assignment 2
tdt4125 - algorithm construction

fredrik robertsen

March 2, 2026

task 1

this sounds like the vertex cover problem, but with additional vertex weights to minimize.

a) recall the general key components of a linear program

- problem input vector x
- coefficient vector c
- constraint matrix A
- target vector b

such that we minimize $c^\mathsf{T} x$ with respect to $Ax \geq b$.

here we can interpret the input vector x as a binary vector where the i-th bit encodes the inclusion of a given node $v_i \in V$ in the subset C. x has as many bits as there are vertices.

furthermore, the vector c represents the weights of the vertices such that

$$c^\mathsf{T} x = \sum_{v \in V} c_v x_v = \sum_{v \in C} c_v$$

becomes the minimization goal.

we can express the binary string x through constraints

- $x_u + x_v \geq 1 \quad \forall (u, v) \in E$
- $x_v \in \{0, 1\} \quad \forall v \in V$

the former constraint ensures that every edge is covered by at least one endpoint.

bringing it all together

$$\begin{aligned} \min & \sum_{v \in V} c_v x_v \\ \text{s.t.} \quad & x_u + x_v \geq 1 \quad \forall (u, v) \in E \\ & x_v \in \{0, 1\} \quad \forall v \in V \end{aligned}$$

is our complete linear integer program.

we have implicitly defined A to be the $|E| \times |V|$-matrix that has 1-entries for vertices v where $e = (u, v)$. it is zero otherwise.

b) by relaxing the linear integer program we let $0 \leq x_v \leq 1$. this can be shortened to saying $x_v \geq 0$ since the first constraint forces $x_v \leq 1$. requiring a positive value is a common restriction on linear programs and allows us to take the dual.

we then obtain the dual program

$$\begin{aligned} \max & \mathbf{1}^\mathsf{T} y = \sum_{e \in E} y_e \\ \text{s.t.} \quad & A^\mathsf{T} y \leq c \\ \Leftrightarrow & \sum_{v \in e} y_e \leq c_v \\ & \forall v \in V, \quad y \geq 0 \end{aligned}$$

c) to construct a primal-dual algorithm to approximate this problem, we use the formulations given previously.

Algorithm 1: Primal-Dual Weighted Vertex Cover Approximation

```
1: procedure VERTEX-COVER(G, c)
2:     ▷ initialize primal (x) and dual (y) variables
3:     for v ∈ V, e ∈ E do
4:         y_e ← 0
```

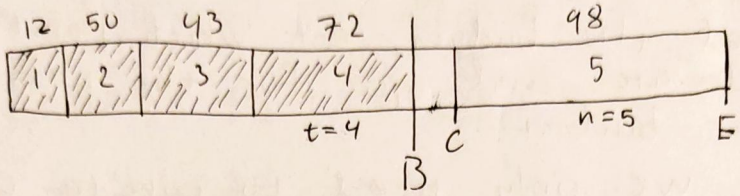
```
    |  x_v ← 0
    end
    
    while there exists an uncovered edge e = (u, v) do
    |  ▷ calculate minimum slack
    |  slack_u ← c_u − Σ_{e′∈δ(u)} y_{e′}
    |  slack_v ← c_v − Σ_{e′∈δ(v)} y_{e′}
    |  α ← min(slack_u, slack_v)
    |
    |  ▷ increase dual variable for edge e
    |  y_e ← y_e + α
    |
    |  ▷ Update primal if constraint becomes tight
    |  if Σ_{e′∈δ(u)} y_{e′} = c_u then
    |  |  x_u ← 1
    |  end
    |  if Σ_{e′∈δ(v)} y_{e′} = c_v then
    |  |  x_v ← 1
    |  end
    end
    return C = {v ∈ V | x_v = 1}
end
```

where $\delta(u)$ means the set of all edges going out of vertex u.

other

the rest of the tasks are attached or can be found at https://git.pvv.ntnu.no/frero-uni/TDT4125/src/branch/main/assignment-2

Task 2 a) greedy cheapest first. reminds me of OS-schedaling: SJF.

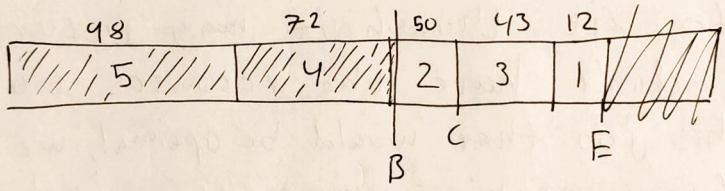


With n projects, assume t is the last one. Resources required to complete t is C, such that $B \leq C$. To complete all projects, E.

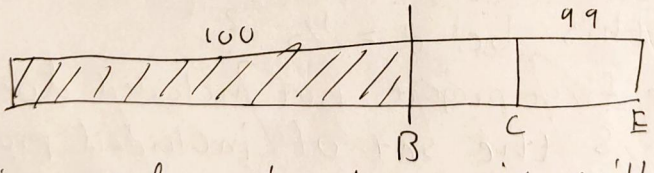
If all projects had a uniform profit value, this would be optimal, but we are disregarding the profits completely now, thus we are always less than or equal to the optimum.

Because of this, I posit that there is no constant approximation factor.

b)



Consider:

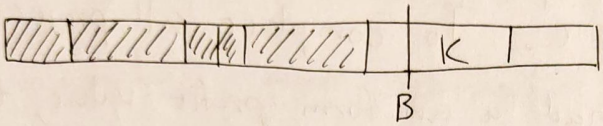


This greedy algorithm is still not optimal, since it disregards project costs.
Thus I still don't think we are quite able to show any approximation factors.

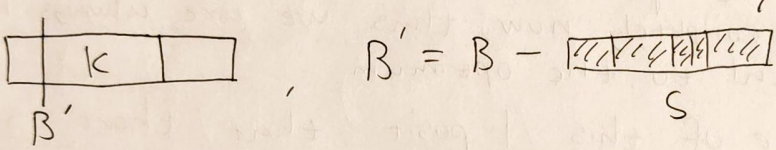
c) We have to consider both profit and cost, so sort decreasing by density p_i/r_i. Then, all resources spent will be spent optimally. Sorting is polynomical time, so the greedy algorithm is too.

d) With $x_i \in \{0, 1\}$, we can still attempt the algorithm from c). We still take the highest density jobs until we meet one that would exceed the budget, at which point we must deviate and opt for the next project that fits our budget.

I'm saying we only need to consider what happens from after a problematic project shows up:



Can be seen as / reduced to



However in this case, we can't find another project to fit, though it may occur. Since we don't have the resources to complete this job that would be optimal, we have to settle for something worse, thus inducing an approximation factor.

Why should this be $\alpha = 1/2$?

Let k be the first project not included due to budget violation, S the set of included projects. The optimum can either include k or not.

1. if $p_k \geq \sum_{i \in S} p_i$

It shouldn't

The approximation is almost correct, we need to account for a case like

$p_1 = 1 \quad r_1 = 1$, ~~$p_2 = 1$~~, p_m

~~$p_2 = 0.95$~~ $\quad p_2 = 1.95 \quad r_2 = 2$, $\quad B = 2$

then sorting by p_i/r_i yields project 1 first, but then we don't have funds for 2, which should've been picked instead since it is optimum.

Thus, modify the algorithm to take

$$\max\left(\sum_{i \in S} p_i \;,\; p_k\right),$$

where S is selected projects before skipping and k is the first skipped project.

This achieves an $\alpha = 1/2$ ~~due to 2 cases.~~

We have $OPT \leq OPT_{frac} \leq \sum_{i \in S} p_i + p_k$, where OPT is this binary problem and OPT_{frac} is a)–c) where we allowed fractional solutions. These optima are bounded by

$\sum_{i \in S} p_i + p_k > B$, since these are the greedy highest profit projects. but

$\sum_{i \in S} p_i \geq 0$ and $p_k \geq 0$, so

$$\sum_{i \in S} p_i + p_k \leq 2 \cdot \underbrace{\max\left(\sum_{i \in S} p_i \;,\; p_k\right)}_{APX}$$

$\Rightarrow OPT \leq 2 \cdot APX \Rightarrow \alpha = 1/2$

Task 3

Let $G = (V, E)$ be an undirected graph with $w_{uv} > 0 \ \forall \ (u,v)$ in E.

Def a 'cut (A, B)' satisfies $A \subseteq V$, $B = V \setminus A$.

$$cut(A,B) = \sum_{\substack{(u,v)\in E \\ u\in A, v\in B}} w_{uv}$$

---

a) SearchMaximumCut

Let (A, B) be a partition of vertices V of graph G
While $\exists\, u \in A$ s.t. $\Delta u > 0$
move u to B
return $cut(A, B)$
where

$$\Delta u := \sum_{v\in B} w_{uv} - \sum_{v\in A} w_{uv},$$

Since $\Delta u > 0$ and not ≥ 0, the algorithm terminates. Also since we are always moving an element, assuming a finite graph, we must be able to run out of vertices, ending the search.

b) Assume SearchMaximumCut $\rightarrow cut(A, B)$ and the global optimum is $cut(A^*, B^*)$

Goal $cut(A, B) \geq \frac{1}{2} cut(A^*, B^*)$.

We know $\forall u$, $\Delta u \leq 0$ at local optimum.

Then for $u \in A$

$$\sum_{v\in B} w_{uv} \leq \sum_{v\in A} w_{uv}$$

Sum all $u \in A$

$$\underbrace{\sum_{u\in A}\sum_{v\in B} w_{uv}}_{cut(A,B)} \leq \underbrace{\sum_{u\in A}\sum_{v\in A} w_{uv}}_{*} = 2W_A,$$

W_A is the sum of weights in A.

$*$: since $w_{uv} = w_{vu}$, we are summing twice, hence $2W_A$.

We obtain the same for $u \in B$:

$cut(A, B) \leq 2W_B$, adding these

$2\, cut(A,B) \leq 2(W_A + W_B)$

but $W_A + W_B + cut(A, B) = \sum_{(u,v)\in E} w_{uv}$, then

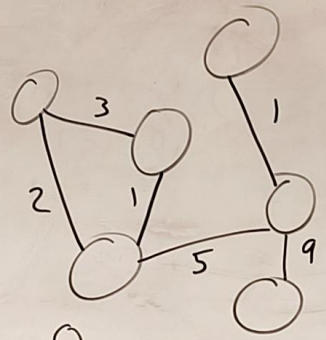
$$cut(A,B) \leq \sum_{(u,v)\in E} w_{uv} - cut(A,B)$$

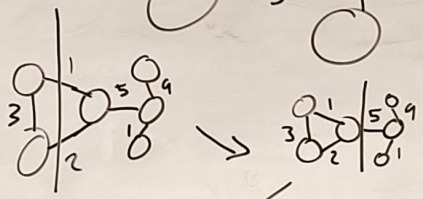
$$2\, cut(A,B) \leq \sum_{(u,v)\in E} w_{uv}$$

close enough

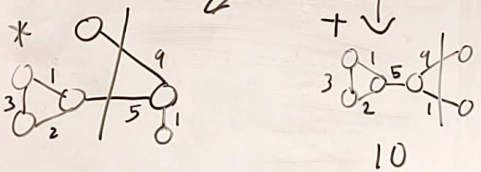
Task 3

c)





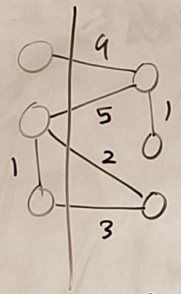
3 5



14 10

Both of these are local optima.

If the algorithm didn't terminate at * or †, we could continue from * to find global optimum by "switching directions".



19

d) Trivially, it can be seen that SEARCHMAXIMUMCUT has a time complexity of $O(|V|)$, if we start with a cut like $A = \emptyset$, $B = V$, we could have to process $|V|$ nodes.

I have probably missed something...

Task 4

I'm not a mathematician yet...

Let $G=(V,E)$ be a bidirectional, connected graph.

Def its cocycle matroid

$M^*(G)=(E,\mathcal{I})$,

$\mathcal{I}=\{I\subseteq E \mid$ the subgraph $(V, E\setminus I)$ is connected$\}$.

Goal Prove $M^*(G)$ is a matroid.

Plan:

1. recover $M(G)$ primal
2. show $M(G)$ is a matroid
3. profit.

1. the mention of a cocycle matroid implies the existence of a cycle matroid $M(G)$. By dualities we have $M(G)=(M^*(G))^*$.
   Looking online, I find that $I\subseteq E$ is independent in $M(G)$ iff I is contained in a complement of a basis of $M^*(G)$.
   The bases of $M^*(G)$ are maximal† I s.t. $(V, E\setminus I)$ is connected $\Rightarrow E\setminus I$ is a spanning tree.
   This is reasonable, since removing as many edges as possible from a connected graph s.t. we still have a connected graph should yield a spanning tree.
   Since $E\setminus I$ are spanning trees of the bases of $M^*(G)$, we have
   $M(G)=(E,\mathcal{I})$,
   $\mathcal{I}=\{I\subseteq E \mid I$ is a spanning forest$\}$ (independent sets formalities)
   i.e. there are no cycles. This is the cycle matroid.
2. use the cryptomorphic circuit axioms of the family $\mathcal{C}$
   (C1) $\emptyset\notin\mathcal{C}$ no empty cycle ✓
   (C2) $C_1,C_2\in\mathcal{C}, C_1\subseteq C_2\Rightarrow C_1=C_2$ cycles are minimal; start anywhere ✓
   (C3) $C_1\neq C_2\in\mathcal{C}$, $e\in C_1\cap C_2$, then $\exists\, C_3\in\mathcal{C}$ s.t. $C_3\subseteq(C_1\cup C_2)\setminus\{e\}$ Cycle elimination; removing a common edge allows splicing into new cycle. ✓
   These hold in undirected graphs, so $M(G)$ is a matroid.
3. Thus, $M^*(G)$ is also a matroid by dualities. ▨

† recall that primal=dual for optimum. I think that is relevant for this maximal magic to convert to primal