Miscellaneous notes in Algorithm Design, Analysis and Complexity

© Tingfeng Xia

Fall 2019, modified on November 6, 2019

Contents

1 6.046J Linear Programming: LP, Reductions, Simplex

1.1 Example: Politics, example of optimization

- Goal: You want to buy elections and you want to minimize the total amount of money spent.
- How to campaign to win an election? Manager estimates votes obtianed per dollar spent.

	Policy	Urban	Suburban	Rural
x_1	Build Roads	-2	5	3
x_2	Gun Control	8	2	-5
x_3	Farm Subsidies	0	0	10
x_4	Gasoline Tax	10	0	2

- Want a mojority for each demographic.

Polulation	100,000	200,000	50,000
Majority	50,000	100,000	25,000

- Want to win by spending the minimum amount of money.
- Algebraic Setup: Let x_1, x_2, x_3, x_4 denote the dollar spent per issue.

$$\begin{cases} \text{minimize} & x_1 + x_2 + x_3 + x_4 \\ \text{subject to} & (1) \quad -2x_1 + 8x_2 + 0x_3 + 10x_4 \geq 50000 \\ & (2) \quad 5x_1 + 2x_2 + 0x_3 + 0x_4 \geq 100000 \\ & (3) \quad 3x_1 - 5x_2 + 10x_3 - 2x_4 \geq 25000 \\ & (4) \quad x_1, x_2, x_3, x_4 \in \mathbb{R}^{\geq 0} \end{cases}$$

  Notice that constraint (4) above denotes there is no negative advertisation.
- The optimal solution is

$$\begin{cases} x_1 = 2050000/111 \\ x_2 = 425000/111 \\ x_3 = 0 \\ x_4 = 625000/111 \end{cases}$$

  and the objective optimized has value $\frac{3100000}{111}$

1.2 Standard From for LP

- Minimize or Maximize[1] linear objective function, subject to linear ineqalities or equations
- Variables $\mathbf{x} = [x_1, x_2, \ldots, x_n]^T$, and the objective function is $\mathbf{c} \cdot \mathbf{x} = c_1 x_1 + \cdots + c_n x_n$ and the ineqalities $\mathbf{A} \cdot \mathbf{x} \leq \mathbf{b}$[2] and $\mathbf{x} \geq \mathbf{0}$[3]

1.3 Certificate of Optimality

Is there a short certificate? [4] Consider

$$25/222(1) + 46/222(2) + 14/222(3)$$

, where we can plug in the equations and simplify to

$$x_1 + x_2 + 140/222 x_3 + x_4 \geq 3100000/111$$

But notice that

$$3100000/111 \leq x_1 + x_2 + 140/222 x_3 + x_4 \leq x_1 + x_2 + x_3 + x_4$$

so the solution must be optimal!

1.4 LP Duality

What is this? This is essentially saying that what we did above was no coincidence, and we can always to this for a linear program.

Theorem For all standard form of LP (called a primal form) there exists a dual form that is equivalent to the primal. Specifically

$$\left.\begin{array}{ll} \text{maximize} & \mathbf{c} \cdot \mathbf{x} \\ \text{subject to} & A\mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \end{array}\right\} \quad \equiv \quad \begin{cases} \text{minimize} & \mathbf{b} \cdot \mathbf{y} \\ \text{subject to} & A^T \mathbf{y} \geq \mathbf{c} \\ & \mathbf{y} \geq \mathbf{0} \end{cases}$$

1.5 Converting to Standard Form

1.5.1 Case 1: Minimize Goal

Suppose that I want to minimize $-2x_1 + 3x_2$, then I can just convert the problem into maximizing the negative of the equation. This case should be easy.

[1] In the standard form, we consider the maximization problem

[2] In general, this could have been $\leq, \geq, =$ but for the standard form, we consider $\leq$

[3] Meaning that each of the slots in the vector should be greater than zero.

[4] For the problem of Politics described above, notice that this is not a general certificate, it only works in this specific case.

1.5.2 Case 2: Missing Non-negative Constraint

Suppose x_j doesn't have a non-negative constraint. In this case, we will replace x_j with $x'_j - x''_j$ such that $x'_j \geq 0 \wedge x''_j \geq 0$.

1.5.3 Case 3: Equlity Constraint

Suppose the constriant was $x_1 + x_2 = 7$, then we can break it into $x_1 + x_2 \leq y \wedge -x_1 - x_2 \leq -7$.[5]

1.5.4 Case 4: GEQ Constraint

We have done this above, translate this into a less than or equal tot problem by multiplying (-1) on both sides (which will flip the ineqality sign).

1.6 Max-Flow using LP

Consider some network $N := G = (V, E)$, and denote the flow in the network to be f. the function $c(\cdot)$ returns the capacity for an edge. The problems then breaks into

$$\begin{cases} \text{maximize} & \sum_{v \in V} f(s, v) = |f| \\ \text{subject to} & f(u, v) = -f(v, u) \quad \forall u, v \in V \\ & \sum_{v \in V} f(u, v) = 0 \quad \forall u \in V \setminus \{s, t\} \\ & f(u, v) \leq c(u, v) \quad \forall u, v \in V \end{cases}$$

where we notice that the above problem is entirely linear and thus could be solved using a linear programming algorithm.

Time Complexity This generalization uisng LP is much slower than the network flow algorithms (Ford-Fulkerson, Edmonds-Karp, et cetra) in **single commodity network flow**.

Multi-Commodity Flow Consider the case where there are two commodities flowing in the network (f_1, c_1, f_2, c_2). In the case where c_1 is independent from c_2, it is not very interesting, we can just run the network flow algorithm twice to find two maximizers seperately. In the case where there is a single capacity constraint c[6], the problem ceases to be a simple network flow problem. However, it is still easy to come up with a LP formulation that describes the maximization.

[5] Notice that we did this in this specific way becasue we want to have this in out standard form where the constraint was a $\leq$. The second constraint here, if we multiply it by -1 on both sides, is equivalent to saying $x_1 + x_2 \geq 7$

[6] A toy example to give would be given a certain road where some cars are running on the road and the raod have a total capacity for all types of cars combined

1.7 Shortest Path using LP

- $d[v]$ represents the shortest path from the source to v[7]
- $w(u, v)$ means the single edge (u, v)'s weight
- The shortest path to some vertex v which is a descendent of u is at least shorter than or equal to the existing path that goes from source to u plus the edge (u, v)
- The shortest path from source to source is zero
- Recall the $\triangle$-ineqality here

This yields the formulation

$$\begin{cases} \text{maximize} & d[v] \\ \text{subject to} & d[v] - d[u] \leq w(u,v) \quad \forall (u,v) \in E \\ & d[s] = 0 \\ & d[v] - d[u_1] \leq w(u_1, v) \\ & d[v] - d[u_2] \leq w(u_2, v) \\ & d[v] = \min(\dots, \dots) \end{cases}$$

Notice that although we are trying to minimize the distance from source to the node v, we have to put this as a maximization problem because otherwise the trivial solution of 0 will work! Our formulation didn't capture the insight "We DO want a path".

Key Insight of MAX The above formulation already captured the minimization problem with the min in the last constraint and hence we want to push up as hard as we can in finding a solution. (We are ANDing together all the constraints and have chosen the one that is the smallest)

1.8 Simplex Algorithm

1.8.1 Work Flow

1. Represent LP in slack form
2. Convert one slack form into an equivalent whose objective value has not decreased and has likely increased (no gurantee of increase)
3. Keep going until the optimal solution becomes obvious

[7] It might be helpful to recall that in the Dijkstra's Algorithm $d[v] \leftarrow \infty$ initially and then it was decremented through out the algorithm until the minimum was reached.

1.8.2 Time Complexity

This is, unfortunately, an exponential iterative algorithm. Denote the number of constraints using m and n as the number of variables then the algorithm has worst case time complexity

$$T(m,n) \in \mathcal{O}\left(\binom{m+n}{n}\right)$$

1.8.3 Procedure Example

Consider the problem

$$\begin{cases} \text{maximize} & 3x_1 + x_2 + x_3 \\ \text{subject to} & x_1 + x_2 + x_3 \leq 30 \\ & 2x_1 + 2x_2 + 5x_3 \leq 24 \\ & 4x_1 + x_2 + 2x_3 \leq 36 \\ & x_1, x_2, x_3 \geq 0 \end{cases}$$

The Slack From [8] The original variables (x_1, x_2, x_3 here) will be called non-basic variables and we will here introduce three *basic* variables[9] x_4, x_5 and x_6. The original problem will then be

$$\left.\begin{cases} z = 3x_1 + x_2 + x_3 \\ x_4 = 30 - x_1 - x_2 - x_3 \\ x_5 = 24 - 2x_1 - 2x_2 - 5x_3 \\ x_6 = 36 - 4x_1 - x_2 - 2x_3 \end{cases}\right\} (I)$$

It is worth mentioning that now the non-negativity constriant becomes $\mathbb{R}^6 \ni \mathbf{x} \geq \mathbf{0}$, we have "added"[10] three more new variables that are also non-negative.

Basic Solution Set all the non-basic variables to zero, and then compute the values of the basic variables. The objective function will be $z = 3(0) + 1(0) + 1(0) = 0$. This is a trivial starting point and we can think of this solution as $\mathbb{R}^6 \ni \mathbf{x} = (0, 0, 0, 30, 24, 30)$

Pivoting

- Select a non-basic variable x_e whose coefficient in the objective function is positive
- Increment the value of x_e as much as possible without violating any of the constraints.

[8] Here the word 'slack' means how much room is still left

[9] This amount will be equal to the number of constraints that the original problem have

[10] This was quoted because we didn't actually introduce any new constraints! In fact, they are pair-wise equivalent to the original ones.

- Varaible x_e becomes basic, some other variable becomes non-basic. (Value of the other basic variable and the objective function may change.)

Running the Procedure

- Suppose we selected the non-basic variable $x_e = x_1$ and we want to increase the value of x_1.
- The third constraint is the tighest one in (I). Rearrange the terms we have

$$x_1 = 9 - x_2/4 - x_3/2 - x_6/4 \tag{1}$$

- Then we will rewrite the other equations with x_6 on the RHS. i.e. replace all occurences of x_1 with (1) above. **Important:** What has happened is that x_1 and x_6 has exchanged their roles. x_1 was non-basic and will now become basic and is the reverse for x_6. The following is the re-written result:

$$\left\{\begin{array}{l} z = 27 + x_2/4 + x_3/2 - 3x_6/4 \\ x_1 = 9 - x_2/4 - x_3/2 - x_6/4 \\ x_4 = 21 - 3x_2/4 - 5x_3/2 + x_6/4 \\ x_5 = 6 - 3x_2/2 - 4x_3 + x_1/2 \end{array}\right\} (II)$$

  Point of the above operation: Recall the original basic solution was $(0, 0, 0, 30, 24, 36)$, which certainly satisfies (II) above and have objective value

$$27 + \frac{1}{4}(0) + \frac{1}{2}(0) - \frac{3}{4}(36) = 0$$

  For the basic solution for (II), we set the non-basic values to zero which will yield the solution $(9, 0, 0, 21, 6, 0)$[11]. The objective value is now $3x_1 + x_2 + x_3 = 9 \times 3 = 27$.[12]

- Repeat the above procesure. In this case, 2 more iterations is required. We will know it is the time to stop when the objective function is some constant followed by negative copies of non-basic variables (which are non-negative) in which case the objective fucntion cannot be increased anymore. We call this the convergence of the Simplex Algorithm.

2 6.006 Computational Complexity

2.1 Complexity Classes

- **P** Set of all problems that can be solved in less than or equal to polynomial time.

[11] The solution was computed by setting variables on the RHS of the equations (which are the non-basic vars now) to zero in (II)

[12] The original objective function was used here

- **NP** Decision problems that can be solved in polynomial time via a lucky algorithm.
- **NP-Hard** As hard as every problem in NP.
- **NP-Complete** NP $\cap$ NP-Hard.
- **EXP** Set of all problems that can be solved in less than or equal to exponential time.
- **R** Set of all problems that can be computed/solved in finite time

2.1.1 Theorem: Alomost All Decision Problems are not in R

This might be counter-intuitive but is mathematically valid. We are here limiting our scope to decision problems only, which is defined as 'problems that outputs yes/no (some binary outcome)'. We will now prove this result, *Proof:*

- First, note that a program, what we write on a computer, is just a bianry string in some sense. But if it could be represented in a binary number (which might be a very big one), it is just a natural number in base 2. This is our space of programs.
- Next, a decision problem is a funtion that is $f : \text{input} \to \{\text{YES}, \text{NO}\}$. But for sure the input for a program is some string which could be converted into binary which is again some natural number. Hence, our f is one such that $f : \mathbb{N} \to \{\text{YES}, \text{NO}\}$.
- Above, we have argued that a program is a finite string of bits, while a decision problem is a infinite string of bits[13].
- Hence, decision problems $\in \mathbb{R}$ while programs is $\in \mathbb{N}$. But $|\mathbb{R}| \gg |\mathbb{N}|$! So there is not enough program to solve all the decision problems.

■

2.2 The Big Conjecture

$$P \neq NP \approx \text{"You can't engineer luck"}$$

2.3 Reductions

The idea is to convert some problem A into a problem B. When we can reduce A to B, we say B is as hard as A.

Example I To solve unweighted shortest paths, we can use a BFS. But for sure we can also solve this using Dijkstra by setting all the weights to one. This would be an example of a reduction.

[13] For different inputs, the output may or may not be the same but importantly is never stops, i.e. there always exists the next problem.

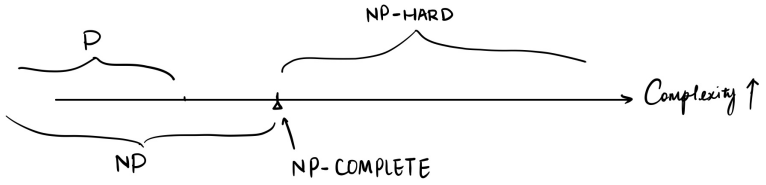
Example II Suppose we want to, again, solve the shortest path (weighted) problem but we are only given the tool of 'min-product path'[14]. You can do this by taking logs, after which products become sum!

3 6.046J Complexity: P, NP, NP-Completeness, Reductions

3.1 Definitions

- **NP** Given any instance I of problem P and witness W, if there exists a verifier V so that given the ordered pair (I, W) as input, V returns "yes" in polynomial time if the witness proves that the answer is "yes" and "no" in polynomial time otherwise, then P is in NP. **Important:** Notice that to show that a problem is in NP, we have to consider all possible certificates ($\forall$ quantified) and for each certificate there should exist some verfier (which could certainly be different each time).
- **NP-Completeness** A problem X is NP-Complete if X$\in$NP and X$\in$NP-Hard
- **NP-Hard** X is NP-Hard if every problem Y$\in$NP reduces to X.
- **Redution** from problem A to a problem B = poly-time algorithm converting A inputs to some equivalent B inputs.[15]

3.2 Class Relationships



3.3 Proving NP-Completeness

From a algorithm perspective, the question is 'How to prove X is NP-Complete?'

1. Show that X in NP. (Give a non-detrministeic algorithm OR give a certificate and verify.)
2. Reduce from some known NP-Complete problem Y to your problem X. **Important:** Always reduce from the thing that you know is hard to the thing that you want to show is hard. This direction matters.

[14] That is, minimize the product of the weights along the path
[15] Two important observations here: 1. if B in P then A in P; 2. if B in NP then A in NP

3.4 3 SAT

Given a boolean formula, which is an AND of triplet ORs. A clause is one of the triplet ORs and a literal is one element in a triplet OR. The question is does there exist a assignment of variables such that the final result (the entire boolean formula) evaluates to true.

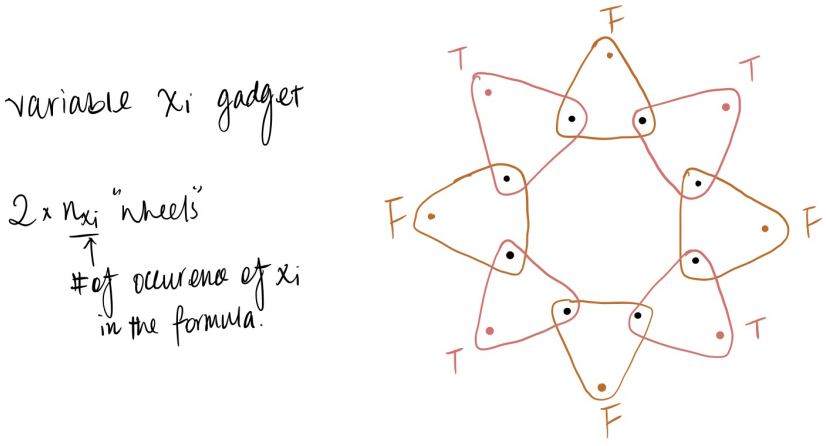
Intuition behind the Proof of NP-Completeness of 3 SAT

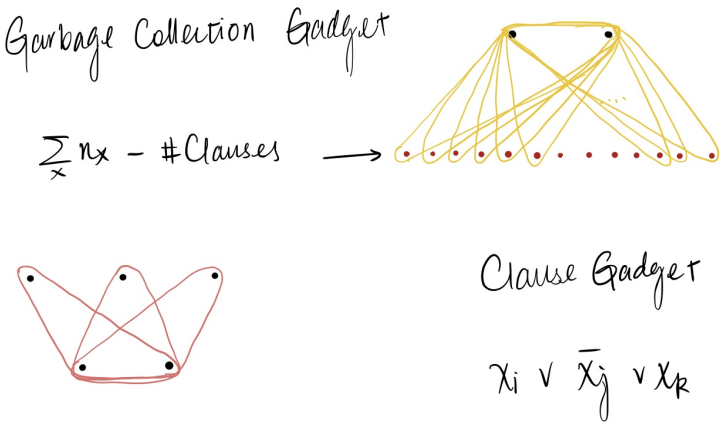
- Software $\approx$ Hardware, you can build a circuit for a software/algorithm using logic gates
- If I have a circuit, than I can convert that into a boolean formula.
- (Fun Fact; Less obvious) If I have formula, then I can convert it into the form 3 SAT problem requires.
- Then the 'witness/oracle' is described by the literals in the formula
- Then deciding whether there's some way to set the literals to make the formula true is the same thing as saying 'Is there some certificate where the verifier says yes', which is the same thing as saying that the problem has answer yes.

3.5 3-Dimensional Matching (3DM)

Problem Statement You are given disjoint sets X, Y and Z, each of size n. You are also given triples $T \subseteq X \times Y \times Z$. The goal is to choose subset $S \subseteq T$ such that every element $\in X \cup Y \cup Z$ is in exactly one $s \in S$.

Reducing From 3 SAT We will need the following three gadgets, namely the variable gadget, the garbage collection gadget, and the clause gadget.





- In the variable gadget, the dots on the inner circle exists in only that gadget, and there is a variable gadget for each and every variable.
- The reason that we have $2n_{x_i}$ sized[16] variable gadgets is that here in the wheel we are only allowed to choose all the T's or all the F's but not mixed. This way, we have twice the number of needed variable connectors and then we choose half of them which is exactly what we need for the gadget to work.
- The clause gadget has the two dots on the lower level local and the top three are connected to the outer T/F dots on the variable gadgets. Notice that we are only allowed to choose one of the three triangles in the gadget drawn. Importantly, the one choice gurantees the bottom two nodes are chosen, meaning the clause is satisfied.
- Doing the above is not enough since then not all T/F nodes[17] will be used which is problematic. To solve this, we add a garbage collection gadget.
- The Garbage Collection Gadget has all the T/F nodes on its lower row, and the gadget itself is repeated $\sum_x n_x - \#Clause$ times. This will help to connect all the unconnected nodes left from the previous steps.

3.6 Subset Sum

Problem Statement Given n integers $A = \{a_1, ..., a_n\}$ and target sum t. Does there exist a subset sum $S \subseteq A$ such that $\sum S = \sum_{a_i \in S} a_i = t$, i.e. adds up to a given sum.

[16] Notice that here n_{x_i} counts all occurences of the variable x_i, including the $\overline{x_i}$'s

[17] That is, all the top three nodes in all the clause gadgets and all the nodes in the outer ring in the variable gadgets

Reducing From 3DM

- View numbers in base $b = 1 + \max_i n_{n_i}$, a large enough base. The max part is the total number of colliding ones and we are setting our base to be one larger than that so that there will never be carries.
- Triple (x_i, x_j, x_k) will be converted into 0000100001001000_b where the 1's are at the i, j, k-th position in the number.
- The goal is to add up to some number $1111111111 = \sum_{i'} b^i$

How does this reduction work? Notice that my goal now in base b is a bunch of ones, and I am trying to add up to this number from numbers in base b composed of three 1's and 0's on the rest positions. So, I need to gurantee that there is no collision meaning no two occurences of 1 on the same position will be accepted. Then, the problem is essentially choosing non-overlapping subsets that can add up to the target number which is exactly what 3DM solves. $\mathcal{Q}.\mathcal{E}.\mathcal{D}.\dagger$

Weakly NP-Hardness The numbers in the above reduction are giant. The number of digits in the target number in base b have order $\mathcal{O}(n)$ digits. But, the actual values of the numbers are exponential, and this is not allowed in a Strong NP-Hardness setting.

3.7 Partition

Problem Statement Given $A = \{a_1, ..., a_n\}$ integers, is there a subset $S \subseteq A$ such that $\sum S = \sum S \setminus A$, i.e. the set is partitioned into two parts that adds up to the same numeber.

Reducing From Subset Sum

- Let t denote the arbitrary target of my Subset Sum instance
- Let $\sigma = \sum A$
- Add $a_{n+1} = \sigma + t$ and $a_{n+2} = 2\sigma - t$
- After adding the two terms described above, we first note that they can't be on the same side in the partition, because if they do then one side would have sum at least $\sigma + t + 2\sigma - t = 3\sigma$ which is way larger than σ, the total sum of all numbers.
- Now we know that these two numbers have to be on two sides of the partition, we notice that if we add $\sigma - t$ to the side that has $a_{n+1} = \sigma + t$ and add t to the other side, then both sides will have sum 2σ, which is what we want.

- Starring at the above step, we notice that $\sigma - t + t = \sigma$ is the total sum of elements that we want to add to two sides of the partition, which is exactly equal to the total sum of all the original n elements!
- Then, if this partition problem could be solved, my subset sum of arbitrary target t can be solved and this shows Subset Sum could be p-reduced to Partition.

3.8 My Thoughts

P-Reducibility $A \leq_p B$ means A is p-reducible to B. To show this is true, we have to take an arbitrary input of A, and find a polynomial time conversion such that this input is now a appropriate input for B. Then we will let B solve this problem for us, and we will need a polynomial time conversion back from B's solution to the required A's.

Mario Reduced From 3 SAT We tried to show that 3 SAT $\leq_p$ Mario, i.e. 3 SAT is p-reducible to Mario. Given any instance of 3 SAT problem[18], we can build a respective level for Mario and whether Mario dies or not will tell us if 3 SAT should return YES or NO.

Partition Reduced From Subset Sum (Subset Sum $\leq_p$ Partition) Given a Subset Sum problem, we add two elements into the set and give this as the input to the Partition solver. If the partition solver was able to return a YES solution, then we know that the elements in the partitions which contains the $2\sigma - t$ element (exclude the $2\sigma - t$ element) has subset sum equal to t which means the Subset Sum could be solved by using Partition as a black box.

[18] By instance of a problem, we mean that the input of the problem, which will be translated to the input of some other problem and from where the solution yielded will be converted back and thus act as the solution to 'the instance of a problem'.