1.1.1 Hall's Theorem

Hall's theorem gives a necessary and sufficient condition for a bipartite graph to have a matching which saturates (or matches) all vertices of A (i.e. a matching of size $|A|$).

Theorem 1.5 (Hall 1935) *Given a bipartite graph $G = (V, E)$ with bipartition A, B ($V = A \cup B$), G has a matching of size $|A|$ if and only if for every $S \subseteq A$ we have $|N(S)| \geq |S|$, where $N(S) = \{b \in B : \exists a \in S \text{ with } (a, b) \in E\}$.*

Clearly, the condition given in Hall's theorem is necessary; its sufficiency can be derived from König's theorem.

Exercise 1-7. Deduce Hall's theorem from König's theorem.

Exercise 1-8. Consider a bipartite graph $G = (V, E)$ with bipartition (A, B). For $X \subseteq A$, define $\text{def}(X) = |X| - |N(X)|$ where $N(X) = \{b \in B : \exists a \in X \text{ with } (a, b) \in E\}$. Let

$$\text{def}_{max} = \max_{X \subseteq A} \text{def}(X).$$

Since $\text{def}(\emptyset) = 0$, we have $\text{def}_{max} \geq 0$.

1. Generalize Hall's theorem by showing that the maximum size of a matching in a bipartite graph G equals $|A| - \text{def}_{max}$.
2. For any 2 subsets $X, Y \subseteq A$, show that

$$\text{def}(X \cup Y) + \text{def}(X \cap Y) \geq \text{def}(X) + \text{def}(Y).$$

1.2 Minimum weight perfect matching

By assigning infinite costs to the edges not present, one can assume that the bipartite graph is complete. The minimum cost (weight) *perfect* matching problem is often described by the following story: There are n jobs to be processed on n machines or computers and one would like to process exactly one job per machine such that the total cost of processing the jobs is minimized. Formally, we are given costs $c_{ij} \in \mathbb{R} \cup \{\infty\}$ for every $i \in A, j \in B$ and the goal is to find a perfect matching M minimizing $\sum_{(i,j) \in M} c_{ij}$.

In these notes, we present an algorithm for this problem which is based upon linear programming, and we will take this opportunity to illustrate several important concepts in linear programming. These concepts will be formalized and generalized in a subsequent chapter.

The first algorithm given for the assignment problem was given by Kuhn [1955], but he showed only finiteness of the algorithm. A refined version was given by Jim Munkres [1957], and showed a polynomial running time. An algorithm is polynomial-time if its running time (the number of basic operations to run it) is upper bounded by a polynomial in the *size of*

the input (i.e. the number of bits needed to represent the input). Munkres' analysis even shows that the algorithm is *strongly polynomial*, and this means that the running time is polynomial in the *number of numbers* involved (i.e. does not depend on the size of the costs c_{ij}). In this algorithm, the number of operations is upper bounded by $O(n^3)$ where $n = |V|$.

The algorithm is often called the Hungarian method, as it relies on ideas developed by Hungarians, and especially König and Egerváry. In 2006, it was discovered that the method had actually been discovered in the 19th century by Jacobi and this was posthumously published in 1890 in Latin, see http://www.lix.polytechnique.fr/∼ollivier/JACOBI/jacobiEngl.htm.

We start by giving a formulation of the problem as an *integer program*, i.e. an optimization problem in which the variables are restricted to integer values and the constraints and the objective function are linear as a function of these variables. We first need to associate a point to every matching. For this purpose, given a matching M, let its *incidence vector* be x where $x_{ij} = 1$ if $(i, j) \in M$ and 0 otherwise. One can formulate the minimum weight perfect matching problem as follows:

$$\begin{aligned} \text{Min} \quad & \sum_{i,j} c_{ij} x_{ij} & \\ \text{subject to:} & & \\ & \sum_j x_{ij} = 1 & i \in A \\ & \sum_i x_{ij} = 1 & j \in B \\ & x_{ij} \geq 0 & i \in A, j \in B \\ & x_{ij} \text{ integer} & i \in A, j \in B. \end{aligned}$$

This is not a linear program, but a so-called integer program. Notice that any solution to this integer program corresponds to a matching and therefore this is a valid formulation of the minimum weight perfect matching problem in bipartite graphs.

Consider now the linear program (P) obtained by dropping the integrality constraints:

$$(P) \qquad \begin{aligned} \text{Min} \quad & \sum_{i,j} c_{ij} x_{ij} & \\ \text{subject to:} & & \\ & \sum_j x_{ij} = 1 & i \in A \\ & \sum_i x_{ij} = 1 & j \in B \\ & x_{ij} \geq 0 & i \in A, j \in B. \end{aligned}$$

This is the linear programming *relaxation* of the above integer program. In a linear program, the variables can take fractional values and therefore there are many feasible solutions to the set of constraints above which do not correspond to matchings. But we only care

about the *optimum* solutions. The set of feasible solutions to the constraints in (P) forms a *bounded polyhedron* or *polytope*, and when we optimize a linear constraint over a polytope, the optimum will be attained at one of the "corners" or *extreme points* of the polytope. An extreme point x of a set Q is an element $x \in Q$ which cannot be written as $\lambda y + (1-\lambda)z$ with $0 < \lambda < 1$, $y, z \in Q$ with $y \neq z$. (This will be formalized and discussed in greater depth in the chapter on polyhedral combinatorics.)

In general, even if all the coefficients of the constraint matrix in a linear program are either 0 or 1, the extreme points of a linear program are not guaranteed to have all coordinates integral (this is of no surprise since the general integer programming problem is NP-hard, while linear programming is polynomially solvable). As a result, in general, there is no guarantee that the value Z_{IP} of an integer program is equal to the value Z_{LP} of its LP relaxation. However, since the integer program is *more* constrained than the relaxation, we always have that $Z_{IP} \geq Z_{LP}$, implying that Z_{LP} is a lower bound on Z_{IP} for a minimization problem. Moreover, if an optimum solution to a linear programming relaxation is integral (in our case, that would imply it is the incidence vector of a perfect matching) then it must also be an optimum solution to the integer program.

Exercise 1-9. Prove this last claim.

Exercise 1-10. Give an example of an integer program where $Z_{IP} \neq Z_{LP}$.

However, in the case of the perfect matching problem, the constraint matrix has a very special form and one can show the following very important result.

Theorem 1.6 *Any extreme point of (P) is a 0-1 vector and, hence, is the incidence vector of a perfect matching.*

Because of the above theorem, the polytope

$$\begin{aligned} P \quad = \quad \{x : \ & \sum_j x_{ij} = 1 && i \in A \\ & \sum_i x_{ij} = 1 && j \in B \\ & x_{ij} \geq 0 && i \in A, j \in B\} \end{aligned}$$

is called the *bipartite perfect matching polytope*.

To demonstrate the beauty of matchings, we shall give two completely different proofs of this result, one purely algorithmic here and one purely algebraic in the chapter on polyhedral theory. The algebraic proof is related to the notion of *totally unimodularity*.

To prove it algorithmically, we describe an algorithm for solving the minimum weight perfect matching problem. The algorithm is "primal-dual". To explain what this means, we need to introduce the notion of duality of linear programs, and let's do it in the specific case of our bipartite matching problem. Suppose we have values u_i for $i \in A$ and v_j for $j \in B$

such that $u_i + v_j \leq c_{ij}$ for all $i \in A$ and $j \in B$. Then for any perfect matching M, we have that

$$\sum_{(i,j)\in M} c_{ij} \geq \sum_{i\in A} u_i + \sum_{j\in B} v_j. \tag{1}$$

Thus, $\sum_{i\in A} u_i + \sum_{j\in B} v_j$ is a *lower bound* on the cost of the minimum cost perfect matching (for bipartite graphs). To get the best lower bound, we would like to maximize this quantity, and therefore we obtain another linear program

$$\begin{aligned} &\text{Max} && \sum_{i\in A} u_i + \sum_{j\in B} v_j && \\ &\text{subject to:} && && \\ (D) \qquad & && u_i + v_j \leq c_{ij} && i \in A, j \in B. \end{aligned}$$

The constraints can be interpreted as $w_{ij} \geq 0$ where $w_{ij} = c_{ij} - u_i - v_j$. This is a linear program, call it (D). So far, we have shown that this linear program (D) provides a lower bound on the cost of any perfect matching, but we can even prove that it provides a lower bound on the value of any solution to the linear program (P). Indeed consider any $x \in P$. We have

$$\begin{aligned} \sum_{i\in A}\sum_{j\in B} c_{ij}x_{ij} &\geq \sum_{i\in A}\sum_{j\in B}(u_i + v_j)x_{ij} \\ &= \left(\sum_{i\in A} u_i \sum_{j\in B} x_{ij}\right) + \left(\sum_{j\in B} v_j \sum_{i\in A} xij\right) \\ &= \sum_{i\in A} u_i + \sum_{j\in B} v_j, \end{aligned}$$

because of the constraints that x satisfy. (D) is the *dual* linear program in the sense of linear programming duality.

In summary, so far, we know that

$$\left[\min_{\text{perfect matchings } M} \sum_{(i,j)\in M} c_{ij}\right] \geq \left[\min_{x\in P} \sum_{(i,j)} c_{ij}x_{ij}\right] \geq \left[\max_{(u,v)\in D} \sum_{i\in A} u_i + \sum_{j\in B} v_j\right].$$

If, for any instance, we could always find a feasible solution u, v to (D) and a perfect matching M such that we have equality in (1) (i.e. the cost of the perfect matching is equal to the value of the dual solution) then we would know that we have equality throughout, that the matching found is *optimum*, and that furthermore, the incidence vector of the matching M is optimum for the linear program (P). Given a solution u, v to the dual, a perfect matching M would satisfy equality if it contains only edges (i, j) such that $w_{ij} = c_{ij} - u_i - v_j = 0$. This is what is referred to as *complementary slackness*. However, for a given u, v, we may not be able to find a *perfect* matching among the edges with $w_{ij} = 0$.

The algorithm performs a series of iterations to obtain an appropriate u and v. It always maintains a dual feasible solution and tries to find an "almost" primal feasible solution x satisfying complementary slackness. The fact that complementary slackness is imposed is crucial in any primal-dual algorithm. In fact, the most important (and elegant) algorithms in combinatorial optimization are primal-dual. This is one of the most important tool for designing efficient algorithms for combinatorial optimization problems (for problems which, of course, admit such efficient solutions).

More precisely, the algorithm works as follows. It first starts with any dual feasible solution, say $u_i = 0$ for all i and $v_j = \min_i c_{ij}$ for all j. In a given iteration, the algorithm has a dual feasible solution (u, v) or say (u, v, w). Imposing complementary slackness means that we are interested in matchings which are subgraphs of $E = \{(i, j) : w_{ij} = 0\}$. If E has a perfect matching then the incidence vector of that matching is a feasible solution in (P) and satisfies complementary slackness with the current dual solution and, hence, must be optimal. To check whether E has a perfect matching, one can use the cardinality matching algorithm developed earlier in these notes. If the maximum matching output is not perfect then the algorithm will use information from the optimum vertex cover C^* to update the dual solution in such a way that the value of the dual solution increases (we are maximizing in the dual).

In particular, if L is as in the previous section then there is no edge of E between $A \cap L$ and $B - L$. In other words, for every $i \in (A \cap L)$ and every $j \in (B - L)$, we have $w_{ij} > 0$. Let

$$\delta = \min_{i \in (A \cap L), j \in (B-L)} w_{ij}.$$

By the above argument, $\delta > 0$. The dual solution is updated as follows:

$$u_i = \begin{cases} u_i & i \in A - L \\ u_i + \delta & i \in A \cap L \end{cases}$$

and

$$v_j = \begin{cases} v_j & i \in B - L \\ v_j - \delta & j \in B \cap L \end{cases}$$

One easily check that this dual solution is feasible, in the sense that the corresponding vector w satisfies $w_{ij} \geq 0$ for all i and j. What is the value of the new dual solution? The difference between the values of the new dual solution and the old dual solution is equal to:

$$\delta(|A \cap L| - |B \cap L|) = \delta(|A \cap L| + |A - L| - |A - L| - |B \cap L|) = \delta(\frac{n}{2} - |C^*|),$$

where A has size $n/2$ and C^* is the optimum vertex cover for the bipartite graph with edge set E. But by assumption $|C^*| < \frac{n}{2}$, implying that the value of the dual solution strictly increases.

One repeats this procedure until the algorithm terminates. At that point, we have an incidence vector of a perfect matching and also a dual feasible solution which satisfy complementary slackness. They must therefore be optimal and this proves the existence of

an *integral* optimum solution to (P). Since, by carefully choosing the cost function, one can make any extreme point be the *unique* optimum solution to the linear program (this will be formally proved in the polyhedral chapter), this shows that any extreme point is integral and hence this proves Theorem 1.6.

Of course, as some of the readers might have noticed, the proof is not complete yet since one needs to prove that the algorithm indeed terminates. This can be proved by noticing that at least one more vertex of B must be reachable from an exposed vertex of A (and no vertex of B becomes unreachable), since an edge $e = (i, j)$ with $i \in (A \cap L)$ and $j \in B - L$ now has $w_{ij} = 0$ by our choice of δ. This also gives an estimate of the number of iterations. In at most $n/2$ iterations, all vertices of B are reachable or the matching found has increased by at least one unit. Therefore, after $O(n^2)$ iterations, the matching found is perfect. The overall running time of the algorithm is thus $O(n^4)$ since it takes $O(n^2)$ to compute the set L in each iteration. By looking more closely at how vertices get labelled between two increases of the size of the matching, one can reduce the running time analysis to $O(n^3)$.

Exercise 1-11. Check that the running time of the algorithm is indeed $O(n^3)$.

Example: Consider the instance given by the following cost matrix defined on a bipartite graph with 5 vertices on each side of the bipartition:

0	2	7	2	3
1	3	9	3	3
1	3	3	1	2
4	0	1	0	2
0	0	3	0	0

Assume that $u^T = (2, 3, 0, -2, 0)$ and $v^T = (-2, 0, 3, 0, 0)$. The set E of edges with $w_{ij} = 0$ corresponds exactly to the set of edges in Figure 1.1. The maximum cardinality matching algorithm finds the matching $(1, 9)$, $(2, 6)$, $(3, 8)$ and $(5, 7)$, and the set of labelled vertices is $\{3, 4, 8\}$. We compute δ as

$$\delta = \min_{i \in \{3,4\}, j \in \{6,7,9,10\}} w_{ij} = 1$$

corresponding to the edge $(3, 9)$. The new vectors u and v are $u^T = (2, 3, 1, -1, 0)$ and $v^T = (-2, 0, 2, 0, 0)$. The value of the dual solution has increased from 4 units to 5. The corresponding set E now has a perfect matching, namely $(1, 6)$, $(2, 7)$, $(3, 9)$, $(4, 8)$ and $(5, 10)$ of cost 5. Both the matching and the dual solution are optimal.

Exercises

Exercise 1-12. Consider a bipartite graph $G = (V, E)$ in which every vertex has degree k (a so-called k-regular bipartite graph). Prove that such a graph always has a perfect matching in two different ways: