Algorithms II	September 30, 2025

Lecture 7: Approximation Algorithms

Notes by Ola Svensson[1]

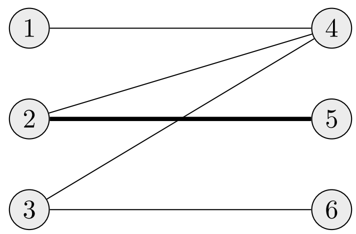
We first finish the formal analysis of the Hungarian algorithm, and then talk about settings in which relaxing integrality has a cost, i.e. about approximation algorithms.

1 The Hungarian Algorithm

We start with some intuition before we give the formal description of the algorithm.

1.1 Intuition

Consider the following instance of the min-cost perfect matching problem:

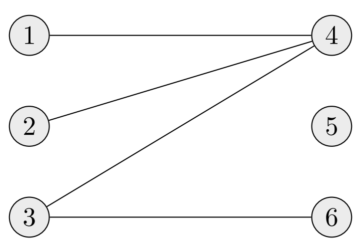


The thin edges have cost 1, whereas the thick edge has cost 2. The Hungarian algorithm will use Lemma 2 in the following way: we will maintain a dual solution u, v that is *feasible at all times*. Then, for a fixed dual solution, the lemma tells us that our perfect matching is only allowed to contain edges that are *tight*, i.e., edges $e = (a, b)$ for which $u_a + v_b = c(e)$. This reduces our problem to finding any perfect matching in the subgraph consisting only of tight edges, i.e., in the graph (V, E') where $E' = \{e = (a, b) \in E : u_a + v_b = c(e)\}$. *Intuitively, we have thus reduced our weighted problem to an unweighted one!*

Let us return to our example. We initialize our procedure with the trivial dual solution

$$v_b = 0, \quad u_a = \min_{b \in B} c_{ab}.$$

(We could have also started from $u = v = 0$.) So in the example, $v_4 = v_5 = v_6 = 0$ and $u_1 = u_2 = u_3 = 1$. The set E' of tight edges is thus:



We then try to find a perfect matching in this graph using e.g. the augmenting path algorithm we saw in Lecture 3. However, the considered graph has *no* perfect matching! This is because we started with a poor lower bound (dual solution). So we will use the fact that there is no perfect matching to improve our dual solution. We make use of *Hall's condition*, which you will prove in the exercise session:

[1]**Disclaimer:** These notes were written as notes for the lecturer. They have not been peer-reviewed and may contain inconsistent notation, typos, and omit citations of relevant works.

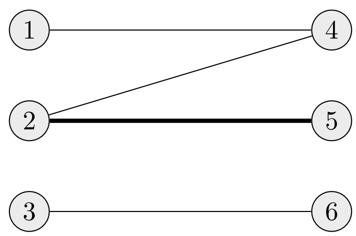
Theorem 1 (Hall's Theorem) *An n-by-n bipartite graph $G = (A \cup B, E')$ has a perfect matching if and only if $|S| \leq |N(S)|$ for all $S \subseteq A$.*

Here $N(S) = \{b \in B : \text{there is an } a \in S \text{ such that } (a,b) \in E'\}$ denotes the neighborhood of the vertices in S.

In the above example, we have $S = \{1, 2\}$ and $N(S) = \{4\}$, which violates Hall's condition (and thus there is no perfect matching). We now use the set S to improve our dual lower bound.

We gradually increase u_a for every $a \in S$ and at the same time decrease v_b for $b \in N(S)$ at the same rate. Let us see what happens to all edges in E'. Notice that the tight edges between S and $N(S)$ will remain tight. Similarly, the tight edges between $A \setminus S$ and $B \setminus N(S)$ will remain tight. Any tight edges between $A \setminus S$ and $N(S)$ will stop being tight. Finally, by definition, there are no edges from S to $B \setminus N(S)$ initially. We continue to gradually change the dual solution until some such edge becomes tight.

In the above example, this will result in updating $u_1 = u_2 = 2$ and $v_4 = -1$. We have thus increased two variables by one unit and decreased one variable by one unit. In total, the dual lower bound was thus increased by one. The set E' of tight edges with respect to the new dual solution is now



Our (augmenting-path) algorithm will now find a perfect matching in this graph, which is optimal by Lemma 2 from last lecture, restated below

Lemma 2 *A perfect matching M is of minimum cost iff there is a feasible dual solution u, v such that*

$$u_a + v_b = c(e) \qquad \text{for every } e = (a,b) \in M.$$

In summary, our algorithm always maintains a dual feasible solution. We then solve the *unweighted* perfect matching problem on the edges allowed by Lemma 2. In the case of failure, we update the dual lower bound to a strictly better lower bound and repeat.

1.2 Formal description

Idea: Maintain a feasible dual solution (u, v). Try to construct a feasible primal solution that satisfies complementarity slackness (Lemma 2) and is thus optimal.

Algorithm:

- **Initialization:**

$$v_b = 0, \quad u_a = \min_{b \in B} c_{ab}.$$

- **Iterative step:**
  - consider $G' = (A \cup B, E')$ where $E' = \{e = (a,b) \in E : u_a + v_b = c(e)\}$ (E' is the set of all tight edges);

- find a maximum-cardinality matching in G':
  * if it is a perfect matching, then we are done (this is a primal feasible solution and it satisfies complementarity slackness, because we consider only the edges in E' and all of them satisfy slackness by construction),
  * otherwise the algorithm finds a set $S \subseteq A$ s.t. $|S| > |N(S)|$ (which is guaranteed to exist by Hall's theorem)
- we can choose a small $\varepsilon > 0$ and improve the dual solution:

$$u'_a = \begin{cases} u_a + \epsilon & \text{if } a \in S, \\ u_a & \text{if } a \notin S, \end{cases}$$

$$v'_b = \begin{cases} v_b - \epsilon & \text{if } b \in N(S), \\ v_b & \text{if } b \notin N(S), \end{cases}$$

  which remains dual feasible because:
  * edges in $S \times N(S)$ are unchanged $(+\varepsilon - \varepsilon)$,
  * edges in $(A \setminus S) \times (B \setminus N(S))$ are unchanged,
  * edges in $(A \setminus S) \times N(S)$ are decreased by ε,
  * edges in $S \times (B \setminus N(S))$ are increased by ε but they were not tight (they were not in E'),
- the dual value increases by $(|S| - |N(S)|)\varepsilon$;
- to make as much progress as possible (and to get a new tight edge), we should choose ε as large as possible for which the dual remains feasible, that is

$$\varepsilon = \min_{e=(a,b) \in S \times (B \setminus N(S))} c(e) - u_a - v_b > 0.$$

The above algorithm can be implemented to run in time $O(n^3)$. (This is quite non-trivial to see. It is somewhat easier to implement in $O(n^4)$.)

1.3 An alternative proof that the bipartite perfect matching polytope is integral

We have shown that for any cost function c we can obtain a minimum-cost perfect matching and it will be integral. By carefully choosing the cost function, one can make any extreme point of the polytope to be the unique optimum solution to the minimum-cost perfect matching problem. This shows that all extreme points are integral, i.e., it is an integral polytope. See Figure 1 for an example.

2 When Relaxing Integrality is Not for Free: Approximation Algorithms

Thousands (or millions :)) of optimization problems have been proved to be NP-hard, which means that finding optimal solutions for them is very likely to be intractable. More specifically, unless P = NP, there is no algorithm for an NP-hard optimization problem that satisfies all of the following three desiderata:

1. The algorithm is efficient (runs in polynomial time).
2. The algorithm is reliable (works for any input instance).
3. The algorithm finds an optimal solution (optimality).

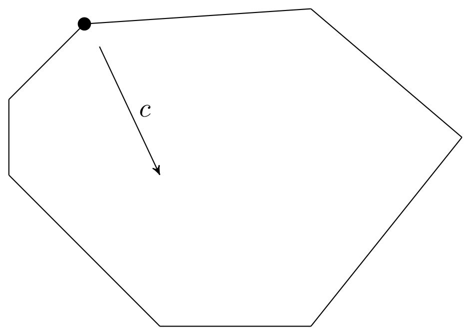


Figure 1: Example of a perfect matching polytope and a cost function vector c for the highlighted vertex.

Therefore, when confronted which such problems, we need to relax one of the above conditions when dealing with NP-hard optimization problems. If we relax reliability, then we get heuristics that are designed to work well on instances that commonly appear in practice. If we relax the third condition, we obtain *approximation algorithms*. The notion of approximation algorithms allows us to obtain a more fine-grained picture of the difficulty of NP-hard optimization problems: some problems turn out to have very good approximations, whereas others resist such attempts.

The formal definition of approximation algorithms is as follows.

Definition 3 *An α-approximation algorithm for a given optimization problem is an algorithm that runs in polynomial time and outputs a solution S such that:*

- $\frac{cost(S)}{cost(Optimal\ solution)} \leq \alpha$ *if the problem is a minimization problem,*
- $\frac{profit(S)}{profit(Optimal\ solution)} \geq \alpha$ *if the problem is a maximization problem.*

It is clear that we will have $\alpha \geq 1$ for minimization problems and $\alpha \leq 1$ for maximization problems. Moreover, if $\alpha = 1$, then we have an efficient exact algorithm (and the problem is in P). The goal when designing approximation algorithms is to achieve a guarantee α that is as close to 1 as possible. We now introduce a general framework for using linear programming in the design of approximation algorithms followed by an application: vertex cover.

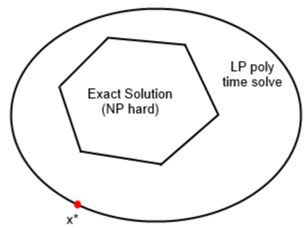
3 Using Linear Programming to Design Approximation Algorithms

Consider a minimization problem. When considering the definition of approximation algorithms, it seems very hard, even for a specific instance of the problem, to analyze the approximation guarantee $\frac{\text{cost}(S)}{\text{cost(Optimal solution)}}$ of the algorithm on the instance, as we are comparing ourselves with an optimal solution (that is most likely very hard to compute and does not possess any nice structure). The solution to this is to compare ourselves with a *lower bound* on the optimum. Indeed, we then have that

$$\frac{\text{cost}(S)}{\text{cost(Optimal solution)}} \leq \frac{\text{cost}(S)}{\text{lower bound on opt}} \leq \alpha\,.$$

From the above, it is clear that we need a good lower bound to be able to claim a good guarantee α. To obtain such a lower bound (and to design the approximation algorithm), linear programming is super handy. A popular framework is as follows:

1. Give an exact formulation of the problem as Integer LP – usually with binary variables ($x_i \in \{0,1\}$).
2. Relax to LP – $x_i \in [0,1]$



3. Solve LP to get a optimal solution x^* to the LP which is a lower (upper) bound on the optimal solution to Integer LP and thus the original problem. Then somehow round x^* to an integral solution "without losing too much" (which will determine the guarantee α).

We now use the above framework for the vertex cover problem, and then introduce the concept of integrality gap.

3.1 Vertex Cover

Here, we apply the framework to get an approximation algorithm for the *Vertex Cover* problem. First recall the definition:

Definition 4 (Vertex Cover (VC) Problem) *Given a graph $G = (V, E)$ and a weight function on the vertices $w : V \to \mathbb{R}_+$, output a set $C \subseteq V$ of minimum weight such that for all $\{u, v\} \in E$, $u \in C$ or $v \in C$.*

First, we define an ILP solving VC: For all $v \in V$, we introduce a variable x_v, which is 1 if $v \in C$, and 0 otherwise. The objective function is

$$\min \sum_{v \in V} w(v) x_v$$

and for each $\{u, v\} \in E$ we introduce the constraint $x_u + x_v \geq 1$. This ensures that for each edge, at least one endpoint is in C. Additionally, we require that for each $v \in V$ we have $x_v \in \{0, 1\}$. Note that at this point our Integer LP formulation is exactly equivalent to the original problem.

Second, we relax the ILP to an LP by allowing that $x_v \in [0, 1]$. Actually, it's sufficient to require that $x_v \geq 0$, because having an x_v greater than 1 will not satisfy any additional constraint, but only increase the value of the objective function, so this will not happen in an optimal solution.

Third, we have to do the rounding: Suppose we've solved the LP and got an optimal solution x^*. We will return $C = \{v \in V : x^*_v \geq \frac{1}{2}\}$ as our solution to VC.

Claim 5 *C is a feasible solution.*

Proof Consider any edge $\{u, v\}$ and its constraint. Since $x^*_u + x^*_v \geq 1$, at least one of x^*_u, x^*_v is $\geq \frac{1}{2}$ and thus in C. ■

Claim 6 *The weight of C is at most twice the value of the optimal solution of VC.*

Proof We have

$$\sum_{v \in C} w(v) = \sum_{v \in V: x_v^* \geq \frac{1}{2}} w(v) \leq \sum_{v \in V: x_v^* \geq \frac{1}{2}} 2x_v^* w(v) \leq \sum_{v \in V} 2x_v^* w(v) = 2 \sum_{v \in V} x_v^* w(v) = 2LP_{OPT} \leq 2VC_{OPT}$$

where the first inequality holds because $2x_v^* \geq 1$. ■

So, we have designed a 2-approximation algorithm for Vertex Cover. This is a simple algorithm using linear programming. To appreciate the power of the framework, I challenge you to find a 2-approximation algorithm for Vertex Cover (with node-weights) without the use of LPs. (It is possible but quite difficult.)

3.2 Integrality Gap

The notion of the *integrality gap* allows us to bound the power of our linear programming relaxation. Let $\mathcal{I}$ be the set of all instances of a given problem. In the case of a minimization problem, the integrality gap g is defined as

$$g = \max_{I \in \mathcal{I}} \frac{OPT(I)}{OPT_{LP}(I)} .$$

As an example, suppose $g = 2$, and that LP found that $OPT_{LP} = 70$. Then, since our problem instance might be the one which maximizes the expression for g, all we can guarantee is that $OPT(I) \leq 2 \cdot OPT_{LP}(I) = 140$, so it's not possible to find an approximation algorithm (using only this linear programming relaxation) that approximates better than within a factor of $g = 2$.

3.2.1 Integrality Gap for Vertex Cover

Claim 7 *The integrality gap for vertex cover is at least* $2 - \frac{2}{n}$.

Proof Consider the complete graph on n vertices. We have $OPT = n - 1$, because if there are 2 vertices that we don't choose, the edge between them is not covered. However, $LP_{OPT} \leq \frac{n}{2}$, because assigning $\frac{1}{2}$ to each vertex is a feasible solution of cost $\frac{n}{2}$, so the optimum can only be smaller. Now,

$$g \geq \frac{n-1}{\frac{n}{2}} = 2 - \frac{2}{n} .$$

■

We remark that our 2-approximation algorithm for vertex cover implies that the integrality gap is *at most* 2.

References

[1] Mateusz Golebiewski, Maciej Duleba: *Scribes of Lecture 5 in Topics in TCS 2015.* *http://theory.epfl.ch/courses/topicstcs/Lecture52015.pdf*