Differential Geometry

Instructor: Jianfeng Lin

Notes Taker: Zejin Lin

TSINGHUA UNIVERSITY.

linzj23@mails.tsinghua.edu.cn

lzjmaths.github.io

February 25, 2025

Contents

1 Introduction

Discrete (combinatorial) optimization is a subfield of mathematical optimization that consists of finding an optimal object from a finite set of objects, where the set of feasible solution is discrete or can be reduced to a discrete set.

However, usually this feasible solution set is very large (due to combinatorial explosion) and it is computationally infeasible to go through all feasible solutions and find the one with opimal objective function value.

Example 1.1 (Task Assignment)**.** There are n tasks and n workers. Each task has an importance score a_i and each worker has a skill level b_i. We need to assign each task to a worker such that the sum of $\sum_{i=1}^{n} a_i b_{\sigma(i)}$ is maximized.

1.1 Models of Computation: Turing Machines

Definition 1.2 (A Deterministic Turing Machine(DTM))**.** It consists of an infinitely-long tape (memory) and a deterministic finite automata that controls the head to move along the tape and read/write symbols from/to the tape cells.

Definition 1.3 (Complexity measure)**.** Running time is the number of steps of Turing machine.

Memory is the number of tape cells used.

Definition 1.4 (Caveat)**.** No random access of memory

- Single-tape DTM requires $\geqslant n^2$ steps to detect n bit palindromes.
- EAsy to detect palindromes within c_n steps on a real computer.

1.2 Models of Computation: word RAM

Definition 1.5. Each memory location and input/output cell stores a w-bit integer (assume $w \geqslant \log_2 \omega$).

Primitive Operations:

1.3 Polynomial Running Time

Definition 1.6. We say that an algorithm is **efficient** if its running time is polynomial of input size n.

Example 1.7 (Task machine)**.** Polynomial-time algorithm: selection sort/inserting sort/quick sort/merge sort.

Non-polynomial-time algorithm: try all possible matching and output the one with the highest score.

- Definition is relatively insensitive to model of computation.
- The poly-times algorithm that people develop have both small constants and small exponents
- Breaking through the exponential barrier is a major challenge.

1.4 Notation

Definition 1.8. $f(n)$ is $O(g(n))$ if there exist constants $c > 0$ and $n_0 \geqslant 1$ such that $0 \leqslant f(n) \leqslant c \cdot g(n)$ for all $n \geqslant n_0$.

$f(n)$ is $\Omega(g(n))$ is $g(n) \in O(f(n))$.

$f(n)$ is $\Theta(g(n))$ is both $f(n) \in O(g(n))$ and $g(n) \in O(f(n))$.

1.5 Tentative Syllabus

We will introduce three exact discrete optimization algorithms(6 weeks):

- Greedy algorithms
- Dynamic programming
- Network flows

And some approximation algorithms for intractable discrete optimization problems(9 weeks)

- Definition of approximation algorithms
  - Algorithm techniques: greedy, linear programming relaxation, semidefinite programming relaxation.
- Hardness of approximation
  - Techniques: hardness reductions, Fourier analysis of Boolean functions.
- Problems studied: Set-Cover, facility location, K-center, Multi-Cut, Max-Cut, $\cdots$

2 Greedy Algorithms

2.1 Interval Scheduling

Example 2.1 (Interval Scheduling)**.** Input: n jobs, $\{(s_i, f_i)\}_{i=1}^n$. Goal: How to choose jobs with maximized number such that each pair of intervals do not intersect.

Greedy Framework Consider jobs in order $\pi(1), \pi(2), \cdots, \pi(n)$. For each $\pi(i)$, $i = 1, 2, \cdots, n$, if $\pi(i)$ compatible with all selected jobs, then select $\pi(i)$.

The choice of π: Earliest-start-time-first, Earliest-finish-time-first, Longest-job-first, Shortest-job-first, etc.

Theorem 2.2. *Earliest-finish-time-first greedy returns an optimal solution.*

Proof. Suppose algorithm selects $i_1, i_2, \cdots, i_k$, opt selects $k' > k$ jobs.

Choose an optimal solution agrees with algorithm in first r jobs so that r maxmized, $j_1, j_2, \cdots, j_{k'}$.

Obviously, $r < k$. Then $f_{i_{r+1}} < f_{j_{r+1}}$. Therefore, we can replace i_{r+1} with j_{r+1} to get another optimal solution, which contradicts to the fact that r maxmized. $\square$

2.2 Interval Partitioning

Example 2.3 (Interval Partitioning)**.** Input: n lectures, $\{(s_i, f_i)\}_{i=1}^n$.

Goal: Position lectures into minimum number of classrooms so that in each classroom lectures are compatible.

Greedy Framework Lectures in order $\pi(1), \cdots, \pi(n)$, the number of opening classrooms is zero in the beginning. For each $\pi(i)$,

If $\exists$ opening classroom j *s.t.* lecture $\pi(i)$ compatible with lectures in j, then $\pi(i) \rightarrow$ classroom j.

Else, open a new classroom for $\pi(i)$.

Proof. Introduce a concept **Depth**: $d(t) =$ Number of lectures active at time t, and $d = \max_t\{d(t)\}$.

Claim. OPT $\geqslant d$.

Lemma 2.4. *Alg* $\leqslant d$

Proof. Assume for contradiction.

At some point, Alg opens $d+1$ classroom.

Denote the lecture being considered by i. Then it is not compatible with other d lectures. Hence, there should be a time when $d+1$ lectures are active, which causes contradiction. □

□

2.3 Single-Source Shortest Path

Example 2.5 (Single-Source Shortest Path(SSSP))**.** Input: Graph $G = (V, E, w)$, V is the set of point and E is the set of edge with direction and $\omega : E \to \mathbb{R}_{\geqslant 0}$.

We want to find a path from s to t with minimum total cost.

Dijkstra's Algorithm Choose s as a source. $d[s] = 0, d[u] = \begin{cases} \omega(s,u) & \text{if } (s,u) \in E \\ +\infty & \text{otherwise} \end{cases}$, $S = \{s\}$ first. To record the path, we can use $\mathrm{Pred}[u] \leftarrow s$.

Algorithm 1 Dijkstra's Algorithm

while $S \neq V$ **do**
  Choose $u \in \arg\min_{\notin S}\{d[x]\}$.
  Update $S \leftarrow S \cup \{u\}$.
  for each $x \in V - S$,$(u, x) \in E$ **do**
    $d[x] \leftarrow \min\{d[x], d[u] + \omega(u, x)\}$.
    if $d[u] + \omega(u, x) < d[x]$ **then**
      $d[x] \leftarrow d[u] + \omega(u, x)$
      $\mathrm{Pred}[x] \leftarrow u$
    end if
  end for
end while

Theorem 2.6 (Invariant)**.** $\forall u \in S$, $d[u]$ *is the shortest path distance* $s \rightsquigarrow u$

Proof. Induction on $|S|$.

For $|S| = 1$ true.

Induction Step: Every time executing 2 in Algorithm 1, we need to prove $d[u]$ is the shortest distance $s \rightsquigarrow u$.

If $v = \mathrm{Pred}[u] \in S$, then $d[u] = d[v] + \omega(v, u)$.

For any path from s to u, there exists $(\alpha, \beta) \in E$ such that $\alpha \in S, \beta \notin S$. Then

$$\begin{aligned}
\mathrm{length}(P) &\geqslant \mathrm{length}(P[s \to \beta]) \\
&= \mathrm{length}(P[s \to \alpha]) + \omega(\alpha, \beta) \\
&\geqslant d[\alpha] + \omega(\alpha, \beta) \\
&\geqslant d[\beta] \geqslant d[u]
\end{aligned}$$

□

Remark 2.7. The straightforward implementation of Dijkstra's Algorithm is of $O(|v|^2)$.

If we use priority queue: Q with priority $Q.\pi()$. It has some methods:

- ExtractMin: Return $\arg\min_{x \in Q}\{Q.\pi(x)\}$ and remove x from Q.
- DecreaseKey: Update $Q.\pi(v)$ with newkey.

The time complexity is $|V|\times$ ExtractMin + $|E|\times$ DecreaseKey

Runtime	ExtractMin	DecreaseKey	Dijkstra
Simple Array	$O(\|V\|)$	$O(1)$	$O(\|v\|^2)$
Binary Heap	$O(\log \|V\|)$	$O(\log \|V\|)$	$O(\|E\| \cdot \log \|V\|)$
Fibonacci Heap	$O(\log \|V\|)$	$O(1)$ (amorized)	$O(\|E\| + \|V\| \log \|V\|)$

2.4 Minimum Spanning Tree

Example 2.8 (Minimum Spanning Tree (MST))**.** Input: Connected, undirected graph $G = (V, E, \omega)$.

Definition 2.9 (Spanning Tree)**.** $T \subset E$ is a **spanning tree** if $|T| = |V| - 1$, $G' = (V, T)$ is connected.

Goal of MST Find spanning tree T so that $\omega(T) = \sum_{e \in T} \omega(e)$ minimized.

Theorem 2.10 (Cayley Theorem)**.** *The number of spanning trees of n-vertex complete graph is n^{n-2}*

A **cut** $(S, V - S)$ has a **cutset** of $S = \{e = (u, v) : u \in S, v \notin S\}$.

Claim. Any cycle C and cutset D has intersection $|C \cap D|$ even.

Fundamental Cycle: Given G and spanning tree $T \subset E$, for each $e \in E \backslash T$, the unique cycle in $T \cup \{e\}$ is called **Fundamental cycle**.

Claim. For a fundamental cycle C related with e, $\forall f \in C \cap T$, $(T \cup \{e\}) \backslash \{f\}$ is also a spanning tree.

If T is MST, then $\omega(e) \geqslant \omega(f)$.

Fundamental Cut: Spanning tree $T \subset E$. For each $f \in T$, $T \backslash \{f\}$ has two connected components, whose cutset is called **fundamental cut**.

Claim. $\forall e \in D \backslash T$, $(T \cup \{e\}) \backslash \{f\}$ is a spanning tree.

If T is MST, then $\omega(e) \geqslant \omega(f)$.

MST Algorithm There are some rules. **Red rule**:Let C a cycle without red edges. Select an uncolored edge in C with max weight and color it red.

Blue rule: Let D be a cutset without blue edges. Select an un colored edge in D with min weight and color it blue.

Greedy Algorithm:Apply red or blue rules in any order iteratively until all edges colored.

Theorem 2.11. *Greedy algorithm terminates and blue edges from MST.*

Proof. Observed that during the algorithm, blue edges always from a forest. □

Invariant $\exists$ MST T^* *s.t.* T^* contains all blue edges and no red edges.

Proof. Proof by induction. If there is a MST T^* contains all blue edges no red edges now. If we apply blue rule, with cutset D and $f \in D$ but $f \notin T^*$, then for fundamental cycle C of f, $\forall e \in C \cap T$, $\omega(e) \geqslant \omega(f)$. Since C has even edges in the cutset by the claim, $\exists e \in C \cap T$ *s.t.* $e \in D$, which contradicts the fact that f is the edge in cutset D with min weight.

The case that we apply red rule is similar. □

Algorithm 2 Prim's Algorithm

Initialize $S \leftarrow \{s\}$.
while $n - 1$ times **do**
    Choose e be the min weight edge in the cutset $(S, V \backslash S)$
    add e to T, another endpoint of e to S.
end while

Remark 2.12. It is compatible with the simple idea: Each time chooses the min weight edge. However, it is more powerful since we only need to do this process in the cutset.

It is similar to Dijkstra's Algorithm. So its time complexity is $O(|E| + |V| \log |V|)$

Remark 2.13. The first step need time complexity $O(|E| \log |E|)$.

The second step need time complexity $O(|E| \cdot \alpha(|V|))$ using **Union-Find** data structure.

Algorithm 3 Kruskal's Algorithm

1: Consider edges in weight increasing order.
2: Add each edge to T if not introducing a cycle.

WLOG we can assume edge weights are distinct.

Algorithm 4 Boruvka's Algorithm

1: **while** $< (n-1)$ blue edges **do**
2: Simultaneously apply blue rule to each blue compunent.
3: **end while**

Claim. WHILE loop iterates $\leqslant O(\log |V|)$.

So time complexity is $O(|E| \log |V|)$.

Remark 2.14. There is a "contraction View". For each step, we can view each component as a single point with edges to other components.

If the graph is **Planar Graph**, then $|E| \leqslant O(V)$. At the i-th WHILE iteration, $|V_i| \leqslant \frac{|V|}{2^{i-1}}$, $|E_i| \leqslant O(|V_i|)$.

So the time complexity is $\sum_i O(|E_i|) \leqslant \sum_i O\left(\frac{|V|}{2^{i-1}}\right) \leqslant O(|V|)$ which is linear!

Using the contraction view, we can get another algorithm:

Prim+Boruvka

- Run Boruvka for k iterates.
- Run Prim on the contracted graph.

Remark 2.15.

For step 1, time complexity is $k \cdot |E|$.

For step 2, time complexity is $|E| + \frac{|V|}{2^k} \cdot \log \frac{|V|}{2^k}$.

So the total time complexity is $k|E| + \frac{|V|}{2^k} \cdot \log \frac{|V|}{2^k}$.

Choose $k = \log_2 \log_2 |V|$, it comes to $(\log\log |V|) \cdot |E| + \frac{|V|}{\log_2 |V|} \cdot \log_2 |V| \leqslant O(|E| \log\log |V| + |V|)$.

2.5 Minimum Arborescence

Example 2.16 (Minimum Arborescence)**.** Input: Directed $G = (V, E)$, source $s \in V$ and weight $\omega : E \to \mathbb{R}$.

We want to find a directed tree $T = (V, E)$ with root s of minimum total weight.

Index

List of Theorems