Opbygning af C#-program til bestemmelse af nucleolus

Bachelorprojekt på Erhvervsøkonomi-Matematik-studiet

Copenhagen Business School (Handelshøjskolen i København)

Bachelorprojekt for

Niels Madsen Kloster

ved vejleder

Jens Leth Hougaard

afleveret 2. Juni, 2009

Indholdsfortegnelse

Indledning – om kooperative spil.

Kooperative spil findes overalt. Så snart nogen har grund til at samarbejde, er der også brug for at fordele de fordele, der er kommet ud af samarbejdet. Det kan være lige fra børn, der samler deres sparepenge sammen for at købe en større pose slik eller et dyrere spil, som herefter skal fordeles eller udlånes mellem dem og op til større institutioner som EU, der fordeler forskellige tilskud, som er fremkommet af EU-landenes samarbejdsfordele. Et andet typisk eksempel er fremkommet som konsekvens af en øget handel over internet, hvor der for det meste er tilknyttet nogle forsendelsesomkostninger, hvorfor man til tider vælger at lægge en større ordre sammen med venner, der har brug for den samme type vare. Ved forholdsvis små modeller som med børnene og internethandel, er der typisk ikke grundlag for at lave dybdegående beregninger for at fordele omkostningerne, og en fair model kan for det meste hurtigt nås. Det kan være en lige fordeling eller en form for vægtning af over antallet af modtagne varer eller lignende.

Ideen, som nucleolus-løsningen bygger på er, at en del af spillerne i en koalition måske ikke har særlig stor fordel af at indgå i den samlede koalition. Heraf kommer forslaget om at finde den løsning, hvor den mindste forbedring gøres så stor som mulig. Hvis vi igen ser på eksemplet med internethandel, hvor forsendelsesomkostningerne skal fordeles, vil en typisk model være at fordele omkostningerne med en vægtning, som er proportionel med værdien af de modtagne varer. Dvs., at de personer, som har købt mest kommer også af med mest i forsendelsesomkostninger. Det er dog ofte således, at hvis nogle få personer hver har lagt en stor ordre, bliver fordelen ved at medtage flere ordrer, som er forholdsvis små, ikke særlig stor, mens personer, som vil bestille ganske få varer, får en stor fordel af at komme med i den samlede bestilling. Her findes nucleolus-løsningen så, som nævnt tidligere, ved at finde den største fordel for de værst stillede, det vil sige ved at forøge fordelen for de store købere, og formindske fordelen for de små købere.

Problemformulering

Målet med denne opgave er at opstille en metode til at finde nucleolus-løsningen for kooperative spil, som kan gennemføres systematisk, for efterfølgende at lave et program i programmeringssproget C# der kan gennemføre denne metode. Det er svært at finde empiriske eksempler, som kan benyttes til beregninger, så programmet vil blive sammenlignet med regneeksempler der har angivet en løsning for nucleolus, og andre løsningskoncepter.

Teori

Grundlæggende begreber:

De gevinster eller afgifter, som spillerne i en koalition har, angives i en payoff-vektor $x=\{x_1;x_2;\ldots;x_n\}$, for n spillere. Den samlede gevinst eller de samlede afgifter for koalitionen er angivet ved *v(N)*, hvor gevinster skal fordeles, kaldet værdispil eller *c(N)*, hvor omkostninger skal fordeles, kaldet omkostningsspil. Spillerne deles om de gevinster eller de afgifter, som er vedhæftet koalitionen. Dvs. $\sum_{i\in N} x_i=v(N)$ eller $\sum_{i\in N} x_i=c(N)$. En koalition dannet af en mindre mængde af spillerne kaldes en subkoalition. De spillere der tilhører subkoalitionen, S, vil foretrække, hvis gevinsten eller omkostningerne er hhv. større end eller mindre end det de får fra den samlede koalition, N. Dvs. hvis $\sum_{i\in S} x_i<v(S)$ eller $\sum_{i\in S} x_i>c(S)$. Når to vilkårlige subkoalitioner, som ikke har nogen spiller til fælles, kan slåes sammen for derved at opnå en bedre koalition, kaldes spillet superadditivt for værdispil og subadditivt for omkostningsspil. Det kan opstilles som hhv. $v(S\cup T)\geqslant v(S)+v(T)$ og $c(S\cup T)\leqslant c(S)+c(T)$ og betyder, at en koalition vil drage fordel af at medtage flere spillere. Hvis to vilkårlige koalitioner, der har spiller til fælles kan drage fordel af at bryde koalitionerne for at danne en samlet koalition kaldes spillet konvekst for værdispil og konkavt for omkostningsspil. Det kan opstilles som hhv. $v(S\cup T)+v(S\cap T)\geqslant v(S)+v(T)$ og $c(S\cup T)+c(S\cap T)\leqslant c(S)+c(T)$, og det ses, at dette er identisk med superadditivitet og subadditivitet for subkoalitioner, hvor det gælder, at $S\cap T=\emptyset$.

Løsningskoncepter:

Teorien gennemgåes herunder for værdispil. Hvor opstillingen af formelerne afviger ved omkostningsspil er disse opstillet i grå baggrund, og begreber i teksten, som er anderledes for omkostningsspil, skrives i kantet parentes, []. Der forekommer undtagelser til denne opstilling i nogle eksempler, og det vil da fremgå af den tilhørende tekst.

Core(Kernen)

Kernen er mængden af de mulige payoffs for en koalition, som ikke giver nogen spiller eller subkoalition af spillere grund til at forlade koalitionen.

Disse kernebetingelser kan opstilles matematisk som

$$core(N,v)=\{x\in\mathbb{R}^n|\sum_{i\in N}x_i=v(N),\sum_{i\in S}x_i\geq v(S),\forall\ S\subset N\}$$

$$core(N,c)=\{x\in\mathbb{R}^n|\sum_{i\in N}x_i=c(N),\sum_{i\in S}x_i\leq v(S),\forall\ S\subset N\}$$ [1]

Der kan forekomme tilfælde, hvor man ønsker at undersøge værdi- eller omkostningsfordelinger i en koalition, som kan opnå en bedre payoff vektor ved at dele sig i subkoalitioner. Her vil kernebetingelserne ikke være opfyldt, og kernen vil derfor være tom.

Man kan teste om kernen er tom eller ej med Bondavera-Shapley sætningen. Før man kan gøre dette må man dog definere det man kalder balancerede vægte. For hver koalition indføres en vægt sådan at summen af vægtene for de koalitioner en given spiller optræder i er normeret til 1.

Matematisk ser det ud på følgende måde:

$$\sum_{j:i\in S_j}\delta_j=1,\forall\ i\in N$$ [2]

Bondavera-Shapley sætningen siger så, at kernen er ikke-tom hvis og kun hvis der, for ethvert system af balancerede vægte gælder, at $\sum_{S\subset N}\delta_S v(S)\leq v(N)$ [3]

Hvis man ser på et eksempel med 3 spillere, gælder det altså, at de balancerede vægte er givet ved de tre ligninger:

$$\begin{aligned}&\delta_1+\delta_{12}+\delta_{13}=1,\\&\delta_2+\delta_{12}+\delta_{23}=1,\\&\delta_3+\delta_{13}+\delta_{23}=1.\end{aligned}$$ [4]

Bondavera-Shapley sætningen leder os nu frem til, at kernen i dette tilfælde er ikke-tom hvis og kun

1 Jens Leth Hougaard – An Introduction To Allocation Rules s. 85
2 Ibid s. 86
3 Ibid s. 86
4 Jens Leth Hougaard – An Introduction To Allocation Rules s. 87

hvis,

$v(1)+v(2,3)\leqslant v(1,2,3),$	$c(1)+c(2,3)\geqslant c(1,2,3),$
$v(2)+v(1,3)\leqslant v(1,2,3),$	$c(2)+c(1,3)\geqslant c(1,2,3),$
$v(3)+v(1,2)\leqslant v(1,2,3),$	$c(3)+c(1,2)\geqslant c(1,2,3),$
$v(1)+v(2)+v(3)\leqslant v(1,2,3),$	$c(1)+c(2)+c(3)\geqslant c(1,2,3),$
$\frac{1}{2}(v(1,2)+v(1,3)+v(2,3))\leqslant v(1,2,3).$	$\frac{1}{2}(c(1,2)+c(1,3)+c(2,3))\geqslant c(1,2,3).$

[5]

Det ses, at de fire første linjer er direkte sammenhængende med superadditivitet [subadditivitet], hvilket derfor er en nødvendighed for at have en ikke-tom kerne. I øvrigt ses det, at alle linjer er opfyldt hvis spillet er konvekst [konkavt], men dette er dog ikke nødvendigt.

Ved spil med flere spillere, vil der opstå flere linjer af samme type som den nederste i det ovenstående.

For eksempel med 4 spillere:

$$\frac{1}{3}(v(1,2,3)+v(1,2,4)+v(1,3,4)+v(2,3,4))\leqslant v(1,2,3,4)$$
$$\frac{1}{2}(v(1,2)+v(2,3)+v(3,4)+v(1,4))\leqslant v(1,2,3,4)$$
$$\frac{1}{3}(v(1,2)+v(1,3)+v(1,4)+2*v(2,3,4))\leqslant v(1,2,3,4)$$
$$\frac{1}{2}(v(1,4)+v(1,2,3)+v(2,3,4))\leqslant v(1,2,3,4)$$

[6]

$\ldots$ *OSV.*

For at teste om spillet er balanceret skal der altså, udover superadditivitet [subadditivitet], testes for opfyldelse af flere ligninger, hvilket hurtigt bliver et omfattende problem.

Eksempel:

3 *spillere*

$N=1,2,3 \quad v(N)=6 \quad v(\varnothing)=0$

$v(1,2)=v(1,3)=4 \quad v(2,3)=3$

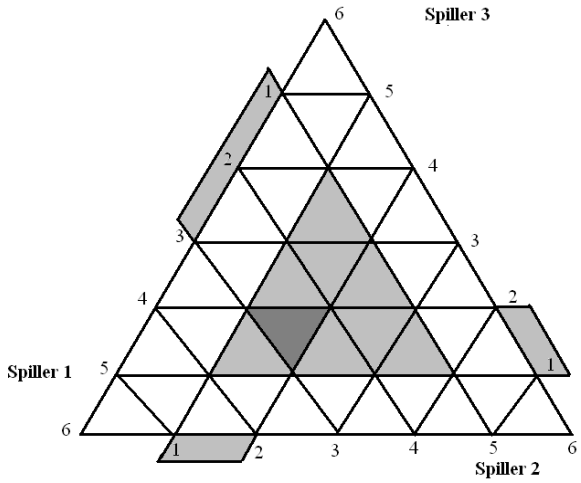
$v(1)=v(2)=v(3)=1$

I dette simple eksempel ses det, at spillet er superadditivt, det er ikke konvekst, men det er alligevel balanceret.

Spil med 3 spillere kan desuden illustreres grafisk. Dette spil kan se ud på følgende måde:

5 Ibid s. 87

6 Guillermo Owen



Inden for de lysegrå område er der ingen spillere der har grund til at forlade den samlede koalition for at arbejde alene. Man skal dog ind i det mørkegrå område for at spillerne hverken vil forlade koalitionen alene eller i samarbejde med en anden spiller, og det er dette område der er kernen.

Kernel

Et koncept til løsning af forhandlingsproblemet er kernel. For at nå frem til en definition af kernel, er det først nødvendigt, at definere to andre koncepter, nemlig excess og surplus.

Excess

Excess er et mål for den fordel en spiller eller gruppe af spillere kan opnå ved at gå med i en given koalition:

$$e(S,x)=v(S)-\sum_{i\in S} x_i \qquad e(S,x)=c(S)-\sum_{i\in S} x_i$$ [7]

Hvis værdien [omkostningerne] af koalitionen S er større [mindre] end summen af de payoffs spillerne får i det givne løsningsforslag, vil de med fordel kunne forlade deres nuværende koalitionsformation og danne koalitionen S. Så længe en payoff-vektor har positive [negative] excess er den ikke et optimalt valg for dens spillere.

Surplus

Surplus er en sammenligning af to spillere, og angiver den største excess en spiller kan håbe på at opnå uden samarbejde fra den anden spiller:

7 Guillermo Owen s. 319

$s_{ij}(x) = max\, e(S, x)$, $s_{ij}(x) = min\, e(S, x)$, [8]

hvor max findes blandt de koalitioner S, som i er en del af, og som j ikke er en del af. Hvis $s_{ij}(x) > s_{ji}(x)$ [$s_{ij}(x) < s_{ji}(x)$], vil spiller i være i en favorabel forhandlingsposition over spiller j, og kan derfor kræve noget værdi fra spiller j, og payoff-vektoren vil kunne ændres.

Kernel kan nu defineres, som de løsninger, hvor ingen spiller har en forhandlingsfordel over andre spillere. Dvs. hvor $s_{ij}(x) = s_{ji}\ \forall\ i, j \in N$.

Nucleolus

Nucleolus er en et unikt løsningsforslag for payoff-vektoren, som igen benytter konceptet excess. For en given payoff-vektor, x , kan der opstilles en vektor, $\theta(x)$, der indeholder excess for alle koalitioner i spillet, og hvor alle elementer i vektoren er arrangeret med faldende [stigende] værdi. Hvis man, som det typisk er eksemplet, undersøger fordelingen af den samlede koalitions værdi blandt alle dens deltagere og antager, at en tom koalition er værdiløs, kan man se bort fra disse, og opstille vektoren for alle ikke-tomme subkoalitioner af den samlede koalition.

Eksempel (Taget fra eksempel i Owen side 322)

$v(S) = 1$, for to eller tre spillere, og 0 ellers.

payoff vektor $x = (0{,}3\,;\, 0{,}5\,;\, 0{,}2)$ *payoff vektor* $y = (0{,}1\,;\, 0{,}5\,;\, 0{,}4)$

S	$v(S)$	$e(S, x)$	$e(S, y)$
$1\,;2$	1	0,2	0,4
$1\,;3$	1	0,5	0,5
$2\,;3$	1	0,3	0,4
1	0	−0,3	−0,1
2	0	−0,5	−0,5
3	0	−0,2	−0,4

For dette eksempels to payoff-vektorer, x og y , kan der nu opstilles vektorerne $\theta(x)$ og $\theta(y)$:

$\theta(x) = (0{,}5\,;\, 0{,}3\,;\, 0{,}2\,;\, -0{,}2\,;\, -0{,}3\,;\, -0{,}5)$
$\theta(y) = (0{,}5\,;\, 0{,}4\,;\, 0{,}1\,;\, -0{,}1\,;\, -0{,}4\,;\, -0{,}5)$

Disse vektorer kan nu arrangeres således, at den værst stillede subkoalition kommer til at få det bedst mulige udgangspunkt. Dvs. den payoff-vektor, hvor muligheden for den største forbedring i en subkoalition er mindst vil rangere højest.For dette eksempel vil koalitionen mellem spiller 1 og

8 Guillermo Owen s. 319

spiller 3 give den største forbedring i begge tilfælde idet de to spillere kan inddrage og fordele den payoff på 0,5, som ellers vil tilkomme spiller 2 i de to payoff-vektorer. Da denne forbedring er lige stor i de to tilfælde ses der herefter på den næst-dårligst stillede subkoalition, som er hhv. mellem spiller 2 og spiller 3 ved payoff-vektoren x, og mellem spiller 1 og spiller 2 i payoff-vektoren y. Her ses det, at det er payoff-vektoren x, der giver anledning til den mindste forbedring, og det er derfor $\theta(x)$, som rangerer højest.

Nucleolus er så den payoff-vektor, x, der hører til den $\theta(x)$, som rangerer højest blandt alle de mulige payoff-vektorer.

Det lineære program

Nucleolus kan identificeres ved lineær programmering i flere gennemløb. Hvis man kan identificere den største [mindste] excess-værdi, α_1, for nucleolus vil der gælde, at $e(S,x) \leqslant \alpha_1, \forall\, x \in X, S \subset N$ [$e(S,x) \geqslant \alpha_1, \forall\, x \in X, S \subset N$], hvilket kan omskrives til $\sum_{i \in S} x_i + \alpha_1 \geqslant v(S), \forall\, S \subset N$ [$\sum_{i \in S} x_i + \alpha_1 \leqslant c(S), \forall\, S \subset N$].

Den største [mindste] α-værdi kan findes ved at løse det lineære program

$$\begin{array}{c} \textit{minimer}\ \alpha \\ \\ \textit{under bibetingelserne} \\ \sum_{i \in S} x_i + \alpha \geqslant v(S), \quad S \subset N \\ \sum_{i \in N} x_i = v(N) \end{array} \qquad \begin{array}{c} \textit{maksimer}\ \alpha \\ \\ \textit{under bibetingelserne} \\ \sum_{i \in S} x_i + \alpha \leqslant c(S), \quad S \subset N \\ \sum_{i \in N} x_i = c(N) \end{array}$$

Det lineære program for omkostningsspillet kan opstilles på en lidt anden måde idet kriteriefunktionen kun består af α. Dette betyder, at α i stedet kan minimeres, hvis man ændrer fortegnet for α i bibetingelserne, og det lineære program for omkostningsspil bliver da:

$$\begin{array}{c} \textit{minimer}\ \alpha \\ \\ \textit{under bibetingelserne} \\ \sum_{i \in S} x_i - \alpha \leqslant c(S), \quad S \subset N \\ \sum_{i \in N} x_i = c(N) \end{array}$$

Denne opstilling gør, at de to spiltyper skal gennemgå de samme funktioner efter hjælpeproblemet.

Det ses her, at der er tilføjet en ekstra bibetingelse, idet det vides, at spillerne skal deles om værdien [omkostningerne] af den samlede koalition. Hvis dette ligningssystem giver anledning til en entydig løsning, er dette da nucleolus. Hvis det ikke er tilfældet identificeres det sæt af koalitioner A_1 hvor $e(S,x)=\alpha_1$, og den næste α-værdi findes ved at løse det lineære program

$$\begin{array}{c} \textit{minimer}\ \alpha \\ \\ \textit{under bibetingelserne} \\ \sum_{i\in S} x_i+\alpha \geq v(S), \quad S\subset N \\ \sum_{i\in S} x_i+\alpha_1=v(S), \quad \forall\ S\subset A_1 \\ \sum_{i\in N} x_i=v(N) \end{array} \qquad \begin{array}{c} \textit{minimer}\ \alpha \\ \\ \textit{under bibetingelserne} \\ \sum_{i\in S} x_i-\alpha \leqslant c(S), \quad S\subset N \\ \sum_{i\in S} x_i-\alpha_1=c(S), \quad \forall\ S\subset A_1 \\ \sum_{i\in N} x_i=c(N) \end{array}$$

Denne løkke fortsættes indtil en entydig løsning er fremkommet.

Simplex

Det lineære program kan opstilles på flere måder i simplex, alt efter hvordan man vælger at definere problemet. I den simpleste opstilling af simplex kan metoden løse et maksimeringsproblem, hvor ligningerne i bibetingelserne skal holdes under en given værdi. Hvis der er bibetingelser, som er lig med en given værdi, eller som skal holdes over en given værdi, er der brug for et hjælpeproblem, hvor der tilføres kunstige slack-værdier. Dette vil altid være tilfældet her, idet bibetingelsen om spillernes samlede payoff er lig med en given værdi.

Simplex-tableauet for hjælpeproblemet opstilles:

	x_1	⋯	x_n	α	s_1	⋯	s_m	as_1	⋯	as_m	as_{m+1}
0	0			⋯			0	-1	⋯	-1	-1
$v(S_1)$	1*	⋯	0*	1	-1		0	1		0	
⋯	⋮	⋱	⋮	⋮		⋱			⋱		
$v(S_m)$	0*	⋯	1*	1	0		-1			1	
v(N)	1	⋯	1	0			⋯			0	1

	x_1	⋯	x_n	α	s_1	⋯	s_m	as_1	⋯	as_m	as_{m+1}
0	0			⋯			0	0	⋯	0	-1
$c(S_1)$	1*	⋯	0*	-1	1		0	0	⋯	0	
⋯		⋱		⋮		⋱		⋮	⋱	⋮	
$c(S_m)$	0*	⋯	1*	-1	0		1	0	⋯	0	
c(N)	1	⋯	1	0			⋯			0	1

, hvor *s* er slacks, der er tilføjet ulighederne, og *as* er kunstige slacks der er tilføjet hvor slacks er -1 eller hvor der ikke forekommer slacks. Bemærk desuden er området markeret med * ikke en diagonal række af 1-taller, men skal ses som alle mulige kombinationer af 0- og 1-taller.

I første skridt fjernes de negative værdier i øverste række ved rækkeoperationer ved at lægge de der indeholder kunstige slacks til den øverste række:

	x_1	⋯	x_n	α	s_1	⋯	s_m	as_1	⋯	as_m	as_{m+1}
$\sum_{S \subseteq N} v($	2^{n-1}		2^{n-1}	2^n	-1	⋯	-1	0	⋯	0	0
v(S_1)	1*	⋯	0*	1	-1		0	1		0	
⋯		⋱		⋯		⋱			⋱		
v(S_m)	0*	⋯	1*	1	0		-1	0		1	
v(N)	1	⋯	1	0			⋯			0	1

	x_1	⋯	x_n	α	s_1	⋯	s_m	as_1	⋯	as_m	as_{m+1}
c(N)	1	⋯	1	0		⋯		0	⋯	0	0
c(S_1)	1*	⋯	0*	-1	1		0	0	⋯	0	
⋯		⋱		⋯		⋱		⋯	⋱	⋯	
c(S_m)	0*	⋯	1*	-1	0		1	0	⋯	0	
c(N)	1	⋯	1	0			⋯			0	1

Når hjælpeproblemet er løst, dvs. de kunstige slacks er minimeret, løses de oprindelige problem, som et maksimeringsproblem. Det vil sige, at for værdispillet skal funktionen -α maksimeres, og der udføres kun yderligere operationer, hvis der i hjælpeproblemet er fremkommet en basis i søjlen for α. For omkostningsspillet skal α maksimeres. Disse funktioner indsættes som kriteriefunktioner i det tableau, der er fremkommet fra hjælpeproblemet, hvor søjlerne for de kunstige slacks udelades:

	x_1	…	x_n	α	s_1	…	s_m
0	0	…	0	-1	0	…	0
V_1	$x_{1\,1}$	…	$x_{1\,n}$	$x_{1\,n+1}$	$x_{1\,n+2}$		$x_{1\,n+m+1}$
…	…	…	…	…		⋱	
V_m	$x_{m\,1}$	…	$x_{m\,n}$	$x_{m\,n+1}$	$x_{m\,n+2}$		$x_{m\,n+m+1}$
V_{m+1}	$x_{m+1\,1}$	…	$x_{m+1\,n}$	$x_{m+1\,n+1}$	$x_{m+1\,n+2}$	…	$x_{m+1\,n+m+1}$

	x_1	…	x_n	α	s_1	…	s_m
0	0	…	0	-1	0	…	0
C_1	$x_{1\,1}$	…	$x_{1\,n}$	$x_{1\,n+1}$	$x_{1\,n+2}$		$x_{1\,n+m+1}$
…	…	…	…	…		⋱	
C_m	$x_{m\,1}$	…	$x_{m\,n}$	$x_{m\,n+1}$	$x_{m\,n+2}$		$x_{m\,n+m+1}$
C_{m+1}	$x_{m+1\,1}$	…	$x_{m+1\,n}$	$x_{m+1\,n+1}$	$x_{m+1\,n+2}$	…	$x_{m+1\,n+m+1}$

Når denne simplex er løst kan man i øverste række aflæse hvilke rækker der opfylder bibetingelserne uden slack ved at identificere de pladser for slack-værdierne, der indeholder et tal forskelligt fra 0. Der kan godt være andre søjler for slack, som ikke er i basis, men hvis værdien i øverste række er 0, er der plads til yderligere forbedring. For disse ligninger kan man så trække den fundne α-værdi, α_1, fra funktionsværdien og slette slack-variablen. Det næste simplex tableau kan så

opstilles for at finde den næste α-værdi, α_2. Hvis der for eksempel blev fundet en værdi i øverste række under s_k opstilles det nye hjælpeproblem:

	x_1	⋮	x_n	α	s_1	⋮	s_{k-1}	s_k	s_{k+1}	⋮	s_m	as_1	⋮	as_{k-1}	as_k	as_{k+1}	⋮	as_m	as_{m+1}
0	0			⋮							0	-1			⋮			-1	-1
$v(S_1)$	1^*	$⋮^*$	0^*	1	-1	0		⋮			0	1	0			⋮			0
⋯				⋯	0	⋱						0	⋱						
$v(S_{k-1})$				1			-1				⋯	⋯		1					
$v(S_k)$- α_1	$⋯^*$	$⋯^*$	$⋯^*$	0				0				0	⋱		1				
$v(S_{k+1})$				1	⋯				-1							1			
⋯				⋯						⋱	0	⋯					⋱		
$v(S_m)$	0^*	$⋮^*$	1^*	1	0			⋯		0	-1	0			⋮		0	1	0
v(N)	1	⋮	1	0				⋯				0	⋮		⋮			0	1

	x_1	⋮	x_n	α	s_1	⋱	s_{k-1}	s_k	s_{k+1}	⋱	s_m	as_1	⋱	as_{k-1}	as_k	as_{k+1}		as_m	as_{m+1}
0	0			⋮							0	0	⋮	0	-1	0	⋱	0	-1
$c(S_1)$	1^*	$⋮^*$	0^*	-1	1	0		⋯			0	0	0			⋯			0
⋯				⋯	0	⋱						0	⋱						
$c(S_{k-1})$				-1			1				⋯	⋯		0					
$c(S_k)$- α_1	$⋯^*$	$⋯^*$	$⋯^*$	0				0				0	⋱		1				
$c(S_{k+1})$				-1	⋯				1							0			
⋯				⋯						⋱	0	⋯					⋱		
$c(S_m)$	0^*	$⋮^*$	1^*	-1	0			⋯		0	1	0			⋯		0	0	0
c(N)	1	⋯	1	0				⋯				0	⋯		⋯			0	1

Dette fortsættes som nævnt tidligere indtil en entydig løsning er nået.

Programmet – Ses på bilag 2

For hver plads i tableauerne benytter programmet en 64-bit double-variabel. Idet størrelsen på tableauerne bliver ca. 2^{2N+1} er det nødvendigt at overveje hvor stor hukommelse programmet optager. Der er derfor valgt at begrænse programmet til 10 spillere, hvilket giver et maksimalt hukommelsesforbrug på ca. 17 MB per tableau. Programmet benytter 3 af disse tableauer, og allerede ved 15 spillere er den samlede hukommelsesgrænse tæt på 50GB.

Programmet er delt op i 3 klasser, som står for hver sin del af programmet: Dataindsamling, Beregninger og præsentation. En fjerde klasse kaldet **Program** indeholder funktionen *Main*, som er den funktion, der automatisk bliver kørt, og som samler en hovedfunktion fra hver af de 3 andre klasser.

Dataindsamling håndteres af klassen **Input**, som kører hovedfunktionen *MainIn*. Det første der sker er, at der fastsættes en tolerance, som bruges senere i beregnings-klassen, **Calc**. Grunden til, at den fastsættes her er, at der nemt kan laves en ændring, så tolerancen bliver et bruger-input, mens man så stadig holder al dataindsamling i dataindsamlings-klassen, **Input**. Den næste funktion er *Types*, som spørger brugeren, om der er tale om et værdispil eller et omkostningsspil, også selvom brugeren efterfølgende vælger at indlæse data fra en fil. *Types* spørger også brugeren, om resten af dataene skal indlæses fra en fil, eller om de bliver indtastet manuelt.Hvis brugeren vælger, at indtaste alle værdier manuelt, beder programmet først, i funktionen *ManualSize*, om at få angivet antallet af spillere. Dette benyttes til at opstille alle kombinationer af koalitioner i spillet i funktionen *SetupMat*. Den naturlige måde at opstille disse kombinationer på, vil for de fleste være at fastsætte en koalitionsstørrelse, og derefter gennemgå de kombinationer der findes for denne størrelse. En sådan rækkefølge, hvor der startes med den største koalition, vil, for 3 spillere, se således ud:

x_1	x_2	x_3
1	1	1
1	1	0
1	0	1
1	0	0
0	1	1
0	1	0
0	0	1
0	0	0

Funktionen *SetupMat* gennemløber i stedet 3 løkker, som opstiller disse kombinationer som en binær talrække:

x1	x2	x3	Som binære tal
1	1	1	$111 = 1*2^2 + 1*2^1 + 1*2^0 = 7$
1	1	0	$110 = 1*2^2 + 1*2^1 + 0*2^0 = 6$
1	0	1	$101 = 1*2^2 + 0*2^1 + 1*2^0 = 5$
1	0	0	$100 = 1*2^2 + 0*2^1 + 0*2^0 = 4$
0	1	1	$011 = 0*2^2 + 1*2^1 + 1*2^0 = 3$
0	1	0	$010 = 0*2^2 + 1*2^1 + 0*2^0 = 2$
0	0	1	$001 = 0*2^2 + 0*2^1 + 1*2^0 = 1$
0	0	0	$000 = 0*2^2 + 0*2^1 + 0*2^0 = 0$

Dette er dels nemmere at opstille, og dels bliver det også nemmere at identificere rækken for en koalition, når man ved hvilke spillere der indgår i den. Rækken for koalition S er $2^N - \sum_{i \in S} 2^{N-i}$.

Funktionen sørger desuden for at opstille en søjle med α-værdier og de slacks der, som udgangspunkt, er brug for. For værdispil er søjlen med α-værdier positiv, og der tilføjes en diagonal række med negative slack-værdier og en diagonal række med positive kunstige slack-værdier. For omkostningsspil er søjlen med α-værdier negativ og der tilføjes kun en diagonal række med positive slack-værdier. For begge typer af spil gælder det, at den fulde koalition ikke har en α-værdi, men altid en kunstig slack-værdi.

Brugeren kan herefter indtaste nogle generelle koalitionsværdier for koalitioner af en vis størrelse via funktionen *ManualValueGen*. Funktionen gennemløber alle rækker og tilføjer værdierne for koalitioner der indeholder det antal spiller som er angivet. Når dette er overstået har brugeren mulighed for at angive værdier for specifikke koalitioner i funktionen *ManualValuesSpec,* ved at angive mængden af spiller der indgår i den. Rækken for koalitionen identificeres ved hjælp af formlen, der ses ovenfor.

Hvis brugeren vælger, at indlæse data fra en fil, bliver brugeren, i funktionen *FileIn*, bedt om at indtaste placeringen for filen, og filen bliver indlæst i et array af strenge, som er opdelt ved hvert linieskift. Herefter opdeles hvert element i to, hvor der deles ved tabulator tegn. Antallet af spillere identificeres i funktionen *FileSize* ved at aflæse det første element i arrayet af strenge. *SetupMat* kører på samme måde som ved den manuelle indtastning. Værdierne for koalitionerne bestemmes

med funktionen *FileValues* på samme måde som ved *ManualValuesSpec* hvor koalitionernes spiller er det første element i hver linje og hvor koalitionernes værdier er angivet efter tabulator tegnet i hver linje.

For at indlæse værdispillet fra G. Owen side 332 kan filen se således ud:

```
4
1,2,3,4	100
1,2,3	95
1,2,4	85
1,3,4	80
2,3,4	55
1,2	50
1,3	50
1,4	50
2,3	50
2,4	50
3,4	50
```

Bemærk, at der kun er angivet de værdier der er forskellige fra 0, da 0 er angivet som standard. Opstillingen hvor celler er adskilt af linjeskift og tabulatortegn er en standard eksportmetode for de fleste regneark. Man skal dog huske at sætte decimalseparatoren til punktum (.), hvilket ikke er standard på dansk.

Den sidste funktion i klassen **Input** er SetupMat2. Denne funktion sørger for at trække en eventuel værdi for den tomme koalition fra de andre værdier og flytte den fulde koalition nederst, så der bliver plads til en kriteriefunktion øverst når α-værdien skal findes. På bilag 1 ses den endelige opstilling af værdispillet fra G. Owen side 332.

Beregninger bliver udført i klassen **Calc**. Her opstilles først simplex tableauet til hjælpeproblemet, som det vil se ud efter første gennemgang, hvor de kunstige koefficienter er fjernet ved hjælp af funktionen *SimpHlpSetupV,* for værdispil og *SimpHlpSetupC* for omkosningsspil. For værdispillet lægges alle rækkerne sammen for at danne kriteriefunktionen, idet der findes kunstige slack-værdier for alle rækker. For omkostningsspillet dannes kriteriefunktionen, i første gennemløb, kun af , som indeholder den samlede koalition, da det kun er den der indeholder en kunstig slack-værdi.

I funktionen *Simplex* udføres simplex-metoden på det dannede tableau for hjælpeproblemet gennem flere adskilte funktioner:

Funktionen *FindBig* finder den største værdi i kriteriefunktionen, hvilket angiver den søjle, som skal gøres til en ny basissøjle. Hvis denne værdi er større end tolerancen benyttes funktionen *FindPivorElement* til at finde den række hvor forholdet mellem koalitionsværdien i første søjle og værdien i søjlen bestemt ved *FindBig* er mindst og ikke-negativ. Hvis der ikke findes et ikke-negativt forhold udskrives en fejlmeddelelse, da det ikke bør forekomme, og man kan efterfølgende ikke regne med at få et fornuftigt resultat.

Såfremt der ikke forekommer en fejl bliver det næste tableau dannet ved at funktionen *NewTable* omdanner den fundne søjle til en basissøjle ved hjælp af rækkeoperationer. Den største positive værdi i kriteriefunktionen identificeres igen, og de efterfølgende kører igen. Dette gentages indtil der ikke længere findes positive værdier i kriteriefunktionen.

Når simplex for hjælpeproblemet er kørt til ende identificeres de søjler, som indgår i basis med funktionen *FindBasis.* Funktionen finder yderligere frem til hvilke rækker der indeholder søjlernes 1-taller. I tilfælde af, at hjælpeproblemet har løst simplex til ende, finder funktionen *FindNucleolus* den foreløbige løsning på problemet. En løkke finder ud af, om søjlen, der hører til α er i basis, og gemmer den række, hvor 1-tallet findes.

Hvis der findes en basis i søjlen for α sørger funktionen *SimpMainSetup* for at føre den relevante række op i kriteriefunktionen. Hernæst kører funktionen *Simplex* med dette nye tableau, hvor der ikke længere medregnes de kunstige slacks. Funktionen *FindSlacks* identificerer de pladser, for slack-værdierne, i kriteriefunktionen, som er forskellige fra 0, hvilket betyder, at de tilhørende uligheder i ligningssystemet er entydigt optimeret, og derfor kan fjernes fra ligningssystemet. Funktionen *UpdateMain* sørger nu for, at omdanne de uligheder, som er entydigt optimeret til ligninger ved at trække den fundne α-værdi fra værdien for den tilhørende koalition, hvis der er tale om et værdispil, eller lægge α-værdien til, hvis der er tale om et omkostningsspil. Desuden fjernes α i rækken. Hvis der er fremkommet nye ligninger i ligningssystemet skal hjælpeproblemet opstilles igen. Som tidligere bliver dette håndteret af funktionerne *SimpHlpSetupV* og *SimpHlpSetupC*. For at lave de ændringer, som er nødvendige for at indføre de nye ligninger køres funktionen *SimpHlpSetupSlack*. Her fjernes slacks for alle de ligninger, som er blevet ændret, mens der tilføjes kunstige slacks. For værdispil findes disse kunstige slacks i forvejen, og ændrer derfor ikke noget, men for omkostningspil tilføjes de kunstige slacks, hvor der før ikke var nogen. For værdispil er kriteriefunktionen stadig fremkommet ved at lægge alle rækker sammen, og den eneste ændring for

kriteriefunktionen er dens værdi for α samt fjernelse af slack. For Omkostningsspil skal hver række, som er blevet ændret lægges til i kriteriefunktionen. Dens værdier for α og slacks vil forblive 0.
Programmet går tilbage og udfører simplex for hjælpeproblemet, og kører i ring indtil der ikke findes nye ligninger for ligningssystemet.
Præsentationen af resultaterne sker i klassen **Pre**, og består kun af en udskrift af nucleolus gennem funktionen *PreNuc*.

Test

Det har vist sig svært at finde gennemregnede eksempler for nucleolus, men to online værktøjer er benyttet til at gennemføre simplex beregninger, så metoden kan afprøves i flere skridt. Det drejer sig om *The Simplex Java Applet*[9] og *Simplex Method Tool*[10]. *The Simplex Java Applet* udfører simplex beregninger, hvor hvert skridt kan vises. Det er delt op i hjælpeproblemet og hovedproblemet, og idet hvert tableau vises, er det nemt at identificere de uligheder, der har en entydig løsning i henhold til teorien. Dette er klart det bedste værktøj at lave tests med, men det er begrænset til 7 variable og 7 bibetingelser, hvilket betyder, at det kun kan benyttes til kooperative spil for 3 spillere. *Simplex Method Tool* har ikke umiddelbart oplyst nogen begrænsning, men de opstillede tableauer følger ikke den gængse opstilling, og det har ikke været muligt at benytte værktøjet til at begrænse antallet af uligheder i det lineære program.

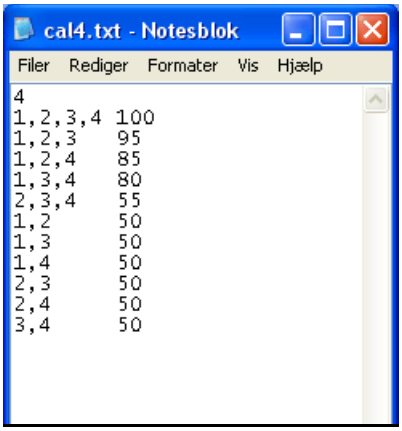
Test 1 – eksempel fra G. Owen side 332.

Spillets opstilling er følgende:

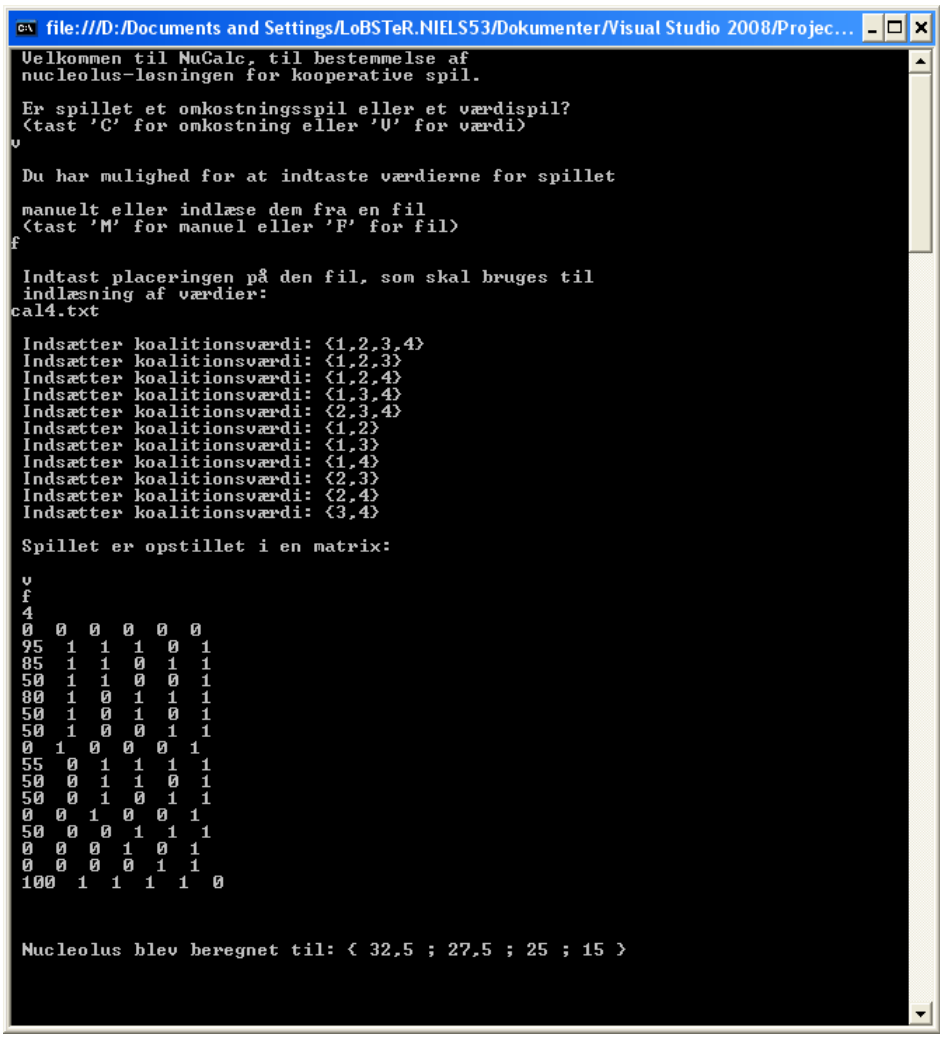
$$N=1,2,3,4 \qquad v(N)=100$$
$$v(\{1,2,3\})=95 \qquad v(\{1,2,4\})=85 \quad v(\{1,3,4\})=80 \qquad v(\{2,3,4\})=55$$
$$v(\{i,j\})=50 \quad \forall\, i,j\in N, i\neq j \qquad v(\{i\})=0 \quad \forall\, i\in N$$

9 http://neos.mcs.anl.gov/CaseStudies/simplex/applet/SimplexTool.html
10 *http://www.zweigmedia.com/RealWorld/simplex.html*

Spillet er opstillet i en tekstfil: til indlæsning i programmet:

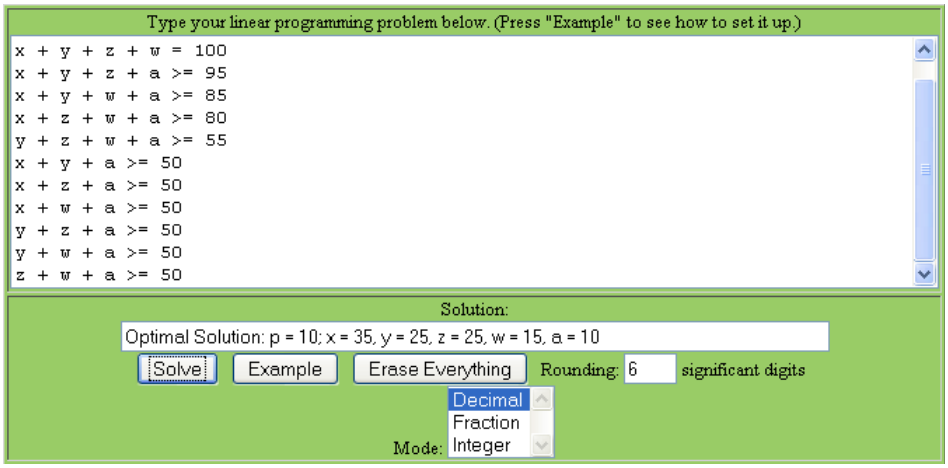


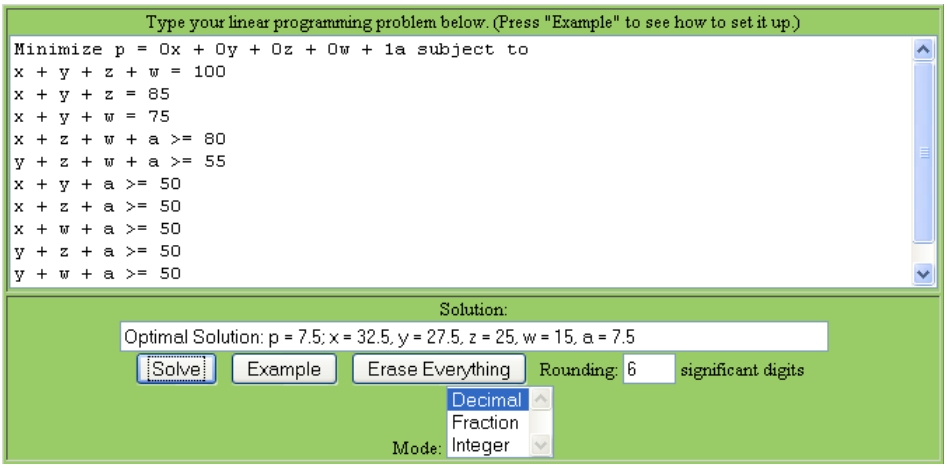
Programmet køres:



Det ses, at programmet kører som forventet, og den fundne løsning på (32,5; 27,5; 25; 15) passer

med den der er fundet hos Owen. Desuden kan simplex tableauerne løses med *Simplex Method Tool.* Som nævnt tidligere kan det ikke umiddelbart aflæses hvilke uligheder skal ændres før næste gennemløb, med de to relevante ligningssystemer, som er angivet hos Owen kan opstilles og løses:





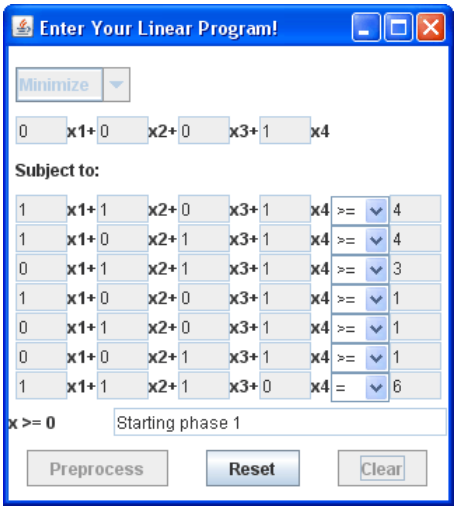
Test 2 – eksemplet med 3 spillere fra tidligere:

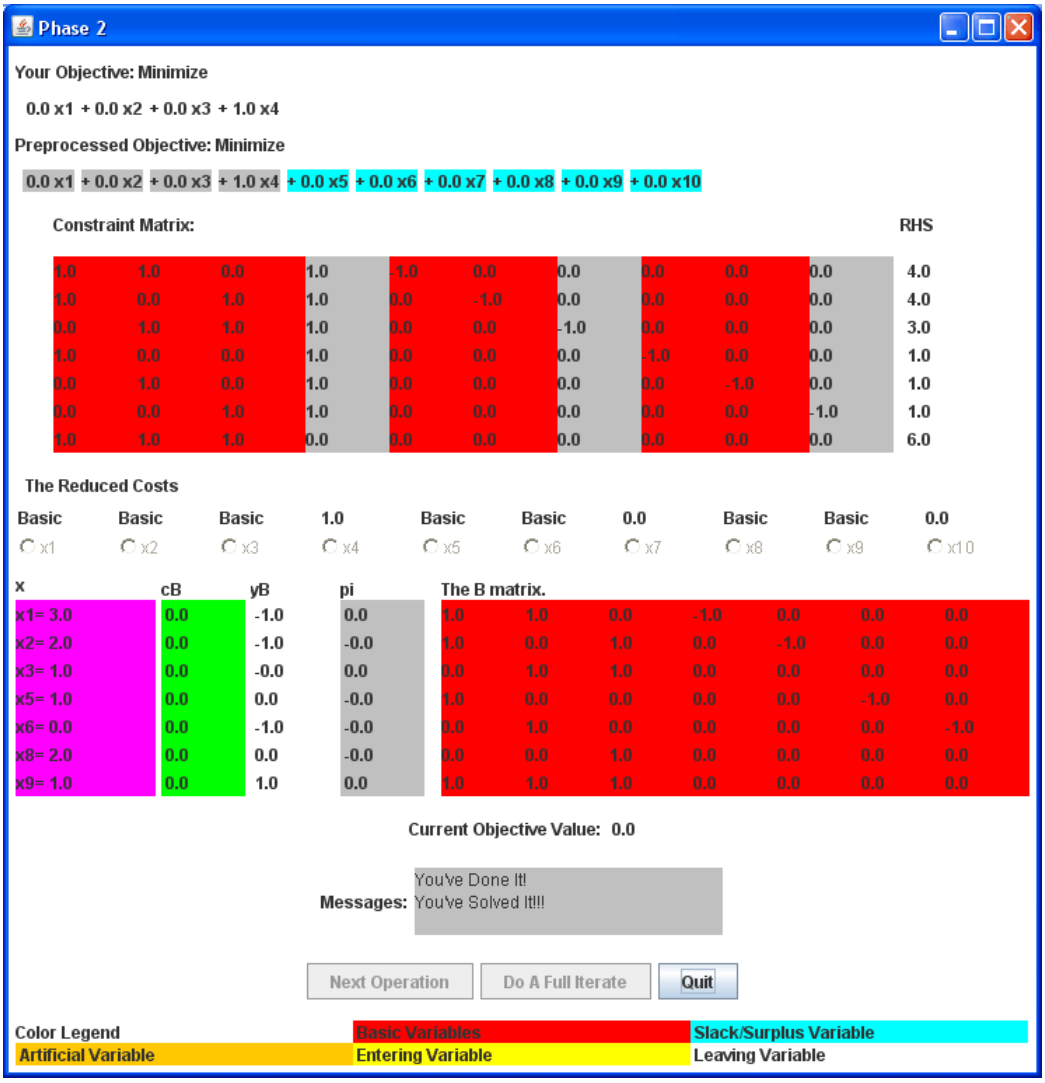
Spillets opstilling er følgende:

$$3\ \textit{spillere}$$
$$N=1,2,3 \qquad v(N)=6 \qquad v(\varnothing)=0$$
$$v(1,2)=v(1,3)=4 \qquad v(2,3)=3$$
$$v(1)=v(2)=v(3)=1$$

Programmet køres, hvor værdierne indtastes manuelt:

Her nås løsningen (3; 2; 1), hvilket også er tilfældet for *The Simplex Java Applet*:





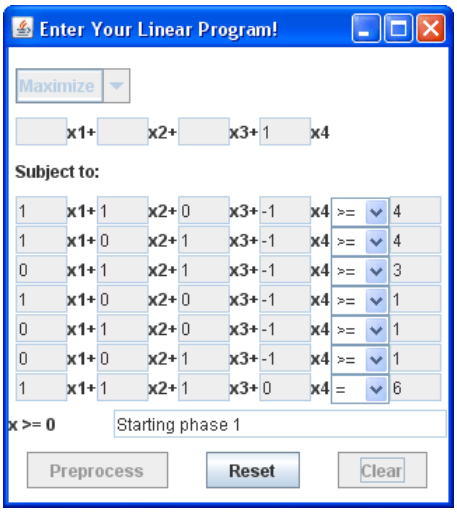
Opbygningen af spillet viser dog tydeligt, at spiller 2 og spiller 3 er symmetriske, og det giver

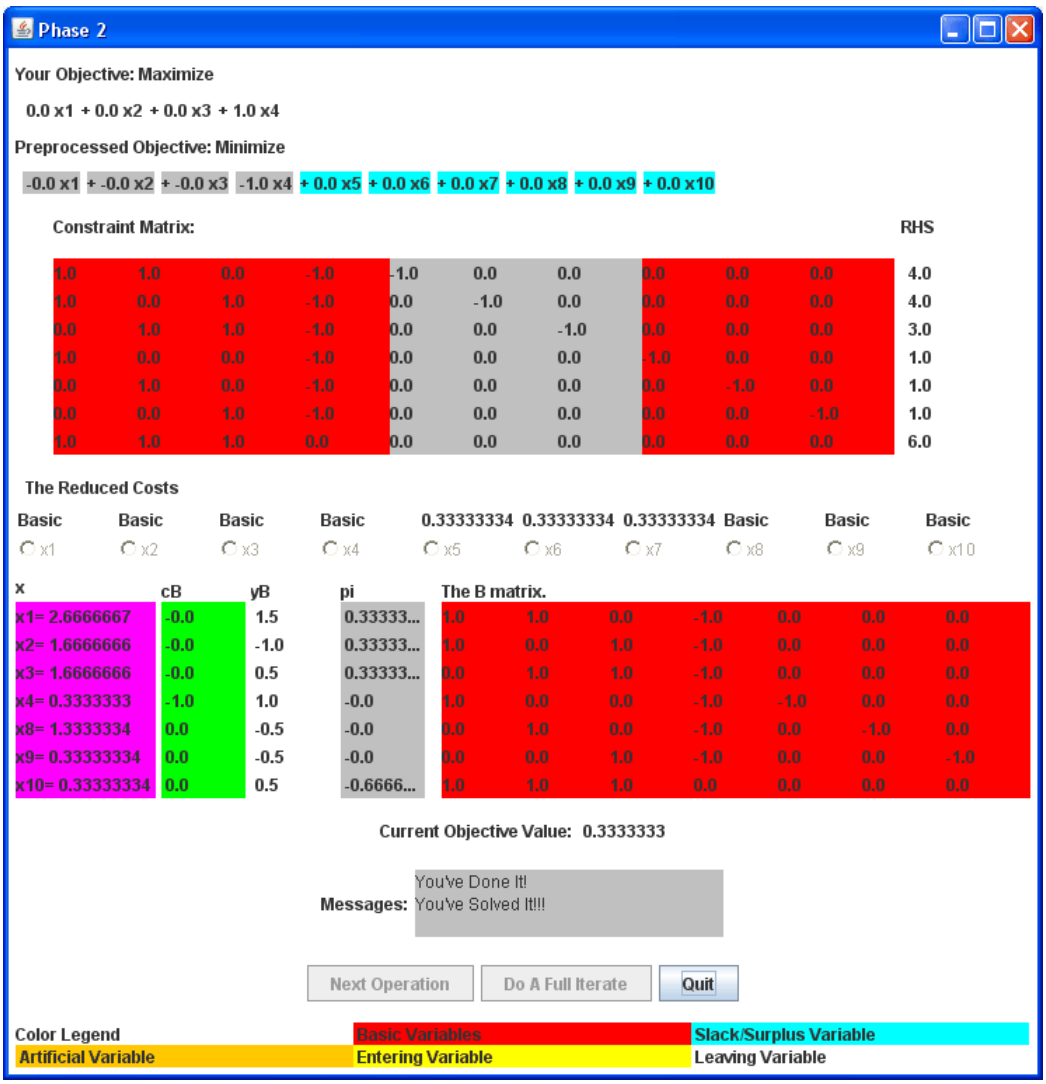
derfor ikke mening, at de får forskellige værdier. Desuden ses det af tegningen i teoriafsnittet, at denne løsning ligger i hjørnet af kernen, hvilket ikke giver god mening.

Delkonklusion

Det er selvfølgelig ikke selvfølgelig ikke særlig brugbart at fremlægge resultater, der ikke stemmer, men det har af tidsmæssige årsager vist sig som den bedste måde at illustrere de problemer projektet er stødt på. Efter megen tid brugt på fejlfinding i programkoden, og revurdering af teorien, har det på et meget fremskredent tidspunkt vist sig, at problemerne stammer fra begrænsninger i simplex-metoden. Værdien for α bliver kun fundet hvis den er positiv. Eksemplet fra Owen giver den rigtige løsning, da en entydig løsning bliver fundet mens der stadig ses på koalitioner der giver mulighed for forbedring. Da løsningen på spillet med 3 spillere findes ved at minimere α til 0, hvilket i ord betyder, at man minimerer den mulighed spillerne har for forbedring, bør man herefter undersøge hvor stor muligheden for forværring er. Dvs. man maksimerer værdien for α, når α er den værdi man kan trække fra.

Test 2 kan nu opstilles i The Simplex Java Applet for at maksimere α:





Det ses nu, at det rigtige minimerede α er -1/3 og at de første 3 uligheder kan fjernes. Det ses umiddelbart, at den nåede løsning er entydigt bestemt, hvilket også kan demonstreres ved at ændre ulighederne til ligninger og køre simplex igen. Nucleolus løsningen er altså (8/3; 5/3; 5/3).

Teori II

For at få programmet til at fungere korrekt, er der altså brug for at give plads til to gennemløb. Hvis problemet opstilles som det ser ud når det er standardiseret til simplex ser det første problem således ud:

$$\begin{aligned} &\textit{maksimer} \quad -\alpha \\ &\textit{under bibetingelserne} \\ &\sum_{i\in S} x_i + \alpha \geqslant v(S), \quad S \subset N \\ &\sum_{i\in N} x_i = v(N) \end{aligned}$$

$$\begin{aligned} &\textit{maksimer} \quad -\alpha \\ &\textit{under bibetingelserne} \\ &\sum_{i\in S} x_i - \alpha \leqslant c(S), \quad S \subset N \\ &\sum_{i\in N} x_i = c(N) \end{aligned}$$

Såfremt der findes en mængde koalitioner A_1, hvor værdien α_1 opfylder ulighederne entydigt fortsættes der i lighed med tidligere ved:

$$\begin{array}{c} \textit{maksimer} \quad -\alpha \\ \\ \textit{under bibetingelserne} \\ \sum_{i\in S} x_i + \alpha \geqslant v(S), \quad S \subset N \\ \sum_{i\in S} x_i = v(S) - \alpha_1, \quad \forall\ S \subset A_1 \\ \sum_{i\in N} x_i = v(N) \end{array} \qquad \begin{array}{c} \textit{maksimer} \quad -\alpha \\ \\ \textit{under bibetingelserne} \\ \sum_{i\in S} x_i - \alpha \leqslant c(S), \quad S \subset N \\ \sum_{i\in S} x_i = c(S) + \alpha_1, \quad \forall\ S \subset A_1 \\ \sum_{i\in N} x_i = c(N) \end{array}$$

Hvis der her fremkommer en løsning, hvor α bliver 0, ses der bort fra denne, og problemet ændres nu til:

$$\begin{array}{c} \textit{maksimer} \quad \alpha \\ \\ \textit{under bibetingelserne} \\ \sum_{i\in S} x_i - \alpha \geqslant v(S), \quad S \subset N \\ \sum_{i\in S} x_i = v(S) - \alpha_1, \quad \forall\ S \subset A_1 \\ \sum_{i\in N} x_i = v(N) \end{array} \qquad \begin{array}{c} \textit{maksimer} \quad \alpha \\ \\ \textit{under bibetingelserne} \\ \sum_{i\in S} x_i + \alpha \leqslant c(S), \quad S \subset N \\ \sum_{i\in S} x_i = c(S) + \alpha_1, \quad \forall\ S \subset A_1 \\ \sum_{i\in N} x_i = c(N) \end{array}$$

Hvis der her findes en mængde koalitioner A2, hvor værdien α_2 opfylder ulighederne entydigt, bliver det næste problem da:

$$\begin{array}{c} \textit{maksimer} \quad \alpha \\ \\ \textit{under bibetingelserne} \\ \sum_{i\in S} x_i - \alpha \geqslant v(S), \quad S \subset N \\ \sum_{i\in S} x_i = v(S) - \alpha_1, \quad \forall\ S \subset A_1 \\ \sum_{i\in S} x_i = v(S) + \alpha_2, \quad \forall\ S \subset A_2 \\ \sum_{i\in N} x_i = v(N) \end{array} \qquad \begin{array}{c} \textit{maksimer} \quad \alpha \\ \\ \textit{under bibetingelserne} \\ \sum_{i\in S} x_i + \alpha \leqslant c(S), \quad S \subset N \\ \sum_{i\in S} x_i = c(S) + \alpha_1, \quad \forall\ S \subset A_1 \\ \sum_{i\in S} x_i = c(S) - \alpha_2, \quad \forall\ S \subset A_2 \\ \sum_{i\in N} x_i = c(N) \end{array}$$

Konklusion

Målet om at have et færdigt program er ikke nået, men metoden, som skal benyttes for at nå dette mål er nu klarlagt. Der er desværre blevet spildt for megen tid på at lede efter fejl det forkerte sted, og der er derfor heller ikke blevet plads til yderligere funktioner, som ville være forholdsvis simple implementeringer. Det kunne have været yderst informativt at vide, om kernen i et spil er tom. Dette kan verificeres med en simpel kontrol for konvekse/konkave spil. Den entydige test er som nævnt tidligere Bondavera-Shapley sætningen, men dette er dog en mere kompliceret test. Det ville være brugbart at have et alternativt løsningskoncept, og her ville Shapley-værdien også være yderst simpel at implementere. Denne løsning findes ved at gennemløbe alle de mulige rækkefølger af spillere, og for hver spiller findes den gennemsnitlige forbedring, som spilleren medbringer til koalitionen af spillere der står før. Et fuldt funktionsdygtigt program med de førnævnte funktioner kan efter alt at dømme udarbejdes inden for kort tid selvom det ikke er færdiggjort til afleveringen af denne opgave. Alt i alt har udarbejdningen af opgaven været lærerig, men resultatmæssigt en fattig oplevelse.

v(S)	x_1	x_2	x_3	x_4	α	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}	s_{11}	s_{12}	s_{13}	s_{14}	as_1	as_2	as_3	as_4	as_5	as_6	as_7	as_8	as_9	as_{10}	as_{11}	as_{12}	as_{13}	as_{14}	as_{15}
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
95	1	1	1	0	1	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
85	1	1	0	1	1	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
80	1	0	1	1	1	0	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
55	0	1	1	1	1	0	0	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
50	1	1	0	0	1	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
50	1	0	1	0	1	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
50	1	0	0	1	1	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
50	0	1	1	0	1	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
50	0	1	0	1	1	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
50	0	0	1	1	1	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
100	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Kilder:

Guillermo Owen – Game Theory, 3rd edition, Academic Press, 1995

Jens Leth Hougaard – An Introduction To Allocation Rules, Springer, 2009

www.phpsimplex.com/en/index.htm

neos.mcs.anl.gov/CaseStudies/simplex/applet/SimplexTool.html

www.zweigmedia.com/RealWorld/simplex.html