Automatisk planlegging i oljeindustrien

av

Teis Lindemark

v0.1

AVHANDLING

for en grad av

MASTER OF SCIENCE

Master's Avhandling, Institutt for informatikk

Fakultetet for teknologi og realfag
Universitetet i Bergen

Juni 2012

Fakultetet for teknologi og realfag
Universitetet i Bergen

Sammendrag

TEST

Innhold

Forord

Nomenclature

AI	Kunstig intelligens (engelsk: artificial inteligence)
Concert	IBM ILOG Concert Technology
CP	Constraint programming
Scheduler	IBM ILOG Scheduler
Solver	IBM ILOG Solver

1 Introduksjon

I dette prosjektet vil utvikleren bli gitt et eksisterende ILOG Scheduler program og benchmarksett. Det eksisterende ILOG Scheduler programmet skal utvides med flere ressurser og benchmarksettene skal kjøres for å kunne sammenligne løsningene med de nye resurssene mot de løsningene uten de nye resurssene.

Denne skriver jeg tilslutt tenker jeg

1.1 Bakgrunn

I dette prosjektet, verktøyene som er brukt for å utføre eksperimenter på emnet er ILOG Scheduler som er endel av IBM sitt ILOG CP. ILOG Scheduler er et C++ bibliotek som gjør det mulig å definere planleggingsbegrensninger i form av ressurser og aktiviteter. Planlegging er en prosess ved å tildele ressurser til aktiviteter og tildele en tid til aktiviteter så det ikke er noen konflikt med begrensingene.[5] Automatisk planlegging er endel av det som kalles kunstig intelligens (AI) .

I prosjektet vil også utvikleren evaluere løsningene og sammenligne løsningene med og uten tilleggsbegrensinger.

Det overordnede målet med denne avhandlingen er å utvide ILOG Scheduler løsningen til Bård Henning Tvedt med flere resursser for å sjekke om det å legge til flere ressurser vil gi bedre og flere løsninger enn de opprinnelige løsningene til Bård Henning Tvedt .

1.1.1 Relatert arbeid

Skrive om tidligere gjort forskning på området.

1.2 Målet med prosjektet

Målet med prosjektet er todelt, og består i å vurdere den modifiserte problemstillingen mot den opprinnelige i forhold til:

- minimere *makespan*
- antall begrensninger
- implementasjon i ILOG Scheduler

I den opprinnelige problemstillingen vil noen aktiviteter være relativt lite begrenset. Dette gjør at løsningsrommet er stort, og traverseringen opp og ned i

søketreet tar lang tid. På tross av et antatt stort løsningsrom så sliter den ILOG Scheduler implementerte løsningsstrategien med å finne løsninger i mange av probleminstansene.

- Vil flere begrensninger gjøre det lettere å finne en løsning?
- Er det noe spesielt med akkurat disse instansene eller er det implementasjon i ILOG Scheduler som er årsaken?

1.3 Beskrivelse av kommende kapitler

I kapittelet om metode, vil fremgangsmåten for prosjektet bli lagt frem og hvordan løsningene har blitt evaluert. I kapittelet om eksperimenteringen vil det legges frem løsninger ved forskjellige strategier og med og uten ressursene. Her vil også løsningene bli kommentert og tilslutt vil det bli sammenfattet en konklusjon over det arbeidet som er gjort.

1.4 Motivasjon

Forskningen er motivert av praktisk erfaring at planleggingsproblemer er veldig aktuelt i bedrifter og i samfunnet idag. Det er ikke bare i oljeindustrien som jeg fokuserer oppgaven på hvor planleggingsløsninger er aktuelt, men generelt bemmaningsproblematikken som alle personalavdelinger sitter med i det daglige. I hverdagen er det også mange planleggingsproblemer fra buss- og togtabeller til personlige gjøremål med å få tid til alt man skal ha tid til.

Planlegging i oljeindustrien er viktig for å på en mest mulig effektiv måte benytte seg av de ressursene som er tilgjengelig til enhver tid, samtidig som visse begrensinger blir fastsatt med tanke på sikkerheten. Det er mye penger involvert i olje- og gassindustrien og det å utføre aktiviteter på en ineffektiv måte kan koste selskapene veldig mye penger. Det er derfor viktig å ha gode løsninger for å ta seg av planleggingen av aktivitetene og ressursene. Operatører innen olje- og gassektorens mål er å minimere antallet farlige situasjoner, minimere miljømessige fotspor og maksimere produksjon. Det å imøtekomme disse kompliserte og utfordrende målene, kan det i enkelte situasjoner oppstå konflikter.

Spesifikke planleggingssituasjoner har forskjellige forutsetninger i forhold til om plattformen er offshore eller på land. Offshore kan mange aktiviteter bli utført på et relativt lite lukket område, mens plattformer på land kan ha mange aktiviteter utført på et relativt stort område. Selvom forutsetningene for disse to typene plattformer er ganske forskjellige, så har de flere likhetstrekk som:

- størrelse - antall aktiviteter kan være noen hunder til titusener, for å gjøre planleggingen interaktiv.
- kompleksitet - et stort antall av begrensningene som skal bli gjennomført, gjør en mulig planlegging vanskelig.
- dynamikk - avhengigheter som vær, logistikk og utstyr som feiler kan avbryte planleggingen.

1.4.1 Kort om begrensningsprogrammering

Begrensningsprogrammering er en programmeringsparadigme hvor relasjoner mellom variable blir satt i form av begrensninger. Begrensninger er en form for deklarativ programmering, som skiller seg fra den mer vanlige imperativ programmeringsspråk[1] ved at løsningen blir til ved å tilfredsstille begrensningene. Det er forskjellige områder i begrensningsprogrammering som "Constraint Satisfaction problems"og planleggingsproblemer. Det mest kjente planleggingsproblemet er "Job Shop Scheduling".[1]

1.4.2 Utfordringer med begrensningsprogrammering

Utfordringer med CP

1.4.3 Begrensingsprogrammeringsverktøy idag

Det finnes idag flere forskjellige verktøy for begrensningsprogrammering, både i form av egne programmeringsspråk som er skreddersydd for begrensningsprogrammering og biblioteker til godt kjente programmeringsspråk som Java[2] og C++[3]. I begge disse kategoriene så finnes det løsninger som er kommersielle og med åpen kildekode. Noen eksempler på egne programmeringsspråk for begrensningsprogrammering er Prolog og Comet[4]. Sistnevnte er et programmeringspråk for begrensningsprogrammering med lokalt søk og er en kommersiell løsning. Eksempler på begrensningsprogrammeringsbiblotek så er det IBM ILOG CP.

Det er ingen løsninger som passer uansett hva slags del innenfor begrensningsprogrammering du skal gjøre. Logisk programløsning eller planleggingsløsning må tas med når det skal bestemmes hvilke verktøy som brukes.

[1] Imperativ programmeringsspråk har sekvenser med som blir utført.
[2] Java
[3] C++
[4] Comet

1.5 Problembeskrivelse

Problemstillingen tar utgangspunkt i den opprinnelige problemstillingen til Bård Henning Tvedt . Problemet er på en innbilt oljeplattform inndelt i et sett av lokasjoner. Utstyr som er krevd for vedlikehold er tilfeldig plassert rundt på plattformen, og ulike aktiviteter skal bli planlagt. Aktivitetene blir opprettet med et gitt sett av ressurskrav og muligens avhengigheter til andre aktiviteter. Alle aktiviteter krever et mannskap til å utføre dem og en lokasjon til å bli utført på. I tillegg krever noen aktiviteter kranressurser, fordi tung løfting er involvert. Mannskap- og kranressurser er knappe, som betyr at de er begrenset tilførsel.

En utvidet problemstilling med ekstra ressurser på beliggenhet, hvor hvert mannskap avgir en varme og hver lokasjon har en gitt varmekapasitet. Summen av varme kan ikke overstige varmekapasiteten.

$$\forall t, l : \sum \{c_{heat}(Crew_j) \mid t \in [v_{sta}(Act_i), w_{end}(Act_i)) \wedge c_{crew}(Act_i) = Crew_j$$

$$\wedge c_{loc}(Act_i) = Loc_l]\} \leq c_{heatcap}(Loc_l)$$

I tillegg kan en beliggenhet ha begrensninger på hvor mange av et gitt mannskap, som kan arbeide på en lokasjon samtidig.

$$\forall t, l : \sharp\{Crew_j \mid t \in [v_{sta}(Act_i), w_{end}(Act_i)) \wedge c_{crew}(Act_i = Crew_j)]$$

$$\wedge c_{crewlimit}(Loc_l) = Crew_j\} \leq c_{crewcapacity}(Loc_l)$$

Antallet beliggenheter er redusert fra 25 i de opprinnelige problemene til 10 i de modifiserte. Med aktiviteter spredt utover 25 beliggenheter, så ville det vært så få aktiviteter på hver beliggenhet at varmekapasiteten til en lokasjon aldri ville blitt oversteget. Det er ikke sjekket om en reduksjon til 10 beliggenheter er tilstrekkelig. Målet for løsningen er å minimalisere makespan, den totale tiden på å fullføre alle aktivitetene.

2 Metode

I denne delen, blir verktøy og teknologier som er brukt i prosjektet beskrevet. I tilegg vil forskingsmetoder som er brukt bli beskrevet og beskrivelse av strategiene for evaluering av løsningene.

2.1 Verktøy brukt i prosjektet

De følgende verktøyene og teknologiene utviklet av IBM var brukt for å gjennomføre formålet med prosjektet.

2.2 Kort om IBM ILOG Concert Technology

Concert er et C++ bibliotek med funksjoner som gir mulighet til å designe modeller av problemer innen matematisk programmering og innen begrensningsprogrammering. Det er ikke noe eget programmeringsspråk, som da gir muligheter til å bruke datastrukturer og kontrollstrukturer som allerede finnes i C++. Igjen så gir det gode muligheter til å integrere Concert i allerede eksisterende løsninger og systemer. Alle navn på typer, klasser og funksjoner har prefiksen Ilo.

De enkleste klassene (eks. IloNumVar og IloConstraint) i Concert har også tilhørende en klasse med matriser hvor matrisen er instanser av den enkle klassen. Et eksempel på det er IloConstraintArray er instanser av klassen IloConstraint.[4]

Concert gjør det mulig å lage en modell av optimaliseringsproblemer uavhengig av algoritmene som er brukt for å løse det. Det tilbyr en utvidelse modelerings lag tatt fra flere forskjellige algoritmer som er klare til å brukes ut av boksen. Dette modeleringslaget gjør det mulig å endre modellen uten å skrive om applikasjonen.[3]

2.3 Kort om IBM ILOG Solver

IBM ILOG Solver er et C++ bibliotek utviklet for å løse komplekse kombinatoriske problemer innen forskjellige områder. Eksempler på anvendelsesområder kan være produksjonsplanlegging, resurs tildeling, timeplanplanlegging, personellplanlegging, osv. Solver er basert på Concert. Som i Concert, så er heller ikke Solver noe eget programmeringsspråk, som gir mulighetene til å bruke egenskapene til C++.

Det å gjøre det enkelest mulig å omgjøre applikasjoner fra plattformer til plattformer, Solver og Concert utelukkes karaktertrekk som skiller seg fra forskjellige

systemer. Av den grunn, anbefales det å bytte ut de enkle typene i C++ med ILOG sine egne:

- IloInt som er signed long integers
- IloAny som er pekere
- IloNum som er double presisjon floating-point verdier
- IloBool som er boolean verdier: IloTrue og IloFalse

Solver bruker begrensningsprogrammering for å finne løsninger til optimaliseringsproblemer. Det å finne løsninger med Solver er basert på tre steg: beskrive, modell og løse. De tre stegene nærmere forklart følger:

Først må problemet beskrives i programmeringsspråket som brukes.

Det andre steget er å bruke Concert klassene for å opprette en modell av problemet. Modellen blir da satt sammen av besluttningsvariable og begrensninger. Besluttningsvariablene er den ukjente informasjonen i problemet som skal løses. Alle besluttingsvariablene har et domene med mulige verdier. Begrensningene setter grensene for kombinasjonene av verdier for de besluttingsvariablene.

Det siste steget er å bruke Solver for å løse problemet. Det inneholder å finne verdier for alle besluttingsvariablene samt ikke bryte noen av de definerte begrensningene og dermed enten maksimere eller minimere målet, hvis det er et mål inkludert i modellen. Solver ser etter løsninger i et søkeområdet. Søkeområdet er alle mulige kombinasjoners av verdier.[3]

2.4 Kort om IBM ILOG Scheduler

IBM ILOG Scheduler hjelper med å utvikle problemløsnings-applikasjoner som krever behandling av ressurser fordelt på tid. Scheduler er et C++ bibliotek som baserer seg på Solver, og som Solver, så gir det alle mulighetene med objektorientering og begrensningsprogrammering. Scheduler har spesifisert funksjonalitet på å løse problemer innen planlegging og ressurs tildeling.[2]

2.5 Forskningsmetoder

Forskningsmetoden som er brukt i prosjektet er å eksperimentere med implementasjonen av ressursene og løsningsstrategien.

2.5.1 Implementeringsprosessen

Første delen av implementeringsprosessen er å utvide modellen for å inkludere de nye ressursene. For å gjøre det eksperimenterte jeg litt med Scheduler for prøve å finne den beste måten å implementere det på.

Den andre delen av implementeringsprosessen er å eksperimentere med forskjellige løsningstrategier gitt av i Solver. Her er det mange forskjellige kombinasjoner og rekkefølgen de forskjellige strategiene (i ILOG; Goals) kan ha en innvirkning på løsningen tilslutt.

Tilslutt så kommer det an på tidsgrensen som blir satt på hvor lenge ILOG skal søke etter bedre løsninger.

Denne delen skal utvides ytterligere.

2.5.2 Evaluering av prosessen

Prosessen med eksperimenteringen av implementasjonen blir evaluert ved å undersøke makespan opp mot teoretisk øvregrense og teoretisk nedregrense i både løsningene uten tilleggsressurser og med tilleggsressurser. Løsningene fra prosessen med og uten tilleggsresursene vil også bli evaluert opp mot hverandre. Dette innebærer bruk av kvantitative metoder.

Forskningsmetoden vil bli evaluert ved å bruke genererte benchmarksett, som er generert av et eksternt program. Benchmarksettene som genereres kan bestemmes hvor mange av de forskjellige ressursene som skal være med i benchmarksettet. I dette prosjektet er det et sprang på 50 - 5000 aktiviteter som implementasjon blir evaluert på.

2.6 Evalueringsstrategi

3 Eksperimenter

Eksperimenteringen er utført på en Mackbook Air med 1.8 GHz Intel Core i7 prosessor og 4 GB 1333 MHz DDR3 minne. Det er totalt sett 71 benchmarksett som implementasjonene er evaluert på, hvor mange det ble funnet en løsning varierte med eller uten tilleggsressursene og tidsgrensen som var satt.

3.1 Uten tilleggsressurser

3.2 Med tilleggsressurser

4 Fremtidig arbeid

5 Konklusjon

6 Vedlegg

Benchmark	Teoretisk nedregrense	Teoretisk øvregrense bound	Makespan
Act1000Loc10Crew5Crane2_1	416	3529	449
Act100Loc10Crew5Crane2_1	50	356	50
Act100Loc10Crew5Crane2_2	66	364	67
Act100Loc10Crew5Crane2_3	55	352	56
Act200Loc10Crew5Crane2_1	92	711	104
Act200Loc10Crew5Crane2_2	124	742	124
Act200Loc10Crew5Crane2_3	66	704	89
Act200Loc10Crew5Crane2_4	71	734	109
Act200Loc10Crew5Crane2_5	88	698	93
Act300Loc10Crew5Crane2_1	144	1049	144
Act300Loc10Crew5Crane2_4	137	1071	148
Act400Loc10Crew5Crane2_1	177	1400	192
Act400Loc10Crew5Crane2_3	198	1369	198
Act400Loc10Crew5Crane2_4	168	1362	187
Act400Loc10Crew5Crane2_5	136	1361	190
Act500Loc10Crew5Crane2_2	234	1723	257
Act500Loc10Crew5Crane2_4	212	1798	243
Act600Loc10Crew5Crane2_2	282	2201	297
Act500Loc25Crew5Crane2_5	223	1708	223
Act50Loc10Crew5Crane2_1	28	177	34
Act50Loc10Crew5Crane2_4	21	172	22
Act50Loc10Crew5Crane2_5	22	177	27
Act60Loc10Crew5Crane2_1	29	211	35
Act60Loc10Crew5Crane2_2	31	215	39
Act60Loc10Crew5Crane2_3	31	209	38
Act60Loc10Crew5Crane2_4	27	203	27
Act60Loc10Crew5Crane2_5	27	205	46

Tabell 1: Løsninger uten tilleggsressurser og tidsgrense på 10 sekunder

Benchmark	Teoretisk nedregrense	Teoretisk øvregrense	Makespan
Act500Loc25Crew5Crane2_5	223	1708	223
Act50Loc10Crew5Crane2_1	28	177	34
Act50Loc10Crew5Crane2_4	21	172	23
Act50Loc10Crew5Crane2_5	22	177	29
Act60Loc10Crew5Crane2_1	29	211	32
Act60Loc10Crew5Crane2_2	31	215	41
Act60Loc10Crew5Crane2_3	31	209	36
Act60Loc10Crew5Crane2_4	27	203	27
Act60Loc10Crew5Crane2_5	27	205	46

Tabell 2: Løsninger med tilleggsressurser og tidsgrense på 10 sekunder

Referanser

[1] Wikipedia. `http://en.wikipedia.org/wiki/Constraint_programming`.

[2] IBM. *IBM ILOG Scheduler V6.8*. IBM.

[3] IBM. *IBM ILOG Solver V6.8*. IBM.

[4] ILOG. *ILOG Concert Technology 2.0 Reference Manual*. ILOG.

[5] Claude Le Pape. Implementation of resource constraints in ilog schedule: A library for the development of constraint-based scheduling systems. *Intelligent Systems Engineering*, 3:55–66, 1994.