1. Velkommen til presentasjon av masteroppgaven min
2. Fortelle litt om hvordan oppgaven har blitt til, eller oppstått
3. Goodtech og Mathconsult har utviklet et program PROMAPS, som kalkulerer leveransepåliteligheten i et nettverk ved å kalkulere risiko ved utfall av grener i nettverket
4. Kommer tilbake til dette
5. Dette er formulert som et QP-problem, eller en rekke veldig like QP-problemer
6. Det viser seg at QP-løseren er flaskehals. Og derfor oppdaterer skjermbildet seg hvert 5. minutt
7. Vi skal se på et skjermbilde av PROMAPS her i en artikkel fra Teknisk Ukeblad
8. Tar en titt på QP-problemet
9. Objektfunksjonen representerer leveransekostnader, det her er da pengeenhet per sekund, eller penger per sekund
10. x er en ukjent, og representerer antall Watt vi sender over hver gren
11. Φ representerer strømtap over hver gren i Watt
12. D er kostnader for å sende strøm over hver gren, penger per Joule
13. g er kostnader for å generere strøm i penger per Joule
14. c er leveransepris i penger per Joule, dette er da altså inntekt
15. Vi forkorter dette til en mer gjenkjennelig objektfunksjon
16. Her representerer både H og b kostnader. H er en diagonalmatrise og er positivt semidefinitt, som betyr at vi jobber med et konveks QP-problem
17. Vi definerer et QP-problem min... underlagt $Ax = 0$, og x mellom l u. l og u er nedre og øvre grenkapasitet i Watt
18. Goodtech vil løse QP-problemet vi definerte, bare med utfall i forskjellige grener. Vi modellerer dette ved å sette $l_i = u_i = 0$
19. Vi har et QP problem $\mathcal{Q}$ som definert, uten utfall, som vi kaller en instans
20. Så har vi også subinstanser $\mathcal{Q}_k$ som er en instans med et eller flere utfall
21. Vi vil helst løse så mange subinstanser som mulig, for å få mest mulig nøyaktig analyse
22. For et problem med n grener, er det 2^n subinstanser. Men usannsynlig det er mange utfall
23. Prøver å begrense antall subinstanser ved å sette en realistisk grense for antall utfall
24. Kalkulerer antall utfall ved funksjonen sigma
25. Hver variabel representerer en gren, så vi har en mengde med variabler som representerer utfall, som vi noterer $\mathcal{M}_k$, for modifikator
26. Vi har en en-til-en korrenspondanse mellom kombinasjoner av utfall og deres indekser
27. Vi har en subinstans $\mathcal{Q}_k$ som defineres av $\mathcal{Q}$ og $\mathcal{M}_k$, og vi noterer dens optimale løsning x_k^*
28. Jeg fikk tilsendt tre instanser av Goodtech, small, large og vlarge. Her ser vi verdiene på diagonalen til H, etter kolonne
29. Vi ser at alle verdiene er lavere enn 10^{-1}, og at de fleste verdiene er lavere enn 10^{-2}
30. Videre så ser vi her størrelsen på de tre instansene, og merk at over 50% av diagonalelementene i H er 0 i alle tre instansene
31. Et viktig poeng å dra fram her er at verdiene i det lineære leddet er mye høyere enn verdiene i det kvadratiske leddet, så det gir oss motivasjon til å se på metoder basert på lineær programmering
32. Hva skjer hvis vi bare ignorerer det kvadratiske leddet? Har det stor innflytelse på optimal verdi?
33. Vi noterer en lineær Taylor-utvikling av f i punktet a for T_a, og da er $T_0 = b^T x$

34. Vi definerer et LP program hvor vi minimerer det lineære leddet, underlagt de samme sidekravene som i QP-problemet

35. Kort fortalt her så noterer vi avviket mellom optimal løsning til $\mathcal{L}$ og $\mathcal{Q}$ for Δ

36. Vi ser her et 3D-plot som viser avviket mellom $\mathcal{L}$ og $\mathcal{Q}$ som en funksjon av tettheten i objektfunksjonen

37. Vi ser at tettheten i H har veldig lite innflytelse på avviket

38. Vi ser også at b har mye større innflytelse på avviket, men legg merke til at avviket er aldri større enn 5%, det er faktisk såvidt over 4%

39. På grunn av dette kommer altså successive linear programming inn i bildet. Vi vil oppnå 95% av optimal verdi etter første iterasjon hvis vi begynner i 0

40. Her har vi selge algoritmen, vi gjør først en taylorutvikling i det punktet vi står i, så løser vi $\mathcal{L}$

41. Så gjør vi et LINJESØK mellom punktet vi står i og optimale løsning til $\mathcal{L}$. Vi finner altså optimal målfunksjonsverdi av alle punktene på linja mellom de to endepunktene

42. Så flytter vi oss til det punktet vi fant i linjesøket, og fortsetter algoritmen helt til vi når termineringskravet vårt

43. Et eksempel. Vi minimerer dette problemet. Det optimalet punktet er her $x = 1$ og $y = 1$ hvor optimale objektverdi er -2

44. Vi ser her den lineære objektfunksjonen

45. Her har vi et bilde over det tilatte området. Det i rødt representerer ting som har med LP å gjøre. Vi ser her den lineære objektfunksjonen.

46. Vi gjør et linjesøk her mellom x_0 og $\hat{x}_0$ som vist her

47. Gjør et nytt linjesøk her, og finner da x_2

48. Nytt linjesøk, og havner på $x_3 = (0.96, 1.02)$

49. Her ser vi stien som algoritmen tar fra startpunktet til vi terminerer

50. Etter at vi har løst en instans, så er det veldig sannsynlig at vi skal løse et veldig likt problem like etterpå, med en liten endring i høyresiden av sidekravene

51. Hvis vi gjør det, så er det mulig at vi gjør den primale løsningen ikke-tillatt. Vi vet at den duale likningslisten fortsatt er tillatt etter en endring i verdiene på høyresiden, så vi bruker den duale simplex-metoden til å oppnå x_0

52. Vi ser at hele linjesøket tar plass på denne linja, hvor optimale punkt ligger, så linjesøket finner x^* og terminerer

53. Så slik fungerer metoden basert på SLP. Det neste vi skal se på er ikke en egen optimeringsalgoritme, men heller en måte å redusere antall subinstanser vi trenger å løse med et kall til en QP-løser

54. Så husk at vi har en mengde med grener som faller ut som vi noterer som $\mathcal{M}_k$. Og vi definerer en subinstans $\mathcal{Q}_k$ utifra $\mathcal{Q}$ og $\mathcal{M}_k$

55. Vi noterer også en mengde med variabler som er 0 i den optimale løsningen til $\mathcal{Q}_k$ for $\mathcal{Z}_k$

56. Et viktig poeng å dra fram her er at hvis vi tvinger en variabel som allerede er 0 i en optimal løsning til en instans, så vil ikke den optimale løsningen endre seg

57. I den ene instansen fra Goodtech, så viste det seg at kardinaliteten til $\mathcal{Z}_0 = 1749$. Dvs. at det var 1749 variabler lik null i den optimale løsningen. Det betyr at 2^{1749} subinstanser har samme løsning

58. Vi lager en tre-struktur over subinstanser som vi kan søke etter løsninger i

59. Jeg tenkte jeg skulle prøve å forklare trestrukturen ved å vise hvordan man konstruere treet jeg har her, men for å spare tid så konstruerer vi det kun for to variabler. Vi bruker de samme mengdene som her, så har vi en fasit å sjekke opp mot

60. Hver node tilsvarer en subinstans, så vi har en modifikator, en optimal løsning, og en mengde over nuller i løsningen

61. Vi begynner først med å generere alle mulige kombinasjoner av utfall

62. Vi begynner med rot-noden, som tilsvarer instansen uten utfall

63. Og for hver node i treet, så vil hver modifikator til en barnnode være en utvidelse av modifikatoren til noden

64. Jeg går rett til eksperimentene jeg har utført.

65. Her ser vi fire forskjellige implementasjoner. Slp her er algoritmen som jeg viste fram tidligere. Clp er en åpen og gratis LP og QP-løser ledet av en ansatt hos IBM.

66. c-en her står for construct, som betyr at vi løser subinstansene ved å konstruere et tre slik som jeg gjorde.

67. n-en her står for naive, som betyr at den naivt løser alle kombinasjonene av subinstanser.

68. Tabellen her viser resultater av de fire implementasjonene når de løser *small* og 28442 av dens subinstanser

69. Det er dessverre veldig tydelig at Slp ikke greier å henge med.

70. Her ser vi et loglog-plot av verdiene i tabellen.

71. I det neste eksperimentet prøver vi implementasjonene på de tre instansene vi fikk tildelt. Vi ser her også dessverre at Slp ikke henger med.

72. På de neste to eksperimentene så genererer vi tilfeldige instanser med lik karakteristikk som instansene fra Goodtech.

73. Vi tester her kun Clp ettersom Slp ikke greide å henge med. Vi ser her at tre-strukturen reduserer kjøretiden med opp til 23% som n øker.

74. På det siste eksperimentet så genererer vi tilfeldige instanser med 50 grener, og øker antall utfall.

75. Ettersom β øker, så ser vi at vi øker hastigheten med opp mot 70%

76. Her ser vi antall CPU-sekunder brukt i gjennomsnitt til å løse en subinstans. Vi ser at ettersom β øker, så beveger vi oss ned mot en fjerdedel av tiden per subinstans