Syddansk Universitet
Institut for Matematik & Datalogi, IMADA
Oktober 2008

DM811 - Heuristikker og lokalsøgningsalgoritmer for kombinatorisk optimering

Eksamensopgave

Underviser:
Marco Chiarandini

Indhold

Kapitel 1

Hypergrafer

Opgaven handler om Hypergrafer. En Hypergraf er en graf, hvor en kant ikke nødvendigvis kun har to knuder, men kan have eksempelvis 20 knuder. Der findes også n-uniforme hypergrafer, en 2-uniform hypergraf er en graf hvor alle kanter har præcis 2 knuder, dvs det kaldes bare en graf. Og en n-uniform hypergraf er en graf hvor alle kanter har præcis n knuder.
Opgaven her går ud på at man skal finde det største stabile set på hypergrafer.
Dette problem går ud på at man skal finde en sådan kombination, af knuder, at man har taget så mange knuder som muligt, og samtidigt opfylde at der skal være mindst en knude tilbage på en kant, som ikke er taget med. Sådan man tager en del-graf ud af hypergrafen, som opfylder dette. Så denne opgave går ud på at man skal have fundet et maximalt stabilt set.

Kapitel 2

Heuristikken

Da jeg skulle finde en konstruktionsheuristik, satte jeg mig ned og tænkte grundigt over problemets udformning. Der er en del forskellige måder at lave en sådan heuristik på. Eksempler på heuristikker man kunne lave, kan være:

1. Man kunne tage den første kant, så kunne man vælge og udvælge alle knuder pånær én knude, og så kunne man gå videre til næste kant, og så gøre det samme, men stadig holde øje med om man overholder kravene i alle kanter.

2. Man kunne bruge samme måde at gøre det på som ovenfor, man kunne i stedet så bare vælge en vilkårlig knude og så gå den vej gennem, til det ikke længere er muligt at udvælge flere knuder.

3. Man kunne også tage og i stedet kigge på knuderne. Man kunne så se på hvilken eller hvilke knuder, der er representeret i flest kanter. Så kunne man så tage og have den knude fri, og så vælge alle andre knuder i de kanter som indgår, dog skal man stadig holde øje med at man ikke violater betingelsen i andre kanter, som denne knude ikke indgår i.

4. En mulighed mere ville være at vælge en knude stokatisk, og så udføre samme operation på denne, som i man udfører i punktet ovenover. Og når man så har gjort det, så kan man vælge en ny knude, og så gøre det sammen med den, hvis den ikke er blevet taget med i en tidligere operation.

5. En mulighed kunne også være at kan bare valgte en tilfældig knude og tog den med hvis den ikke violater med kravene, så tage den med, ellers lade den være.

Der er mange andre måder Konstruktionsheuristikken kunne laves på, dette er kun et lille udsnit af hvordan man kunne gøre.
Den måde jeg har valgt at gøre det på er, den i punkt 4. Jeg har først forsøgt mig med metoden i punkt 3. Denne metode virkede også fint men da der ikke er nogen stokastik i metoden valgte jeg at lave den i punkt 4, og i 3 er man på intet tidspunkt sikker på at man får det rigtige resultat, men man låser sig selv fast i et punkt hvor det ikke er sikkert man kan finde et globalt maksimum, men kun et lokalt. Så derfor valgte jeg i stedet at lave den omtalte konstruktionsheuristik fra punkt 4.
Måden jeg gør det på er at jeg laver 3 "Matrixes", det er ArrayList af ArrayList af Integer som jeg bruger, hver ArrayList af Integer representere en kant, ArrayList'en er af størrels, det totale antal af knuder plus en. På første plads holder jeg styr på hvor mange knuder der ikke er taget endnu minus en. Resten af pladserne tager jeg og sætter et, et-tal på den plads som svarer til at knuden ville være med i kanten. Så laver jeg en matrix mere som

holder styr Hvilke knanter der indeholder hvilke knuder. Og Så laver jeg også en matrix som indeholder information om, hvilke knuder der er i hvilke kanter.
I heuristikken, tager jeg så og laver en liste med en plads der representere hver knude. For at kunne vælge tilfældigt, tager jeg så og vælger stokastiks hvilken af disse jeg vil bruge, når jeg har valgt en knude jeg gerne bruge så holder jeg den knude åben, dvs jeg vælger at at tage alle de andre knuder med som er i de samme kanter som denne. Jeg opdatere så min matrix hvor jeg kan se hvor mange knuder der er ledige, og derved holder styr på jeg ikke violater betingelserne. Når jeg så har gjort det, så vælger jeg en ny knude i min liste, hvis den knude allerede er taget tager jeg en ny, osv. Sådan fortsætter jeg til jeg ikke har flere knuder at vælge ud fra. Grunden til jeg har valgt at gøre det på denne måde er at jeg vil forsøge at bruge så mange knuder som muligt.
Når jeg kører denne heuristik på udleverede data. får jeg for filen `u-1000-10-1000-xx.mss` får jeg hver gang et antal af knuder jeg har vælgt på omkring 800 stk. Hvilket er inden for grænsen i tabellen som er med i opgaven.
Jeg har også haft prøvet implementere algoritmen i punkt 5. Da jeg kørte den heuritstik, fik jeg hver gang jeg kørte den på filerne `u-1000-10-1000-xx.mss`, at den ville medtage ca 450 knuder. Det må så siges at denne kontruktion er meget langt fra gappet i tabelen i opgaven. Så derfor valgte jeg at fjerne denne metode igen.

Kapitel 3

Lokalsøgning

Under lokalsøgning er der flere forskellige metoder at gøre det på, dog kan ikke alle former for lokalsøgning bruges her. Forskellige typer der kan bruges i dette tilfælde er

1. **Swap:** Der tager man 2 knuder ved siden af hinanden og bytte deres værdi rundt hvis det ikke violater vores betingelser.
2. **Interchange:** Der tager man to knuder, som ikke nødvendigvis er ved siden af hinanden og bytter rundt på, hvis det ikke violater.
3. **Blockmoves:** Der tager man tre knuder og bytter rundt på deres placering og ser om det kan lade sig gøre. Og vil give et bedre resultat.

Jeg har valgt at implementere Interchange, da der umiddelbart er størst mulighed for at forbedre i forhold til f.eks metoden Swap. Når jeg vil lave Interchange, så tager jeg og tjekker at de to jeg vil gøre det på ikke begge allerede er med eller ikke er med. For er de ens, så vil det selvfølgelig ikke være interresant at bytte rundt på dem. Hvis de ikke er "ens"så tager jeg den der er valgt, tager den ud af mit valg, og så tjekker om jeg så kan tage den anden af knuderne med i stedet. Hvis jeg kan det. Så kigger jeg på om jeg nu andre steder har mulighed for at tage nogle knuder med. Hvis jeg ikke kan sætte den anden knude uden at violate, så medtager jeg bare den første igen, og gør ikke mere.
Med Swap ville man kun have kigget på naboen, og dette vil indskrænke mine muligheder for at optimere i dette opsæt af opgave. Med Blockmives, vil det heller ikke være bedre end med Interchange, da man i denne opgave kun har to muligheder, med eller ikke med. Blockmoves ville være bedre i eksempelvis et TSP problem, hvor man kan bytte flere placeringer rundt.

Kapitel 4

Algoritmen

I algoritmen har jeg valgt kun at bruge min Heuristik, fra koden kaldet random, da jeg som tidligere nævnt med den Heuristik som jeg har kaldet mostEdgeHeu, ikke ville så noget stokastisk ind i mit resultat.
Men i algoritmen kalder jeg først min Heuristik på mine data. Derefter kalder jeg min Lokalsøgning, hvor jeg har lagt min Interchange ind i en while-løkke som så kører indtil jeg har forsøgt alle interchange uden nogen forbedring, hvilket må som minimum give et lokalt maksimum. Når jeg har gjort dette nulstiller jeg mine data, og starter forfra med min heuristik, indtil tiden er gået.
Hvis jeg på et tidspunkt ikke har fundet et lokalt maksimum på denne måde, inden tiden er udløbet, så vil jeg jeg bare i miin udskriftsfil skrive det ude jeg indtilvidere har fået af resultat, og bruge dette. Så mit program ikke har kørt forgæves. Men jeg har fundet noget som kan bruges.
Komplexiteten af min Heuristik vil være, først hver gang jeg kalder en knude tilføjer jeg antalet af kanter pr. knude, og det gør jeg for alle n-knuder, dvs. at min algoritme køre i mindst $O(n^3)$ hvilket er forholdsvis højt, men da der ikke i alle kald vil blive lave nogle udregninger, så vil dette kunne være reduceret kraftigt. Min lokalsøgning med vil være jeg køre $O(antalletafkanterprknude)$ og det gør jeg $\sum_{i=1}^{n} i$ dvs at den køre så O(n) og dette gøres en konstant ganget på, antal gange. Dvs hele min algoritme køre i $O(n^3)$

Kapitel 5

Resultater

Jeg har kørt min algortime på alle instanser, og har der i alle tilfælde nået indenfor gappet indenfor tiden, og har nået et godt stykke ind. Ved 1000 knuder og 1000 kanter med 50 kanter per knude, vil jeg med en forbedring på ca 3% komme op og ramme den øvre grænse, hvilket jo er en teoretisk grænse, og kan derved ikke med sikkerhed opnås, så den vil være meget tæt på. Kigger man på samme resultat med 10 knuder pr kant, så er jeg ca 5.5% fra at ramme den øvre grænse. Hvilket også må være en okay tilnærmelse.
I instanser med 1000 knuder og 10000 kanter er det svart at sige hvor tæt man kommer på den optimale løsning.

Instans	Bedste fundne løsning	Øvre grænse	Nedre grænse
u-1000-10-1000-01.mss	862	910.49	792
u-1000-10-1000-02.mss	861	910.65	792
u-1000-10-1000-03.mss	862	910.54	791
u-1000-10-1000-04.mss	862	910.34	795
u-1000-10-1000-05.mss	864	910.28	790
u-1000-10-1000-06.mss	862	911.42	789
u-1000-10-1000-07.mss	862	910.30	792
u-1000-10-1000-08.mss	865	911.13	801
u-1000-10-1000-09.mss	860	909.84	793
u-1000-10-1000-10.mss	861	911.18	790
u-1000-50-1000-01.mss	953	981.01	924
u-1000-50-1000-02.mss	953	981.08	926
u-1000-50-1000-03.mss	954	981.01	928
u-1000-50-1000-04.mss	955	981.00	926
u-1000-50-1000-05.mss	953	981.09	927
u-1000-50-1000-06.mss	953	980.99	930
u-1000-50-1000-07.mss	953	980.88	931
u-1000-50-1000-08.mss	954	981.01	930
u-1000-50-1000-09.mss	954	981.01	927
u-1000-50-1000-10.mss	954	981.01	929
u-1000-10-10000-01.mss	710	1000.00	$-\infty$
u-1000-10-10000-02.mss	709	1000.00	$-\infty$
u-1000-10-10000-03.mss	710	1000.00	$-\infty$
u-1000-10-10000-04.mss	705	1000.00	$-\infty$
u-1000-10-10000-05.mss	713	1000.00	$-\infty$
u-1000-10-10000-06.mss	709	1000.00	$-\infty$
u-1000-10-10000-07.mss	706	1000.00	$-\infty$
u-1000-10-10000-08.mss	707	1000.00	$-\infty$
u-1000-10-10000-09.mss	704	1000.00	$-\infty$
u-1000-10-10000-10.mss	707	1000.00	$-\infty$
u-1000-50-10000-01.mss	912	1000.00	$-\infty$
u-1000-50-10000-02.mss	914	1000.00	$-\infty$
u-1000-50-10000-03.mss	916	1000.00	$-\infty$
u-1000-50-10000-04.mss	912	1000.00	$-\infty$
u-1000-50-10000-05.mss	913	1000.00	$-\infty$
u-1000-50-10000-06.mss	914	1000.00	$-\infty$
u-1000-50-10000-07.mss	913	1000.00	$-\infty$
u-1000-50-10000-08.mss	913	1000.00	$-\infty$
u-1000-50-10000-09.mss	914	1000.00	$-\infty$
u-1000-50-10000-10.mss	914	1000.00	$-\infty$

Tabel 5.1: Denne tabel er lavet ved at lade algoritmen køre i 300 sek, med seed valgt til 1. De er udarbejdet på maskinen erda.imada.sdu.dk, under linux.

Kapitel 6

Konklution

Når man kigger på Resultaterne der er kommet ud af algoritmen, må det siges at ligge pænt indenfor de grænser som er. Jeg kan derfor sige at min algoritme med kontruktionsheuristik og lokalsøgning, må fungere fint, og komme med fineresultater som må tilnærme sig forholdsvis hurtigt til Optimale løsning. Og da dette problem ligger i NP-klassen, så må det siges at være okay, at man kan tilnærme sig en løsning på så forholdsvis kort tid, og hvis man er tilfreds med ikke at få en optimal løsning men en løsning tæt på , vil dette være yderst fint.