Approximation Algorithms 1: Vertex Cover and MAX-3SAT

Yufei Tao

In computer science, there is a set of **NP-hard** problems such that

nobody has found a polynomial-time algorithm for **any** of those problems;

no polynomial-time algorithms can exist for **any** of those problems **unless** P = NP.

P = the set of problems that can be solved in polynomial time on a **deterministic** Turing machine

NP = the set of problems that can be solved in polynomial time on a **non-deterministic** Turing machine

Turing machines are formalized in CSCI3130 (Formal Languages and Automata Theory), and so is the notion of NP-hard.

Whether P = NP is still unsolved to this day.

What can we do if a problem is NP-hard?

The rest of the course will focus on a principled approach for tack- ling NP-hard problems: **approximation**.

In many problems, even though an optimal solution may be expensive to find, we can find **near-optimal** solutions efficiently.

Next, we will see two examples: **vertex cover** and **MAX-3SAT**.

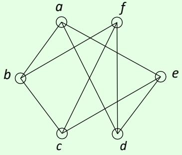
The Vertex Cover Problem

$G = (V, E)$ is a simple undirected graph.

A subset $S \subseteq V$ is a **vertex cover** of G if every edge $\{u, v\} \in E$ is incident to at least one vertex in S.

The V.C. Problem: Find a vertex cover of the smallest size.

Example:



An optimal solution is $\{a, f, c, e\}$.

The vertex cover problem is NP-hard.

No one has found an algorithm solving the problem in time polynomial in $|V|$.

Such algorithms cannot exist if P $\neq$ NP.

Approximation Algorithms

A = an algorithm that, given any legal input $G = (V, E)$, returns a vertex cover of G.

OPT_G = the smallest size of all the vertex covers of G.

A is a ρ**-approximate algorithm** for the vertex cover problem if, for any legal input $G = (V, E)$, A can return a vertex cover with size at most $\rho \cdot OPT_G$.

The value ρ is the **approximation ratio**.
We say that A achieves an approximation ratio of ρ.

Consider the following algorithm.

Input: $G = (V, E)$

$S = \emptyset$

while E is not empty **do**

pick an arbitrary edge $\{u, v\}$ in E

add u, v to S

remove from E all the edges of u and all the edges of v

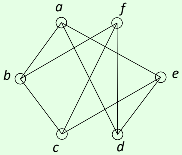
return S

It is easy to show:

- S is a vertex cover of G;
- The algorithm runs in time polynomial to $|V|$ and $|E|$.

We will prove later that the algorithm is 2-approximate.

Example:



Suppose we start by picking edge $\{b, c\}$.
Then, $S = \{b, c\}$ and $E = \{\{a, e\}, \{a, d\}, \{d, e\}, \{d, f\}\}$.

Any edge in E can then be chosen. Suppose we pick $\{a, e\}$. Then, $S = \{a, b, c, e\}$ and $E = \{\{d, f\}\}$.

Finally, pick $\{d, f\}$.
$S = \{a, b, c, d, e, f\}$ and $E = \emptyset$.

Theorem 1: The algorithm returns a set of at most $2 \cdot OPT_G$ vertices.

Let M be the set of edges picked.

Example: In the previous example, $M = \{\{b,c\},\{a,e\},\{d,f\}\}$.

Lemma 1: The edges in M do not share any vertices.

Proof: Suppose that M has edges e_1 and e_2 both incident to a vertex v. W.l.o.g., assume that e_1 was picked before e_2. After picking e_1, the algorithm deleted all the edges of v, because of which e_2 could not have been picked, giving a contradiction. $\square$

Lemma 2: $|M| \le OPT_G$.

Proof: Any vertex cover must include at least one vertex of each edge in M. $|M| \le OPT$ follows from Lemma 1. $\square$

Theorem 1 holds because the algorithm returns exactly $2|M|$ vertices.

The MAX-SAT Problem

A **variable**: a boolean unknown x whose value is 0 or 1. A **literal**: a variable x or its negation $\bar{x}$.
A **clause**: the OR of 3 literals with different variables.

S = a set of clauses
X = the set of variables appearing in at least one clause of S
A **truth assignment** of S : a function from X to {0, 1}.

A truth assignment f **satisfies** a clause in S if the clause evaluates to 1 under f.

The MAX-3SAT Problem: Let S be a set of n clauses. Find a truth assignment of S to maximize the number of clauses satisfied.

Example:

- $S = \{x_1 \vee x_2 \vee x_3,$
- $x_1 \vee x_2 \vee \bar{x}_3,\ x_1 \vee \bar{x}_2 \vee x_3,\ x_1 \vee \bar{x}_2 \vee \bar{x}_3,\ \bar{x}_1 \vee x_3 \vee x_4,\ \bar{x}_1 \vee x_3 \vee \bar{x}_4,\ \bar{x}_1 \vee \bar{x}_3 \vee x_4.\ \bar{x}_1 \vee \bar{x}_3 \vee \bar{x}_4\}.$

- $n = 8$ and $X = \{x_1, x_2, x_3, x_4\}$.

- The truth assignment $x_1 = x_2 = x_3 = x_4 = 1$ satisfies 7 clauses. It is impossible to satisfy 8.

The MAX-3SAT problem is NP-hard.

No one has found an algorithm solving the problem in time polynomial in n.

Such algorithms cannot exist if P $\neq$ NP.

Approximation Algorithms

A = an algorithm that, given any legal input S, returns a truth assignment of S.

OPT_S = the largest number of clauses that a truth assignment of S can satisfy.

Z_S = the number of clauses satisfied by the truth assignment A returns.

Z_S is a random variable if A is randomized.

A is a **randomized ρ-approximate algorithm** for MAX-3SAT if $\boldsymbol{E}[Z_S] \geq \rho \cdot OPT_S$ holds for any legal input S.

The value ρ is the **approximation ratio**.
We also say that A achieves an approximation ratio of ρ in expectation.

Consider the following algorithm.

Input: a set S of clauses with variable set X

for each variable $x \in X$ **do**
toss a fair coin
if the coin comes up heads **then** $x \leftarrow 1$
else $x \leftarrow 0$

It is clear that the algorithm runs in $O(n)$ time.
Next, we show that the algorithm achieves an approximation ratio $7/8$ in expectation.

Job-shop Scheduling

- n jobs m machines
- No recirculation
  - Jobs do not revisit the same machine
- (i, j) is referred to as an operation in which job j is processed on machine i
- Processing time is p_{ij}
- Objective is to minimize Cmax – makespan – max completion time
- Jobs have a machine sequence
- Find the sequence for each machine

Training matrix analogous to Job shop Scheduling

- There are n trainees
- There are m departments
- Each trainee has a pre-determined sequence based on their qualification
- Each trainee spends a different number of weeks in each department based on their final placements.
- Find a schedule that minimizes the completion time of all required training for this new batch of trainees.

Hospital sequencing

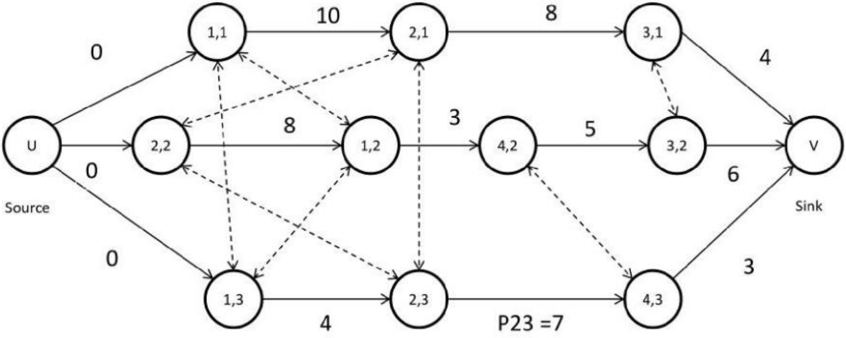
- Medical depts have equipment, doctors which are like the machines
- Jobs are patients
- Each patient has a pre-determined sequence for testing and consultation based on their ailment
- Each patient spends a different amount of time in each medical department (with equipment, doctors) based on their ailment.
- Find a schedule that minimizes the completion time of all patient diagnostics.

Job-shop Scheduling

Jobs	m/c seq	pij
1	1 2 3	$p_{11}=10$, $p_{21}=8$, $p_{31}=4$
2	2 1 4 3	$p_{22} = 8$, $p_{12}=3$, $p_{42}=5$, $p_{32}=6$
3	1 2 4	$p_{13}=4$, $p_{23}=7$, $p_{43}=3$

Job-shop Scheduling

- Conjunctive (solid arcs) and disjunctive (dotted arcs)
- Conjunctive – machine sequence for a job
- Disjunctive – job sequence for a machine



- 1 sink because the completion time of the last job is important

Job-shop Scheduling

- IP solution
- Let y_{ij} be starting time of job j on machine i

Minimize Cmax

s.t.

$$y_{kj} - y_{ij} \ \ >= \ p_{ij} \quad \text{for all } (i,j) \longrightarrow (k,j)$$

$$Cmax \ - \ y_{ij} \ \ >= \ p_{ij} \quad \text{for all } (i,j)$$

y_{ij}-y_{il} >= p_{il} or y_{il} –y_{ij} >= p_{ij} for all (i,l) and (i,j) i = 1……m

y_{ij} >= 0

- Solve using branch and bound
  - Computationally prohibitive for large n and m

Shifting Bottleneck Heuristic

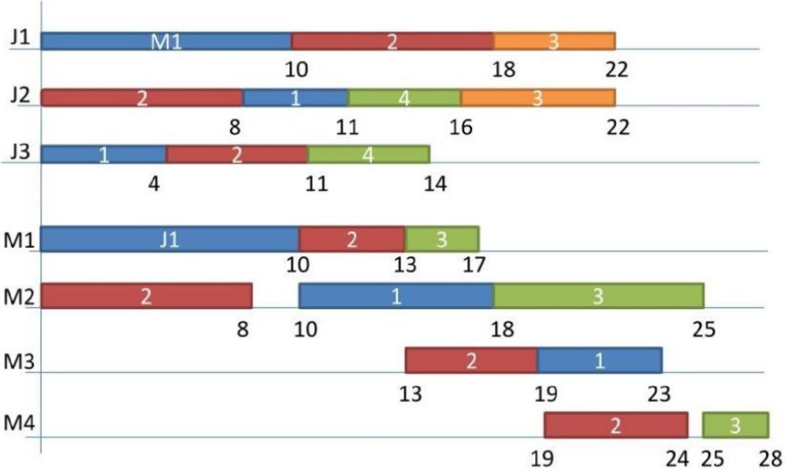
- Very efficient heuristic for n job m machine job shop with jobs having a pre-determined sequence
- Has been proven to be very close to optimal, which has been verified numerous times with the branch and bound optimal search.
- Proven to be very fast compared to B&B

Shifting Bottleneck Heuristic – completion time

- M is the set of all machines
- Mo is the set of machines for which the sequence has been determined
- An iteration results in selecting a machine from M-Mo for inclusion in Mo.
  - Each machine in M-Mo is considered as a single machine problem with release and due dates for which the maximum lateness is to be minimized (Lmax)
  - Then the machine with the largest Lmax is chosen and is termed as a bottleneck. This is included in Mo
  - Update Cmax = Cmax + Lmax
  - Re-sequence all machines in Mo-the last machine added.
  - Continue until M-Mo is a null set.
- Release date of job j on machine i is the longest path from source to node (i,j)
- Due date of job j on machine i is the longest path from node (i,j) to sink – p_{ij} and the resultant is subtracted from Cmax of set Mo

Shifting Bottleneck Heuristic

- Gantt Chart



See handout for solution

Composite Dispatching rules

- ATC Apparent tardiness cost
- ATCS Apparent tardiness cost with set up
  - See page 446

Shifting Bottleneck Heuristic – weighted tardiness

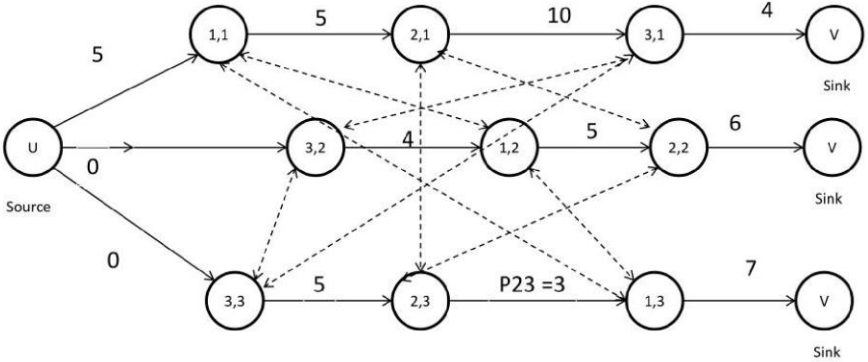
- M is the set of all machines
- Mo is the set of machines for which the sequence has been determined
- An iteration results in selecting a machine from M-Mo for inclusion in Mo.
  - Each machine in M-Mo is considered as a single machine problem with release and due dates for which a priority index is calculated $I_{ij}(t)$
  - $I_{ij}(t)$ is computed using the ATC rule (Apparent Tardiness Cost)
  - Sequence jobs on the machine with the highest to lowest $I_{ij}(t)$
  - Calculate weighted tardiness
  - Then the machine with the largest weighted tardiness is chosen and is termed as a bottleneck. This is included in Mo
  - Re-sequence all machines in Mo-the last machine added.
  - Continue until M-Mo is a null set.
- Release date of job j on machine i is the longest path from source to node (i,j)
- Due date of job j on machine i is the longest path from node (i,j) to sink – p_{ij} and the resultant is subtracted from Cmax of set Mo
  - If a path does not exist then make it infinity.

Shifting Bottleneck Heuristic – weighted tardiness

Jobs	wj	rj	dj	m/c seq	pij
1	1	5	24	1 2 3	p11=5, p21=10, p31=4
2	2	0	18	3 1 2	p32 = 4, p12=5, p22=6
3	2	0	16	3 2 1	p33=5, p23=3, p13=7

Shifting Bottleneck Heuristic – weighted tardiness

- 3 sinks because the completion time of all jobs is important



See handout for solution

Shifting Bottleneck Heuristic – weighted tardiness

- ATC Rule:

$$I_{ij}(t) = \sum_{k=1}^{n} \frac{w_k}{p_{ij}} \exp\left(-\frac{\max(d_{ij}{}^{k} - pij + rij - t, 0)}{K\bar{p}}\right)$$

  - t is the earliest time at which machine i can be used
  - K is a scaling parameter
  - $\bar{p}$ is the average processing time for machine i

- Weighted Tardiness

$$= \sum_{k=1}^{n} w_k \left(C_k{}'' - Ck'\right) \exp\left(-\frac{\max(d_k - C_k{}'', 0)}{K}\right)$$

C_k' is the completion time of job k at the beginning of an iteration

C_k'' is the new completion time of job k at the end of an iteration

Software for Job shop scheduling

- LEKIN
- Summary
  - Single machine scheduling
    - With or without setup time
  - Parallel machines (machines can process all jobs)
  - Flow shop (All jobs have to flow first on one machine and then on another)
  - Job-shop
    - Minimize Completion time
    - Minimize Weighted Tardiness
  - Flexible flow shop
- Methods
  - Dispatching rules, Tabu, Simulated Annealing, Shifting bottleneck heuristics for completion time and weighted tardiness objective.

PRACTICE MCQs

- Consider a simple graph G with 18 vertices. What will be the size of the maximum independent set of G, if the size of the minimum vertex cover of G is 10?

a) 8
b) 18
c) 28
d) 10

- Which among the following problem uses the vertex cover approach?

a) Traveling salesperson problem
b) Assignment problem
c) Activity selection problem
d) Knapsack problem