## Linear Programming Duality, Network Flows, and Reductions

### 1. Linear Programming Duality

Recall the max-flow, min-cut theorem: a cut is a set of nodes containing $S$ but not $T$, and the *capacity* of this cut is the sum of the capacities of the edges going out of this set. The max-flow, min-cut theorem says that the capacity of the smallest cut is exactly equal to the maximum flow that can be pushed from $S$ to $T$. Indeed, it is the existence of such a cut that establishes that the flow that simplex finds is optimal.

As it turns out, the max-flow min-cut theorem is a special case of a more general phenomenon called *duality*. Basically, duality means that a maximization and a minimization problem have the property that any feasible solution of the min problem is greater than or equal any feasible solution of the max problem (see Figure). Furthermore, and more importantly, *they have teh same optimum*.

Consider the network shown in the Figure below, and the corresponding max-flow problem. We know that it can be written as a linear program as follows:

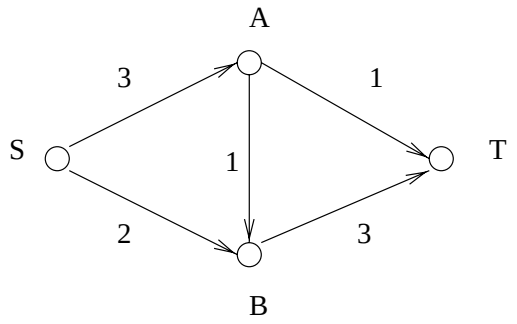


Figure 1: A simple max-flow problem

$$
\begin{array}{lllllll}
\max f_{SA} & +f_{SB} & & & & & \\
f_{SA} & & & & & \leq 3 & \\
 & f_{SB} & & & & \leq 2 & \\
 & & f_{AB} & & & \leq 1 & \\
 & & & f_{AT} & & \leq 1 & P \\
 & & & & f_{BT} & \leq 3 & \\
f_{SA} & & -f_{AB} & -f_{AT} & & \bar{0} & \\
 & f_{SA} & +f_{AB} & & -f_{BT} & \bar{0} & \\
 & & & & & f \geq 0 &
\end{array}
$$

Consider now the following linear program:

$$
\begin{array}{lllllll}
\min 3y_{SA} & +2y_{SB} & +y_{AB} & +y_{AT} & +3y_{BT} & & \\
y_{SA} & & & & +u_A & & \geq 1 \\
 & y_{SB} & & & & +u_B & \geq 1 \\
 & & y_{AB} & & -u_A & +u_B & \geq 0 \quad D \\
 & & & y_{AT} & -u_A & & \geq 0 \\
 & & & & y_{BT} & -u_B & \geq 0 \\
 & & & & & & y \geq 0
\end{array}
$$

This LP describes the min-cut problem! To see why, suppose that the $u_A$ variable is meant to be 1 if $A$ is in the cut with $S$, and 0 otherwise, and similarly for $B$ (naturally, by the definition of a cut, $S$ will always be with $S$ in the cut, and $T$ will never be with $S$). Each of the $y$ variables is to be 1 if the corresponding edge contributes to the cut capacity, and 0

otherwise. Then the constraints make sure that these variables behave exactly as they should. For example, the second constraint states that *if $A$ is not with $S$, then $SA$ must be added to the cut.* The third one states that *if $A$ is with $S$ and $B$ is not* (this is the only case in which the sum $-u_A + u_B$ becomes $-1$), *then $AB$ must contribute to the cut.* And so on. Although the $y$ and $u$'s are free to take values larger than one, they will be "slammed" by the minimization down to 1 or 0.

Let us now make a remarkable observation: These two programs have strikingly symmetric, *dual*, structure. Each variable of $P$ corresponds to a constraint of $D$, and vice-versa. Equality constraints correpond to unrestricted variables (the $u$'s), and inequality constraints to restricted variables. Minimization becomes maximization. The matrices are transpose of one another, and the roles of right-hand side and objective function are interchanged.

Such LP's are called *dual* to each other. It is mechanical, given an LP, to form its dual: *Transpose the matrix, invert maximization to minimization and vice-versa, interchange the roles of the right-hand side and the objective function, and introduce a nonnegative variable for each inequality, and an unrestricted one for each equality.*

By the max-flow min-cut theorem, the two LP's $P$ and $D$ above have the same optimum. *In fact, this is true for general dual LP's!* This is the *duality theorem*, which can be stated as follows (we shall not prove it; the best proof comes from the simplex algorithm, very much as the max-flow min-cut theorem comes from the max-flow algorithm):

*If an LP has a bounded optimum, then so does its dual, and the two optimal values coincide.*

## 2. Matching

It is often useful to *compose* reductions. That is, we can reduce a problem A to B, and B to C, and since C we know how to solve, we end up solving A. A good example is the matching problem.

Suppose that the *bipartite* graph shown in Figure 6 records the compatibility relation between four boys and four girls. We seek a maximum matching, that is, a set of edges that is as large as possible, and in which no two edges share a node. For example, in the figure below there is a *complete* matching (a matching that involves all nodes).

To reduce this problem to max-flow we do this: We create a new source and a new sink, connect the source with all boys and all girls with the sinks, and direct all edges of the original bipartite graph from the boys to the girls. All edges have capacity one. It is easy to see that the maximum flow in this network corresponds to the maximum matching.

Well, the situation is slightly more complicated than was stated above: What is easy to see is that the optimum *integer-valued* flow corresponds to the optimum matching. We would be at a loss interpreting as a matching a flow that ships .7 units along the edge Al-Eve! Fortunately, what the algorithm in the previous section establishes is that *if the capacities are integers, then the maximum flow is integer.* This is because we only deal with integers throughout the algorithm. Hence *integrality comes for free in the max-flow problem.*

Unfortunately, max-flow is about the only problem for which integrality comes for free. It is a very difficult problem to find the optimum solution (or *any* solution) of a general linear program with the additional constraint that (some or all of) the variables be integers. To see why, notice that the NP-complete *satisfiability* problem can be reduced to *integer linear programming* as this problem is called: The clause $(x_1 \vee \overline{x_2} \vee x_3)$ can be represented by the constraints

$$x_1 + (1 - x_2) + x_3 \geq 1, \quad , 0 \geq x_i \geq 1, \quad x_i \text{integers}.$$

Repeating for all clauses of a given Boolean formula, we get an integer linear program in which finding any feasible solution is equivalent to solving the original instance of satisfiability!

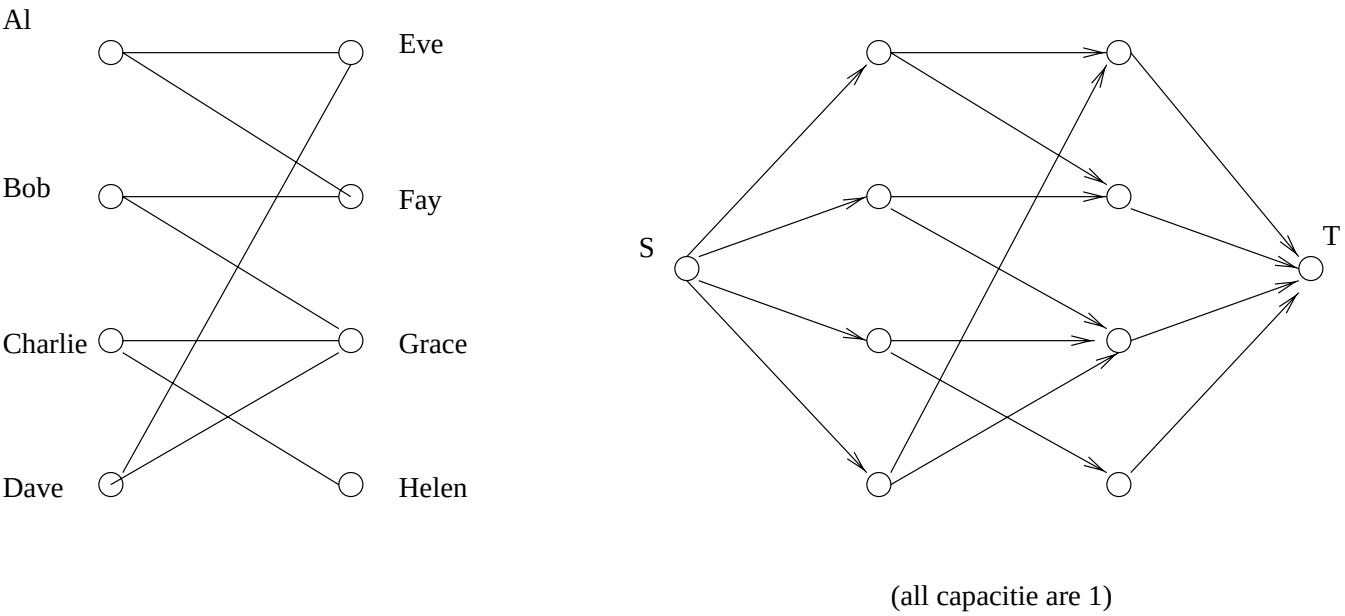


Figure 2: Reduction from matching to max-flow