

Mixed Integer Linear Programs
Course notes for Optimization
Spring 2005

Peter Bro Miltersen

March 3, 2005

Version 1.2

1 Total Unimodularity

We have seen that the max flow problem is a special case of the min cost flow problem which is again a special case of linear programming. Thus, an LP solver should be sufficient to tackle any of the problem we have seen so far.

However, if we use a general LP solver based on, say, the simplex method to solve max flow instances or min cost flow instances rather than an implementation of the Ford-Fulkerson method or Klein's algorithm, it *seems* that we may lose an important property of the latter algorithms, namely the *integrality property*: If the Ford-Fulkerson algorithm or Klein's algorithm are given integer-valued inputs, the solutions returned are also guaranteed to be integer-valued. We saw previously that this integrality property is often very important when modelling combinatorial optimization instances as max flow or min cost flow instances. A good example is the reduction of maximum cardinality matching to max flow: There, we constructed a reduction r converting bipartite graphs G into max flow instances $r(G)$ with the property that any matching of size v in G leads to a flow of value v in $r(G)$ and any flow *with an integer valued flow along every arc* of value v in $r(G)$ leads to a matching of size v in G . This reduction enables us to use any max flow algorithm with the integrality property, such as Ford-Fulkerson, to find maximum cardinality matchings. Suppose we now instead try to use the simplex algorithm to find a maximum matching in a bipartite graph by expressing the appropriate max flow instance as a linear program. The value of the solution found will give us the *cardinality* of a maximum matching as desired, but it is not *a priori* clear that we can use the solution to extract the matching *itself*. Indeed, this only seems to be the case if we can prove an integrality theorem for the simplex algorithm. Fortunately, it turns out that we *can* prove such an integrality theorem for a special class of linear programs which will turn out to include the linear programs that express max flow and min cost flow instances. The relevant property of the simplex algorithm is that it is guaranteed to return not only an optimal solution, but an optimal *basic* solution and the relevant property of network flow problems is that the matrices of the corresponding linear programs are *totally unimodular*.

Definition A matrix A is called *totally unimodular* if every square submatrix¹ of A has determinant either 0, 1 or -1 .

We are going to need two basic lemmas which will be useful for proving that

¹An $r \times r$ submatrix of an $m \times n$ matrix A is obtained by first throwing away all but r rows of A and then throwing away all but r columns of the resulting $r \times n$ matrix

a given matrix is totally unimodular.

Lemma 1 *Let A be an $n \times m$ totally unimodular matrix, I_m be the $m \times m$ identity matrix and I_n be the $n \times n$ identity matrix. Then, the $n \times (m+n)$ matrices $[A \ I_n]$ and $[A \ -I_n]$ and the $(n+m) \times n$ matrices $\begin{bmatrix} A \\ I_m \end{bmatrix}$ and $\begin{bmatrix} A \\ -I_m \end{bmatrix}$ are also totally unimodular.*

Proof We prove the lemma only for the matrix $[A \ I_n]$, the other three cases being completely analogous.

So assume that $[A \ I_n]$ is not totally unimodular. Then, let B be the *smallest* square submatrix of $[A \ I_n]$ for which $\det(B) \notin \{0, 1, -1\}$ (i.e., any square proper submatrix of B has a determinant in $\{0, 1, -1\}$).

Since A is totally unimodular, we must have that B contains entries from I_n . Suppose first that all such entries are 0. Then, an entire column of B is 0 and thus B has determinant 0, a contradiction. Thus, we conclude that we can assume that some entry b_{ij} of B is taken from I_n and is non-zero, i.e., $b_{ij} = 1$. We now use the fact that for any matrix B and any j we have $\det(B) = \sum_{k=1}^n (-1)^{k+j} b_{kj} \det(B^{(kj)})$, where $B^{(ij)}$ is the matrix resulting from removing the i 'th column and the j 'th row from B . In our case, we have $\det(B) = \sum_{k=1}^n (-1)^{k+j} b_{kj} \det(B^{(kj)}) = (-1)^{i+j} \det(B^{(ij)})$. But then, $B^{(ij)}$ is a proper submatrix of B which does not have its determinant in $\{0, 1, -1\}$, a contradiction. **QED**

Lemma 2 *If A is a matrix with entries from $\{0, 1, -1\}$ with the property that each column contains at most two non-zero entries, at most one being 1 and at most one being -1 , then A is totally unimodular.*

Proof Assume not. Then, let A be a smallest matrix (in terms of number of entries) not satisfying the claim and let B be the smallest square submatrix of A for which $\det(B) \notin \{0, 1, -1\}$. Look at the leftmost column of B . We consider three cases.

- All entries in the leftmost column of B are 0. Then, the determinant of B is 0, a contradiction.
- Only one entry in the leftmost column, say b_{i1} of B is non-zero. We may then write $\det(B) = \sum_{k=1}^n (-1)^{k+1} b_{k1} \det(B^{(k1)}) = (-1)^{i+1} b_{i1} \det(B^{(i1)})$ and conclude that the submatrix $B^{(i1)}$ contradicts the minimality of B .

- The leftmost column of B contains a 1-entry, say b_{i_1} and a (-1) -entry, say b_{j_1} . The rest of the entries of the leftmost column of B are then 0, by the assumption about A . We now appeal to the fact that for any matrix B , $\det(B) = \det(B')$ where B' is the result of replacing some row of B with the sum of that row and another row of B . In particular, we let C be the matrix resulting from replacing the j 'th row of B with the sum of the i 'th row and the j 'th row. We may now write $\det(B) = \det(C) = \sum_{k=1}^n (-1)^{k+1} c_{k1} \det(C^{(k1)}) = (-1)^{i+1} c_{i1} \det(C^{(i1)}) = (-1)^{i+1} \det(C^{(i1)})$. But $C^{(i1)}$ also has the property that it has at most two non-zero entries in each column, at most one being 1 and at most one being -1 and must have a determinant outside of $\{-1, 0, 1\}$ if B does, contradicting the minimality of A .

QED

We are now ready to prove the main theorem of this section.

Theorem 3 (Integrality theorem for totally unimodular linear programs)

Let P be a linear program in standard form with constraints $Ax \leq b, x \geq 0$ where A is an $n \times m$ totally unimodular matrix and $b \in \mathbf{Z}^m$ an integral vector. Every basic feasible solution $x \in \mathbf{R}^n$ to P is integer-valued, i.e., that we have $x \in \mathbf{Z}^n$.

Proof Any basic feasible solution to $Ax \leq b, x \geq 0$ may be obtained by introducing slack variables $x_{n+1}, x_{n+2}, \dots, x_{n+m}$, setting n variables to 0 in the system

$$x_{n+i} = b_i - \sum_{j=1}^n a_{ij}x_j, i = 1..m \tag{1}$$

and solving (uniquely) for the remaining m (basic) variables.

In matrix form, the system (1) can be written

$$b = [A \ I]x' \tag{2}$$

where $x' \in \mathbf{R}^{m+n}$ is the vector containing all original variables and slack variables.

Setting n variables in (2) to 0, we obtain a system

$$\tilde{A}\tilde{x} = \tilde{b}, \tag{3}$$

where \tilde{x} is the vector of basic variables in the basic solution we are considering, $\tilde{A} \in \mathbf{Z}^{m \times m}$ is a square submatrix of $[A \ I]$ and $\tilde{b} \in \mathbf{Z}^m$ consists of entries from b and zero-entries.

As the system (3) has a unique solution, \tilde{A} must be non-singular. Thus, since A is totally unimodular and applying Lemma 1, we have that $\det \tilde{A}$ is either 1 or -1 . We now apply *Cramer's rule* for solving the system $\tilde{A}\tilde{x} = \tilde{b}$: The value of any of the basic variable \tilde{x}_j in the solution is of the form $\tilde{x}_j = \det B^{(j)} / \det \tilde{A}$, where $B^{(j)}$ is the result of replacing the j 'th column of \tilde{A} with \tilde{b} . Since $\det \tilde{A}$ is either -1 and 1 and since every entry in $B^{(j)}$ is an integer, this value is an integer. Since the value of any non-basic variable in the solution is, by definition, 0, the basic solution is thus integer-valued.

QED

We finally show that we may phrase the min cost flow problem as a linear program in standard form in such a way that the resulting constraint matrix is totally unimodular. The max flow case is covered by the reduction from max flow to min cost flow or we may do it directly in a similar way; this is left as an exercise.

Given a min cost flow instance (V, E, c, b, k) where $V = \{1, 2, \dots, n\}$, $\forall u, v : k(u, v) = -k(v, u)$ and $\sum_{v \in V} b(v) = 0$. Recall that we can express the instance as the following linear program, P :

P : Find $x_{u,v}, u, v \in V$ minimizing $\sum_{u < v} k(u, v)x_{u,v}$ so that

$$\begin{aligned} \sum_{j \in V} x_{i,j} &= b(i) & \forall i \in V \\ x_{i,j} &\leq c(i, j) & \forall i, j \in V \\ x_{i,j} &= -x_{j,i} & \forall i, j \in V \end{aligned}$$

This program is not in standard form; in particular the variables may be negative, so we need to rewrite it. We define $y_{i,j} = \max(0, x_{i,j})$ (i.e., we replace negative flow values with the value 0) and reformulate the linear program in terms of the new non-negative variables as follows:

P' : Find $y_{u,v} \geq 0, u, v \in V$ minimizing $\sum_{u,v} k(u, v)y_{u,v}$ so that

$$\begin{aligned} \sum_{j \in V} (y_{i,j} - y_{j,i}) &= b(i) & \forall i \in V \\ y_{i,j} &\leq \max(0, c(i, j)) & \forall i, j \in V \\ -y_{i,j} &\leq c(j, i) & \forall i, j \in V \end{aligned}$$

This is still not a program in standard form, as the last constraints are equalities, rather than inequalities. However, we may observe that since $\sum b(i) = 0$, any solution satisfying $\forall i \in V : \sum_{j \in V} (y_{i,j} - y_{j,i}) \leq b(i)$, must in fact satisfy $\forall i \in V : \sum_{j \in V} (y_{i,j} - y_{j,i}) = b(i)$.

Thus, the program

P'' : Find $y_{u,v} \geq 0, u, v \in V$ minimizing $\sum_{u,v} k(u,v)y_{u,v}$ so that

$$\begin{aligned} \sum_{j \in V} (y_{i,j} - y_{j,i}) &\leq b(i) && \forall i \in V \\ y_{i,j} &\leq \max(0, c(i,j)) && \forall i, j \in V \\ -y_{i,j} &\leq c(j,i) && \forall i, j \in V \end{aligned}$$

is a linear program in standard form, correctly capturing the min cost flow instance.

In matrix form, we can write the constraints of P'' as $Ay \leq e, y \geq 0$ (we write e rather than b as we already used the symbol b to denote the balance vector), where the matrix A can be written

$$A = \begin{bmatrix} A' \\ I \\ -I \end{bmatrix}$$

where I is a square identity matrix. Combining Lemma 1 and Lemma 2, we see that the matrix A is totally unimodular, as desired. Thus, if we solve the program P'' by the simplex method, the integrality theorem provides us with a guarantee that the optimal solution found will be integer valued.

Of course, we might choose to solve our linear program by methods other than the simplex algorithm. In particular, we might prefer an interior point algorithm. If we do, we may conceivably have a solver that is guaranteed to return optimal solutions but not necessarily *basic* optimal solutions. If this is the case, we can easily build a “shell” around the solver that converts any non-basic optimal solution found to a basic optimal solution and Theorem 3 again applies. How to do this is left as Exercise 2.

2 Mixed integer linear programming and the travelling salesman problem

Theorem 3 takes care of the integrality property for the special class of linear programs corresponding to flow instances. However, being able to impose integrality constraints on the solutions we seek is relevant for many other optimization tasks that can be expressed using linear constraints and for these, Theorem 3 may not apply. Therefore, we shall look at a generalization of linear programming that allows us to explicitly express such integrality constraints:

A *Mixed Integer Linear Program* (MILP) is an optimization instance of the following kind: Find a point $x \in \mathbf{R}^n$ maximizing (or minimizing) a linear form $\langle c, x \rangle$ so that a given set of linear inequalities and/or equations *and* a given set of *integrality constraints* $x_i \in \mathbf{Z}$ are satisfied. A point x satisfying the constraints is called a *feasible solution* to the program.

Thus, the *only* addition to the LP formalism we make is the ability to require certain components of the solution to be integer valued. We shall see, however, that this seemingly modest added expressiveness in general makes the task of finding the optimal solution painfully harder. In particular, all algorithms for solving MILPs have exponential worst case complexity and unlike the case of the simplex algorithm which is also exponential but only on artificially constructed instances, the exponential running time of any of the MILP solvers we know is *typical* behavior which we have to live with.

An important special case of Mixed Integer Linear Programs is the case where we add the requirement $x_i \in \mathbf{Z}$ for *all* $i \in \{1, 2, \dots, n\}$, i.e, we look for a solution $x \in \mathbf{Z}^n$. Such a program is called an *Integer Linear Program* (ILP). Our remarks about the lack of worst-case efficient algorithms for solving MILPs also applies to this special case.

We shall study a concrete and very important example of expressing an optimization problem in the Integer Linear Program formalism. The *Traveling Salesman Problem* (TSP) is the following celebrated² optimization problem. An instance to the travelling salesman is given by a finite set of *cities* $V = \{0, 1, 2, \dots, n - 1\}$ and real values $d_{i,j} \geq 0$ for $i, j \in \{0, 1, \dots, n - 1\}$ and $i \neq j$. The value $d_{i,j}$ is called the *distance* between city i and city j . Our task is to find a permutation π of the cities, so that the $\sum_{i=1}^n d_{\pi(i), \pi((i+1) \bmod n)}$ is minimized. Equivalently, if we make a complete weighted directed graph with V being the set of vertices and $d_{i,j}$ being the weight of the arc from i to j , we ask to find the lightest simple directed cycle in the graph covering all vertices. The permutation is also called a *travelling salesman tour* and the sum to be minimized is called the *total length* of the tour.

We may interpret the problem in the following way: If V is thought of as a set of actual cities on a map and $d_{u,v}$ is the actual Euclidean distance between city u and city v (instances for which this is the case are called Euclidean instances and form an important special case of the travelling

²Perhaps the travelling salesman problem is the most well studied optimization problem in terms of number of papers and in terms of man-years spent on researching it. If you want to attack an optimization problem *without* consulting the literature (which is actually never a good idea) *at least* do not choose this one as you'd thereby maximize the probability of making a fool out of yourself! Unfortunately (and amazingly), every year new groups of researchers or students fail to follow this fairly obvious piece of advice!

salesman problem), we ask to find a route for a travelling salesman that will bring him through all the cities, make him end up in the city he starts in and minimize the total distance travelled.

We shall show that an instance of the travelling salesman problem can be formalized as an instance of integer linear programming. This formalization will also show us how to use the integer constraints of the ILP formalism in a fruitful and non-trivial way. Our integer linear program is going to have $n^2 - n$ integer variables $x_{i,j}$ for $i, j \in \{0, 1, \dots, n - 1\}$ and $i \neq j$. We are further going to constrain each $x_{i,j}$ to take values between 0 and 1 and, $x_{i,j}$ being an integer, this means that 0 and 1 are then the only possible values. The intention is to let the variables represent *Boolean* values, with 0 representing **false** and 1 representing **true**. With this interpretation, we want $x_{i,j}$ to indicate whether the travelling salesman should visit the city j right after he visits i . That is, if π is the tour that our solution x is meant to represent, we intend to have $x_{i,j} = 1$ iff $\exists k : i = \pi(k) \wedge j = \pi((k + 1) \bmod n)$. Our first attempt at an ILP formalization is then the following integer linear program.

P : Find $x_{i,j} \in \mathbf{Z}$, $i, j \in \{0, 1, \dots, n - 1\}, i \neq j$, minimizing $\sum d_{i,j}x_{i,j}$ and satisfying

$$\begin{aligned} x_{i,j} &\geq 0 \quad \forall i, j \\ x_{i,j} &\leq 1 \quad \forall i, j \\ \sum_{j \neq i} x_{i,j} &= 1 \quad \forall i \\ \sum_{j \neq i} x_{j,i} &= 1 \quad \forall i \end{aligned}$$

The constraints $\sum_{j \neq i} x_{i,j} = 1$ and $\sum_{j \neq i} x_{j,i} = 1$ express that the salesman visits exactly one city just before and exactly one city just after he visits a given city. For any permutation π , there is a solution x to P with objective function value equal to the total length of the tour, namely the solution x with $x_{i,j} = 1$ if and only if $\exists k : i = \pi(k) \wedge j = \pi((k + 1) \bmod n)$. If one could prove the converse implication, we would have a formalization of the travelling salesman problem already. However, this is not the case: There are solutions x to P that do not correspond to a tour, for instance the solution with $x_{0,1} = x_{1,0} = 1$ and $x_{2,3} = x_{3,4} = \dots = x_{n,(n-1)} = x_{n,2} = 1$ and all other $x_{i,j} = 0$. In the graph interpretation, we see that this solution corresponds to a “tour” consisting of two separate cycles, one visiting the cities 0 and 1 and a separate one visiting the rest of the cities and hence it does not correspond to a feasible solution to an actual tour that should contain only one cycle. To rule out such solutions, we need to add some extra constraints. There are several ways of doing this. A particularly clean option is to first add the restriction to the original problem that $\pi(0) = 0$, i.e., we assume that the

first city visited is 0. As the salesman is going through a cycle, this is clearly without loss of generality. We then add to our integer linear program integer variables u_i for $i = 0, 1, 2, \dots, n-1$ with the intention that if π is the tour corresponding to the solution (x, u) , then we have $\pi(u_i) = i$, i.e., we want $u_i = j$ to mean that i is the j 'th city visited by the travelling salesman. If we can now express the constraint that for all cities i and all cities j *except* $j = 0$, we have $x_{i,j} = 1 \Rightarrow u_j \geq u_i + 1$, then we rule out malformed solutions such as the one above as we rule out any cycle not containing the special city 0 as part of the solution. Since we know that $x_{i,j}$ is either 0 or 1 and u_i and u_j are between 0 and $n-1$, the statement $x_{i,j} = 1 \Rightarrow u_j \geq u_i + 1$ can be expressed by a linear inequality in the following elegant way: $u_i - u_j + nx_{i,j} \leq n-1$. This leads to the following program.

P' : Find $x_{i,j} \in \mathbf{Z}$, $i, j \in \{0, 1, \dots, n-1\}$, $i \neq j$, and $u_i \in \mathbf{Z}$, $i \in \{0, 1, \dots, n-1\}$ minimizing $\sum d_{i,j}x_{i,j}$ and satisfying

$$\begin{aligned}
x_{i,j} &\geq 0 && \forall i, j \\
x_{i,j} &\leq 1 && \forall i, j \\
\sum_{j \neq i} x_{i,j} &= 1 && \forall i \\
\sum_{j \neq i} x_{j,i} &= 1 && \forall i \\
u_0 &= 0 \\
u_i &\geq 0 && \forall i \\
u_i &\leq n-1 && \forall i \\
u_i - u_j + nx_{i,j} &\leq n-1 && \forall i \in \{0, 1, 2, \dots, n-1\}, j \in \{1, 2, 3, \dots, n-1\}
\end{aligned}$$

In fact, the requirements that $u_0 = 0$ and u_i are between 0 and $n-1$ turn out not to be necessary and we get the following more succinct program.

P'' : Find $x_{i,j} \in \mathbf{Z}$, $i, j \in \{0, 1, \dots, n-1\}$, $i \neq j$, and $u_i \in \mathbf{Z}$, $i \in \{0, 1, \dots, n-1\}$ minimizing $\sum d_{i,j}x_{i,j}$ and satisfying

$$\begin{aligned}
x_{i,j} &\geq 0 && \forall i, j \\
x_{i,j} &\leq 1 && \forall i, j \\
\sum_{j \neq i} x_{i,j} &= 1 && \forall i \\
\sum_{j \neq i} x_{j,i} &= 1 && \forall i \\
u_i - u_j + nx_{i,j} &\leq n-1 && \forall i \in \{0, 1, 2, \dots, n-1\}, j \in \{1, 2, 3, \dots, n-1\}
\end{aligned}$$

The formal statement that we have now correctly captured TSP is the following:

Fact 4 *Given any feasible solution x to P'' of value v , we can easily find a travelling salesman tour π of length v . Conversely, given any travelling salesman tour π of length v , we can easily find a feasible solution x to P'' of value v .*

A proof can be made along the lines of our discussion preceding the definition of P'' and is left as Exercise 4.

Thus, TSP reduces to ILP. The tricks of using integrality constraints to specify Boolean variables and translating logical constraints on these into inequalities are very general. In fact, we shall prove in the course “Combinatorial Search” that this makes integer linear programming a *universal* language for expressing search and optimization problems: We shall define what we mean by a “simple” search problem (capturing, in an adequate way all the problems we have studied so far, and more), formalize this notion as the class of problems **NP**, and show that all problems in **NP** can be formalized as Integer Linear Programs. We shall refer to the latter fact as Integer Linear Programming being **NP-complete**. Thus, an efficient algorithm for solving ILPs would be extremely useful. Unfortunately, as already stated we do not know of an asymptotically worst-case efficient algorithm for ILP. Furthermore, we believe that there *is* no such algorithm for solving general integer linear programs, *precisely* because it is such a general language for expressing search and optimization problems. The existence of a worst case efficient algorithm would seem “too good to be true”, though we do not have a proof that it does not exist. This point will be discussed in detail in the course “Combinatorial Search”.

Much more surprisingly, we shall see that the Travelling Salesman Problem, which *seems* to be a very special case of integer linear programming and unlike ILP and MILP not very useful as a “programming language” in which to phrase other problems in fact has the *same* property: TSP is *also* **NP-complete**, i.e., we can “translate” instances of arbitrary “simple” search problems into Travelling Salesman instances and thus a worst case efficient algorithm for this problem would also seem “too good to be true” and hence we do not believe that it exists. Even more importantly, this property of the Travelling Salesman Problem is also possessed by literally thousands of other concrete problems for which we would like an efficient solution. On the surface, these problems look very similar to the problems we were able to solve efficiently using the flow or LP formalisms but the **NP-completeness** theory shall give us strong evidence that *none* of these problems have algorithms that are efficient in the worst case. This is the real power and relevance of the theory we shall develop.

3 Exercises

Exercise 1 Show how to express a max flow instance as a totally unimodular

linear program.

Exercise 2 Suppose you are given a linear program P in standard form *and* a feasible solution x to P . Show how to easily (that is, without running the simplex algorithm!) compute a *basic* feasible solution x^* to P with a value of the objective function at least as good as the value of x (In particular, if x is optimal, then so is x^*).

Exercise 3 Give examples of optimization instances in the exercises of Chvatal that were modelled using linear programs, but where (mixed) integer linear programs really would have been more appropriate. Why was the use of the LP formalism still acceptable in those cases?

Exercise 4 Prove Fact 4.

Exercise 5 Show that we still capture the TSP if we change P'' into a mixed integer linear program by allowing the u_i variables (but not the $x_{i,j}$ variables) to take non-integer values.

Exercise 6 The *Maximum Independent Set Problem* is the following problem: Given an undirected graph $G = (V, E)$, find a subset S of V , as big as possible, so that no two vertices in S are adjacent (i.e., connected by an edge). Show how to phrase this maximization problem as an integer linear program.

Exercise 7 The *Graph Coloring Problem* is the following problem: Given an undirected graph $G = (V, E)$, color the vertices of V with the fewest possible number of colors so that no two adjacent vertices are assigned the same color. Show how to phrase this minimization problem as an integer linear program.

Hint: Represent colors by integers. For each vertex u , you should have an integer variable x_u denoting its color. Also, you are going to need to invent some additional variables in order to express the desired constraints on the colors, similar to the way we added the u_i variables to express TSP.