$) are deleted. In this case, nodes 1, 3, $1'$, $3'$, and r form a component of size five. Therefore, there may be an advantage to considering solutions in which the subtree

FIGURE 5. Infeasible solution stemming from the use of two subtree solutions having a nonempty open set.



Downloaded from informs.org by [216.73.216.247] on 14 November 2026, at 09:15 . For personal use only, all rights reserved.

root node (e.g., 1, or 1′) is deleted, even if it results in an extra deletion (and thus in an increase of one in the objective function). Hence, solution (b) is promising as well.

Now, consider solution (c). Both solutions (a) and (c) involve the deletion of two nodes, but the subtree root (node 1) in solution (a) belongs to a component of size two, whereas the subtree root in solution (c) belongs to a component of size three. Hence, consider the situation in which nodes $1,\ldots,11$ form a subgraph, and that some node r serves as the parent of node 1. If node r is not deleted, then appending node r to solution (c) creates an open set of size four, whereas appending node r to solution (a) creates an open set size of only three. Intuitively, solution (a) is at least as good as solution (c); this can be quickly shown by proving that if solution (c) is part of an optimal solution to the entire tree, then solution (a) must be as well. Therefore, a feasible subtree solution need not be designated as promising unless it uses fewer node deletions than another promising solution, or has a smaller open set size than another promising solution.

The work done to this point has been to establish the direction in which the DP recursion should work and which state variable must be retained (the open set size). The DP can then be solved in stages from the leaf nodes to the root. Denote $f_i(o_i)$ as the fewest number of deletions required on the tree rooted at node i that generates an open set of size o_i, where $o_i \le \tau$, and yields a largest component size of no more than τ. Initially, each leaf node i can either be deleted (so that $f_i(0)=1$) or not ($f_i(1)=0$); if the students have determined that leaf nodes never need to be deleted, then only $f_i(1)=0$ is necessary to retain as part of an optimal solution.

For nonleaf nodes, the recursions are based on whether or not node i is deleted. Let S_i be the set of child nodes for node i. If $o_i = 0$, then

$$f_i(0) = \min \sum_{v\in S_i} f_v(o_v) + 1. \tag{6}$$

Here, the term $\sum_{v\in S_i} f_v(o_v)$ includes the number of nodes deleted in all subtrees of i, and the "+ 1" term accounts for the deletion of node i itself (which led to the empty open set). Note that there are no restrictions on the open set sizes of the child trees that are merged with node i: Because the open set sizes of the child subtrees are not more than τ, and those open sets do not merge when i is deleted, their sizes are irrelevant.

The recursion for the case in which $0 < o_i \le \tau$ is similar, but accounts for the fact that child subtree open sets do merge when node i is not deleted.

$$f_i(o_i) = \min \sum_{v\in S_i} f_v(o_v) \tag{7a}$$

$$\text{s.t. } o_i = \sum_{v\in S_i} o_v + 1 \le \tau. \tag{7b}$$

The objective (7a) is identical to (6), without the deletion of node i. Constraint (7b) states that the open set size at node i will be the sum of child tree open set sizes plus one, because all open sets merge with node i to form a new open set for the subtree rooted at i.

Several implementation issues remain, which are of varying importances depending on the course being taught. For one, identifying an optimal solution once the recursion reaches the root node, r, must be done by backtracking. The minimum of $f_r(s)$ over all s is the optimal objective function value. If this minimum value occurs at $s=0$, then node r is deleted; otherwise, it is not. In either case, the choices for o_v that led to the optimal value, for each $v \in S_r$, must be retained to backtrack and identify the full solution.

Next, whereas (6) is a straightforward computation, computing $f_i(o_i)$ via (7) is more complex. Ideally, this would be performed by merging the first subtree with the root node first. Then, the second subtree would be merged with the combined solutions to node i and its first subtree. Hence, in this manner, one subtree at a time is merged with its parent, which

Downloaded from informs.org by [216.73.216.247] on 14 November 2026, at 09:15 . For personal use only, all rights reserved.

ensures the polynomiality of the algorithm. Note also that after the merging operations, at most two solutions need to be retained: $f_i(0)$ and (if it exists) the smallest o_i such that $f_i(o_i) = f_i(0) - 1$.

Remark 2. To tie this exercise back to the study of duality, one could easily modify (6) and (7) to include τ as a *state variable* rather than a parameter. The idea would be to define $f_i(o_i, \tau)$ as the fewest number of nodes to be deleted so that the open set size of the subtree rooted at node i is o_i and no component size exceeds τ. At the end of the procedure, it is then possible to determine the fewest number of nodes to be deleted to restrict the largest component size to be no more than τ, for each $\tau = 1, \ldots, n$. This function is neither nonconvex nor nonconcave (in general), as opposed to the convex cost function when the right-hand side of a minimization problem is treated as a parameter. This behavior may be appropriate in comparing parametric combinatorial optimization problems to parametric linear programming problems.

3. Interdiction Models and Insights

Interdiction models often analyze scenarios that involve two competing entities, where one entity seeks to maximize the minimum cost incurred by the other (or, e.g., minimize the maximum profit obtained by the other agent). These problems are often cast over networks, although the underlying mathematical optimization principles often do not require the presence of network structures. These problems again afford the opportunity for students to solve puzzles in a "game" format in class and challenge them to integrate their knowledge of mathematical modeling, network flows, and duality to correctly formulate these problems. In the case of two-stage problems, on which we will focus here, an "attacker" acts first to remove some critical resources from the other player or to affect the other player's objective. The other player, which we call the "defender," then optimizes its objective function given the resources remaining after the attacker's action. A general statement of these problems is given as

$$\min_{\mathbf{x} \in X} \max \quad f(\mathbf{x})^\top \mathbf{y} \tag{8a}$$

$$\text{s.t.} \quad D\mathbf{y} \le h(\mathbf{x}), \tag{8b}$$

$$\mathbf{y} \ge \mathbf{0}, \tag{8c}$$

where $\mathbf{x}$ ($\mathbf{y}$) represents the vector of attacker (defender) actions, $f(\mathbf{x})$ is the defender's objective function (which is itself a function of the attacker's action), $h(\mathbf{x})$ is the available resource vector for the defender (also depending on the attacker's action), and D is a constraint coefficient matrix that restricts the defender's decisions. (Note that all vectors used here are column-vectors, and $\top$ denotes the transpose operator.) This particular model is also known as a Stackelberg [34] (leader–follower) game. Some settings also regard the case in which the defender is able to fortify some of its resources in a preliminary stage to mitigate the maximum amount of damage that the attacker can inflict.

Section 3.1 shows how some formulations can be modeled using ad hoc constraints designed to capture the defender's behavior. In particular, we revisit the node deletion problem presented in §2.3 on a general graph and consider the interdiction of a shortest path problem using insights from the duality concepts presented in §2.2. The dual interpretation of the shortest path interdiction problem motivates a general approach for formulating these problems using duality in §3.2.

3.1. Interdiction Using Ad Hoc Constraints

As a mixed-integer programming (MIP) modeling exercise, it is possible in certain cases to naturally formulate interdiction models as a single mixed-integer program. We consider three such examples in this subsection: facility interdiction, node deletion, and shortest path interdiction.

Downloaded from informs.org by [216.73.216.247] on 14 November 2026, at 09:15 . For personal use only, all rights reserved.

Downloaded from informs.org by [216.73.216.247] on 14 November 2026, at 09:15 . For personal use only, all rights reserved.

3.1.1. Facility Interdiction. Consider the following facility interdiction problem, originally studied by Church et al. [16]. There exists a set F of facilities, which serve a set N of demand points. Each demand point is assigned to its closest facility (e.g., as customers may select a closest store). Whereas each demand point must be assigned to exactly one facility, a facility can be assigned to any number of demand points. The defender's goal is to minimize the sum of weighted distances from each demand point to its assigned facility.

The interdiction problem at hand is to remove a subset $G \subset F$ of facilities, where $|G|$ is restricted to be no more than a given integer r, so that the defender's optimal objective function value is as large as possible. Define the following data for this facility interdiction problem (in addition to F, N, and r): Let a_i denote the weight of node $i \in N$, and let d_{ij} denote the distance between node $i \in N$ and facility $j \in F$. A reasonable way to get students to address this problem is to start by asking what the decision variables are for the attacker and what they are for the defender. The attacker must determine which facilities to delete, and hence binary variables x_j, for all $j \in F$, equal 1 if the attacker removes facility $i \in F$, and 0 otherwise. The defender must determine which facilities to assign to which nodes, and so binary variables y_{ij}, for each $i \in N$ and $j \in F$, equal to 1 if and only if node i is assigned to facility j.

Students can then begin approaching this problem from the *attacker's* perspective. The objective of the attacker is to maximize total assignment cost, i.e.,

$$\max \sum_{i \in N} \sum_{j \in F} a_i d_{ij} y_{ij}. \tag{9a}$$

The attacker's constraints require that no more than r facilities are removed, and the defender is constrained to assign every node to one facility, as required by the following constraints:

$$\sum_{j \in F} x_j = r, \tag{9b}$$

$$\sum_{j \in F} y_{ij} = 1 \quad \forall i \in N. \tag{9c}$$

However, a formulation consisting only of these constraints (and bounds on the x and y variables) is insufficient, because the defender is not constrained to make an optimal recourse decision. Thus, the challenge is to stipulate a set of constraints that forces the defender to choose optimal y-values, given removed facilities as determined by x. Fortunately, the assumption that the defender can assign as many nodes to a facility as desired simplifies the problem considerably.

As a hint to steer students in the right direction, define T_{ij} as the set of all facilities that are farther than facility $j \in F$ from demand point $i \in N$ (i.e., $T_{ij} = \{k \in F\colon d_{ik} > d_{ij}\}$). Consider a node/facility pair i, j. For each $k \in T_{ij}$, the insight should be that y_{ik} must be zero, unless facility j is interdicted ($x_j = 1$). This statement is simply written as $y_{ik} \le x_j$, for each $i \in N$, $j \in F$, and $k \in T_{ij}$. Noting that if $x_j = 0$, then *none* of the facilities in $k \in T_{ij}$ can be assigned to node i, the stronger constraints

$$\sum_{k \in T_{ij}} y_{ik} \le x_j \quad \forall i \in N, \forall j \in F \tag{9d}$$

force the y variables to correspond to the follower's actions. The overall formulation is given by

$$\max \sum_{i \in N} \sum_{j \in F} a_i d_{ij} y_{ij} \tag{10a}$$

$$\text{s.t. constraints (9b)–(9d)}, \tag{10b}$$

$$y_{ij} \in \{0,1\} \quad \forall i \in N, \forall j \in F, \qquad x_j \in \{0,1\} \quad \forall j \in F, \tag{10c}$$

where the binary restrictions on y can be equivalently relaxed by enforcing $0 \le y_{ij} \le 1$.

3.1.2. Node Deletion. We now revisit the node deletion problem on general graphs $G(N, E)$ and formulate another interdiction model using the same process as above. There is no literal "defender" in this problem, but the problem of limiting the maximum component size can be cast as a Stackelberg game: The attacker deletes nodes (with a minimization objective), whereas the defender seeks to find the largest component possible in the graph. If that largest component exceeds τ, then the solution is not feasible; otherwise, the solution is feasible.

Following the same process as above, the objective is again clear. Let x_i, $\forall i \in N$, be binary variables that equal 1 if and only if node i is deleted. The objective is to simply minimize $\sum_{i \in N} x_i$. However, defining constraints on the maximum component size is not straightforward. Perhaps the most elegant approach defines variables $y_{ij} = 1$ if nodes i and j belong to the same component after the attacker has deleted nodes. Note that $\sum_{j \in N} y_{ij}$ is the number of nodes in the same component as node i. Hence, one set of constraints enforces the conditions

$$\sum_{j \in N} y_{ij} \le \tau \quad \forall i \in N. \tag{11a}$$

However, a similar challenge now arises as before: Whereas the attacker wishes to push down the y-values toward zero, constraints must be in place from the defender's perspective that force y_{ij} to equal 1 when i and j belong to the same component. A starting point is to realize that $y_{ii} = 1$ always, for each i. The challenge is then reduced to the following: For each edge $(i, j) \in E$, derive constraints that require $y_{ik} = y_{jk}$, for each $k \in N$, unless either node i or j has been deleted. A further hint is to translate this condition to the following: Both $y_{ik} - y_{jk} \ge 0$ and $y_{jk} - y_{ik} \ge 0$ whenever edge (i, j) survives the attack, and otherwise the constraints should be ignored. To ignore the constraints, their right-hand sides can be set to any number less than or equal to -1. This gives rise to the following set of constraints:

$$y_{ik} - y_{jk} \ge -(x_i + x_j) \quad \forall (i, j) \in E, \forall k \in N; \tag{11b}$$
$$y_{jk} - y_{ik} \ge -(x_i + x_j) \quad \forall (i, j) \in E, \forall k \in N. \tag{11c}$$

The overall model is

$$\min \sum_{i \in N} x_i \tag{12a}$$
$$\text{s.t. constraints (11a)–(11c)}, \tag{12b}$$
$$y_{ij} \in \{0, 1\} \quad \forall i, j \in N, i \ne j, \qquad y_{ii} = 1 \quad \forall i \in N, \qquad x_i \in \{0, 1\} \quad \forall i \in N. \tag{12c}$$

This model is more difficult for students to work out without instructor guidance, but it has value in expanding students' imagination on what kinds of interdiction problems can be captured with (linear) MIP formulations. This model is also straightforward to extend to the case when the objective is to minimize the maximum component size, subject to a budget on the number of nodes the attacker can delete. (Introduce an auxiliary variable z, representing the maximum component size, and minimize z. Then, constraints enforce $z \ge \sum_{j \in N} y_{ij}$, for each $i \in N$.) This extension thus helps to illustrate several modeling tricks that may be of use to students in formulating other complex problems.

3.1.3. Shortest Path Interdiction. In this problem, we reconsider a network $G(N, A)$ having arc costs c_{ij}. However, in this case, if an interdictor has attacked $(i, j) \in A$, the cost of using (i, j) is increased to $c_{ij} + d_{ij}$. The defender will select a shortest path given the arc costs after the attack has been made; hence, the attacker's goal is to maximize the defender's shortest path cost. As in the facility interdiction problem, we assume for simplicity that the attacker is simply constrained to attack no more than some integer r arcs. Similar to the node deletion problem, though, the approach also requires a clever selection of follower variables to complete the formulation.

Downloaded from informs.org by [216.73.216.247] on 14 November 2026, at 09:15 . For personal use only, all rights reserved.

Downloaded from informs.org by [216.73.216.247] on 14 November 2026, at 09:15 . For personal use only, all rights reserved.

For this problem, suppose that we bring back the dual variables π_i for the interdiction problem and adopt their interpretation as right-hand shadow prices, i.e., π_i is the shortest path cost from node i to node n (the destination), given costs as determined by the attacker. Thus, π_n is defined as 0, and π_1 represents the defender's shortest path cost. The interdiction problem thus maximizes π_1 and makes the restriction that no more than r arcs can be attacked. To ensure that the π-values match the defender's shortest path costs as appropriate, we merely need to ensure that $\pi_i - \pi_j \leq$ (cost of using arc (i,j)): A rational defender would never pay a cost of more than $\pi_j +$ (cost of arc (i,j)) to move from node i to node n. This implies that the shortest path interdiction model can be formulated as

$$\max \ \pi_1 \tag{13a}$$
$$\text{s.t.} \ \pi_i - \pi_j \leq c_{ij} + d_{ij}x_{ij} \quad \forall (i,j) \in A, \tag{13b}$$
$$\sum_{(i,j)\in A} x_{ij} \leq r, \tag{13c}$$
$$x_{ij} \in \{0,1\} \quad \forall (i,j) \in E, \tag{13d}$$
$$\pi_i \text{ unrestricted } \forall i \in N\backslash\{n\}, \text{ and } \pi_n = 0. \tag{13e}$$

Note that (13b) takes exactly the form of the dual feasibility constraints. In the next section, we demonstrate that encountering the defender's dual formulation in these MIP formulations is not merely a coincidence.

3.2. Duality as a Tool for Formulating Interdiction Models

The preceding three formulations required modeling analysis for each specific problem, some of which involved nonintuitive defender decision variables. Rather than formulating defender variables and constraints for each interdiction problem on an ad hoc basis, formulation (13) hints that we may be able to employ duality methods to automatically formulate interdiction problems.

To this end, consider the following generic interdiction problem. The attacker's variables are given by $\mathbf{x}$ and are constrained to belong to the set X. Often in linear models, $X = \{\mathbf{x}\text{: } G\mathbf{x} \leq \mathbf{g}, \text{ all } \mathbf{x} \text{ binary}\}$, so that the attacker decisions must be binary valued and constrained to lie within some polyhedron. The generic (linear) interdiction problem can be stated as follows, where for convenience we assume that $\mathbf{x}$ and $\mathbf{y}$ are both n-dimensional vectors:

$$\min_{\mathbf{x}\in X} \max \ \sum_{j=1}^{n} (c_j - d_j x_j) y_j \tag{14a}$$
$$\text{s.t.} \ \mathbf{t}_i^\top \mathbf{y} \leq h_i - \mathbf{w}_i^\top \mathbf{x} \quad \forall i = 1, \ldots, m, \tag{14b}$$
$$\mathbf{y} \geq \mathbf{0}, \tag{14c}$$

where all constraint vectors $\mathbf{t}_i$ and $\mathbf{w}_i$ are n-dimensional, c_j and d_j are scalars for all $j = 1, \ldots, n$, and h_i is a scalar for all $i = 1, \ldots, m$. Let T be the matrix composed of rows $\mathbf{t}_1^\top, \ldots, \mathbf{t}_m^\top$, and denote the jth column of T as $\mathbf{t}^j$, for $j = 1, \ldots, n$. Taking the dual of the defender's maximization problem yields the following problem, where duals $\boldsymbol{\mu}$ are associated with (14b).

$$\min_{\mathbf{x}\in X} \min \ \sum_{i=1}^{m} (h_i - \mathbf{w}_i^\top \mathbf{x}) \mu_i \tag{15a}$$
$$\text{s.t.} \ \boldsymbol{\mu}^\top \mathbf{t}_j \geq c_j - d_j x_j \quad \forall j = 1, \ldots, n, \tag{15b}$$
$$\boldsymbol{\mu} \geq \mathbf{0}. \tag{15c}$$

Now, because both the attacker and defender problems are written as minimization problems, the formulation can be stated as the following (single) optimization problem:

$$\min \sum_{i=1}^{m} (h_i - \mathbf{w}_i^\top \mathbf{x})\mu_i \quad \text{subject to constraints (15b), (15c), and } \mathbf{x} \in X. \tag{16}$$

Unfortunately, the objective is nonlinear due to the multiplication of x and μ variables in the objective, unless all of the $\mathbf{w}_i$ vectors are zero (as they would be, for instance, in the previous shortest path model). Fortunately, given bounds (perhaps implied) on μ variables, it is possible to linearize all such quadratic terms if all x variables are binary valued. Suppose that $\mu_i \in [l_i, u_i]$, for $i = 1, \ldots, m$. We replace each quadratic term $\mu_i x_j$ with a continuous variable v_{ij} and constrain this variable by

$$v_{ij} \le u_i x_j, \tag{17a}$$
$$v_{ij} \le \mu_i + l_i x_j - l_i, \tag{17b}$$
$$v_{ij} \ge u_i x_j + \mu_i - u_i, \tag{17c}$$
$$v_{ij} \ge l_i x_j. \tag{17d}$$

These standard linearization inequalities can be derived by noting that $l_i \le \mu_i \le u_i$ and $0 \le x_j \le 1$, and therefore the following four inequalities hold: $(\mu_i - l_i)x_j \ge 0$, $(u_i - \mu_i)x_j \ge 0$, $(\mu_i - l_i)(1 - x_j) \ge 0$, $(u_i - \mu_i)(1 - x_j) \ge 0$. Expanding those terms in the inequalities and substituting v_{ij} where appropriate yields inequalities (17a)–(17d). To see that these inequalities force $v_{ij} = \mu_i x_j$, note that if $x_j = 0$, then (17a) and (17d) force $v_{ij} = 0$. If $x_j = 1$ on the other hand, (17b) and (17c) force $v_{ij} = \mu_i$, as desired. An important note is that the linearization trick here is applicable so long as at least one variable in each quadratic term is forced to take on its lower or upper bound, but no value in between (which is most commonly observed when a variable is restricted to be binary). In the above example, if μ_i and x_j are both allowed to assume variable values that are strictly within their lower and upper bounds, then the constraints given in (17a)–(17d) remain valid, but do not force $v_{ij} = \mu_i x_j$.

Hence, there is a three-step "recipe" for formulating these interdiction problems. Step one is to naturally formulate the problem as a two-stage min-max (or max-min) problem. Step two is to dualize the inner problem (the defender's problem). Step three is to combine the attacker and defender optimization problems and linearize any quadratic terms arising in the objective function. It is instructive to see how this process works in the context of facility location example given in §3.1, and also how it might work on a production planning model. (The node deletion example is slightly more involved and is omitted here; see Shen and Smith [26] for more details. The shortest path interdiction problem follows immediately by taking the dual of the defender's problem, because no linearization step is necessary due to the presence of x variables only in the defender problem's objective.)

3.2.1. Revisiting the Facility Interdiction Problem. The next exercise is ideal for illustrating the three-step process above; moreover, with additional analysis regarding duality interpretations and max-min structures in linear MIP formulations, we can recover the original formulation derived in §3.1. First, we start with a natural max-min formulation, where w_{ij} is the defender's decision variable that equals 1 if and only if node $i \in N$ is assigned to facility $j \in F$:

$$\max_{\mathbf{x} \in X} \min \quad \sum_{i \in N} \sum_{j \in F} a_i d_{ij} w_{ij} \tag{18a}$$
$$\text{s.t.} \quad \sum_{j \in F} w_{ij} = 1 \quad \forall\, i \in N, \tag{18b}$$
$$w_{ij} \le 1 - x_j \quad \forall\, i \in N, \forall\, j \in F, \tag{18c}$$
$$\mathbf{w} \ge \mathbf{0}, \tag{18d}$$

Downloaded from informs.org by [216.73.216.247] on 14 November 2026, at 09:15 . For personal use only, all rights reserved.

where X represents the attacker's feasible region, and (18c) restricts the defender to utilize only those facilities that have not been attacked. Associating λ_i with (18b) and $-\mu_{ij}$ with (18c), the dual of the defender's problem is given by

$$\max \left\{ \sum_{i \in N} \lambda_i - \sum_{i \in N} \sum_{j \in F} \mu_{ij}(1 - x_j) \right\} \tag{19a}$$

$$\text{s.t. } \lambda_i - \mu_{ij} \le a_i d_{ij} \quad \forall i \in N, \forall j \in F, \tag{19b}$$

$$\boldsymbol{\mu} \ge \mathbf{0}. \tag{19c}$$

Note, however, that (18c) is redundant whenever $x_j = 0$, being implied by (18b). Therefore, the right-hand shadow price of (18b) is zero whenever the x_j variable present in this constraint is zero. Thus, one may suspect that $\mu_{ij}(1 - x_j) = 0$ for every i, j pair. (A separate analysis directly on (19a)–(19c) proves that if $\mu_{ij}(1 - x_j) > 0$ at optimality, an alternative optimal solution exists in which λ_i is reduced by μ_{ij}, and μ_{ij} takes a value of zero.) Therefore, it is valid to restrict $\mu_{ij}(1 - x_j) = 0$ for every i, j pair, which implies that we can simply omit the term $\sum_i \sum_j \mu_{ij}(1 - x_j)$ in the objective. To force μ_{ij} to equal zero whenever $x_j = 0$, we state constraints $\mu_{ij} \le M x_j$, for all i and j. However, at optimality, the μ variables always take their largest permissible values (which in turn allow λ variables to become large). Hence, we can substitute $\mu_{ij} = M x_j$, and the resulting formulation is

$$\max \sum_{i \in N} \lambda_i \tag{20a}$$

$$\text{s.t. } \lambda_i \le a_i d_{ij} + M x_j, \tag{20b}$$

where M is some very large constant. At this point, however, we see that λ_i is the smallest value of $a_i d_{ij}$, from among all $j \in F$: $x_j = 0$. Hence, if one defines binary indicator variables y_{ij}, which equal 1 if and only if both $x_j = 0$ and $d_{ij} \le d_{ik}$ for all $k \in F$: $x_k = 0$, the objective (20a) becomes

$$\max \sum_{i \in N} \sum_{j \in F} a_i d_{ij} y_{ij},$$

and $\mathbf{y}$ is defined exactly by constraints (9c) and (9d) as before.

3.2.2. Production Planning Problem. Consider a planning problem on T periods. In period $t = 1, \ldots, T$, each item can be produced at a cost of c_t, there exists a demand of d_t items (which must be satisfied without backordering), and excess items can be stored for use in future periods at a cost of h_t per unit stored. The defender's objective is to meet all production requirements at a minimum cost. The attacker wishes to modify the defender's data to maximize its minimum production cost. Several variations of attacks can be considered; here, we examine the case in which the attacker can cause demand to increase by δ units in r different periods.

Define x_t to be the attacker's binary decision variables, which equal 1 if and only if the attacker increases demand in period t. The attacker's actions are defined by $X = \{\mathbf{x}\text{: } \sum_{t=1}^{T} x_t \le r,\ x_t \in \{0,1\}\ \forall t = 1, \ldots, T\}$. The defender's decision variables are given by y_t, the amount of items to produce in period t (which are immediately available to satisfy demand in that period), and s_t, the amount inventory held over after period t. Following the three-step process as before, we have that the defender's problem is given by

$$\min \sum_{i \in T} (c_t y_t + h_t s_t) \tag{21a}$$

$$\text{s.t. } y_t + s_{t-1} - s_t = d_t + \delta x_t \quad \forall t = 1, \ldots, T, \tag{21b}$$

$$\mathbf{s}, \mathbf{y} \ge \mathbf{0}, \tag{21c}$$

Downloaded from informs.org by [216.73.216.247] on 14 November 2026, at 09:15 . For personal use only, all rights reserved.

where s_0 is defined to be 0. Associating dual variables μ_t with (21b), the dual to (21) becomes

$$\max \sum_{t=1}^{T}(d_t + \delta x_t)\mu_t \tag{22a}$$

$$\text{s.t. } \mu_t \le c_t \quad \forall t = 1, \ldots, T, \tag{22b}$$

$$\mu_{t+1} - \mu_t \le h_t \quad \forall t = 1, \ldots, T, \tag{22c}$$

with π being unrestricted, and where $\mu_{T+1} \equiv 0$. The final step is to combine the attacker and defender optimization problems. The feasible region will combine restrictions enforcing $x \in X$ with (22b) and (22c). But to achieve this, we must have simple lower and upper bounds on the μ variables. In this case, (22b) provides upper bounds directly on these variables, which makes sense from a shadow-price perspective: If demand increases by one in period t, then it could be satisfied by simply producing another item in period t at a cost of c_t. Therefore, the shadow price for constraint (21b) corresponding to period t should never exceed c_t. Logically, a lower bound is 0: The minimum cost to supply all demand is a nondecreasing function of d_t. (This could also be shown by revising (21b) as a greater-than-or-equal-to constraint, which does not affect the optimization model but implies that μ-values should be nonnegative.) With these bounds, the overall formulation can be stated as follows, where v_t is a variable that linearizes $x_t\mu_t$:

$$\max \sum_{t=1}^{T}(d_t\mu_t + \delta v_t) \tag{23a}$$

$$\text{s.t. constraints (22b), (22c), } x \in X, \tag{23b}$$

$$v_t \le c_t x_t \text{ and } v_t \le \mu_t \quad \forall t = 1, \ldots, T, \tag{23c}$$

$$v_t \ge c_t x_t + \mu_t - c_t \text{ and } v_t \ge 0 \quad \forall t = 1, \ldots, T. \tag{23d}$$

In fact, because the objective seeks to make v_t as large as possible, (23c) suffices to enforce the linearization of $v_t = x_t\mu_t$, and (23d) can be omitted.

Remark 3. Note that if production problem places some limit p_t on the total number of items that can be produced in each period, (23b) would be altered such that μ_t can actually exceed c_t. In this case, it is necessary to derive insights about the model to determine the largest dual values that can arise per period. Another interesting problem arises when there exist fixed costs to produce in any period. For the capacitated case (i.e., the CLSP), though, this normally entails the introduction of binary variables to the defender problem, which precludes the dualization of the defender problem. Formulating an interdiction turns out to be far more difficult in this case.

For an advanced class, though, one can actually integrate dynamic programming lessons very well into this challenge. The CLSP can in fact be solved by DP, and moving from the DP algorithm for the CLSP to an equivalent shortest path problem (containing a pseudopolynomial number of variables and constraints) is straightforward. Now, the defender's problem can indeed be represented by a (large) linear programming problem, and the foregoing analysis can be applied.

4. Robust Optimization Models

The formulation of optimization problems under uncertainty is another modern mathematical programming area that has received substantial interest over the past decade. Stochastic and robust optimization define two primary branches that address the presence of uncertainty in mathematical optimization. At the risk of oversimplifying the distinction between these fields, stochastic programming models are typically associated with the optimization

Downloaded from informs.org by [216.73.216.247] on 14 November 2026, at 09:15 . For personal use only, all rights reserved.

Downloaded from informs.org by [216.73.216.247] on 14 November 2026, at 09:15 . For personal use only, all rights reserved.

of expected costs, whereas robust optimization models tend to optimize over worst-case realizations of uncertain data. The latter idea is related to the interdiction problems discussed in §3, in that the problem solver wishes to, e.g., minimize the maximum cost that could result from uncertain data. The latter approach also guarantees that the chosen decision remains feasible under any possible outcome of data.

Because the min-max (or max-min) structure of robust optimization models arises in robust optimization models as it does in interdiction models, we discuss it as a natural extension accessible to students taking a first course in mathematical programming. The scope of robust optimization research extends far beyond model formulation, and we point the interested reader to related research on a broad range of robust optimization studies mentioned in §5. For this chapter, we limit the robust optimization discussion to that required to correctly formulate robust optimization problems with uncertain data occurring in the objective.

Consider the following linear programming problem, having deterministic data:

$$\min \ \mathbf{c}^\top \mathbf{x} \tag{24a}$$
$$\text{s.t.} \ A\mathbf{x} \geq \mathbf{b}, \tag{24b}$$
$$\mathbf{x} \geq \mathbf{0}. \tag{24c}$$

In (24), A is an $m \times n$ matrix and all other variables and data have conforming dimensions. For brevity in this discussion, we assume that $\mathbf{c}$ is the only uncertain data, though this analysis can be extended to uncertainty arising in A and $\mathbf{b}$ as well (see Remark 4). Although the exact value of $\mathbf{c}$ is not known, a robust optimization model assumes that it must lie in some *uncertainty set* $\mathcal{C}$. The problem is to now find a value for $\mathbf{x}$ that is feasible to (24b) and (24c), such that the worst-case objective function value, i.e., the largest value of $\mathbf{c}^\top \mathbf{x}$ over all $\mathbf{c} \in \mathcal{C}$, is as small as possible. Compared to the interdiction models discussed previously, the defender now commits to a decision *first*, and the attacker, with full knowledge of the defender's action, makes its decision last. Hence, a min-max game is again being played, and, again, dualizing the inner (attacker) problem permits us to formulate a single minimization problem.

To approach this problem in the classroom, a rudimentary uncertainty set can be given as follows, where $\mathbf{c}$ is assumed to be n-dimensional, δ_j is a positive constant for each $j = 1, \ldots, n$, and B is a parameter whose interpretation we return to shortly:

$$\mathcal{C} = \left\{ \mathbf{c} \colon \bar{c}_j \leq c_j \leq \bar{c}_j + \delta_j, \forall\, j = 1, \ldots, n, \text{ and } \sum_{j=1}^{n} \frac{c_j - \bar{c}_j}{\delta_j} \leq B \right\}. \tag{25}$$

Equivalently, $\mathbf{c}$ belongs to $\mathcal{C}$ if and only if it can be written as a solution to the following system of inequalities, where z_j is a variable for all $j = 1, \ldots, n$:

$$c_j = \bar{c}_j + \delta_j z_j \quad \forall\, j = 1, \ldots, n; \tag{26a}$$
$$\sum_{j=1}^{n} z_j \leq B; \tag{26b}$$
$$0 \leq z_j \leq 1 \quad \forall\, j = 1, \ldots, n. \tag{26c}$$

Note that z_j measures how far c_j goes from its smallest possible (nominal) value of $\bar{c}_j$ to its largest possible value of $\bar{c}_j + \delta_j z_j$.

The motivation behind robust optimization models such as this is important to explain and natural for the class to understand. If $\mathbf{c}$ were simply constrained to lie in the region constrained by (26a) and (26c), then there is a unique worst-case vector regardless of $\mathbf{x}$, attained by letting $z_j = 1$, $\forall\, j = 1, \ldots, n$. However, this would imply that each cost component randomly and simultaneously took on its largest possible value, which, although

possible, is often exceedingly unlikely. The parameter B helps to control the extent to which a decision maker hedges his or her decision against worst-case outcomes. The case of $B = 0$ is exactly the deterministic model, whereas the case of $B = n$ is the case in which all cost data simultaneously take their largest possible values.

A confusing point in class often arises when trying to formulate the min-max robust optimization problem as a single linear programming problem. Logically, one can substitute c_j as in (26a), and the objective function (24a) is revised to minimize $\bar{\mathbf{c}}^\top \mathbf{x} + \sum_{j=1}^{n} \delta_j z_j x_j$, where $\bar{\mathbf{c}}$ represents the vector of $\bar{c}_j$-values. However, if one simply includes (26b) and (26c) alongside the original problem constraints, then it is the *defender* who is making the choice of $\mathbf{z}$, and not the attacker. Hence, all $z_j = 0$ at optimality and the original problem (24) is solved. In fact, a class demonstration may help to underscore this point. One effective way of demonstrating the fallacy of this approach is to formulate the problem incorrectly and ask the students if they can anticipate what the z-values will be. After obtaining an all-zero solution (or in any event, one in which the z variables do not reflect the worst-case realization of $\mathbf{c}$ from $\mathcal{C}$), the students should see that the defender is incorrectly allowed to set the z variables.

As a challenge to determine the correct way to set z variables, the answer once again lies in duality. The optimization problem at hand can be written now as

$$\min\{\bar{\mathbf{c}}^\top \mathbf{x} + \Psi(\mathbf{x})\} \quad \text{s.t. (24b) and (24c)}, \tag{27}$$

where $\Psi(\mathbf{x})$ is the optimal objective function value of the attacker's problem, given by

$$\Psi(\mathbf{x}) = \max \sum_{j=1}^{n} \delta_j x_j z_j \tag{28a}$$

$$\text{s.t.} \sum_{j=1}^{n} z_j \le B, \tag{28b}$$

$$0 \le z_j \le 1 \quad \forall j = 1, \ldots, n, \tag{28c}$$

where the x-values in (28a) are known and can be treated as data in the solution of this problem. Note that $\Psi(\mathbf{x})$ can be thought of as an "additional cost" incurred beyond that anticipated by the deterministic objective in (24a), which the attacker seeks to maximize.

Because (28) is a linear program, $\Psi(\mathbf{x})$ can also be obtained as the objective function value to the dual of (28), given as follows, where λ is associated with (28b), and μ_j is the dual associated with the upper-bounding inequalities in (28c):

$$\min \left\{ B\lambda + \sum_{j=1}^{n} \mu_j \right\} \tag{29a}$$

$$\text{s.t.} \quad \lambda + \mu_j \ge \delta_j x_j \quad \forall j = 1, \ldots, n, \tag{29b}$$

$$\lambda \ge 0, \quad \mu_j \ge 0, \forall j = 1, \ldots, n. \tag{29c}$$

Hence, the defender seeks to set λ and $\boldsymbol{\mu}$ in a way that minimizes the extra cost term Ψ; this minimum cost must correspond to the worst-case additional cost that can occur due to the attacker's actions. Now, it is possible to state one single linear programming model as follows:

$$\min \left\{ \bar{\mathbf{c}}^\top \mathbf{x} + B\lambda + \sum_{j=1}^{n} \mu_j \right\} \tag{30a}$$

$$\text{s.t.} \quad \text{constraints (24b), (24c), (29b), and (29c).} \tag{30b}$$

This problem is the *robust counterpart* of (24), given $\mathcal{C}$. Note that it remains a linear program, and will remain a linear program so long as $\mathcal{C}$ is itself polyhedral.

Downloaded from informs.org by [216.73.216.247] on 14 November 2026, at 09:15 . For personal use only, all rights reserved.

Remark 4. Before proceeding with an illustrative example, it is important to comment on how uncertainty in the A matrix and $\mathbf{b}$ vector would be handled in a robust setting. The concept is that $\mathbf{x}$ must be feasible to all constraints, regardless of the actual outcome of A and $\mathbf{b}$.

Perhaps the simplest way of explaining this idea is to imagine that the attacker plays the following game. Let $\mathbf{a}_i$ denote the ith row of A, and let $\mathcal{U}$ be the uncertainty set to which A must belong. The attacker first determines if there is any outcome of $(\mathbf{a}_1, b_1)$ so that $\mathbf{a}^\top{}_1\mathbf{x} < b_1$, i.e., so that $\mathbf{x}$ violates the first constraint. If not, the attacker considers $(\mathbf{a}_2, b_2)$ instead, trying to make $\mathbf{x}$ infeasible to the second constraint, and so on for each constraint. Note that the analysis can, perhaps surprisingly, be done independently for each row. Hence, there is a notion of rowwise independence in stating $\mathcal{U}$, i.e., $\mathcal{U}$ can be stated as all possible sets of $(\mathbf{a}_1, b_1)$, unioned with all possible sets of $(\mathbf{a}_2, b_2)$, and so on.

A robust formulation therefore ensures that for every row i of the constraint set, $\mathbf{x}$ is feasible for any $(\mathbf{a}_i, b_i)$ that belongs to $\mathcal{U}$. For instance, consider the case in which uncertainty is confined to A and $\mathbf{b}$ is deterministic (the case in which $\mathbf{b}$ is uncertain as well is an easy extension). Define a_{ij} to be the row i, column j element of A. Suppose that the uncertainty set is similar to (26), with (26a) replaced by $a_{ij} = \bar{a}_{ij} - \delta^i_j z_j$, $\forall\, j = 1, \ldots, n$, and in which the right-hand side of (26b) is given by the parameter B_i. In this case, constraint i becomes

$$\bar{\mathbf{a}}_i^\top \mathbf{x} \geq b_i + \sum_{j=1}^{n} \delta^i_j z_j x_j,$$

where $\bar{\mathbf{a}}_i$ is the vector $[\bar{a}_{i1}, \ldots, \bar{a}_{in}]^\top$, and the summation on the right-hand side of this expression is optimized as in (28) (or (29)). Hence, constraint i becomes

$$\bar{\mathbf{a}}_i^\top \mathbf{x} \geq b_i + B_i\lambda^i + \sum_{j=1}^{n} \mu^i_j, \tag{31}$$

and the additional constraints

$$\lambda^i + \mu^i_j \geq \delta^i_j x_j \quad \forall\, j = 1, \ldots, n, \tag{32}$$

are added to the model along with nonnegativity restrictions on the λ^i and μ^i variables.

We now proceed with an illustration of robust optimization in the context of shortest path programs, which once again carries interesting duality interpretations. Confining uncertainty to the objective function, and using exactly the uncertainty set embodied by (26) (with all data and variables indexed by arcs $(i, j) \in A$), the robust equivalent formulation of the shortest path problem is given as

$$\min \left\{ \sum_{(i,j)\in A} c_{ij}x_{ij} + B\lambda + \sum_{(i,j)\in A} \mu_{ij} \right\} \tag{33a}$$

$$\text{s.t.} \sum_{j\in \mathrm{FS}(i)} x_{ij} - \sum_{h\in \mathrm{RS}(i)} x_{hi} = b_i \quad \forall\, i \in N, \tag{33b}$$

$$\lambda + \mu_{ij} \geq \delta_{ij}x_{ij} \quad \forall\, (i,j) \in A, \tag{33c}$$

$$\mathbf{x} \geq \mathbf{0}, \quad \lambda \geq 0, \quad \boldsymbol{\mu} \geq \mathbf{0}. \tag{33d}$$

Note here that δ_{ij} plays the role of d_{ij} used in the shortest path interdiction model. In fact, students should now define the precise difference between these two models: Robust models allow the attacker to observe the defender's decision (i.e., the defender's shortest path) first, whereas interdiction models require the attacker to determine which arcs are interdicted first, thus allowing the defender to choose a path as a recourse to the interdicted arcs.

Downloaded from informs.org by [216.73.216.247] on 14 November 2026, at 09:15 . For personal use only, all rights reserved.

The goal is now to interpret the terms $B\lambda$ and $\sum_{(i,j)\in A}\mu_{ij}$ in the objective, so that they make sense to the student, rather than simply being introduced as dual variable values that somehow "fall out" of the robust formulation analysis. First, observe that the attacker can optimize its problem by the following simple sorting algorithm: Let P be the set of arcs used by the defender ($P=\{(i,j)\in A\colon x_{ij}=1\}$), and set $z_{ij}=1$ for the B largest values of δ_{ij} among those in P. For this discussion, we say that arc (i,j) is interdicted when $z_{ij}=1$.

Now, consider the interpretation of λ: This is essentially the amount by which the attacker could increase the defender's cost if it were allowed to interdict an additional arc (in the right-hand shadow price interpretation), or the amount of decrease in total cost if the attacker were constrained to interdict one fewer arc (in the left-hand shadow price interpretation). Let costs $\theta_1\ge\theta_2\ge\cdots\ge\theta_{|P|}$ be the cost data resulting from sorting δ_{ij}, $(i,j)\in P$, in nonincreasing order of their values. For simplicity, assume that $0<B<|P|$.

In the left-hand shadow price interpretation of λ, decreasing the budget B by one forces the attacker to interdict one fewer arc. The resulting change in cost would be θ_B. In the right-hand shadow price interpretation, the next arc to be interdicted by the attacker would change the cost by θ_{B+1}. Hence, $\lambda=\theta_B$ or θ_{B+1}, depending on whether λ is interpreted as a left-hand or right-hand shadow price, respectively. In either case, having determined λ, it is easy to determine μ-values. The objective seeks to minimize μ-values, and hence (33c) forces $\mu_{ij}=\max\{\delta_{ij}x_{ij}-\lambda,0\}$; that is, μ_{ij} equals 0, unless $(i,j)\in P$ and $\delta_{ij}>\lambda$.

Suppose that we adopt the left-hand shadow price interpretation of λ $(=\theta_B)$, and consider the right-hand shadow price interpretation of μ_{ij}. (A parallel exercise can adopt the right-hand shadow price of λ and seek to interpret left-hand shadow prices of the μ-values.) If (i,j) is not currently being interdicted by the attacker, then clearly $\mu_{ij}=0$ by complementary slackness. Otherwise, suppose (i,j) is being interdicted by the attacker. If $z_{ij}\le 2$ instead of being forced to be no more than 1, then the attacker would set $z_{ij}=2$, which would replace the attack on the least valuable interdiction currently being made. The net impact on the defender's objective function is $\delta_{ij}-\theta_B$, which corresponds exactly to the definition for μ_{ij} given above. Therefore, because the cost of every interdicted arc is at least $\lambda=\theta_B$, its cost can be written as λ plus the nonnegative term $(\delta_{ij}-\lambda)$. Equivalently, the sum of interdicted costs is given by $\lambda B+\sum_{(i,j)\in A}\mu_{ij}$, which matches exactly the objective in (33a). Constraints (33c) and nonnegativity restrictions on λ and $\boldsymbol{\mu}$ serve to enforce the definition of $\mu_{ij}=\max\{\delta_{ij}x_{ij}-\lambda,0\}$.

5. Summary and Further Reading

The purpose of this tutorial has been to suggest the instruction of dynamic programming and duality material within foundational operations research classes in a more intuitive manner, and to expand course material to encompass contemporary interdiction and robust optimization modeling concepts. The end goal in the dynamic programming module is to help students develop a natural skill in solving problems by recursion, rather than introducing recursive equations that tend to confound beginning students. This material can be linked to duality study as well, especially through problems such as shortest path optimization that can be solved and interpreted via dynamic programming or linear programming. The educational goal behind the study of duality is twofold: improve students' understanding of what dual variables mean and how they are practically important in solving real problems. The competition theme of the problems presented here tends to pique students' interest in the material and help them understand the concepts at a higher level.

This paper is also intended as a tutorial for those wanting to learn the basics of the foregoing material, but is not meant as a literature survey in this area. The list of dynamic programming references in the literature is quite extensive. In addition to the textbook introduction in Winston [35] referenced earlier, recent works by Sniedovich [32] and Bertsekas [9, 10] are comprehensive volumes providing a contemporary and thorough coverage of dynamic programming theory and practice.

Downloaded from informs.org by [216.73.216.247] on 14 November 2026, at 09:15 . For personal use only, all rights reserved.

As for modern network interdiction studies, the work of Wood [36] is an essential starting point for the technical basics that underscore the field. The survey articles by Brown et al. [13, 14], Smith [28], and Smith and Lim [30] provide starting points for readers wanting to learn more about the area. For three-stage games in which the defender acts first to fortify networks against an attacker, a few representative works of interest include Church and Scaparra [15], Scaparra and Church [24], and Smith et al. [31]. Interdiction under uncertainty is another area that has received substantial interest in the last few decades; see Cormican et al. [17] and Janjarassuk and Linderoth [22] for two approaches to these problems.

Robust optimization, conceived as described above with continuous uncertainty sets, was pioneered by Ben-Tal and Nemirovski [3, 4, 5, 6], El-Ghaoui and Lebret [19], and El-Ghaoui et al. [20], but the earliest such study in this area dates back Soyster's [33] work in 1973. The uncertainty model used in the present chapter follows the "budgeted uncertainty" idea of Bertsimas and Sim [11]; see also Ben-Tal and Nemirovski [7] for robust counterpart formulations in the presence of alternative (nonpolyhedral) uncertainty sets. A recent tutorial chapter by Bertsimas and Thiele [12] was published in this tutorial series in 2006, and the book by Ben-Tal et al. [8] summarizes a decade's worth of progress in this field. Nemirovski [23] provides a thorough discussion on robust optimization theory, whereas a more introductory article on basic robust optimization principles can be found in Smith and Ahmed [29].

Acknowledgments

The author is very grateful to Pitu Mirchandani for his careful reading of this paper, and for his insightful remarks that led to a clearer presentation of the material.

References

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows*: *Theory, Algorithms, and Applications*. Prentice Hall, Upper Saddle River, NJ, 1993.

[2] R. Albert, H. Jeong, and A.-L. Barabási. Error and attack tolerance of complex networks. *Nature* 406:378–382, 2000.

[3] A. Ben-Tal and A. Nemirovski. Stable truss topology design via semidefinite programming. *SIAM Journal on Optimization* 7:991–1016, 1997.

[4] A. Ben-Tal and A. Nemirovski. Robust convex optimization. *Mathematics of Operations Research* 23:769–805, 1998.

[5] A. Ben-Tal and A. Nemirovski. Robust solutions to uncertain linear programs. *Operations Research Letters* 25:1–13, 1999.

[6] A. Ben-Tal and A. Nemirovski. Robust solutions of linear programming problems contaminated with uncertain data. *Mathematical Programming* 88(3):411–424, 2000.

[7] A. Ben-Tal and A. Nemirovski. Robust optimization—Methodology and applications. *Mathematical Programming, Series B* 92:453–480, 2002.

[8] A. Ben-Tal, L. El-Ghaoui, and A. Nemirovski. *Robust Optimization*. Princeton University Press, Princeton, NJ, 2009.

[9] D. P. Bertsekas. *Dynamic Programming and Optimal Control*, Vol. 1, 3rd ed. Athena Scientific, Nashua, NH, 2005.

[10] D. P. Bertsekas. *Dynamic Programming and Optimal Control*, Vol. 2, 3rd ed. Athena Scientific, Nashua, NH, 2007.

[11] D. Bertsimas and M. Sim. The price of robustness. *Operations Research* 52:35–53, 2004.

[12] D. Bertsimas and A. Thiele. Robust and data-driven optimization: Modern decision-making under uncertainty. M. P. Johnson, B. Norman, and N. Secomandi, eds. *Tutorials in Operations Research: Models, Methods, and Applications for Innovative Decision Making*. INFORMS, Hanover, MD, 95–122, 2006.

[13] G. G. Brown, W. M. Carlyle, J. Salmerón, and K. Wood. Analyzing the vulnerability of critical infrastructure to attack and planning defenses. H. J. Greenberg and J. C. Smith, eds. *Tutorials in Operations Research: Emerging Theory, Methods, and Applications*. INFORMS, Hanover, MD, 102–123, 2005.

Downloaded from informs.org by [216.73.216.247] on 14 November 2026, at 09:15 . For personal use only, all rights reserved.

[14] G. G. Brown, W. M. Carlyle, J. Salmerón, and K. Wood. Defending critical infrastructure. *Interfaces* 36(6):530–544, 2006.

[15] R. L. Church and M. P. Scaparra. The r-interdiction median problem with fortification. *Geographical Analysis* 39:129–146, 2007.

[16] R. L. Church, M. P. Scaparra, and R. Middleton. Identifying critical infrastructure: The median and covering interdiction models. *Annals of the Association of American Geographers* 94(3):491–502, 2004.

[17] K. J. Cormican, D. P. Morton, and R. K. Wood. Stochastic network interdiction. *Operations Research* 46:184–197, 1998.

[18] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik* 1:269–271, 1959.

[19] L. El-Ghaoui and H. Lebret. Robust solutions to least-square problems with uncertain data matrices. *SIAM Journal of Matrix Analysis and Applications* 18:1035–1064, 1997.

[20] L. El-Ghaoui, F. Oustry, and H. Lebret. Robust solutions to uncertain semidefinite programs. *SIAM Journal on Optimization* 9:33–52, 1998.

[21] M. Grötschel, C. L. Monma, and M. Stoer. Design of survivable networks. M. O. Ball, T. L. Magnanti, C. L. Monma, G. L. Nemhauser, eds. *Network Models*, Handbooks on Operations Research and Management Science, Vol. 7. Elsevier Science, Amsterdam, 617–672, 1995.

[22] U. Janjarassuk and J. Linderoth. Reformulation and sampling to solve a stochastic network interdiction problem. *Networks* 52(3):120–132, 2008.

[23] A. Nemirovski. Lectures on robust convex optimization. http://www2.isye.gatech.edu/~nemirovs/, last accessed July 12, 2012.

[24] M. P. Scaparra and R. L. Church. A bilevel mixed-integer program for critical infrastructure protection planning. *Computers and Operations Research* 35:1905–1923, 2008.

[25] S. Shen and J. C. Smith. Exact interdiction models and algorithms for disconnecting networks via node deletions. Technical report, Department of Industrial and Systems Engineering, University of Florida, Gainesville, 1997.

[26] S. Shen and J. C. Smith. Polynomial-time algorithms for solving a class of critical node problems on trees and series-parallel graphs. *Networks*, ePub ahead of print October 28, http://onlinelibrary.wiley.com/doi/10.1002/net.20464/abstract, 2011.

[27] D. T. Shier. *Network Reliability and Algebraic Structures*. Oxford University Press, Oxford, UK, 1991.

[28] J. C. Smith. Basic interdiction models. J. Cochran, ed. *Wiley Encyclopedia of Operations Research and Management Science*. John Wiley & Sons, Hoboken, NJ, 323–330, 2011.

[29] J. C. Smith and S. Ahmed. Introduction to robust optimization. J. Cochran, ed. *Wiley Encyclopedia of Operations Research and Management Science*. John Wiley & Sons, Hoboken, NJ, 2574–2581, 2011.

[30] J. C. Smith and C. Lim. Algorithms for network interdiction and fortification games. A. Migdalas, P. M. Pardalos, L. Pitsoulis, and A. Chinchuluun, eds. *Pareto Optimality, Game Theory and Equilibria*, Nonconvex Optimization and Its Applications Series. Springer, New York, 609–644, 2008.

[31] J. C. Smith, C. Lim, and F. Sudargho. Survivable network design under optimal and heuristic interdiction scenarios. *Journal of Global Optimization* 38:181–199, 2007.

[32] M. Sniedovich. *Dynamic Programming: Foundations and Principles*, 2nd ed. CRC Press, Boca Raton, FL, 2011.

[33] A. L. Soyster. Convex programming with set-inclusive constraints and applications to inexact linear programming. *Operations Research* 21:1154–1157, 1973.

[34] H. Stackelberg. *The Theory of the Market Economy*. Oxford University Press, Oxford, UK, 1952.

[35] W. Winston. *Operations Research: Applications and Algorithms*, 4th ed. Duxbury Press, Boston, 2004.

[36] R. K. Wood. Deterministic network interdiction. *Mathematical and Computer Modelling* 17(2):1–18, 1993.

[37] M. Yannakakis. Node-and-edge-deletion NP-complete problems. *Proceedings of the 10th Annual ACM Symposium on Theory of Computing*. ACM, New York, 253–264, 1978.

Downloaded from informs.org by [216.73.216.247] on 14 November 2026, at 09:15 . For personal use only, all rights reserved.