Lenkeanalysemetoder og Rangering av Dokumenter i Domener med få Lenker

Sammendrag

For å rangere dokumenter ved søking har det blitt investert store ressurser i å finne metoder som er effektive og gir gode resultater. Denne jobben blir mer komplisert og krevende i tråd med størrelsen på dokumentsamlingen man analyserer. Internett er en uoversiktlig samling dokumenter som vokser seg større for hver dag. I det omfattende forsøket på å holde oversikt over denne samlingen har spesielt en metode som baserer seg på analyse av lenker mellom dokumenter vist seg å være et nyttig hjelpemiddel. Dette er mulig fordi størsteparten av dokumentene på Internett inneholder lenker, eller anbefalinger, til andre dokumenter. Det er interessant å se om det er mulig å ta i bruk disse vel etablerte og godt fungerende metodene for analyse av lenker som hjelpemiddel til rangering av dokumenter som ikke inneholder lenker.

Denne oppgaven tar for seg mulige løsninger for hvordan man kan rangere dokumenter funnet i domener uten lenker, uten bruk av tekstrelevansanalyse. Mange forskjellige metoder blir foreslått. Felles for disse er at de bruker lenkeanalyseringsmetoder til å analysere simulerte lenker i domenet. For å simulere lenker i domenet regnes det ut likheter mellom alle par av dokumenter. Lenkeanalyseringsmetoden T-Rank brukes så til å beregne en viktighetsscore for hvert av dokumentene, som om domenet inneholdt lenker, basert på disse autogenererte ”likhets-lenkene”.

Målet med oppgaven er å finne ut hvor godt likhetsberegninger mellom dokumenter kan fungere som en substitusjon for tradisjonelle lenker i domener uten egen lenkestruktur.

Til å teste de forskjellige rangeringsmetodene brukes Googles rangering, for like søk i samme domene, som fasit. Resultatene av testingen viser at likhetsberegninger mellom dokumenter kan brukes som et nyttig hjelpemiddel til å rangere dokumenter i domener med få lenker hvor tradisjonell lenkeanalyse kommer til kort.

Forord

Jeg ønsker å takke mine veiledere på Telenor Fornebu, Kenth Engø-Monsen og Geoffrey Canright for deres verdifulle hjelp og veiledning gjennom dette prosjektet. Det ville ikke vært mulig for meg å gjennomføre denne oppgaven uten deres hjelp. En spesiell takk må også gå til Josip Zoric, min kontakt på Telenor i Trondheim, og Roger Midtstraum som har vært min veileder på NTNU.

Grunnen til at denne oppgaven er konfidensiell er fordi den inneholder bruk og beskrivelse av funksjonaliteten til en patentsøkt lenkeanalyseringsmetode utviklet på Telenor R&D.

1 Innledning

I denne oppgaven har vi utviklet en søkemotor som vi har valgt å kalle for B3000. Hver gang navnet ”B3000” nevnes i oppgaven refereres det til resultatet av det ferdige systemet (søkemotoren) utviklet. Flere av modelleringsfigurene i oppgaven inneholder engelske navn, funksjoner og beskrivelser. Vi har valgt å bruke engelske kommentarer i kildekoden, og det lot seg best gjøre å bruke engelske ord i utviklingsverktøyet ”Enterprise Architect”. Vi så det derfor som naturlig å også skrive kommentarene i modelleringsfigurene på engelsk. Hovedsakelig gjelder dette figurer i kapittel 7 – ”*Utvikling”*. I tillegg er det brukt engelske uttrykk enkelte steder i oppgaven der det virket naturlig. De figurer som ikke er en direkte modellering av systemet inneholder norske ord og uttrykk.

Dette kapittelet starter med å introdusere leseren for noen forskjellige metoder som er vanlig å bruke for rangering av dokumenter på Internett. Det diskuteres så litt generelt rundt de svake og sterke sidene ved metoder som er relevant for denne oppgaven. I slutten av introduksjonen nevnes hvordan vi har brukt kombinasjoner av forskjellige metoder til å utvikle B3000. Leseren blir i løpet av innledning innført i problemstillingen og målene i oppgaven, før vi gir en overordnet oversikt over komposisjonen til B3000. Til slutt i innledningen nevnes noen krav til oppgaven og resten av oppgavens oppbygning.

1.1 Introduksjon

Metoder som ofte brukes for effektivt å finne relevante dokumenter ved søking er lenkeanalyse[1] og tekstrelevansanalyse[2]. Det finnes mange forskjellige måter å foreta tekstrelevansanalyse på (Baeza-Yates and Ribeiro-Neto 1999), det finnes også flere måter man kan foreta lenkeanalyse (Gevrey and Ruger 2002). Tekstrelevansanalysen plukker ut de dokumentene i en samling som er mest relevante i forhold til dataen man supplerer søkemotoren med (søkeordene). Resultatet er ofte så stort at tekstrelevansanalyse alene ikke er nok til å presisere hvilke dokumenter som skal vises til brukeren. Derfor brukes lenkeanalyse i tillegg for å plukke ut de beste dokumentene blant mange relevante dokumenter. Men er det mulig å bruke disse vel etablerte og godt fungerende lenkeanalysemetodene som hjelpemiddel om dokumentsamlingen ikke inneholder lenker?

[1] Med ”lenkeanalyse” menes å, basert på analyser av lenker funnet i domenet, angi en verdi til hvert dokument som mål på dokumentets viktighet. Dette kan ofte gjøres offline, uavhengig av brukeren som søker.
[2] Med “tekstrelevansanalyse” menes å analysere hvert dokuments relevans til søkeordene en bruker gir. Dette gjøres i sanntid ved søk.

Det finnes i dag forskjellige metoder for å rangere resultater ved søking i store datamengder. Noen av de mest brukte metodene er: betalt plassering, manuell rangering, tekstrelevansanalyse, rangering basert på lenkeanalyse eller kombinasjoner av disse.

Betalt plassering er en relativt enkel rangeringsmetode, men det er ikke en bra metode for å finne den best mulig tilgjengelige informasjonen.

Manuell rangering har den svakheten at den er omfattende, dyr og for treg for store datamengder som for eksempel Internett.

Tekstrelevansanalyse er meget viktig for å finne relevant informasjon. Det er flere påstander om at metoder som utelukkende bruker tekstrelevansanalyse i seg selv kan være bra nok for bruk i store datamengder (Maton 1959; Baeza-Yates and Ribeiro-Neto 1999), men bra nok operatorer og metoder for å klare dette er vanskelig å utarbeide (Langville and Meyer 2004). De fleste kommersielle søkemotorer i dag bruker tekstrelevans teknikker sammen med andre teknikker, som for eksempel lenkeanalyse for å foreta rangeringen. Google er et eksempel på dette.

Tradisjonelle lenkeanalyseringsmetoder har vist seg å fungere bra i dokumentsamlinger inneholdende tradisjonelle lenker (Langville and Meyer 2004). Det hadde derfor vært fint om man kunne overføre metoder som brukes for lenkeanalyse (f.eks. PageRank[3] eller T-Rank[4]) på en slik måte at de fungerer i dokumentsamlinger som mangler en bra tradisjonell lenkestruktur. Men tradisjonell lenkeanalyse har den åpenbare svakheten at den ikke fungerer i domener uten en egen lenkestruktur. I denne oppgaven skal vi se nærmere på hvordan man kan lage operatorer som bruker lenkeanalyseringsmetoder til å rangere en samling med dokumenter som har en dårlig iboende lenkestruktur, eller ikke inneholder tradisjonelle lenker i det hele tatt.

[3] PageRank er lenkeanalysemetoden oppfunnet av Larry Page og Sergei Brin. Den brukes av søkemotoren Google som har vist seg å ha stor sukksess.
[4] T-Rank er en patentsøkt lenkeanalysemetode oppfunnet på Telenor R&D som har flere likheter med PageRank.

Forståelsen av pekerstruktur på Internett og lenkestruktur generelt i denne oppgaven er todelt:

1. En peker sier noe om at det den peker på, direkte eller indirekte og til en hvis grad, har en likhet eller noe til felles med den konteksten som er der pekeren befinner seg.

2. Det er en anbefaling fra dokumentet pekeren befinner seg i til dokumentet pekeren peker til.

Det har seg slik, at likhet mellom dokumenter kan beregnes automatisk av en maskin på forskjellige måter. På denne måten er ”1” oppfylt, og kan løses ved hjelp av nærmere forskning. Men vi vil påstå at det er lite sannsynlig at man noen gang vil kunne oppnå ”2” automatisk. En maskin kan ikke anbefale dokumenter på lik linje med oss mennesker. Det er mennesker som kommer med anbefalinger. Disse anbefalingene er direkte eller indirekte basert på analyser foretatt i menneskehjernen som er så avanserte, og inkluderer så mange uvisse faktorer, at en maskin vil ha veldig vanskelig for å gjøre det. Derfor er pekere en veldig god ressurs å bruke for å analysere seg frem til hvor viktige dokumenter er, og hvordan de skiller seg fra hverandre, i domener som inneholder mange pekere.

En ulempe ved utelukkende å bruke pekere for å analysere domenet er at man begrenser seg til de domener som inneholder (mange) pekere. Og resultatene av analysen blir dårligere i tråd med hvor svak lenkestrukturen i domenet er. I denne oppgaven skal vi se nærmere på hvordan man kan optimalisere viktighetsanalysen for dokumenter i domener med en svak lenkestruktur. Dette skal gjøres ved å se på (den svake) lenkestrukturen og annen tilgjengelig informasjon. Vi skal også komme med forslag til hvordan en maskin kan finne indirekte / gjemte lenker i domener.

Hovedkjernen i denne oppgaven var å prøve å forbedre rangeringsmulighetene ved søk i dokumenter som befinner seg i domener med en dårlig lenkestruktur. Dette har blitt gjort ved å bruke ideen om at likhet mellom dokumentpar innenfor et visst tema kan brukes som hjelpemiddel til å rangere dokumentene, eller gi dem en score. I tillegg har vi kommet med forslag til hvordan man kan lete etter og finne lenker i domener uten, eller med dårlig egen lenkestruktur. Disse lenkene vil høyst sannsynlig være spredt og få, og vil ikke alene fungere godt som basis for viktighetsberegning av dokumenter. Vi har også forsket på og testet ut forskjellige kombinasjoner av likhetsberegning og lenkeanalyse. Til slutt konkluderer oppgaven med forslag til metoder som sannsynligvis egner seg godt for rangering av dokumenter i domener med få eller ingen lenker.

1.2 Problemstilling

Problemstillingen i denne oppgaven er:

Hvordan kan man automatisk skille viktige dokumenter fra mindre viktige dokumenter basert på lenkeanalysemetoder og uten bruk av tekstrelevansanalyse, i domener med svak lenkestruktur?

Vi skal se nærmere på hvordan man kan beregne viktighet mellom dokumenter som befinner seg i en samling med en svak lenkestruktur, eller uten en egen konkret lenkestruktur. Spesielt skal vi se på hva slags metoder som kan taes i bruk *sammen* med vanlig lenkeanalyse i dokumentsamlinger med få lenker og hvordan disse kan kombineres med vanlig lenkeanalyse.

Målet er å finne en eller flere rangeringsmetoder som gir gode og logiske resultater uten bruk av tekstrelevansanalyse. I metodene som utarbeides i rapporten vil det ikke tas i betraktning tekstrelevans fordi fokus skal være på kombinasjoner av lenkeanalyse og likhetsanalyse av dokumenter. Å ta i betraktning et dokuments relevans til en søkefrase kan være vanskelig, og vil virke forstyrrende i denne sammenheng. Måten resultatene skal måles på er å sammenlikne med de resultatene søkemotoren Google gir for like søk i samme samling. Delmål i oppgaven er å bygge en søkbar invertert indeks over en dokumentsamling, plukke ut lenker i samlingen og analysere disse med hhv. PageRank og T-Rank, samt finne og utvikle en metode for å beregne likhet mellom dokumentpar. T-Rank er en lenkeanalyseringsmetode utviklet av Telenor R&D, som ikke har blitt testet på en reell dokumentsamling før. Å teste T-Rank på en reell dokumentsamling er derfor også en av delmålene i oppgaven.

Hovedkjernen i oppgaven blir så å teste ut og finne forskjellige måter å kombinere enten likhetsmatrisa[5] eller resultatene fra analyser av den med lenkestrukturmatrisa[6] eller resultatene fra analyser av den. For å simulere en dokumentsamling med dårlig egen lenkestruktur vil vi tilfeldig fjerne størsteparten av lenkene som allerede finnes i samlingen[7]. Hovedkjernen i oppgaven baserer seg på ideen om at en normalisert ordfrekvensanalyse[8] av hvert dokument i en samling kan brukes til å lage en virtuell relasjonsstruktur[9] mellom dokumentene. Denne virtuelle relasjonsstrukturen kan så

[5] En likhetsmatrise er en komplett oversikt over målet på likhet mellom hvert par dokumenter i en samling. Gjennom oppgaven kalles også en likhetsmatrise for en ”similaritetsmatrise”.
[6] En lenkestrukturmatrise er en komplett oversikt over alle lenkene mellom dokumentene i en samling.
[7] Se kapittel 1 for informasjon angående dokumentsamlingen som brukes i denne oppgaven
[8] Se kapittel 4 for informasjon om hva ordfrekvensanalyse er og hvordan dette beregnes.
[9] En virtuell relasjonsstruktur er en måte å bruke likhetsamtrisa som en lenkestrukturmatrise.

brukes, enten alene eller som supplerende informasjon til tradisjonelle lenkeanalyseringsmetoder, til å beregne dokumentenes viktighet i forhold til hverandre. Videre skal vi finne teknikker for å spleise resultater fra analyse av lenkestrukturen i dokumentsamlingen med analyse av likheten mellom dokumentpar i samlingen. Likhet mellom dokumentpar i samlingen er derfor en viktig del av oppgaven. Det er ikke trivielt å regne likhet mellom dokumentpar i en stor samling dokumenter, og et av delmålene er derfor å se nærmere på metoder for å finne ut hvilke dokumentpar i en samling som har en viss likhet. Resultatene av likheter mellom dokumentpar lagres i en likhetsmatrise. Likheten mellom dokumentpar vil bli representert med en verdi mellom 0 og 1 hvor 0 er helt ulik, og 1 er helt lik. Hvis vi oppnår en komplett oversikt over likhetene mellom alle dokumentpar i samlingen kan vi bruke T-Rank til å analysere den urettede likhetsmatrisa som om den var en rettet matrise over lenkene i domenet. Hvis vi i tillegg kombinerer / supplerer denne likhetsmatrisa med de få rettede lenker det er mulig å finne i domenet vil vi oppnå en svakt rettet graf, med mye tilleggsinformasjon (likhetsverdiene), som vil passe godt for T-Rank. På denne måten har vi brukt ideen om at likhet mellom mange dokumenter i en samling kan brukes til å oppnå en ”intern formening” om relasjoner mellom dokumentene, og basert på dette gitt hvert dokument en poengverdi som mål på viktighet. Denne viktigheten er et resultat av allerede eksisterende data funnet i domenet, selv om der ikke finnes lenker mellom dokumentene, og den kan videre brukes som hjelpemiddel for å filtrere bort eller nedprioritere støy ved søking.

Telenor R&D har en formening om at enkelte dokumentsamlinger uten egne tradisjonelle lenker kan, basert på allerede eksisterende data og meta-data, analyseres (ikke basert på likheter mellom dokumenter!) slik at man finner ”utradisjonelle” lenker. Lenker funnet på denne måten antas å være (veldig) svake, og vil derfor tilsvare et tilfelle hvor man har et domene med en begrenset iboende lenkestruktur. Vi vil komme med eksempler på hvordan vi ser for oss slike tilfeller senere i oppgaven. Slike tilfeller er vel egnet for de metodene som utvikles og testes, og det er slike tilfeller vi forsøker å simulere ved å tilfeldig fjerne lenker i samlingen.

Inspirasjonen til dette arbeidet er interessen for automatisk å finne data som kan brukes til å rangere dokumenter i domener uten egen linkstruktur. Et dårlig alternativ er å utvikle systemer som lar brukeren angi viktighet, eller legge inn lenker selv. Brukeren har en iboende latskap og er en større kilde for å oppnå feil eller dårlige resultater enn en automatisk metode. Det bør gjøres automatisk det som lar seg automatisere. T-Rank er en metode som ingen andre bruker eller har brukt på en stor, reell dokumentsamling før. Det er med på å gjøre oppgaven spennende og nyskapende. Å foreta lenkeanalyse på en matrise laget ved likhetskalkulering av alle dokumentene i en samling har vi ikke funnet noe informasjon på at andre har gjort. Å bruke ideen om dokumentlikheter sammen med

T-Rank, og et begrenset antall vanlige lenker, virker derfor som noe nyskapende og er hovedinspirasjonen bak dette prosjektet.

Målet og delmålene i denne oppgaven kan kort oppsummeres slik:

1.2.1 Hovedmål

- **Finne ut hvor godt likhetsberegninger mellom dokumenter kan fungere som en substitusjon for tradisjonelle lenker i domener uten egen lenkestruktur.**

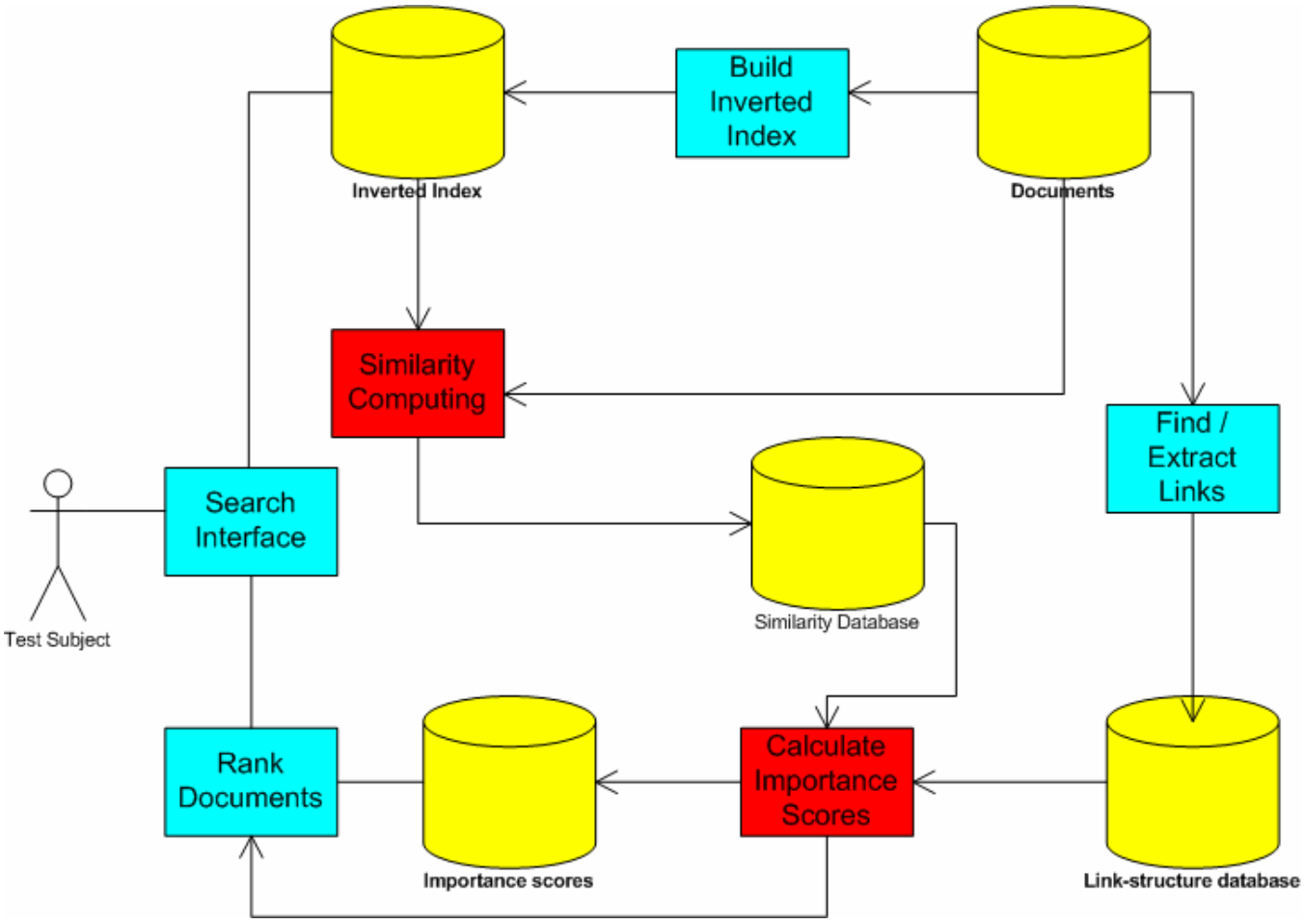
1.2.2 Delmål

- Finne og utvikle metoder for å beregne likheter mellom dokumentpar.
- Finne og implementere en metode som er i stand til å beregne likheter mellom alle dokumentpar i en samling.
- Finne og implementere metoder for å kombinere analyser av likheter mellom dokumenter med lenkeanalyse.
- Bygge en søkbar invertert indeks.
- Automatisk finne og plukke ut lenker i en dokumentsamling.
- Analysere lenker funnet i dokumentsamlingen med T-Rank og PageRank.

1.3 Komponenter utviklet i dette prosjektet

T-Rank algoritmen og teorien bak den er utviklet av Geoffrey Canright og Kenth Engø-Monsen ved Telenor R&D (Canright and Engø-Monsen 2005). Denne teorien er implementert i programmeringsspråket C i en annen masteroppgave hos Telenor som har pågått parallelt med dette prosjektet. Den samme komponenten er også i stand til å regne PageRank.

1.3.1 Oversikt over komponenter og deres relasjoner



Figur 1 - Oversiktsdiagram over alle komponenter og deres relasjoner til hverandre. Dette er en videreutvikling fra arkitekturen som ble brukt i høstprosjektet jeg deltok i på Telenor i trondheim.

De to viktigste komponentene er de røde ”Similarity Computing” og ”Calculate Importance Scores”. Similarity Computing består av en algoritme for å beregne likhet mellom alle de dokumentpar som er forventet å ha en viss likhet i en samling. ”Calculate Importance Scores” er spesiell på den måten at den kombinerer resultater fra ”Similarity Computing” med analyse av lenkestrukturdatabasen på forskjellige måter for å oppnå best mulig rangering. Alle de andre komponentene er nødvendige og ligger til grunn for at dette skal være mulig. De er også en del av dette prosjektet ved at en del av oppgaven er å teste T-Rank på en reell dokumentsamling. ”Search Interface” muliggjør testing av metodene i sanntid. Komponentene ”Calculate Importance Scores” og ”Rank Documents” har en relasjon mellom hverandre fordi det i enkelte av rangeringsmetodene må brukes lenkeanalyse i sanntid.

1.4 Krav

- **Kunne sammenligne rangeringsresultatene med de rangeringsresultater Google gir for like søkefraser i samme domene.** Resultatene skal være sammenlignbare med tilsvarende resultater fra andre kjente rangeringsmetoder tatt i bruk på samme domene, eller i samme problemsetting. En tilsvarende metode er for eksempel Google, og resultater den produserer.

- Dokumentsamlingen som skal brukes for analyse av metodene utviklet i prosjektet skal ikke inneholde spam. Metodene skal ikke lages med tanke på å motvirke spam, men det bør forklares hvorfor en evt. metode vil være resistent, eller ikke, mot spam.

- Oppgaven utarbeides parallelt med andre prosjekter i Telenor, og deler av resultatene fra denne oppgaven skal kunne brukes i samsvar med disse. De forskjellige komponentene utviklet i denne oppgaven skal kunne samarbeide med andre komponenter eller resultater som andre komponenter gir. Dette kravet imøtekommes ved å bruke et felles grensesnitt hvor alle mellomlagringer og resultater holdes. Dette felles grensesnittet er MySQL.

1.5 Oppgavens oppbygning

I starten av oppgaven, kapittel 2, nevnes noen sentrale og generelle lenkeanalysemetoder som er vel etablerte og kjente i dagens ”søkeverden”.

Videre, i kapittel 3, blir lenkeanalysemetoden T-Rank forklart, og leseren blir innført i teorien bak denne metoden.

Siden hovedkjernen i oppgavene baserer seg på å beregne likheter mellom dokumenter har vi så i kapittel 4, valgt å illustrere og diskuterer rundt hvordan dette kan gjøres. I dette kapittelet nevens forskjellige metoder for hvordan likhet kan beregnes og det diskuteres rund ytelsen til de forskjellige metodene foreslått.

Hovedkjernen i oppgaven, som er å kombinere likhetsanalyse med lenkeanalyse, finnes i kapittel 5. Dette kapittelet tar for seg i detalj flere forskjellige måter likhetsanalyse og lenkeanalyse kan kombineres med metoden T-Rank, og brukes til å gi alle dokumenter en avgjørende verdi som mål på viktighet.

Som en forberedelse til testingen av metodene som foreslås i kapittel 5 argumenteres det i kapittel 6 for hvorfor vi har valgt dokumentsettet som brukes under utvikling (kapittel 7) og i testfasen (kapittel 8).

I kapittel 7 har vi modellert de nødvendige komponentene som er utviklet underveis. Dette kapittelet tar for seg design og implementeringen av systemet som er laget. Vi forklarer hver komponent og deres funksjon som enkeltstående komponent så vel som en samarbeidende helhet. Utviklingskapittelet består av to faser.

I kapittel 8 foretar vi en test av systemet og evaluerer resultatet ved å sammenligne med den rangeringen Google gjør av de samme dokumentene for like søk.

Til slutt, i kapittel 9, kommer konklusjonen basert på resultatene fra testene.

Det finnes også et appendiks hvor leseren vil finne nyttig informasjon referert fra forskjellige kapitler i oppgaven. Dette appendikset består av:

10.1 – Informasjon om hvordan leseren kan teste resultatet av prosjektet på Internett.
10.2 – Noen tilleggstanker rund resultatet av testene.
10.3 – En mattematisk utredning av formelen brukt for å måle resultatet av testene.
10.4 – Noen resultater av beregninger på matrisa over dokumentlikheter.
10.5 – Informasjon om materialet vedlagt på CD, med beskrivelse av hvilke komponenter de forskjellige filnavnene representerer.
10.6- Forklaring av en vanlig måte å foreta evaluering av søketester.
10.7– Definisjoner av hva vi mener med enkelte ord og uttrykk som brukes i rapporten.
10.8 – En referanselist.

2 Sentrale deler av dagens rangeringsteknologi

I dette kapittelet nevner vi kort teorien bak noen sentrale metoder som kan brukes som hjelp til å rangere dokumenter. Generelt er flere av disse metodene en viktig del av evolusjonen bak dagens kommersielle søkemotorer (Langville and Meyer 2004).

Å søke i en informasjonsdatabase vil ofte resultere i mange *treff*. Dette er tilfellet hvis man søker ved bruk av nøkkelord, og ikke leter etter et spesielt dokument, men generell informasjon relatert til nøkkelordene. Antallet treff er ofte så stort at det er uoversiktlig og umulig for brukeren å se gjennom alle resultatene. Derfor er rangering av dokumentene funnet viktig. Foruten en viss rangering eller pekepinn på hvilke treff som er verdifulle kan viktige treff forsvinne i mengden av uviktige treff.

Det finnes to hovedmetoder, som ofte kombineres, for å rangere dokumenttreff fra et søk. Den ene metoden er å analysere nøkkelordenes relevans til dokumentene i samlingen. Den andre er å foreta analyse av lenkene i samlingen.

2.1 Søkeords relevans til dokumenter

Det finnes flere påstander om at avansert relevans matching er nok til å rangere dokumenter til å ikke miste de viktigste i mengden (Baeza-Yates and Ribeiro-Neto 1999). Problemet med ren relevans matching er at forfatteren av et dokument kan manipulere søkemotoren ved å legge inn unaturlig tekst som er antatt relevant for å score høyt blant treffresultatene. Det finnes også påstander om at dette ikke er løsningen (Hawking and Craswell 2004), og at man trenger hjelp fra andre metoder for å være i stand til å skille ut de mindre viktige treffene. En kombinasjon av metodene kan også være en god løsning. Det er ikke tekstanalyse og tekstrelevans som er fokus i denne oppgaven, og følgelig vil det blir brukt relativt enkle tekstanalyseringemetoder. Det er allikevel viktig å være klar over at Google, som er sammenligningskriteriet, bruker tekstrelevans i tillegg til lenkeanalyse av dokumentsamlingen vi skal se på.

Det er mulig at likhetsanalyse mellom dokumenter kan fungere som en slags tekstrelevans, spesielt om man ser på en subgraf generert rundt et visst tema. Hvis man for eksempel plukker ut alle dokumenter som inneholder ordet ”appelsin”, vil man ved hjelp av likhetsanalyse blant disse, (i mange tilfeller) klare å skille de dokumentene som handler mest om ”appelsin” fra de som handler minst om det. Denne ideen baseres på at dokumentene som handler mest om det aktuelle ordet (appelsin) vil ha en høyere likhetsverdi med de andre dokumentene som også handler mye om ”appelsin” enn de

dokumentene som ikke handler så mye om det. De dokumentene som ikke handler så mye om ”appelsin” vil sannsynligvis handle mer om noe annet. Det er lite sannsynlig at det vil dreie seg om noe felles annet, og de vil derfor få en lavere score ved at de skiller seg ut fra hovedtemaet.

2.2 Lenkeanalyse

Lenkeanalyse baserer seg ikke på tekstrelevans, men underliggende informasjon som for eksempel lenker, som ofte er å finne i en stor dokumentsamling. Jo større dokumentsamlingen er, jo mer data vil en slik metode ha tilgjengelig for sine kalkulasjoner, og jo mer nøyaktig vil den være i stand til å rangere treff etter viktighet. Ofte er den underliggende informasjonen lenker dokumentene imellom. Disse lenkene kan sees på som pekere som peker i ĕn retning. Til sammen fungerer alle pekerne i en dokumentsamling som en rettet graf. Dagens mest brukte rangeringsteknologi bruker metoder for å gi hvert dokument i en dokumentsamling (graf) en vekt basert på strukturen til pekerne mellom dokumentene.

2.2.1 Lenke Popularitet

Det er forskjellige måter å tolke lenker mellom dokumenter på. En måte er å tilegne hvert dokument en poengverdi basert kun på antallet lenker som peker mot dokumentet. Denne metoden er ofte kalt ”lenke popularitet”, og er relativt enkel å utnytte for en person som ønsker å oppnå en høy poengverdi for dokumentet sitt. Dette kan gjøres ved å lage mange dokumenter som peker til det ene dokumentet man ønsker å heve poengverdien til. Men denne metoden er allikevel brukt av mange kommersielle søkemotorer, antagelig på grunn av sin enkle virkemåte (Baeza-Yates and Ribeiro-Neto 1999; Langville and Meyer 2004).

2.2.2 Random surfer teorien

En annen metode, som er mer resistent mot kunstig genererte lenker er den vel kjente PageRank. Den baserer seg på en teori om en ”tilfeldig surfer” (Brin and Page 1998). Denne teorien går ut på å måle tiden en tilfeldig surfer i gjennomsnitt er innom hver side i hele dokumentsamlingen. Med tilfeldig surfing menes at surferen velger et helt tilfeldig dokument i den totale grafen (dokumentsamlingen), ser på dette dokumentet og velger et helt tilfeldig annet dokument som dette dokumentet peker til. Metoden er meget resistent mot store antall kunstig genererte dokumenter med pekere mot andre dokumenter med den hensikt å øke poengvekten til dokumentene det pekes mot. Grunnen til det er at

”random surfer” teorien innebærer at vekten til et dokument som peker mot et annet har stor betydning for hvor ofte den tilfeldige surferen vil være innom dokumentet det pekes mot. På denne måten blir vekt propagert gjennom grafen. Det har ingen hensikt å lage et stort antall kunstige dokumenter fordi den tilfeldige surferen vil omtrent aldri være innom disse dokumentene, og følgelig vil surferen heller ikke være mer frekvent innom dokumentet som forsøker å øke sin vekt enn surferen ellers ville. Dette er fordi de kunstige dokumentene vil ha meget lav eller ingen vekt. På denne måten er tilfeldig surfer teorien mer sensitiv til hele grafens topologiske struktur.

Et problem med tilfeldig surfer teorien er ”sinks”. Dette er situasjonen hvor den tilfeldige surferen enten havner på en side uten noen utlenker, eller havner i en gruppe dokumenter uten noen måte å komme ut av denne gruppen (subgrafen). Et dokument, eller gruppe av dokumenter som har en måte å komme seg til, men ikke fra, kalles for en sink[10]. PageRank løser dette problemet ved å introdusere tilfeldige hopp, uavhengig av lenkestrukturen, i surfingen. Surferen kan på denne måten havne på en hvilken som helst ny side i hele grafen. Slike hopp inntreffer med en hvis sannsynlighet for hver side surferen er innom. Det er forskjellige måter å løse problemet med sinks på. Sinks er et fenomen i enhver rettet graf, og alle metoder som har med rettede grafer å gjøre må løse dette problemet på en eller annen måte.

2.2.3 HITS

En annen metode er oppfunnet av Jon Kleinberg fra Cornell Universitetet i USA, basert på arbeid gjort med IBM’s CLEVER prosjekt (Gevrey and Ruger 2002). Denne algoritmen er ofte kalt HITS (”Hypertext Induced Topic Selection”). Den er enklest forklart ved å definere to enkle operatorer: **F** (forover) og **B** (bakover). I lys av tilfeldig surfer teorien forestiller han seg en vekt med positiv verdi assosiert med hver node i en rettet graf. **F** operatoren tar vekten *w(i)* ved hver node *i* og sender den forover til alle noder som pekes til av dokument *i*. **B** operatoren tar vekten *w(i)* i motsatt retning, med andre ord mot hver node som peker mot node *i*.[11]

HITS algoritmen bruker komposisjonene **BF** og **FB** gjentatte ganger for å oppnå to forskjellige vekter for hver node. Etter mange repetisjoner av operatoren **FB** vil vekten til hver node i grafen konvergere mot en stabil verdi, som kalles ”Autoritetsverdi” (FB =

[10] Se kapittel 5.1 for en illustrasjon av dette fenomenet.
[11] Å kjøre F metoden på en rettet graf og normaliserer resultatene er det samme som den berømte søkemotoren Google’s PageRank (omtalt som ”tilfeldig surfer” metoden i avsnitt 2.2.2). Med Google’s suksess som referanse vil vi påstå at det er tydelig at denne metoden fungerer bra for en kommersiell søkemotor og gir meningsfulle resultater. Å kjøre B metoden på en rettet graf og normalisere resultatene resulterer i en annen viktighetsverdi som er mer omtalt i Ding et al (LNBL Tech Report 49372).

Autoritet). Likeledes vil repetert bruk av **BF** operatoren resultere i et annet verdimål for hver node i en graf kalt ”Hubverdi” (BF = Hub). Hubverdi og Autoritetsverdi henger sammen på den måten at noder med høy Autoritetsverdi pekes på av gode Hubnoder. Med andre ord har en node bra hubverdi hvis den peker til (anbefaler) en eller flere gode autoritetsnoder. Dette kan forklares på den måten at hubnoder peker til noder med relevant innhold. En node har høy autoritetsverdi hvis den blir pekt på av en eller flere gode hubnoder. På denne måten er hub og autoritet ”mutually defining”[12]. HITS metoden har ikke samme problemet som ”tilfeldig surfer” metoden når det kommer til sinks, fordi man veksler med å følge retningene i den rettede grafen. Man følger retningen til ”pilene” bakover og forover ved annenhver iterasjon.

En viktig egenskap ved HITS metoden er at operatorene **F** og **B** ikke er normaliserte. Dette medfører at vekten et dokument genererer ut fra seg multipliseres med antallet dokumenter det aktuelle dokumentet peker mot. Dette er i stor kontrast til en normalisert metode hvor den totale vekten ut er en fordeling av det aktuelle dokumentets vekt over alle de dokumentene som det pekes mot. Denne egenskapen kan ved første øyekast virke liten og uviktige, men den har veldig store konsekvenser. Hvis man bruker normaliserte FB og BF operatorer blir resultatet det samme som ”lenke popularitets” metoden beskrevet i 2.2.1 (Lempel and Moran 2000; Langville and Meyer 2004).

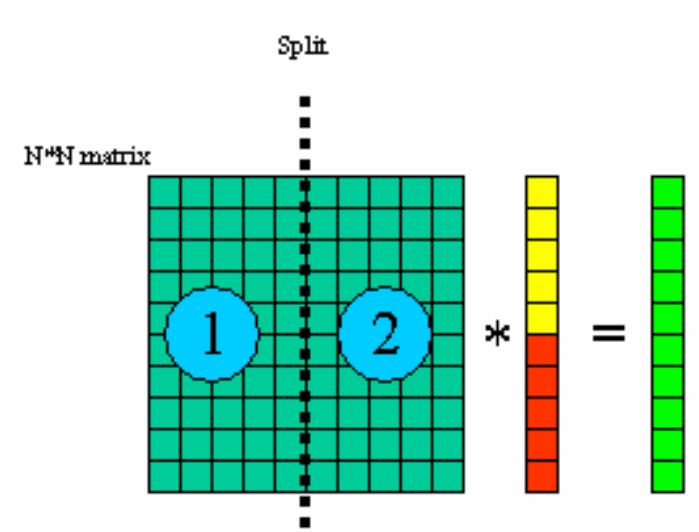
Når det gjelder urettede grafer, som er den typen graf som skal analyseres nærmere i dette prosjektet (likhetsgrafer), blir F og B operatorene like. De gir like resultater fordi vektene mellom dokumentene er like store i begge retninger. I dette tilfellet vil en normalisert HITS metode resultere i graden av noder. En ikke normalisert versjon av HITS metoden på en urettet graf vil gi vekter til hver node kalt ”eigenvector centrality” (EVC) (Lempel and Moran 2000). Disse vektene vil være avhengige av lenkestrukturen i hele grafen, eller likhetsstrukturen i tilfellet vi skal se nærmere på.

2.3 Behandling av store matriser i minnet

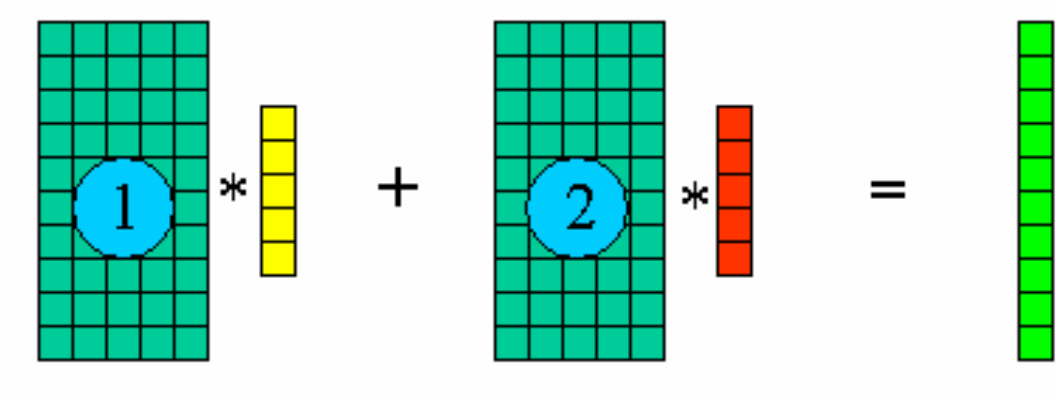
For at lenkeanalyse skal gi mest mulige presise svar må oversikten over lenkene i domenet være så komplett som mulig. All denne dataen må tas i betraktning når man regner ut viktighetsscore for dokumentene. Jo større domenet er jo mer minne trengs i teorien for å kunne gjennomføre utregningene. Dette kan fort bli et problem ettersom antallet dokumenter i domenet vokser. Man trenger metoder for å kunne behandle all denne informasjonen basert på minnet tilgjengelig. En metode for å gjøre dette, som er brukt i dette prosjektet, er å dele opp matrisen som inneholder oversikten over alle

[12] Se appendiks 10.6 for et praktisk eksempel på å forklare “mutually defining”.

dokumentene og deres lenker seg i mellom, i mindre deler. Videre beregnes hver del for seg, før resultatene fra de forskjellige beregningene adderes sammen til det ferdige, komplette resultatet. Et eksempel på dette er vist i figurene under.



Figur 2 - Illustrasjon over hvordan man teoretisk regner ut viktighetsverdier for alle dokumenter i et domene med N dokumenter.



Figur 3 - Illustrasjon over hvordan man praktisk regner ut viktighetsverdier for dokumenter i et domene med N dokumenter avhengig av tilgjengelig minne.

3 Beskrivelse av T-Rank og hvordan den fungerer

Vi vil i dette kapittelet beskrive teorien og funksjonaliteten til en lenkeanalyseringsmetode oppfunnet ved Telenor R&D som har fått navnet T-Rank. Vi vill også komme med noen eksempler på hvordan T-Rank brukes og hva slags resultater den gir. Til slutt i kapittelet forteller vi hvordan T-Rank brukes i denne oppgaven.

T-Rank sin posisjon i lenkeanalysebildet skapt i kapittel 2.2 kan best vises ved en tabell.

Operator \ Normalisering	Normalisert	Ikke normalisert
FB and BF	Lenke Popularitet	HITS
Forward	PageRank	**T-RANK (F)**
Backward	DHHZS	**T-RANK (B)**

Tabell 1 - T-Rank sin posisjon i lenkeanalyseverdenen

3.1 T-Rank - overordnet

T-Rank er en lenkeanalyseringsmetode utviklet ved Telenor R&D (Canright and Engø-Monsen 2005). Den fungerer etter samme prinsipp som den mer kjente PageRank metoden som søkemotoren Google har hatt stor suksess med. Mer spesifikt ser både PageRank og T-Rank på lenker funnet i et dokument som en anbefaling fra dette dokumentet til der hvor lenken peker til. En oversikt over dokumenter i et domene, og deres anbefalinger seg imellom, utgjør en graf hvor nodene er dokumenter og anbefalingene er lenker.

T-Rank skiller seg ut fra PageRank på den måten at den ikke straffer lenker funnet i dokumenter som inneholder flere andre lenker ved å redusere poeng til de dokumenter hver enkelt lenke peker til. Med andre ord anser T-Rank alle lenker i et dokument som like viktige i forhold til hvor viktig den mener dokumentet hvor lenken ble funnet er ansett. Målet på viktighet som T-Rank gir til et dokument er en tallverdi som i denne oppgaven vil kalles ”poeng”. Disse poengene er ferdig utregnet etter at hele grafen er analysert. De ferdig utregnede poengene til hvert dokument i en graf kan brukes til å rangere dokumentene i grafen etter hvor viktig T-Rank lenkeanalyse mener dokumentene er. Poengene kan derfor anses som rangeringsverdier. Den faktiske størrelsen på poengene er uinteressant, det er forholdet mellom hvert enkelt dokuments poeng i hele grafen som er interessant. I motsetning til PageRank, som fordeler poengene over alle lenker funnet i et dokument, har T-Rank som filosofi at en peker (en anbefaling fra et

dokument til et annet) ikke er mindre viktig fordi om den befinner seg i et dokument som også anbefaler flere andre dokumenter.

T-Rank har ikke blitt testet på en reell dokumentsamling før.

3.2 T-Rank - inngående

T-Rank er et sett algoritmer for hvordan man skal rangere, eller finne viktighetspoeng, eller vekter, blant dokumenter som inneholder lenker eller pekere seg imellom.

Rangering av dokumenter oppnås ved at hvert dokument tilegnes en vekt, og dokumentene kan så sorteres i henhold til disse vektene. Nodene i grafen er dokumentene i domenet, og de (rettede) lenkene er pekerne mellom dokumentene. T-Rank bruker strukturen i den rettede grafen til å finne vektene til hver node.

Vektene til hver node oppnås ved gjentatt anvendelse av en operator. Denne operatoren fordeler vektene som er tilgjengelig (er blitt akkumulert) for hver iterasjon. Etter mange iterasjoner slutter vektene å forandre seg, og konvergerer til stabile verdier. Disse konvergerte verdiene er viktighetspoengene for hver node. T-Rank trenger en mattematisk metode for å iterere. Metoden for å iterere brukt i dette prosjektet kalles PowerMethod (Langville and Meyer 2004).

T-Rank metoden er ny på den måten at den ikke har blitt brukt, publisert eller patentert før, og består av to nye operatorer som finner vekter til noder i en graf. Disse er ”ikke normalisert Forward operator F” og ”ikke normalisert Backward operator B”, som man kan se i Tabell 1 ovenfor.

3.3 Forskjellige typer rangeringsverdier som T-Rank kan gi

I kapittel 2.2.3, hvor HITS sin virkemåte er forklart nevnes ”Authority” og ”Hub”. Dette er definisjoner laget av Kleinberg i forbindelse med HITS metoden. Videre i denne oppgaven vil derfor ”authority” og ”hub” bli brukt for å beskrive den generelle ideen ved hva Kleinberg legger i sine definisjoner av ”Authority” og ”Hub”.

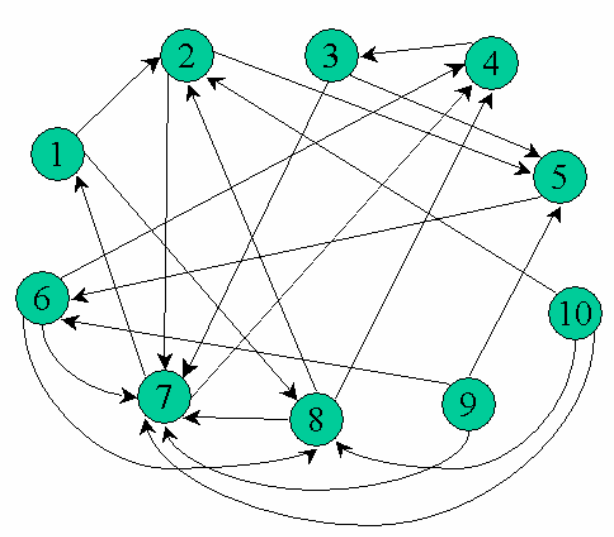
T-Rank er i stand til å regne forskjellige typer rangeringsverdier. Disse er kalt henholdsvis ”authority” og ”hub”, og blir begge funnet ved hjelp av lenkeanalyse. For å forklare de forskjellige rangeringsverdiene vil vi bruke som eksempel vanlige sider på Internett med lenker seg imellom.

Authority-poeng er en verdi som gjenspeiler hvor mange andre dokumenter som har anbefalt et aktuelt dokument. Slike poeng er regnet ut og tilegnet hvert dokument i en graf. Det er forskjellige metoder å regne ut denne poengverdien på (Brin and Page 1998), f.eks. T-Rank eller PageRank. T-Rank poengene som regnes ut i dette prosjektet er en form for authority poeng.

Hub-poeng reflekterer hvor god en side er til å henvise til andre sider som har høy Authority-verdi. Hub-poeng er ofte nyttige om man ønsker søkeresultater med sider som ikke nødvendigvis handler om det man søker etter, men inneholder en bra oversikt over forskjellige typer steder med relatert informasjon om søket man har gjort. Dette er ofte tilfellet i websøk, hvor brukeren angir søket sitt med ord som ikke alene er i stand til å definere hva brukeren mener ettersom den relaterte informasjonen funnet er veldig omfattende og av forskjellig art. Hub-poeng kan sees på som en bra måte å finne sider som opererer som en indeks over informasjonen man er ute etter. Et eksempel er et søk på ”BUSH” som kan returnere en side som definerer ordet på forskjellige måter og henviser til andre steder hvor ordet er relatert, men med forskjellig betydning: ”Deep in the bush-jungle where people rarely ever go”, ”Whitehouse with George Bush”, ”Get a cheap scrubbing bush”, ”Bush industries” etc.

3.4 T-Rank eksempel rettet graf

Under er et eksempel på T-Rank kjørt i et domene bestående av ti noder med rettede lenker seg imellom. Grunnet det lave antallet noder i dette eksempelet, er det mulig å foreta en manuell inspeksjon av resultatene og få en følelse av hvordan T-Rank fungerer på rettede grafer.



Figur 4 - Eksempeldomene bestående av ti noder med rettede lenker seg imellom

En rask manuell inspeksjon av figuren på forrige side gir følgende forventede resultater:

- Sterk ”authority”: Node 7 (og muligens 4)
- Svak ”authority”: Nodene 9 og 10

T-Rank kjørt på dette eksempelet, med tilfeldigsurferverdi satt til 0.2, gav følgende resultater for ”authority”:

The Matrix

	Node 1	Node 2	Node 3	Node 4	Node 5	Node 6	Node 7	Node 8	Node 9	Node 10
Node 1	0.200	0.200	0.200	0.200	0.200	0.200	1.000	0.200	0.200	0.200
Node 2	1.000	0.200	0.200	0.200	0.200	0.200	0.200	1.000	0.200	1.000
Node 3	0.200	0.200	0.200	1.000	0.200	0.200	0.200	0.200	0.200	0.200
Node 4	0.200	0.200	0.200	0.200	0.200	1.000	1.000	1.000	0.200	0.200
Node 5	0.200	1.000	1.000	0.200	0.200	0.200	0.200	0.200	1.000	0.200
Node 6	0.200	0.200	0.200	0.200	1.000	0.200	0.200	0.200	1.000	0.200
Node 7	0.200	1.000	1.000	0.200	0.200	1.000	0.200	1.000	1.000	1.000
Node 8	1.000	0.200	0.200	0.200	0.200	1.000	0.200	0.200	0.200	1.000
Node 9	0.200	0.200	0.200	0.200	0.200	0.200	0.200	0.200	0.200	0.200
Node 10	0.200	0.200	0.200	0.200	0.200	0.200	0.200	0.200	0.200	0.200

The Vector

Node 1	5.520
Node 2	6.686
Node 3	5.114
Node 4	8.147
Node 5	6.643
Node 6	5.514
Node 7	10.000
Node 8	6.475
Node 9	3.328
Node 10	3.328

Figur 5 - "autority"-verdi resultater av T-Rank kjørt på eksempeldomenet med 10 noder

Man kan se at resultatet stemmer godt overens med forventningene.

3.5 Hvordan T-Rank brukes i denne oppgaven

I denne oppgaven brukes T-Rank på den ”tradisjonelle måten” til å analysere lenkestrukturdatabasen (som tilsvarer en rettet graf) i dokumentsamlingen. Men i tillegg brukes den på en annen og noe uvanlig måte, nemlig på likhetsmatriser, eller symmetriske grafer. Vi har ikke funnet noe informasjon om andre som har gjort dette før. Startmatrisen vil da ha forskjellige startverdier. Det er veldig spennende å se hvilke relasjoner dokumentene danner seg imellom basert på dokumentlikheter med denne metoden. Og om det vil gi logiske resultater.

Det er viktig å ha klart for seg at PageRank resulterer i ”node degree”, som er det samme som å telle antall lenker pekende mot et dokument, i symmetriske grafer. Mens T-Rank resulterer i ”Eigenvector Centrality”, som er en mer beskrivende og avansert form for poengberegning, i symmetriske grafer (nærmere forklar i kapittel 2).

Vi vil påstå at T-Rank passer godt til dette formålet, og at den egner seg mye bedre enn PageRank til dette formålet *nettopp fordi* den ikke straffer vektflyt (poeng rettet) mot dokumenter hvis dokumentet som gir poeng (fordi det er likt det dokumentet det gir poeng til) er likt mange andre dokumenter i tillegg til det aktuelle dokumentet. Med PageRank ville disse poengene blitt svekket fordi den normaliserer poengene over mange av de andre dokumentene i dokumentsamlingen ettersom dokument er litt likt *mange* andre. Dette er antageligvis ikke en fordel, og følgelig vil antageligvis T-Rank gjøre det bedre enn PageRank brukt i denne sammenhengen. Denne straffen blir, med T-Rank, allikevel på en måte ”tatt hensyn til” ved at et dokuments likhet til andre dokumenter varierer. Derfor er det unødvendig å straffe ekstra (ved å normalisere) når et dokument har en viss likhet med mange andre. Denne likheten er av forskjellig størrelse for de forskjellige dokumentene og det er ”straff” nok.

I neste kapittel skal vi se på hvordan vi har tenkt å regne likheter mellom dokumenter og forskjellige metoder for å gjøre dette effektivt i store samlinger.

4 Beregning av likhet mellom dokumentpar

Dette kapittelet starter vi i avsnitt 4.1 med å diskutere rundt temaet å beregne likhet mellom mange dokumenter i en samling. Vi nevner noen fordeler og ulemper på hvordan dette bør gjøres. Og vi gir vårt syn på hvorfor vi mener dette kan fungere som en substitusjon i domener hvor lenker er utilgjengelige. Avsnitt 4.2 tar for seg hvordan vi har tenkt å regne likhet mellom to dokumenter. En mattematisk metode for dette blir beskrevet, og dette er den metoden som brukes under utviklingen av B3000. Neste kapittel tar for seg noen praktiske detaljer rundt hva slike beregninger innebærer for en maskin. Videre i kapittelet kommer vi med tre forslag til forskjellige metoder som kan brukes for å beregne likhet mellom alle mulige kombinasjoner av dokumentpar i en samling. Dette antallet kan fort bli stor, og det er derfor viktig å ta seg tid til å finne en bra metode for dette før utviklingen begynner. På denne måten sparer vi oss for mye unødvendig prosessering under utviklingen av søkemotoren. Den første metoden nevnt (Metode 1, avsnitt 4.4) er mer eller mindre ”rett fram” og er den enkleste av de tre som nevnes. Den siste metoden (Metode 3, avsnitt 4.6) er kanskje den mest avanserte, og er den som brukes under utvikling i kapittel 7.

Generelt mener vi at likhet mellom dokumenter, til en viss grad, vil kunne fungere som tekstrelevans. Dette mener vi kan skje ved å regne likhet mellom kun de dokumenter som inneholder de ordene brukt i søkefrasen. På denne måten lages en subgraf bestående av dokumenter med en viss relasjon til søkestrengen brukt. Mer informasjon om tekstrelevans finnes i kapittel 2.1 ”*Søkeords relevans til dokumenter*”.

4.1 Fordeler og ulemper ved likhetsanalyse som substitusjon for lenker

En metode for å beregne likheten mellom dokumenter er basert på en normalisert ordfrekvensfordeling mellom par av dokumenter. Denne metoden innebærer $N^2/2$ antall par, når N er antall dokumenter i domenet. Hvert par kan ha sin egen relevansverdi, og alle relevansverdiene til sammen danner en fullkommen graf[13] over domenet, hvor dokumentene er noder. Denne grafen kan minne veldig om en graf som beskriver alle (rettede) lenker mellom dokumenter i et domene med lenker. Men det er en stor forskjell: Denne likhetsgrafen er nemlig urettet. Det medfører at vanlige lenkeanalyseringsmetoder, som er laget med tanke på å analysere rettede grafer, vil gi annerledes resultater om man kjører de på en urettet eller symmetrisk graf.

[13] Se appendiks 10.6 for informasjon om hava en “fullkommen graf” er

Mer spesifikt kan alle lenkeanalyseringsmetodene HITS, T-Rank og PageRank fungere bra og på forskjellige måter på rettede grafer. PageRank's virkemåte kollapser, og blir til "node in degree"[14] i symmetriske univers. Mens HITS og T-Rank blir til en og samme metode (gir like resultater) i symmetriske omgivelser. Forward operatoren F og backward operatoren B, som er nærmere beskrevet i kapittel 2.2.3, smelter sammen til en og samme operator (EVC) i symmetriske univers i og med at "lenkene" ikke har noen retning.

Om man ønsker å bruke lenkeanalyseringsmetoder som hjelp til å rangere dokumenter funnet i domener uten lenker kan et enkelt første forsøk være å beregne dokumentenes likhet mellom hverandre. Videre kan man bruke disse beregningene alene eller i tillegg til den dårlige pekerinformasjonen som man kan klare å finne i samlingen. Fordelen med denne metoden er at den ikke er avhengig av at dokumentsamlingen har en god lenkestruktur. Ulempen er at likhet mellom dokumenter i en samling sier noe om hvor likt et dokument er et eller flere andre dokumenter i samlingen, og lite om hvorvidt dokumentet er viktig eller ikke. Det gir altså ingen informasjon om hvilke dokumenter som anbefaler et annet. Det kan allikevel være interessant å analysere disse likhetene for å få et mål på hvor unikt eller generelt et dokument er i den totale samlingen, eller blant et utvalg av samlingen innen et visst tema. Denne informasjonen kan sannsynligvis med fordel brukes som et hjelpemiddel til å forbedre den sparsomme informasjonen man har til rådighet for analyse til å rangere dokumenter i domener med dårlig lenkestruktur.

En likhetsmatrise vil muligens gi mer meningsfylte resultater om man ser på likheten mellom dokumenter innen et gitt tema. Et dårlig alternativ er å i stedet ta i betraktning alle dokumentene i en samling for å lage en parvis likhetsmatrise over disse og analysere denne matrisa. Dette fordi å regne likheten mellom alle par av mange dokumenter fort kan bli en meget tidkrevende og tungvint jobb, og fordi det er usikkert hva resultatet egentlig vil bety. Det er vanskelig å forestille seg hvordan et slikt resultat kan brukes til noe fornuftig, det sier egentlig ikke annet en hvor gjennomsnittlig eller uvanlig et dokument er blant mange andre.

[14] "node degree" er å gi dokumenter poeng basert utelukkende på antallet andre dokumenter som peker mot det aktuelle dokumentet.

Det virker derfor fornuftig å påstå at en likhetsmatrise er forventet å gi mest mening om man kan si at to dokumenter er like fordi de handler om et visst tema

- Lenker sier to ting: ”du er bra” og ”du er mer eller mindre lik meg”. Lenker egner seg derfor godt som base for analyse for å finne ut hvilke dokumenter som er verdifulle eller ikke.

- Å beregne likhet mellom dokumenter og analysere dette sier noe om hvor unikt eller gjennomsnittlig hvert dokument er i forhold til andre dokumenter. Dette vil forhåpentligvis fungere som et bra mål om man ser på dokumenter som handler om et spesielt tema, for så å gjøre analysen på disse dokumentene (subgrafen).

Konklusjonen på dette blir at det er meget interessant å se på hvilke resultater det er mulig å oppnå dersom T-Rank tas i bruk på en symmetrisk likhetsgraf[15] alene, eller en symmetrisk likhetsgraf addert sammen med en graf over den svake lenkestrukturen funnet i domenet. På denne måten vil T-Rank få en svakt rettet graf å analysere. Med svakt rettet menes det at veldig få av nodene i domenet peker til andre noder uten at den / de andre nodene peker tilbake med samme verdi. Grunnen til at det er mulig å lage seg slike grafer er fordi man adderer sammen likhetsmatrisa over alle dokumentene med en matrise over den svake lenkestrukturen funnet. Se kapittel 5 for informasjon angående hvordan en slik addisjon gjøres i praksis.

4.2 Likhet mellom to dokumenter

Likheten (similariteten) mellom to dokumenter skal, i denne oppgaven, gi et svar som skal gjenspeile om dokumentene handler om det samme, og gi en tallverdi på hvor like de er. Forskjellige måter finnes for å avgjøre dette (Chen 1997; Allan, Leuski et al. 2000; Liu, Meng et al. 2000) flere av disse baserer seg på kunstig intelligens og naturlig språk analyse (Nilsson 1998). Det er i denne oppgaven ønskelig å se på en rent statistisk metode istedenfor AI baserte metoder.

Basert på undersøkelser av metoder som allerede finnes har vi kommet frem til følgende metode som er interessant og se videre på. Den baserer seg på å normalisere ordfrekvensen i hvert dokument. På denne måten tilegnes et tall til vert unike ord i dokumentet som gjenspeiler ordets vekt i dokumentet i forhold til de andre unike ordene i dokumentet. Metoden vi skal bruke for denne utregningen fungere som beskrevet:

[15] Se appendiks 10.6 for informasjon om hva en “symmetrisk likhetsgraf” er.

For dokument *A*, tell antall ganger hvert ord (ord *i*) er brukt i dokumentet, og kall dette nummeret $n_i(A)$. Deretter vil vi finne ut hvor stor vekt vi skal legge på dette ordet i forhold til resten av dokumentet. Resultatet kaller vi for *p*. Dette gjør vi ved å regne ut: $p_i(A) = n_i(A) / \sum_i n_i(A)$. Vi ønsker å legge større vekt på et ord jo flere ganger det oppstår i et dokument, men vekten må stå i forhold til dokumentets størrelse slik at vi må ta i betraktning alle andre unike ord som er i dokumentet, når vi regner ut p_i.

Vi vil ikke ta i betraktning stoppord fordi stoppord ikke er bra til å beskrive tekstens mening. Det er også ganske enkelt å implementere fjerning av stoppord. Det er ikke brukt stemming[16] i dette prosjektet, men det vil være en fordel i fremtidige prosjekter for å få en bedre sammenligning av konteksten til dokumentene.

Videre ønsker vi å få angitt en verdi mellom 0 og 1 som antyder hvor like to dokumenter er. Det gjøres med følgende formel: $Sim(A,B) = \sum_i \sqrt{p_i(A) p_i(B)}$. Kvadratroten i formelen sørger for at svaret ikke blir for lite. Dette synes godt i de tilfeller hvor A og B er veldig like. En sjekk på dette er tilfellet *Sim(A, A)* som skal bli 1. Vi vet at $\sum_i p_i(A) = 1$, men siden alle $p_i(A) < 1$ medfører dette at $Sim(A,A) = \sum_i p_i^2(A) < 1$. Det er derfor viktig med kvadratroten i formelen, det sørger for at svaret er større enn 0 og mindre enn eller lik 1, og at likheten mellom to like dokumenter alltid blir 1 fordi:
$Sim(A,A) = \sum_i \sqrt{p_i^2(A)} = \sum_i p_i(A) = 1$

4.3 Likhet mellom alle dokumentpar i en samling - praktisk

En forutsetning for å bruke metoden beskrevet i 4.2 er at man først regner ut en normalisert ordfrekvensfordeling over alle dokumentene. Dette vil resultere i en vektor for hvert dokument i samlingen. For å være i stand til å regne likheten mellom disse vektorene så fort som mulig må de regnes ut *en* gang, og lagres. Et alternativ er å lage vektoren for hvert dokument hver gang den trengs. Dette vil resultere i at kjøretiden blir en eksponent større i stedet for å bli multiplisert med en relativt liten konstant k (definert under). Det er derfor absolutt nødvendig å lage disse ordfordelingstabellene *en* gang på forhånd, og ta vare på dem til den store prosessen med å regne likhet mellom alle disse tabellene.

[16] Se appendiks 10.7 for en definisjon av hva "stemming" er.

For små dokumentsamlinger kan denne vektoren lagres i minnet. Koden og selve utførelsen vil være relativt enkel. For store dokumentsamlinger derimot, vil den samme fremgangsmåten komme til kort. Man kan bruke samme metode som for små dokumentsamlinger, og la operativsystemet ta seg av mellomlagring (paging) av vektorene etter behov, men dette vil antagelig være ineffektivt ettersom man ikke vil ha noe kontroll over hvordan det gjøres. En av likhetsberegningsteknikkene skal sammenligne vektorene iterativt alle med alle, man vil derfor trenge raskest mulig tilgang til hver vektor en gang for hver iterasjon. Antall iterasjoner er antall dokumenter i samlingen. En iterasjon er en sammenligning mellom et dokument og alle andre i samlingen hver for seg.

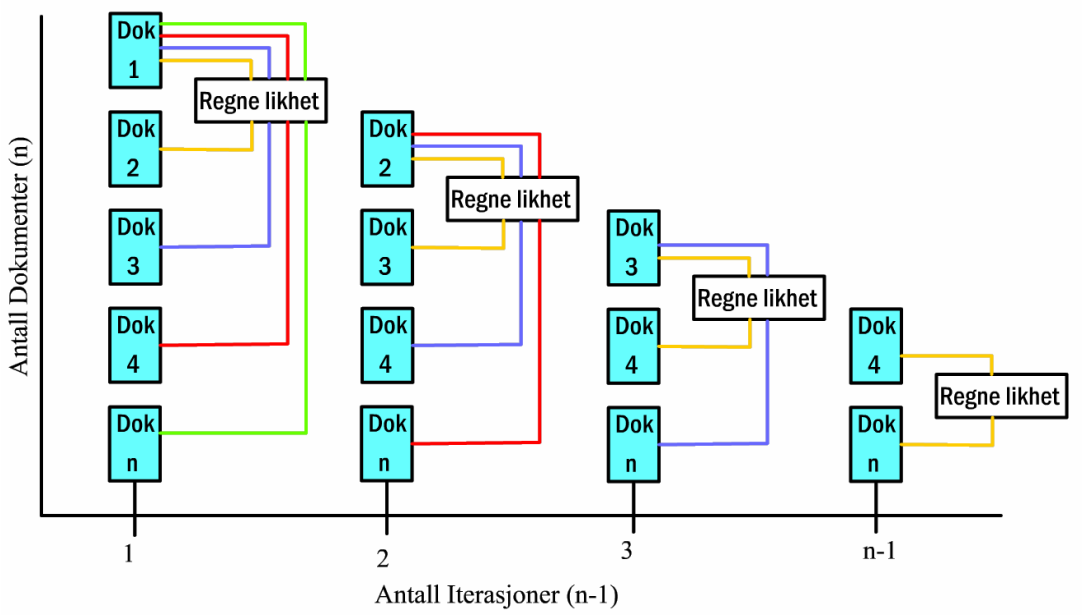
For å få til dette raskest mulig vil det være lurt å lage en metode hvor man selv har full kontroll over hvordan vektorene mellomlagres i stedet for å la operativsystemet ta seg av dette. En måte å gjøre det på er å lagre en ordfrekvensfordelingvektor som en struct-datatype for hvert dokument. Disse kan igjen leses i store kvantum tilbake i minnet etter hvert som de trengs. For eksempel kan man ved begynnelsen lese opp structene til dokumentene 1 – 10.000 i minnet og gjøre alle sammenligninger der. Deretter lese opp dokumentene 10.001 – 20.000 osv. Tilgjengelig RAM og corpus[17] bestemmer antallet structer man kan lese inn i minnet av gangen. Jo flere structer man kan lese inn i minnet samtidig, jo raskere vil kjøretiden bli. Det er denne metoden vi har brukt under utviklingen av komponenter for beregning av dokumentlikheter (kode for dette finnes vedlagt på CD).

Videre i dette kapittelet kommer vi med tre forskjellige forslag til hvordan det kan være mulig å gå frem for å regne likhet mellom alle dokumenter i en samling.

4.4 Likhet mellom alle dokumentpar i en samling - Metode 1

Et første forslag til å beregne likhet mellom alle dokumentpar i en samling vil være enkelt og greit å ta for seg hvert dokument og starte den tunge prosessen med å regne likheten mellom dette dokumentet og alle de andre, deretter neste dokument og likheten mellom dette og alle de andre bortsett fra det første, deretter det tredje dokumentet og likheten mellom dette og alle de andre bortsett fra de to foregående, etc. For hver iterasjon med sammenligninger mellom et dokument og alle de andre minker antall sammenligninger med 1. Dette illustreres i Figur 6.

[17] Se appendiks 10.6 for definisjon av “corpus”



Figur 6 - Enkel metode for å beregne likhet mellom alle dokumentene i en samling.

Antall beregninger i denne operasjonen blir $\sum_{i=1}^{N}\sum_{j=1}^{i-1}1 = \frac{1}{2}\sum_{i=1}^{N}\sum_{j\neq i}^{N}1 = \frac{N(N-1)}{2} \approx \frac{N^2}{2} = \underline{\underline{kN^2}}$,

hvor *i* er ett enkelt dokument i samlingen og *N* er antall dokumenter i hele samlingen, og *k* er en konstant på 0.5. Med tanke på at hver enkelt "Regne likhet" boks er iterasjoner med beregninger mellom hvert ord i hvert par dokumenter blir kjøretiden ytterligere øket i forhold til corpus. Denne økningen vil være $Avg(\#ord) * kN^2$, hvor *Avg(#ord)* er gjennomsnittet av antall ord i hvert dokument i samlingen. Dette er med på å øke *k* betraktelig. Om dette gjennomsnittet er for eksempel 1000 får man en kjøretid på omtrent $500t * N^2$, hvor *t* er tiden det tar å beregne (eller hente en allerede beregnet verdi) prosentvis likhet for ett ord i to dokumenter. Å finne prosentvis likhet for et ord mellom to dokumenter gjøres ved å telle antall ganger ordet oppstår i begge dokumentene og dele dette antallet på det totale antallet ord i hvert dokument. På denne måten har man funnet fraksjonen av dokumentet som inneholder (eller "handler om") det aktuelle ordet. Videre skal man gange disse to verdiene sammen og ta kvadratet av svaret. Dette skal man gjøre for alle de felles ordene i dokumentparet. Se for øvrig kapittel 4.2 for mer detaljer om hvordan likheter mellom dokumenter beregnes i dette prosjektet. Alle disse operasjonene for å sammenlikne frekvensen av et ord i et dokumentpar utgjør en nevneverdig kjøretid på en maskin.

Ved å foreta et "test-run" på dokumentsamlingen[18] fant vi ut at beregningene til sammen ville ta minst to uker! Av resultatene vi fikk så vi at de aller fleste sammenligningene resulterte i 0, eller en veldig lav verdi, noe som ikke var uventet. Det er dumt å måtte bruke mye tid på å beregne nullverdier, ettersom dette resultatet ikke er interessant. Det hadde derfor vært fint om det kunne la seg gjøre å plukke ut de dokumentparene vi antar vil ha en høy eller lav likhet slik at vi kan unngå de unødvendige beregningene av 0.

Eksempler på domener hvor denne metoden kan brukes kan være en lokal harddisk eller intranett. Det ikke usannsynlig at antallet dokumenter i en dokumentsamling kan komme opp i flere hundre tusen. Det er et problem å foreta alle til alle sammenlikninger av disse dokumentene. Kjøretiden i dette scenarioet er nesten $O(N^2)$. Denne kjøretiden baseres på antall sammenlikninger *etter* at hvert dokument har fått beregnet sin ordfrekvens fordeling $p_i(dok)$.

Hvis dokument 1 har fått beregnet en score i forhold til dokument 2, så har også dokument 2 fått beregnet sin score i forhold til dokument 1. Svaret er altså symmetrisk, siden vi regner svaret til et *dokumentpar* for hver utregning. I tillegg slipper vi å regne ut et dokuments likhet med seg selv, da vi vet at dette er 1 i følge formelen angitt over. Dette medfører at kjøretiden blir: $(N-1) * (N/2) \approx N^2$, hvor N er antallet dokumenter i domenet. I tillegg kommer den ekstra kjøretiden, som beskrevet i 4.3.

Metoden skal med fordel brukes offline, men i enkelte tilfeller (se kapittel 5) er det nødvendig å bruke den i sanntid. Hvis vi har et domene med 100.000 dokumenter (noe som ikke er usannsynlig) vil antallet sammenlikninger bli $100.000^2 \approx 10$ milliarder sammenlikninger. Under testing fant vi ut at 100.000 sammenlikninger tok ca 10 minutter[19] på en 3GHz Pentium. Dette tilsvarer i gjennomsnitt 0,006 sekunder per sammenlikning. Å beregne likheten mellom alle dokumentpar i en samling på 100.000 dokumenter er derfor forventet å ta omtrent: $(100.000^2 * 0{,}006) / (60 * 60 * 24 * 365) \approx$ 2 år!

Denne metoden opererer ikke med noen stoppkriterier eller terskelverdier i motsetning til de to andre metodene vi foreslår under. Den eneste måten å finne antall dokumenter med en viss likhet med denne metoden er å foreta samme beregning for alle mulige par. Det er tydelig at det trengs en annen metode for alle til alle sammenlikninger mellom dokumenter.

[18] Se kapittel 6 for informasjon om hvilken dokumentsamling som er brukt i denne oppgaven.
[19] Denne testen ble ikke gjennomført på samme server som er beskrevet i kapittel 7.1.1, som er serveren brukt for å utvikle resten av metodene i oppgaven.

4.5 Likhet mellom alle dokumentpar i en samling - Metode 2

Metode 2 har som utgangspunkt at man kun er interessert i de dokumentene i en dokumentsamling som er mest mulig like i forhold til resten av samlingen. Vi er altså ikke interessert i dokumenter som er antatt å ikke være like hverandre i forhold til samlingen. Hvor mange dokumenter som er lik det aktuelle dokumentet man ser på vil variere fra dokument til dokument.

Først må man lage *en felles* ordfrekvensvektor over *alle* dokumentene i samlingen. Denne matrisa skal fungere som en slags felles sammenligningsmatrise. I stedet for å sammenlikne alle dokumentene mot hverandre sammenlikner man alle dokumentene med denne ene matrisa. Dette har en kjøretid på O(N) som er absolutt lønnsomt. Men metoden er ikke på langt nær ferdig. Etter å ha gjort dette vil alle dokumentene få en score som beskriver hvor like de er denne felles matrisa, eller hvor like de er med gjennomsnittet av ordfordelingen i dokumentsamlingen. Denne likheten mistenker vi kan være ganske liten for hvert dokument, men det har ingen betydning for den videre utføringen. Ideen går nå ut på at jo likere to dokumenters score mot denne felles matrisa er *i forhold til* alle scorene i dokumentsamlingen, jo større sannsynlighet er det for at disse to dokumentene vil ha en stor likhet. Vi kan derfor bruke denne antagelsen til å velge ut et visst antall dokumenter, eller alle dokumenter med en score likere enn en viss grense og beregne den korrekte likheten mellom det aktuelle dokumentet og alle de antatt like. På denne måten vil vi få redusert kjøretiden i fra den opprinnelige $O(N^2)$ til O(kN). ”k” er avhengig av hvor mange dokumenter som er funnet antatt like for hvert av de aktuelle dokumentene. Dette tallet vil variere etter hvilket dokument man ser på.

Denne metoden vil garantert ikke finne dokumenter til å være like som ikke er det. Den vil *kanskje* unnlate å finne enkelte dokumenter som er like over en hvis grense, men det vil avhenge av hvor lang tid vi selv velger å bruke på sammenlikning, eller hva vi setter terskelverdien til. Den vil antagelig kunne optimaliseres en del.

Denne metoden må kun brukes til å sammenligne resultatet med en ”brute-force” metode. Dette fordi ALLE sammenligningssvar er viktige når vi skal begynne å kjøre T-Rank på matrisa. Det kan være interessant å se på resultatet når vi ekskluderer de minst like dokumentene før vi begynner å bruke T-Rank på matrisa.

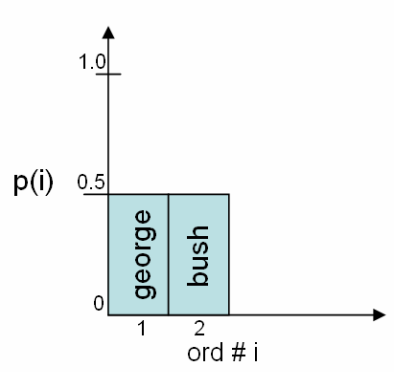
4.6 Likhet mellom alle dokumentpar i en samling - Metode 3

Ideen bak metode 3 baserer seg på tanken om at beregning av de mange 0-verdiene, eller lave verdier for dokumenter som bare er *litt* like, er unødvendig, og ønskelig å unngå. Løsningen på dette kan være å bruke en invertert indeks. Vi lager en indeks over alle ord i dokumentsamlingen og hvilke dokumenter ordene forekommer i. Vi fjerner stoppord og foretar med fordel stemming[20]. Vi vil så rangere alle dokumenter relatert til hvert ord etter hvor relevant de er til ordet. For å finne denne relevansen vil vi ta i bruk den samme likhetsmodellen vi skal bruke for å beregne likhet mellom ord i dokumentpar (beskrevet i 4.2). Et ords vekt vil vi kalle for *p-verdi*. For eksempel vil et dokument som kun består av teksten "Georg Bush" få p(Georg)=0.5 og p(Bush)=0.5. Eksempel:

Dokument 1

Består av kun teksten
"George Bush"

p(georg)=0.5
p(bush)=0.5

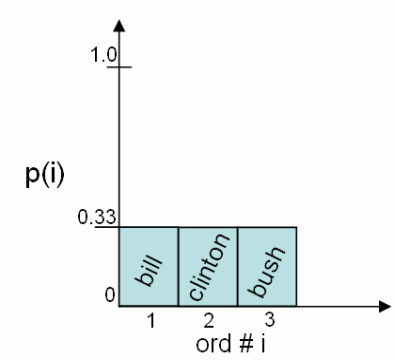


Figur 7, Ordfrekvensprofil for dokument 1

Dokument 2

Består av kun teksten
"Bill Clinton in the Bush"

p(bill)=0.33
p(clinton)=0.33
p(bush)=0.33



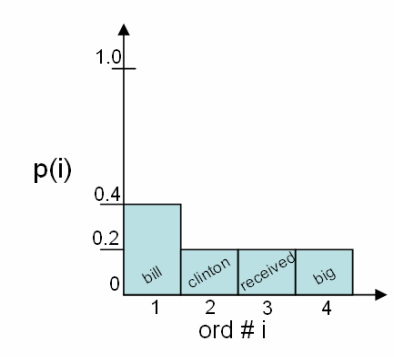
Figur 8, Ordfrekvensprofil for dokument 2

[20] Å fjerne stoppord og foreta stemming vil være med på å redusere corpus betraktelig og vil følgelig ha stor påvirkning på kjøretiden. I dette prosjektet vil vi kun fjerne de vanligste former for stoppord i norsk språk men ikke foreta noe stemming. Begge disse operasjonene kan få store utslag på kjøretiden etter hvor intelligent og avansert man gjør dem.

Dokument 3

Består av kun teksten
”Bill Clinton received a Big Bill”

p(bill)=0.4
p(clinton)=0.2
p(received)=0.2
p(big)=0.2



Figur 9, Ordfrekvensprofil for dokument 3

Dette eksempelet gir oss følgende ord i corpuset etter å ha fjernet stoppordene ”in”, ”the” og ”a”, men ikke foretatt noe stemming:
1: georg
2: bush
3: bill
4: clinton
5: received
6: big
Den ferdige rangerte inverterte indeksen for denne dokumentsamlingen blir som følger. ”Dok #” i tabellen er for hvert ord sortert etter det dokumentet hvor ord # veier tyngst, eller p(i) er størst.

Ord# \ Dok #	Dok # ↓	Dok # ↓	Dok # ↓	Dok # ↓
Ord 1 (georg)	**1** sim(d1,o1)=0.5	2 sim(d2,o1)=0	3 sim(d3,o1)=0	Etc.
Ord 2 (bush)	**1** sim(d1,o2)=0.5	**2** sim(d2,o2)=0.33	3 sim(d3,o2)=0	Etc.
Ord 3 (bill)	**3** sim(d3,o3)=0.4	**2** sim(d2,o3)=0.33	1 sim(d1,o3)=0	Etc.
Ord 4 (clinton)	**2** sim(d2,o4)=0.33	**3** sim(d3,o4)=0.2	1 sim(d1,o4)=0	Etc.
Ord 5 (received)	**3** sim(d3,o5)=0.2	2 sim(d2,o5)=0	1 sim(d1,o5)=0	Etc.
Ord 6 (big)	**3** sim(d3,o6)=0.2	2 sim(d2,o6)=0	1 sim(d1,o6)=0	Etc.
Ord ω	Dok # med p(max)	Dok # med p(2. største)	Dok # med p(3. største)	Dok # med p(4. største)

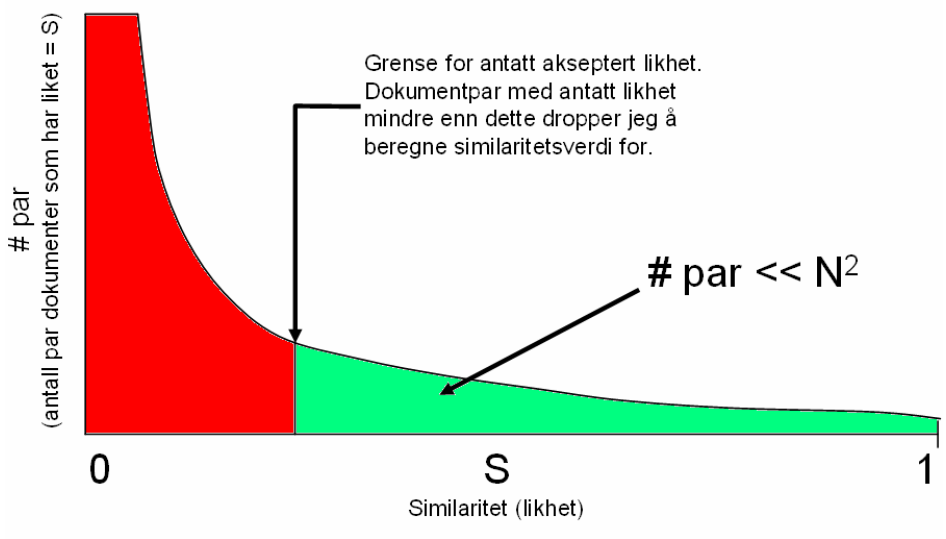
Tabell 2 - Eksempel på invertert indeks over alle ord i dokumentsamlingen. Hver dokument-ID er for hvert ord # sortert etter det dokumentet som er mest relatert til ordet.

Merk at det ikke er nødvendig å inkludere i den inverterte indeksen de dokument numre som har likhet 0 med ordet i venstre kolonne, det er kun tatt med her for oversiktens skyld. NB! I dette eksempelet er ”Dok #” kun et nummer som identifiserer et dokument, eksempelet sier altså ingenting om *hvor* relatert dokumentet er til ordet. Det er egentlig ikke nødvendig ettersom metoden skal regne likhet mellom dokumentpar og ikke likhet mellom dokumenter og et ord. I Tabell 2 har vi skrevet inn likheten i tillegg til dokument nummeret for oversiktens skyld. Vi har uthevet de dokument numrene som har en likhet til ord numre som kan gjøre det verdt å se nærmere på. ”Etc.” i Tabell 2 er ment som utvidelse når dokumentsamlingen eventuelt skulle vokse. Dette eksempelet er ment for å gi leseren et inntrykk av hvordan metoden virker. I praksis vil den være noe annerledes, se vedlagt kildekode på CD.

Intuitivt vil dette gi en god mulighet til å finne de dokumentpar vi kan anta har en viss likhet. Måten vi skal finne disse dokumentparene på er å plukke ut det dokumentet som har størst likhet til et ord i samlingen og beregne likheten mellom dette dokumentet og det dokumentet i samlingen som har nest størst likhet til det samme ordet. Disse to dokumentene er lett å finne ettersom alle dokumentene allerede er rangert fra største til minste likhet for hvert ord. Etter å ha beregnet likhet for dette dokumentparet fortsetter vi ved å beregne likheten mellom dokumentet mest relatert til ordet og dokumentet som har 3. mest likhet til ordet. Deretter likheten mellom 1. og 4. dokument mest relatert til ordet, deretter 1. og 5. mest relatert til ordet, etc. Vi forventer at likheten mellom dokumentparene, mer ofte enn ikke, minker ettersom vi fortsetter slik. Slik fortsetter vi helt til likheten mellom dokumentparene blir mindre enn en akseptabel terskelverdi (terskelverdi 1). Da er vi ferdig med å beregne likhet mellom det dokumentet mest relatert til et spesielt ord i den inverterte indeksen og de dokumentene vi mener har høyest sannsynlighet for å være like dette dokumentet. Videre plukker vi ut det dokumentet som har nest mest likhet med det samme ordet i invertert indeks og foretar den samme prosessen igjen. Når vi er ferdige fortsetter vi ved å iterere oss nedover skalaen med det dokumentet med 3. størst likhet til invertert indeks ordet, osv til vi når det dokumentet med så liten likhet til ordet at den er mindre enn en terskelverdi (terskelverdi 2). Da begynner vi på neste ord i invertert indeks og foretar samme prosedyre her. Dette gjøre vi for alle ordene i den inverterte indeksen.

Et ords likhet med et dokument i den inverterte indeksen representerer hvor likt dokumentet er med dette ordet basert på hvor prosentvis stor vekt ordet har i dokumentet. Dette innebærer at to dokumenter med en viss likhet også **må** ha ett eller flere felles ord som de er like, og følgelig vil dokumentparet bli funnet av metoden beskrevet gitt at terskelverdiene 1 og 2 er lave nok. Figur 10 illustrerer hvordan vi ser for oss antallet dokumentpar i forhold til likheten deres.

En tungtveiende fordel ved denne metoden er at dokumentpar med lav likhet vil unngå å bli tatt med som kandidater for likhetsberegning. Og det kan være snakk om veldig mange. Ulempen ved denne metoden er at den vil, i spesielle tilfeller og avhengig av terskelverdiene 1 og 2, kunne utelate å beregne likhet mellom dokumenter som er ganske eller helt like.



Figur 10 - Forventet antall dokumentpar kontra likheten mellom dem.

Hvis hyppigheten til ordene i dokumentene er prosentvis veldig jevne vil dokumentene antagelig havne langt ned på den inverterte indeks lista fordi ordene vil få en lav eller gjennomsnittlig likhet med dokumentene. Eksempelvis vil to dokumenter som kun inneholder hver bokstav i alfabetet med mellomrom mellom være helt like, men de vil antagelig aldri bli plukket ut som kandidater av metoden fordi de vil rangeres langt ned i invertert indeks for alle ordene. Hvert ord (i eksempeltilfellet en alene stående bokstav) i begge dokumentene vil få en vekt på p(1/26) hvis dokumentet kun inneholder bokstavene A-Z[21]. For å "fange" slike like dokumentpar blir vi nødt til å eksperimentere og senke terskelverdiene 1 og / eller 2. Jo lavere disse settes, jo flere dokumentpar er metoden forventet å finne.

Terskelverdi 2 er en minimumsgrense for hvor relatert et ord må være i et dokument. Hvis et dokument er mindre relatert til et ord i den inverterte indeksen enn terskelverdi 2 vil metoden stoppe beregningen av likheter mellom dokumenter oppført for det aktuelle ordet. På samme måte vil terskelverdi 1 stoppe metoden fra å fortsette å beregne likheter mellom dokumenter oppført for det aktuelle ordet, hvis to dokumenters likhet er mindre enn terskelverdi 1. Hvis terskelverdi 1 og 2 er annerledes fra 0 *har vi ingen garanti* for å

[21] Slike dokumentpar virker uinteressante å ta med i betraktning ettersom de er så gjennomsnittlige og veldig vanskelige å vite hva handler om. Likhetsmatrisa er forventet å gi mest mening om man kan si at to dokumenter er like fordi de handler om et visst tema.

finne dokumentpar med en viss eller stor likhet. Men sannsynligheten for at vi utelater slike par minker jo lavere terskelverdiene settes. Det er forventet at metoden vil finne *alle* dokumentpar med likhet større enn 0 om terskelverdiene 1 og 2 settes til 0. Vi forventer også at metoden vil kjøre fortere enn alle til alle dokumentsammenligning, selv om begge terskelverdiene settes til 0. Dette fordi den vil unngå å beregne likhet mellom alle de (mange) dokumentparene som ikke er like i det hele tatt.

I neste kapittel skal vi se nærmere på hvilke muligheter vi mener vi har for å kombinere beregning av dokumentlikheter med lenkeanalysemetoder og resultater fra analyse av lenkestruktur.

5 Kombinering av dokumentlikheter og lenkeanalyse

Dette kapittelet er bygget opp ved at vi først nevner noen generelle aspekter ved matriseregning, og problemer som kan oppstå rundt dette med forslag til løsning. Samtidig gir vi leseren en innføring i et fenomen kalt ”sinks”. Avsnittet om ”sinks” kunne kanskje hørt hjemme i kapittel 2. Vi har valgt å ha det her fordi vi mener at sinkproblematikken blir en del annerledes i tilfeller hvor man tar dokumentlikheter i betraktning i tillegg til lenker.

Videre i kapittelet innfører vi leseren i de muligheter vi mener vi har for å kombinere beregningene av dokumentlikheter med de tradisjonelle metodene for å analysere lenker. Vi kommer også med flere forslag til hvordan vi ser for oss at en matrise over dokumentlikheter kan brukes sammen med en matrise over lenker, for analyse av tradisjonelle lenkeanalyseringsmetoder. Til slutt konkluderer vi med hvilke av metodene foreslått som er interessante å se videre på.

5.1 Tanker rundt matriseregning

Under er noen tanker om mattematikken rundt matriseregning, hvordan vi vil addere to matriser av ulik størrelse, og noen problemer og løsninger ved et fenomen kalt sinks. I tillegg er informasjon om hva som må til før vi kan begynne å regne på lenke- og likhets-matrisene (similaritetsmatrisene). Det kan være greit for leseren å ha dette klart for seg før kapittel ”5.3 - *Mulige kombinasjoner av lenkeanalyse og analyse av dokumentlikheter*”.

H = Hyperlenke Matrise
S = Similaritets Matrise
$\mathbf{H_h}$ = hele grafen H
$\mathbf{H_s}$ = subgraf av H
$\mathbf{S_h}$ = hele grafen S
$\mathbf{S_s}$ = subgraf av S

Vi ser for oss at H_h i mange tilfeller har en sparsom / liten lenkestruktur. Videre ser vi for oss at S_s kan inneholde mye (kompakt) informasjon om hvordan man kan vekte og rangere dokumentene som grafen gjenspeiler. En viktig forskjell mellom S og H er at et dokument i H som regel vil ha relativt få lenker inn og ut fra seg (typisk 5 – 10 lenker),

mens et dokument i S *kan* ha N-1 lenker inn og ut fra seg[22], hvor N er antall dokumenter i samlingen eller utvalget. Totalt antall lenker i domenet blir i verste fall omtrent N^2. Dette antallet kontra 10*N utgjør en vesentlig forskjell i realistiske domener hvor dokumentantallet fort kan komme opp i mange hundre tusen eller flere millioner. Se for øvrig kapittel 1 for mer informasjon om hvordan likhetsberegning kan foretas.

Vi tror at å bruke hele lenkegrafen H_h gir mening, men at det vil gi enda mer mening å bruke den sammen med S_s, ettersom S_s er generert rundt et visst tema. Man kan tenke seg S_s som å legge til ekstra vekt rundt dette temaet på en utvalgt subgraf i dokumentsettet H_h. Måten vi velger ut subgrafen i S_s er ved bruk av *enkel* tekstrelevans søk.

"H_h og S_s" kan bety:

1. $(H_h + S_s) * e_{hs} = \lambda_{hs} * e_{hs}$

2. $$\left.\begin{array}{l} H_h * e_H = \lambda_H * e_H \\ S_s * e_S = \lambda_S * e_S \end{array}\right\rangle e_{final} = \alpha e_H + \beta e_S$$

1 og 2 er forskjellige ettersom:

$e_{hs} \neq e_H + e_S$
og
$\lambda_H \neq \lambda_S$

Fortsetter med 1. og 2. over:

[22] Dette antallet avhenger av hvor mange dokumenter det aktuelle dokumentet har en likhet med. Før første fasen i utvikling trodde vi dette antallet ville være høyt, gjerne lik N for det aktuelle dokumentet. Men det har vist seg at det er en del mindre. Det er uansett forventet å være mange flere lenker i tilfellet S enn tilfellet H.

$$e_h = \begin{pmatrix} e_{H,n} \\ e_{H,N-n} \end{pmatrix}$$

$$e_s = \begin{pmatrix} e_s \\ 0 \end{pmatrix}$$

$$H_h * e_s = H_h * \begin{pmatrix} e_s \\ 0 \end{pmatrix} = \begin{pmatrix} H_h * e_s \\ 0 \end{pmatrix}$$

$$S_s * e_H = \begin{pmatrix} (n*n) & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} e_{H,n} \\ e_{H,N-n} \end{pmatrix} = \begin{pmatrix} S_s * e_{H,n} \\ 0 \end{pmatrix}$$

$$(H_h + S_s) * (e_h + e_s) = H_h * e_H + H_h * e_s + S_s * e_H + S_s * e_s =$$
$$\lambda_H \begin{pmatrix} e_{H,n} \\ e_{H,N-n} \end{pmatrix} + \underbrace{\begin{pmatrix} H_h * e_s \\ 0 \end{pmatrix} + \begin{pmatrix} S_s * e_{H,n} \\ 0 \end{pmatrix}}_{MERK} + \lambda_s \begin{pmatrix} e_s \\ 0 \end{pmatrix}$$

MERK: Disse to vektorene forandrer seg etter [$\alpha e_{H,n} + \beta e_s$]. Dette er interessant å se videre på.

Et problem i denne sammenheng er at H_h og S_s er av ulik størrelse. Å legge sammen matrise av forskjellig størrelse lar seg ikke gjøre. Men siden ideen virker interessant ønsker vi ikke å forkaste den. Løsningen blir å fylle den minste matrisa med nuller til den når riktig størrelse:

$H_h + S_s =$

(N*N) Sparse	(N*N) sparse
(N*N) sparse	(N*N) sparse

+

(n * n) dense	0
0	0

=

(n*n) dense	sparse
sparse	sparse

Tabell 3 - Eksempel på addisjon av en matrise over lenker som er større enn en matrise over likhet mellom dokumentpar. Den minste matrisa fylles med nuller slik at den blir like stor som den største matrisa.

(sparse: svak / dårlig lenkestruktur, få lenker[23])
(dense: tett / god lenkestruktur)

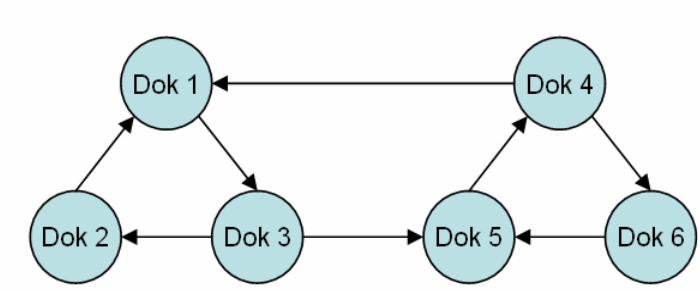
[23] Se Appendiks 10.6 for komplett definisjon av *sparse* og *dense*.

Vi antar at hele grafen er N * N og en subgraf er n * n, hvor n << N. Å legge sammen disse to matrisene kan gjøres ved å skrive nullverdier på de plassene i den minste matrisa som gjør den mindre enn den største matrisa. Deretter kan vi addere matrisene sammen. Man kan se at resultantmatrisa ser mer lovende ut enn begge de andre matrisene hver for seg.

I dette prosjektet vil vi bruke T-Rank på en komplett graf over alle dokumenter i samlingen. Dette vil resultere i en matrise som inneholder mange nuller. Det er matematisk praktisk å lagre denne matrisa *uten* de mange tilfellene som inneholder null, og det vil resultere i *mye* raskere kjøretid. Siden denne metoden er en del mer avansert å implementere, og siden kjøretid ikke er fokus i denne oppgaven vil ikke vi ta høyde for å lagre og behandle matriser på denne måten. Det er uansett viktig å være klar over at en slik løsning er å foretrekke i fremtidig arbeid, og produksjonskode. Spesielt om man må bruke T-Rank i sanntid.

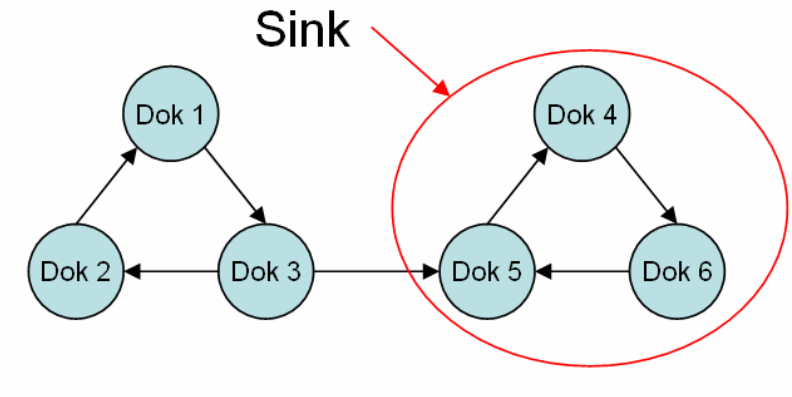
5.2 Sinks

Det er stor sannsynlighet for at H_h ikke er ”fullkomment forbundet” (strongly connected). Med ”fullkomment forbundet” menes at man kan være sikker på å nå en hver node i grafen kun ved å følge retningen til pekerne, uansett hvilken node man starter på. I slike situasjoner opptrer fenomenet sinks. Hvis man følger pekerne i en graf som ikke er ”fullkomment forbundet” vil man kunne nå noder (dokumenter), eller nodesamlinger (flere dokumenter som peker seg i mellom) man ikke kommer ut av ved kun å følge pekerne. Disse nodene kalles for ”sinks”. Sinks ødelegger for tradisjonell lenkeanalyse. I slike tilfeller trenger man derfor et sink botemiddel, eller ”sink remedy”.



Figur 11 - Eksempel på et sett dokumenter med lenker seg imellom som ER "fullkomment forbundet". Domenet inneholder ingen ”sinks”.

I Figur 11 vises et dokumentsett med lenker seg imellom som er fullkomment forbundet. Dette er det ideelle tilfellet for lenkeanalysemetoder.



Figur 12 - Eksempel på et sett dokumenter med lenker seg imellom som IKKE er "fullkomment forbundet". Domenet inneholder "sinks".

Figur 12 viser en dokumentsamling hvor man, ved å følge pilsenes retning, kan bli ”fanget” i ”sinken” og aldri komme ut av den. Dessverre er slike situasjoner ofte tilfellet, og det er derfor viktig å finne et botemiddel, eller ”sink remedy” for dette. Googles botemiddel er en metode som heter ”random surfer”. Denne metoden går ut på at det er en viss sannsynlighet for at en Internett-surfer vil hoppe fra en side til en hvilken som helst tilfeldig annen siden på Internett (dokumentsamlingen). Indirekte har Google løst problemet med ”sinks” på denne måten ved å legge til ”svake” lenker mellom alle dokumentene i samlingen. Dette er N^2 ekstra lenker. Med ”svake lenker” mener vi at man i N * N lenkematrisen ikke bruker 0 som tegn på at en lenke ikke eksisterer ved analyse start, men en relativt lav verdi (botemiddel verdien). Google sier de brukte 0.15 i begynnelsen (Brin and Page 1998).

Ved å gjøre det slik inkluderer ”random surfer” metoden lenker mellom dokumentene helt uten hensyn til hvor viktig dokumentene er ansett av den totale grafen (de egentlige lenkene mellom dokumentene). Sagt på en annen måte er ”random surfer” omtrent det samme som en likegyldig S_h matrise (like verdier for alle dokumentpar). Istedenfor å betrakte alle dokumentene som like når man legger til botemiddelet, vil vi påstå at det er bedre om man kan variere verdien man velger å legge til etter et kriterium som gjenspeiler noe informasjon mellom dokumentene. Dette kriteriet kan være verdien et dokumentpar har fått ved likhetsberegning. Det virker intuitivt som et mer riktig og rettferdig botemiddel i stedet for en felles verdi for alle dokumentene som i random surfer metoden. I de par hvor likheten er 0, kan man isteden bruke en felles, lav verdi R som for eksempel 0.15. Dette gir følgende:

$H_h + R + S_s = H_h' + S_s$.

Under er et eksempel på likhetsberegninger byttet ut med random surfer teorien:

Eksempel på sink botemiddelmatrise i følge "random surfer".

Dok	1	2	3	N
1	X	0.02	0.02	0.02
2	0.02	X	0.02	0.02
3	0.02	0.02	X	0.02
N	0.02	0.02	0.02	X

Tilfeldig Surfer

Eksempel på sink botemiddelmatrise basert på likhet mellom dokumentpar. Tilfellene med likhet like 0 blir tatt hånd om av *R* matrisa.

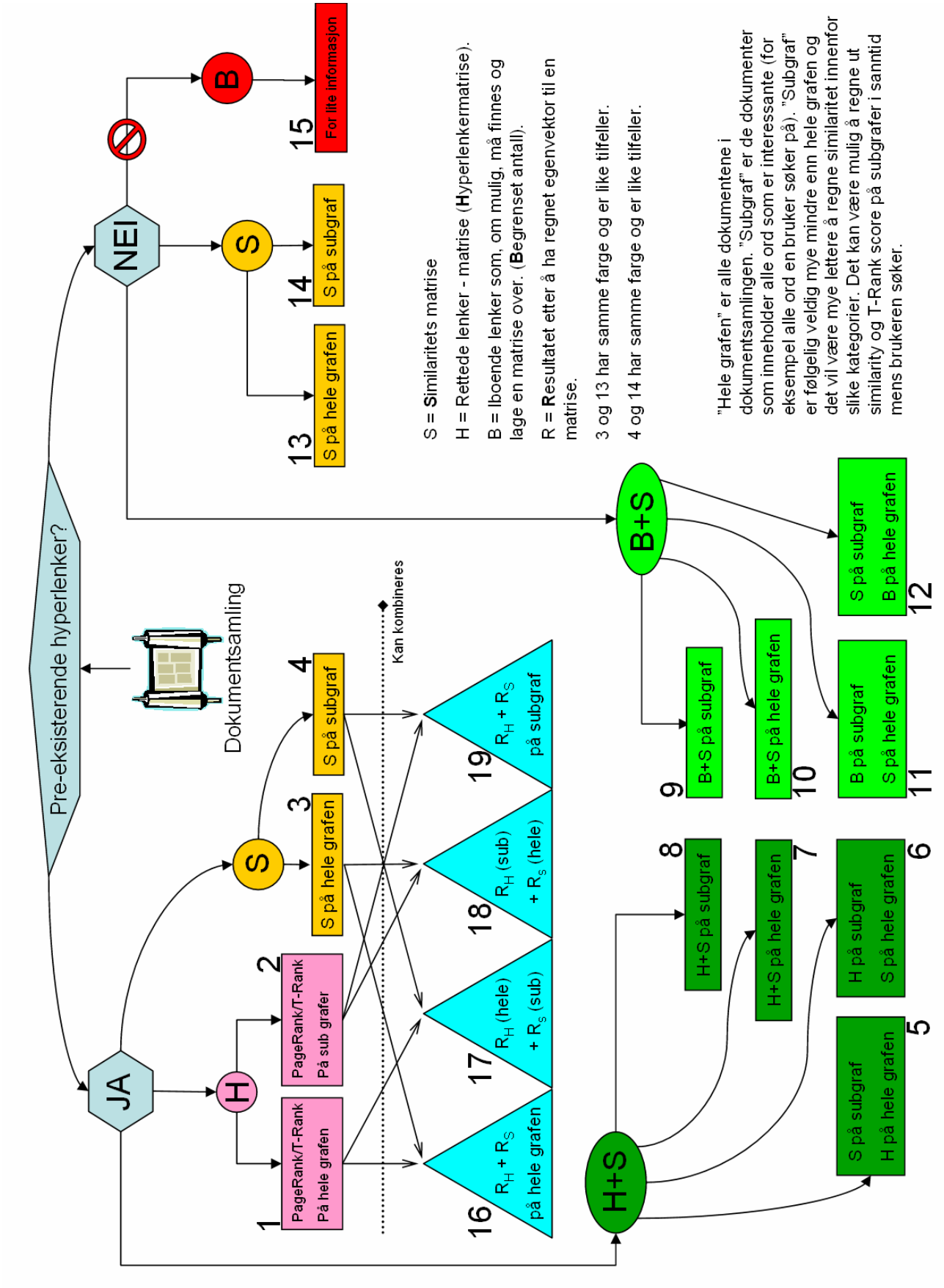
Dok	1	2	3	N
1	X	0.21	0.14	0.33
2	0.21	X	0.05	0.56
3	0.14	0.05	X	0.1
n	0.33	0.56	0.1	X

S_s, (kan også være S_h)

Tabell 4 - Eksempel å sinkbotemiddel ved bruk av "tilfeldig surfer" metoden versus "Similarity" metoden.

5.3 Mulige kombinasjoner av lenkeanalyse og analyse av dokumentlikheter

Basert på de mulighetene vi mener similaritetsmatrisa S og anbefalingsmatrisa H gir, har vi under satt opp en figur for å illustrere disse. Diagrammet er ment som en oversikt over alle mulighetene og inneholder derfor noen operasjoner som ikke er ønskelige å se nærmere på. Vi ønsker å presisere at likhetsberegning mellom alle dokumentene i en samling fort kan bli ulønnsomt om vi ikke finner en metode for å unngå å beregne alle de lave likhetene som typisk vil være i en stor samling. Dette er med på å nedfavorisere tilfeller hvor det ellers ville vært interessant å se på metoder som inkluderer matrisa S_h. I kapittel 4 diskuteres det rundt forslag for å beregne likhet mellom mange dokumenter raskt. I denne oppgaven vil dokumentsamlingen vi ser på være relativt liten i forhold til hva den kan være i praksis, så vi vil kanskje se på enkelte av metodene som innebærer å beregne S_h, men vi vil definitivt se på S_s.



Figur 13 - Oversikt over forskjellige forslag til metoder og ideer som kan brukes for å beregne viktighetspoeng for dokumenter med god eller dårlig lenkestruktur.

5.4 Forklaring til Figur 13

Man kan se på de runde og ovale boksene i figuren som databaser som inneholder informasjon på matriseform om relasjonene mellom dokumentene i samlingen. Disse matrisene er kvadratiske. I tilfellet hvor vi har N dokumenter vil matrisene være av størrelse N x N.

Man kan se på de firkantede beholderne i figuren som metoder for å evaluere matrisene. Måten vi evaluerer matrisene på er slik at vi får en verdi for hver node i matrisen. De lyse og mørkegrønne boksene er *addisjon av matriser*. Det er til forskjell fra de blå trekantene som er *addisjon av resultatene av metodene anvendt på matrisene*.

"**S**" betyr likhetsberegning (similaritet) på *par med dokumenter* i samlingen. S er med andre ord en matrise med likhetsverdier mellom dokumentpar. Man kan se på S som en informasjonskilde eller database som sier noe om hvert dokuments likhet med alle de andre dokumentene i samlingen. S er forventet å fungere best om man ser på subgrafer som omhandler et tema.

"**H**" betyr analyse av lenker i dokumentene (hyperlenker). H er med andre ord en matrise over pekere mellom dokumentene. Man kan se på H som en informasjonskilde eller database som sier noe om hvilke dokumenter som peker på, eller anbefaler, et gitt dokument. H er best om man ser på hele grafen fordi man da får mest mulig informasjon til å basere analysen på. I motsetning til S vil H fungere fint uavhengig av om man ser på et spesielt tema. Det er med andre ord uvesentlig for beregning av viktighet basert kun på lenkeanalyse om vi tar i betraktning lenker funnet i dokumenter som ikke handler om det samme.

Både S og H kan legge til rette for to forskjellige typer evaluering av en matrise M over dokumentsamlingen hver for seg. Den ene evalueringsmetoden er analyse av den totale matrisa M over hele dokumentsamlingen. Den andre evalueringsmetoden er å analysere en *del* av matrisa M, en subgraf, for å få et sterkere fokus på et gitt tema. Et eksempel på dette er om man først velger seg ut dokumenter som man vet handler om et visst tema, for så å lage en matrise over disse dokumentene og evaluere denne matrisa.

"**B**" betyr analyse av et begrenset antall lenker i dokumentene funnet ved å finne måter å lage disse lenkene automatisk eller manuelt. Hvordan man skal finne disse lenkene kan variere avhengig av hva slags dokumenter man ser på og domenet de er i. Det er egentlig bare fantasien og domenet dokumentene er i som setter grenser. Eksempler på "B":

- Et svar på en Email i Outlook kan sees på som en peker fra svar-mailen til den mailen det ble svart på.
- Brukeren kan på forskjellige måter relatere dokumenter med hverandre manuelt.
- I mail med vedlegg kan man se på mailen som en peker til vedlegget.
- Man kan analysere hvordan brukeren organiserer dokumentene på maskinen sin, og finne et begrenset og spredt antall lenker på denne måten.

Et viktig poeng er at vi har et begrenset antall lenker å basere lenkeanalysen på, hvis ikke befinner vi oss på "JA" siden i figuren. Derfor er vi avhengige av å supplere B med S, hvor S er to forskjellige måter å foreta likhetsberegning på.

Når vi legger sammen to matriser, for eksempel S og H kan disse to representere grafen i helt forskjellig størrelsesorden. Derfor er det viktig å vekte matrisene slik at denne størrelsesforskjellen blir utjevnet. Man kan også, av forskjellige grunner, ville vekte de forskjellige matrisene slik at den ene veier tyngre enn den andre. En mer korrekt matematisk skrivemåte som tar i betraktning denne vektingen ville vært: $M = \alpha S + \beta H$, hvor α og β er verdier for å redusere eller øke henholdsvis matrise S eller H, mens M er resultantmatrisa. Det samme gjelder når vi legger sammen egenvektorer funnet ved beregning på en likhetsmatrise og en lenkematrise (for R i figuren). Det kan da være ønskelig å ville vekte den ene egenvektoren mer enn den andre. Uttrykket ville i dette tilfellet blitt slik: $Score = \alpha R_H + \beta R_S$, hvor αR_H er skalert egenvektor funnet ved å regne på lenkematrisa H, og βR_S er skalert egenvektor funnet ved å regne på similaritetsmatrisa S. Vi har ikke brukt denne skrivemåten i figuren fordi den skal være så oversiktlig og enkel som mulig og bare vise mulighetene.
Matriseaddisjon foretars i boksene 5 til 12.
Addisjon av to egenvektorer foretarsi trekantene 16 til 19.
Trekantene 16 – 19 og boksene 5 – 12 er to forskjellige måter som er ganske like i tankegang og utførelse, men som sannsynligvis vil gi forskjellige resultater.

Under, i avsnittene 5.4.1 og 5.4.2 beskriver vi hver av komponentene i Figur 13 steg for steg. Når vi beskriver en komponent direkte, brukers innrykk. Når vi skriver noe om komponenter generelt eller samlinger av komponenter brukers ikke innrykk.

I forklaringene under, og i resten av rapporten, gjelder følgende definisjoner:
lsdb: Lenkestrukturdatabase.
lsdb_sub: subgraf av lsdb.
lsdb_sparse: Lenkestrukturdatabasen med mange tilfeldige lenker slettet.
simdb: Similaritetsdatabase (oversikt over likheter mellom dokumentpar i samlingen).
simdb_sub: subgraf av simdb.

5.4.1 På "JA"-siden av figuren

5.4.1.1 Rosa

1: Dette er tilfellet hvor vi bare kjører en eller annen form for lenkeanalyse over alle lenkene i alle dokumentene i hele dokumentsamlingen. Dette er gjort før og vi vet det fungerer bra, bare man har nok lenker. Denne metoden kan være interessant å utføre, bare for å ha noe å sammenligne de andre metodene med. Resultatene denne metoden gir er også interessante for å bruke sammen med andre metoder, for eksempel de blå trekantene 16 og 17, for å se om vi oppnår bedre resultater eller et annet resultat som ser logisk og brukbart ut.

2: Dette er tilfellet hvor vi bare kjører en eller annen form for lenkeanalyse over alle lenkene i et *utvalg* av dokumentene i hele dokumentsamlingen. Dette kan være nyttig i de tilfeller hvor det er ønskelig å se på lenkestrukturen i en del av den totale grafen (en subgraf).

5.4.1.2 Oransje

De oransje boksene i figuren er det samme for dokumenter med eller uten hyperlenker (i "JA" eller "NEI" siden av figuren). De går kun ut på likhetsberegning mellom dokumentene i samlingen på to forskjellige måter. Den ene av disse måtene (i figuren 3 og 13) er å regne likhet mellom alle dokumentene i samlingen (hele grafen). Den andre måten (i figuren 4 og 14) er å regne likhet mellom et utvalg av dokumentene i samlingen som vi vet handler om minst ett spesielt tema (subgraf). Legg merke til at 3, 4, 13 og 14 har samme farge.

3: Dette er tilfellet hvor vi regner likhet mellom alle dokumentene i hele dokumentsamlingen. Dette vil si noe om hvor likt et dokument er hele samlingen. Med andre ord forteller det hvor mye / om hvert dokument skiller seg ut eller ikke fra den totale samlingen. Dette kan være den første og enkleste måten å prøve å etablere et svakt analyseringsgrunnlag på, blant dokumenter i en samling uten lenker. Det er viktig å merke seg at dette tilfellet kan være vanskelig å gjennomføre i praksis med mange dokumenter ettersom vi må regne likhet mellom alle dokumentene i samlingen. Vi har for øvrig kommet med forslag til hvordan dette kan gjøres i kapittel 4.6.

4: Dette tilfellet er mer interessant enn tilfelle 3. Det fordi vi i dette tilfellet vil regne likhet mellom alle de dokumenter som omhandler et gitt tema. For eksempel vil vi for søkeordet ”Trondheim” først plukke ut de dokumentene som inneholder ordet ”Trondheim” og så finne likheten mellom alle disse dokumentene. Dette vil fortelle oss noe om hvor mye det aktuelle dokumentet handler om ”Trondheim” i forhold til de andre dokumentene i samlingen som også, i større eller mindre grad, handler om ”Trondheim”.

Alle beregninger i de grønne boksene er addisjonsoperasjoner på matriser.

5.4.1.3 Mørkegrønn

Alle de mørkegrønne metodene er tradisjonell linkanalyse på en asymmetrisk lenkematrise som inneholder mer informasjon enn hvilke dokumenter de peker til. Denne ekstra informasjonen er similariteter (likheter) mellom dokumentene på forskjellige måter. $S_{i,j}$ = similaritet(i,j). $H_{i,j}$ = eksisterer en lenker mellom dok(i,j).

5: Dette tilfellet tar i betraktning likhetsprinsippet mellom de dokumenter som omhandler et visst tema (som beskrevet i punkt 4), i tillegg til å ta i betraktning viktigheten dokumentene er gitt ved hyperlenkeanalyse av den totale grafen. Vi får på denne måten brukt hele samlingen til å si noe om viktigheten til de aktuelle dokumentene samtidig som vi foretar likhetsberegning mellom de interessante dokumentene. På denne måten har vi to verdier, S_{ij} og $H_{ij,}$ for hvert dokumentpar, som vi kan bruke som utgangspunkt til å beregne dokumentpoeng. S vet vi er symmetrisk, og H er rettet. Ettersom vi til sammen har to verdier, en i matrisa S og en i matrisa H, for alle dokumentpar i samlingen kan vi addere S og H sammen og få en alle til alle matrise M. Resultantmatrisa M vil være usymmetrisk og er

det utgangspunktet vi bruker for å beregne en score for hvert enkelt dokument. Disse dokumentscorene er basert på mer bakgrunnsinformasjon enn ved vanlig lenkeanalyse og vil forhåpentligvis være et bedre mål på viktigheten til dokumentene.

Det er for øvrig viktig å være klar over at dette er den mest beregningskrevende metoden av alle forslagene. Dette fordi vi blir nødt til å kjøre T-Rank over en komplett matrise over alle dokumentene i grafen for hvert eneste søk.

Det er viktig å forstå at en matrise på en subgraf vil være mindre enn en matrise over den totale grafen, og at det er umulig å addere sammen to matriser av ulik størrelse. Dette kan løses ved å lage den minste matrisa like stor som den største ved å legge inn verdien 0 i alle de tilfeller som ikke har noen verdi når man utvider den minste matrisa. Matematisk sett kan det gi noe merkelige svar å foreta en slik addisjons operasjon for så å regne egenvektoren til dokumentene. Dette fordi lenkeanalyseberegningen starter på en matrise som vil ha detaljerte verdier et konsentrert sted i matrisa og mindre detaljerte verdier i resten av matrisa. Det er addisjonen av den mindre similaritetsmatrisa som er skyld i de mer konsentrert detaljerte verdiene. For mer matematisk informasjon rundt dette, se forklaringen i kapittel 5.1 – ”*Tanker rundt matriseregning*”.

6: Dette tilfellet er antagelig ganske ubrukelig. Det kan være nødvendig å se på hyperlenkestruktur mellom dokumenter i en subgraf, men det vil sannsynligvis ikke gi noen mening å se på dette sammen med similaritet mellom alle dokumentene i hele grafen uten et spesielt tema.

7: H+S på hele grafen kan være interessant å se resultatene av for å sammenligne med resultatet fra 5. Dette tilfellet er rett og slett vanlig lenkeanalyse av hyperlenkene i samlingen, men også tatt i betraktning hvert dokuments likhet med alle de andre dokumentene i samlingen. På denne måten får vi en graf-oversikt som er annerledes enn vanlig linkanalyse på den måten at den inneholder mer informasjon. Om denne ekstra informasjonen vil hjelpe oss å kunne beregne hvert enkelt dokuments viktighet *bedre* enn tradisjonell lenkeanalyse alene er usikkert. Det kan være interessant å sammenligne resultatet med andre forsøk vi skal foreta.

8 → H_s+S_s: Dette tilfellet ser på hyperlenkestrukturen blant et utvalg av dokumentene i samlingen i tillegg til likheten mellom disse dokumentene. Denne metoden er relativt lik metode 5, men gir dårligere / mer upresise svar enn metode 5 fordi den bare tar i betraktning deler av samlingen for å finne de rettede lenkene. På den annen side er den forventet å være mye raskere fordi vi i dette tilfellet kun har med en subgraf å gjøre når vi skal starte å kjøre T-Rank.

5.4.1.4 Blå

Felles for alle de blå trekantene er at det foretas to samtidige egenvektorutregninger på to forskjellige matriser. Den ene av disse matrisene er S (likhetsmatrise), den andre er H (lenkematrise). Resultatet fra hver beregning, som er en egenvektor for hver node i grafen (hvert dokument i samlingen) adderes så sammen til å bli den endelige scoren for dokumentene i samlingen. Alle operasjonene i de blå trekantene er forventet å være raskere enn metodene 5-12 fordi de fleste tilfellene her er kombinasjoner av allerede ferdig beregnede verdier, og kun små sanntidsberegninger.

16 → **R_{Hh} + R_{Sh}**: Dette er tilfellet hvor vi regner ut en score for hvert dokument basert på tradisjonell lenkeanalyse over hele grafen. Til dette skal vi bruke T-Rank. Samtidig regner vi ut en score for hvert dokument basert på likhetsmatrisa over alle dokumentene. Vi kan for eksempel kjøre T-Rank på denne likhetsmatrisa siden T-Rank til en viss grad gir meningsfulle resultater på en symmetrisk graf. T-Rank vil på en symmetrisk graf gi som resultat EVC, eller Eigenvector Centrality. PageRank derimot vil, på en symmetrisk graf, gi relativt uinteressante resultater, nemlig ”node degree”. Dette er det samme som å simpelthen telle antall lenker inn til en side og bruke dette som score for siden (se kapittel 2.2 for mer informasjon rundt dette). Resultatet av disse to parallelle kjøringene er to verdier for hvert dokument. Disse to verdiene adderer vi så sammen til en endelig verdi som er den ferdige scoren til dokumentene.

17 → **$R_{Hh} + R_{Ss}$**: I dette tilfellet regner vi tradisjonell lenkeanalyse med T-Rank på den asymmetriske matrise over lenkene i grafen. Dette gir oss en score for hvert dokument. Samtidig regner vi ut en score for hvert dokument i en subgraf basert på likheten mellom dokumentene i subgrafen. Dokumentene i subgrafen plukkes ut etter et tema relatert til søkeforespørselen fra brukeren. Dette vil gi oss en score for hvert dokument i subgrafen. Resultatet vil være to scores for hvert dokument i subgrafen fordi dokumentene i subgrafen også har fått en score etter lenkeanalyseringen av den totale grafen. Disse to scorene adderer vi sammen til den endelige verdien som skal representere dokumentene's viktighet.

Likhetsberegning mellom dokumenter i en subgraf antar vi å være en relativt rask beregning i forhold til å beregne likhet mellom alle dokumentene i en komplett graf (alle dokumentene i et domene). Om antallet dokumenter i subgrafen skulle bli høyere enn et visst antall, slik at beregningen ikke lenger er lønnsom, kan metoden i slike tilfeller droppes / byttes ut med en annen metode. R_H tar i betraktning hele grafen basert på lenkeanalyse, R_S en subgraf basert på likhet mellom dokumentene. Dette er ganske likt tilfelle 5, og det virker interessant å sammenligne resultatet fra 17 med det fra 5 og 8. Basert på denne analysen av tilfelle 17, er denne metoden veldig interessant å se videre på.

18 → **$R_{Hs} + R_{Sh}$**: Dette tilfellet er på en måte motsatt av tilfelle 17, følgelig er ønsket om å ta med 18 i den videre utviklingen også motsatt av konklusjonen ved 17. I dette tilfellet settes til krav å måtte regne likhet mellom alle dokumentene i samlingen, noe som gir både uvisse resultater og er tidskrevende. Denne metoden er antagelig dårlig og vil produsere ulogiske resultater, og den er kun med for oversiktens skyld.

19 → **$R_{Hs} + R_{Ss}$**: I dette tilfellet må vi regne likhet mellom dokumenter på en subgraf. Dette kan gi gode resultater i seg selv. I tillegg ser vi på lenkeanalyse i en subgraf, noe som gir gode eller dårlige resultater avhengig av hva slags metode som brukes for å analysere subgrafen med lenker. PageRank er laget med hensikt å ta en hel graf i betraktning, ikke bare en subgraf, og er derfor antagelig forventet å gi best resultater om man bruker den på hele grafen, og ikke en subgraf. Når det gjelder T-Rank er ikke den testet på bare en subgraf ennå, det er mulig den vil gi gode og logiske resultater, men det er usikkert. Vi vil derfor påstå at dette tilfellet er litt interessant å se nærmere på.

5.4.2 På "NEI"-siden av figuren

5.4.2.1 Lysegrønn

Felles for de lysegrønne boksene er at de alle ser på dokumenter med en dårlig lenkestruktur i seg. Det vil derfor være viktig å, i tillegg til å analysere lsdb-sparse, ta ideen om similaritet i betraktningen for å få gode resultater. Det er forutsatt at lenkestrukturen i samlingen er så dårlig at den ikke er sammenhengende og følgelig tror vi ikke det vil gi mening å bruke lsdb alene på hele grafen, i så fall befinner vi oss på "JA"-siden i Figur 13. B på en subgraf vil antagelig gi mest meningsfylte resultater ettersom man får konsentrert fokuset av analysen. Det har seg derfor slik at det er mest ønskelig å se nærmere på de tilfeller hvor S foretas på en subgraf og B på en graf med størrelse mindre enn, eller like S. Til sammen vil en slik kombinasjon antagelig gi enda bedre resultater enn hver for seg. Fordi B er så usammenhengende at det ikke er forventet å få logiske resultater basert på den alene, synes vi heller ikke den bør brukes på grafer som er større en S. Kort sagt synes vi det gir mest mening om B brukes til å supplere S i alle de tilfeller hvor anvendelse av S er nyttig i seg selv[24].

I kapittel 1 diskuterer vi rundt forslag for hvordan det muligens kan gjøres å regne likhet mellom alle dokumentene som er forventet å ha en viss likhet raskt. Hvis dette lar seg gjøre vil evalueringen av de lysegrønne boksene kanskje bli noe annerledes. Men *S* er allikevel ikke forventet å gi så veldig meningsfylte resultater på hele grafen da det bare sier noe om hvor mye eller lite gjennomsnittlig et dokument er. I avsnittene under har vi for ordensskyld skrevet matriseoperasjonen for hvert tilfelle etter pilen.

> **$9 \rightarrow (B+S)_s$**: Dette tilfellet er veldig likt tilfelle 8, med den forskjellen at "B" betyr en begrenset lenkestruktur. Denne metoden vil derfor antagelig ha god bruk for likhetsberegningene i subgrafen ettersom S på en subgraf er forventet å gi mening. Tatt i betraktning at B er usammenhengende for hele grafen, men er forventet å gi mening på en subgraf og kan derfor gi nyttig tilleggsinfo til S_s, og at S fungerer godt som hjelp til å beregne viktighetspoeng i en subgraf, vil vi påstå at denne metoden er interessant å se nærmere på.

[24] Til tross for dette vil vi antagelig se nærmere på tilfellet $B_h + S_s$ ettersom tilfellet omtrent gir seg selv når det gjelder programmeringsarbeide. Dette fordi det kun er en kombinasjon av to andre resultater vi uansett skal regne.

10 → $(B+S)_h$: Dette tilfellet er veldig likt tilfelle 7, med den forskjellen at "B" betyr en begrenset lenkestruktur og er forventet å fungere dårlig på hele grafen. Denne metoden vil antagelig har god bruk for likhetsberegningene mellom alle dokumentpar. Ettersom B er forventet å være usammenhengende, og S for hele grafen logisk ikke gir noe mening vil vi påstå denne metoden vil fungere dårlig. Men siden analysen gjelder hele grafen, er dette en metode som kan gjøres off-line *en* gang og brukes for alle typer søkeord senere. Dette gjør metoden litt interessant å se nærmere på.

11 → $B_s + S_h$: Denne metoden er kun med for oversiktens skyld. B *kan* gi mening på en subgraf, men ikke nok til å være nyttig. I tillegg å ta i betraktning S på hele grafen, som for øvrig er ulønnsomt og ikke er forventet å gi veldig mye mening, gjør ikke situasjonen noe bedre. Denne metoden er ikke interessant å se noe nærmere på.

12 → $B_h + S_s$: Denne metoden er hovedsakelig med for oversiktens skyld. Similaritet på en subgraf er forventet å være ok. B på hele grafen er forventet å ikke gi noe mening i seg selv fordi lenkene ikke er sammenhengende. At lenkene ikke er sammenhengende betyr at de er for spredt til å alene kunne gi nok informasjon til å bruke lenkeanalyse til noe fornuftig. Egentlig er ikke denne metoden interessant å se noe nærmere på, men siden vi uansett vil implementere koder for å regne Ss og Bh, vil det ikke være mye ekstra arbeid å legge til rette for dette tilfellet. Se for øvrig fotnote 24.

13 → S_h: Lik metode som 3. Disse oransje boksene er kun med for oversiktens skyld.

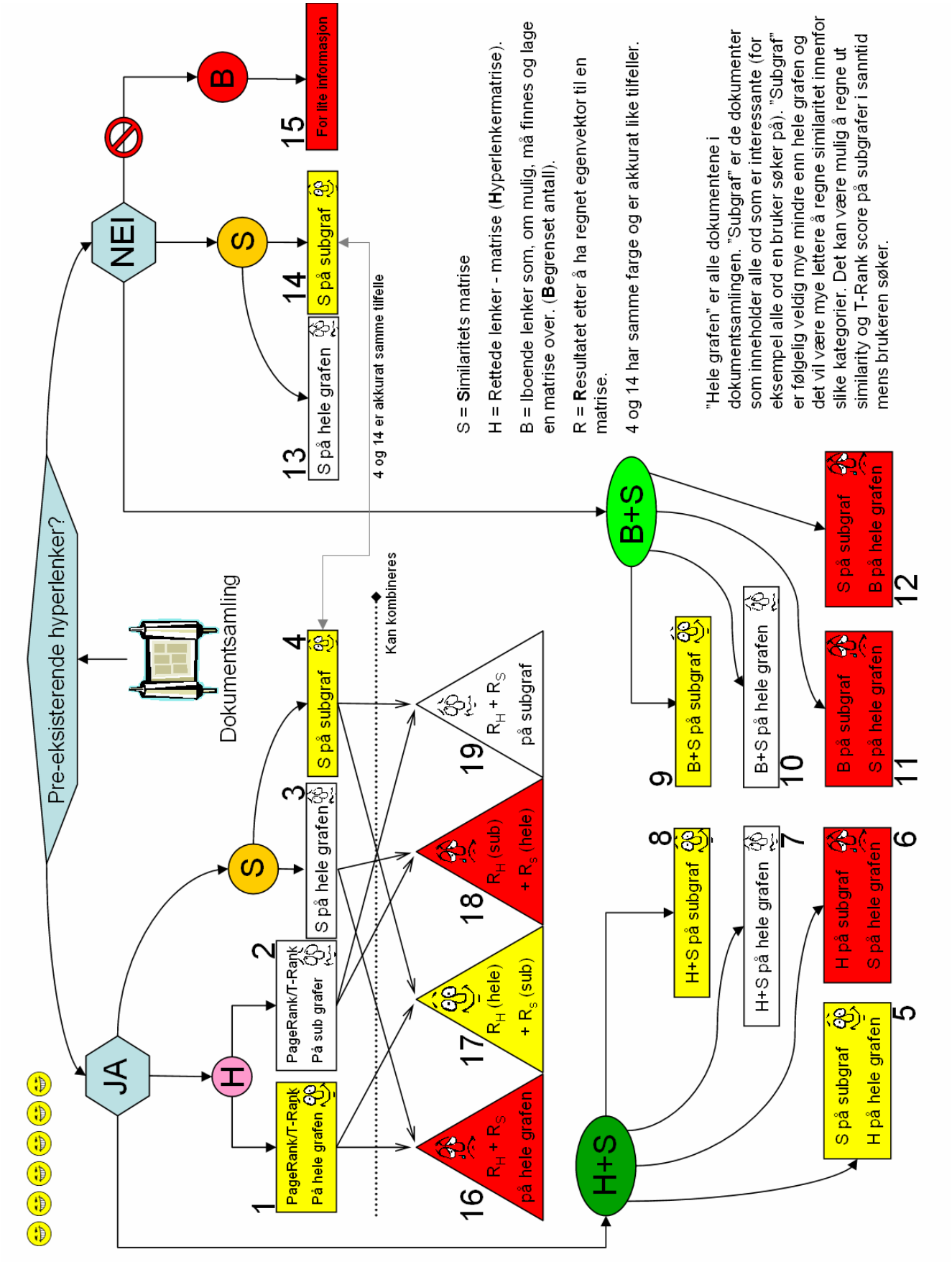
14 → S_s: Lik metode som 4. Disse oransje boksene er kun med for oversiktens skyld.

15 → B: Å bare se på de automatisk eller manuelt innlagte lenkene vil ikke være nok til å få noe meningsfylt informasjon ut av. Derfor har vi kun tatt med denne boksen for oversiktens skyld.

5.5 Konklusjon på mulighetene valgene gir

Metodene vi har for å kombinere likhet mellom dokumentpar med lenkeanalyse vil avhenge av hvor raske metodene for å beregne likhet er. Hvis metoden for å beregne likhet mellom alle mulige par i samlingen viser seg å fungere dårlig, betyr dette at vi må begrense oss til å bruke likhetsprinsippet kun på subgrafer.

Med grunnlag i vurderingene over i avsnitt 5.4 kan oversiktsfiguren over de forskjellige forslagene tegnes på nytt, med de aktuelle og interessante metodene og ideene mer klart. Basert på argumentene og resonneringene ovenfor konkluderer vi i Figur 14 med at det er mest interessant å se nærmere på de boksene som er markert gule. De boksene som er markert hvite er også interessante, men ikke så interessante som de gule. Vi vil, ved å se nærmere på de hvite og gule boksene, få et inntrykk av hvordan similaritetsberegninger fungerer brukt med og uten å ta en rettet lenkestruktur i betraktning.



Figur 14 - Oversikt over de metoder og ideer vi mener det er interessant å se nærmere på (markert i gult og hvitt)

6 Valg av dokumentsamling

For å teste metodene er det viktig å ha et sett med dokumenter som tilfredsstiller visse kriterier. Kriteriene vil være med på å gjøre testingen så enkel og korrekt som mulig. Grunnen til at det er så viktig å legge en del arbeide i å velge riktig testsett er hovedsakelig fordi testing på et mest mulig realistisk datasett med uavhengige forslag til svar eller rangering vil være med på å beskrive hvor godt målene og delmålene i oppgaven er nådd. En vanskeliggjørende faktor i et arbeid som gjøres i denne oppgaven er at vi ikke vil være helt sikker på hvor godt metodene fungerer før det faktisk er satt ut i praksis i et ekte scenario. Men det vil være helt umulig å bedrive testing i ekte scenarioer fordi det vil ta for lang tid. En av fordelene ved å velge testsett er at det gir oss mulighet til å teste metodene på kort tid. Denne korte tiden vil antagelig gå på bekostning av hvor nøyaktige og pålitelige resultatene blir. Vi må derfor velge et testdomene som er så likt som mulig et ekte scenario, og velge størrelse ut i fra hvor mye tid vi har til rådighet.

I dette kapittelet starter vi med å gi leseren noe praktisk informasjon rundt fenomenet spam. Deretter definerer vi forskjellige typer lenkestrukturer. Det er viktig å ha klart for seg de forskjellige typene lenker et dokument kan inneholde for å være i stand til å velge et best mulig passende testsett. I avsnitt 6.3 diskuteres det en del rundt noen kriterier og krav vi har til dokumentsettet vi ønsker å bruke. Dette avsnittet etterfølges av et avsnitt hvor vi nevner noen kandidater for dokumentsett som kan passe til disse kravene. Disse er alle et resultat av tidkrevende undersøkelser i vår søken etter et passende sett. I slutten av dette avsnittet listes alle mulige kandidater funnet i en tabell med oversikt over fordeler og ulemper for hver av kandidatsettene. Så følger et avsnitt med diskusjon og evaluering hver av kandidatene nevnt, før vi i siste avsnitt konkluderer med det settet vi mener passer best for vårt formål.

6.1 Spam

6.1.1 Redundant informasjon

En av hovedfordelene ved lenkeanalyse generelt, men spesielt på Internett, er den potensielle muligheten det gir for å fjerne spam. Sider eller dokumenter som ikke tilfører datasettet noen ny eller nyttig informasjon anser vi som spam, og dette er et økende problem på Internett i dag (Yaltaghian and Chignell; Langville and Meyer 2004). Vi vil anta at similaritetsmetodene utarbeidet i dette prosjektet vil fungere dårlig i datasett inneholdende spam[25]. Dette fordi tilfeller med mange like dokumenter vil score høyt i similairtetsmodellene. Men sammen med lenkeanalyse, og med riktig vektlegging av similaritetsanalyse kontra lenkeanalyse, kan det kanskje fungere godt. I domener som inneholder spam, vil vi derfor påstå at lenkeanalyse er et viktig botemiddel. Metodene utarbeidet i denne rapporten er ment som et supplement til tradisjonell lenkeanalyse, og er beregnet for bruk i domener inneholdende få eller ingen lenker. Vi vil tro at spam i slike domener ikke er et stort problem, og følgelig er ikke spam noe vi vil ta hensyn til. Spam er et fenomen som oppstod på Internett etter at Internett vokste seg stort (Langville and Meyer 2004), og følgelig vil man ha lenker som middel til å bote mot dette. I domener uten lenker, og i domener hvor metodene i denne rapporten hovedsakelig er ment for bruk, er sannsynligvis spamproblemet ikke tilstedeværende.

6.1.2 Link farming

Med ”link farming” tenker vi spesielt på et fenomen som har oppstått på Internett hvor enkelte sider har fått mange lenker pekende mot seg i forsøk på å øke sin score som sett av tradisjonell lenkeanalyse. Vi vil påstå at en kombinasjon av tradisjonell lenkeanalyse og similaritetsanalyse innenfor et visst tema vil fungere godt som botemiddel mot link farming. Måten vi tenker oss dette på er at vi, istedenfor å sette verdi 1 mellom dokumenter hvor man finner lenker, kan vi bruke verdien som gjenspeiler hvor like dokumentene er. Eller vi kan addere lenkeanalyse med similaritetsanalyse og vektlegge disse hensiktsmessig. Dokumenter som har flere ikke-relaterte andre dokumenter med lenker pekende mot seg vil i slike tilfeller score lavere enn dokumenter som har relaterte andre dokumenter pekende mot seg, og følgelig kan de nedprioriteres i rangeringen av resultatet.

[25] Vi tror at likhetsanalyse alene vil være veldig utsatt for å bli missbrukt og manipulert av spam om det skulle brukes i en kommersiell internettsøkemotor.

6.2 Definering av typer lenkestruktur

Når man ser på hyperlenker på Internett er det vanlig å prøve å skille mellom navigasjonslenker og anbefalingslenker. I et forsøk på å finne slike lenker brukes ofte følgende definisjoner (Soboroff 2003):

1. **on-site in**: Dette er typisk navigasjonslenker. ”on-site” betyr at lenken er funnet i dokumentet man skal beregne score for. ”in” betyr at lenken ikke peker utenfor ”domenet” dokumentet befinner seg i.

2. **on-site out**: Dette er typisk anbefalingslenker. Når man analyserer et dokument side på Internett ligger dette ofte kategorisert i en mindre og sammenhengende samling (for eksempel et domenenavn). ”on-site” betyr at lenken befinner seg i det aktuelle domenet, mens ”out” betyr at lenken peker utenfor dette domenet.

3. **off-site in:** Dette er vanligvis anbefalingslenker som peker fra et eksternt domene og inn på den aktuelle siden i det aktuelle domenet man analyserer. Om et dokument har mange ”off-site in” pekere er dette det samme som ”link popularity”.

4. **off-site out**: Dette er et tilfelle man ikke er interessert i. Det er kun en abstrakt definisjon som følger logisk av de tre andre tilfellene. En ”off-site out” lenke er i realiteten en lenke som befinner seg utenfor dokumentet og domenet man analyserer, og lenken peker et annet sted utenfor domenet. Altså er ”off-site out” lenker aldri av interesse.

I vårt tilfelle, når vi skal skaffe et realistisk og passende datasett, ser vi for oss at vi kan bli nødt til å omdefinere disse vanlige begrepene av lenketyper. Avhengig av hvordan datasettet skaffes vil vi bli nødt til å manuelt analysere oss fram til hvilke typer lenker som er navigasjonslenker og hvilke som er anbefalingslenker. For eksempel vil dette variere fra definisjonene over om alle dokumentene som skaffes ligger innenfor samme domenenavn.

6.3 Kriteriene for et passende testsett

1. **Størrelse:**
   Størrelsen på dokumentsettet må ikke være så stort at det blir veldig tidkrevende å jobbe med. Samtidig er det viktig at det ikke er så lite at det gir upålitelige resultater. I små dokumentsamlinger vil urealistiske resultater antagelig oppstå fordi lenkestrukturen vil blir for dårlig samtidig som similaritetsberegninger mellom dokumentene ikke vil fungere som tenkt. En størrelse på mellom 10.000 – 100.000 dokumenter som inneholder mellom 1-10 sider med tekst og lenker antas å være passende.

2. **Hyperlenker:**
   Dokumentsettet må inneholde mange hyperlenker. Det er viktig at disse lenkene ikke er ment som vanlige navigasjonslenker, men lenker som anbefalinger om relaterte dokumenter. Det er også ønskelig at lenkene inneholder relatert deskriptiv ankertekst. Vi kan også slette mange tilfeldige lenker for å simulere et domene med dårlig lenkestruktur for å kunne teste NEI siden av "*Figur 13 - Oversikt over forskjellige forslag til metoder og ideer som kan brukes for å beregne viktighetspoeng for dokumenter med god eller dårlig lenkestruktur.*"

3. **Formatet til dokumentsettet**
   Det er ønskelig med dokumenter som er lette å parse og analysere. XML og HTML anses som bra dokumentformater. PDF er dårlig fordi det ikke er enkelt å analysere automatisk, og fordi PDF sjeldent inneholder mange hyperlenker.

4. **Pris**
   Gratis er best.

5. **Dokumentsettet bør komme med en uavhengig evaluering / fasit til hvordan dokumentene skal rangeres etter visse kriterier.**
   Denne fasiten vil være med på å fortelle oss hvor godt metodene vi har utviklet fungerer. Vi ser allikevel for oss at en slik fasit vil være sjelden å finne og derfor vanskelig å få tak i. I tillegg er vi litt skeptisk til en statisk[26] fasit løsning ettersom vi tror det vil være unaturlig å sammenligne med. En slik fasit må i så fall komme med et forslag til hvordan man skal *rangere* dokumentene i søkeresultatene, og ikke bare enten eller om de skal være med. Å sammenligne med en anbefaling som resultat av manuell analyse synes vi virker merkelig fordi vi da har å gjøre med hva andre mennesker synes, og ikke et realistisk scenario (resultat av andre godt etablerte rangeringsmetoder som for eksempel Google).

6. **Lenke Struktur Data Base (LSDB)**
   Om dokumentsettet allerede kommer med en oversikt over hvilke dokumenter som har lenker til hvilke andre og med hvilken ankertekst vil det spare oss mye arbeid. Dette er ikke noe absolutt krav, men det hadde vært fint.

7. **Samlingen skal ikke inneholde spam.**
   Om dokumentsettet innholder spam vil analysen av resultatene bli vanskelig. Resultatene vil bli sterkt påvirket av spam, og dette vil virke meget forstyrrende.

6.4 Kandidater som kan passe til disse kriteriene

Generelt om TREC:
I et forsøk på å hjelpe forskere med å evaluere arbeidene deres har ”National Institute of Standards and Technology” (NIST), ”Information Techonlogy Laboratory’s” (ITL) og ”Advanced Research and Development Activity” (ARDA) sponset en serie konferanser kalt ”Text Retrieval Conference” (TREC). Dette er en årlig konferanse i Maryland som går ut på å eksperimentere med store samlinger på flere millioner dokumenter. På disse konferansene kommer forskere fra hele verden sammen for å teste og sammenligne produktene sine (Voorhes and Harman 1999; Hawking, Voorhees et al. 2000). I forbindelse med disse konferansene er det laget dokumentsamlinger som er sett gjennom av eksperter som foreslår et hvert dokuments tema og relevans til enkelte søkeord / søkefraser. Disse dokumentsamlingene har blitt en standard for å måle tradisjonelle Information Retrieval metoder (relevans) opp mot. Det er laget fem forskjellige dokumentsamlinger som er tilgjengelige og distribueres kun til forskningsformål. Enkelte

[26] Med “statisk fasit” menes en fasit som kun sier om et dokument bør være en del av resultatet for et søk, og ikke hvilken plassering dokumentet bør ha i rangeringen. Vi ønsker en fasit som sier noe om hvordan dokumentene bør rangeres.

av dokumentsamlingene består av tidsskrifter og artikler, men de største er kopier av en crawl av en stor andel .gov domener. Disse crawle-samlingene inneholder lenker og blir behandlet (ryddet opp i) på forskjellige måter.

TREC workshoppene har tre hovedmål. 1: Å støtte forskning innen IR på store tekst samlinger. 2: Å øke kommunikasjon mellom industri, akademia og regjering ved å lage et åpent forum for forsknings ideer. 3: Å fremskynde overføringen av teknologi fra forskningslaboratorier til kommersielle produkter ved å vise betydelige forbedringer av metoder eller løsninger på virkelige problemer.

1. **TREC wt2g**
   ”wt2g” står for ”Web Track 2 gigabyte” og er den minste av TREC samlingene. Dette er ett sett web sider tatt fra en crawl foretatt av NIST i 1997. Både denne samlingen og wt10g er et ekstrakt av VLC2 (Very Large Collection 2, 100GB) samlingen. ”wt2g” samlingen har en relativt dårlig lenkestruktur.

2. **TREC wt10g**
   ”wt10g” står for ”Web Track 10 gigabyte” og har, som wt2g, en relativt dårlig lenkestruktur sammenlignet med flere av de andre alternativene i denne oversikten. Lenkene i alle TREC samlingene fungerer ikke godt som en etterlikning etter hvordan hyperlenkestrukturen på nettet generelt er (Soboroff 2002). Grunnen til dette er fordi de fleste lenkene er ”on-site”, eller navigasjonslenker i stedet for anbefalingslenker.

3. **TREC .GOV1**
   Denne samlingen het først bare ”.GOV”. Det er en crawl av .GOV domenet foretatt av NIST i 2002. Hovedformålet ved denne samlingen var at den skulle ha en størrelse som var håndterbar for testing, men fortsatt gjenspeile et realistisk domene.

4. **TREC .GOV2**
   Er det samme som ”.GOV1”, men laget i 2004 og er mye større.

5. **Wikipedia**
   Wikipedia er en gratis online leksikonløsning som er laget av flere hundre tusen frivillige, personer over hele verden. Hvert innlegg i samlingen inneholder *mange* lenker til andre relaterte innlegg.

6. **Amazon**
   Vi kan crawle internettsidene til Amazon.com. Hver bok, eller andre artikler som for eksempel CD'er kan vi se på som et dokument, hvor teksten ikke er teksten i boka, men en beskrivelse av hva boka handler om og annen informasjon rundt den (en anmeldelse av boka). Amazon gir hyperlenker som anbefalinger mellom de forskjellige anmeldelsene av produktene deres. Disse hyperlenkene er strukturert slik at en anmeldelse kan ha mange lenker til andre anmeldelser. Disse lenkene sier noe om likheten så vel som å være en anbefaling mellom produktene.

7. **Kleinberg Graf**
   Denne typen testsett ble brukt av Kleinberg (http://es.csiro.au/pubs/trecbook_for_website.pdf) for å teste hans egen HITS metode. Metoden er som følger: Man starter med de 100 – 200 beste resultatene for et søk rundt et spesielt tema i en kommersiell søkemotor. Dette settet er startsettet. Man tar så alle utlenkene i dette startsettet og putter dokumentene disse peker til inn i dokumentsettet. Deretter finner man alle innlenker (ved å søke spesielt i søkemotoren) til startsettet, og legger dokumentene som peker inn til dokumentsettet. Til slutt må man finne alle lenker mellom dokumentene i settet man har. Denne metoden har en del fordeler. Den er enkel på den måten at man bruker en kommersiell søkemotor for å finne datasettet. Dette datasettet er "tema basert", noe som passer denne oppgaven perfekt etter som vi skal se på likheter rundt forskjellige temaer. Deretter bruker man den samme kommersielle søkemotoren til å finne innlenker for hvert dokument i dette datasettet. Disse innlenkene sier noe om viktigheten til de forskjellige dokumentene i settet ettersom søkemotoren man bruker allerede har tatt hele Internett-grafen i betraktning for å finne denne viktigheten. Når man så finner alle lenkene mellom dokumentene i dokumentsette man har fått fra søkemotoren har man laget en "Kleinberg graf".

8. **Stanford WebBase**
   Dette er en database laget på "Stanford University". De har foretatt en crawl av Internett som er gratis for alle å bruke. http://www-diglib.stanford.edu/~testbed/doc2/WebBase/

Etter å ha foretatt en del "research" på de forskjellige kandidatsettene har vi kommet frem til følgende tabell med fordeler (gult) og ulemper (rødt):

Datasamling	Wt2g	Wt10g	.GOV1	.GOV2	Wikipedia	Amazon	Kleinberg	Stanford
Størrelse	250.000	1.7 mill	1.2 mill	25 mill	**Veldig BRA**, kan velge selv (500K engelsk, 89K Norsk) 	**BRA**, kan velge selv. 	1000 – 5000	STOR?
Lenkestruktur	**Dårlig**	**Dårlig**	OK / Dårlig	?	**Veldig BRA** 	**BRA**, avh. av størrelsen. Kommer med "Related items" 	Middels / Dårlig	**BRA**
Hvordan formatet til teksten i dokumentene leveres	**OK** Ascii 	**OK** Ascii 	**OK** Ascii 	**OK** Ascii 	**BRA** Tekst, MySQL. 	**OK** Oracle / XML-SOAP, avh av hvordan vi får tak i dataen. 	**BRA**, HTML 	OK
Pris	**$100**	**$100**	**$100**	**$500**	**Gratis** 	**Gratis** 	**Gratis** 	**Gratis** 
Uavhengig Evaluering av dokumentene	**JA minus** (binært). Dvs. ingen rangering bare utvalg av mer el mindre relaterte dok.	**JA minus** (binært). Dvs. ingen rangering bare utvalg av mer el mindre relaterte dok.	**JA minus** (binært). Dvs. ingen rangering bare utvalg av mer el mindre relaterte dok.	**JA minus** (binært). Dvs. ingen rangering bare utvalg av mer el mindre relaterte dok.	Ok. Google søkemotor resultat sammenligning.	**Antagelig BRA**, kommer med "Ratings" 	**BRA**, Søkemotor resultat sammenligning 	**NEI**
LSDB	**JA** 	**JA** 	**JA** 	**JA** 	Nei / Ja, vanskelig å få fatt i men grei å lage selv.	**NEI**	**NEI**	?
Brukerstøtte	Middels/ Dårlig	Middels/ Dårlig	Middels/ Dårlig	Middels / Dårlig	**GOD** 	OK / dårlig.	?	**Dårlig**
Spam	Nei	Nei	Nei	Nei	**Nei**	Nei	?	?

Tabell 5 - Oversikt over fordeler og ulemper over forskjellige dokumentsett som kandidater for bruk til testing.

6.5 Kommentarer til evalueringen av de forskjellige kandidatene

Generelt om TREC

Alle TREC samlingene har en ganske dårlig lenkestruktur (Hawking, Voorhees et al. 2000; Soboroff 2002), og de er derfor ikke bra nok for dette prosjektets formål. Prisen for å få tak i TREC samlingene er ikke et stort minus, men det er bedre å kunne få en gratis samling hvis de andre forutsetningene er like. Vi hadde forventet at TREC samlingene skulle komme med en ganske bra uavhengig evaluering, fordi vi har lest at IR eksperter har manuelt valgt ut dokumenter i fra de forskjellige samlingene og klassifisert disse som relevante til forskjellige temaer. Det kom derfor som en stor overraskelse at evalueringsresultatene til alle TREC samlingene er binære. Med ”binære” mener vi at svaret på om et dokument er relatert til et gitt tema eller ikke, er enten ”JA” eller ”NEI”. Svaret er altså ikke, som vi hadde forventet, en tallverdi som angir eller sier noe om likheten dokumentet har med de forskjellige temaene. Dette betyr at den uavhengige evalueringen som vi hadde på følelsen skulle være veldig bra for vårt formål egentlig ikke egner seg for å rangere dokumenter. Dokumentene er enten relatert eller ikke til et tema, og hvis et dokument er relatert så er det like relatert som alle de andre dokumentene som er relatert til det samme temaet.

TREC wt2g

Denne samlingen har passende størrelse, men har alle de negative kriteriene beskrevet over.

TREC wt10g

Denne samlingen har alle svakheter beskrevet over. I tillegg er det for stor for vårt formål.

TREC .GOV1

Denne samlingen er altfor stor for vårt formål.

TREC .GOV2

Altfor stor for vårt formål, og har alle de negative aspektene beskrevet over.

Wikipedia

Da vi sjekket ut wikipedia.org fikk vi en stor forkjærlighet for denne samlingen. Det har flere fordeler. Vi kan velge forskjellige typer størrelser. Wikipedia er et gratis online leksikon som har utgaver i forskjellige språk. De forskjellige språkene er av veldig forskjellige størrelse ettersom leksikonet lages av frivillige mennesker over hele verden. Den norske versjonen av leksikonet virker som en størrelse som passer bra for vårt formål. Den er på rundt 25.000 dokumenter (alle disse dokumentene er ikke interessante for dette prosjektet, men rundt 20.000 av dem). Hvert dokument er på mellom 1 – 10 sider og inneholder mange anbefalingslenker. For å få tak i et dokumentsett av en passende størrelse kan vi velge hvilket språk (hvilken database) vi skal laste ned. Denne måten å velge størrelse på datasettet har en fordel som kanskje ikke er så åpenbar. Databasen vi laster ned inneholder nemlig en komplett (og rik) lenkestruktur. Dette er en stor fordel fremfor å for eksempel laste ned deler av den enkelske versjonen av leksikonet fordi hele versjonen er for stor. Videre tilbyr wikipedia direkte og gratis nedlasting av de forskjellige databasene deres. Formatet til dataen i dokumentene man kan laste ned er vanlig tekst med en egen formattering rundt lenker. Wikipedia har en veldig rik lenkestruktur. Wikipedia tilbyr nedlasting i MySQL-format (som var ønskelig fra vår side), og har en viss brukerstøtte for MySQL, PHP og APACHE. Denne brukerstøtten har gitt tilfredsstillende svar på spørsmål vi har hatt. Et negativt aspekt ved wikipedia er at det er vanskelig å få tak i en LSDB. Men dette behøver ikke vær noe stort problem da vi uansett skal parse all teksten i dokumentene for å plukke ut og telle ordene. Det er ikke mye ekstra kode som skal til for å gjøre oss i stand til å plukke ut lenkene, ettersom datasettet kommer med en enkel og godt organisert formattering av lenkene i teksten. Alt i alt virker wikipedia som en veldig lovende kandidat, og dette til stor kontrast fra hva vi trodde før vi startet researchen på de forskjellige kandidatene.

Amazon

Dette virker også som en lovende kandidat. Det ser ut som om den har en rik lenkestruktur. Hovedfordelen til Amazon er at den har veldig bra uavhengige evalueringer og anbefalinger i form av ”Related Items”-lenker. Bortsett fra dette virker det som om brukerstøtten er ganske bra, og at formatet dokumentene ”kommer i” også er bra ettersom det er XML. Det negative med denne samlinge er at vi må crawle amazon sine sider for å få tak i selve dataen i dokumentene. De har nedlastbare databaser over lenkestrukturen, men selve teksten tilbys ved XML over deres internettsider. Vi kan altså ikke laste ned en komplett eller delvis database over anmeldelsene deres. Dette er fordi Amazon tilbyr denne tjenesten som et system for å selge varene sine gjennom andre sammarbeidspartnere over Internett. Det kan bli tungvint og unødvendig å crawle sidene til amazon synes vi, selv om det kan ha fordelen at vi kan velge størrelsen på settet på denne måten. Det fine met dokumentene i Amazon er at anmeldelsene kan hentes

kronologisk. Dette har den fordelen at dokumentene vi henter kun har lenker til tidligere anmeldelser, og vi vil følgelig ikke få døde lenker (som er tilfellet om vi henter anmeldelsene i for eksempel alfabetisk eller tilfeldig rekkefølge). Vi vet ikke hvilke kriterier Amazon legger til grunn for hvordan og hvorfor de legger ut lenkene sine.

Kleinberg-graf
Basert på erfaringer blant ansatte i Telenor R&D er det vanskelig å oppnå store generelle grafer, eller dokumentsett, ved bruk av denne metoden. Med ”generell” mener vi at grafen ikke er begrenset til et spesielt tema. Derfor blir denne løsningen innskrenket til kun å finne tema-baserte subgrafer. De har også erfart at lenkestrukturen i kleinberg-grafer ikke ligner mye på lenkestrukturen som generelt utvikler seg på Internett. Hovedstyrken til kleinberg graf metoden er at vi finner fort veldig mange relaterte sider, eller dokumenter, basert på ønsket tema ettersom søkemotoren (for eksempel Google) kan gi oss disse.

Stanford
Denne løsningen ser dårlig ut. Det eneste positive er at den er gratis og har en god lenkestruktur. Men siden den er så stor, blir vi nød til å lage metoder for å plukke ut dokumenter, noe som vil være vanskelig å gjøre uten å ødelegge mye av lenkestrukturen i samlingen.

6.6 Konklusjon på valg av dokumentsett og evalueringskriterier

Blant kandidatene har valget blitt ganske åpenbart. Det står mellom Wikipedia og Amazon. **Wikipedia** er antagelig den samlingen som passer aller best. Når det gjelder uavhengig evaluering av dokumentene kan vi bruke søkemotoren Google som utgangspunkt. Å bruke Google som uavhengig evalueringsressurs har den fordelen at det gir oss en rangering rundt forskjellige temaer. Det er forskjellige måter dette kan gjøres på, men det er viktig å ha klart for seg at Google i tillegg til lenkeanalyse bruker tekstrelevans som et av kriteriene for rangering av resultatene. Vi skal ikke bruke tekstrelevans sammen med lenkeanalyse og similaritetsanalyse i denne oppgaven, vi vet heller ikke hvordan Google’s tekstrelevansalgoritme fungerer. Å ta i bruk tekstrelevans for oss sammen med de foreslåtte metodene tror vi vil virke forstyrrende og ta fokus bort fra hvordan metodene i seg selv fungerer.

Måten vi skal bruke Google som uavhengig evalueringskriterie på er å bruke Wikipedias Google-søkefunksjon. Denne sørger for at Google kun søker i wikipedia sine sider og returnerer en rangert liste over matchende dokumenter.

Vi har ikke klart å finne noe informasjon på hvordan Google søker blant dokumenter som befinner seg under et spesielt domenenavn. Vi antar på at de foretar et vanlig søk (med alle lenker funnet på Internett tatt i betraktning) og filtrerer ut de dokumentene som ikke matcher domenet. Dette har den fordelen at rangeringen av dokumentene er et resultat som følge av alle lenkene på hele Internett tatt i betraktning. Alternativt kan det hende at Google har laget sin egen indeks kun over sidene i wikipedia og analyserer lenker i dokumentene i dette lokale domenet og søker i den egne indeksen ved begrenset søk. Men denne påstanden virker ganske urealistisk med tanke på behovet for lagringsplass. En tredje tanke er at Google foretar lenkeanalyse blant wikipedias dokumenter i realtime når man gjør et begrenset søk, men det virker prosessortungvint og unødvendig.

Hvis det er slik at Google tar hele lenkestrukturen på Internett i betraktning når den rangere dokumentene i wikipedia har dette ulempen at Google har en fordel fremfor B3000 når den skal vise resultater. Men vi vil tro at de fleste lenkene som peker til wikipedia fra eksterne sider peker til wikipedias hovedside. Og at dokumentscoren til de fleste artiklene i wikipedia hovedsakelig er et resultat av lenker innad i wikipedia som peker seg imellom, og ikke fra eksterne sider. Denne faktoren er viktig ettersom det vil være med på å gjøre evalueringskriteriet mest mulig ”rettferdig” sammenlignet med B3000 sin rangering

Som påpekt i kapittel 6.2 blir vi nødt til å finne ut hvilke typer lenker som er navigasjonslenker og hvilke som er anbefalingslenker. Dette fordi alle dokumentene i wikipedia ligger innenfor samme domenenavn. Wikipedia inneholder en del navigasjonslenker i de fleste artikler, men disse er lett å gjenkjenne og skille fra anbefalingslenkene. Antallet anbefalingslenker i wikipedia er veldig bra.

7 Utvikling

Utviklingen i dette prosjektet består av to faser. Den første fasen består av grunnleggende arbeid som å klargjøre for videre utvikling av B3000. I tillegg har fase 1 bestått av implementasjon av nødvendige komponenter for å kunne foreta oppslag (enkel søking uten rangering) blant Wikipeidadokumentene. Fase 1 har også bestått av implementering av metoden som brukes for beregning av dokumentlikheter. Alt arbeid i fase 1 ligger til grunn for å kunne gjennomføre den neste fasen i utviklingen. Komponentene implementert i fase 1 brukes "off line"[27].

Fase 2 dreier seg hovedsakelig om implementasjon av komponenter som brukes i sanntid[28]. Dette gjelder de fleste av de metodene som er foreslått i kapittel 5 - "*Kombinering av dokumentlikheter og lenkeanalyse*".

Som nevnt i innledningskapittelet har vi brukt "Enterprise Architect" til moddelering. I dette programmet passet det best å bruke engelsk som navn på funksjoner og hendelser. Derfor er kommentarene som befinner seg i modelleringsfigurene i dette kapittelet på engelsk. I tillegg er alle kommentarer i kildekoden som er vedlagt på CD (appendiks 10.5) skrevet på engelsk da dette føltes mest naturlig. Det kan være nyttig for leseren å ta en titt i appendiks 10.5 for detaljer rundt hvordan de forskjellige komponenter og funksjoner er implementert. All kildekode som er vedlagt på CD får i appendiks 10.5 forklart sin posisjon relatert til de forskjellige modellene som er laget og beskrevet i kapittelet.

7.1 Fase 1

Denne fasen har hovedsakelig bestått av implementering av de komponenter som brukes "offline" for å klargjøre datasettet for oppslag. I denne fasen har vi også gjort en del arbeid for utviklet noen grunnleggende komponenter som er viktig å ha ferdig for å hjelpe oss til å velge hvilke av metodene vi vil jobbe videre med i utviklingen. Hovedsakelig bestod denne fasen av implementering av følgende komponenter:

- Plukke ut ord i dokumentene og lage Invertert Indeks
- Plukke ut lenker i dokumentene og lage lenkestrukturdatabasen
- Beregning av T-Rank Forward verdier basert på lenkestrukturdatabasen
- Beregning av likheter mellom dokumentpar

[27] Med "off line" menes handlinger som skjer uavhengig av en bruker av systemet. Dette er typisk handlinger som er iverksatt manuelt av utviklerne av systemet for å klargjøre eller vedlikeholde.
[28] Med "sanntid" menes handlinger som iverksettes i det en bruker foretar et søk i B3000.

I tillegg har denne fasen inneholdt følgende generelle arbeid:

- Generelt oppsett av FreeBSD server, og installering av MySQL og PHP software for å kunne starte utviklingen.
- Nedlasting og konfigurering av wikipedia databasen. I beskrivelsene under omtaler vi ofte dokumentene i wikipedia som ”artikler”.

7.1.1 Oppsett av hardware og software for testing og utvikling

Datamaskinen vi bruker står på Telenor i Trondheim. Grunnet tidligere erfaringer vi har med utvikling av liknende materiale som i dette prosjektet, hadde vi et ønske om å bruke FreeBSD, PHP og MySQL. Serveren vi har brukt under utvikling og testing har følgende spesifikasjoner:

- **Server hardware**
  Prosessor: Interl Pentium 4, 2.4GHz
  Minne: 1GB Ram
  Annet: 60GB HDD.

- **Operativsystem**
  FreeBSD 5.3-RELEASE (Med modifisert kernel)

- **Databasesystem**
  MySQL Server v. 4.0.21

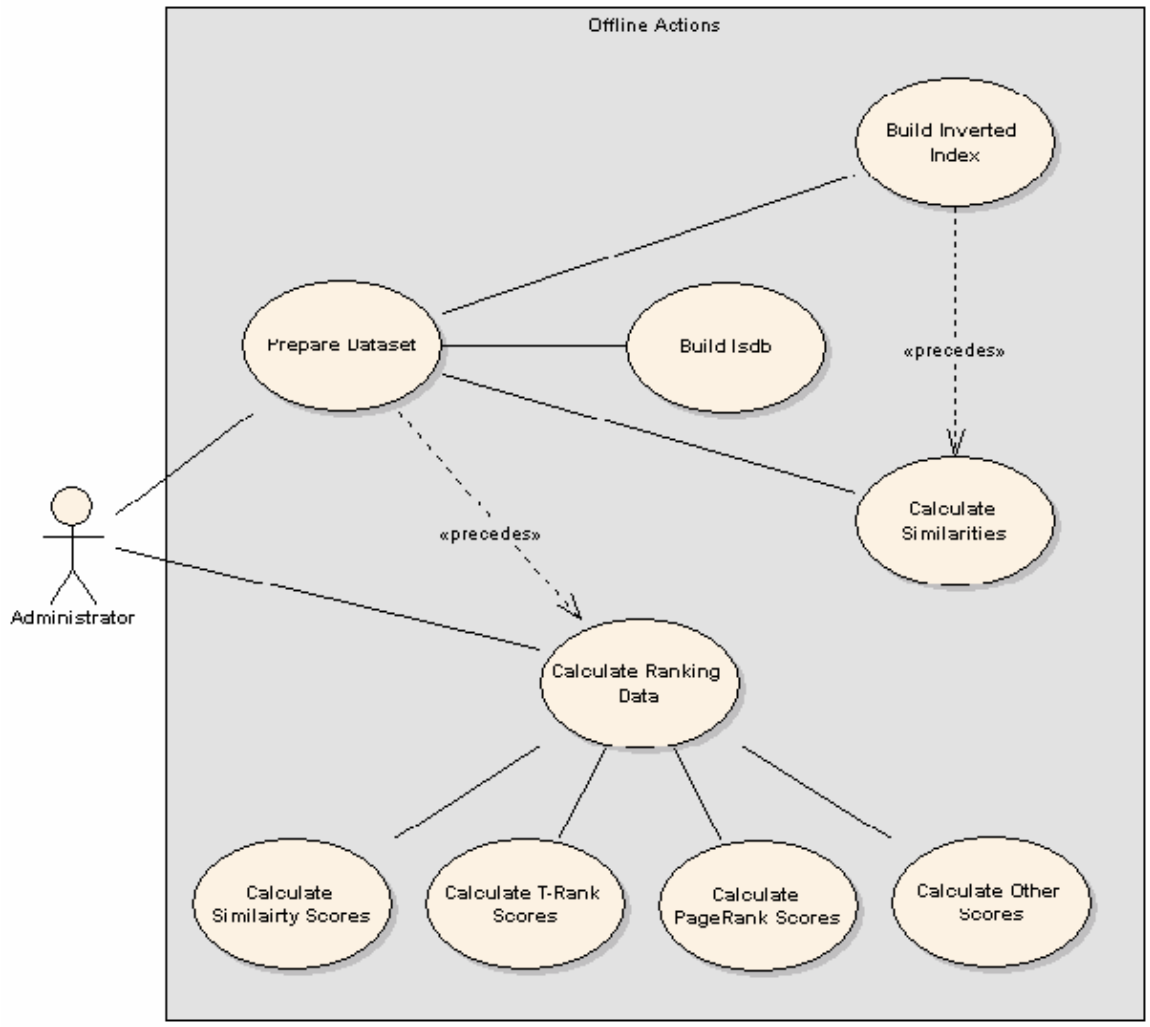
- **Programmeringsverktøy**
  PHP5-5.0.2 (PHP er mer et scriptspråk enn et programmeringsspråk)
  C++

- **Annet**
  Apache web server v. 1.3.33
  (som hjelp til søkegrensesnittet).

Videre i denne fasen vil vi vise modeller av komponenter utviklet under fase 1. Disse modellene vil vi forklare hver for seg, samt sette i sammenheng med hverandre. De forskjellige modellene kan bestå av flere komponenter, som igjen kan inneholde flere funksjoner.

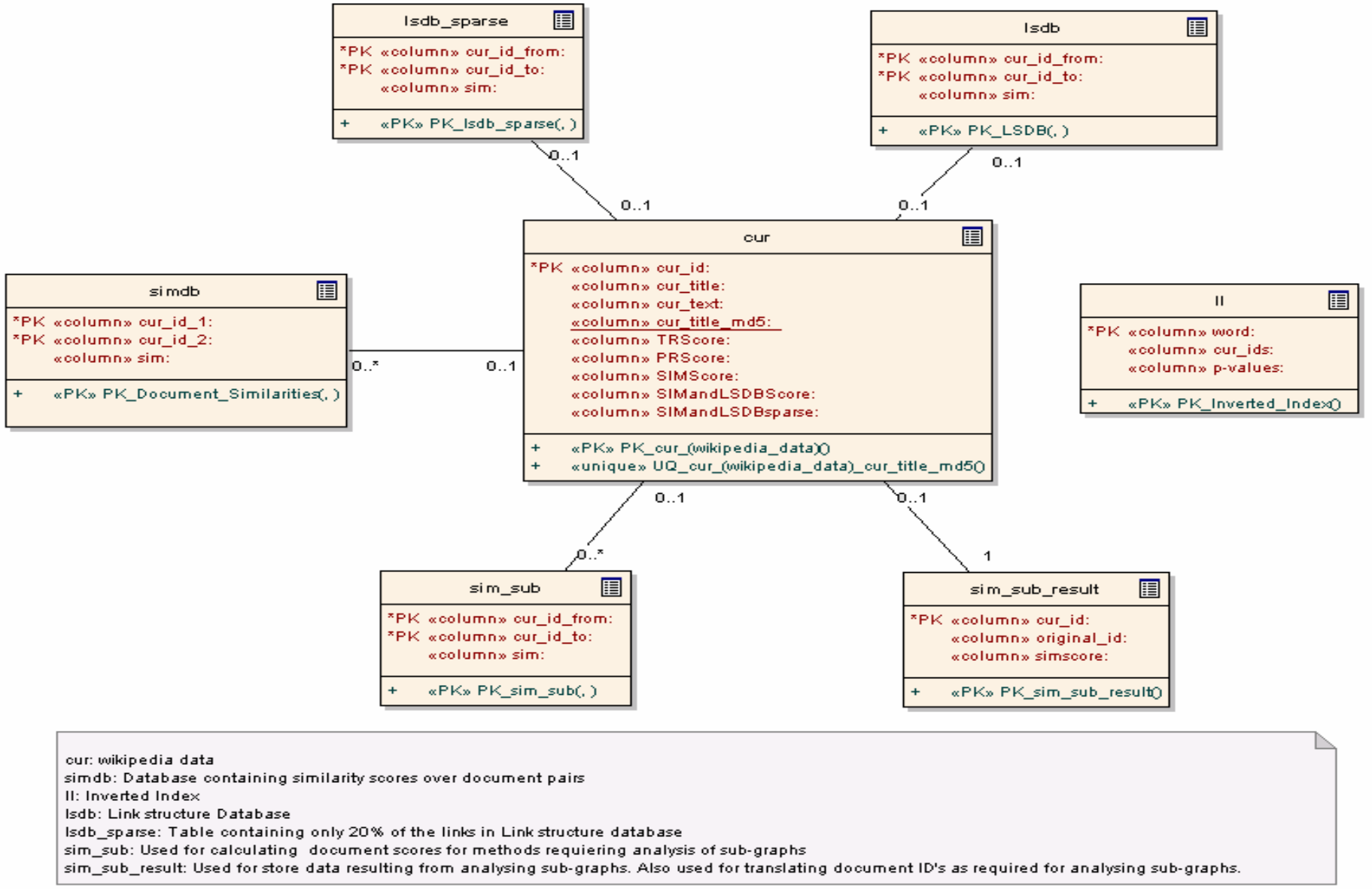
7.1.2 Handlinger som må til for å klargjøre datasettet for søking

"Use Caset" under viser samspillet mellom forskjellige handlinger som må til for å klargjøre datasettet for oppslag og for søking med metodene planlagt i kapittel 5.



Figur 15 - Overordnet oversikt handlinger som må til for å klargjøre datasettet for oppslag.

7.1.3 Oversikt over hvordan databasen er konstruert



Figur 16 - EAR diagram. Oversikt over hvordan MySQL databasen er konstruert.

EAR diagrammet på forrige side viser hvordan databasen er bygget opp, med de relasjoner som er mellom forskjellige tabeller. Denne databasen brukes hovedsakelig til å lagre informasjon som er nødvendig for å foreta oppslag. Men databasen brukes også av komponenter som kjører i sanntid ved søking (sanntidskomponenter beskrevet i fase 2 av utviklingen). Dette gjøres ved at informasjon mellomlagres på en slik måte at de forskjellige komponentene kan kommunisere effektivt. Mange dokumentscores kan forhåndsberegnes uavhengig av søket en bruker foretar. Alle disse lagres i databasen relatert til hver artikkel i wikipedia. De søk som krever ny lenke- eller similaritets-analyse av en subgraf kan ikke forhåndslagres, men må regnes ut i sanntid.

- **simdb (similarity database)**: Dette er en tabell som inneholder oversikt over likheten mellom alle dokumentpar. Hvis to dokumenter er helt ulike inneholder den ikke noe informasjon om disse. Det er unødvendig å lagre informasjon om alle de dokumentene som har likhet 0. Hvis et par dokumenter ikke er oppfør i denne tabellen har de mao. ingen likhet.

- **II (Inverted Index)**:
  word: Denne tabellen er laget på den måten at det er *en* oppføring hvor hvert unike ord i dokumentsamlingen. Disse ordene er primærnøkler i tabellen, derfor går det fort å slå opp dokument ID'er basert på ord.
  cur_ids: Neste kolonne er en "blob". I denne legges ID'en til alle dokumenter som inneholder ordet i forrige kolonne. DokumentID'en "paddes"[29] til en tekststreng bestående av 5 tall. Hvis en dokumentID omgjort til en tekststreng, består av mindre enn 5 tall legges 0 til på begynnelsen av tekststrengen inntil tekststrengen får lenge 5. Dette fordi vi da raskt, *og enkelt*, kan finne alle dokumenter relatert til et ord ved å splitte kolonne "cur_ids" i lengder på 5. Antall dokumenter relatert til et ord kan finnes ved å dele lengden av "cur_ids" på 5. Dette er en måte å gjøre det på som fungerer veldig bra for forskningsformål, men bør kanskje gjøres mer avansert i produksjonskode hvor datasamlingen man skal lage inverter indeks over er mye større enn i vårt tilfelle.
  p-values: Denne kolonnen fungerer på samme måte som kolonne "cur_ids". Den inneholder tallverdier, lagret som en tekststreng, over hvor relaterte de tilhørende dokument ID'ene i forrige kolonne er til orde de står oppført i. Tallet som beskriver dokumentenes relaterthet til ordet er en tekststreng på lengde 9, hvor en av tegnene i tekststrengen er punktum (største mulige likhet er 1.0000000), dette gir oss en presisjon på 10^{-7}, som er mer enn presist nok til å beskrive dokumentets relevans til ordet. Følgelig må "p-values" strengen splittes i biter på lengde 9 for å

[29] Se kapittel 10.6 for en beskrivelse av hva dette ordet betyr.

få ut denne dataen, og dette kan optimaliseres bedre ved produksjonskode, men fungerer fint for vårt formål. Det er for øvrig viktig å merke seg at rekkefølgen til verdiene som lagres i cur_ids og p_values til en hver til må være synkronisert, da de avhenger av hverandre. Det er ingen mekanisme som sørger for dette. Man må i koden, selv ta høyde for at det blir synkronisert etter hvert som verdier lagres i disse colonnene.

- **lsdb (Link Structure Database)**: Dette er en tabell som enkelt og greit inneholder to kolonner av typen int. Verdiene som puttes i kolonnene er hhv. Dokument ID'en hvor lenken er funnet (new_id), og DokumentID'en hvor lenken peker til (new_id_to). Begge kolonnene til sammen utgjør primærnøkkelen og beskriver om det finnes en lenke mellom "new_id" og "new_id_to". Vi lagrer ikke informasjon i de tilfeller hvor det ikke er lenker mellom dokumentene (selv om T-Rank matriseanalyse krever å vite dette). Ingen oppføring = ingen lenke. Når vi setter i gang T-Rank bygger vi en matrise som også inneholder informasjon om de dokumenter som ikke har lenker mellom seg.

- **lsdb_sarse (Link Structure Database Sparse)**: Dette er en tabell som kun inneholder 20% tilfeldig valgte lenker fra den original lsdb tabellen.

- **cur (wikipedia_data):** Dette er en tabell som utgjør wikipedia leksikonet. Den har vi lastet ned fra: http://download.wikimedia.org og importert inn i mysqldatabasen på serveren, som en del av tabellstrukturen der. Den original e cur tabellen inneholder opprinnelig mange flere kolonner. Cur tabellen i Figur 16 er en tabell med kun de data som er nødvendig og relevant til dette prosjektet. Det står mer om valg av datasett i kapittel 1.

  Tabellen inneholder teksten for hvert dokument i wikipedia, samt en del informasjon relatert til denne teksten. Blant annet inneholder tabellen en scoreverdi for hvert dokument for alle de metoder som muliggjør beregning av disse offline. Dette gjelder beregningsmetoder som tar i betraktning hele lenkestrukturdatabasen og / eller hele similaritetsdatabasen:
  - *TRScore* er resultater fra T-Rank brukt på hele lsdb.
  - *PRScore* er resultater fra PageRank brukt på hele lsdb.
  - *SIMandLSDBScore* er resultater fra en kombinasjon av T-Rank og similaritycomputing brukt på hele lsdb.
  - *SIMandLSDBSparse* er resultater fra en kombinasjon av T-Rank og similaritycomputing bruk på hele lsdb_sub.

Et problem med wikipedia tabellen er at den innholder en del informasjon (entries) som er uønskelig i dette prosjektet. Eksempler på slik informasjon er

- Entries brukt for å lagre bilde-data
- Informasjon om at wikipedia ønsker en hel artikkel om emnet, eller en forbedret og mer utdypende versjon av artikkelen slik den er.
- Diskusjonstekst mellom de forskjellige forfatterne i wikipedia.
- Artikler som er merket som spam
- Artikler som er foreslått for sletting.
- Med mer.

I alt har wikipedia 16 forskjellige typer entries i tabellen. De forskjellige entriesene er merket med forskjellige ”cur_namespace”-verdier. Det er bare entries med en spesiell verdi i cur_namespace vi er interessert i (selve artiklene). Akkurat disse artiklene er merket med ”cur_namespace = 0”. Men i tillegg er disse artiklene blandet med artikler av typen ”#REDIRECT: Nytt_Artikkel_Navn” (som også er merket ”cur_namespace=0”). Det er ingen annen måte å finne disse typene artikler enn å lete gjennom artikkelteksten. Vi kan ikke simpelthen slette de artiklene som er ”#REDIRECT” i artikkel teksten, for andre artikler kan ha lenker til slike dokumenter som sender oss videre, og følgelig er denne informasjonen viktig for å bygge en korrekt lenkestrukturdatabase. Alle andre typer entries i tabellen har cur_namespace=1-15, og disse kan fint fjernes.

cur_text i denne tabellen er den rå teksten som inneholder artikkelinformasjonen (eller redirect informasjon). Denne teksten består delvis av HTML og delvis av wikipedias egne og enkle tekstformattering, i tillegg til vanlig tekst.

Lenkene i wikipedia er formatter til å være lik tittelen til dokumentet lenken peker til[30]. Et problem i denne sammenheng er at wikipedia kan inneholde forskjellige dokumenter, med lik tittel, avhengig av om tittelen inneholder store og små bokstaver på forskjellige steder. For eksempel inneholder dokumenter med følgende titler forskjellig informasjon:

[30] Dette stemmer hvis man ser i den rå dokument-teksten til en artikkel. Det stemmer også, i de fleste tilfeller, hvis man ser på artiklene online. Men en lenke *kan* se annerledes ut en tittelen til artikkelen den peker på online. Formatteringen som brukes for dette er [[lenke_adresse_og_tittel_navn | ankertekst]]. Men de fleste lenker er formattert på formen [[lenke_adresse_og_tittel_navn_og_ankertekst]] se kildekode vedlagt på CD for hvordan dette gjøres i praksis.

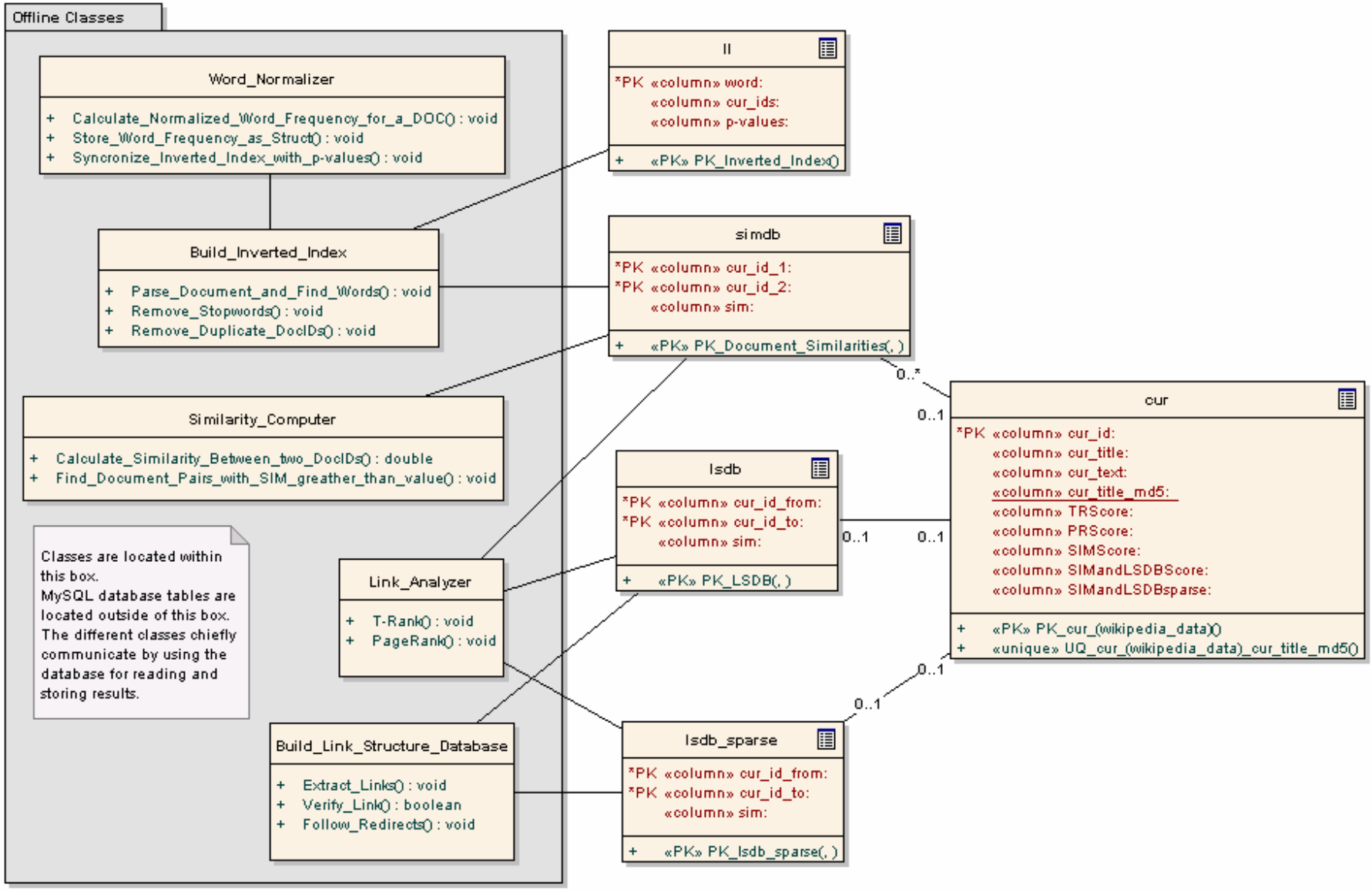
Tittel: ”**MB**” (Informasjon om hva en MegaByte er)
Tittel: ”**Mb**” (Informasjon om hva en MegaBit er)
Tittel: ”**mb**” (en enhet for trykk lik en hektopascal)

I Invertert Indeks lagrer vi all informasjon med små bokstaver, uavhengig om noen eller alle bokstavene i et ord er store. Dette har medført at funksjonen ”*Find / Extract Links*” i Figur 1 har blitt en helt egen funksjon (som alene parser dokumentene) og mer avansert enn den var til å begynne med (før vi innså dette problemet). Til å begynne med var den en del av funksjonen ”*Build Inverted Index*” i Figur 1 for å spare oss for unødvendig kjøretid ved å måtte parse alle dokumenter flere ganger.

I den norske versjonen av wikipedia rangerer dokument ID’ene fra 1 – 25599, men det er kun 24846 av disse som beskriver en faktisk artikkel, videre er det kun 19129 igjen etter vi fjerner redirect-artikler . ”Hullene” i dokumentlista er grunnet at dokumenter har blitt slettet, eller at ”dokumentet” faktisk var en ”#REDIRECT” til et annet dokument i stedet for en artikkel (grunnet wikipedias oppbygning og virkemåte). Det er nødvendig for virkemåten av T-Rank at dokumentID’ene starter på 1, og slutter på siste dokumentID *uten hull* i lista. Vi har derfor konvertert dokumentID’ene I wikipedia til å rangere fra 1 – 19129.

- **sim_sub**: Denne tabellen er et resultat av måten MySQL fungerer som kommunikasjonsgrensesnitt mellom de forskjellige komponentene i prosjektet. sim_sub er nødvendig for å regne analyse av lenker, eller likheter mellom dokumentpar ,eller en kombinasjon, over dokumenter i en subgraf. Måten T-Rank er implementert på krever at matrisen den skal regne på starter med dokument ID=1 og inneholder en ID for hvert dokument helt opp til det siste dokumentet. Det er derfor nødvendig å konvertere dokumentID’ene når vi skal analysere subgrafer, mellomlagre resultatet her, og be T-Rank analysere matrisen som denne tabellen representerer. T-Rank lagrer så resultatvektoren i **sim_sub_result**, og vi må videre hente ut dette, konvertere ID’ene tilbake til de opprinnelige for så å kunne rangere dokumentene etter denne analysen. Dette er tungvint, men gjelder kun subgrafer av samlingen som ofte er små, og er derfor ikke noe problem. Det må også her understrekes at dette prosjektet er et forskningsprosjekt for å teste ut ideer.

7.1.4 Oversikt over offline komponenter og deres relasjoner.



Figur 17 - Oversikt over de forskjellige komponenter og deres relasjoner som brukes offline.

Alle komponenter i klassediagrammet er frittstående, med MySQL som et felles grensesnitt for kommunikasjon mellom hverandre. Hver klasse leser data og lagrer ferdigbehandlede resultater i en felles database slik at andre komponenter kan bruke resultatene. Forfatteren av oppgaven har valgt at vi skal gjøre det slik fordi han kan MySQL og PHP veldig godt fra før av. Følgelig var denne løsningen med å få hans fokus på selve implementeringen av komponentene raskest mulig, og unngå unødvendig frustrasjon som sannsynligvis ville oppstått om det hadde blitt gjort annerledes. Alternativet var å implementere komponentene i C eller C++. PHP er enklere og mye raskere å programmere i enn C. Ulempen er at PHP resulterer i lengre kjøretid. Siden oppgaven er å teste ideer for forskningsformål, og ikke krever stabil produksjonskode, er dette en bra løsning. Å velge et annet programmeringsspråk ville fått den konsekvensen at vi ikke hadde rukket å utvikle og teste så mye som vi nå har rukket.

- **Link_Analyzer**: Denne komponenten er laget og implementert i programmeringsspråket C av Hans Kristian Kismul. Den bygger et array i minnet på størrelse med antall dokumenter^2. Deretter fylles dette arrayet med data fra linkstrukturdatabasen, og en metode kalt PowerMethod brukes iterativt på matrisa inntil egenvektoren inneholdende en verdi for hver DocID i samlingen konvergerer. Dataen i denne egenvektoren er viktighetsscoren for dokumentene som sett av rangeringsfunksjonen som er brukt, hhv. T-Rank eller PageRank. Hvordan T-Rank og lenkeanalyse generelt fungerer, er mer detaljert beskrevet i kapittel 0. Det er gjort en del ekstra arbeid med denne komponenten for å håndtere minnebruken som beskrevet i kapittel 2.3 - ”*Behandling av store matriser i minnet*”. Arrayet over lenker mellom dokumentene er på 400 millioner entries som er allokert som ”double” (8 bytes). Dette tilsvarer omtrent 4 Gigabytes, noe vi ikke har tilgjengelig. Derfor brukes matrise oppdeling og matriseaddisjon som beskrevet i kapitel 2.3. Denne matriseoppdelingen medfører at de forskjellige oppdelingene må mellomlagres på harddisken, og dette er med på å øke kjøretiden en del ettersom Link_Analyzer ikke klarer å utnytte prosessoren mer enn ca. 50% under store deler av analyseringsprosessen.

Alle komponenter beskrevet under implementert av meg i PHP. Kildekode finnes vedlagt på CD.

- **Build_Link_Structure_Database**:

  Extract_Links(): Denne funksjonen leser alle dokumentene i wikipedia databasen, parser teksten og plukker ut lenkene.

  Verify_Links(): Denne funksjonen verifiserer at lenkene funnet peker til et gyldig dokument som finnes i wikipedia.

  Follow_Redirects(): Denne funksjonen sjekker om en peker til et dokument i wikipedia ikke inneholder tekst, men videresender brukeren til et annet dokument. Hvis det er tilfellet finner den DocID'en til dokumentet man blir videresendt til og lagrer denne DocID'en som target i stedenfor DocID'en som faktisk pekes til. Eksempel på et slikt tilfelle er en lenke som peker til "Arbeiderpartiet", blir videresendt til dokument "Det_Norske_Arbeiderparti". Det er grunnet wikipedias oppbygning og funksjonalitet at slike tilfeller oppstår, og det er nødvendig å behandle tilfellene slik for å få en korrekt lenkestrukturdatabase. Denne funksjonen følger et maksimum av 10 videresendinger for å unngå uendelige løkker.

- **Build_Inverted_Index**
  Parse_Document_and_Find_Words(): Denne funksjonen leser alle dokumentene i wikipedia, fjerner all uønsket formattering som HTML og wikipedias egne interne tekstformattering. Dette fordi vi, i dette prosjektet, ikke skal ta hensyn til hvor i dokumentet tekst står eller hvordan den er formattert (bold, italic, overskrifter og lignende). Hadde vi skulle foretatt tekst relevans mellom dokumenter og søkeord ville vi måtte ta hensyn til denne tekstformatteringen i dokumentene. Videre fjerner funksjonen alle uønskede ord. Uønskede ord kan enten være ord som består av tall eller tegn som ikke er med i det norske alfabetet. Den fjerner også ord som inneholder tall. Grunnen til det er at en del html-formattering (som ikke fanges når vi fjerner html-tags) resulterer i ord inneholdende tall. For eksempel blir "Erwin Schrödinger" (Østerrisk fysiker) til *Erwin Schr&#246dinger*, som igjen blir *Erwin Schr256dinger* etter første operasjon i tekstformatteringen. Siden dette er en norsk oppgave har vi valgt å se bort fra slik ord, og løsningen på det er å fjerne ord som, etter første skritt i formatteringen, inneholder tall. Ord som inneholder tall virker for oss uansett som merkelige, uinteressante og dårlig til å beskrive hva teksten handler om. Ord som inneholder æ, ø eller å derimot, har vi i

programmeringen tatt høyde for skal bli tatt med som ord. Wikipedia har sin egen formattering av enkelte tegn, deriblant æ, ø og å. Disse tegnene parser vi om til tegn som systemet vi har laget forstår som hhv. æ, ø og å. Kildekode for dette finnes vedlagt på CD. Når all teksten i et dokument er gjort om til bokstaver vi er interessert i, putter vi hvert ord inn i et array. Etter å ha fjernet stoppord legger vi så, for hvert ord vi fant i dokumentet, ID'en til dokumentet vi har parset inn i Invertert Indeks.

Remove_stopwords(): Dette er en relativt enkel funksjon som fjerner stoppord i arrayet (laget av "Build_Inverted_Index") over alle ord for et dokument.

Remove_Duplicate_DocIDs(): Etter at hele prosessen med å bygge det inverterte indekset er ferdig bruker vi denne funksjonen. Den går gjennom alle DocID'er for hvert ord i Invertert Indeks og fjerner duplikate ID'er. Dette er egentlig ikke nødvendig ettersom duplikater egentlig ikke oppstår når vi bygger ferdig invertert indeks i minnet før vi lagrer den. Å fjerne duplikater er en veldig rask oppgave, og kun en sikkerhet for at redundant informasjon ikke skal bli lagret i Invertert Indeks. Denne funksjonen er også nødvendig å ha om vi ikke hadde hatt nok minne tilgjengelig til å bygge hele det inverterte indekset i minnet før vi lagrer det i databasen. I slike tilfeller hadde vi trinnvis måttet lagre informasjon i databasen. Det ville økt kjøretiden betraktelig om vi da måtte ta hensyn til om en ID allerede er lagret i databasen for hvert ord før vi legger til en DocID, enn heller å kjøre denne funksjonen etter at invertert indeks er ferdig bygget (med duplikate ID'er). På denne måten slipper vi å ta hensyn til duplikater underveis for hvert dokument, men heller *en felles* gang til slutt.

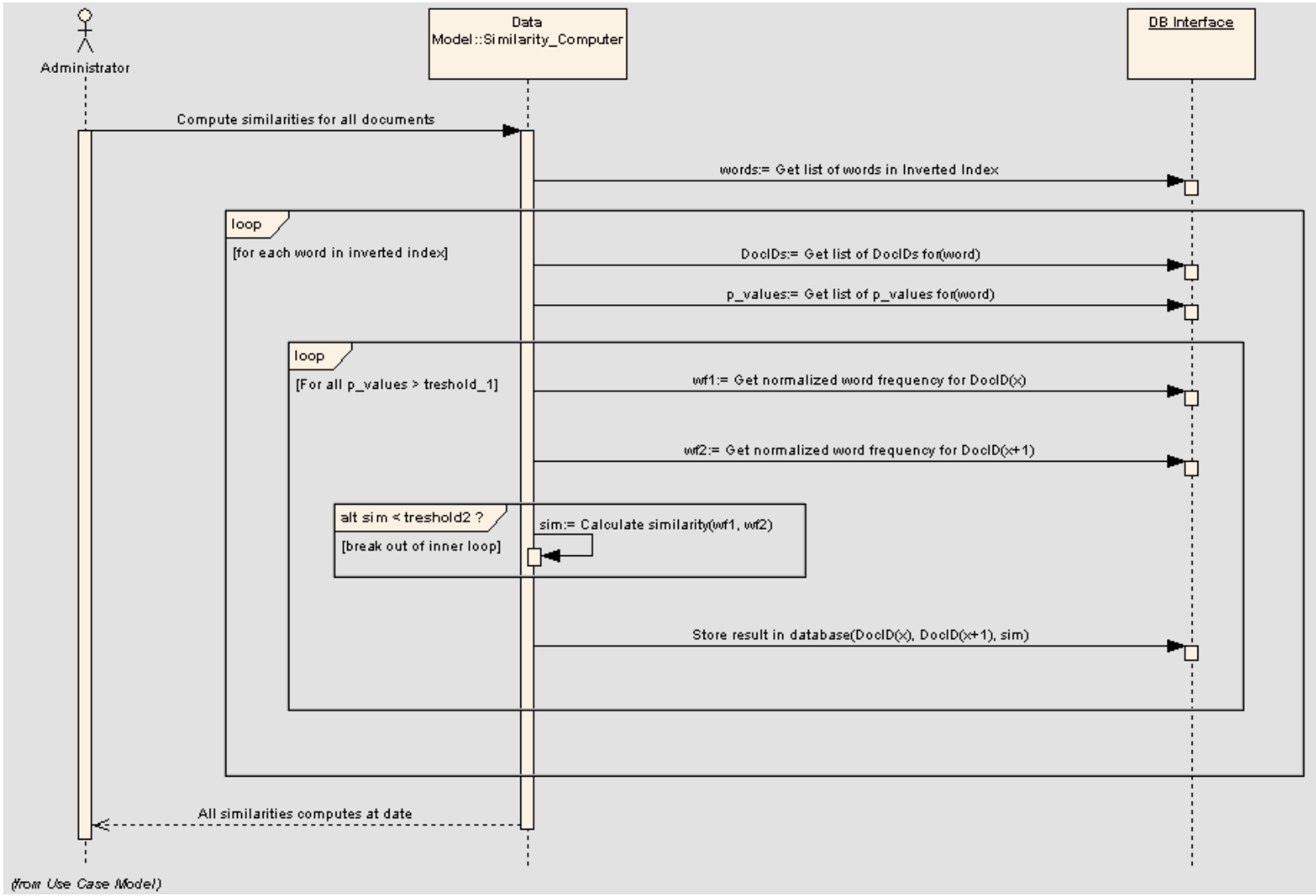
- **Word_Normalizer**
  Calculate_normalized_word_frequency_for_a_DOC(): Denne funksjonen tar utgangspunkt i resultatet fra "Build_Inverted_Index" sin "Parse_Document_and_Find_Words()". Den starter derfor med et array over alle ordene i et dokument. Dette arrayet inneholder ikke bare unike ord, men alle ordene i dokumentet. Den lager så en ny liste over de unike ordene i dokumentet, og skriver i denne listen hvor mange ganger hvert ord forekommer i dokumentet. Deretter deler den alle resultatene på det totale antall (inkludert ord som forekommer flere ganger) ord i dokumentet. Resultatet er en array over hvor stor vekt hvert ord i et dokument har. Hver slike verdi kalles for *p-value*. Mer informasjon om hvordan vi tenker oss dette kan finnes i kapittel 4.6. Dette resultatet er meget viktig for både funksjonen "Synchronize_Inverted_Index_with_p-values()" og for "Similarity_Computer"

sin funksjon ”Calculate_normalized_word_frequency_for_a_DOC()”.

Store_word_frequency_as_struct(): Dette er en funksjon som lagrer den normaliserte ordfordelingsfrekvensen, som er produsert av ”Calculate_normalized_word_frequency_for_a_DOC()”, som en strukt i en fil på serveren. Vi fant ut at dette er den mest effektive måten å lagre denne informasjonen på for å raskest mulig kunne lese de tilbake i minnet når ”*Similarity_Computer*” sin ”Calculate_Similarity_between_two_DocIDs()” skal regne likhet mellom dokumenter. Det er usikkert hvor mange ganger den må lese denne informasjonen opp i minnet, og det verst tenkelige tilfellet er N^2, hvor N er antall dokumenter i samlingen. Å lagre informasjonen som en tabell i MySQL så vi for oss ville bli komplisert og ta unødvendig mye tid for hver gang informasjonen skal leses tilbake. For mer informasjon om strukter se kapittel 4.3 – ”*Likhet mellom alle dokumentpar i en samling - praktisk*”.

Syncronize_Inverted_Index_with_p-values(): Denne funksjonen sorterer DocID’ene i Invertert Indeks etter hvor relatert hvert dokument er til det ordet dokumentet står oppført i. Dette er viktig for at terskelverdi 1, som brukes i ”*Similarity_Computer*” sin ”Find_Document_pairs_with_SIM_greater_than_zero()” skal kunne droppe å regne likhet mellom dokumenter som er *forventet* å ha en likhet mindre enn terskelverdi 1.

- **Similarity_Computer (figur neste side)**



Figur 18 - Detaljert oversikt over virkemåten til "Similarity_Computer"

Figur 18 viser interaksjoner mellom funksjonen ”Similarity_Computer” og databasen. Under følger en beskrivelse av de forskjellige funksjonene i denne komponenten.

Calculate_Similarity_Between_two_DocIDs(): Denne funksjonen blir gitt to dokument ID’er, leser så opp ordfrekvensfordelingen allerede laget av ”Word_Normalizer” og kalkulerer likheten mellom dokumentene som beskrevet i kapittel 4.2 – ”*Likhet mellom to dokumenter*”.

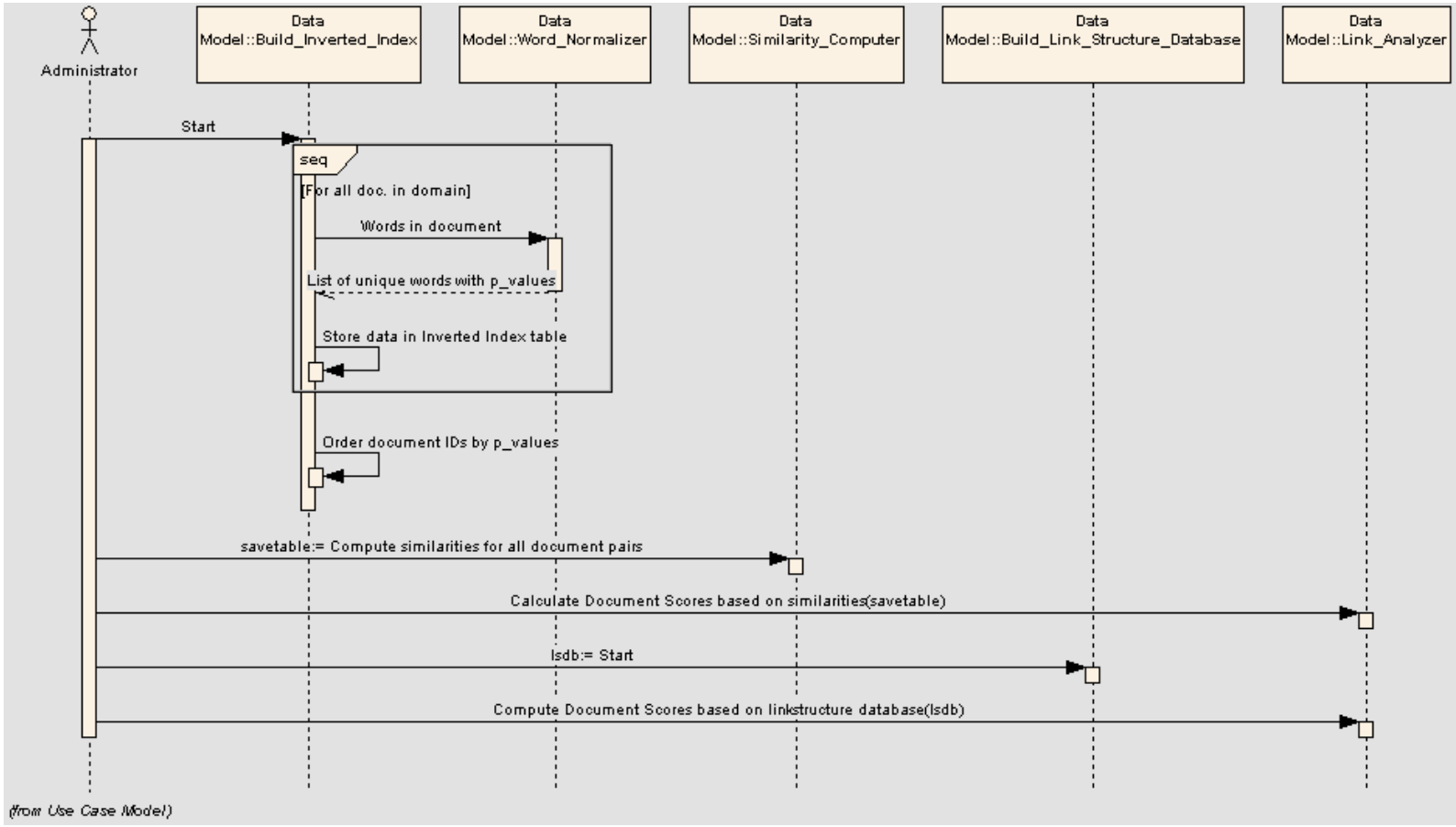
Find_Document_pairs_with_SIM_greater_than_zero(): Dette er en meget nyttig funksjon som er et resultat av ideene beskrevet i kapittel 4.6 - ”

Likhet mellom alle dokumentpar i en samling - Metode 3". Kort sagt finner den de dokumentpar som er garantert å ha en likhet større enn en terskelverdi. Funksjonen opererer med to terskelverdier. Terskelverdi 1 forteller at funksjonen skal stoppe å beregne likhet (mellom dokumenter oppført for et aktuelt ord i invertert indeks), og heller gå videre i invertert indeks, når den når et dokument som er mindre relatert til det aktuelle ordet enn denne terskelverdien. Terskelverdi 2 forteller funksjonen at den skal stoppe å beregne likhet (mellom dokumenter oppført for et aktuelt ord i Invertert Indeks), og heller gå videre, hvis likheten mellom to dokumenter er mindre enn denne terskelverdien.

Dette er veldig tidsbesparende ettersom de aller fleste dokumentpar i en stor samling naturlig nok vil ha likhet null, eller veldig lav likhet. Jo høyere likheten er, jo lettere er det å finne ut hva begge dokumentene handler om. Mine veiledere på Fornebu ble veldig glade da de fikk se med egne øyne at vi hadde klart å finne og beregne likhet mellom alle dokumentpar som hadde likhet større enn 0. Under er et interaksjonsdiagram som viser, i mer detalj, hvordan funksjonen "Similarity_Computer" fungerer.

7.1.5 Oversikt over interaksjoner mellom offline komponenter

Det er ikke så mange automatiske interaksjoner mellom de forskjellige komponentene som brukes off line. Det er viktig at forskjellige oppgaver blir iverksatt til riktig tid i forhold til hverandre. For det meste er de forskjellige jobbene iverksatt i riktig rekkefølge av oss. Under er et interaksjonsdiagram over hvordan de forskjellige komponentene må iverksettes for at forberedelsene av datasettet skal la seg gjøre.



Figur 19 - Oversikt over interaskjoner mellom forskjellige funksjoner og i hvilken rekkefølge de må iverksettes for å forberede datasettet for søking i sanntid.

Som figuren viser er det kun Build_Inverted_Index som har en interaksjon med Word_Normalizer. Alle prosesser settes i gang av oss manuelt, og i riktig rekkefølge. Build_Inverted_Index bruker Word_Normalizer for hvert dokument i samlingen.

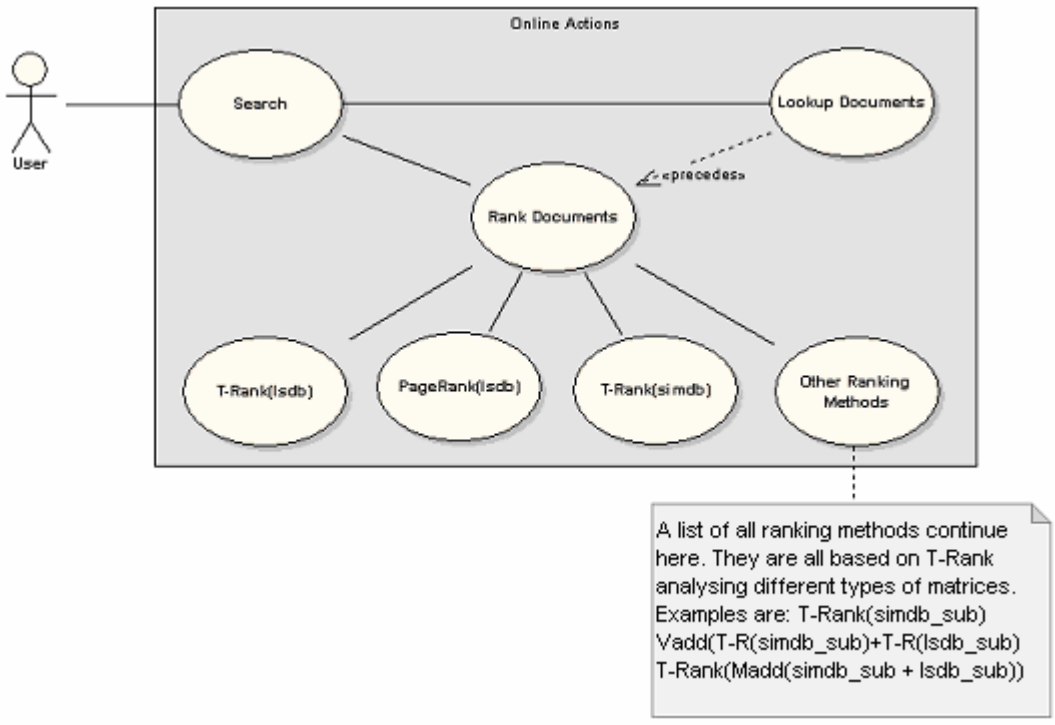
Fase 2

I denne fasen prioriteres metodene i kapittel 5, i lys av resultatene fra fase 1 i utviklingen. Det var positivt at det skulle gå så fort å beregne dokumentlikheter etter fase 1.

Hovedsakelig har denne fasen inneholdt følgende arbeider:

- Design av den delen av systemet som brukes i sanntid ved søking.
- Implementering av nødvendig logikk for å få til kombinasjoner av de analysemulighetene fase 1 har muliggjort.
- Implementering av nødvendig logikk for å foreta analyser av deler av lenkestrukturmatrise, eller deler av likhetsmatrise eller kombinasjoner av disse i sanntid..
- Lage logikk for hvordan de forskjellige komponentene skal brukes sammen for å utgjøre det ferdige søkbare produktet.
- Implementasjon av et utvalg av rangeringsmetodene beskrevet i kapittel 5.

7.1.6 Handlinger som må til for å søke i sanntid

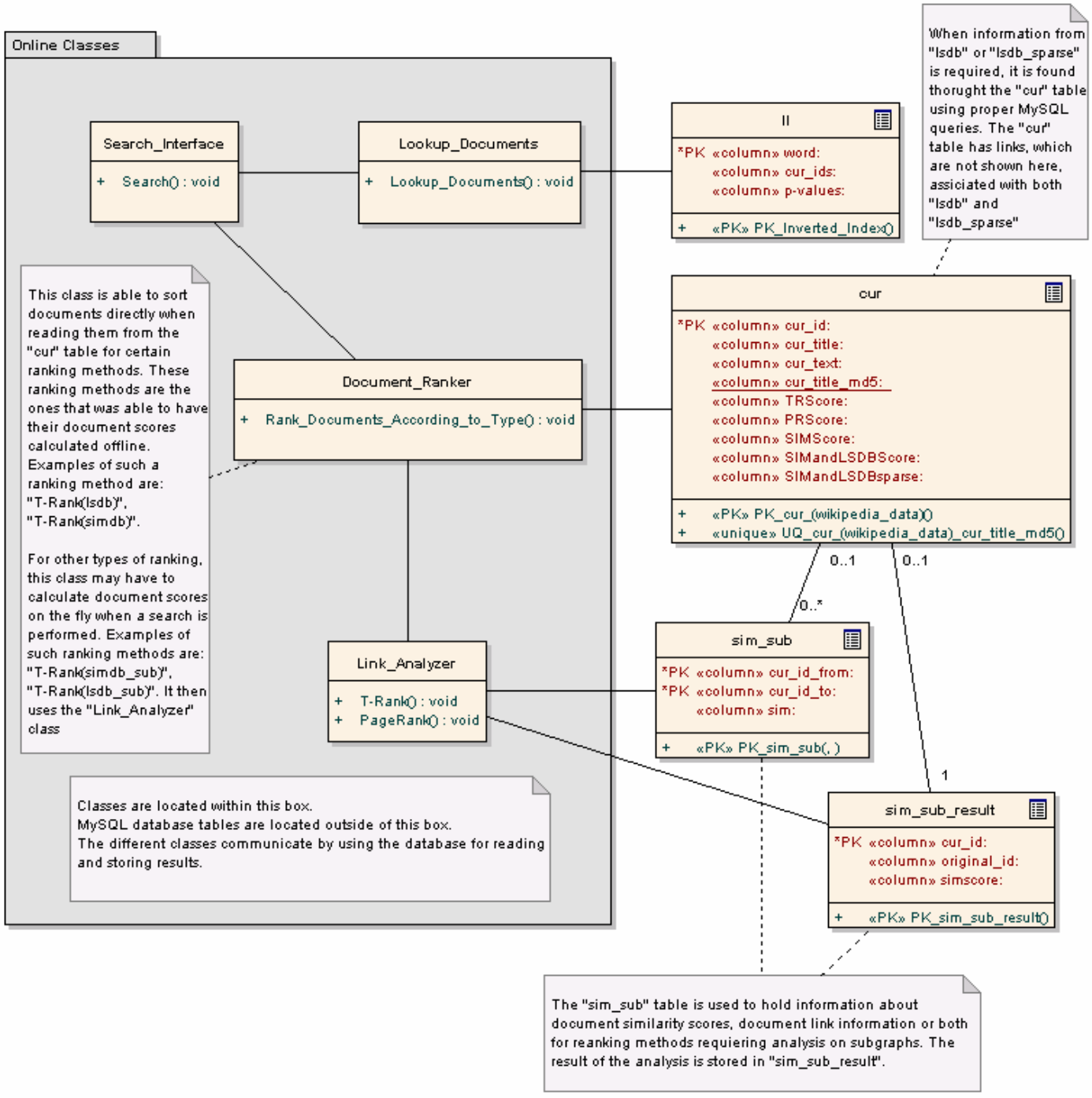


Figur 20 – Overordnet oversikt over handlinger som må til for å søke i sanntid.

”Use Caset” over viser de handlinger som brukes i sanntid ved søking. De nederste boblene inneholder logikk for å bygge vektorer (lister) og matriser i minnet som representere sub-grafene som skal analyseres. Etter at disse vektorene og matrisene

matrisene er ferdig bygget i minnet brukes T-Rank på denne/disse. Deretter blir resultatet behandlet om nødvendig og brukt videre til å rangere dokumentene i sanntid.

7.1.7 Komponenter brukt i sanntid ved søking og deres relasjoner



Figur 21 – Oversikt over de forskjellige komponenter som brukes i sanntid når et søk foretas.

Klassediagrammet over illustrerer hvordan forskjellige komponenter i systemet jobber sammen for å muliggjøre rangering av subgrafer i sanntid. Under følger en beskrivelse av komponenter og funksjoner i denne figuren.

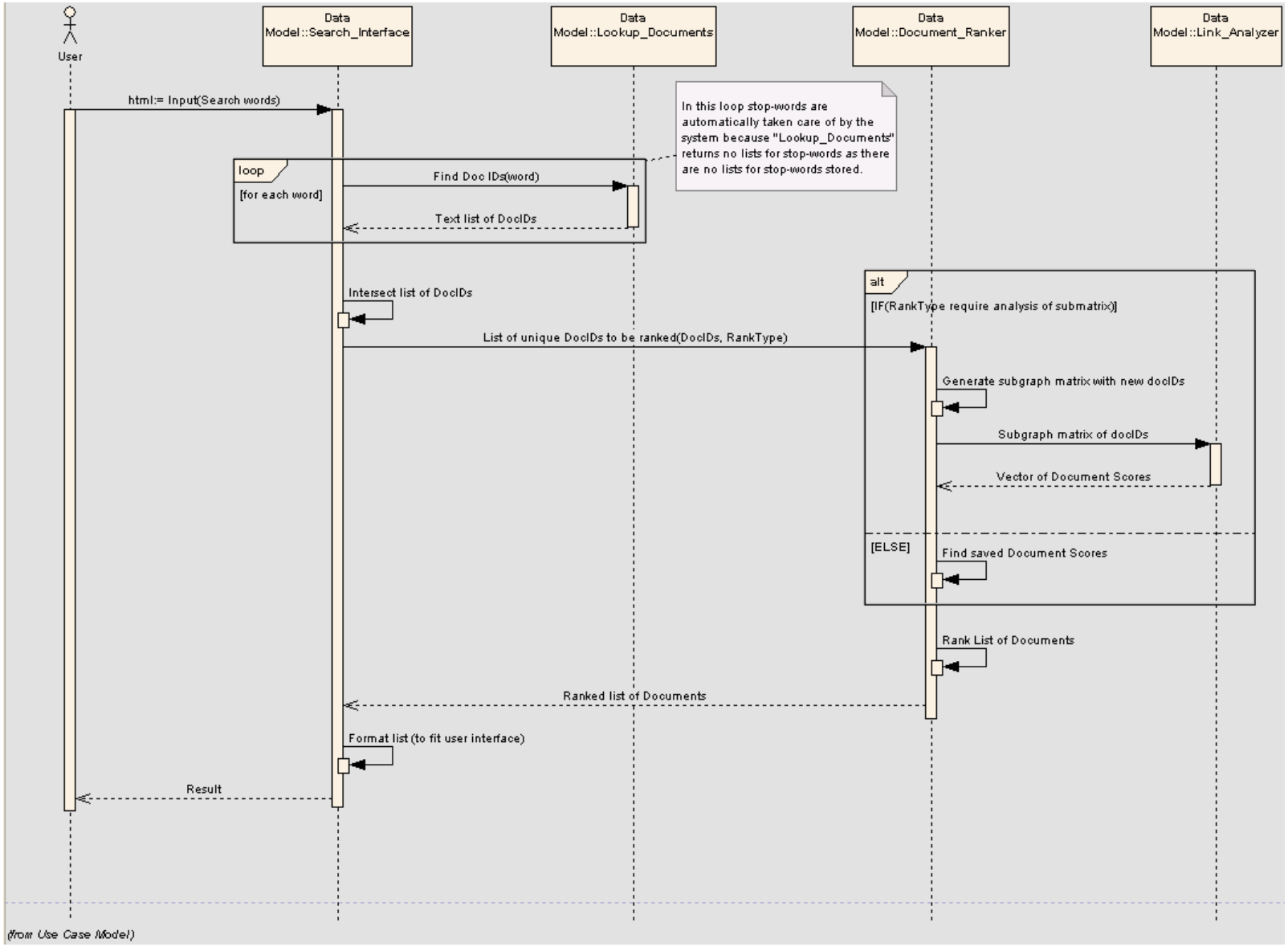
- **Dokument_Ranker**
  Rank_Dokuments_according_to_type(): Dette er en funksjon som, basert på allerede utregnede og lagrede verdier for hvert dokument, rangerer resultatene av en spørring etter dokumenter. Den kan også rangere dokumenter basert på kombinasjoner av de allerede utregnede og lagrede dokumentverdiene. Avhengig av hvilken søkemetode brukeren velger vil denne komponenten velge riktig rangeringsmetode, eller iverksett nødvendige tiltak for online analyser av subgrafer for rangering. I tilfeller hvor brukeren har valgt å basere rangeringen på analyser av subgrafer bygger denne komponenten en komplett matrise over dokumentID'er, med oversikt over lenker og likheter disse imellom. DokumentID'ene blir så konvertert som beskrevet i kapittel 7.1.3, og T-Rank blir kalt til å analysere matrisa. Etter at denne analysejobben er ferdig blir dokument ID'ene konvertert tilbake og rangert etter de tilhørende scorene som T-Rank har beregnet. Dokumentenes tittel som relatert til ID'ene blir så plukket ut av "cur" databasen, og sendt tilbake til brukeren.

- **Lookup_Documents**
  Denne komponenter slår opp i invertert indeks de dokument ID'er som på forhånd er relatert til et ord. Dette gjør det for hvert ord (bortsett fra stoppord) brukeren supplerer søkemotoren med.

- **Search_Interface**
  Search(): Dette er i all hovedsak koden som tar imot spørringer og sørger for at de når riktige komponenter på riktig måte. Den tar også imot resultatet av spørringer og formatterer det til et ønskelig format. Dette formatet er HTML på websiden http://b3000.glenn-erik.com . Websiden lar brukeren velge mellom alle rangeringsmetodene implementert. I tillegg lar interfacet, for enkelte rangeringsmetoder, brukeren vektlegge analysen av lenker eller analysen av likheter i forskjellig grad. Dette gjøres ved å brukeren kan angi en verdi "TR tweak" for sterkere vekting av lenkeanalyse, og en verdi "SIM tweak" for sterkere vekting av likhetsanalyse. Denne vektleggingen har kun påvirkning på de metoder som bruker både likhetsanalyse og lenkeanalyse kombinert. Dette gjelder metodene 17, 16, 8, 19, 5 og 9 i Figur 14. Avhengig av om man velger en rangeringsmetode som resulterer i addisjon av to vektorer eller addisjon av to matriser, vil vektingen med "TR tweak" og "SIM tweak" fungere forskjellig. I det

første tilfellet, addisjon av to matriser, multipliseres ”TR tweak” til matrisen som inneholder informasjon om lenkestrukturen. Hver lenke vil få verdien i ”TR tweak” satt til istedenfor verdien 1. Deretter multipliseres ”SIM tweak” til matrisen som inneholder informasjon om hvor like alle dokumentpar er. Så kjøres ”vanlig” prosedyre med addisjon av disse matrisene og T-Rank analyse av den. I det andre tilfellet regner T-Rank ut resultatet av både lenkestrukturdatabasen og similaritetsdatabasen hver for seg *uten* at ”TR weak” og ”SIM tweak” er tatt i betraktning. Det er først når resultatet av T-Rank utregningene skal adderes sammen (to vektorer) at de først multipliseres med hhv. ”TR tweak” og ”SIM tweak”.

7.1.8 Interaksjoner mellom komponenter brukt ved søking i sanntid



Figur 22 - Oversikt over de forskjellige komponentene og deres interaksjoner som må til for å søke i sanntid.

Figur 22 viser de interaksjoner som må til mellom forskjellige komponenter i systemet når et sanntidssøk foretas. I kortetrekk skjer det på følgende måte: Search_Interface tar imot spørringen, bruker Lookup_Documents til å finne en liste med relaterte dokumenter for hvert ord i spørringen. Tar snittet av alle disse listene. Sender lista sammen med ønsket rangeringsmetode til Document_Ranker, som foretar all nødvendig prosessering for å rangere dokumentene og returnerer en ferdig rangert liste.

7.1.9 Definisjon av uttrykk som har oppstått underveis

Under arbeidet med å kombinere likheter mellom dokumentpar med lenkeanalyse har det oppstått noen uttrykk. Disse defineres under, og brukes videre i rapporten.

Lsdb (H_h)	Hele lenkestrukturdatabasen
Lsdb_sub (H_s)	En matrise inneholdende de dokumenter med lenker seg imellom som lagret i lsdb, og som alle inneholder søkeordene brukeren gir.
Lsdb_sparse	En matrise over dokumenter med lenker seg imellom som inneholder kun 20 % av de lenkene som er i lsdb
Lsdb_sparse_sub	En matrise inneholdende de dokumenter med lenker seg imellom som lagret i lsdbsparse, og som alle inneholder søkeordene brukeren gir.
simdb (S_h)	Database over likhetene mellom alle dokumentpar i samlingen
simdb_sub (S_s)	En matrise inneholdende de dokumentpar som alle inneholder søkeordene brukere har gitt
Madd	Addere sammen to matriser. Disse matrisene kan være av lik eller ulik størrelse.
Vadd	Addere sammen to vektorer som er funnet ved T-Rank analyse (nodescores).
T-Rank	T-Rank analyse av en matrise.
Enkel relevans	Denne rangeringen bruker ”p_values” i invertert indeks til å sortere dokumentene etter hvor relaterte de er til ordene i søket.

Tabell 6 - Tabell som viser uttrykksdefinisjoner som har oppstått underveis.

Kombinasjoner av disse metodene brukes videre i kapittelet. For eksempel beskrives T-Rank analyse av addisjon av en lenkestrukturmatrise og en likhetsmatrise på følgende måte:
T-Rank(Madd(lsdb + simdb)).
Addisjon av resultater av T-Rank analyse av en lenkestrukturmatrise og T-Rank analyse av en likhetsmatrise over en subgraf beskrives slik:
Vadd(T-Rank(lsdb) + T-Rank(simdb_sub)).

7.1.10 Hvilke metoder som er implementert

Av alle rangeringsmetodene i Figur 14 har vi valgt å implementere følgende. En stjerne (*) i lista under markerer at T-Rank brukes i sanntid for hvert søk. I de andre tilfellene brukes rangeringsmetoder hvor T-Rank har vært tatt i bruk på forhånd, eller andre metoder som for eksempel PageRank eller T-Rank(simdb). De metodene skrevet i *kursiv* lar seg påvirke av ”TR tweak” og ”SIM tweak”, som er nærmere beskrevet i slutten av kapittel 7.1.7. For oversiktens skyld lister vi under opp rangeringsmetodene i grupper og fargekodet i samsvar med fargekodingen i ”*Figur 13 - Oversikt over forskjellige forslag til metoder og ideer som kan brukes for å beregne viktighetspoeng for dokumenter med god eller dårlig lenkestruktur.*” Nummeret til hver av rangeringsmetodene under er det samme som numrene brukt i Figur 13 og Figur 14.

- **1**: T-Rank(lsdb)
- **1**: PageRank(lsdb)
- **3 / 13**: T-Rank(simdb)
- **4 / 14**: T-Rank(simdb_sub) *
- **10**: T-Rank(Madd(simdb + lsdb_sparse))
- ***9***: *T-Rank(Madd(simdb_sub + lsdb_sparse_sub))* *
- **7**: T-Rank(Madd(simdb + lsdb))
- ***5***: *T-Rank(Madd(simdb_sub + lsdb))* *
- ***8***: *T-Rank(Madd(simdb_sub + lsdb_sub))* *
- ***16***: *Vadd(T-Rank(lsdb) + T-Rank(simdb))*
- ***17***: *Vadd(T-Rank(lsdb) + T-Rank(simdb_sub))*
- ***19***: *Vadd(T-Rank(lsdb_sub) + T-Rank(simdb_sub))*
- ”Enkel” relevans
- Ingenting

Liste 1 – Oversikt over de metoder i Figur 13 som er implementert og mulig å teste på websiden B3000.glenn-erik.com

Disse metodene kan det være mulig å teste på websiden B3000.glenn-erik.com
I appendiks 10.1 finnes mer informasjon om hvordan websiden fungerer. I appendiks 10.5 finnes kildekoden til de forskjellige metodene i denne lista.

Som man ser av Liste 1 på forrige side er to metoder, 7 og 10, ikke kursive selv om de inneholder både lenkeanalyse og likhetsanalyse. Grunnen til at disse ikke lar seg påvirke av ”TR tweak” og ”SIM tweak” er fordi de er implementert som offline-rangeringsmetoder, og analysene som brukes for dette er gjort på forhånd og tar lang tid. Metode 5 tar også lang tid, men denne baserer seg på analyse av en subgraf i motsetning til 7 og 10, og følgelig kan det ikke beregnes rangeringsverdier for metode 5 på forhånd. Derfor har vi implementert mulighet for å la metode 5 påvirkes av ”TR tweak” og ”SIM tweak” selv om det tar ekstremt lang tid for hvert søk med denne metoden.

8 Testing og evaluering av resultater

Dette kapittelet tar for seg forskjellige aspekter ved testingen av komponentene som er laget. Til å begynne med viser vi en interessant illustrasjon av beregninger foretatt i Matlab på matrisa over dokumentlikheter (avsnitt 8.1). I neste avsnitt forklarer vi evalueringsmetoden brukt.

I avsnitt 8.3 forklarer vi detaljer rundt hvordan testingen er gjennomført, hvordan de skal leses og vi viser eksempler på søkeresultater brukt under testingen og resultatene av selve testingen. Dette avsnittet er videre delt opp i to underavsnitt. Vi starter avsnitt 8.3 med å diskutere generelt rundt hvordan testene er gjennomført, og nevner noen utfordringer ved testingen og hvordan vi har valgt å løse disse. Deretter, i avsnitt 8.3.1 viser vi eksempler på søk brukt under testingen av B3000 ved hjelp av skjermbilder tatt av resultatene B3000 gir. Dette delavsnittet forklarer hva resultatene betyr og hvordan de skal leses. I neste delavsnitt (8.3.2) viser vi resultatene av selve testingen av metodene vi har implementert. Disse resultatene er basert på sammenligning mellom et utvalg av rangeringsmetodene[31] til B3000 og den rangeringen Google gir for like søk.

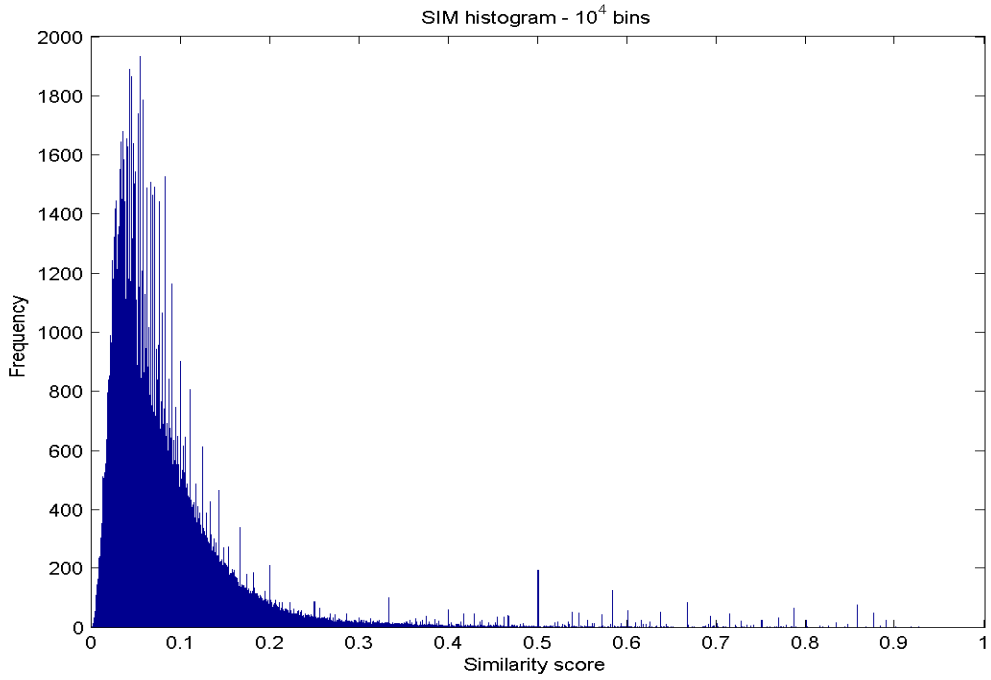
Videre evaluerer vi resultatene av testene i avsnitt 8.4.

8.1 Beregning av likhet mellom dokumentpar

Å beregne likhet mellom dokumentpar var noe av kjernen i oppgaven. Det var mye usikkerhet rundt hvordan ”

[31] Dette utvalget er vist i Liste 2 på 98.

Likhet mellom alle dokumentpar i en samling - Metode 3" i kapittel 4.6 skulle fungere i praksis. I løpet av kun 30 minutter hadde metoden funnet og beregnet likhetene mellom ca. 700.000 dokumentpar som var forventet å ha en likhet på over 1‰. I dette tilfellet satte vi både terskelverdi 1 og 2 til 10^{-3}. Siden dette gikk så fort, ønsket vi å teste hvor mange dokumenter som hadde en likhet i det hele tatt og satt begge terskelverdiene til 0. Dette resulterte i en kjøretid på ca. 1 time, og et funn av over 800.000 dokumentpar (ca 4‰ av totalt antall mulige par!). Under er en illustrasjon over fordelingen av antallet par ok deres likhet.



Figur 23 - Distribusjon av antallet dokumentpar i samlingen og deres likhet.

I utregningen er alle dokumentpar som ikke har noen likhet fjernet fra plottet. Det samme gjelder alle dokumentpar som har likhet 1. Resultatene for slike dokumenter ble en diskret verdi som tegnes som en veldig lang strek i histogrammet (for dokumenter med likhet 0). Det er interessant å sammenligne dette resultatet med hva vi hadde forventet i "*Figur 10 - Forventet antall dokumentpar kontra likheten mellom dem.*". Det er altså ikke slik at det er flest dokumentlikheter nærmest 0, men litt over. Ut fra histogrammet ser det ut som om vi kunne forkastet alle beregninger av dokumentpar som var forventet å ha en likhet mindre enn 0.2 – 0.3. Dette resultatet gir en god pekepinn på hvor man bør sette terskelverdiene neste gang man skal regne dokumentlikheter på en større samling.

Flere diagrammer over beregninger på datasettet over likheter mellom dokumentpar finnes i appendiks 10.4

8.2 Evalueringsmetoden brukt for måling av ytelse

Det er viktig å evaluere metodene våre for å få en pekepinn på hvor gode de er, og hvor bra vi har klart å oppnå målene i oppgaven. Det er mange forskjellige måter vi kan gjøre dette på. Noen av disse er "Precision & Recall", "The Harmonic Mean" og "The E Measure" (Baeza-Yates and Ribeiro-Neto 1999). Men vi ønsker en metode som sier noe om hvor god *rangeringen* av dokumentene er. Dette har vi ikke funnet noen god standard måte for hvordan gjøres. Vi har derfor valgt å lage vår egen mattematiske formel for bruk som målekriterium av ytelsen til de forskjellige metodene vi tester.

I denne formelen tas det utgangspunkt i gjennomsnittlig avvik blant B3000 sine ti første treff og Google sine *korrigerte* treff. Det er forventet at gjennomsnittlig avvik blant B3000 sine første ti treff sammenlignet med Googles resultater vil øke i samsvar med økende antall felles treff. For flere søk vil antallet felles treff lavt, og det gjennomsnittlige avviket vil derfor bli begrenset av dette. Det trengtes derfor en metode som viser et mål på ytelse uavhengig av antallet felles treff (gitt at antallet er over 10). Denne metoden er forklart under, og den tar i betraktning

1. *gj* = Gjennomsnittlig avvik blant B3000 sine topp 10 treff kontra Googles (korrigerte) rangering.
2. *N* = Antallet felles korrigerte treff mellom B3000 og Google.
3. g = forventet resultat om Google skulle rangere dokumentene *tilfeldig*.

g kommer man fram til på følgende måte:
f(x,N) skal gjenspeile en verdi for hvordan et treff, *i*, scorer om B3000 hadde gjort sin rangering tilfeldig. Og så skal $g := \frac{1}{10}\sum_{x=1}^{10} f(x,N)$ gi forventet verdi dersom B3000 hadde benyttet tilfeldig rangering. Hvis man forestiller seg en tilfeldig posisjon i Googles rangering blant alle *N* felles treff, så vil forventet gjennomsnittlig avvik for B3000 sitt første treff vil bli: $f := \frac{1}{N}\sum_{i=1}^{N}|1-i|$. Siden denne testen skal ta for seg gjennomsnittlig avvik blant B3000 sine 10 første treff sammenlignet med Google, og hvis man tenker seg at Google rangerer tilfeldig må det gjennomsnittlige avviket bli: $g(N) := \frac{1}{10}\sum_{x=1}^{10} f(x,N)$

Resultatet av denne summen blir: $f := \frac{N}{2} - 5 + \frac{33}{N}$. En fullstendig utredning finnes i appendiks 10.2. Endelig mål for et søkeord for en metode blir så: *s = gj/g(N)*, dermed korrigeres det for begrenset avvik som skyldes begrenset antall felles treff (liten N), siden for liten N, g(N) er også liten.

8.3 Sammenligning av rangeringen til B3000 og Google

For søkene foretatt i denne testen er resultatene som søkemotoren Google gir, for like søk i B3000, sammenligningskriteriumet. Resultatene er ikke forventet å være like bra som de Google gir, ettersom det ikke er brukt noe tekstrelevans i rangeringsmetodene til B3000[32]. Google bruker store ressurser på tekstrelevansanalyse (Prakash; Yaltaghian and Chignell). Om B3000, utelukkende basert på likhetsberegninger og analyser av det, eller lenkeanalyse, eller en kombinasjon klarer å rangere og skille ut de viktigste dokumentene blant mange, vil vi si oss fornøyd.

Innad i Wikipedia vil kanskje avansert tekstrelevans alene være nok for å rangere dokumentene på en god måte. Men lenkeanalyse vil allikevel være med på å få viktige dokumenter høyt i rangeringene. Det er viktig å være klar over at lenkeanalysen Google bruker for å rangere Wikipedia dokumentene, selv ved begrenset søk, antakeligvis inneholder mange flere lenker enn det som har vært tilgjengelig i denne oppgaven. B3000 har kun lenkene funnet internt blant Wikipedia samlingen, mens Google antagelig i tillegg tar eksterne lenker funnet i betraktning.

En utfordring med testingen er at Google ikke tillater mer enn 1000 automatiske søk per døgn, for forsknings formål, gjennom GoogleAPI. Hvert søk kan i GoogleAPI maks resultere i 10 treff. I denne testen er det ønskelig med så mange treffresultater som mulig, helst alle treffene Google kan gi, for å få et sikkert og nøyaktig sammenligningskriterium. Ofte har Google flere hundre treff for de søkene vi har foretatt, og B3000 må da ”bruke opp” flere titalls søk i GoogleAPI per enkelte søk vi foretar i B3000. Vi har derfor valgt å ikke bruke GoogleAPI, men heller laget et script som foretar et vanlig søk i Google ved å simulere en vanlig bruker. Dette har den fordelen at vi kan be om å få returnert 100 treff per søk, og det holder derfor med 1-3 søk i Google per søk som foretas i B3000, avhengig av søkefrasen valgt. Ulempen ved denne metoden er at Google, om en bruker søker veldig mange ganger i B3000 på kort tid, vil sperre IP adressen til B3000 slik at flere søk ikke kan foretas på en stund. For hvert enkelt søk i B3000 skal det i denne testingen foretas en sammenligning av flere av de forskjellige rangeringsmetodene til B3000 mot rangeringen til Google. B3000 må pga. dette gjennomføre det samme søket flere ganger, og vi har derfor valgt å cache Googles søkeresultater hver gang en bruker foretar et nytt søk.

En annen utfordring ved testingen er at Wikipedia er veldig streng på hvordan crawlere skal oppføre seg når de besøker sidene deres. Dette har resultert i at Google har relativt

[32] Ingen av metodene som testes innebærer noe sanntids tekstrelevansanalyse. Men en ekstra rangeringsmetode som ble implementer baserer seg på ”enkel” relevansanalyse. Denne tar ikke i betraktning verken lenker eller dokumentlikheter.

dårlig oversikt over de dokumentene Wikipedia inneholder. Vi, som har lastet ned alle dokumentene som en database dump, har en komplett oversikt over hvordan den norske Wikipedia samlingen var da den ble lastet ned. Dette gjorde vi 9. mars 2005, over tre måneder før testingen startet. Grunnet dette mangler Google mange av de dokumentene B3000 tar med i betraktning for rangering. På den annen side mangler B3000 *noen* av dokumentene Google tar i betraktning (de dokumentene som Google faktisk har funnet, som ble laget etter at vi lastet ned Wikipedia databasen). Konsekvensene av dette er at rangeringen inneholder flere ”hull” i sammenligningen mellom resultatene til B3000 og resultatene til Google.

8.3.1 Fremstilling av søkeresultatene brukt under testingen

Dette delavsnittet forklarer hva testresultatene betyr og hvordan de skal leses. Vi viser her en del skjermbilder av B3000 for søk brukt under testingen. For et søk regnes det ut flere forskjellige verdier. Disse verdiene kan gjenspeile informasjon basert på hele eller deler av resultatet. Det kan være informasjon som sier noe om B3000, eller Google eller felles for begge, for hvert enkelt treff. Detaljer rund disse resultatene beskrives videre i dette kapittelet. Figurene i dette kapittelet er skjermbilder av B3000 for søk på ”Matematikk” rangert med Metode 8 (som er nærmere beskrevet i kapittel 7.1.10). Dette er samme Metode som er vist som nummer 8 i Figur 13 og Figur 14.

Resultatet ved et søk gir brukeren 2 tabeller rett ved siden av hverandre. Den første tabellen, til venstre som hovedsakelig er grå og hvit, er rangert etter hvordan **B3000** mener Wikipediadokumentene bør være rangert i henhold til analysemetoden brukeren har valgt. Kolonnen som inneholder hvert dokuments score som er brukt for rangeringen, er markert med mørkerødt. Denne tabellen inneholder en del annen informasjon, bl.a. resultatene av de analyser det var mulig å foreta på forhånd (offline). Dette gjelder score-verdiene: ”T-Rank”, ”PageRank”, ”Similaritets Score”, ”SIM + LSDB Score”, ”SIM + LSDB Sparse”. Alle disse analysene er foretatt over en graf som inkluderer hele samlingen. Dette gir brukeren en mulighet til å sammenligne resultatene mellom de forskjellige rangeringsmetodene innad i B3000 for hvert søk.

For mange av metodene brukeren kan velge, må rangeringsanalysene foretas i sanntid. Dette vises ved at det oppstår en ekstra kolonne som heter ”REALTIME DokScore”. Eksempel på et slikt tilfelle kan være om brukeren velger en rangering som krever analyse av en subgraf.

Tabellen til høyre, som hovedsakelig er grønn, viser rangeringen av de Wikipediadokumentene slik **Google** mener det bør være, for det samme søket (figur som viser dette på neste side).

B3000 Treff #	Google Treff #	Felles Treff #	Dokumentets **tittel** og link til wikipedia	PageRank Score	T-Rank Score	Similaritets Score	SIM+LSDB Score	SIM+LSDB Sparse	**REALTIME DokScore**	Like dok.	rå txt	Google rangerer slik (ukorrigert / korrigert / tittel / felles / b3000)				
1	13	**1**	Niels_Henrik_Abel	0.021171	0.522015	0.457343	0.476687	0.3654	10	like	txt	1	*1*	Matematikk	**3**	5
2	-	-	Abelprisen	0.004865	0.181382	0.522496	0.198114	0.28198	6.50343	like	txt	2	*2*	Diskret_matematikk	**5**	7
3	-	-	Sophus_Lie	0.007937	0.088453	0.377843	0.167827	0.290343	6.103007	like	txt	3	*3*	Gruppe_(matematikk)	**6**	9
4	9	**2**	Grafteori	0.000367	0.01321	0.476081	0.082774	0.237807	2.581571	like	txt	4	*4*	Familie_(matematikk)	**20**	55
5	1	**3**	Matematikk	0.002539	0.050996	0.344811	0.150665	0.289228	2.265493	like	txt	5	*5*	Fakultet_(matematikk)	**15**	33
6	30	**4**	Geometri	0.003509	0.013867	0.50441	0.084889	0.273874	1.934577	like	txt	6	*6*	Norges_teknisk-naturvitensk...	**13**	31
7	2	**5**	Diskret_matematikk	0.000406	0.014297	0.216813	0.049978	0.109589	1.861231	like	txt	7	*7*	Statistikk	**23**	66
8	-	-	Informasjonsteori	0.000325	0.011939	0.287222	0.054247	0.143826	1.699549	like	txt	8	*8*	Ring	**9**	19
9	3	**6**	Gruppe_(matematikk)	3e-05	0.004215	0.606012	0.073929	0.303987	1.615158	like	txt	9	*9*	Grafteori	**2**	4
10	-	-	Abstrakt_algebra	0.000228	0.009562	0.124096	0.032079	0.065763	1.434233	like	txt	10	*10*	Hovedside/Tysk	**30**	78
11	16	**7**	Leonhard_Euler	0.010295	0.072998	0.21546	0.128465	0.137956	1.383626	like	txt	11	*11*	Vektor	**19**	51
12	-	-	Christiaan_Huygens	0.003025	0.060676	0.407482	0.172592	0.3026	1.088693	like	txt	12	*12*	Grad	**29**	77
13	-	-	Carl_Friedrich_Gauss	0.001508	0.033875	0.704444	0.172673	0.418076	1.023331	like	txt	13	*13*	Niels_Henrik_Abel	**1**	1
14	-	-	Tellbar	9e-06	0.003556	0.527512	0.062353	0.265541	0.959944	like	txt	14	-	Minutt	-	-
15	-	-	Bijeksjon	9e-06	0.003556	0.118946	0.014553	0.059945	0.914323	like	txt	15	*14*	Reinhard_Selten	**10**	20
16	17	**8**	Trigonometri	2.7e-05	0.00395	0.166772	0.025303	0.087502	0.756857	like	txt	16	*15*	Leonhard_Euler	**7**	11
17	-	-	Permutasjon	9e-06	0.003556	0.189011	0.022624	0.094031	0.714469	like	txt	17	*16*	Trigonometri	**8**	16
18	-	-	Mengde	0.000148	0.007204	0.757036	0.11282	0.389242	0.677803	like	txt	18	*17*	Koordinatsystem	**16**	34
19	8	**9**	Ring	9e-06	0.003556	0.239297	0.029019	0.120946	0.663701	like	txt	19	*18*	Edmund_Husserl	**25**	70
20	15	**10**	Reinhard_Selten	0.016708	0.193729	0.374617	0.165717	0.285875	0.613866	like	txt	20	*19*	Georg_Milbradt	**35**	92
21	-	-	Naturvitenskap	0.001282	0.017359	0.356433	0.068065	0.184996	0.472267	like	txt	21	*20*	Christian_Andreas_Doppler	**28**	75
22	-	-	Nikolaus_Copernicus	0.020099	0.277821	0.723262	0.380926	0.432077	0.463884	like	txt	22	*21*	Georg_Simon_Ohm	**41**	107
23	-	-	Babylonia	0.001801	0.017304	0.828444	0.153773	0.571113	0.405432	like	txt	23	*22*	Gresk_alfabet	**37**	100
24	-	-	Louis_Braille	0.000583	0.042077	0.429692	0.097305	0.222412	0.389629	like	txt	24	*23*	Melitta_Schenk_von_Stauffenberg	**33**	84
25	63	**11**	Stamtre_(evolusjon)	1.3e-05	0.003688	3.417415	0.388295	1.68363	0.376719	like	txt	25	*24*	Napoleon_Bonaparte	**24**	68
26	-	-	Paul_Keres	9e-06	0.003556	0.425335	0.065809	0.226527	0.35914	like	txt	26	*25*	Josep_Borrell_Fontelles	**32**	83
27	-	-	Reelt_tall	0.001374	0.024795	0.256559	0.07466	0.166625	0.346114	like	txt	27	-	Platon	-	-
28	-	-	Aksiom	9e-06	0.003556	0.174197	0.022956	0.089598	0.276926	like	txt	28	*26*	Informasjonsvitenskap	**17**	35
29	-	-	Rasjonalt_tall	0.001374	0.024795	0.298818	0.082904	0.22194	0.250952	like	txt	29	-	Språk	-	-
30	45	**12**	Fermats_teorem	9e-06	0.003556	0.355898	0.044183	0.178693	0.250411	like	txt	30	*27*	Geometri	**4**	6

Figur 24 – (Skjermbilde fra B3000) Utsnitt av de topp 30 resultatene for søk på "Matematikk" i B3000 ved bruk av Metode 8. Figuren viser sammenligninger med de resultater Google gir.

De tre første kolonnene i tabellen til venstre viser forskjellige treff data. Den første, grå kolonnen, viser treff slik B3000 rangerer dem. Den andre, gule kolonnen, viser den ukorrigerte rangeringsposisjonen for det samme treffet slik Google mener det bør være. Den tredje kolonnen viser nummeret til de treffene som er felles. Ingen av de treffene Google har funnet, som ikke B3000 har funnet, er vist i tabellen til venstre.

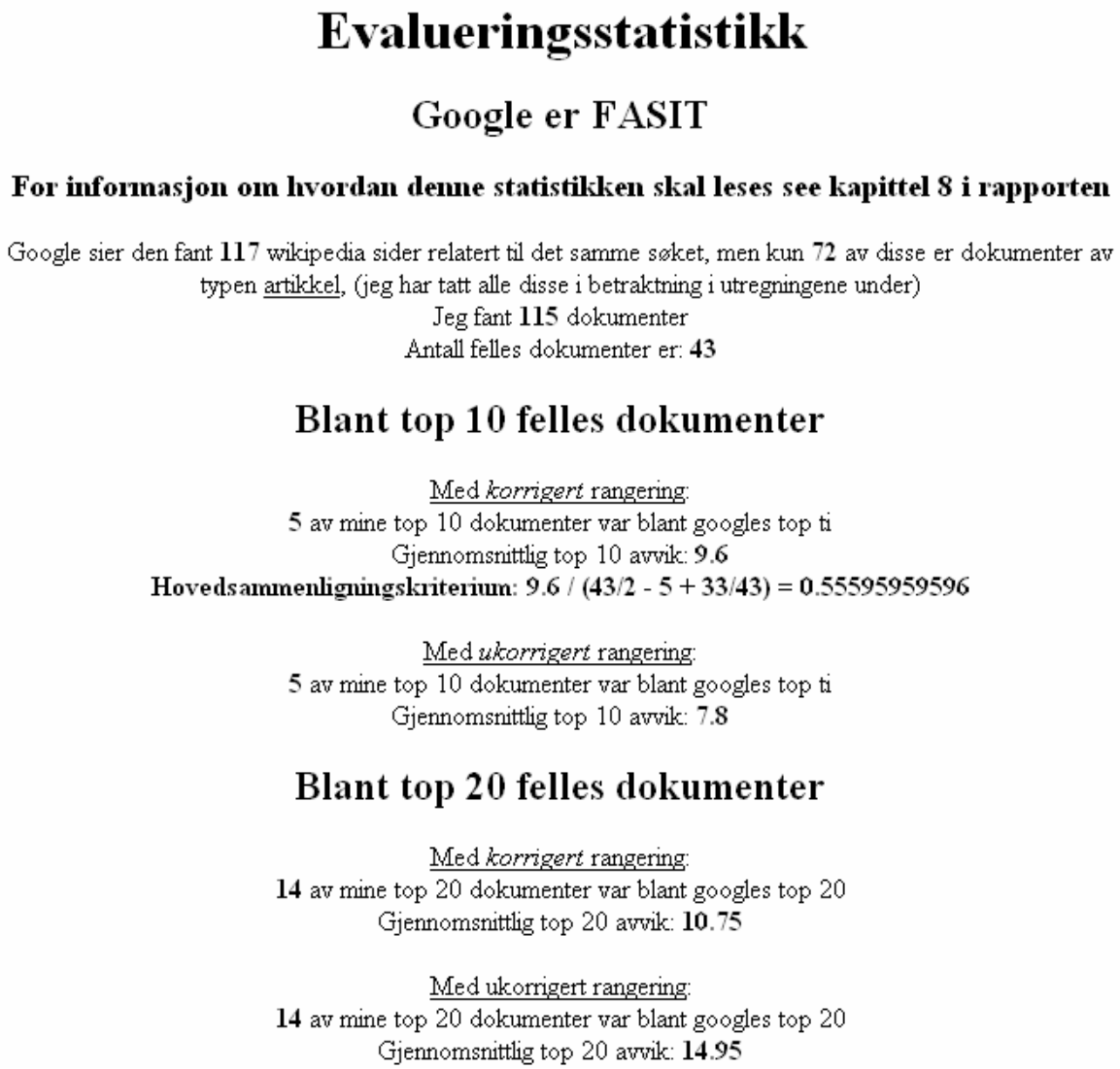
I den grønne tabellen[33], til høyre, vises korrigerte og ukorrigerte treff rangert av Google, dokumentets tittel, felles treff og B3000 sitt treffnummer. Et ukorrigert treff er det treffnummeret som Google eller B3000 gir et dokument uten hensyn til om det samme dokumentet finnes i hhv. B3000 eller Google. Korrigerte treff er de treff som er felles for både B3000 og Google, dvs. snittet mellom de to treffsettene. De korrigerte treffene i den oransje kolonnen er de viktigste, og gir mest mening å sammenligne og evaluere.

For å gjøre det mest mulig oversiktlig er de tilsvarende kolonnene med treff data i den høyre og den venstre tabellen fargelagt med like farger for å vise hvilke av kolonnene som gjenspeiler det samme som i den andre tabellen. Ved bruk av denne fargekodingen kan man lettere finne det samme dokumentets rangering i de to forskjellige tabellene.

Man kan tydelig se at det er en del ”hull” i kolonnene ”Google Treff #” og ”Felles Treff #” i tabellen til venstre. Dette er tilfeller hvor B3000 har funnet treff som Google ikke har med i sin rangering fordi den ikke har indeksert eller funnet dokumentene. Videre kan man se i den oransje kolonnen ”korrigerte treff” i tabellen til høyre at det er noen hull. Dette er tilfeller hvor Google har funnet dokumenter som ikke er inkludert i den versjonen av Wikipediadatabasen vi har brukt.

Nederst på resultatsiden finner man analyser som omfatter hele eller deler av de enkelte treffdataene (Figur 25 på neste side viser resultater for søk ”Matematikk”). Dette omfatter generell statistikk om hvor mange dokumenter som var felles, hvor mange Google fant, hvor mange B3000 fant, og lignende. Deretter vises det viktigste analyseresultatet som ligger til grunn for ytelsesmålingen B3000. Dette er **gjennomsnittlig avvik blant B3000 sine topp 10 dokumenter sammenlignet med Google**. Det samme måles også for topp 20 dokumenter, og for alle dokumentene. I tillegg finner man nyttig informasjon om hvor mange av B3000 sine topp 10 dokumenter som var blant Googles topp ti dokumenter. Slik informasjon er mulig å lese ut fra tabellene vist i Figur 24, men det er tungvint og kan være vanskelig.

[33] Blå rader i den høyre tabellen representerer dette var et dokument som Google linket til gjennom et ”REDIRECT” dokument i Wikipeida. Se kapittel 7.1.3 og kildekode vedlagt på CD for hvordan dokumenter av typen ”REDIRECT” behandles i B3000.



Evalueringsstatistikk

Google er FASIT

For informasjon om hvordan denne statistikken skal leses see kapittel 8 i rapporten

Google sier den fant **117** wikipedia sider relatert til det samme søket, men kun **72** av disse er dokumenter av typen artikkel, (jeg har tatt alle disse i betraktning i utregningene under)
Jeg fant **115** dokumenter
Antall felles dokumenter er: **43**

Blant top 10 felles dokumenter

Med *korrigert* rangering:
5 av mine top 10 dokumenter var blant googles top ti
Gjennomsnittlig top 10 avvik: **9.6**
Hovedsammenligningskriterium: 9.6 / (43/2 - 5 + 33/43) = 0.55595959596

Med *ukorrigert* rangering:
5 av mine top 10 dokumenter var blant googles top ti
Gjennomsnittlig top 10 avvik: **7.8**

Blant top 20 felles dokumenter

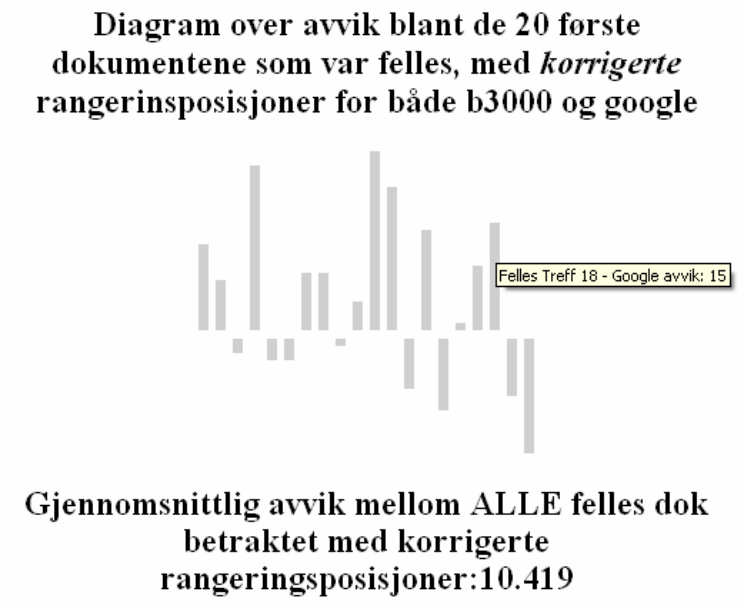
Med *korrigert* rangering:
14 av mine top 20 dokumenter var blant googles top 20
Gjennomsnittlig top 20 avvik: **10.75**

Med ukorrigert rangering:
14 av mine top 20 dokumenter var blant googles top 20
Gjennomsnittlig top 20 avvik: **14.95**

Figur 25 – (Skjermbilde fra B3000) Evalueringsstatistikk for søk på "Matematikk" i B3000 ved bruk av Metode 8 sammenlignet med resultater fra Google.

Man kan se at hovedsammenligningskriteriumet stemmer overens med verdien listet for metode 8, ved søk på ”Matematikk” i Tabell 7 under.

I tillegg til disse dataene vil man finne diagrammer som gir en visuell fremstilling av avvikene blant de topp 20 treffene mellom B3000 og Google. Det er to slike diagrammer. Den grå, viser resultatene tatt de *korrigerte* treff dataene i betraktning. Den andre, mørkerøde, viser resultatene tatt de *ukorrigerte* treff dataene i betraktning. Flytter man musa over en av stolpene i diagrammet vil kan kunne se hvilket treff stolpen gjelder, og hvor stort avviket er. Er avviket positivt (over midten av diagrammet) betyr dette at B3000 rangerer dokumentet høyere enn Google. Er avviket negativt (under midten av diagrammet) betyr dette at B3000 rangerte dokumentet lavere enn Google.



Figur 26 – (Skjermbilde fra B3000) Evalueringsstatistikk vist for søk "Matemtatikk" i B3000 ved bruk av Metode 8

Stolpediagrammene gir mye nyttig informasjon på en enkel måte ved at man raskt ser om det er enkelte dokumenter som skiller seg ut. Er gjennomsnittlig avvik høyt, og stolpediagrammet jevnt, betyr dette at rangeringen er dårlig. Er gjennomsnittlig avvikt høyt, men det viser seg at kun en av stolpene skiller seg ut, kan rangeringen fortsatt være god.

8.3.2 Gjennomføring og resultater

I denne testen foretas søk med et utvalg av de forskjellige metodene implementert under prosjektet. Alle de gule boksene i Figur 14 testes, med unntak av metode 5 da den er for tidkrevende. Operasjonene brukt på matrisene lsdb, simdb, lsdb_sub og simdb_sub er nærmere forklart i kapittel 7.1.9. De avgjørende søkemetodene brukt for å rangere dokumentene i B3000 i denne testingen er:

- Metode 1: *T-Rank(lsdb)*. Resultat basert utelukkende på lenkeanalyse av lsdb med T-Rank.

- Metode 1: *PageRank(lsdb)*. Resultat basert utelukkende på lenkeanalyse av lsdb med PageRank.

- Metode 17: *Vadd(T-Rank(lsdb) + T-Rank(simdb_sub))*. Addert resultat av T-Rank analyse av simdb og T-Rank analyse av lsdb hver for seg. Resultat basert på rangeringsmetode 17 i Figur 14.

- Metode 8: *T-Rank(Madd(simdb_sub + lsdb_sub))*. T-Rank analyse av matrisene simdb_sub og lsdb_sub addert sammen. Resultat basert på rangeringsmetode 8 i Figur 14.

- Metode 9: *T-Rank(Madd(lsdb_sparse_sub + simdb_sub))*. Analyse av lsdb_sparse_sub og sim_db_sub. Resultat basert på rangeringsmetode 9 i Figur 14, med 70% av lenkene tilfeldig fjernet. Resultatene vist i tabellen over er basert på gjennomsnittsresultatet av 10 søk foretatt.

- Metode 4 og 14: *T-Rank(simdb_sub)*. T-Rank analyse av simdb_sub. Resultat basert på rangeringsmetode 4 og 14 i Figur 14. (Metodene 4 og 14 er sammen tilfelle).

- "Enkel Relevans". Resultat basert på kun bruk av enkel relevans matching (for sammenligning med de andre resultatene).

- "Ingenting". Tilfeldig rangering. Ikke uventet er reultatscoren til *s* veldig nært 1.0.

Liste 2 - Liste over avgjørende søkemetoder brukt for å rangere dokumentene under testing.

"TR tweak" og "SIM tweak" (nærmere omtalt i slutten av kapittel 7.1.7) settes begge til 1.0. For hvert enkelt søk vil B3000 gi en felles poengsum som mål på hvor god rangeringen er. s = gj/N er **gjennomsnittlig avvik blant topp ti treff i B3000 kontra Google, delt på forventet resultat ved tilfeldig rangering i B3000**. Denne metoden er blitt brukt for å flate ut poengsummen for søk med resultater med mange felles dokumenter og søk med resultater uten så mange felles dokumenter. Det avgjørende målet på ytelsen til B3000 baseres på gjennomsnittet av poengene s gitt for hvert av de forskjellige søkefrasene.

N: Antallet felles treff mellom B3000 og Google.
gj: Gjennomsnittlig avvik mellom alle topp 10 treff i B3000 sett i forhold til Google.
s: Avgjørende score: **gj/(N/2 – 5 + 33/N)**[34]

I metode 9, hvor 70 % av lenkene blir tilfeldig slettet for hvert søk, er hver av verdiene for *gj* resultatet av gjennomsnittet av ti forskjellige søk. Dette fordi resultatet av rangeringen for metode 9 vil variere avhengig av hvilke lenker som slettes. For de andre metodene er *gj* resultatet av ètt deterministisk søk.

Vi har valgt søkefraser som gir tjue eller flere felles treff. I tillegg har vi med hensikt valgt noen av søkefrasene til å være tvetydige for å se hvordan likhetsanalyse scorer i slike tilfeller. Vi har foretatt 24 søk, med følgende søkefraser og resultater (tabell på neste side):

[34] Forklaring av denne formelen beskrives avsnitt 8.2, en fullstendig mattematisk utredning finnes appendiks 10.3.

Rangeringsmetode►		T-Rank		PR		17		8		9 (70%)		4 / 14		Relevans		Ingenting	
▼Søkeord▼	*N*	*gj*	*S*	*gj*	*s*	*gj*	*s*	*gj*	*s*	*gj*	*s*	*gj*	*s*	*gj*	*s*	*gj*	*s*
Oslo	275	99,5	0,750	98,1	0,740	83	0,626	97,9	0,738	64,5	0,486	56,5	0,426	83	0,626	122	0,918
Matematikk	43	15,7	0,909	18,1	1,048	14,8	0,857	9,6	0,556	11,6	0,671	11,9	0,689	19,4	1,124	15	0,869
Krig	142	39,6	0,598	40,2	0,607	20,4	0,308	18,4	0,278	19	0,286	22,4	0,338	75,8	1,144	55,9	0,844
Norge Oslo	122	25,4	0,451	22,3	0,396	25,7	0,457	28,1	0,499	22,6	0,401	20,9	0,371	53,9	0,958	60,5	1,075
Trondheim	85	43,8	1,156	44,5	1,175	29,2	0,771	22,8	0,602	28,8	0,759	28,4	0,750	28,2	0,744	33,8	0,892
George Bush	26	11,8	1,273	12,2	1,316	11,7	1,262	10	1,079	9,83	1,060	10,7	1,154	7,1	0,766	11	1,187
Internett	60	30,1	1,178	30,8	1,205	23,4	0,916	14,1	0,552	16,7	0,654	18,3	0,716	34,5	1,350	25,9	1,014
Windows	26	9,1	0,982	8,4	0,906	13,5	1,456	9,8	1,057	9,91	1,069	11,4	1,230	7,6	0,820	5,3	0,572
Astronomi	20	4,9	0,737	5	0,752	8,2	1,233	7,7	1,158	8,37	1,259	8,1	1,218	7	1,053	4,9	0,737
Fiske	21	7,8	1,103	8	1,131	7,2	1,018	8,2	1,160	7,66	1,083	7,2	1,018	9	1,273	9,3	1,315
Jakt	21	5,1	0,721	6	0,848	7,4	1,046	9,2	1,301	8,74	1,236	8,6	1,216	8,9	1,259	7,2	1,018
Mat	46	12,6	0,673	15,4	0,823	18,4	0,983	18,6	0,994	19,3	1,032	21	1,122	23,8	1,272	23	1,229
Økonomi	107	58,6	1,201	58,2	1,192	16,6	0,340	27,1	0,555	18,5	0,378	22,7	0,465	65,5	1,342	25,2	0,516
Penger	49	16,5	0,818	16,2	0,803	18,7	0,927	19,1	0,947	19	0,941	18,9	0,937	25,6	1,269	11	0,545
Hus	86	47,4	1,235	44,9	1,170	47	1,224	41,2	1,073	34,7	0,904	34,7	0,904	33,7	0,878	40,4	1,053
Rike	135	34,5	0,550	26,4	0,421	36,5	0,582	35,9	0,572	36	0,574	37,2	0,593	79	1,259	52,9	0,843
Vann	223	41,6	0,390	42,6	0,399	53,1	0,498	54,1	0,507	74,7	0,700	83,7	0,785	119	1,119	85,7	0,804
Bok	78	28,9	0,840	21,4	0,622	39,5	1,147	28,2	0,819	40,1	1,164	42	1,220	41,6	1,208	33,8	0,982
Natur	33	12,8	1,024	14,8	1,184	9,3	0,744	11,7	0,936	9,04	0,723	9,5	0,760	12,7	1,016	13,6	1,088
Dyr	46	19,3	1,031	21,9	1,170	16,2	0,866	16,4	0,876	16,1	0,857	15,8	0,844	24	1,282	19,4	1,036
Militær	55	11,7	0,506	11,4	0,494	13,7	0,593	15	0,649	14,2	0,614	14,4	0,623	35,5	1,537	20,4	0,883
Forsvar	24	7,8	0,931	8,2	0,979	10,6	1,266	11,3	1,349	11,2	1,333	11	1,313	8,1	0,967	10,5	1,254
Forsvaret	60	20,6	0,806	20,4	0,798	26,6	1,041	24,9	0,975	26,2	1,027	26,4	1,033	33,1	1,295	25,2	0,986
Metode	24	8,4	1,003	10	1,194	12,1	1,445	12,4	1,481	12	1,438	12,1	1,445	8,1	0,967	7,2	0,860
Gjennomsnitt:		25,6	0,869	25,2	0,891	23,5	0,900	23,0	0,863	**22,4**	**0,86**	23,1	0,882	35,2	1,105	30,0	0,938

Tabell 7 – HOVEDRESULTATER AV TESTING. Tabellen viser resultater (ytelsesverdi "s" for en metode) av testing for 24 søkefraser med 8 forskjellige rangeringsmetoder sammenlignet med Googles rangering av de samme dokumentene.

Rangeres metodene brukt basert på deres avgjørende ytelsesmål (s) oppnås følgende liste:

- Metode 9
- Metode 8
- Metode 1 (T-Rank)
- Metode 4 / 14
- Metode 1 (PageRank)
- Metode 17
- Ingenting
- ”Relevans” (enkel)

Liste 3 - Liste over rangeringsmetodene i B3000 oppført fra beste til dårligst ytende.

8.4 Evaluering av resultatene

Med det faktum i minnet at ytelsesverdi 1.0 er totalt tilfeldig rangering, mens 0.0 er helt lik rangering som fasiten, kan man dra den konklusjonen at avansert tekstrelevansanlyse er viktig for å oppnå perfekt resultat. Men det er viktig å ha i minnet at metodene brukt er ment som et supplement til tekstrelevans i en ferdig søkemotor, og det er tydelig at de kan brukes som hjelpemiddel til dette. Det er veldig interessant å se at similaritetsanalyse alene (Metode 4 / 14) yter omtrent like bra som kun lenkeanalyse (kun 13/1000 forskjell mellom Metode 4 / 14 og T-Rank)! Grunnet dette vil vi påstå at hovedmålet i oppgaven er nådd, ved at man kan bruke similaritetsanalyse som et like bra suppleringsgrunnlag for rangering som lenkeanalyse! Basert på disse testene vil vi si at similaritetsanalyse kan fungere som erstatning for lenkeanalyse i domener uten egne lenker. Og resultatet blir enda bedre om man supplerer similaritetsanalyse med en evt. svak lenkestruktur man kan finne. Similaritetsanalyse supplert med en svak lenkestruktur (Metode 9) rangerte 9/1000 bedre en kun lenkeanalyse med T-Rank i testen over.

Videre er det interessant å se at T-Rank faktisk gjør det bedre enn Googles egen lenkeanalysemetode PageRank. Forskjellen er kun 22/1000 i T-Ranks favør, dette kan være tilfeldig for denne testen, men det kan også bety at T-Rank faktisk egner seg bedre for lenkeanalyse i domener uten spam. Vi føler det er riktig å kunne dra denne konklusjonen ettersom tekstrelevansanalyse tydeligvis er den dominerende analysefaktoren for å oppnå gode resultater i denne testen. Men det er selvfølgelig viktig å ha i minnet at Googles rangering *kan* være en del påvirket av lenker funnet utenfor domenet brukt i testen. Hvor mye denne påvirkningen veier i forhold til selve tekstrelevansanalysen til Google er vanskelig å si. Men ved inspeksjon av tabellen til høyre i Figur 24 (Googles rangering) gir det en sterk følelse av at tekstrelevansanalyse veier tungt i rangeringen av treffene, ettersom alle de fem første treffene inneholder ordet

”Matematikk” i dokumentets tittel. Det er lite sannsynlig at grunnen til disse plasseringene er hovedsakelig fordi flere eksterne sider peker til ”Gruppe_(Matematikk)” og ”Familie_(Matematikk)” (3. og 4. plass) enn eksterne sider som peker til for eksempel ”NTNU” og ”Vektor” (6. og 11. plass).

Presisjonen i ytelsesmålet på resultatene i disse testene avviker med ca 100/1000 – 150/1000 fra tilfeldig rangering. I disse testene er det tatt 24 søkeord i betraktning, og for hvert av søkeordene er det mellom 20 – 300 felles treff. Antagelig er dette gode nok kriterier til å kunne si at vi ikke har vært ”heldig” med resultatene i denne oppgaven. Det vil allikevel være interessant å foreta mer avanserte tester basert på et mye større domene og mange flere søkeord for å få en pekepinn på hvor generelle og korrekte svarene til testene i denne oppgaven er.

Tabell 7 støtter teoriene om at lenkeanalyse resulterer i bedre rangering enn ingenting. Men man ser også ut fra tabellen at analyse av similaritetsberegninger mellom dokumentpar kan brukes til å rangere dokumentene logisk, og til å oppnå bedre resultater enn man ville oppnådd med primitiv tekstanalyse eller uten noen analyse i det hele tatt! Analyse utelukkende av likheter mellom dokumenter er den metoden som resulterer i tredje beste rangering. Supplerer man denne analysen med en svak lenkestruktur oppnår man den *beste* rangeringen av alle metodene som er tatt med i testen. Dette er *helt uten relevansanalyse* tatt i betraktning, og virker derfor som en meget lovende metode for å rangere dokumenter i domener med dårlig lenkestruktur. Det er interessant å se at likhetsanalyse faktisk gjør det like bra om metoden suppleres med en svak lenkestruktur som med en rik lenkestruktur. Metoden som analyserte resultatet av likhetsberegninger kombinert med en rik lenkestruktur resulterte i den nest beste rangeringen. Forskjellen er ikke stor, men dette kan kanskje tolkes som at en rik lenkestruktur kan virke noe forstyrrende sammen med likhetsanalyse, men at en svak lenkestruktur er med på å dra likhetsanalysen i ”riktig retning”. Det kan i dette tilfellet være interessant å se resultatene ved bruk av forskjellige verdier for ”TR tweak” og ”SIM tweak” i framtidige tester.

Resultatene i Tabell 7 underbygger påstanden om at man trenger avanserte relevansanalysemetoder for å oppnå bra resultater utelukkende med relevansanalyse, og at relevans er vanskelig å bruke. Rangeringsmetoden som B3000 bruker for å rangere etter relevans resulterte i den aller verste rangeringen. Denne rangeringen var enda dårligere enn i tilfellet hvor dokumentene ble rangert tilfeldig. I tilfellet med tilfeldig rangering ble resultatet, ikke uventet, omtrent lik 1.

Generelt kan det nevnes en erfaring vi har hatt når vi har søkt i B3000. Det virker som om de metoder som tar i betraktning likheter fungerer godt som hjelpemiddel til å nedprioritere dokumenter som er store og generelle, og som ofte scorer høyt om man rangerer kun basert på lenkeanalyse. Dette dreier seg typisk om dokumenter som summerer opp saker og ting som har skjedd en spesiell dato, måned eller år. For eksempel handler dokumentet med tittel ”1994” om alle store hendelser og personer som utmerket seg dette året. Grunnen til at slike dokumenter scorer høyt utelukkende ved bruk av lenkeanalyse er fordi veldig mange andre dokumenter i samlingen peker til dem. Mye av det som er skrevet om i Wikipedias dokumenter er saker som skjedde en gang, eller hendte et spesielt sted, og disse dokumentene lenker alltid til gangen og steder det skjedde, som ofte et annet dokument i Wikipedia handler om. For eksempel returnerer B3000 et dokument med tittel ”1960” på toppen for søk på ”matematikk” om vi velger rangeringsmetode ”T-Rank”. For søk med ordet ”fotball” returnerer B3000 18 dokumenter relatert til fotball før det første dokumentet med årstall i tittelen (”2000”) om vi bruker rangeringsmetode 8. For søk på ordet fotball utelukkende med lenkeanalyseringsmetoden T-Rank som rangeringsmetode er de 37 første treffene årstall!

9 Konklusjon

Konklusjonen i denne oppgaven baseres på resultatet av testene, og de erfaringer vi har høstet under gjennomføringen av testene. Vi ønsket å se hvor gode rangeringsresultater det er mulig å oppnå om man bruker lenkeanalysemetoder til å rangere søketreff i domener med få lenker. For å være i stand til å gjøre dette har vi kommet med forslag til alternative dokumentanalysemetoder som kan brukes sammen med tradisjonelle lenkeanalysemetoder. Disse dokumentanalysemetodene baserer seg på å beregne likheter mellom dokumentpar i samlingen. Resultatet lar seg analyseres, alene eller sammen med (de få) lenker funnet blant dokumentene, av tradisjonelle lenkeanalysemetode. Metodene foreslått har vi så bygget og satt sammen til en fungerende søkemotor.

Før vi startet testingen av systemet valgte vi ut de metodene implementert som vi fant mest interessante og lovende. Dette utvalget har vi diskutert i oppgaven og er basert på hvilke metoder vi mener passer godt for sammenligning seg imellom og for å demonstrere de forskjellige hovedtypene vi har for å kombinere likhetsanalyse med lenkeanalyse. Testkapittelet illustrerer rangeringsresultater ved bruk av en av metodene implementert[35] i oppgaven som tar i betraktning lenkeanalyse og likhetsanalyse. Denne metoden har vi valg til å illustrere B3000 fordi den er en av de mest sentrale metodene utviklet og fordi vi fant den til å gi et resultat som passet godt for illustrasjon.

Hovedresultatet av testingen tar i betraktning rangeringsresultater ved bruk utelukkende av lenkeanalyse, og utelukkende bruk av likhetsanalyse, i tillegg til rangeringsresultater ved forskjellige kombinasjoner og analyser av disse. Resultatene av denne testen viser tydelig at analyse av likheter mellom dokumenter kan brukes til å oppnå god rangering om man ser på likheter mellom dokumenter innenfor et tema basert på søkeordene. Denne metoden fungerer godt både alene men også sammen med lenkeanalyse. Det har vist seg at rangering basert på likhetsanalyse kombinert med en svak lenkestruktur var den beste rangeringsmetoden av alle testet. Omtrent like god var rangeringsmetoden som baserte seg på likhetsanalyse kombinert med analyse av en rik lenkestruktur. Google, som var sammenligningskriteriet vårt, bruker en mye kraftigere lenkestruktur kombinert med avansert tekstrelevansanalyse. Å foreta rangering av treff uten noe som helst bruk av tekstrelevans, kan kanskje sammenlignes med å sette en maratonløper med to bein opp mot en med ett bein og krykker. Med dette i minnet vil vi påstå at resultatene oppnådd i dette prosjektet er imponerende og interessante å forske videre på, og prøve å kombinere med tekstrelevansanalyse.

[35] Metode 8

Målet med oppgaven var *å finne ut hvor godt likhetsberegninger mellom dokumenter kan fungere som en substitusjon for tradisjonelle lenker i domener uten egen lenkestruktur*. Basert på testene er hovedkonklusjonen at T-Rank analyse av resultater av likhetsanalyse mellom dokumenter innen et vist tema gir gode resultater selv uten bruk av tekstrelevansanalyse. Resultatene blir enda bedre om man kombinerer resultatene av likhetsanalysen med en svak lenkestruktur før man bruker T-Rank til å analysere resultatet. Resultatene fra enkelte av metodene[36] i testingen oppleves som overraskende bra i mange tilfeller, nesten like bra som det man kunne forvente av en god fullstendig søkemotor. Å kombinere disse, og andre metoder utarbeidet gjennom dette prosjektet, med avansert tekstrelevans virker derfor som en interessant og lovende oppgave å forske videre på.

[36] Metode 8 og 9.

10 Appendiks

10.1 B3000.glenn-erik.com

Resultatet av systemet laget i dette prosjektet kan det være mulig å teste på websiden http://B3000.glenn-erik.com
Her vises også evalueringsberegninger og informasjon om rangeringen sammenlignet med Googles rangering i sanntid. Denne websiden er testgrensesnittet vi har brukt for å teste de forskjellige metodene i oppgaven etter hvert som de er blitt implementert. Søkegrensesnittet hjelper å visualisere resultatene og teste metodene i praksis. Dette var et av kravene i oppgaven.

Websiden lar brukeren søke på opptil fem ord og velge blant 13 forskjellige rangeringsmetoder. Alle disse tar i betraktning enten lenker eller likhetsberegninger, eller lenker og likhetsberegninger sammen på forskjellige måter og kombinerer resultatene.

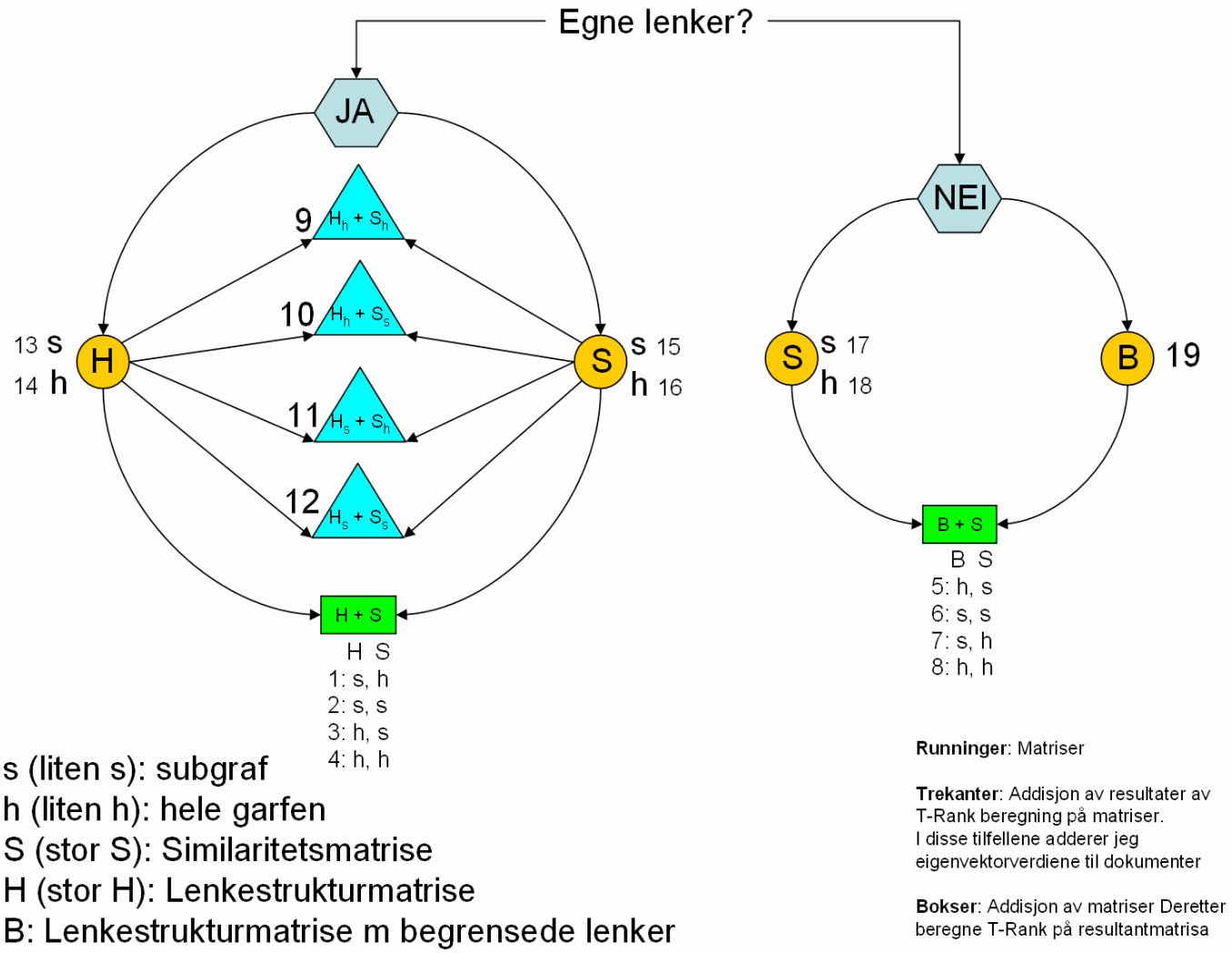
På hovedsiden gjenspeiler tallet skrevet ved hvert av de mulige rangeringsvalgene de sammen tallene som i Figur 13. Mye informasjon vises som resultat av søk, kapittel 8.3.1forklarer hvordan denne informasjonen skal tolkes.

Similaritetsberegningene mellom dokumentpar kan testes på websiden ved å trykke på ”like” etter å ha foretatt et vanlig søk. Den rå teksten som B3000 har brukt til å basere rangeringen på kan ses ved å trykke på ”txt” ved siden av hvert treff etter et søk. Ellers vises all informasjon gitt i Figur 24 i kapittel 8.3.1.

I tillegg til valg av rangeringsmulighet har brukeren mulighet til å angi to verdier, ”TR tweak” og ”SIM tweak”. Disse verdiene påvirker hhv. analysen av lenkestrukturdatabasen og analysen av likhetsberegningene mellom dokumentpar i forhold til hverandre. Det er viktig å være klar over hvordan disse verdiene blir brukt av systemet. Mer informasjon om hvordan disse verdiene påvirker resultatet finnes i slutten av kapittel 7.1.7. Siden ”SIM tweak” og ”TR tweak” påvirker resultatet i forhold til hverandre er det kun enkelte av de rangeringsmetodene som tar i betraktning både likheter mellom dokumenter *og* lenkestruktur som påvirkes av disse variablene. Disse er markert på websiden med understreking.

Grunnet oppgavens konfidensielle fremleggelse er websiden passordbeskyttet. Dette passordet kan det være mulig å få ved å henvende seg til forfatteren av oppgaven eller en av veilederne.

Figuren under er en modifisert utgave av Figur 13 som gir en oversikt over rangeringsmuligheter. Se kapittel 7.1.10 for hvilke av disse metodene som er implementert i dette prosjektet og finnes på websiden.



Figur 27 - Oversikt over muligheter for analyse av dokumentene i Wikipedia og lenker funnet mellom dem og likheter beregnet mellom dokumentpar (modifisert versjon av Figur 13).

10.2 Gruppering av søkefrasene brukt i testingen

Tabellen under viser en liste over søkefraser som gjør det best i forhold til forskjellige metoder. Metodene er gruppert i forhold til hvilke som inneholder ren lenkeanalyse, og hvilke som innbærer likhetsanalyse og evt. lenkeanalyse kombinert. "Enkel relevans" er også tatt med som en egen gruppe. *H* betyr hyperlenkeanalyse og inkluderer metodene T-Rank og PageRank. *S* betyr similaritetsanalyse og inkluderer metodene 8, 9 og 4 / 14. *V* er addisjon av resultatet av T-Rank analyse på en matrise over likheter addert med T-Rank analyse av en matrise over lenker, og står derfor som en egen gruppe. *R* står for relevans. *Uavgjort* viser de tilfeller hvor det er uavgjort mellom *S* og *H*, ikke *R* tatt i betraktning (så sant ikke *R* gjorde det bedre enn *S* eller *H*).

H	*S*	*V*	*R*	*Uavgjort*
Astronomi	Oslo		Windows	Norge Oslo
Jakt	Matematikk		George Bush	Fiske
Mat	Krig		Hus	Rike
Penger	Trondheim		Metode	
Vann	Internett			
Bok	Økonomi			
Militær	Natur			
Forsvar	Dyr			
Forsvaret				

Table 8 - List over søkefraser som gjøre det best i forhold til de forskjellige søkemetoder brukt.

Det må påpekes at *R* gjør det *betraktelig* mye dårligere enn de andre metodene i de tilfeller hvor den ikke gjør det bedre. *V* ligger sånn ca. midt mellom i de fleste tilfeller. Det virker som om gruppen *S* består av søkefraser som handler om ett litt mer spesifikt tema enn søkefrasene som rangerer best i *H*. Søkefrasene med tvetydig betydning, *dyr, fiske og rike*, virker (noe uventet) ikke negativt på similaritetsanalysene. Det ble enten uavgjort, eller *S* vant.

10.3 Formel for å måle B3000 sine rangeringsresultater

Under er en utredning av formelen brukt for å beregne kvaliteten til de topp 10 treffene for en rangeringsmetode i B3000 sammenlignet med forventet verdi ved tilfeldig rangering. Utredningen baserer seg på informasjon allerede beskrevet i kapittel 0

$$f := \frac{1}{N}\sum_{i=1}^{N}|c-i| = \frac{1}{N}\left[\sum_{i=1}^{c}(c-i)+\sum_{i=c+1}^{N}(i-c)\right] = \frac{1}{N}\left[\sum_{a} + \sum_{b}\right]$$

Sum(a):

$$\sum_{a} = c(c) - \sum_{i=1}^{c} i = c(c) - \frac{c(c+1)}{2} = \frac{2c^2 - c^2 - c}{2} = \frac{c^2 - c}{2} = \frac{c(c-1)}{2}$$

For Sum(b), la j=i-c

$$\sum_{i=c+1}^{N}(i-c) = \sum_{j=1}^{N-c} j = \frac{(N-c)(N-c+1)}{2} = \frac{N^2 - (2c-1)N + c^2 - c}{2}$$

1/N*(Sum(a) + Sum(b)) blir så:

$$\frac{1}{N}\left[\frac{c(c-1)}{2} + \frac{N^2 - (2c-1)N + c^2 - c}{2}\right] = \frac{1}{2N}\left[N^2 - (2c-1)N + 2(c^2 - c)\right] = \frac{1}{2}\left[N - (2c-1) + \frac{2}{N}(c^2 - c)\right]$$

La $1 \le c \le 10$,

$$\frac{1}{10}\sum_{c=1}^{10}\left[\frac{N}{2} - \frac{(2c-1)}{2} + \frac{c^2 - c}{N}\right] = \frac{1}{10}\left[\sum_{c=1}^{10}\frac{N}{2} - \frac{1}{2}\sum_{c=1}^{10}(2c-1) + \frac{1}{N}\sum_{c=1}^{10}c^2 - \frac{1}{N}\sum_{c=1}^{10}c\right] =$$

$$\frac{1}{10}\left[5N - \sum_{1} + \sum_{2} - \sum_{3}\right]$$

For Sum 1: $\frac{1}{2}\sum_{c=1}^{10}(2c) - \frac{1}{2}\sum_{c=1}^{10}1 = \frac{10(10+1)}{2} - 5 = 55 - 5 = 50$

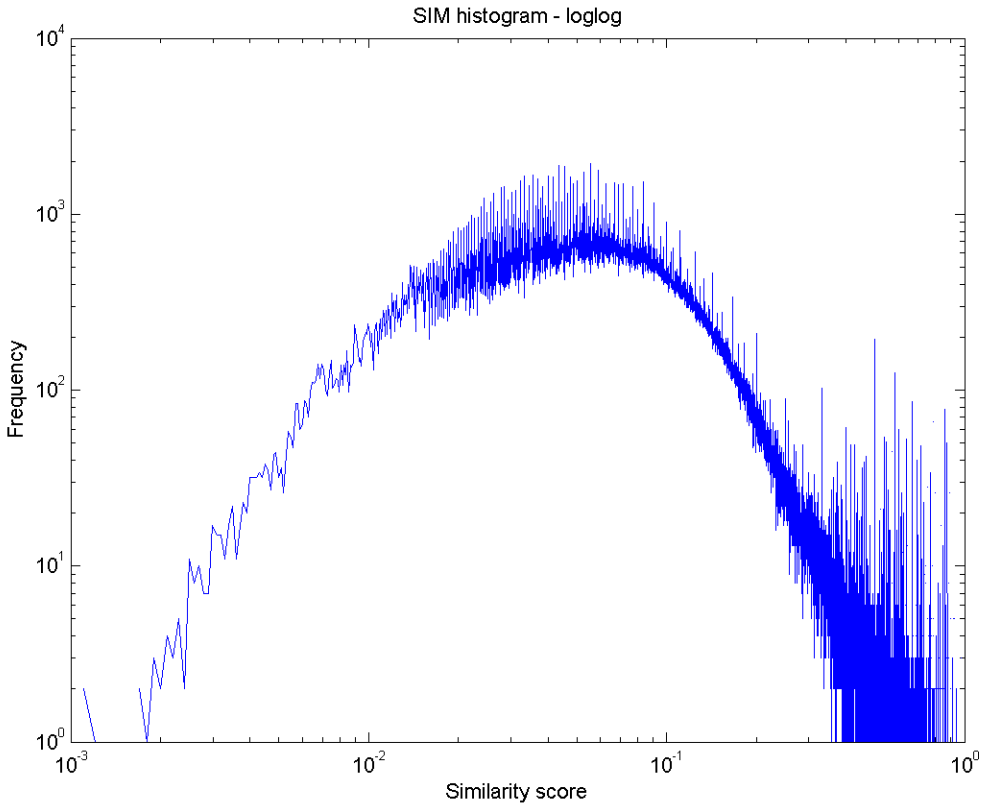
For Sum 2: $\frac{1}{N}\frac{1}{6}(2000 + 300 + 10) = \frac{1}{N}\frac{2310}{6} = \frac{385}{N}$

For Sum 3: $\frac{1}{N}(55)$

Putter inn Summene 1, 2 og 3 i formelen på linja over Sum 1:

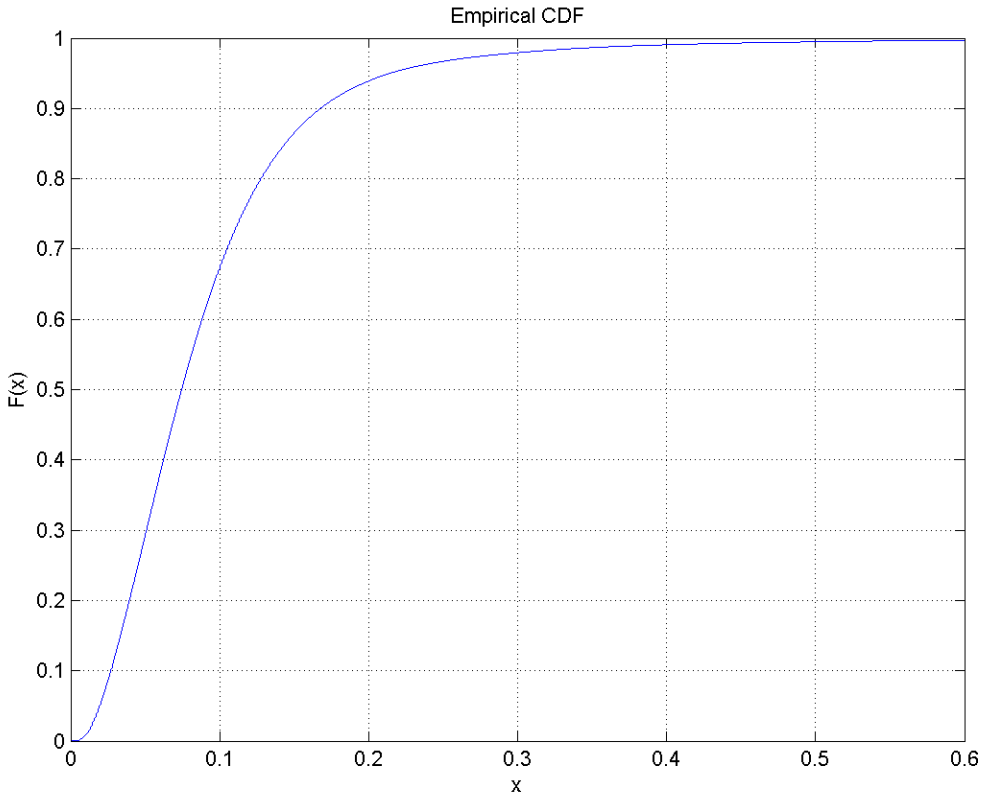
$$\frac{1}{10}\left[5N - 50 + \frac{385}{N} - \frac{55}{N}\right] = \underline{\underline{\frac{N}{2} - 5 + \frac{33}{N}}}$$

10.4 Resultater av beregninger på matrisa over dokumentlikheter



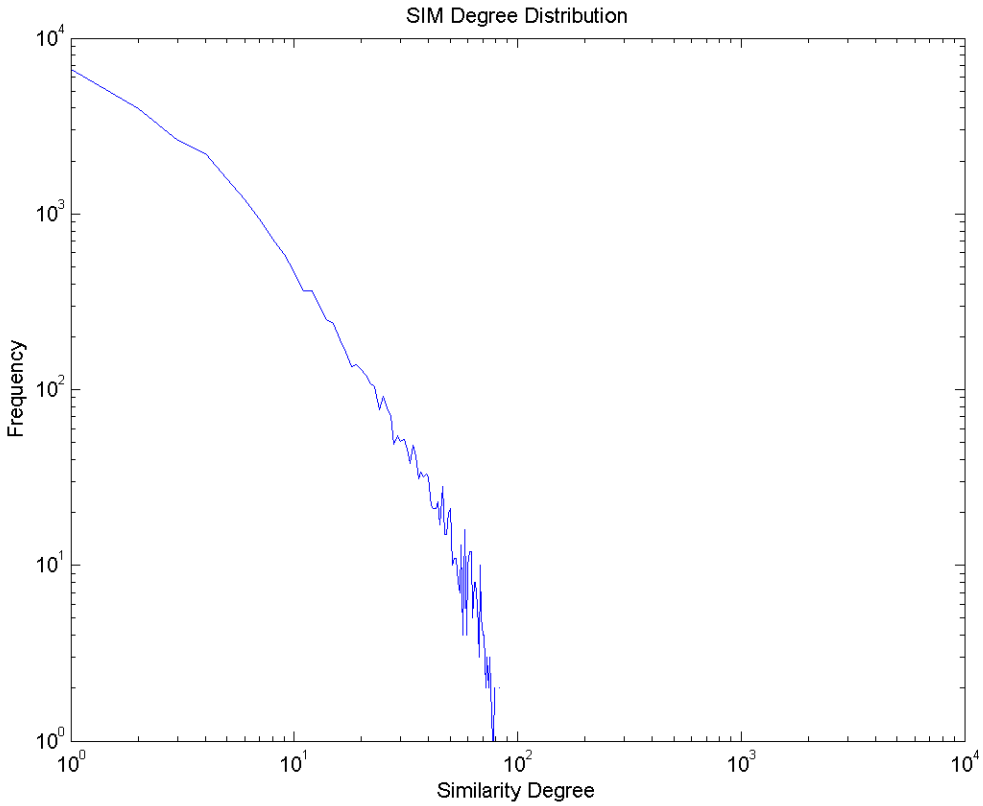
Figur 28 - Oversikt over hvor mange dokumentpar som har en viss likhet plottet logaritmisk

Man kan se at det er flest par (med likhet > 0) som har likheten 0.03. Denne grafen er et loglog plott av ”*Figur 23 - Distribusjon av antallet dokumentpar i samlingen og deres likhet.*”



Figur 29 - Dette viser normalisert (sannsynlighet) integralet av figur 16.

Sannsynligheten for at likheten mellom to dokumenter (med likhet > 0) er mindre enn eller lik x.



Figur 30 – Plott av Similarity degree.

For eksempel kan vi lese ut av grafen at ca. 6500 dokumenter har en likhet med *ett* annet dokument, mens ca 400 dokumenter har en likhet med 10 andre.

10.5 Kildekoder og annet materiale vedlag på CD

På den vedlagte CD'en finnes kildekoder for de forskjellige komponenter implementert samt annen nyttig informasjon. Vi forklarer her hvilke filer vedlagt på CD'en som tilsvarer hvilke komponenter. Under lister vi opp filnavn og beskriver koden som befinner seg i disse, og hvilke komponenter i B3000 systemet de utgjør. Koden i de fleste komponentene beskrevet forutsetter at Wikipediadatabasen er tilgjengelig.

10.5.1 Kode for komponenter som brukes offline

- **calculate_similarity_for_all_documents_EASY.php**
  Dette er komponenten som beregner likheter mellom alle Wikipediadokumentene i henhold til prosedyren beskrevet i kapittel 4.6 – "

Likhet mellom alle dokumentpar i en samling - Metode 3". Dette er den raskeste av alle metodene foreslått for å beregne likhet. Denne komponenten brukes offline og lager selve "*simdb*" matrisa, som er en tabell med oversikt over alle dokumenter og deres likheter. Denne fila er komponenten "*Similarity_Computer*" i Figur 17 på side 71 og den forutsetter at koden i fila "*find_words.php*" er kjørt.

- **calculate_similarity_between_two_cur_ids.php**
  Denne funksjonen brukes av "calculate_similarity_for_all_documents_EASY.php" og er funksjonen "*Calculate_Similarity_Between_two_DocIDs()*" i Figur 17 side 71. Siden denne funksjonen brukes av komponenten beskrevet over krever den at koden i fila "*find_words.php*" er kjørt.
- **calculate_similarity_for_all_documents_HARDWAY.php**
  Dette er komponenten beskrevet i kapittel 4.4 - "*Likhet mellom alle dokumentpar i en samling - Metode 1*". Denne bruker funksjonen "*calculate_similarity_between_two_cur_ids.php*" og forutsetter derfor at koden i fila "*find_words.php*" er kjørt. Koden i denne fila brukes ikke i praksis i det ferdige systemet B3000.
- **find_words.php**
  Dette er komponenten "*Word_Normalizer*" i Figur 17 på side 71. Dette bør være den første komponenten som kjøres blant alle offlineprosedyrene. Koden bruker stoppord funnet i fila "*norske-stoppord.txt*".
- **create_II_sorted_by_p_values.php**
  Dette er komponenten "*Build_Inverted_Index*" i Figur 17 på side 71. Denne bygger den inverterte indeksen i henhold til kravene beskrevet i kapittel 4.6 - "

Likhet mellom alle dokumentpar i en samling - Metode 3". Forutsetningene forklart i dette kapittelet må gjelder dersom komponenten "*Similarity_Computer*" skal fungere som planlagt. Denne komponenten forutsetter at koden i fila "*find_words.php*" er kjørt.

- **create_II.php**
  Denne komponenten bygger in invertert indeks uten hensyn til kravene til "*calculate_similarity_for_all_documents_EASY.php*". Denne komponenten brukes derfor ikke i praksis i B3000.
- **find_links.php**
  Koden i denne fila er komponenten "*Build_Link_Structure_Database*" i Figur 17 på side 71. Komponenten lager matrisa "*lsdb*" og bør være en av de første offlinekomponentene som kjører.
- **analyse_simdb_pluss_lsdb_(matrix-addition)_on_whole_graph.php**
  Dette er koden som bygger en matrise over likheter mellom alle dokumenter, deretter adderes all informasjon i lsdb. Resultatet er en matrise klar til å analyseres av "*linkanalyser.cpp*". Denne komponenten iverksettes offline, og bruker opptil en time på sine beregniner. Resultater er en viktighetsscore for hvert dokument basert på likhetene mellom alle dokumenter og lenkene mellom alle dokumenter tatt i betraktning (analysert med T-Rank). Resultatet av denne fila tilsvarer tilfelle **7** i Figur 13 og Figur 14 og er metoden
  T-Rank(Madd(simdb + lsdb))
  i Liste 2. Koden i denne fila inngår som en subfunksjon til komponenten "*Link_Analyser*" i Figur 17. Koden i denne fila er mer oversiktlig illustrert som en "Action" som startes av "*Calculate Ranking Data*" i Figur 15.
- **analyse_simdb_pluss_lsdbSPARSE_(matrix-addition)_on_whole_graph.php.**
  Denne koden er veldig lik den beskrevet over. Den eneste forskjellen er at den bruker en lsdb hvor 70% av de originale lenkene er tilfeldig slettet. Koden i denne fila tilsvarer tilfelle **10** i Figur 13 og Figur 14 og er metoden
  T-Rank(Madd(simdb + lsdb_sparse))
  i Liste 2. Koden i denne fila inngår som en subfunksjon til komponenten "*Link_Analyser*" i Figur 17. Koden i denne fila er mer oversiktlig illustrert som en "Action" som startes av "*Calculate Ranking Data*" i Figur 15.
- **analyse_simdb_on_whole_graph.php**
  Koden i denne fila tilsvarer tilfelle **3** og **13** i Figur 13 og Figur 14 og er metoden
  T-Rank(simdb)
  i Liste 2. Den klargjør en matrise for analyse av "*simdb*" med *"linkanalyser.cpp*". Koden i denne fila inngår som en subfunksjon til komponenten "*Link_Analyser*" i Figur 17. Koden i denne fila er mer oversiktlig illustrert som en "Action" som startes av "*Calculate Ranking Data*" i Figur 15.

10.5.2 Kode for komponenter som brukes i sanntid

Koden i alle filene under iverksettes av komponenten ”*Search_Interface*” i Figur 21. Figur 20 viser hvordan alle komponentene (som filene under tilsvarer) iverksettes som en ”Action” av brukeren som søker gjennom Use Caset ”*Search*”.

- **SIMILARITY-and-T-Rank-wholegraph-MATRIXADDITION.php**
  Koden i denne fila tilsvarer tilfelle **5** i Figur 13 og Figur 14. Dette er metoden
  T-Rank(Madd(simdb_sub + lsdb))
  i Liste 2 og den bruker veldig lang tid. Hvor lenge er avhengig av hvor mange iterasjoner ”*linkanalyser.cpp*” bruker før den konvergerer. Koden i denne fila er ikke tatt med i testingen av systemet.
- **SIMILARITY-subgraf.php**
  Koden i denne fila tilsvarer tilfelle **4** og **14** i Figur 13 og Figur 14. Dette er metoden
  T-Rank(simdb_sub)
  i Liste 2. Koden i denne fila er tatt med i testingen av systemet.
- **SIMILARITY-sub-and-LSDB-whole-RESULTADDITION.php**
  Koden i denne fila tilsvarer tilfelle **17** i Figur 13 og Figur 14. Dette er metoden
  Vadd(T-Rank(lsdb) + T-Rank(simdb_sub))
  i Liste 2. Koden i dette fila er tatt med i testingen av systemet.
- **SIMILARITY-and-T-Rank-subgraf-MATRIXADDITION.php**
  Koden i denne fila tilsvarer tilfelle **8** i Figur 13 og Figur 14. Dette er metoden
  T-Rank(Madd(simdb_sub + lsdb_sub))
  i Liste 2. Koden i dette fila er tatt med i testingen av systemet.
- **SIMILARITY-and-T-Rank-subgraf-RESULTADDITION.php**
  Koden i denne fila tilsvarer tilfelle **19** i Figur 13 og Figur 14. Dette er metoden
  Vadd(T-Rank(lsdb_sub) + T-Rank(simdb_sub))
  i Liste 2. Koden i dette fila er ikke tatt med i testingen av systemet.
- **B_pluss_H_paa_SUBGRAF_matriseaddisjon.php**
  Koden i denne fila tilsvarer tilfelle **9** i Figur 13 og Figur 14. Dette er metoden
  T-Rank(Madd(simdb_sub + lsdb_sparse_sub))
  i Liste 2. Denne funksjonen er tatt med i testingen av systemet.
- **check_google_rank.php**
  Dette er koden som brukes av ”*Search_Interface”* i Figur 17 til å foreta et tilsvarende søk i Google og forberede resultatene derifra til å lage statistikk for sammenligning med og evaluering av B3000 sine resultater. Denne koden er ikke

tatt med i designet av systemet noe sted siden den egentlig ikke er en del av B3000, kun en nødvendig komponent for å gjennomføre testingen.

- **search_interface.php**
  Dette er koden som tilsvarer ”Action” ”*Search”* i Figur 20 og komponenten ”*Search_Interface*” i Figur 21. Hovedoppgaven til denne komponenten er å kommunisere med brukeren og velge korrekt kommunikasjon mellom nødvendige komponenter i systemet avhengig av den søkemetoden som velges av brukeren.
- **list.php**
  Denne fila inneholder kode som beregner all statistikk nødvendig for å sammenligne rangeringen av dokumentene mellom B3000 og Google. Dette gjøres i sanntid og resultatet kan testes ut på B3000.glenn-erik.com. I tillegg er det koden i denne komponenten som lister resultatet av den faktiske rangeringen av dokumentene utført av B3000. Disse listes opp i en tabell sammen med en tabell over Googles rangering. Koden i denne fila iverksettes og brukes av koden i fila ”*search_interface.php*”.

10.5.3 Kode som brukes både offline og i sanntid

- **linkanalysis.cpp**
  Dette er koden til komponenten ”Link_Analyser” i Figur 17 på side 71. Den er implementert i C++ og brukes online av:
  SIMILARITY-and-T-Rank-wholegraph-MATRIXADDITION.php
  SIMILARITY-subgraf.php
  SIMILARITY-sub-and-LSDB-whole-RESULTADDITION.php
  SIMILARITY-and-T-Rank-subgraf-MATRIXADDITION.php
  SIMILARITY-and-T-Rank-subgraf-RESULTADDITION.php
  B_pluss_H_paa_SUBGRAF_matriseaddisjon.php

  og den brukes offline av:
  analyse_simdb_pluss_lsdb_(matrix-addition)_on_whole_graph.
  analyse_simdb_pluss_lsdbSPARSE_(matrix-addition)_on_whole_graph.php
  analyse_simdb_on_whole_graph.php

10.5.4 Annet

- **20050309_cur_table.sql**
  Dette er den rå database dumpen av Wikipedia samlingen slik den var da den ble lastet ned 9. Mars. Det er denne dokumentsamlingen som brukes gjennom hele prosjektet. Fila inneholder en god del mer informasjon relatert til hvert dokument

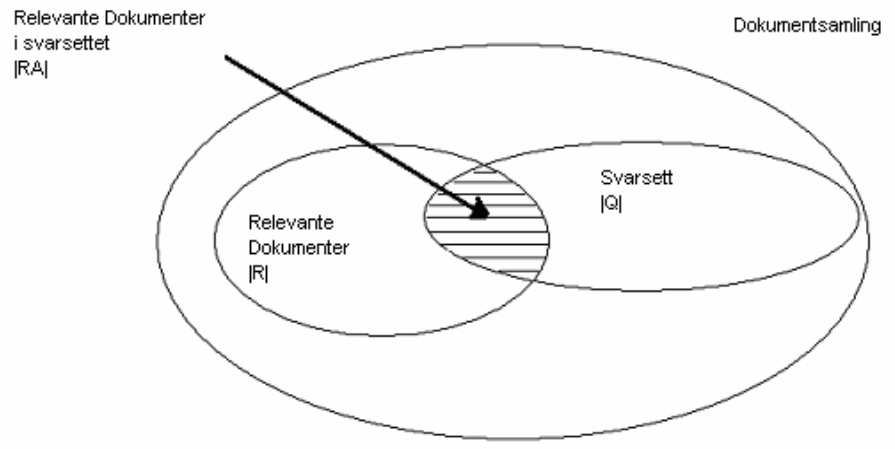
enn den som brukes av B3000. Fila inneholder også en del dokumenter som ikke er artikler. Kun de dokumenter som har kolonne ”cur_namespace” satt til 0 er artikler som brukes av B3000. Bokstavene ”cur” i filnavnet betyr at det er den mest oppdaterte versjonen som brukes akkurat nå. Wikipedia tilbyr også nedlasting av databasen deres inkludert alle versjoner av den. En slik fil heter ”20050309-old_table.sql”. Begge typen filer kan finnes på http://download.wikimedia.org. Versjoner på forskjellige språk finnes tilgjengelig, det er den norske versjonen som er brukt i dette prosjektet.

- **read_arrays_to_memory_from_file.php**
  Dette er kode vi har brukt for å teste kjøretiden på å laste normaliserte ordfrekvenslister over alle dokumentene i Wikipedia opp i minnet. Resultatet av denne koden har vært brukt som hjelp til å beregne forskjellige kjøretidsanalyser i løpet av prosjektet. Spesielt brukt har denne koden vært for å komme frem til resultater i kapittel 4 - ”*Beregning av likhet mellom dokumentpar*”.
- **remove_redirects_from_cur_ids_folder.php**
  Koden i denne fila har blitt brukt til å synkronisere resultatet fra ”*Word_Normalizer*” med dokumentene i Wikipediadatabasen. Dette har blitt gjort ved å sjekke at de normaliserte ordrekvenslistene for hvert dokument ikke inneholder ordfrekvensen til et dokument i Wikipedia som egentlig er en ”REDIRECT” til et annet dokument.

10.6 En vanlig måte å foreta evalueringer av søketester

"Precision & Recall" er en veldig vanlig måte å foreta evaluering av oppslag i dokumenter på (Baeza-Yates and Ribeiro-Neto 1999). Vi har derfor valgt å beskrive kort hvordan denne fungerer i dette appendiksavsnittet, selv om vi ikke bruker denne metoden som mål på våre testresultater.

Precision (presisjon) er en måte å måle hvor mange relevante dokumenter i svarsettet som er relevante. Recall (tilbaketrekk) er en måte å måle hvor mange relevante dokumenter i fra samlingen som har blitt funnet. Det er viktig å score høyest mulig på begge disse for å ha et bra produkt. Det er ikke nødvendigvis sånn at produktet er bra selv om det har en veldig god presisjon. Presisjonen alene er nesten unyttig om metoden scorer dårlig på tilbaketrekk.For å forklare denne evalueringsmetoden bedre vil vi bruke et eksempel. I dette eksempelet ser vi for oss at vi har en forespørsel (et søk) ***I***. Et fasitsett ***R*** med relevante dokumenter. **|*R*|** er antallet dokumenter i fasiten. Videre vil vi kalle svaret vi får ved søk for ***A***. **|*A*|** er da antallet dokumenter i dette svaret. Antallet dokumenter i svaret som er relevant blir så krysningen (intersection) av settene ***R*** og ***A***, som vi kaller **|RA|**.



Figur 31 – Illustrasjon av presisjon og tilbaketrekk for resultatet av et søk.

Presisjon blir så brøkdelen av de returnerte dokumentene (***A***) som er relevante:

$$\text{Presisjon} = \frac{|RA|}{|Q|}$$

Dette gir et mål på hvor god metoden er til å filtrere bort uønskede eller urelevante dokumenter. Tilbaketrekk, eller det mer kjente engelske utrykket "recall", blir så brøkdelen av de relevante dokumentene som blir returnert ved et søk:

$$\text{Tilbaketrekk} = \frac{|RA|}{|R|}$$

Dette måler hvor god metoden er til å finne de ønskede (mest relevante) dokumentene.

10.7 Uttrykksdefinisjoner

Under er en lister over ord og uttrykk vi bruker i rapporten. Lista er sortert alfabetisk.

- **Ankertekst**
  Ankertekst betyr teksten som beskriver hva en lenke peker på. For eksempel er ankerteksten ”NTNU” i følgende HTML-pekerkode:
  NTNU
- **Array**
  Et array er en måte å ta var på data i minnet i en liste. Det er en liste med entries. For eksempel kan et array med to entries være:

  Array[1]=”Person”
  Array[2]=”Bil”

  På denne måten kan et array i minnet representere en vektor. Arrays i minnet kan også være mer kompliserte. Et typisk eksempel på dette er et array hvor entriene også er arrays, noe som resulterer i en to-dimensjonal array. Arrays hjelper oss på denne måten å holde orden på dataen i minnet ved å for eksempel kategorisere den. Vi kan utvide eksempelet over til:

  Array[1][1]=”Glenn-Erik”
  Arra[1][2]=”Aristoteles”
  Array[2][1]=”Ford Mustang”
  Array[2][2]=”Lada”

  På denne måten kan et array i minnet representere en matrise. I dette eksempelet vet vi at data som befinner vi i Array[1][x] er personer, og data i array[2][x] er biler. Slik kan Arrays hjelpe oss å holde orden på dataen vi skal behandle i minnet.
- **Autoritetsscore**
  En poengverdi som tilegnes noder i en graf basert på gjentatt bruk av FB operatoren. Verdien til hver node vil konvergere og stabilisere seg til en verdi. En Autoritetsscore er et mål på hvor mange pekere det er mot en node, og tatt i betraktning hvor viktige nodene som peker er ansett, eller hvor gode Hubber nodene som peker er. I oppgaven skiller vi mellom ”Autoritet” og ”autoritet”. Den første bruker vi når vi direkte relaterer til Kleinbergs definisjon og arbeide om autoriteter (Kleinberg oppfant autoritet og hub). Den andre bruker vi når vi

generelt omtaler fenomenet autoritet. For eksempel finner også T-Rank autoritets- og hub-verdier, men ikke på samme måte som Kleinbergs HITS metode.

- **BF**
  En operator (metode) som brukes flere ganger tilbake og frem (mot og med pekeres retning) over noder i en graf, og resulterer i en Hubscore for hver node.
- **Bra lenkestruktur (dense link structure)**
  Mange lenker i et domene, og lenkene er godt spredt blant dokumentene i samlingen. Mao. er "bra lenkestruktur" til god hjelp når vi skal analysere viktigheten blant dokumentene i domenet.
- **Case-insensitive**
  Dette betyr at man ikke tar hensyn til om det er store eller små bokstaver man jobber med. Store og små bokstaver tolkes som de samme tegnene.
- **Case-sensitive**
  Det motsatte av "Case-insensitive". Man behandler store og små bokstaver som forskjellige tegn.
- **Corpus**
- Definisjon av *alle* ord i dokumentsamlingen vi har brukt, og hvor mange ganger ordene forekommer. Corpus er viktig for å kunne foreta beregninger om hvor lang tid forskjellige operasjoner på tekstsamlingen vil ta. For eksempel er kjøretiden til å plukke ut alle ord i alle dokumenter i samlingen, og telle hvor mange ganger de forekommer, avhengig av størrelsen til corpus.
- **Crawle**
  Laste ned dokumenter fra en ekstern server, og følge header informasjon og lenker funnet.
- **Dårlig lenkestruktur (sparse link structure)**
  (dårlig lenkestruktur = "glissen" lenkestruktur). Dette er tilfellet hvor man har få lenker blant dokumentene i en samling, eller lenker som er dårlig spredt mellom dokumentene. Med få lenker mener vi typisk:
  $$Sparse < 0.005 * \frac{N(N-1)}{2}$$
  Hvor N er antall dokumenter i domenet. Mao. er "dårlig lenkestruktur" til liten hjelp når vi skal analysere viktigheten blant dokumentene i domenet.
- **EVC**
  Egenvektor Centrality. Dette er svarene man oppnår ved å kjøre for eksempel T-Rank på en symmetrisk graf. Da får man til svar nodenes eigenvektor centrality. Verdiene er gitt som prinsipal egenvektor til en symmetrisk grafmatrise Dette er i motsetning til å kjøre T-Rank på en rettet graf hvor man kan få T-Rank forward eller T-Rank backward verdier for nodene.

- **Entries**
  Med ”entrie” mener vi tilfeller hvor man har en verdi å putte inn i et array i minnet eller tabell i en database. Et array uten ”entries” er et tomt array. En tabell uten ”entries” er en tom tabell.
- **FB**
  En operator (metode) som brukes flere ganger fram og tilbake (med og mot pekeres retning) over noder i en graf, og resulterer i en Autoritetsscore for hver node.
- **Fullkommen graf**
  En samling noder hvor relasjonene mellom nodene er rettede piler, og det er mulig å nå en hver annen node i grafen ved å følge pilenes retning.
- **Graf**
  Med graf mener vi en oversikt over alle dokumenter og alle relasjonene mellom dokumentene i et domene. Dokumentene i en graf er noder, og relasjonene mellom dem kan være rettede lenker eller symmetriske likhetsberegninger.
- **HTML-Tags**
  Med ”HTML-Tags” mener vi all informasjon i et dokument som står mellom tegnene: ”<” og ”>” (uten gåseøynene). Å fjerne HTML-Tags betyr å fjerne denne dataen fra dokumentet, inkludert mindre og større tegnene < og >.
- **Hubscore**
  En poengverdi som tilegnes noder i en graf basert på gjentatt bruk av BF operatoren. Verdien til hver node vil konvergere og stabilisere seg til en verdi. En Hubscore er et mål på hvor god en node er til å formidle andre noder som er ansett som gode Autoriteter. I oppgaven skiller vi mellom ”Hub” og ”hub”. Den første bruker vi når vi direkte relaterer til Kleinbergs definisjon og arbeide om huber (Kleinberg oppfant autoritet og hub). Den andre bruker vi når vi generelt omtaler fenomenet hub. For eksempel finner også T-Rank hub- og autoritets-verdier, men ikke på samme måte som Kleinbergs HITS metode.
- **IR**
  Information Retrieval. Utrykk for måter å finne / søke / ordne informasjon på.
- **Lenke popularitet**
  Telle antallet lenker som peker mot et dokument i en dokumentsamling.
- **Lenkeanalyse**
  Basert på analyser av lenker funnet i domenet, angis en verdi til hvert dokument som mål på dokumentets viktighet. Det finnes forskjellige metoder for å gjøre dette. Dette kan ofte gjøres offline, uavhengig av brukeren som søker. I de tilfeller dette gjøres i sanntid er det som regel fordi man ser på en del av den total lenkestrukturen i domenet.

- **Likhet**
  Med ”likhet” mener vi en verdi mellom 0 og 1 som beskriver likheten mellom to dokumenter. 0 beskriver helt ulik, og 1 beskriver helt lik, sett av metoden brukt.
- **Matlab**
  Et program optimalisert for å foreta mattematiske beregninger.
- **Metode**
  En spesiell måte å gjøre noe systematisk i forskjellige steg. Et sett med regler og hvordan man skal bruke dem for å oppnå logiske resultater.
- **Mutually defining**
  Hvis du spør meg: ”Hvordan kan vi fortest mulig bli rik?”. Og jeg svarer: ”Da blir du nødt til å gå til Roger Midtstraum, han er god på sånt”. Så går du til Roger og spør han om det samme og han svarer: ”Det lureste du kan gjøre er å gå til Glenn-Erik og spørre han, han har peiling på det”. Da er Glenn-Erik og Roger ”mutually defining”.
- **Node**
  Et dokument eller side i en graf. I wikipedia omtaler vi ofte dokumenter som ”artikler”.
- **Node degree**
  Dette er tilfellet hvor man kun teller antall lenker pekende mot et dokument og poengverdien til dette dokumentet er et resultat utelukkende av antallet slike lenker. Hvis PageRank brukes på en symmetrisk graf vil ”node degree” være resultatet.
- **”Off line”**
  Handlinger som skjer uavhengig av en bruker av systemet. Dette er typisk handlinger som er iverksatt manuelt av utviklerne av systemet for å klargjøre eller vedlikeholde systemet.
- **”Online”**
  Handlinger som iverksettes i det en bruker foretar et søk i B3000.
- **Operator**
  Et symbol som representerer en funksjon fra funksjoner til funksjoner (konstant som binder fri variabel).
- **Padde**
  Med ”padde” mener vi å foreta en operasjon på en tekststreng som sørger for at tekststrengen får en viss størrelse eller lengde.
- **Poeng**
  Med poeng menes en tallverdi som gis til en internettside, eller dokument i et domene basert på lenkeanalyse.

- **Produksjonskode (production code)**
  Produksjonskode er nøye forseggjort, testet, tweaket og debugget kode som skal være optimalisert til å fungere raskt, feilfritt og effektivt. I et ferdig produkt er det viktig å ha produksjonskode. I et forskningsprosjekt, som dette, er ikke produksjonskode av høy prioritet. Vi kan ta oss store friheter i kodingen i denne oppgaven da det er å se på, og studere resultatene som er av interesse, og ikke kjøretiden og feilhåndteringen. Fordelen ved å ikke måtte skrive produksjonskode er at vi får fokusert mer på å teste teoriene, og ikke bruker så mye tid på feilhåndtering og optimalisering.
- **Rettet graf**
  Er det samme som en usymmetrisk graf.
- **Sanntid**
  Handlinger som iverksettes i det en bruker foretar et søk i B3000.
- **Side**
  Med side mener vi som regel en unik internettside (URL).
- **Similaritet**
  Det samme som likhet.
- **Similaritetsmatrise**
  En matrise over alle dokumentpar i en samling og likheten mellom hvert dokumentpar.
- **Spam**
  Med spam mener vi dokumenter som ikke tilføyer datasamlingen vår noe mer verdi enn den har fra før. Vi har ikke spam i datasettet vi bruker, men det er et økende problem på nettet, og en del av grunnen til hvorfor lenkeanalyse har blitt så populært. Spam oppstår ofte fordi enkelte mennesker ønsker å skaffe folks oppmerksomhet.
- **Struct**
  En måte å lagre data på harddiske og behandle data i minnet som ikke er en vanlig datatype som for eksempel int, double, char, etc. En strukt kan for eksempel bestå av både en int og en double og et array med chars.
- **Symmetrisk**
  Med ordet symmetrisk mener vi en graf hvor relasjonene mellom nodene går begge veier mellom to noder. Med andre ord, en node kan ikke ha en verdi pekende mot en annen node, uten at den andre noden også peker tilbake med samme verdi på den aktuelle noden. Symmetrisk er det samme om urettet. En similaritetsmatrise over likheten mellom dokumentpar i en graf er symmetrisk.
- **Symmetrisk likhetsgraf**
  Dette er en oversikt over alle dokumentene i et domene, og likhetsverdien mellom

par av dokumenter. Likheten mellom to dokumenter er like stor i begge retninger, og dette gjør grafen symmetrisk. I praksis består denne grafen kun av de dokumentpar som har en viss likhet mellom hverandre, og ikke de som er ulike, kun teoretisk inneholder grafen alle mulige dokumentpar. Dette gjør grafen enklere å jobbe med og behandle i minnet. Man kan lett finne ut om et dokumentpar ikke har noen likhet ved å sjekke om paret eksistere i grafen eller ikke.

- **Tekstrelevansanalyse**
  Å analysere hvert relevante dokument i domenet i et forsøk på å finne en verdi som angir dokumentets relevans til søkeordene en bruker supplere søkemotoren med. Dette gjøres i sanntid ved et søk.
- **Treff**
  Med treff menes et enkelt resultat når man foretar et søk. Ved et søk er det ofte mange treff.
- **Urettet graf**
  Er det samme som en symmetrisk graf.
- **Usymmetrisk**
  Med usymmetrisk mener vi en graf hvor relasjonene mellom nodene kan gå en vei. Med andre ord kan en node peke til en annen uten at den påpekte noden behøver å peke tilbake. Usymmetrisk er det samme som rettet. Typisk er en usymmetrisk graf resultater etter å analysere lenkene i et domene.
- **Vekt**
  Med vekt menes det samme som poeng for noder, eller dokumenter, i en graf relativt til andre noder i grafen. En node i en graf kan på forskjellige måter tilegnes vekt (viktighets poeng, eller et mål på viktighet).

10.8 Referanseliste

Allan, J., A. Leuski, et al. (2000). Evaluating combinations of ranked lists and visualizations of inter-document similarity. Center for Intelligent Information Retrieval, Amherst, MA, University of Massachusetts.

Baeza-Yates, R. and B. Ribeiro-Neto (1999). Modern Information Retrieval. New York, Addison-Wesley
ACM Press.

Brin, S. and L. Page (1998). The Anatomy of a Large-Scale Hypertextual Web Search Engine. Computer Science Department. Stanford, CA, Stanford University**:** 20.

Canright, G. and K. Engø-Monsen (2005). Backward and forward non-normalized link weight analysis method, system, and computer program product. US Patent & Trademark Office. US.

Chen, C. (1997). Structuring and Visualising the WWW by Generalised Similarity Analysis. Departement of Computer Studies. Glasgow, UK, Gladgow Caledonian University**:** 10.

Gevrey, J. and S. M. Ruger (2002). Link-based Approaches for Text Retrieval. Departement of Computing. London, Imperial College of Science, Technology and Medicine**:** 7.

Hawking, D. and N. Craswell (2004). Very Large Scale Retrieval and Web Search. Canberra Act, Commonwealth Scientific and Industrial Research Organisation (CSIRO)**:** 27.

Hawking, D., E. Voorhees, et al. (2000). Overview of the TREC-8 Web Track.
Departement of Computer Science
CSIRO Mathematical and Information Sciences
Natioinal Institute of Standards and Technology. Canberra, Australia
Gaithersburg, Maryland, The Australian National University (ANU)**:** 18.

Langville, A. N. and C. D. Meyer (2004). Deeper Inside PageRank**:** 46.

Lempel, R. and S. Moran (2000). The stochastic approach for link-structure analysis (SALSA) and the TKC effect. Department of Computer Science. Haifa, Israel**:** 15.

Liu, K.-L., W. Meng, et al. (2000). Discovery of Similarity Computations of Search Engines. Conference of Information and Knowledge Management (CIKM), McLean, VA USA, ACM Press.

Maton, M. E. (1959). On Relevance, Probabilistic Indexing and Information Retrieval. Santa Monica, California, The RAND Corporation**:** 216 - 243.

Nilsson, N. J. (1998). Artificial Intelligence: A New Syntesis. San Francisco, California, Morgan Kaufmann Publishers, Inc.

Prakash, K. S. S. Relevancy in Search Engines: A knowledge Centric Approach. Chennai, India, Indian institute of Technology Madras**:** 10.

Soboroff, I. (2002). Does WT10g Look Like the Web? Special Interest Ground on Information Retrieval (SIGIR), Tampere, Finland, ACM Press.

Soboroff, I. (2003). Do TREC Web Collections Look Like the Web? National Institute of Standards and Technology (NIST). Geithersburg, Maryland.

Voorhes, E. M. and D. Harman (1999). Overview of the Eight Text REtrieval Conference. TREC-8, Gaithersburg, Maryland, National Institute of Standards and Technology (NIST).

Yaltaghian, B. and M. Chignell Re-Ranking Search Results using Network analysis A Case Study with Google. Toronto, Ontario, Interactive Media Laboratory Bahen Center for Information Technology University of Toronto**:** 10.