Integer Programs

CS 149 Staff

November 23, 2006

1 Introduction

Formulating a production problem as a linear program could lead to a recommendation of producing 7.5 cars and 1.5 trucks. The customer that gets the front half of a car and the back half of a truck is unlikely to be happy. To remedy this, it is natural to add additional constraints to the linear program indicating that certain variables must take only integer values. This is called an integer program (IP). So how powerful are integer programs? It turns out that most NP hard optimization problems can be naturally formulated as integer programs, including the knapsack problem, satisfibility, graph bisection and traveling salesman problem.

Some problems, such as knapsack, can be translated into integer programs with barely any work: $\max p^T x$ subject to $w^T x \leq C$ and $x \in \{0,1\}^n$. Many other problems can be formulated as LPs with some effort. Here are some tricks typically used when translating problems into integer programs.

1.1 Disjunction

Linear programs translating a conjunction into a linear program is easy, since all constraints of a linear program must be satisfied. Many problems, such Boolean satisfibility (SAT), contain disjunctions as well. Consider the constraint $x_1 \vee \neg x_2$ is true. This can easily be translated into $x_1 + (1 - x_2) \geq 1$ with $x_i \in \{0,1\}$. What about disjunctions of expressions, such as $|x| \geq 10$, which is equivalent to $x \geq 10 \vee x \leq -10$?

The solution here is to introduce a 0/1 slack variable s and rewrite the equations as $x + Ms \geq 10$ and $x - (1-s)M \leq -10$, where M is a very large number. If $s = 0$, the first equation is in its original form and the second is trivially satisfied regardless of the value of x. If $s = 1$, the first equation is trivially satisfied and the second is not. Therefore this models disjunction.

1.2 Relying on the force of gravity

Consider the shortest path problem. One natural formulation is as a min cost flow of 1 unit from s to t where every edge has a flow of either zero or one. At first glance this might seem to be insufficient since there could be cyclic flows in addition to the path we're looking for. If all edge weights are positive, any optimal solution

to the LP must be acyclic and therefore this potential problem won't come up with the optimal solutions we care about.

To show that two problems are equivalent, you need to show that given an optimal solution to one of the problems you can find a feasible solution to the other problem (in polynomial time) that is no worse. If you can show this in both directions, then solving one problem implies solving the other.

2 TU

Certain problems that can be formulated as integer programs, such as shortest path, have polynomial-time algorithms even though we have seen that general integer programming is NP hard. It turns out that there is a condition on the matrix of the integer program that allows us to show that any extreme point is integral and therefore the problem can be solved in polynomial time. This condition is *total unimodularity*. A matrix is TU if and only if all sub matrices have determinant zero, one or negative one.

There are many ways to show that a matrix is TU. One can of course prove it directly from the definition, but that is usually quite hard. The slides give a partition theorem which is often easier than using the definition directly but still requires thought. The following condition gives a more or less thoughtless way that can prove whether or not many matrices are TU.

If A is TU then it remains TU if A is modified by:

1. Deleting one or more rows/columns
2. Performing a pivot operation
3. Adding or removing duplicate rows/columns or rows/columns with only one non-zero, which is -1 or 1.
4. Negating rows/columns
5. Permuting Rows or columns
6. Transposed
7. Inverted (assuming square and invertible)

Note that for all of these operations except the first two are if and only if conditions. This implies that given a matrix one wishes to evaluate to see if it is TU, one can repeatedly apply those conditions, simplifying the problem until the answer is obvious. For example, consider the question of determining for what values of X the following matrix is TU:

$$\begin{matrix} -1 & -1 & -1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & X & 1 & 0 \end{matrix}$$

The last column has only one nonzero, so it is necessary and sufficient to prove that the matrix with that row eliminated is TU. After that elimination, the second row can be eliminated for the same reason. Then negating the first row yields:

$$\begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & X & 1 \end{matrix}$$

The first and second rows are identical, so the first can be removed. The resulting first and third columns are identical, so the third can be removed, leaving:

$$\begin{matrix} 1 & 1 \\ 1 & X \end{matrix}$$

The determinant of this matrix can now be calculated directly as $X - 1$ (the one-element submatrices are obviously ok iff $x \in \{-1, 0, 1\}$). Therefore if $X \in \{0, 1\}$ the original matrix is TU but if $X = -1$ the original matrix is not TU. Note that because these conditions are if and only if, the last matrix is TU if and only if the original matrix is TU. This sort of "backwards" reasoning would be incorrect if irreversible steps (not if and only if) were used.