CHAPTER 3: INTEGER PROGRAMMING

Overview

To this point, we have considered optimization problems with *continuous* design variables. That is, the design variables can take any value within a continuous feasible set. In Ex. 1.1 from Ch. 1, we optimized the mass of concrete and length of steel to minimize the cost of building a wall. Imagine, for example, that the mass of concrete and length of steel was constrained to take integer values, since your suppliers do not sell fractions of kilograms for cement, nor fractions of meters for steel. In this case, the design variables can only take *integer* values. This situation commonly arises in engineering problems, as will be motivated throughout this chapter. Consequently, a entire category of optimization theory is dedicated to the formulation, solution, and analysis of integer programming (IP) problems.

Generally speaking, IP problems are very difficult to solve. The computational complexity of the associated solvers scale poorly with respect to the number of design variables. However, there are several special cases which exhibit structure that we may exploit to efficiently solve the IP. In this chapter, we first motivate IPs and so-called fractional solutions as a "quick and dirty" method to obtain bounds for the optimal solution. Next we introduce the ubiquitous Dijkstra's Algorithm for solving IPs with a network flow structure. In particular, we revisit the shortest path problem and solve it using Dijkstra's algorithm. For problems without special structure, we can use the Branch and Bound technique to determine integer solutions. Finally, we introduce mixed integer programming (MIP) problems, in which some design variables are continuous and some are integer. This class of optimization problems commonly occur in practice, and will be discussed within the context of an Air Traffic Control problem.

Chapter Organization

This chapter is organized as follows:

- (Section 1) Integer Programming
- (Section 2) Dijkstra's Algorithm & Shortest Path Problem
- (Section 3) Branch & Bound
- (Section 4) Mixed Integer Programming (MIP)
- (Section 5) MIP Example: Air Traffic Control

1 Integer Programs

Suppose you work for a logistics company, and need to determine how many drivers to hire and trucks to purchase for shipping goods. Imagine you are given constraints that produce the feasible set visualized in Fig. 1. Moreover, assume the optimal solution is given by Z^* and $(x_1^*, x_2^*) = (2.2, 8.9)$, where x_1, x_2 correspond to the number of trucks and drivers, respectively. Physically, it makes no sense to hire 2.2 drivers and purchase 8.9 trucks. We require integer solutions.

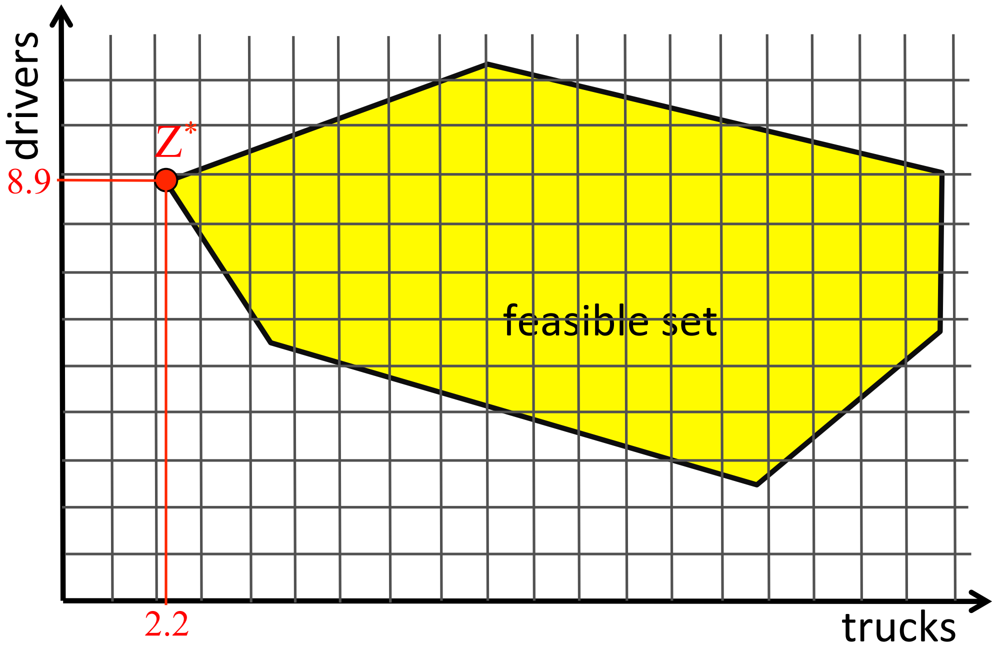

Figure 1: The optimal number of drivers and trucks should take an integer solution, given by the grid. LP and QP tools would give us the solution $(x_1^*, x_2^*) = (2.2, 8.9)$.

Consider the zoom-in of Z^* shown in the left-side of Fig. 2. We call $(x_1^*, x_2^*) = (2.2, 8.9)$ the **fractional solution**, which represents the solution of an IP problem without the integer constraints. One possible approach is to round the fractional solution to the nearest integer. This would produce 2 trucks and 9 drivers as the solution. However, this solution is clearly not feasible. Alternatively, we could round to the nearest feasible solution, either (3 trucks, 9 drivers) or (3 trucks, 8 drivers). We call this **LP rounding**, in the case of linear integer programs. That is, we solve the *relaxed* IP by removing the integer constraints and obtain a fractional solution from the resulting LP. Then we round the fractional solution to the nearest integer solution within the feasible set. However, it is not clear which integer solution is best, (3 trucks, 9 drivers) or (3 trucks, 8 drivers).

Fractional solutions, by themselves, are useful. Namely, they provide upper or lower bounds on the optimum. As illustrated by the right-side of Fig. 2,

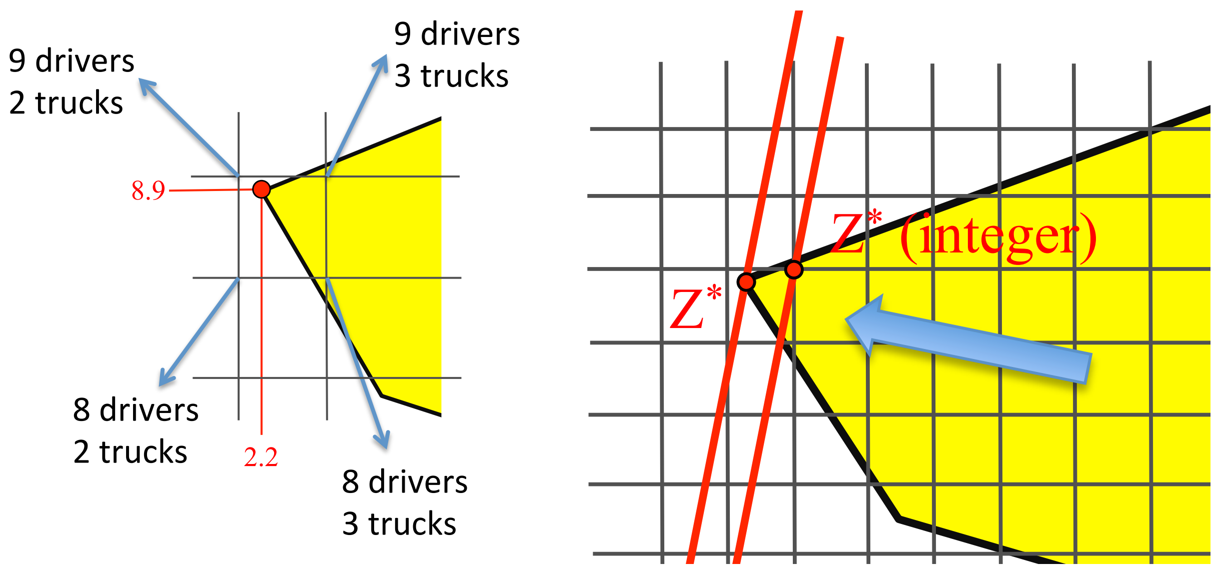

Figure 2: [LEFT] The fractional solution is $(x_1^*, x_2^*) = (2.2, 8.9)$, and could be rounded to find an integer solution. [RIGHT] Fractional solutions provide upper or lower bounds on the integer solution.

- If a min problem, then it provides a lower bound.
- If a max problem, then it provides an upper bound.

Recall our statement that IPs are sometimes very hard to solve exactly. However, guaranteed bounds on the optimal cost are often sufficient. They provide "quick & dirty" estimates for the optimal solution which themselves are not optimal, but provide bounds on the best possible solution. For many situations this is sufficient to achieve one's objective, despite the fraction solutions not making physical sense, i.e. 8.9 trucks and 2.2 drivers. For example, one might wish to understand if it is possible to reduce logistics costs below some given value, by optimizing the number of hired drivers and trucks. If the fractional solution provides a lower bound greater than this given value, then one can conclude optimization will NOT yield a solution that achieves this target cost.

Nevertheless, we remain interested in finding optimal solutions to IPs in addition to approximate solutions. This is generally a hard problem. Consider, for example, the feasibility question: Does a solution to the IP exist? As illustrated by Fig. 3, one can have bounded or unbounded non-empty feasible sets for the relaxed problem, yet no points exist on the integer grid. In these cases, the feasible set for the IP is empty. Consequently, efficient algorithms based upon the insights of graphical LP are no longer applicable.

Note that a variety of design variables can be represented as integer variables. For example, a decision variable could encode a "yes/no" decision, i.e. a discrete choice. Other examples for integer decision variables include:

- Do I hire this worker $(x = 1)$ or not $(x = 0)$?

- How many cars are allowed in this parking lot everyday: $x = 0, 1, 2, 3, 4, 5, 6$?
- Do I take the first ($x = 1$), second ($x = 2$), or fifth train ($x = 5$)?
- At this intersection, do I take the first left, the second left, the first right?

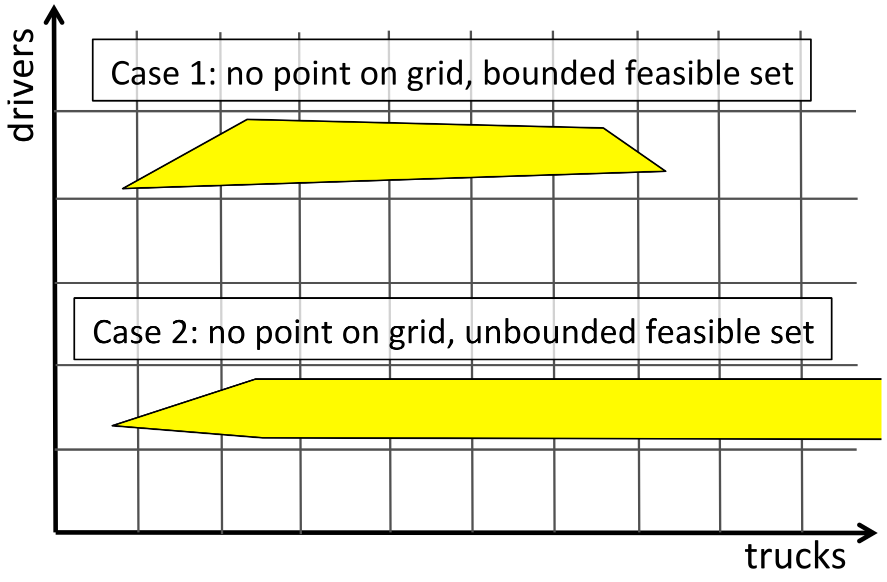
Case 1: no point on grid, bounded feasible set
Case 2: no point on grid, unbounded feasible set
drivers
trucks

Figure 3: Does an integer solution to the problem exist?

Exercise 1. *Consider the linear program*

$$\begin{aligned} \text{min:} \quad f(x) &= -(x_1 + x_2) && (1)\\ \text{s. to:} \quad & 5x_1 + 3x_2 \le 15 && (2)\\ & x_2 \le 3 && (3)\\ & x_1 \ge 0 && (4)\\ & x_1, x_2 \in \mathbb{Z} && (5) \end{aligned}$$

where the last constraint means x_1, x_2 must take integer values.

(i) Determine a fractional solution by solving the relaxed IP graphically. What is the minimum cost? What are the minimizers?
(ii) Determine a lower bound for the IP.

2 Dijkstra's Algorithm & Shortest Path Problem

Recall the shortest path problem. We seek to find the shortest path from node A to node B in the network graph of Fig. 4. Let c_{ij} represent the distance from node i to node j. Let x_{ij} represent whether the edge between nodes i and j is included in the chosen path. That is, $x_{ij} = 1$ for every (i, j) on the chosen path and $x_{ij} = 0$ for every (i, j) NOT on the chosen path. Consequently, x_{ij} can only take integer values, and the shortest path problem can be solved as in IP.

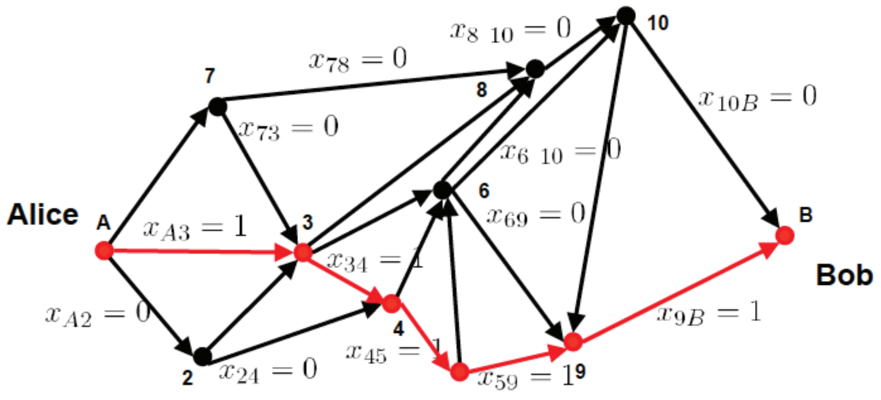

Figure 4: The shortest path problem is to determine the shortest path between nodes A and B. This is, in fact, an integer programming problem.

The shortest path problem contains special structure that allows for efficient IP solutions. In particular, it can be solved by Dijkstra's algorithm. Edsger Dijkstra (1930-2002) was a Dutch computer scientist and professor at the University of Texas, Austin from 1984-2000. Dijkstra's algorithm has worst case computational complexity $\mathcal{O}(|E| + |V| \log |V|)$, where $|E|$ and $|V|$ indicate the number of edges and vertices, respectively. It is the fastest known algorithm for solving shortest path problems on graphs with non-negative edge path costs. It is often used for routing, or as a subroutine in more complex algorithms. We demonstrate Dijkstra's algorithm with the following example.

Consider the directed graph in Fig. 5. We seek to determine the shortest path from node A to any other node. To perform Dijkstra's algorithm, we complete the table in Fig. 6 row-by-row as follows:

Step 1 Start from A. Assign cost to each node, infinity for non-connected nodes. Next consider the node with "shortest path distance" from A, which is B.

Step 2 Through B, compute cumulative cost to each connected node. Remaining costs are unchanged. Non-connected nodes are assigned ∞. Consider the remaining node with "shortest path distance" from A, which is F.

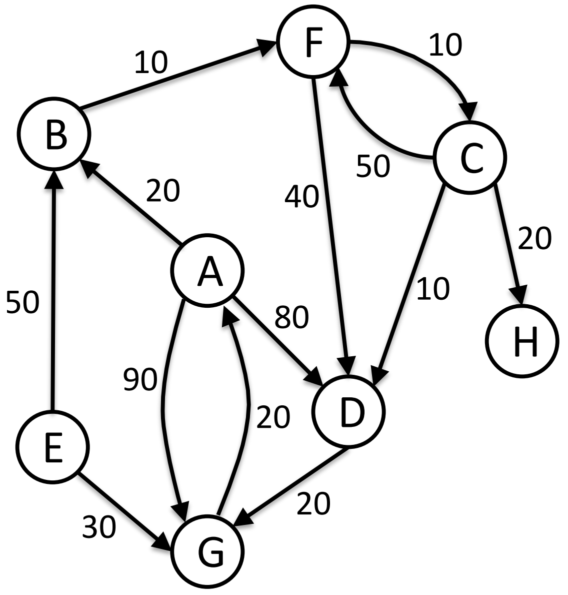

Figure 5: Directed graph for shortest path problem. Determine shortest path from node A to any other node.

A $\rightarrow$	B	C	D	E	F	G	H
(1) A	20	∞	80	∞	∞	90	∞
(2) B	20	∞	80	∞	30	90	∞
(3) F	20	40	70	∞	30	90	∞
(4) C	20	40	50	∞	30	90	60
(5) D	20	40	50	∞	30	70	60
(6) H	20	40	50	∞	30	70	60
(7) G	20	40	50	∞	30	70	60
(8) E	20	40	50	∞	30	70	60

Figure 6: Table for completing Dijkstra's algorithm. The left-most column indicates the final destination node. The numbers indicate the shortest path length, going through node indicated by column header.

Step 3 Through F, compute cumulative cost to each connected node. Consider the remaining node with "shortest path distance" from A, which is C.

Step 4 Through C, compute cumulative cost to each connected node. Consider the remaining node with "shortest path distance" from A, which is D.

Step 5 Through D, compute cumulative cost to each connected node. Consider the remaining node with "shortest path distance" from A, which is H.

Step 6 Through H, compute cumulative cost to each connected node. *Note no nodes connect from H*. Consider the remaining node with "shortest path distance" from A, which is G.

Step 7 Through G, compute cumulative cost to each connected node. Consider the remaining node with "shortest path distance" from A. *Note only E is left, which is unreachable - cost is ∞.*

Step 8 Only E remains. Assign cumulative cost to E as ∞. Remaining costs are unchanged.

Dijkstra's algorithm can be summarized as follows:

1. Pick initial node (A). Shortest-path to (A) is zero.
2. Assign ∞ to non-connected nodes, path length to connected nodes.
3. Consider unfinished node with shortest-path length from (A), denoted $(\cdot)$.
4. Remove $(\cdot)$ from unfinished set. If unfinished set is empty - done.

5. Compute cumulative cost to each connected node through $(\cdot)$, ∞ otherwise.
6. Go back to Step 3.

Exercise 2. *Consider the shortest-path problem from node A to node H in the network shown in Fig. 7. The numbers along the arcs represent path-lengths or costs to traverse between the respective nodes. Solve the shortest-path problem using Dijkstra's algorithm.*
(i) Write a table to solve Dijkstra's Algorithm.
(ii) What is the value of the shortest path cost from node A to H?
(iii) What is the shortest path from node A to H?

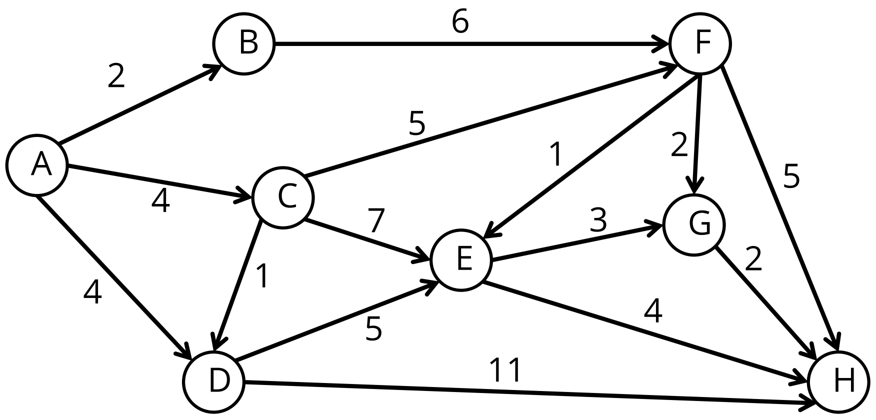

Figure 7: Network for Exercise 2.

3 Branch & Bound

Dijkstra's algorithm provides an efficient method to solve shortest path problems on network graphs. However, we are also interested in solving more general IP problems when no such structure exists, such as the following integer linear program:

$$\begin{aligned} \text{min} \quad & c^T x && (6)\\ \text{s. to} \quad & Ax \leq b && (7)\\ & x \in \mathbb{Z} && (8) \end{aligned}$$

Towards this end, we introduce the Branch & Bound (B&B) algorithm. In essence, B&B solves a sequence of relaxed IP problems which add constraints that bound away from the previous fractional solution. This is performed in a tree structure, where each new relaxed IP constitutes a new branch. It is best described by example.

Example 3.1 (Branch & Bound for a LP)**.** Consider the LP

$$\begin{aligned}
\text{min} \quad & x_1 - 2x_2 && (9)\\
\text{s. to} \quad & -4x_1 + 6x_2 \leq 9 && (10)\\
& x_1 + x_2 \leq 4 && (11)\\
& x_1 \geq 0 && (12)\\
& x_2 \geq 0 && (13)\\
& x_1, x_2 \in \mathbb{Z} && (14)
\end{aligned}$$

P0: First, solve the LP which results from relaxing the integer constraints. We denote this subproblem $P0$. This can be solved graphically, by taking the intersection of the half-spaces defined by the constraints and drawing the objective function isolines, as shown in Fig. 8. This produces the LP solution $x^* = (1.5, 2.5), f^* = -3.5$. In the subsequent steps, we add constraints that *bound away* from the fractional solution.

P1: Add the constraint $x_2 \geq 3$ to $P0$. Call this subproblem $P1$. It is easy to verify the feasible set is empty, and therefore no solution exists as shown in Fig. 9. This problem is infeasible.

P2: Next we consider $P0$ but add constraint $x_2 \leq 2$. As shown in Fig. 10, it is straight-forward to verify the solution is $x^* = (0.75, 2), f^* = -3.25$. This solution still produces a fractional solution. Consequently, we need to bound away from this fractional solution in the following subproblems.

P3: Add constraint $x_1 \leq 0$ to $P2$. It is straight-forward to verify the solution is $x^* = (0, 1.5), f^* = -3$ as shown in Fig. 11.

P4: Add constraint $x_1 \geq 1$ to $P2$. It is straight-forward to verify the solution is $x^* = (1, 2), f^* = -3$ as shown in Fig. 12. We have finally arrived at an integer solution, and the algorithm terminates.

The entire algorithm takes a tree-like structure, as shown in Fig. 13.

Consequently, we summarize the Branch & Bound algorithm as follows:

1. Get bound on optimum by solving relaxed LP
2. For variable with fractional solution, add constraints bounding away
3. Solve subproblem. If infeasible, then stop. If feasible, compute optimum.
4. Go to step 2. Repeat until optimum has integer solution.

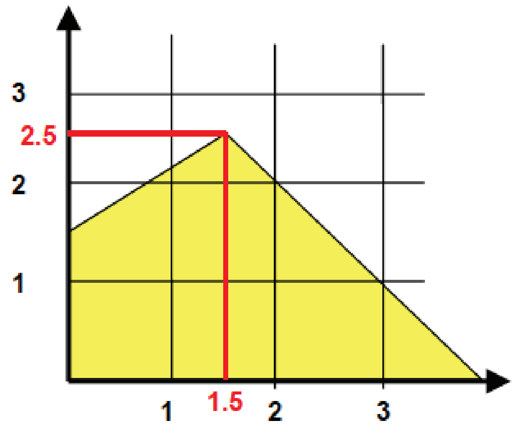

Figure 8: Subproblem $P0$.

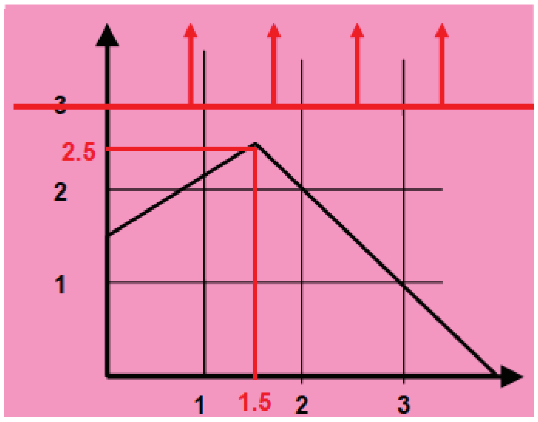

Figure 9: Subproblem $P1$.

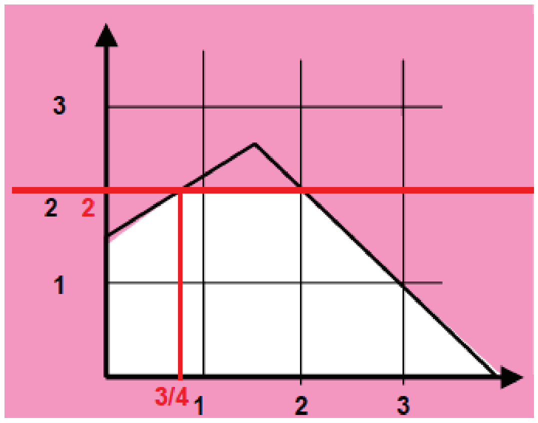

Figure 10: Subproblem $P2$.

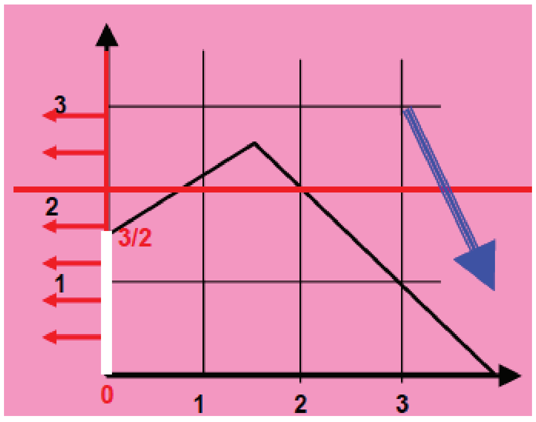

Figure 11: Subproblem $P3$.

Exercise 3. *Consider the linear integer program*

$$
\begin{aligned}
\textit{min:} \quad & f(x) = -(x_1 + x_2) && (15)\\
\textit{s. to:} \quad & 5x_1 + 3x_2 \leq 15 && (16)\\
& x_2 \leq 3 && (17)\\
& x_1 \geq 0 && (18)\\
& x_1, x_2 \in \mathbb{Z} && (19)
\end{aligned}
$$

where the last constraint means x_1, x_2 must take integer values. Solve this problem using the Branch & Bound algorithm. Provide the solution to each subproblem. Draw the tree structure.

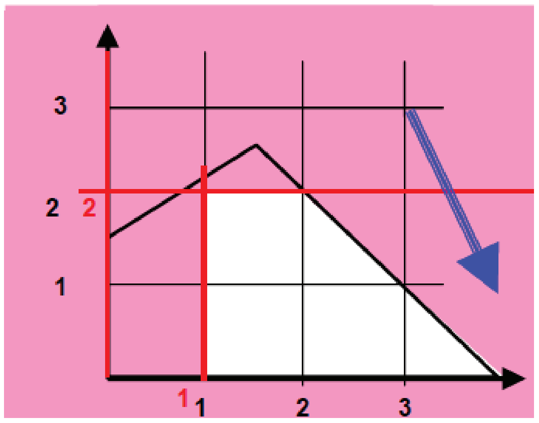

Figure 12: Subproblem $P4$.

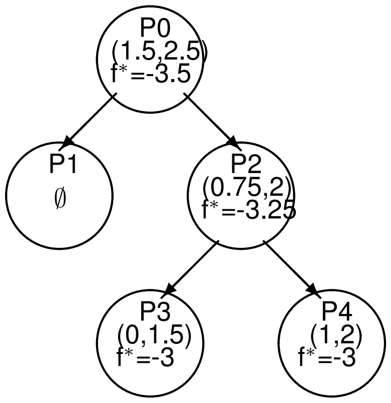

Figure 13: Tree structure of B&B algorithm.

4 Mixed Integer Programming (MIP)

Mixed integer programming (MIP) problems involve design variables in which some are constrained to be integers and others are not. For example, suppose $x = [x_1, x_2, x_3, x_4]^T$. An example mixed integer linear program (MILP) would be

$$\begin{aligned} \text{min} \quad & c^T x && (20)\\ \text{s. to} \quad & Ax \leq b && (21)\\ & x_1, x_2 \in \mathbb{R} && (22)\\ & x_3, x_4 \in \mathbb{Z} && (23) \end{aligned}$$

where x_1, x_2 can take continuous values and x_3, x_4 must take integer values. MIPs can model many civil and environmental engineering system design problems involving both continuous and discrete variables. Consider the three problem classes demonstrated in Fig. 14. A problem with continuous variables and a convex feasible set, as shown on the left, is not modeled by a MIP. In contrast, a system with design variables with mixed integer (drivers, trucks) and continuous variables (concrete mass, steel length) is modeled as a MIP. Interestingly, a problem with continuous variables and a non convex feasible set can be approximated by a discrete number of convex sets. This can be approximated by a MIP. Consequently, MIPs occur in physically and mathematically motivated situations.

4.1 Big M Method

Suppose you are an air traffic controller, managing the arrival of successive aircrafts at an airport. One can think about this problem from two perspectives, illustrated for two aircrafts in Fig. 15.

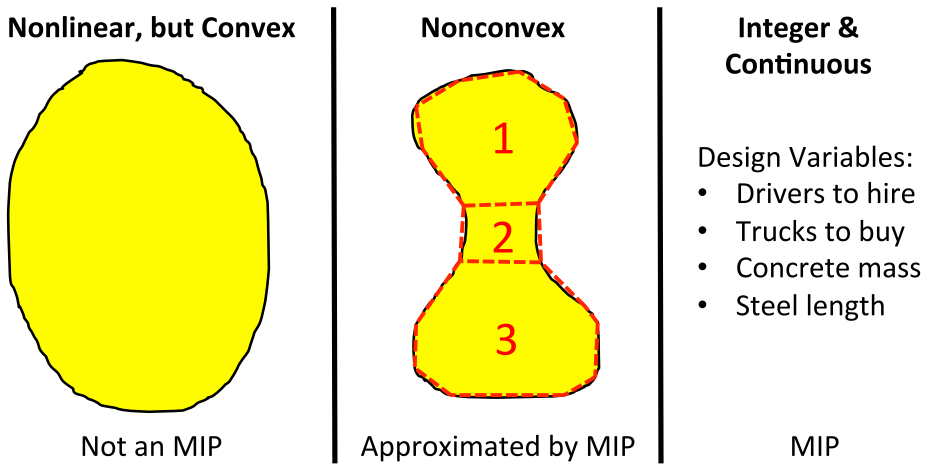

Figure 14: [LEFT] A nonlinear programming problem with a convex feasible set is not an appropriate situation for an MIP. [RIGHT] A mix of design variables which physically make sense as either continuous or discrete variables is an MIP. [CENTER] A nonconvex feasible set can be modeled by a discrete number of convex sets. This can be approximated by an MIP.

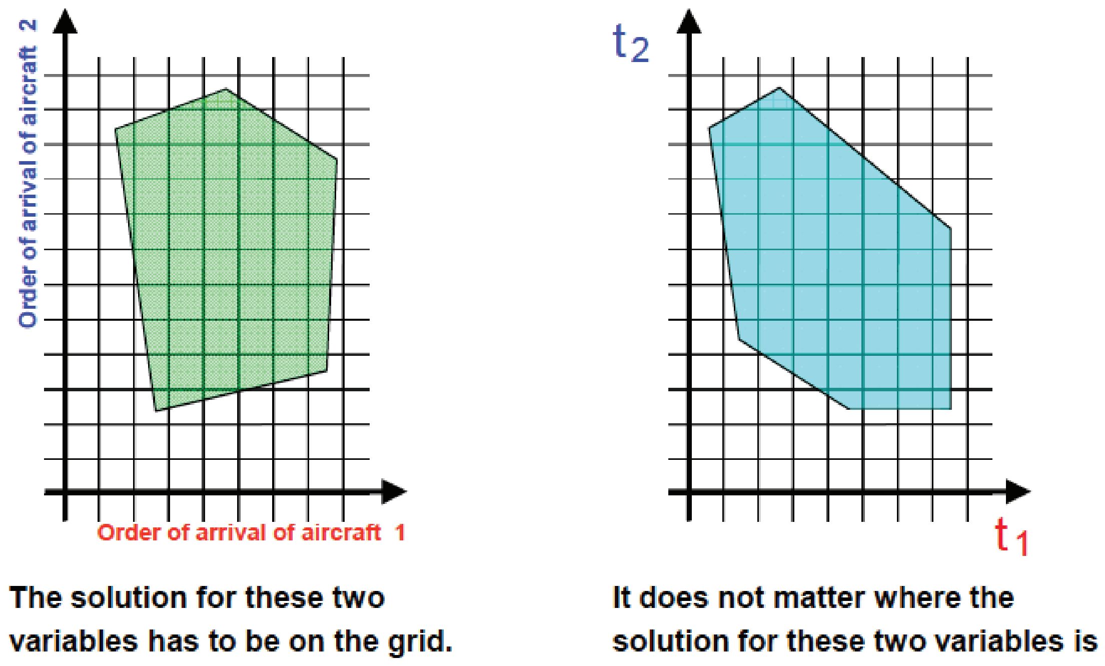
Order of arrival of aircraft 2
Order of arrival of aircraft 1
The solution for these two variables has to be on the grid.
t2
t1
It does not matter where the solution for these two variables is

Figure 15: Two perspectives for optimizing arrival of two aircraft at an airport.

One may consider the *order of arrival*, which is an integer problem shown on the left. One may alternatively consider the time of arrival, which is not an integer problem, as shown on the right.

Suppose that regardless of the arrival order, you must ensure at least $\Delta = 3$ minutes between

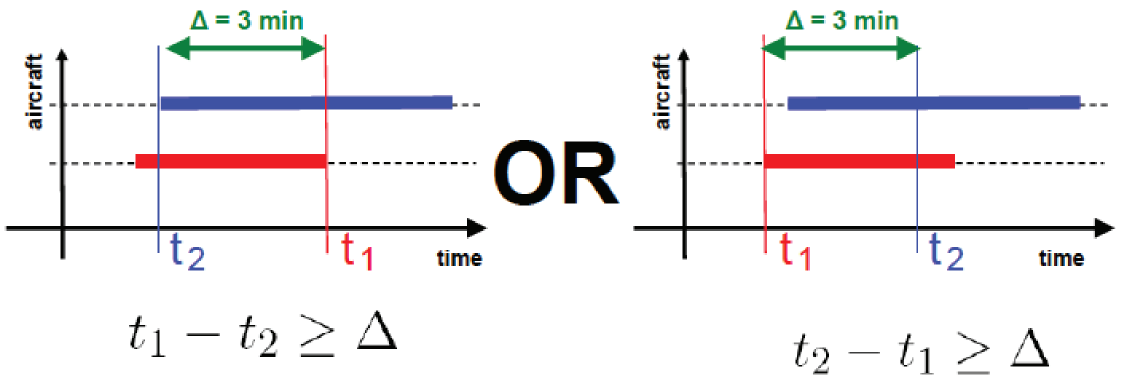

Figure 16: Two perspectives for optimizing arrival of two aircraft at an airport.

each aircraft arrival for safety reasons. Let t_1, t_2 represent the arrival times of aircrafts 1 and 2, respectively. Then regardless of order, the difference between t_1 and t_2 must be greater than Δ, as demonstrated in Fig. 16. That is,

$$|t_1 - t_2| \geq \Delta \tag{24}$$

Absolute values are, in general, not linear nor affine. Consequently they represent a nonlinear problem that is generally difficult to solve. The so-called "Big-M Method" is a solution to this situation.

Absolute values can be expressed as a **logical disjunction**. That is, they are a mathematical representation of an "**OR**" logical operator,

$$\begin{array}{ccc} t_2 - t_1 \geq \Delta & \text{OR} & t_1 - t_2 \geq \Delta \\ \text{if} \quad t_2 \geq t_1 & & \text{otherwise} \end{array} \tag{25}$$

Note that we have already discussed the "**AND**" operator many times, for example in the problem

$$\begin{aligned} \text{min:} \quad f(x_1, x_2) &= c_1 x_1 + a_2 x_2 \\ \text{s. to:} \quad a_1 x_1 &\leq c_{\max} \\ a_2 x_2 &\geq a_{\min} \\ a_1 x_1 + a_2 x_2 &\geq a_{\min} \\ a_2 x_2 &\geq 2a_1 x_1 \\ a_2 x_2 &\leq 2a_1 x_1 \end{aligned}$$

all the constraints are logical "**AND**" constraints. That is, they must all be satisfied simultaneously. The Big-M method transforms an **OR** operator into an **AND** operator, thereby allowing us to express optimization models naturally. In the process an integer variable is created, thus creating a MIP. The Big-M method is given as follows.

Proposition 1 (The Big M Method)**.** *Pick a very large positive number $M \gg 0 \in \mathbb{R}$. Also, consider an integer decision variable $d \in \{0, 1\}$. For sufficiently large M, the following two statements are equivalent:*

$$\textit{Statement 1:} \qquad \textbf{\textit{OR}} \qquad \begin{cases} t_1 - t_2 \geq \Delta & \textit{if } t_1 \geq t_2 \\ t_2 - t_1 \geq \Delta & \textit{o.w.} \end{cases} \tag{26}$$

$$\textit{Statement 2:} \qquad \textbf{\textit{AND}} \qquad \begin{cases} t_1 - t_2 \geq \Delta - Md \\ t_1 - t_2 \leq -\Delta + M(1-d) \end{cases} \tag{27}$$

Proof. We can provide this statement is true, using the following logical explanation. There are two cases to investigate.

Case 1 : $d = 0$ (Order: t_2, t_1, i.e. $t_2 < t_1$)

$$t_1 - t_2 \geq \Delta - Md \quad \rightarrow \quad t_1 - t_2 \geq \Delta \tag{28}$$

$$t_1 - t_2 \leq -\Delta + M(1-d) \quad \rightarrow \quad t_1 - t_2 \leq -\Delta + M \leq M \tag{29}$$

which can be combined to say

$$\Delta \leq t_1 - t_2 \leq M \tag{30}$$

Case 2 : $d = 1$ (Order: t_1, t_2, i.e. $t_1 < t_2$)

$$t_1 - t_2 \geq \Delta - Md \quad \rightarrow \quad t_1 - t_2 \geq \Delta - M \geq -M \tag{31}$$

$$t_1 - t_2 \leq -\Delta + M(1-d) \quad \rightarrow \quad t_1 - t_2 \leq -\Delta \tag{32}$$

which can be combined to say

$$\Delta \leq t_2 - t_1 \leq M \tag{33}$$

which is exactly the logical **OR** statement. □

Consequently, we see if is possible to transform an **OR** condition to an **AND** condition at the expense of creating an added binary variable d. This variable d encodes the **order**. That is,
$d = 0 \rightarrow$ Order : t_2, t_1.
$d = 1 \rightarrow$ Order : t_1, t_2.

5 MIP Example: Air Traffic Control

Example 5.1 (Two Aircrafts)**.** We demonstrate the utility of the Big M method and MIP formulation on two air traffic control examples. Consider two aircraft landing at an airport. You seek to determine the landing times t_1, t_2 such that the last aircraft lands as early as possible. At least Δ minutes must be allowed between landings. Moreover each aircraft $i = 1, 2$ can only land during time interval $t_i \in [a_i, b_i]$. We may formulate this problem as

$$\min_{t_1,t_2,d} : \qquad dt_1 + (1-d)t_2 \tag{34}$$

$$\begin{aligned}
\text{s. to:} \quad t_1 - t_2 &\geq \Delta - Md && (35)\\
t_1 - t_2 &\leq -\Delta + M(1-d) && (36)\\
t_1 &\leq b_1 && (37)\\
t_1 &\geq a_1 && (38)\\
t_2 &\leq b_2 && (39)\\
t_2 &\geq a_2 && (40)\\
d &\in \{0, 1\} && (41)
\end{aligned}$$

Variable $d = 1$ or $d = 0$ if aircraft 1 or 2 lands last, respectively. As a result, integer variable d encodes order. Constraints (35)-(36) encode the minimum separation time between landings. Constraints (37)-(40) encode the possible time windows for landings.

Example 5.2 (Multiple Aircraft and Holding Patterns)**.** Let us now consider a more complex and interesting example in air traffic control. Incoming aircraft can be sent on holding patterns, meaning they can circle above the airport to delay their landing. However, with each holding pattern cycle, the available time window for landing is offset, as shown in Fig. 17. These periodic landing windows have a period given by the holding pattern time, call it T, and vary in length depending on the size and maneuverability of the aircraft.

For concreteness, let us consider four aircraft as shown in Fig. 17, index by $i = 1, 2, 3, 4$. Let $[a_i, b_i]$ represent the feasible arrival time window, without any holding patterns. Then when considering multiple holding patterns, the set of feasible arrival times for the four aircraft are given in Fig. 17. Let $n_i \in \mathbb{Z}$ be an integer decision variable that represents how many holding patterns we instruct for aircraft i. Then we can write the feasible landing time as

$$a_i + n_i T \leq t_i \leq b_i + n_i T, \qquad i = 1, 2, 3, 4 \tag{42}$$

where n_i, t_i are integer and continuous decision variables, respectively. Moreover, suppose we

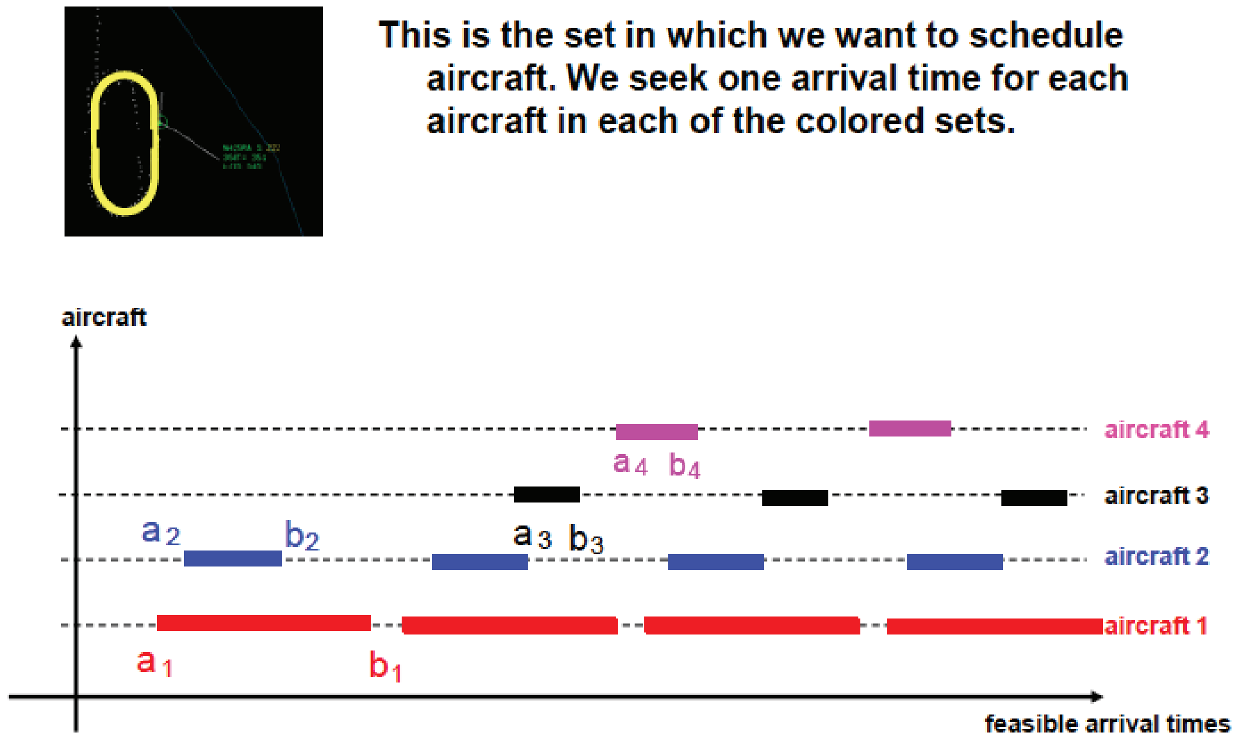
This is the set in which we want to schedule aircraft. We seek one arrival time for each aircraft in each of the colored sets.

Figure 17: Incoming aircraft can be sent on holding patterns of length T, which create periodic windows of feasible arrival times.

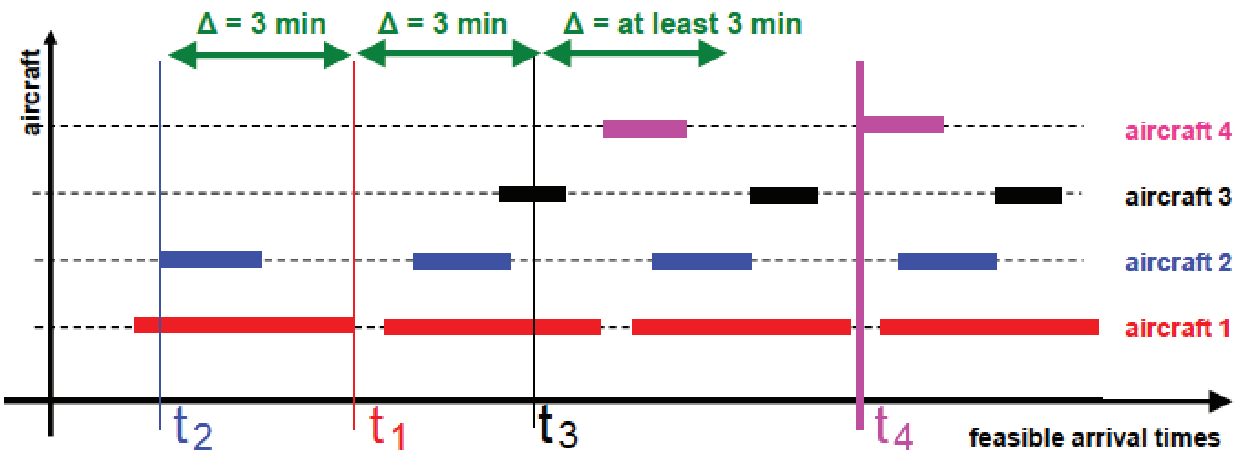

Figure 18: Landing times must be separated by at least $\Delta = 3$ minutes. One possible feasible solution is given here.

require at least $\Delta = 3$ minutes separation between landings. This can be written in **OR** form as

$$|t_i - t_j| \geq \Delta, \qquad \forall\ (i,j) = \{1,2,3,4\}^2,\ \ i > j \tag{43}$$

Alternatively, we can write this in **AND** form using the Big-M method as follows

$$t_i - t_j \geq \Delta - M d_{ij}, \qquad \forall\, (i,j) = \{1,2,3,4\}^2,\ \ i > j, \tag{44}$$

$$t_i - t_j \leq -\Delta + M(1 - d_{ij}), \qquad \forall\, (i,j) = \{1,2,3,4\}^2,\ \ i > j, \tag{45}$$

$$d_{ij} \in \{0,1\}, \qquad \forall\, (i,j) = \{1,2,3,4\}^2,\ \ i > j, \tag{46}$$

where $M \gg 0$ is a very large position number and d_{ij} encodes the relative order between aircraft i and j. One possible feasible solution is demonstrated in Fig. 18, in which the landing times are spaced by at least $\Delta = 3$ minutes. The formulation extends directly from 4 to m aircraft, and can be summarized as

$$\min_{t_i, d_{ij}, n_i} : \qquad \max_i \ t_i \tag{47}$$

$$\begin{array}{llcll}
\text{s. to:} & a_i + n_i T & \leq & t_i \quad i = 1, \cdots, m & \text{[Periodic arrival windows]} \quad (48) \\
& b_i + n_i T & \geq & t_i \quad i = 1, \cdots, m & (49) \\
& n_i & \in & \mathbb{Z} \quad i = 1, \cdots, m & (50) \\
& t_i - t_j & \geq & \Delta - Md, \quad \forall\, (i,j) = \{1, \cdots, m\}^2,\ i > j, & \text{[Min. time b/w landings]} \quad (51) \\
& t_i - t_j & \leq & -\Delta + M(1-d), \quad \forall\, (i,j) = \{1, \cdots, m\}^2,\ i > j, & (52) \\
& d_{ij} & \in & \{0,1\} \quad \forall\, (i,j) = \{1, \cdots, m\}^2,\ i > j, & (53)
\end{array}$$

To summarize, we have continuous decision variables $t_i \in \mathbb{R}$, and integer decision variables $d_{ij} \in \{0,1\}$, $n_i \in \mathbb{Z}$.

6 Notes

There exist solvers specifically designed for mixed integer programs (MIPs), and particularly for mixed integer linear programs (MILPs). In Matlab, the command `intlinprog` solves MILPs. The open source solver `lp_solve` is also very popular for MILPs [1].

A classical textbook on the theory of IPs and combinatorial optimization is written by Nemhauser and Wolsey [2]. An excellent exposition of Dijkstra's algorithm is provided in Ch. 1 of Eric Denardo's textbook [3]. The branch and bound method was first proposed by Land and Doig in 1960 for IP problems [4], and is still the most commonly used algorithm for solving NP-hard problems. The name "branch and bound" first occurred in the work of Little et al. on the traveling salesman problem [5]. The Big M method is a common tool in optimization for converting disjoint feasible sets into joined feasible sets, at the cost of an integer variable.

Applications of mixed integer programming include air traffic scheduling [6], [7], cyber security

[8], robots [9], smart home heating systems [10], and more.

References

[1] M. Berkelaar, K. Eikland, P. Notebaert, et al. (2004) lpsolve: Open source (mixed-integer) linear programming system. Eindhoven U. of Technology. [Online]. Available: http://lpsolve.sourceforge.net/

[2] G. L. Nemhauser and L. A. Wolsey, Integer and combinatorial optimization. Wiley New York, 1988, vol. 18.

[3] E. V. Denardo, Dynamic programming: models and applications. Courier Dover Publications, 2003.

[4] A. H. Land and A. G. Doig, "An automatic method of solving discrete programming problems," Econometrica: Journal of the Econometric Society, pp. 497–520, 1960.

[5] J. D. Little, K. G. Murty, D. W. Sweeney, and C. Karel, "An algorithm for the traveling salesman problem," Operations research, vol. 11, no. 6, pp. 972–989, 1963.

[6] A. Bayen, C. Tomlin, Y. Ye, and J. Zhang, "Milp formulation and polynomial time algorithm for an aircraft scheduling problem," in Decision and Control, 2003. Proceedings. 42nd IEEE Conference on, vol. 5, Dec 2003, pp. 5003–5010 Vol.5.

[7] E. Canepa and C. Claudel, "Exact solutions to traffic density estimation problems involving the lighthill-whitham-richards traffic flow model using mixed integer programming," in Intelligent Transportation Systems (ITSC), 2012 15th International IEEE Conference on, Sept 2012, pp. 832–839.

[8] E. S. Canepa and C. G. Claudel, "Spoofing cyber attack detection in probe-based traffic monitoring systems using mixed integer linear programming," in 2013 International Conference on Computing, Networking and Communications (ICNC). IEEE, 2013, pp. 327–333.

[9] Y. Yin, S. Hosoe, and Z. Luo, "A mixed logic dynamical modeling formulation and optimal control of intelligent robots," Optimization and Engineering, vol. 8, no. 3, pp. 321–340, 2007.

[10] E. Burger, H. Perez, and S. Moura, "Cloud enabled model predictive control of a home heating system," in 2015 American Control Conference, Chicago, IL, USA, July 2015.