TDT4125 Algoritmekonstruksjon

Eksamen, 15. mai 2025, 15:00–19:00

Faglig kontakt	Magnus Lie Hetland
Hjelpemiddelkode	E

Løsningsforslag

Løsningsforslagene i rødt nedenfor er *eksempler* på svar som vil gi full uttelling. Det vil ofte være helt akseptabelt med mange andre, beslektede svar, spesielt der det bes om en forklaring eller lignende. Om du svarte noe litt annet, betyr ikke det nødvendigvis at du svarte feil!

1

Hva er dualen av følgende lineærprogram?

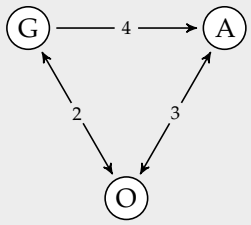
Bruk y_i som navn på dualvariablene.

Det er viktig at du *ikke* normaliserer eller skriver om primalen, f.eks. ved å endre fortegn, legge til variable eller restriksjoner. Du skal finne dualen av lineærprogrammet *direkte*, *nøyaktig som det er skrevet*!

$$
\begin{aligned}
\min \quad & 4x_1 + 5x_2 \\
\text{s.t.} \quad & x_1 + 2x_2 = 2\,, \\
& x_1 + x_2 \geqslant 3\,, \\
& x_1 \qquad\quad \geqslant 0\,, \\
& \qquad\quad x_2 \leqslant 0\,.
\end{aligned}
$$

$$
\begin{aligned}
\max \quad & 2y_1 + 3y_2 \\
\text{s.t.} \quad & y_1 + y_2 \leqslant 4\,, \\
& 2y_1 + y_2 \geqslant 5\,, \\
& \qquad\quad y_2 \geqslant 0\,.
\end{aligned}
$$

Figur 1



I de følgende tre oppgavene skal du diskutere forholdet mellom grådighet (G), approksimasjon (A) og onlinealgoritmer (O), som angitt i figur 1.

2 Hvordan vil du beskrive slektskapet mellom grådige algoritmer og onlinealgoritmer? Hva er felles for dem, og hva er forskjellig?

Det er ikke nødvendig å skrive langt. Man kan få med de sentrale ideene med noen få korte setninger.

Det sentrale likhetspunktet er at begge typer algoritmer tar en serie med bindende beslutninger underveis, basert på lokal informasjon. En forskjell er at grådighet alltid baserer seg på en eller annen form for lokal optimalitet, mens onlinealgoritmer ikke har noen slik standardmekanisme. Til gjengjeld er onlinealgoritmer begrenset til å kun ha deler av instansen tilgjengelig underveis, noe som ikke er en definerende egenskap ved grådighet.

3 Hvordan vil du beskrive slektskapet mellom approksimasjonsalgoritmer og onlinealgoritmer? Hva er felles for dem, og hva er forskjellig?

Det er ikke nødvendig å skrive langt. Man kan få med de sentrale ideene med noen få korte setninger.

Begge gir tilnærmet optimale løsninger, innen en faktor α. For onlinealgoritmer sammenlignes løsningen med den man får om man har tilgang til hele instansen fra starten (offline), mens for approksimasjonsalgoritmer sammenlignes løsningen med den man får om man har ubegrenset med tid tilgjengelig.

Ev.: I begge tilfeller sammenligner man med optimum, der man spesifikt for onlinealgoritmer fjerner online-kravet for å finne optimum.

4 Hva kan du si om bruken av grådighet i approksimasjonsalgoritmer?

Her kan du få poeng for følgende tema, som angitt, opp til 10 poeng totalt:

2 Ryggsekkproblemet
2 Nodedekke
2 Mengdedekke
3 Handelsreiseproblemet (TSP)
3 k-senter-problemet
3 Sekvensering (*scheduling*)
5 Uavhengighetssystemer (*independence systems*)

Det som er oppgitt i tabellen er hvor mange poeng du maksimalt kan få per tema. Hvis poengsummen blir høyere enn 10, får du 10 poeng.

Noen punkter man gjerne kan dekke for de ulike temaene:

Ryggsekkproblemet: Grådig approksimasjon gir $\alpha = 1/2$ for versjonen der verdi og vekt er det samme. (Man får også full uttelling om man svarer $\alpha = 2$, siden det er det som står i den relevante delen av pensum.)

Man kan også trekke inn at for det ubegrensede ryggsekkproblemet, der vekt og verdi kan være forskjellig, men man har vilkårlig mange av hver type, gir grådighet over kilopris også $\alpha = 1/2$. Det samme gjør det begrensede problemet, om man kan velge mellom den grådige løsningen og den første gjenstanden som ikke fikk plass. Disse tingene er utenfor pensum, og krever vil kreve litt forklaring.

Man kan selvfølgelig også nevne at det finnes en PTAS og en pseudopolynomisk algoritme, bruk av primal–dual-metoden, etc., men det er ikke det det spørres om her, så det vil i liten eller ingen grad gi noe ekstra uttelling.

Nodedekke: Om man velger noder som dekker flest mulig udekkede kanter, kan man få en vilkårlig dårlig løsning. Om man heller finner en maksimal (dvs., ikke utvidbar) matching, også vha. grådighet (med bindende, om enn vilkårlige, valg) og inkluderer nodene fra disse kantene, får man $\alpha = 2$. (Hver kant må ha med én av sine noder; om vi tar med begge, får vi maksimalt en dobling.)

Mengdedekke: Om man velger grådig etter lavest vekt, fordelt på udekkede elementer (ev. bare flest udekkede elementer, i det uvektede tilfeller), får man $\alpha = H_n = O(\lg n)$ (eller mer presist, $H_g = O(\lg g)$, der n er antall elementer totalt og g er antall elementer i den største mengden).

Her er det også en kobling til multiplikativ vektoppdatering (*multiplicative weights update*), der en bruken av $\eta = 1$ i praksis gir den grådige algoritmen for mengdedekke, med en tilsvarende garanti.

Handelsreiseproblemet: Her er det flere approksimasjonsalgoritmer som kan nevnes, for TSP i metriske rom, primært nærmeste utvidelse (*nearest addition*) og dobbel-tre-algoritmen (begge $\alpha = 2$) og Christofides' algoritme ($\alpha = 1.5$). Grådigheten i de siste to finner vi i metodene for konstruksjonen av minimale spenntrær.

Man kan også naturligvis nevne at problemet er vanskelig å approksimere

for generelle lengder/vekter, men det er ikke direkte relevant for grådighet.

k-**senter-problemet:** Her velger man hele tiden et nytt senter som er så lengst unna de foregående som mulig (altså, der minimums-avstanden er størst mulig), noe som er sentralt i beviset for at algoritmen gir $\alpha = 2$).

Sekvensering: Scenarioet som er diskutert i pensum er fordeling av jobber på identiske maskiner, og grådigheten handler om å hele tiden gi neste jobb (i vilkårlig rekkefølge) til den maskinen med lavest total arbeidstid, noe som gir $\alpha = 2$.

Man kan forbedre dette til $\alpha = 4/3$ ved å være «enda grådigere», og hele tiden plassere den gjenværende jobben med lengst prosesseringstid.

Det er også en metode basert på lokalt søk i pensum, som går ut på å flytte jobben som slutter sist til maskinen med lavest totaltid. Denne gir også $\alpha = 2$, som gir oss et bevis for den grådige algoritmen, siden det lokale søket ikke vil kunne forbedre den grådige løsningen.

Uavhengighetssystemer:

For et uavhengighetssystem $M = (E, \mathbf{S})$, gir grådighet for maksimering $\alpha = \text{rq}(M)$, der rq er *rangkvotienten* (*rank quotient*), altså det minste forholdet mellom nedre og øvre rang (minste og største kardinalitet for ikke utvidbare uavhengige systemer) over alle delmengder av E.

Ethvert uavhengighetssystem kan også beskrives som snittet av k matroider (for varierende k), og grådighet vil da gi $\alpha = k$.

Det kreves ikke, men kan telle positivt, om man kommer med eksempler på rangkvotienter eller snitt av matroider. Man kan også nevne at man kan optimere over snitt av matroider i polynomisk tid for $k = 2$, men det er ikke direkte relevant for grådighet.

Her kan man også godt nevne at grådighet for matroider gir optimum, men det er ikke så relevant for approksimasjon.

5 Forklar ytelsesgarantien (α) gitt av primal–dual-metoden, når den brukes på mengdedekkeproblemet, som beskrevet i pensum.

Primalvariable rundes bare opp når dualrestriksjonene er stramme, så et ev. avvik kan kun komme i dual komplementær slakkhet. Forskjellen mellom høyre og venstre side i restriksjonene der er maksimalt en faktor f (antall mengder elementet forekommer i), som da også gir ytelsesgarantien.

6 Din venn Lurvik studerer følgende problem:

> Gitt en graf G og et heltall $k \geqslant 1$, finn, om mulig, et spenntre T for G, slik at ingen av nodene er tilknyttet flere enn k av kantene i T.

Han mener problemet kan uttrykkes som snittet av to matroider, der den ene er den grafiske matroiden for G, og den andre legger en begrensning k på antall

nabokanter for hver node, selv om han ikke er helt sikker på hvordan denne andre matroiden skal formuleres.

Hva tenker du om saken? Diskuter og forklar.

Den grafiske matroiden for $G = (V, E)$ er altså mengdesystemet $(E, \mathbf{S})$ der mengdene i $\mathbf{S}$ er skoger.

Dersom man setter $k = 2$ vil man her kunne avgjøre om grafen har en hamiltonsti (et spenntre der ingen noder har flere enn to naboer), som er et NP-komplett problem. Siden man kan løse denne typen problemer for snittet av to matroider i polynomisk tid, vil Lurviks påstand medføre at $P = NP$, som er svært lite plausibelt.

Man kan også relativt lett redusere fra hamiltonsykel til hamiltonsti, der hamiltonsykel er eksplisitt diskutert som NP-komplett i pensum.

Det kan også være relevant å nevne f.eks. at det er mulig å uttrykke problemet med å finne en *rettet* hamiltonsykel som snittet av *tre* matroider, som vi *ikke* har polynomiske algoritmer for.

Her kan man naturligvis diskutere hvorfor den andre matroiden ikke faktisk er en matroide, selv om det kan være litt utfordrende å gjendrive den påstanden, så lenge man ikke vet hvordan den er tenkt formulert.

Den enkleste strategien vil være å rett og slett forby flere enn k naboer, som i en partisjonsmatroide, men der man ikke har en faktisk partisjon (siden nodene deler kanter). Man kan så vise at det resulterende systemet bryter noen av kravene til en matroide. Det viser kunnskap om matroider, men er ikke nødvendigvis et helt vanntett argument uten i hvert fall noe diskusjon om at Lurvik ikke kan oppnå ønsket resultat på andre vis.

7 Du skal avgjøre om et kart med n regioner har k regioner som ikke er naboer (en *uavhengig mengde*). Beskriv en kjerne (*kernel*) av størrelse $O(n)$, som kan finnes i polynomisk tid.

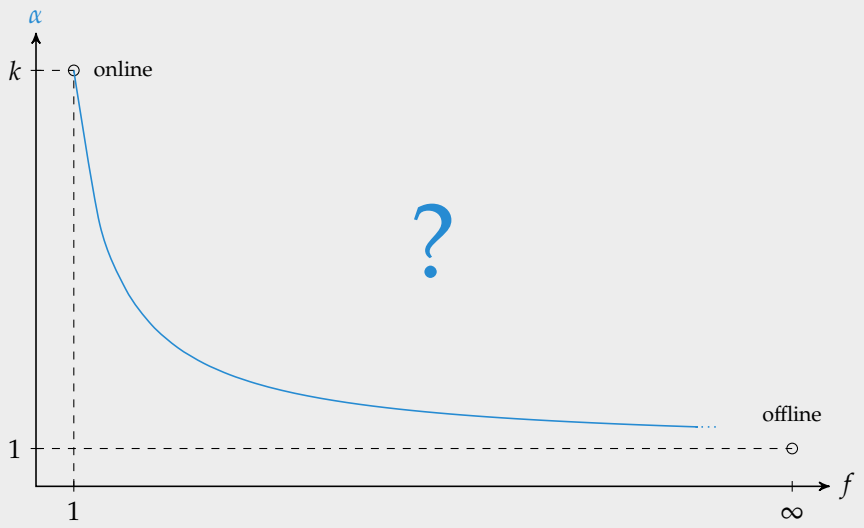
Hint: Et kart kan alltid fargelegges med maks 4 farger i polynomisk tid, slik at ingen naboregioner har samme farge.

Kartet representeres som en graf med kanter mellom naboregioner.

Fargelegg kartet. Hver farge utgjør en uavhengig mengde, og minst én av dem har størrelse minst $n/4$. Hvis $k < n/4$, er svaret *ja*. Hvis ikke, er $n \leqslant 4k$, og kartet er selv en kjerne av størrelse $O(n)$.

8 Din venn Klokland prøver å finne bedre metoder for sideveksling (*paging*). Dere er enige om at med cachestørrelse k, vil en deterministisk onlinealgoritme aldri kunne være bedre enn k-kompetitiv (dvs., ha en *competitive ratio* $\alpha < k$). Hun mener hun kan forbedre dette med *lookahead*, dvs. at algoritmen får vite hva de f neste forespørslene er.

Figur 2



Hun har prøvd å tegne opp sin egen intuisjon for problemet i figur 2, der $f = 1$ gir det vanlige onlinetilfellet, med $\alpha = k$, mens om vi øker f, vil vi «til slutt» (for $f = \infty$) ende med offlinetilfellet, med $\alpha = 1$. Klokland har også prøvd å tegne opp hvordan hun ser for seg at α kanskje oppfører seg for $1 < f < \infty$. Hva tenker du om saken? Diskuter og forklar.

Vi vil ikke få $\alpha < k$ for noen endelig verdi av f. Man kan tvinge frem atferd som må gi $\alpha \geqslant k$ ved å repetere hver verdi f ganger.

9 Du skal sortere en sekvens $s = \langle s_1, \ldots, s_n \rangle$ vha. lineærprogrammering. Variablene dine utgjør en $n \times n$-matrise X, der x_{ij} skal være 1 i svaret hvis element i i den sorterte sekvensen er s_j, og 0 ellers. Hvordan vil du sette opp et lineærprogram for dette? Forklar.

Du kan kombinere matematisk notasjon og tekstlig forklaring som du vil. Det er ikke nødvendig å sette opp et komplett lineærprogram med matematisk notasjon. Du kan anta at elementene i s er unike. Det er *ikke* nødvendig med heltallsrestriksjoner.

Vi innfører restriksjoner $\sum_{j=1}^{n} x_{ij} = 1$ for hver i og $\sum_{i=1}^{n} x_{ij} = 1$ for hver j, sidne hver verdi bare kan havne på én plass i svaret, og hver plass i svaret bare kan holde én verdi. Vi krever også $x_{ij} \geqslant 0$ for alle i og j. Da vil alle hjørnene i mulighetsområdet tilsvare en binærmatrise med én ener i hver rad og hver kolonne.

I målfunksjonen $cx = \sum_{ij} c_{ij} x_{ij}$, kan vi f.eks. la $c_{ij} = i \cdot s_j$. Det vil da lønne seg å plassere det minste elementet i posisjon 1, etc.

10 Din venn Smartnes ser på fordeling (*scheduling*) av n jobber på m identiske maskiner, der hver jobb j har en prosesseringstid p_j. Han vil fordele dem så jevnt som mulig, men heller enn å minimere siste sluttid, C_{max} (*makespan*), vil han *maksimere første sluttid*, C_{min}.

Han uttrykker problemet som et heltallsprogram, der en binærmatrise avgjør hvilke jobber som gis til i hvilken maskin, og så maksimerer han en variabel som han krever skal være mindre enn eller lik hver av sluttidene.

Han har så en idé til en randomisert approksimasjonsalgoritme. Planen hans er å løse LP-relakseringen (dvs., uten heltallsrestriksjoner). Matrisen får da verdier i intervallet $[0, 1]$, og han runder disse av til 0 eller 1, med en sannsynlighet lik verdien.

Han lurer på om det er noe rart med metoden hans. Etter det han kan se, vil forventningsverdien være lik målverdien fra LP-relakseringen, og denne vil jo være minst like god som, og sannsynligvis bedre enn, optimum for heltalsprogrammet. Stemmer det? Hvorfor eller hvorfor ikke?

Hva er eventuelt galt med metoden hans? Kan det korrigeres?

Selv om vi her maksimerer en sluttid, er målet å få en jevn fordeling, ikke å bruke lang tid; derfor skal det fortsatt ikke være noen pauser mellom jobbene.

Metoden vil feile, fordi jobber kan ende opp med å kjøre på flere maskiner. Det kan korrigeres ved å runde av alle variablene for én gjenstand samtidigm, så bare én av dem blir 1. Siden summen av disse variablene er 1, kan vi fortsatt bruke samme sannsynligheter når vi velger maskin, og vil ende med samme forventningsverdi.

Men dette vil uansett ikke gi en forventet målverdi som er bedre enn optimum (som er umulig). Grunnen til det er at vi maksimerer *minimum*, som ikke er lineært; dvs., forventningsverdien til minimum er ikke det samme som minimum av forventningsverdiene.

For spesielt interesserte: Det finnes faktisk en PTAS for dette problemet, som riktignok er mest av teoretisk interesse. Se "A polynomial-time approximation scheme for maximizing the minimum machine completion time" av Gerhard J. Woeginger (*Operations Research Letters* 20, 1997).