An Algebraic Approach
to Geometrical Optimization

Stan Wagon

One of the top 10 algorithms of the 20th century [6] is the simplex algorithm for linear programming (LP). It allows one to maximize or minimize a linear function of several variables, subject to linear constraints. A simple example is illustrated in figure 1: minimize the objective function \( x - 2y \) subject to \( 0 \leq x, \ 0 \leq y, \ 2x + y \leq 5, \) and \( -4x + 4y \leq 5. \) Conceptually, one envisions the polygonal region determined by the constraints and notes that since the objective is linear, its optimal value must occur at a corner of this region. Thus checking a finite number of cases suffices to solve such problems; this is the basis for LP, which works even more efficiently by examining a relatively small set of corners.

LP works extremely well in practice, allowing us to minimize a linear function subject to linear constraints even when the number of variables (and constraints) is in the thousands.

An important variant is integer linear programming (ILP), where we are working not in the realm of rational (or approximate real) numbers, but in the integers. An ILP problem asks for the integer values of the variables that minimize or maximize a linear objective function subject to constraints of linear inequalities. The range of applications of ILP is huge; here we will show how it can be used on some traveling salesman-type problems.

The Traveling Salesman Problem

For the traveling salesman problem (TSP), where one wants the shortest cycle that visits all points in a given planar set, we can use 0-1 valued variables \( x_{ij} \) for each pair of points (i.e., each edge in the complete graph on the point set), the intent being that a value of 1 means that the tour should include the edge. The main constraint is the degree constraint, which forces each point to appear in precisely two edges of the final tour. Thus, the ILP formulation is:

Input: \( n \) points \( P \) in the plane, with distance \( d_{ij} \) defined to be \( || P_i - P_j ||. \)

Variables: \( x_{ij} \) for each pair \( i, j \) with \( i < j. \)

Objective: \( \sum d_{ij} x_{ij}. \)

Constraints: \( x_{ij} \in \{0,1\}, \) and the degree constraints:

for each \( i, \sum_{j=1}^{i-1} x_{ii} + \sum_{j=i+1}^{n} x_{ij} = 2. \)

This is a simple system with roughly \( n^2 \) variables and \( n \) constraints. The catch is that a solution in integers might not yield the desired single cycle: Any union of two or more disjoint cycles will satisfy the degree constraints, and such unions will likely show up in the answer. We would like to add constraints guaranteeing that no cycle with fewer than \( n \) points exists, but the number of such constraints—just under \( 2^n \)—is too large. Following instead the “if it’s not broken, don’t fix it” strategy, we can fix (i.e., break) the cycles as they show up, as opposed to trying to break them all in advance. So we find the cycles \( C_1, C_2, \ldots \) that arise in an ILP solution, and for each \( C = C_k, \) add the cycle-killing constraint that requires at least two edges connecting \( C \) to its complement:
The shortest tour visiting the centroids of the lower 48 states.

The results of the three ILP calls for the traveling salesman problem for the lower 48 states.

Figure 4 shows the best route through all the vertices in a map derived from our campus; seven cycle-breaking steps were used. Note the out-and-back section in the center. Such would be forbidden in a TSP route.

Shortest Hamiltonian Cycle

Often the routing problem is confused with the problem of finding the shortest Hamiltonian cycle (a cycle through the graph that travels on edges, visits each vertex, and never duplicates an edge). A first important distinction is that not all graphs have a
Hamiltonian cycle. But many planar graphs do (Tutte [11] proved that every 4-connected planar graph is Hamiltonian). If a Hamiltonian cycle exists, we can easily find the shortest one. The ILP becomes much simpler since we need variables only for the edges in the graph, of which there are most 6n by Euler’s formula. So we have a system with about n variables and n constraints; we can therefore handle larger sets than the TSP or routing problems (which use about n² variables). Figure 5 shows the best Hamiltonian cycle through our campus.

While an ILP approach cannot be used on very large problems, we can find some nice Hamiltonian tours in medium-sized graphs. Figure 6 shows an example in a 300-vertex graph; computing time was about one minute, with 10 iterations of cycle-breaking.

Other Applications

There are many applications of ILP to real-world problems. One intriguing area is the logistics of arranging kidney exchanges [1, 4]. When a patient and donor are incompatible, they can enter a protocol for kidney exchanges where donor1 gives a kidney to patient2 and donor2 gives one to patient1. Three-way exchanges are also feasible, though less desirable than two-way exchanges. So the setup is a directed graph where each vertex is a donor-patient pair, with a directed edge from one pair to another if the second patient is compatible with the first donor. One then wants to decompose this graph into 2-cycles and 3-cycles so that the maximum number of exchanges is performed and the number of 3-cycles is minimized.

For modest-sized graphs, this can be done with ILP. Set up a variable P_{ij} for each possible pair exchange and a triple T_{ijk} for each possible triple exchange. Constraints are: (1) all variables are either 0 or 1; and (2) conservation of kidneys: for each i,

\[ \sum z \leq 1 \]

where \( z \) runs through all P and T variables that involve i. The objective function is

\[ 2 \sum P_{ij} + 3 \sum T_{ijk} \]

Once the optimal number \( E \) of exchanges is found, one can redo the ILP to minimize the number of triples, but with the additional constraint that the number of exchanges is \( E \).

In a recent discrete applied mathematics course,
all student projects involved ILP. It was used on the facilities location problem (explained below), the kidney exchange problem, the problem of determining whether a soccer team is eliminated from the playoffs where wins earn three points and draws one point (this is known to be \( NP \)-complete [2, 7]), the assignment of students to classes, and the problem of finding a maximum matching in a general graph. For the last, there is a much faster algorithm (the blossom algorithm of Edmonds), but ILP is useful as a check on a blossom implementation.

In the facilities location problem (also known as the \( k \)-median problem), one is given \( n \) points in the plane and wishes to choose \( k \) of them to act as hosts, the idea being that if each point is moved to the nearest host, the total of the \( n - k \) distances is minimized. For an ILP, use 0-1 variables \( x_{ij} \) to indicate that person \( i \) will travel to the house of person \( j \), who is a host, and \( y_j \) to pick out the hosts. Then the constraints are:

\[
0 \leq x_{ij} \leq y_j \leq 1,
\]

which indicates that person \( i \) can go only to the house of a host.

Further Reading

There are various commercial and open-source ILP solvers (e.g., CBC or Symphony by COIN-OR [3]). I use Mathematica, which has good LP and ILP capabilities via its Minimize and Maximize functions.

3. COIN-OR (COmputational INfrastructure for Operations Research), Programs CBC and Symphony; coin-or.org/projects.
How Integer Linear Programming Works

The simplex method for linear programming (LP) is fast and robust, so we can approach an integer linear programming (ILP) problem by repeated calls to LP, adding new constraints each time. Here we will illustrate a basic idea known as branch-and-bound.

We wish to optimize a linear function subject to a set of linear constraints, and we wish to find a solution among only integer values of the variables. A simple example from [10] will convey the idea.

Suppose we wish to minimize the function \( x - 2y \) subject to the constraints \( x, y \in \mathbb{N}, 2x + y \leq 5, \) and \(-4x + 4y \leq 5, \)

We begin by using LP to solve the system in the space of rational (or approximate real) numbers.

See figure 1 on page 22. If the LP solution occurs at integer values of the variables, we are done. Here LP finds the minimum value \(-3.75\) at \( x = 1.25 \) and \( y = 2.5. \) We then choose a variable, say \( x, \) and set up two new LP problems: one with the additional constraint that \( x \geq 2, \) and the other with the constraint that \( x \leq 1. \) That is, we are pushing \( x \) toward the nearest integers. These two problems are placed on a “to do” stack of LP problems. The first tree figure shows the state of the system at this point.

We repeat this process many times, sending the problem at the top of the stack to LP; whenever it produces a solution where one or more variables is a noninteger, say \( x = q, \) we branch into two subproblems, one with the new constraint that \( x \geq [q], \) the other with the new constraint that \( x \leq [q]. \)

The subproblems form a tree, which can have thousands of vertices. There are some choices we must make: the order in which the two new problems are placed on the stack and the choice of branching variable. We might choose the variable whose value is closest to an integer or farthest from an in-
solution in only eight seconds. Such approximations lead to patterns that might well correspond to the optimal placement in the continuous case. In Figure 7 the black lines are the Voronoi diagram of the hosts: the “polygons of nearness” for each host.

Acknowledgment. The author is grateful to Sophors Khut and Tom Halverson for bringing the routing problem to his attention, and Danny Lichtblau and Sándor Fekete for enlightening conversations.

Tree 4. The current best of 2 makes the problem atop the stack unworthy of pursuit.

Tree 5. The fifth call to LP leads to an integer solution with value −3, which defeats the last two remaining problems on the stack and solves the problem.

Another important subtlety is whether to work with rationals or approximate reals (terminating decimals). In this example, the tree is explored in a depth-first fashion from right to left.

Eventually one of the problems is solved with integer values for all the variables, and the minimum value \( B \) (best-so-far) is recorded. In tree 3, \( B = -2 \) is shown in green.

At this point, we take advantage of the fact that the objective \( x - 2y \) assumes an integer value when both variables are integers. This means that only values of \(-3\) or less are candidates for improving on our current best-so-far. As subsequent problems on the stack are sent to LP, rather than branch, we ignore any nodes where LP returns a value greater than \(-3\). These dead-end nodes are shown in red. Once the green \(-2\) is found, there is no need to explore the next node (the \( y \geq 2 \) branch). Then a green \(-3\) is found, telling us that the last two red nodes are dead ends. The answer is therefore \(-3\), realized at \((x, y) = (1, 2)\).

For situations such as the traveling salesman problem, where the objective has real coefficients, one can scale up all the input points, assumed rational, by a large integer and round all distances to the nearest integer.

---

Stan Wagon is a professor of mathematics at Macalester College. He has written many math books, such as The Banach-Tarski Paradox and Mathematica in Action. He is an avid mountaineer, skier, and runner and is a founding editor of Ultrarunning magazine.

Email: wagon@macalester.edu

http://dx.doi.org/10.4169/mathhorizons.19.3.22