

# Dynamisk programmering

Betrakta ett lagerhållningsproblem i flera tidsperioder.

# Dynamisk programmering

Betrakta ett lagerhållningsproblem i flera tidsperioder.

Vi har tillverkning och försäljning av produkter i varje tidsperiod.

# Dynamisk programmering

Betrakta ett lagerhållningsproblem i flera tidsperioder.

Vi har tillverkning och försäljning av produkter i varje tidsperiod.

Dessutom kan vi lagra produkter mellan tidsperioder, för att utnyttja stordriftsfördelar i produktionen.

# Dynamisk programmering

Betrakta ett lagerhållningsproblem i flera tidsperioder.

Vi har tillverkning och försäljning av produkter i varje tidsperiod.

Dessutom kan vi lagra produkter mellan tidsperioder, för att utnyttja stordriftsfördelar i produktionen.

Antag att försäljningen är given i varje period.

# Dynamisk programmering

Betrakta ett lagerhållningsproblem i flera tidsperioder.

Vi har tillverkning och försäljning av produkter i varje tidsperiod.

Dessutom kan vi lagra produkter mellan tidsperioder, för att utnyttja stordriftsfördelar i produktionen.

Antag att försäljningen är given i varje period.

Variabler blir då hur mycket vi ska tillverka varje period,

# Dynamisk programmering

Betrakta ett lagerhållningsproblem i flera tidsperioder.

Vi har tillverkning och försäljning av produkter i varje tidsperiod.

Dessutom kan vi lagra produkter mellan tidsperioder, för att utnyttja stordriftsfördelar i produktionen.

Antag att försäljningen är given i varje period.

Variabler blir då hur mycket vi ska tillverka varje period, och (som följd av det) hur mycket vi ska lagerhålla mellan perioderna.

# Dynamisk programmering

Beteckningar:  $d_k$  är försäljningen i period  $k$ .

# Dynamisk programmering

Beteckningar:  $d_k$  är försäljningen i period  $k$ .

Variabler:  $x_k$  är antal tillverkade produkter i period  $k$ .



# Dynamisk programmering

Beteckningar:  $d_k$  är försäljningen i period  $k$ .

Variabler:  $x_k$  är antal tillverkade produkter i period  $k$ .

$s_k$  är antal enheter som lagerhålls från tidsperiod  $k$  till tidsperiod  $k + 1$ .

# Dynamisk programmering

Beteckningar:  $d_k$  är försäljningen i period  $k$ .

Variabler:  $x_k$  är antal tillverkade produkter i period  $k$ .

$s_k$  är antal enheter som lagerhålls från tidsperiod  $k$  till tidsperiod  $k + 1$ .

Vi får bivillkor som säger att:

# Dynamisk programmering

Beteckningar:  $d_k$  är försäljningen i period  $k$ .

Variabler:  $x_k$  är antal tillverkade produkter i period  $k$ .

$s_k$  är antal enheter som lagerhålls från tidsperiod  $k$  till tidsperiod  $k + 1$ .

Vi får bivillkor som säger att:

ingående lager + tillverkad mängd = försäljning + utgående lager:

# Dynamisk programmering

Beteckningar:  $d_k$  är försäljningen i period  $k$ .

Variabler:  $x_k$  är antal tillverkade produkter i period  $k$ .

$s_k$  är antal enheter som lagerhålls från tidsperiod  $k$  till tidsperiod  $k + 1$ .

Vi får bivillkor som säger att:

ingående lager + tillverkad mängd = försäljning + utgående lager:

$$s_{k-1} + x_k = d_k + s_k \quad \text{för alla } k$$

# Dynamisk programmering

Beteckningar:  $d_k$  är försäljningen i period  $k$ .

Variabler:  $x_k$  är antal tillverkade produkter i period  $k$ .

$s_k$  är antal enheter som lagerhålls från tidsperiod  $k$  till tidsperiod  $k + 1$ .

Vi får bivillkor som säger att:

ingående lager + tillverkad mängd = försäljning + utgående lager:

$$s_{k-1} + x_k = d_k + s_k \quad \text{för alla } k$$

Om man har många tidsperioder, kan det bli svårt att lösa hela problemet på en gång.

# Dynamisk programmering

Utnyttja tidsstrukturen: Tiden går bara framåt.

# Dynamisk programmering

Utnyttja tidsstrukturen: Tiden går bara framåt.

Betrakta en viss tidsperiod: Optimeringen handlar om hur mycket vi ska tillverka den perioden.

# Dynamisk programmering

Utnyttja tidsstrukturen: Tiden går bara framåt.

Betrakta en viss tidsperiod: Optimeringen handlar om hur mycket vi ska tillverka den perioden.

Om man inte tillåter något lager, har vi ett lätt optimeringsproblem för varje tidsperiod.



# Dynamisk programmering

Utnyttja tidsstrukturen: Tiden går bara framåt.

Betrakta en viss tidsperiod: Optimeringen handlar om hur mycket vi ska tillverka den perioden.

Om man inte tillåter något lager, har vi ett lätt optimeringsproblem för varje tidsperiod.

Likaså, om lagernivåerna är givna, har vi ett lätt optimeringsproblem för varje tidsperiod.

# Dynamisk programmering

Utnyttja tidsstrukturen: Tiden går bara framåt.

Betrakta en viss tidsperiod: Optimeringen handlar om hur mycket vi ska tillverka den perioden.

Om man inte tillåter något lager, har vi ett lätt optimeringsproblem för varje tidsperiod.

Likaså, om lagernivåerna är givna, har vi ett lätt optimeringsproblem för varje tidsperiod.

Slutsats: Lös för alla värden på lagernivåerna.

# Dynamisk programmering

Antag att  $s_0 = 0$ , dvs. att vi inte har något i lager innan första tidsperioden.

# Dynamisk programmering

Antag att  $s_0 = 0$ , dvs. att vi inte har något i lager innan första tidsperioden. Då kan vi lösa optimeringsproblemet i period 1 för alla möjliga värden på  $s_1$ , dvs. beroende på önskat lager efter period 1.

# Dynamisk programmering

Antag att  $s_0 = 0$ , dvs. att vi inte har något i lager innan första tidsperioden.

Då kan vi lösa optimeringsproblemet i period 1 för alla möjliga värden på  $s_1$ , dvs. beroende på önskat lager efter period 1.

Alltså:

# Dynamisk programmering

Antag att  $s_0 = 0$ , dvs. att vi inte har något i lager innan första tidsperioden.

Då kan vi lösa optimeringsproblemet i period 1 för alla möjliga värden på  $s_1$ , dvs. beroende på önskat lager efter period 1.

Alltså:

Finn optimal produktion om  $s_1 = 0$ .

# Dynamisk programmering

Antag att  $s_0 = 0$ , dvs. att vi inte har något i lager innan första tidsperioden.

Då kan vi lösa optimeringsproblemet i period 1 för alla möjliga värden på  $s_1$ , dvs. beroende på önskat lager efter period 1.

Alltså:

Finn optimal produktion om  $s_1 = 0$ .

Finn optimal produktion om  $s_1 = 1$ .

# Dynamisk programmering

Antag att  $s_0 = 0$ , dvs. att vi inte har något i lager innan första tidsperioden. Då kan vi lösa optimeringsproblemet i period 1 för alla möjliga värden på  $s_1$ , dvs. beroende på önskat lager efter period 1.

Alltså:

Finn optimal produktion om  $s_1 = 0$ .

Finn optimal produktion om  $s_1 = 1$ .

Finn optimal produktion om  $s_1 = 2$ .



# Dynamisk programmering

Antag att  $s_0 = 0$ , dvs. att vi inte har något i lager innan första tidsperioden. Då kan vi lösa optimeringsproblemet i period 1 för alla möjliga värden på  $s_1$ , dvs. beroende på önskat lager efter period 1.

Alltså:

Finn optimal produktion om  $s_1 = 0$ .

Finn optimal produktion om  $s_1 = 1$ .

Finn optimal produktion om  $s_1 = 2$ .

Finn optimal produktion om  $s_1 = 3$ .

# Dynamisk programmering

Antag att  $s_0 = 0$ , dvs. att vi inte har något i lager innan första tidsperioden.

Då kan vi lösa optimeringsproblemet i period 1 för alla möjliga värden på  $s_1$ , dvs. beroende på önskat lager efter period 1.

Alltså:

Finn optimal produktion om  $s_1 = 0$ .

Finn optimal produktion om  $s_1 = 1$ .

Finn optimal produktion om  $s_1 = 2$ .

Finn optimal produktion om  $s_1 = 3$ .

osv.

# Dynamisk programmering

Antag att  $s_0 = 0$ , dvs. att vi inte har något i lager innan första tidsperioden. Då kan vi lösa optimeringsproblemet i period 1 för alla möjliga värden på  $s_1$ , dvs. beroende på önskat lager efter period 1.

Alltså:

Finn optimal produktion om  $s_1 = 0$ .

Finn optimal produktion om  $s_1 = 1$ .

Finn optimal produktion om  $s_1 = 2$ .

Finn optimal produktion om  $s_1 = 3$ .

osv.

upp till  $s_1 =$  maximalt lager.

# Dynamisk programmering

Antag att  $s_0 = 0$ , dvs. att vi inte har något i lager innan första tidsperioden. Då kan vi lösa optimeringsproblemet i period 1 för alla möjliga värden på  $s_1$ , dvs. beroende på önskat lager efter period 1.

Alltså:

Finn optimal produktion om  $s_1 = 0$ .

Finn optimal produktion om  $s_1 = 1$ .

Finn optimal produktion om  $s_1 = 2$ .

Finn optimal produktion om  $s_1 = 3$ .

osv.

upp till  $s_1 =$  maximalt lager.

Detta ger en optimal kostnad för varje lagernivå.

# Dynamisk programmering

I period 2 har vi inget givet värde på  $s_1$ .

# Dynamisk programmering

I period 2 har vi inget givet värde på  $s_1$ .

Det kan då finnas flera sätt att uppnå en viss lagernivå  $s_2$ .

# Dynamisk programmering

I period 2 har vi inget givet värde på  $s_1$ .

Det kan då finnas flera sätt att uppnå en viss lagernivå  $s_2$ .

Välj det billigaste av dessa.

# Dynamisk programmering

I period 2 har vi inget givet värde på  $s_1$ .

Det kan då finnas flera sätt att uppnå en viss lagernivå  $s_2$ .

Välj det billigaste av dessa.

Kostnaden blir tillverkningskostnaden aktuell period



# Dynamisk programmering

I period 2 har vi inget givet värde på  $s_1$ .

Det kan då finnas flera sätt att uppnå en viss lagernivå  $s_2$ .

Välj det billigaste av dessa.

Kostnaden blir tillverkningskostnaden aktuell period plus kostnaden för tidigare perioder.

# Dynamisk programmering

I period 2 har vi inget givet värde på  $s_1$ .

Det kan då finnas flera sätt att uppnå en viss lagernivå  $s_2$ .

Välj det billigaste av dessa.

Kostnaden blir tillverkningskostnaden aktuell period plus kostnaden för tidigare perioder.

Man behöver inte veta exakt hur kostnaden för föregående perioder uppstod.

# Dynamisk programmering

Nya beteckningar:

# Dynamisk programmering

Nya beteckningar:

Låt  $s_k$ , lagernivån efter period  $k$ , kallas "tillstånd".

# Dynamisk programmering

Nya beteckningar:

Låt  $s_k$ , lagernivån efter period  $k$ , kallas "tillstånd".

Vi ska bestämma optimal "styrning" ( $x_k$ ) för varje tillstånd.

# Dynamisk programmering

Nya beteckningar:

Låt  $s_k$ , lagernivån efter period  $k$ , kallas "tillstånd".

Vi ska bestämma optimal "styrning" ( $x_k$ ) för varje tillstånd.

Låt  $f_k(s_k)$  vara kostnaden för att uppnå tillstånd  $s_k$  i tidsperiod  $k$ .

# Dynamisk programmering

Nya beteckningar:

Låt  $s_k$ , lagernivån efter period  $k$ , kallas "tillstånd".

Vi ska bestämma optimal "styrning" ( $x_k$ ) för varje tillstånd.

Låt  $f_k(s_k)$  vara kostnaden för att uppnå tillstånd  $s_k$  i tidsperiod  $k$ .

Låt  $c_k(x_k, s_k)$  vara kostnaden för att tillverka  $x_k$  enheter i period  $k$  och att lagerhålla  $s_k$  enheter till nästa period.

# Dynamisk programmering

Nya beteckningar:

Låt  $s_k$ , lagernivån efter period  $k$ , kallas "tillstånd".

Vi ska bestämma optimal "styrning" ( $x_k$ ) för varje tillstånd.

Låt  $f_k(s_k)$  vara kostnaden för att uppnå tillstånd  $s_k$  i tidsperiod  $k$ .

Låt  $c_k(x_k, s_k)$  vara kostnaden för att tillverka  $x_k$  enheter i period  $k$  och att lagrhålla  $s_k$  enheter till nästa period.

När vi ska optimera i steg  $k$ , vill vi alltså lösa



# Dynamisk programmering

Nya beteckningar:

Låt  $s_k$ , lagernivån efter period  $k$ , kallas "tillstånd".

Vi ska bestämma optimal "styrning" ( $x_k$ ) för varje tillstånd.

Låt  $f_k(s_k)$  vara kostnaden för att uppnå tillstånd  $s_k$  i tidsperiod  $k$ .

Låt  $c_k(x_k, s_k)$  vara kostnaden för att tillverka  $x_k$  enheter i period  $k$  och att lagrhålla  $s_k$  enheter till nästa period.

När vi ska optimera i steg  $k$ , vill vi alltså lösa

$$f_k(s_k) = \min_{x_k} (c_k(x_k, s_k) + f_{k-1}(s_{k-1}))$$

# Dynamisk programmering

Nya beteckningar:

Låt  $s_k$ , lagernivån efter period  $k$ , kallas "tillstånd".

Vi ska bestämma optimal "styrning" ( $x_k$ ) för varje tillstånd.

Låt  $f_k(s_k)$  vara kostnaden för att uppnå tillstånd  $s_k$  i tidsperiod  $k$ .

Låt  $c_k(x_k, s_k)$  vara kostnaden för att tillverka  $x_k$  enheter i period  $k$  och att lagerhålla  $s_k$  enheter till nästa period.

När vi ska optimera i steg  $k$ , vill vi alltså lösa

$$f_k(s_k) = \min_{x_k} (c_k(x_k, s_k) + f_{k-1}(s_{k-1}))$$

där  $s_{k-1} = s_k - x_k + d_k$  (ty  $s_k = s_{k-1} + x_k - d_k$ ).

# Dynamisk programmering

Nya beteckningar:

Låt  $s_k$ , lagernivån efter period  $k$ , kallas "tillstånd".

Vi ska bestämma optimal "styrning" ( $x_k$ ) för varje tillstånd.

Låt  $f_k(s_k)$  vara kostnaden för att uppnå tillstånd  $s_k$  i tidsperiod  $k$ .

Låt  $c_k(x_k, s_k)$  vara kostnaden för att tillverka  $x_k$  enheter i period  $k$  och att lagerhålla  $s_k$  enheter till nästa period.

När vi ska optimera i steg  $k$ , vill vi alltså lösa

$$f_k(s_k) = \min_{x_k} (c_k(x_k, s_k) + f_{k-1}(s_{k-1}))$$

där  $s_{k-1} = s_k - x_k + d_k$  (ty  $s_k = s_{k-1} + x_k - d_k$ ). (Så  $s_{k-1}$  beror på  $x_k$ .)

# Dynamisk programmering

Nya beteckningar:

Låt  $s_k$ , lagernivån efter period  $k$ , kallas "tillstånd".

Vi ska bestämma optimal "styrning" ( $x_k$ ) för varje tillstånd.

Låt  $f_k(s_k)$  vara kostnaden för att uppnå tillstånd  $s_k$  i tidsperiod  $k$ .

Låt  $c_k(x_k, s_k)$  vara kostnaden för att tillverka  $x_k$  enheter i period  $k$  och att lagrhålla  $s_k$  enheter till nästa period.

När vi ska optimera i steg  $k$ , vill vi alltså lösa

$$f_k(s_k) = \min_{x_k} (c_k(x_k, s_k) + f_{k-1}(s_{k-1}))$$

där  $s_{k-1} = s_k - x_k + d_k$  (ty  $s_k = s_{k-1} + x_k - d_k$ ). (Så  $s_{k-1}$  beror på  $x_k$ .)

Vi fixerar alltså  $s_k$  och beräknar det bästa värdet på  $x_k$ .

# Dynamisk programmering

Nya beteckningar:

Låt  $s_k$ , lagernivån efter period  $k$ , kallas "tillstånd".

Vi ska bestämma optimal "styrning" ( $x_k$ ) för varje tillstånd.

Låt  $f_k(s_k)$  vara kostnaden för att uppnå tillstånd  $s_k$  i tidsperiod  $k$ .

Låt  $c_k(x_k, s_k)$  vara kostnaden för att tillverka  $x_k$  enheter i period  $k$  och att lagerhålla  $s_k$  enheter till nästa period.

När vi ska optimera i steg  $k$ , vill vi alltså lösa

$$f_k(s_k) = \min_{x_k} (c_k(x_k, s_k) + f_{k-1}(s_{k-1}))$$

där  $s_{k-1} = s_k - x_k + d_k$  (ty  $s_k = s_{k-1} + x_k - d_k$ ). (Så  $s_{k-1}$  beror på  $x_k$ .)

Vi fixerar alltså  $s_k$  och beräknar det bästa värdet på  $x_k$ .

Detta görs för varje möjligt värde på  $s_k$ .

# Dynamisk programmering

För varje tidsperiod  $k$  görs följande:

# Dynamisk programmering

För varje tidsperiod  $k$  görs följande:

För varje möjligt värde på  $s_k$  beräknas  $f_k(s_k)$ , vilket är det billigaste sättet att uppnå tillstånd  $s_k$ .

# Dynamisk programmering

För varje tidsperiod  $k$  görs följande:

För varje möjligt värde på  $s_k$  beräknas  $f_k(s_k)$ , vilket är det billigaste sättet att uppnå tillstånd  $s_k$ .

Detta ger ett optimalt värde på  $x_k$  för varje värde på  $s_k$ .



# Dynamisk programmering

För varje tidsperiod  $k$  görs följande:

För varje möjligt värde på  $s_k$  beräknas  $f_k(s_k)$ , vilket är det billigaste sättet att uppnå tillstånd  $s_k$ .

Detta ger ett optimalt värde på  $x_k$  för varje värde på  $s_k$ .

Kopplingen till föregående tidsperioder sker via "överföringsfunktionen":

$$s_{k-1} = s_k - x_k + d_k.$$

# Dynamisk programmering

För varje tidsperiod  $k$  görs följande:

För varje möjligt värde på  $s_k$  beräknas  $f_k(s_k)$ , vilket är det billigaste sättet att uppnå tillstånd  $s_k$ .

Detta ger ett optimalt värde på  $x_k$  för varje värde på  $s_k$ .

Kopplingen till föregående tidsperioder sker via "överföringsfunktionen":

$$s_{k-1} = s_k - x_k + d_k.$$

När man är färdig, nystas optimal lösning upp bakifrån.

# Dynamisk programmering

För varje tidsperiod  $k$  görs följande:

För varje möjligt värde på  $s_k$  beräknas  $f_k(s_k)$ , vilket är det billigaste sättet att uppnå tillstånd  $s_k$ .

Detta ger ett optimalt värde på  $x_k$  för varje värde på  $s_k$ .

Kopplingen till föregående tidsperioder sker via "överföringsfunktionen":

$$s_{k-1} = s_k - x_k + d_k.$$

När man är färdig, nystas optimal lösning upp bakifrån.

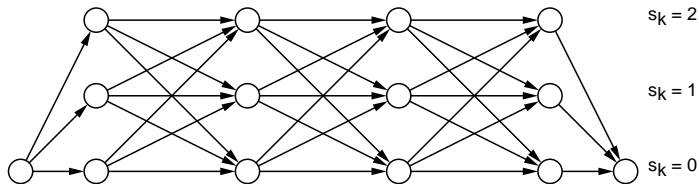
Det får bara finnas ett tillåtet sluttillstånd.

# Dynamisk programmering

Man kan rita upp det som en nivåindelad acyklisk graf.

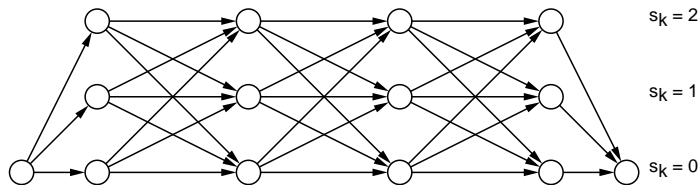
# Dynamisk programmering

Man kan rita upp det som en nivåindelad acyklisk graf.



# Dynamisk programmering

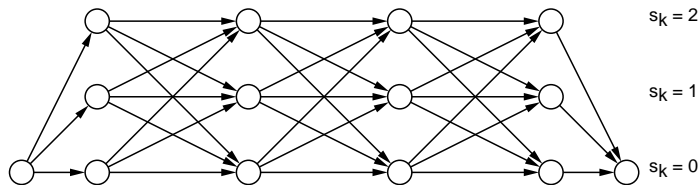
Man kan rita upp det som en nivåindelad acyklisk graf.



Varje nivå motsvarar starten på en tidsperiod.

# Dynamisk programmering

Man kan rita upp det som en nivåindelad acyklisk graf.

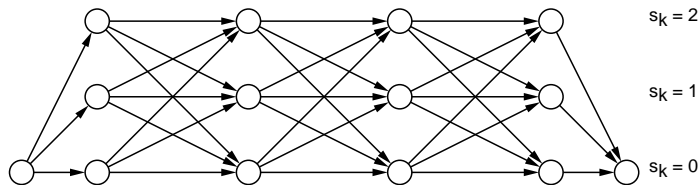


Varje nivå motsvarar starten på en tidsperiod.

I varje nivå ges varje tillstånd av en nod.

# Dynamisk programmering

Man kan rita upp det som en nivåindelad acyklisk graf.



Varje nivå motsvarar starten på en tidsperiod.

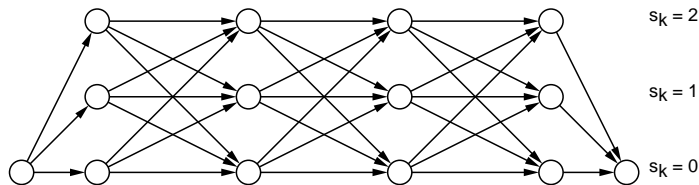
I varje nivå ges varje tillstånd av en nod.

Bågarna går alla framåt i tiden, och avspeglar möjliga förändringar i lagernivån.



# Dynamisk programmering

Man kan rita upp det som en nivåindelad acyklisk graf.



Varje nivå motsvarar starten på en tidsperiod.

I varje nivå ges varje tillstånd av en nod.

Bågarna går alla framåt i tiden, och avspeglar möjliga förändringar i lagernivån.

Optimallösningen är helt enkelt en väg genom detta nätverk.

# Dynamisk programmering

Detta angreppssätt kan användas på allt som kan beskrivas med ett acykliskt nivåindelad nätverk.

# Dynamisk programmering

Detta angreppssätt kan användas på allt som kan beskrivas med ett acykliskt nivåindelad nätverk.

Vi kallar nivåerna *steg*, de olika noderna i varje steg *tillstånd* och optimala variabelvärdet *styrning*.

# Dynamisk programmering

Detta angreppssätt kan användas på allt som kan beskrivas med ett acykliskt nivåindelad nätverk.

Vi kallar nivåerna *steg*, de olika noderna i varje steg *tillstånd* och optimala variabelvärdet *styrning*.

Man gör följande:

# Dynamisk programmering

Detta angreppssätt kan användas på allt som kan beskrivas med ett acykliskt nivåindelad nätverk.

Vi kallar nivåerna *steg*, de olika noderna i varje steg *tillstånd* och optimala variabelvärdet *styrning*.

Man gör följande:

- Dela upp problemet i steg.

# Dynamisk programmering

Detta angreppssätt kan användas på allt som kan beskrivas med ett acykliskt nivåindelad nätverk.

Vi kallar nivåerna *steg*, de olika noderna i varje steg *tillstånd* och optimala variabelvärdet *styrning*.

Man gör följande:

- Dela upp problemet i steg.
- Ta ett steg i taget. Bestäm bästa sättet (styrningen) att uppnå varje tillstånd i steget. Notera den optimala styrningen och den optimala kostnaden för att uppnå varje tillstånd.

# Dynamisk programmering

Detta angreppssätt kan användas på allt som kan beskrivas med ett acykliskt nivåindelad nätverk.

Vi kallar nivåerna *steg*, de olika noderna i varje steg *tillstånd* och optimala variabelvärdet *styrning*.

Man gör följande:

- Dela upp problemet i steg.
- Ta ett steg i taget. Bestäm bästa sättet (styrningen) att uppnå varje tillstånd i steget. Notera den optimala styrningen och den optimala kostnaden för att uppnå varje tillstånd.
- Gå till nästa steg.

# Dynamisk programmering

Detta angreppssätt kan användas på allt som kan beskrivas med ett acykliskt nivåindelad nätverk.

Vi kallar nivåerna *steg*, de olika noderna i varje steg *tillstånd* och optimala variabelvärdet *styrning*.

Man gör följande:

- Dela upp problemet i steg.
- Ta ett steg i taget. Bestäm bästa sättet (styrningen) att uppnå varje tillstånd i steget. Notera den optimala styrningen och den optimala kostnaden för att uppnå varje tillstånd.
- Gå till nästa steg.
- När sista steget nåtts, nysta upp lösningen m.h.a. de noterade styrningarna.



# Dynamisk programmering

För att metoden skall fungera krävs följande.

# Dynamisk programmering

För att metoden skall fungera krävs följande.

## Definition (Optimalitetsprincipen i dynamisk programmering)

*Valet av efterföljande styrningar från ett visst tillstånd får inte bero på hur tillståndet uppnåtts, utan bara på kostnaden att uppnå det.*

# Dynamisk programmering

För att metoden skall fungera krävs följande.

## Definition (Optimalitetsprincipen i dynamisk programmering)

*Valet av efterföljande styrningar från ett visst tillstånd får inte bero på hur tillståndet uppnåtts, utan bara på kostnaden att uppnå det.*

Dynamisk programmering är ett sätt att göra implicit uppräknig av alla möjliga lösningar.

# Dynamisk programmering

För att metoden skall fungera krävs följande.

## Definition (Optimalitetsprincipen i dynamisk programmering)

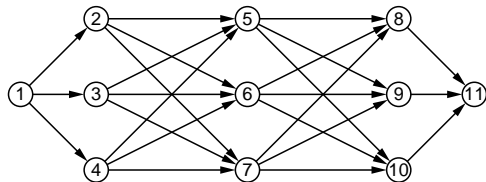
*Valet av efterföljande styrningar från ett visst tillstånd får inte bero på hur tillståndet uppnåtts, utan bara på kostnaden att uppnå det.*

Dynamisk programmering är ett sätt att göra implicit uppräknig av alla möjliga lösningar.

Man undviker att räkna upp alla möjliga vägar från startnod till slutnod, genom att utnyttja nodpriserna för noderna i det föregående steget.

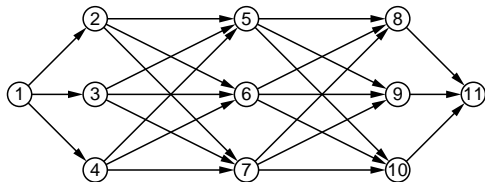
# Dynamisk programmering

Billigaste väg-problem i acyklisk nivåindelad graf.



# Dynamisk programmering

Billigaste väg-problem i acyklisk nivåindelad graf.

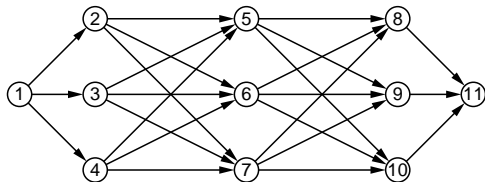


Nodmärkningsmetod: Ta en nivå i taget:

- $y_1 = 0$ .
- För  $k = 1, \dots, N$ :  
För varje  $j \in S_k$ : Sätt  $y_j = c_{ij} + y_i = \min_i(c_{ij} + y_i)$  och  $p_j = i$ .

# Dynamisk programmering

Billigaste väg-problem i acyklisk nivåindelad graf.



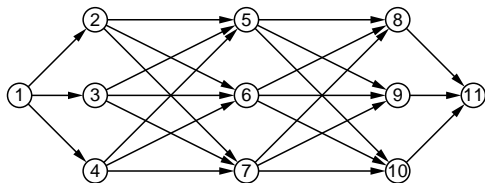
Nodmärkningsmetod: Ta en nivå i taget:

- $y_1 = 0$ .
- För  $k = 1, \dots, N$ :  
För varje  $j \in S_k$ : Sätt  $y_j = c_{ij} + y_i = \min_i(c_{ij} + y_i)$  och  $p_j = i$ .

Ordningen inom ett steg oviktig.

# Dynamisk programmering

Billigaste väg-problem i acyklisk nivåindelad graf.



Nodmärkningsmetod: Ta en nivå i taget:

- $y_1 = 0$ .
- För  $k = 1, \dots, N$ :  
För varje  $j \in S_k$ : Sätt  $y_j = c_{1j} + y_1 = \min_i (c_{ij} + y_i)$  och  $p_j = l$ .

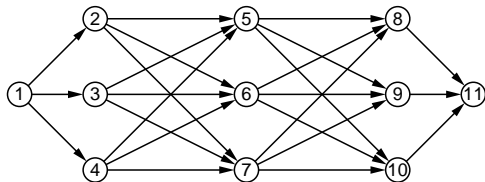
Ordningen inom ett steg oviktig.

Vägen kommer att passera *en* av noderna i varje steg. Vet ej vilken.



# Dynamisk programmering

Billigaste väg-problem i acyklisk nivåindelad graf.



Nodmärkningsmetod: Ta en nivå i taget:

- $y_1 = 0$ .
- För  $k = 1, \dots, N$ :  
För varje  $j \in S_k$ : Sätt  $y_j = c_{ij} + y_i = \min_i (c_{ij} + y_i)$  och  $p_j = i$ .

Ordningen inom ett steg oviktig.

Vägen kommer att passera *en* av noderna i varje steg. Vet ej vilken.

Möjliga målfunktioner:  $\min \sum$ ,  $\max \sum$ ,  $\min \prod$ ,  $\max \prod$ ,  $\min \max$ ,  $\max \min$ .

# Dynamisk programmering: Begrepp

I steg  $k$ :

# Dynamisk programmering: Begrepp

I steg  $k$ :

Nod  $j$  i  $S_k$  kallas **tillstånd**,  $s_k$ .

# Dynamisk programmering: Begrepp

I steg  $k$ :

Nod  $j$  i  $S_k$  kallas **tillstånd**,  $s_k$ .

Noden man kommer från,  $l$ , kallas **styrning**,  $x_k$ .

# Dynamisk programmering: Begrepp

I steg  $k$ :

Nod  $j$  i  $S_k$  kallas **tillstånd**,  $s_k$ .

Noden man kommer från,  $l$ , kallas **styrning**,  $x_k$ .

**Överföringsfunktion**: kopplingen mellan olika steg,  $s_{k-1} = T_k(x_k, s_k)$ .  
(föregångare)

# Dynamisk programmering: Begrepp

I steg  $k$ :

Nod  $j$  i  $S_k$  kallas **tillstånd**,  $s_k$ .

Noden man kommer från,  $l$ , kallas **styrning**,  $x_k$ .

**Överföringsfunktion**: kopplingen mellan olika steg,  $s_{k-1} = T_k(x_k, s_k)$ .  
(föregångare)

**Målfunktionsvärde**,  $f_k(s_k)$ . (nodpris)

# Dynamisk programmering: Begrepp

I steg  $k$ :

Nod  $j$  i  $S_k$  kallas **tillstånd**,  $s_k$ .

Noden man kommer från,  $l$ , kallas **styrning**,  $x_k$ .

**Överföringsfunktion**: kopplingen mellan olika steg,  $s_{k-1} = T_k(x_k, s_k)$ .  
(föregångare)

**Målfunktionsvärde**,  $f_k(s_k)$ . (nodpris)

**Rekursiv målfunktion**:  $f_k(s_k) = \min_{x_k}(c_k(x_k, s_k) + f_{k-1}(s_{k-1}))$

# Dynamisk programmering: Begrepp

I steg  $k$ :

Nod  $j$  i  $S_k$  kallas **tillstånd**,  $s_k$ .

Noden man kommer från,  $l$ , kallas **styrning**,  $x_k$ .

**Överföringsfunktion**: kopplingen mellan olika steg,  $s_{k-1} = T_k(x_k, s_k)$ .  
(föregångare)

**Målfunktionsvärde**,  $f_k(s_k)$ . (nodpris)

**Rekursiv målfunktion**:  $f_k(s_k) = \min_{x_k} (c_k(x_k, s_k) + f_{k-1}(s_{k-1}))$   
 $= \min_{x_k} (c_k(x_k, s_k) + f_{k-1}(T_k(x_k, s_k)))$



# Dynamisk programmering: Begrepp

I steg  $k$ :

Nod  $j$  i  $S_k$  kallas **tillstånd**,  $s_k$ .

Noden man kommer från,  $l$ , kallas **styrning**,  $x_k$ .

**Överföringsfunktion**: kopplingen mellan olika steg,  $s_{k-1} = T_k(x_k, s_k)$ .  
(föregångare)

**Målfunktionsvärde**,  $f_k(s_k)$ . (nodpris)

**Rekursiv målfunktion**:  $f_k(s_k) = \min_{x_k} (c_k(x_k, s_k) + f_{k-1}(s_{k-1}))$   
 $= \min_{x_k} (c_k(x_k, s_k) + f_{k-1}(T_k(x_k, s_k)))$

**Begränsningar** på tillstånd,  $s_k \in S_k$ , och styrning,  $x_k \in X_k$ .

# Dynamisk programmering: Begrepp

I steg  $k$ :

Nod  $j$  i  $S_k$  kallas **tillstånd**,  $s_k$ .

Noden man kommer från,  $l$ , kallas **styrning**,  $x_k$ .

**Överföringsfunktion**: kopplingen mellan olika steg,  $s_{k-1} = T_k(x_k, s_k)$ .  
(föregångare)

**Målfunktionsvärde**,  $f_k(s_k)$ . (nodpris)

**Rekursiv målfunktion**:  $f_k(s_k) = \min_{x_k} (c_k(x_k, s_k) + f_{k-1}(s_{k-1}))$   
 $= \min_{x_k} (c_k(x_k, s_k) + f_{k-1}(T_k(x_k, s_k)))$

**Begränsningar** på tillstånd,  $s_k \in S_k$ , och styrning,  $x_k \in X_k$ .

**Randvillkor** på  $s_0$ ,  $s_N$ , samt  $f_0(s_0)$ .

# Dynamisk programmering

Det får inte vara möjligt att hoppa mellan olika tillstånd i samma steg.

# Dynamisk programmering

Det får inte vara möjligt att hoppa mellan olika tillstånd i samma steg.  
Man får heller inte hoppa över en nivå.

## Dynamisk programmering

Det får inte vara möjligt att hoppa mellan olika tillstånd i samma steg.

Man får heller inte hoppa över en nivå.

$f_k(s_k)$  får endast vara en funktion av  $s_k$ , dvs. inte av  $x_k$  eller  $s_{k-1}$ .

## Dynamisk programmering

Det får inte vara möjligt att hoppa mellan olika tillstånd i samma steg.

Man får heller inte hoppa över en nivå.

$f_k(s_k)$  får endast vara en funktion av  $s_k$ , dvs. inte av  $x_k$  eller  $s_{k-1}$ .

Målfunktionen måste vara separabel i stegen,  $k$ , och monotont växande i argumenten.

## Dynamisk programmering

Det får inte vara möjligt att hoppa mellan olika tillstånd i samma steg.

Man får heller inte hoppa över en nivå.

$f_k(s_k)$  får endast vara en funktion av  $s_k$ , dvs. inte av  $x_k$  eller  $s_{k-1}$ .

Målfunktionen måste vara separabel i stegen,  $k$ , och monotont växande i argumenten.

Sluttillståndet ska vara unikt, annars kan inte lösningen nystas upp.

## Dynamisk programmering

Det får inte vara möjligt att hoppa mellan olika tillstånd i samma steg.

Man får heller inte hoppa över en nivå.

$f_k(s_k)$  får endast vara en funktion av  $s_k$ , dvs. inte av  $x_k$  eller  $s_{k-1}$ .

Målfunktionen måste vara separabel i stegen,  $k$ , och monotont växande i argumenten.

Sluttillståndet ska vara unikt, annars kan inte lösningen nystas upp.

Man kan notera att dynamisk programmering ger optimal styrning till samtliga tillstånd.



## Dynamisk programmering

Det får inte vara möjligt att hoppa mellan olika tillstånd i samma steg.

Man får heller inte hoppa över en nivå.

$f_k(s_k)$  får endast vara en funktion av  $s_k$ , dvs. inte av  $x_k$  eller  $s_{k-1}$ .

Målfunktionen måste vara separabel i stegen,  $k$ , och monotont växande i argumenten.

Sluttillståndet ska vara unikt, annars kan inte lösningen nystas upp.

Man kan notera att dynamisk programmering ger optimal styrning till samtliga tillstånd.

Detta är s.k. *framåtrekursion*, då man finner den bästa styrningen *till* alla tillstånd.

## Dynamisk programmering

Det får inte vara möjligt att hoppa mellan olika tillstånd i samma steg.

Man får heller inte hoppa över en nivå.

$f_k(s_k)$  får endast vara en funktion av  $s_k$ , dvs. inte av  $x_k$  eller  $s_{k-1}$ .

Målfunktionen måste vara separabel i stegen,  $k$ , och monotont växande i argumenten.

Sluttillståndet ska vara unikt, annars kan inte lösningen nystas upp.

Man kan notera att dynamisk programmering ger optimal styrning till samtliga tillstånd.

Detta är s.k. *framåtrekursion*, då man finner den bästa styrningen *till* alla tillstånd.

En annan möjlighet är *bakåtrekursion*, då man finner den bästa styrningen *från* alla tillstånd.

# Dynamisk programmering

Det får inte vara möjligt att hoppa mellan olika tillstånd i samma steg.

Man får heller inte hoppa över en nivå.

$f_k(s_k)$  får endast vara en funktion av  $s_k$ , dvs. inte av  $x_k$  eller  $s_{k-1}$ .

Målfunktionen måste vara separabel i stegen,  $k$ , och monotont växande i argumenten.

Sluttillståndet ska vara unikt, annars kan inte lösningen nystas upp.

Man kan notera att dynamisk programmering ger optimal styrning till samtliga tillstånd.

Detta är s.k. *framåtrekursion*, då man finner den bästa styrningen *till* alla tillstånd.

En annan möjlighet är *bakåtrekursion*, då man finner den bästa styrningen *från* alla tillstånd.

Man börjar då i det sista steget och arbetar sig baklänges mot starttillståndet, för att sedan nysta upp framåt.

# Typproblem för dynamisk programmering

Tre huvudtyper:

# Typproblem för dynamisk programmering

Tre huvudtyper:

- Lagerhållnings- och produktionsproblem i flera tidsperioder (vilket även inkluderar bytespolicyproblem).

# Typproblem för dynamisk programmering

Tre huvudtyper:

- Lagerhållnings- och produktionsproblem i flera tidsperioder (vilket även inkluderar bytespolicyproblem).
- Bästa väg-problem, med olika målfunktioner (dvs. problemet att finna en väg med vissa egenskaper i ett givet acykliskt nätverk).

# Typproblem för dynamisk programmering

Tre huvudtyper:

- Lagerhållnings- och produktionsproblem i flera tidsperioder (vilket även inkluderar bytespolicyproblem).
- Bästa väg-problem, med olika målfunktioner (dvs. problemet att finna en väg med vissa egenskaper i ett givet acykliskt nätverk).
- Kappsäcksproblem, med olika målfunktioner (dvs. problem med ett övergripande bivillkor, vilket ofta motsvarar fördelning av en begränsad resurs).

# Typproblem för dynamisk programmering

Tre huvudtyper:

- Lagerhållnings- och produktionsproblem i flera tidsperioder (vilket även inkluderar bytestpolicyproblem).
- Bästa väg-problem, med olika målfunktioner (dvs. problemet att finna en väg med vissa egenskaper i ett givet acykliskt nätverk).
- Kappsäcksproblem, med olika målfunktioner (dvs. problem med ett övergripande bivillkor, vilket ofta motsvarar fördelning av en begränsad resurs).

Men angreppssättet betyder att alla problem görs om till väg-problem.



# Typproblem för dynamisk programmering

Tre huvudtyper:

- Lagerhållnings- och produktionsproblem i flera tidsperioder (vilket även inkluderar bytespolicyproblem).
- Bästa väg-problem, med olika målfunktioner (dvs. problemet att finna en väg med vissa egenskaper i ett givet acykliskt nätverk).
- Kappsäcksproblem, med olika målfunktioner (dvs. problem med ett övergripande bivillkor, vilket ofta motsvarar fördelning av en begränsad resurs).

Men angreppssättet betyder att alla problem görs om till väg-problem.

Det är ingen stor skillnad mellan lager- och kappsäcksproblem:

# Typproblem för dynamisk programmering

Tre huvudtyper:

- Lagerhållnings- och produktionsproblem i flera tidsperioder (vilket även inkluderar bytespolicyproblem).
- Bästa väg-problem, med olika målfunktioner (dvs. problemet att finna en väg med vissa egenskaper i ett givet acykliskt nätverk).
- Kappsäcksproblem, med olika målfunktioner (dvs. problem med ett övergripande bivillkor, vilket ofta motsvarar fördelning av en begränsad resurs).

Men angreppssättet betyder att alla problem görs om till väg-problem.

Det är ingen stor skillnad mellan lager- och kappsäcksproblem:

Nivån i kappsäcken är lagernivån

# Typproblem för dynamisk programmering

Tre huvudtyper:

- Lagerhållnings- och produktionsproblem i flera tidsperioder (vilket även inkluderar bytespolicyproblem).
- Bästa väg-problem, med olika målfunktioner (dvs. problemet att finna en väg med vissa egenskaper i ett givet acykliskt nätverk).
- Kappsäcksproblem, med olika målfunktioner (dvs. problem med ett övergripande bivillkor, vilket ofta motsvarar fördelning av en begränsad resurs).

Men angreppssättet betyder att alla problem görs om till väg-problem.

Det är ingen stor skillnad mellan lager- och kappsäcksproblem:

Nivån i kappsäcken är lagernivån (men den minskar vanligtvis aldrig).

# Dynamisk programmering för lagerhållningsproblem

**Stegindelning:** Tidsperiod  $k$  motsvarar steg  $k$ .

# Dynamisk programmering för lagerhållningsproblem

**Stegindelning:** Tidsperiod  $k$  motsvarar steg  $k$ .

**Tillstånd:**  $s_k =$  antal enheter i lager efter period  $k$ .

# Dynamisk programmering för lagerhållningsproblem

**Stegindelning:** Tidsperiod  $k$  motsvarar steg  $k$ .

**Tillstånd:**  $s_k$  = antal enheter i lager efter period  $k$ .

**Styrning:**  $x_k$  = antal enheter som köps/produceras/säljs i period  $k$ .

# Dynamisk programmering för lagerhållningsproblem

**Stegindelning:** Tidsperiod  $k$  motsvarar steg  $k$ .

**Tillstånd:**  $s_k$  = antal enheter i lager efter period  $k$ .

**Styrning:**  $x_k$  = antal enheter som köps/produceras/säljs i period  $k$ .

**Överföringsfunktion:**  $s_{k-1} = s_k - x_k + d_k$ .

# Dynamisk programmering för lagerhållningsproblem

**Stegindelning:** Tidsperiod  $k$  motsvarar steg  $k$ .

**Tillstånd:**  $s_k$  = antal enheter i lager efter period  $k$ .

**Styrning:**  $x_k$  = antal enheter som köps/produceras/säljs i period  $k$ .

**Överföringsfunktion:**  $s_{k-1} = s_k - x_k + d_k$ .

**Målfunktionsvärde:**  $f_k(s_k)$  = minimal kostnad för de  $k$  första tidsperioderna, om  $s_k$  enheter skall finnas i lager efteråt.



# Dynamisk programmering för lagerhållningsproblem

**Stegindelning:** Tidsperiod  $k$  motsvarar steg  $k$ .

**Tillstånd:**  $s_k$  = antal enheter i lager efter period  $k$ .

**Styrning:**  $x_k$  = antal enheter som köps/produceras/säljs i period  $k$ .

**Överföringsfunktion:**  $s_{k-1} = s_k - x_k + d_k$ .

**Målfunktionsvärde:**  $f_k(s_k)$  = minimal kostnad för de  $k$  första tidsperioderna, om  $s_k$  enheter skall finnas i lager efteråt.

**Rekursivt målfunktionsuttryck:**  $f_k(s_k) = \min_{x_k} (c_k(x_k, s_k) + f_{k-1}(s_{k-1}))$

# Dynamisk programmering för lagerhållningsproblem

**Stegindelning:** Tidsperiod  $k$  motsvarar steg  $k$ .

**Tillstånd:**  $s_k$  = antal enheter i lager efter period  $k$ .

**Styrning:**  $x_k$  = antal enheter som köps/produceras/säljs i period  $k$ .

**Överföringsfunktion:**  $s_{k-1} = s_k - x_k + d_k$ .

**Målfunktionsvärde:**  $f_k(s_k)$  = minimal kostnad för de  $k$  första tidsperioderna, om  $s_k$  enheter skall finnas i lager efteråt.

**Rekursivt målfunktionsuttryck:**  $f_k(s_k) = \min_{x_k} (c_k(x_k, s_k) + f_{k-1}(s_{k-1}))$

där  $c_k(x_k, s_k)$  är produktionskostnaden för  $x_k$  enheter och lagerhållningskostnaden för period  $k$  om  $s_k$  enheter finns i lager efteråt.

# Dynamisk programmering för lagerhållningsproblem

**Stegindelning:** Tidsperiod  $k$  motsvarar steg  $k$ .

**Tillstånd:**  $s_k$  = antal enheter i lager efter period  $k$ .

**Styrning:**  $x_k$  = antal enheter som köps/produceras/säljs i period  $k$ .

**Överföringsfunktion:**  $s_{k-1} = s_k - x_k + d_k$ .

**Målfunktionsvärde:**  $f_k(s_k)$  = minimal kostnad för de  $k$  första tidsperioderna, om  $s_k$  enheter skall finnas i lager efteråt.

**Rekursivt målfunktionsuttryck:**  $f_k(s_k) = \min_{x_k} (c_k(x_k, s_k) + f_{k-1}(s_{k-1}))$

där  $c_k(x_k, s_k)$  är produktionskostnaden för  $x_k$  enheter och lagerhållningskostnaden för period  $k$  om  $s_k$  enheter finns i lager efteråt.

**Randvillkor:**  $f_0(s_0) = 0$ ,  $s_0$  är antal enheter i lager vid start av period 1,  $s_N$  är det antal enheter som skall vara i lager efter sista tidsperioden.

# Dynamisk programmering för lagerhållningsproblem

**Stegindelning:** Tidsperiod  $k$  motsvarar steg  $k$ .

**Tillstånd:**  $s_k$  = antal enheter i lager efter period  $k$ .

**Styrning:**  $x_k$  = antal enheter som köps/produceras/säljs i period  $k$ .

**Överföringsfunktion:**  $s_{k-1} = s_k - x_k + d_k$ .

**Målfunktionsvärde:**  $f_k(s_k)$  = minimal kostnad för de  $k$  första tidsperioderna, om  $s_k$  enheter skall finnas i lager efteråt.

**Rekursivt målfunktionsuttryck:**  $f_k(s_k) = \min_{x_k} (c_k(x_k, s_k) + f_{k-1}(s_{k-1}))$

där  $c_k(x_k, s_k)$  är produktionskostnaden för  $x_k$  enheter och lagerhållningskostnaden för period  $k$  om  $s_k$  enheter finns i lager efteråt.

**Randvillkor:**  $f_0(s_0) = 0$ ,  $s_0$  är antal enheter i lager vid start av period 1,  $s_N$  är det antal enheter som skall vara i lager efter sista tidsperioden.

**Begränsningar:**  $0 \leq x \leq s_k + d_k$  och heltal,  $s_k \geq 0$ .

# Dynamisk programmering för lagerhållningsproblem

**Stegindelning:** Tidsperiod  $k$  motsvarar steg  $k$ .

**Tillstånd:**  $s_k$  = antal enheter i lager efter period  $k$ .

**Styrning:**  $x_k$  = antal enheter som köps/produceras/säljs i period  $k$ .

**Överföringsfunktion:**  $s_{k-1} = s_k - x_k + d_k$ .

**Målfunktionsvärde:**  $f_k(s_k)$  = minimal kostnad för de  $k$  första tidsperioderna, om  $s_k$  enheter skall finnas i lager efteråt.

**Rekursivt målfunktionsuttryck:**  $f_k(s_k) = \min_{x_k} (c_k(x_k, s_k) + f_{k-1}(s_{k-1}))$

där  $c_k(x_k, s_k)$  är produktionskostnaden för  $x_k$  enheter och lagerhållningskostnaden för period  $k$  om  $s_k$  enheter finns i lager efteråt.

**Randvillkor:**  $f_0(s_0) = 0$ ,  $s_0$  är antal enheter i lager vid start av period 1,  $s_N$  är det antal enheter som skall vara i lager efter sista tidsperioden.

**Begränsningar:**  $0 \leq x \leq s_k + d_k$  och heltal,  $s_k \geq 0$ .

Ytterligare villkor och kostnader som är uppdelade i tidsperioder går bra att ta med, medan villkor eller kostnader som ger kopplingar mellan tidsperioderna inte går bra.

# Dynamisk programmering för vägproblem

**Tillstånd:**  $s_k$  = slutnod i steg  $k$ .

# Dynamisk programmering för vägproblem

**Tillstånd:**  $s_k$  = slutnod i steg  $k$ .

**Styrning:**  $x_k$  = startnod i steg  $k$ .

# Dynamisk programmering för vägproblem

**Tillstånd:**  $s_k =$  slutnod i steg  $k$ .

**Styrning:**  $x_k =$  startnod i steg  $k$ .

**Överföringsfunktion:**  $s_{k-1} = x_k$ .



# Dynamisk programmering för vägproblem

**Tillstånd:**  $s_k$  = slutnod i steg  $k$ .

**Styrning:**  $x_k$  = startnod i steg  $k$ .

**Överföringsfunktion:**  $s_{k-1} = x_k$ .

**Målfunktionsvärde:**  $f_k(s_k)$  = kostnad för billigaste väg från startnoden till nod  $s_k$ .

# Dynamisk programmering för vägproblem

**Tillstånd:**  $s_k$  = slutnod i steg  $k$ .

**Styrning:**  $x_k$  = startnod i steg  $k$ .

**Överföringsfunktion:**  $s_{k-1} = x_k$ .

**Målfunktionsvärde:**  $f_k(s_k)$  = kostnad för billigaste väg från startnoden till nod  $s_k$ .

**Rekursivt målfunktionsuttryck:**  $f_k(s_k) = \min_{x_k} (c_k(x_k, s_k) + f_{k-1}(s_{k-1}))$

där  $c_k(x_k, s_k)$  är kostnaden för bågen från nod  $x_k$  till nod  $s_k$ .

# Dynamisk programmering för vägproblem

**Tillstånd:**  $s_k$  = slutnod i steg  $k$ .

**Styrning:**  $x_k$  = startnod i steg  $k$ .

**Överföringsfunktion:**  $s_{k-1} = x_k$ .

**Målfunktionsvärde:**  $f_k(s_k)$  = kostnad för billigaste väg från startnoden till nod  $s_k$ .

**Rekursivt målfunktionsuttryck:**  $f_k(s_k) = \min_{x_k} (c_k(x_k, s_k) + f_{k-1}(s_{k-1}))$

där  $c_k(x_k, s_k)$  är kostnaden för bågen från nod  $x_k$  till nod  $s_k$ .

**Randvillkor:**  $f_0(s_0) = 0$ ,  $s_0$  = startnoden,  $s_N$  = slutnoden.

# Dynamisk programmering för vägproblem

**Tillstånd:**  $s_k$  = slutnod i steg  $k$ .

**Styrning:**  $x_k$  = startnod i steg  $k$ .

**Överföringsfunktion:**  $s_{k-1} = x_k$ .

**Målfunktionsvärde:**  $f_k(s_k)$  = kostnad för billigaste väg från startnoden till nod  $s_k$ .

**Rekursivt målfunktionsuttryck:**  $f_k(s_k) = \min_{x_k} (c_k(x_k, s_k) + f_{k-1}(s_{k-1}))$

där  $c_k(x_k, s_k)$  är kostnaden för bågen från nod  $x_k$  till nod  $s_k$ .

**Randvillkor:**  $f_0(s_0) = 0$ ,  $s_0$  = startnoden,  $s_N$  = slutnoden.

**Begränsningar:**  $(x_k, s_k) \in B$  (dvs. endast befintliga bågar får användas).

## Dynamisk programmering: Kappsäcksproblem

$$v^* = \max \sum_{j=1}^n c_j x_j$$

## Dynamisk programmering: Kappsäcksproblem

$$\begin{aligned} v^* = \max \quad & \sum_{j=1}^n c_j x_j \\ \text{då} \quad & \sum_{j=1}^n a_j x_j \leq b \end{aligned} \quad (1)$$

## Dynamisk programmering: Kappsäcksproblem

$$v^* = \max \sum_{j=1}^n c_j x_j$$
$$\text{då } \sum_{j=1}^n a_j x_j \leq b \quad (1)$$
$$0 \leq x_j \leq u_j \text{ heltal } \quad \forall j \quad (2)$$

## Dynamisk programmering: Kappsäcksproblem

$$v^* = \max \sum_{j=1}^n c_j x_j$$
$$\text{då } \sum_{j=1}^n a_j x_j \leq b \quad (1)$$
$$0 \leq x_j \leq u_j \text{ heltal } \quad \forall j \quad (2)$$

Observera: Ett bivillkor (förutom gränser).



## Dynamisk programmering: Kappsäcksproblem

$$v^* = \max \sum_{j=1}^n c_j x_j$$
$$\text{då } \sum_{j=1}^n a_j x_j \leq b \quad (1)$$
$$0 \leq x_j \leq u_j \text{ heltal } \quad \forall j \quad (2)$$

Observera: Ett bivillkor (förutom gränser).

Varje variabel ses som en nivå.

## Dynamisk programmering: Kappsäcksproblem

$$v^* = \max \sum_{j=1}^n c_j x_j$$
$$\text{då } \sum_{j=1}^n a_j x_j \leq b \quad (1)$$
$$0 \leq x_j \leq u_j \text{ heltal } \quad \forall j \quad (2)$$

Observera: Ett bivillkor (förutom gränser).

Varje variabel ses som en nivå. Dvs. man bestämmer en variabel i taget.

## Dynamisk programmering: Kappsäcksproblem

$$v^* = \max \sum_{j=1}^n c_j x_j$$
$$\text{då } \sum_{j=1}^n a_j x_j \leq b \quad (1)$$
$$0 \leq x_j \leq u_j \text{ heltal } \quad \forall j \quad (2)$$

Observera: Ett bivillkor (förutom gränser).

Varje variabel ses som en nivå. Dvs. man bestämmer en variabel i taget.

Högerledet  $b$  kan ses som en typ av lagernivå som binder ihop variablerna.

## Dynamisk programmering: Kappsäcksproblem

$$v^* = \max \sum_{j=1}^n c_j x_j$$
$$\text{då } \sum_{j=1}^n a_j x_j \leq b \quad (1)$$
$$0 \leq x_j \leq u_j \text{ heltal } \quad \forall j \quad (2)$$

Observera: Ett bivillkor (förutom gränser).

Varje variabel ses som en nivå. Dvs. man bestämmer en variabel i taget.

Högerledet  $b$  kan ses som en typ av lagernivå som binder ihop variablerna.

Varje nivå innebär ett (möjligt) ökat utnyttjande av den gemensamma resursen  $b$ .

## Dynamisk programmering: Kappsäcksproblem

$$v^* = \max \sum_{j=1}^n c_j x_j$$
$$\text{då } \sum_{j=1}^n a_j x_j \leq b \quad (1)$$
$$0 \leq x_j \leq u_j \text{ heltal } \quad \forall j \quad (2)$$

Observera: Ett bivillkor (förutom gränser).

Varje variabel ses som en nivå. Dvs. man bestämmer en variabel i taget.

Högerledet  $b$  kan ses som en typ av lagernivå som binder ihop variablerna.

Varje nivå innebär ett (möjligt) ökat utnyttjande av den gemensamma resursen  $b$ .

Tillstånd:  $s_k$  = den del av högerledet  $b$  som får användas till de  $k$  första variablerna.

## Dynamisk programmering för kappsäcksproblem

$$\max \sum_j c_j x_j \text{ då } \sum_j a_j x_j \leq b, 0 \leq x_j \leq u_j \text{ heltal } \forall j$$

## Dynamisk programmering för kappsäcksproblem

$$\max \sum_j c_j x_j \text{ då } \sum_j a_j x_j \leq b, 0 \leq x_j \leq u_j \text{ heltal } \forall j$$

En variabel i varje steg.

## Dynamisk programmering för kappsäcksproblem

$$\max \sum_j c_j x_j \text{ då } \sum_j a_j x_j \leq b, 0 \leq x_j \leq u_j \text{ heltal } \forall j$$

En variabel i varje steg.

**Tillstånd:**  $s_k$  = den del av högerledet  $b$  som får användas till de  $k$  första variablerna.



# Dynamisk programmering för kappsäcksproblem

$$\max \sum_j c_j x_j \text{ då } \sum_j a_j x_j \leq b, 0 \leq x_j \leq u_j \text{ heltal } \forall j$$

En variabel i varje steg.

**Tillstånd:**  $s_k$  = den del av högerledet  $b$  som får användas till de  $k$  första variablerna.

**Överföringsfunktion:**  $s_{k-1} = s_k - a_k x_k$

# Dynamisk programmering för kappsäcksproblem

$$\max \sum_j c_j x_j \text{ då } \sum_j a_j x_j \leq b, 0 \leq x_j \leq u_j \text{ heltal } \forall j$$

En variabel i varje steg.

**Tillstånd:**  $s_k$  = den del av högerledet  $b$  som får användas till de  $k$  första variablerna.

**Överföringsfunktion:**  $s_{k-1} = s_k - a_k x_k$

**Målfunktionsvärde:**  $f_k(s_k)$  = maximalt värde av de  $k$  första sorterna, om vi får använda  $s_k$  enheter av kappsäcken.

# Dynamisk programmering för kappsäcksproblem

$$\max \sum_j c_j x_j \text{ då } \sum_j a_j x_j \leq b, 0 \leq x_j \leq u_j \text{ heltal } \forall j$$

En variabel i varje steg.

**Tillstånd:**  $s_k$  = den del av högerledet  $b$  som får användas till de  $k$  första variablerna.

**Överföringsfunktion:**  $s_{k-1} = s_k - a_k x_k$

**Målfunktionsvärde:**  $f_k(s_k)$  = maximalt värde av de  $k$  första sorterna, om vi får använda  $s_k$  enheter av kappsäcken.

**Rekursivt målfunktionsuttryck:**  $f_k(s_k) = \max_{x_k} (c_k x_k + f_{k-1}(s_{k-1}))$ .

# Dynamisk programmering för kappsäcksproblem

$$\max \sum_j c_j x_j \text{ då } \sum_j a_j x_j \leq b, 0 \leq x_j \leq u_j \text{ heltal } \forall j$$

En variabel i varje steg.

**Tillstånd:**  $s_k$  = den del av högerledet  $b$  som får användas till de  $k$  första variablerna.

**Överföringsfunktion:**  $s_{k-1} = s_k - a_k x_k$

**Målfunktionsvärde:**  $f_k(s_k)$  = maximalt värde av de  $k$  första sorterna, om vi får använda  $s_k$  enheter av kappsäcken.

**Rekursivt målfunktionsuttryck:**  $f_k(s_k) = \max_{x_k} (c_k x_k + f_{k-1}(s_{k-1}))$ .

**Randvillkor:**  $f_0(s_0) = 0, s_0 \geq 0, s_N = b$ .

# Dynamisk programmering för kappsäcksproblem

$$\max \sum_j c_j x_j \text{ då } \sum_j a_j x_j \leq b, 0 \leq x_j \leq u_j \text{ heltal } \forall j$$

En variabel i varje steg.

**Tillstånd:**  $s_k$  = den del av högerledet  $b$  som får användas till de  $k$  första variablerna.

**Överföringsfunktion:**  $s_{k-1} = s_k - a_k x_k$

**Målfunktionsvärde:**  $f_k(s_k)$  = maximalt värde av de  $k$  första sorterna, om vi får använda  $s_k$  enheter av kappsäcken.

**Rekursivt målfunktionsuttryck:**  $f_k(s_k) = \max_{x_k} (c_k x_k + f_{k-1}(s_{k-1}))$ .

**Randvillkor:**  $f_0(s_0) = 0$ ,  $s_0 \geq 0$ ,  $s_N = b$ .

**Begränsningar:**  $0 \leq x_k \leq u_k$  och heltal,  $0 \leq s_k \leq b$ ,  $x_k \leq \lfloor s_k/a_k \rfloor$ .

# Dynamisk programmering för kappsäcksproblem

$$\max \sum_j c_j x_j \text{ då } \sum_j a_j x_j \leq b, 0 \leq x_j \leq u_j \text{ heltal } \forall j$$

En variabel i varje steg.

**Tillstånd:**  $s_k$  = den del av högerledet  $b$  som får användas till de  $k$  första variablerna.

**Överföringsfunktion:**  $s_{k-1} = s_k - a_k x_k$

**Målfunktionsvärde:**  $f_k(s_k)$  = maximalt värde av de  $k$  första sorterna, om vi får använda  $s_k$  enheter av kappsäcken.

**Rekursivt målfunktionsuttryck:**  $f_k(s_k) = \max_{x_k} (c_k x_k + f_{k-1}(s_{k-1}))$ .

**Randvillkor:**  $f_0(s_0) = 0$ ,  $s_0 \geq 0$ ,  $s_N = b$ .

**Begränsningar:**  $0 \leq x_k \leq u_k$  och heltal,  $0 \leq s_k \leq b$ ,  $x_k \leq \lfloor s_k/a_k \rfloor$ .

Slacket i kappsäcken ges av  $s_0$ .

# Dynamisk programmering för kappsäcksproblem

$$\max \sum_j c_j x_j \text{ då } \sum_j a_j x_j \leq b, 0 \leq x_j \leq u_j \text{ heltal } \forall j$$

En variabel i varje steg.

**Tillstånd:**  $s_k$  = den del av högerledet  $b$  som får användas till de  $k$  första variablerna.

**Överföringsfunktion:**  $s_{k-1} = s_k - a_k x_k$

**Målfunktionsvärde:**  $f_k(s_k)$  = maximalt värde av de  $k$  första sorterna, om vi får använda  $s_k$  enheter av kappsäcken.

**Rekursivt målfunktionsuttryck:**  $f_k(s_k) = \max_{x_k} (c_k x_k + f_{k-1}(s_{k-1}))$ .

**Randvillkor:**  $f_0(s_0) = 0, s_0 \geq 0, s_N = b$ .

**Begränsningar:**  $0 \leq x_k \leq u_k$  och heltal,  $0 \leq s_k \leq b, x_k \leq \lfloor s_k/a_k \rfloor$ .

Slacket i kappsäcken ges av  $s_0$ . För likhetsvillkor sätts  $s_0 = 0$ .

# Dynamisk programmering för kappsäcksproblem

$$\max \sum_j c_j x_j \text{ då } \sum_j a_j x_j \leq b, 0 \leq x_j \leq u_j \text{ heltal } \forall j$$

En variabel i varje steg.

**Tillstånd:**  $s_k$  = den del av högerledet  $b$  som får användas till de  $k$  första variablerna.

**Överföringsfunktion:**  $s_{k-1} = s_k - a_k x_k$

**Målfunktionsvärde:**  $f_k(s_k)$  = maximalt värde av de  $k$  första sorterna, om vi får använda  $s_k$  enheter av kappsäcken.

**Rekursivt målfunktionsuttryck:**  $f_k(s_k) = \max_{x_k} (c_k x_k + f_{k-1}(s_{k-1}))$ .

**Randvillkor:**  $f_0(s_0) = 0$ ,  $s_0 \geq 0$ ,  $s_N = b$ .

**Begränsningar:**  $0 \leq x_k \leq u_k$  och heltal,  $0 \leq s_k \leq b$ ,  $x_k \leq \lfloor s_k / a_k \rfloor$ .

Slacket i kappsäcken ges av  $s_0$ . För likhetsvillkor sätts  $s_0 = 0$ .

Problemets svårighetsgrad beror på  $b$ , antalet tillstånd i varje steg.



## Dynamisk programmering: Kappsäcksproblem: Exempel

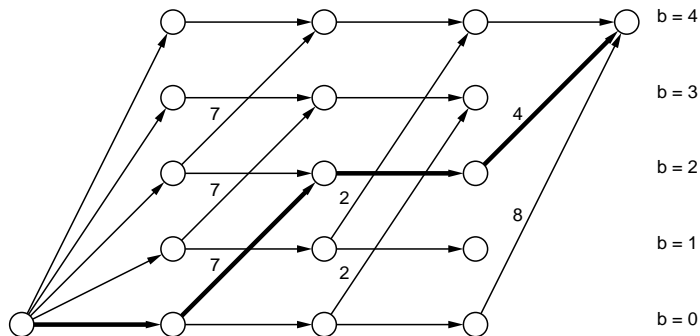
$$\max 7x_1 + 2x_2 + 4x_3$$

$$\text{då } 2x_1 + 3x_2 + 2x_3 \leq 4, x_1 \in \{0, 1\}, x_2 \in \{0, 1\}, x_3 \in \{0, 1, 2\}$$

# Dynamisk programmering: Kappsäcksproblem: Exempel

$$\max 7x_1 + 2x_2 + 4x_3$$

$$\text{då } 2x_1 + 3x_2 + 2x_3 \leq 4, x_1 \in \{0, 1\}, x_2 \in \{0, 1\}, x_3 \in \{0, 1, 2\}$$



## Dynamisk programmering: Kappsäcksproblem: Exempel

$$\max 3x_1 + 4x_2 + 2x_3 + 5x_4 + 6x_5 + 5x_6$$

$$\text{då } 2x_1 + 3x_2 + 5x_3 + 2x_4 + 3x_5 + 4x_6 \leq 9, x_j \in \{0, 1, 2\} \forall j$$

## Dynamisk programmering: Kappsäcksproblem: Exempel

$$\max 3x_1 + 4x_2 + 2x_3 + 5x_4 + 6x_5 + 5x_6$$

$$\text{då } 2x_1 + 3x_2 + 5x_3 + 2x_4 + 3x_5 + 4x_6 \leq 9, x_j \in \{0, 1, 2\} \forall j$$

$$f_0(s_0) = 0, s_0 \geq 0, s_6 = 9.$$

## Dynamisk programmering: Kappsäcksproblem: Exempel

$$\max 3x_1 + 4x_2 + 2x_3 + 5x_4 + 6x_5 + 5x_6$$

$$\text{då } 2x_1 + 3x_2 + 5x_3 + 2x_4 + 3x_5 + 4x_6 \leq 9, x_j \in \{0, 1, 2\} \forall j$$

$$f_0(s_0) = 0, s_0 \geq 0, s_6 = 9.$$

Steg 1 ( $x_1$ ):

## Dynamisk programmering: Kappsäcksproblem: Exempel

$$\max 3x_1 + 4x_2 + 2x_3 + 5x_4 + 6x_5 + 5x_6$$

$$\text{då } 2x_1 + 3x_2 + 5x_3 + 2x_4 + 3x_5 + 4x_6 \leq 9, x_j \in \{0, 1, 2\} \forall j$$

$$f_0(s_0) = 0, s_0 \geq 0, s_6 = 9.$$

Steg 1 ( $x_1$ ):

$$0 \leq s_1 \leq 9, x_1 \in \{0, 1, 2\}.$$

## Dynamisk programmering: Kappsäcksproblem: Exempel

$$\max 3x_1 + 4x_2 + 2x_3 + 5x_4 + 6x_5 + 5x_6$$

$$\text{då } 2x_1 + 3x_2 + 5x_3 + 2x_4 + 3x_5 + 4x_6 \leq 9, x_j \in \{0, 1, 2\} \quad \forall j$$

$$f_0(s_0) = 0, s_0 \geq 0, s_6 = 9.$$

Steg 1 ( $x_1$ ):

$$0 \leq s_1 \leq 9, x_1 \in \{0, 1, 2\}.$$

$$f_1(s_1) = \max_{x_1} (c_1 x_1 + f_0(s_0)) = \max_{x_1} (3x_1) \quad \text{då } x_1 \leq \lfloor s_1/a_1 \rfloor = \lfloor s_1/2 \rfloor$$

## Dynamisk programmering: Kappsäcksproblem: Exempel

$$\max 3x_1 + 4x_2 + 2x_3 + 5x_4 + 6x_5 + 5x_6$$

$$\text{då } 2x_1 + 3x_2 + 5x_3 + 2x_4 + 3x_5 + 4x_6 \leq 9, x_j \in \{0, 1, 2\} \forall j$$

$$f_0(s_0) = 0, s_0 \geq 0, s_6 = 9.$$

Steg 1 ( $x_1$ ):

$$0 \leq s_1 \leq 9, x_1 \in \{0, 1, 2\}.$$

$$f_1(s_1) = \max_{x_1} (c_1 x_1 + f_0(s_0)) = \max_{x_1} (3x_1) \quad \text{då } x_1 \leq \lfloor s_1/a_1 \rfloor = \lfloor s_1/2 \rfloor$$

dvs.  $x_1 = 0$  om  $s_1 < 2$  och  $x_1 \leq 1$  om  $s_1 < 4$ .



## Dynamisk programmering: Kappsäcksproblem: Exempel

$$\max 3x_1 + 4x_2 + 2x_3 + 5x_4 + 6x_5 + 5x_6$$

$$\text{då } 2x_1 + 3x_2 + 5x_3 + 2x_4 + 3x_5 + 4x_6 \leq 9, x_j \in \{0, 1, 2\} \forall j$$

$$f_0(s_0) = 0, s_0 \geq 0, s_6 = 9.$$

Steg 1 ( $x_1$ ):

$$0 \leq s_1 \leq 9, x_1 \in \{0, 1, 2\}.$$

$$f_1(s_1) = \max_{x_1} (c_1 x_1 + f_0(s_0)) = \max_{x_1} (3x_1) \quad \text{då } x_1 \leq \lfloor s_1/a_1 \rfloor = \lfloor s_1/2 \rfloor$$

dvs.  $x_1 = 0$  om  $s_1 < 2$  och  $x_1 \leq 1$  om  $s_1 < 4$ .

Gör optimeringen i en tabell:

## Dynamisk programmering: Kappsäcksproblem: Exempel

$$\max 3x_1 + 4x_2 + 2x_3 + 5x_4 + 6x_5 + 5x_6$$

$$\text{då } 2x_1 + 3x_2 + 5x_3 + 2x_4 + 3x_5 + 4x_6 \leq 9, x_j \in \{0, 1, 2\} \forall j$$

$$f_0(s_0) = 0, s_0 \geq 0, s_6 = 9.$$

Steg 1 ( $x_1$ ):

$$0 \leq s_1 \leq 9, x_1 \in \{0, 1, 2\}.$$

$$f_1(s_1) = \max_{x_1} (c_1 x_1 + f_0(s_0)) = \max_{x_1} (3x_1) \quad \text{då } x_1 \leq \lfloor s_1/a_1 \rfloor = \lfloor s_1/2 \rfloor$$

dvs.  $x_1 = 0$  om  $s_1 < 2$  och  $x_1 \leq 1$  om  $s_1 < 4$ .

Gör optimeringen i en tabell:

$x_1 \backslash s_1$	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	-	-	3	3	3	3	3	3	3	3
2	-	-	-	-	6	6	6	6	6	6
$f_1(s_1)$	0	0	3	3	6	6	6	6	6	6
$\hat{x}_1(s_1)$	0	0	1	1	2	2	2	2	2	2

# Dynamisk programmering: Kappsäcksproblem: Exempel

Steg 2 ( $x_2$ ):

## Dynamisk programmering: Kappsäcksproblem: Exempel

Steg 2 ( $x_2$ ):  $0 \leq s_2 \leq 9$ ,  $x_2 \in \{0, 1, 2\}$ .

## Dynamisk programmering: Kappsäcksproblem: Exempel

Steg 2 ( $x_2$ ):  $0 \leq s_2 \leq 9$ ,  $x_2 \in \{0, 1, 2\}$ .

Det enda som behöver sparas från förra steget är  $f_1(s_1)$ .

## Dynamisk programmering: Kappsäcksproblem: Exempel

Steg 2 ( $x_2$ ):  $0 \leq s_2 \leq 9$ ,  $x_2 \in \{0, 1, 2\}$ .

Det enda som behöver sparas från förra steget är  $f_1(s_1)$ .

$$s_1 = s_2 - a_2x_2 = s_2 - 3x_2.$$

## Dynamisk programmering: Kappsäcksproblem: Exempel

Steg 2 ( $x_2$ ):  $0 \leq s_2 \leq 9$ ,  $x_2 \in \{0, 1, 2\}$ .

Det enda som behöver sparas från förra steget är  $f_1(s_1)$ .

$$s_1 = s_2 - a_2 x_2 = s_2 - 3x_2.$$

$$f_2(s_2) = \max_{x_2} (c_2 x_2 + f_1(s_1)) = \max_{x_2} (4x_2 + f_1(s_2 - 3x_2))$$

då  $x_2 \leq \lfloor s_2/a_2 \rfloor = \lfloor s_2/3 \rfloor$  dvs.  $x_2 = 0$  om  $s_2 < 3$  och  $x_2 \leq 1$  om  $s_2 < 6$ .

## Dynamisk programmering: Kappsäcksproblem: Exempel

Steg 2 ( $x_2$ ):  $0 \leq s_2 \leq 9$ ,  $x_2 \in \{0, 1, 2\}$ .

Det enda som behöver sparas från förra steget är  $f_1(s_1)$ .

$$s_1 = s_2 - a_2 x_2 = s_2 - 3x_2.$$

$$f_2(s_2) = \max_{x_2} (c_2 x_2 + f_1(s_1)) = \max_{x_2} (4x_2 + f_1(s_2 - 3x_2))$$

då  $x_2 \leq \lfloor s_2/a_2 \rfloor = \lfloor s_2/3 \rfloor$  dvs.  $x_2 = 0$  om  $s_2 < 3$  och  $x_2 \leq 1$  om  $s_2 < 6$ .

Om  $x_2 = 0$  fås  $f_1(s_2)$ .



## Dynamisk programmering: Kappsäcksproblem: Exempel

Steg 2 ( $x_2$ ):  $0 \leq s_2 \leq 9$ ,  $x_2 \in \{0, 1, 2\}$ .

Det enda som behöver sparas från förra steget är  $f_1(s_1)$ .

$$s_1 = s_2 - a_2 x_2 = s_2 - 3x_2.$$

$$f_2(s_2) = \max_{x_2} (c_2 x_2 + f_1(s_1)) = \max_{x_2} (4x_2 + f_1(s_2 - 3x_2))$$

då  $x_2 \leq \lfloor s_2/a_2 \rfloor = \lfloor s_2/3 \rfloor$  dvs.  $x_2 = 0$  om  $s_2 < 3$  och  $x_2 \leq 1$  om  $s_2 < 6$ .

Om  $x_2 = 0$  fås  $f_1(s_2)$ .

Om  $x_2 = 1$  fås  $4 + f_1(s_2 - 3)$  (flytta 3 steg åt höger och addera 4).

## Dynamisk programmering: Kappsäcksproblem: Exempel

Steg 2 ( $x_2$ ):  $0 \leq s_2 \leq 9$ ,  $x_2 \in \{0, 1, 2\}$ .

Det enda som behöver sparas från förra steget är  $f_1(s_1)$ .

$$s_1 = s_2 - a_2x_2 = s_2 - 3x_2.$$

$$f_2(s_2) = \max_{x_2}(c_2x_2 + f_1(s_1)) = \max_{x_2}(4x_2 + f_1(s_2 - 3x_2))$$

då  $x_2 \leq \lfloor s_2/a_2 \rfloor = \lfloor s_2/3 \rfloor$  dvs.  $x_2 = 0$  om  $s_2 < 3$  och  $x_2 \leq 1$  om  $s_2 < 6$ .

Om  $x_2 = 0$  fås  $f_1(s_2)$ .

Om  $x_2 = 1$  fås  $4 + f_1(s_2 - 3)$  (flytta 3 steg åt höger och addera 4).

Om  $x_2 = 2$  fås  $8 + f_1(s_2 - 6)$  (flytta 6 steg åt höger och addera 8).

## Dynamisk programmering: Kappsäcksproblem: Exempel

Steg 2 ( $x_2$ ):  $0 \leq s_2 \leq 9$ ,  $x_2 \in \{0, 1, 2\}$ .

Det enda som behöver sparas från förra steget är  $f_1(s_1)$ .

$$s_1 = s_2 - a_2 x_2 = s_2 - 3x_2.$$

$$f_2(s_2) = \max_{x_2} (c_2 x_2 + f_1(s_1)) = \max_{x_2} (4x_2 + f_1(s_2 - 3x_2))$$

då  $x_2 \leq \lfloor s_2/a_2 \rfloor = \lfloor s_2/3 \rfloor$  dvs.  $x_2 = 0$  om  $s_2 < 3$  och  $x_2 \leq 1$  om  $s_2 < 6$ .

Om  $x_2 = 0$  fås  $f_1(s_2)$ .

Om  $x_2 = 1$  fås  $4 + f_1(s_2 - 3)$  (flytta 3 steg åt höger och addera 4).

Om  $x_2 = 2$  fås  $8 + f_1(s_2 - 6)$  (flytta 6 steg åt höger och addera 8).

$x_2 \backslash s_2$	0	1	2	3	4	5	6	7	8	9
0	0	0	3	3	6	6	6	6	6	6
1	-	-	-	4	4	7	7	10	10	10
2	-	-	-	-	-	-	8	8	11	11
$f_2(s_2)$	0	0	3	4	6	7	8	10	11	11
$\hat{x}_2(s_2)$	0	0	0	1	0	1	2	1	2	2

## Dynamisk programmering: Kappsäcksproblem: Exempel

Steg 3 ( $x_3$ ):  $0 \leq s_3 \leq 9$ ,  $x_3 \in \{0, 1, 2\}$ .

## Dynamisk programmering: Kappsäcksproblem: Exempel

Steg 3 ( $x_3$ ):  $0 \leq s_3 \leq 9$ ,  $x_3 \in \{0, 1, 2\}$ .

$$s_2 = s_3 - a_3x_3 = s_3 - 5x_3.$$

## Dynamisk programmering: Kappsäcksproblem: Exempel

Steg 3 ( $x_3$ ):  $0 \leq s_3 \leq 9$ ,  $x_3 \in \{0, 1, 2\}$ .

$$s_2 = s_3 - a_3x_3 = s_3 - 5x_3.$$

$$f_3(s_3) = \max_{x_3}(c_3x_3 + f_2(s_2)) = \max_{x_3}(2x_3 + f_2(s_3 - 5x_3))$$

då  $x_3 \leq \lfloor s_3/a_3 \rfloor = \lfloor s_3/5 \rfloor$  dvs.  $x_3 = 0$  om  $s_3 < 5$  och  $x_3 \leq 1$  om  $s_3 < 10$ .

## Dynamisk programmering: Kappsäcksproblem: Exempel

Steg 3 ( $x_3$ ):  $0 \leq s_3 \leq 9$ ,  $x_3 \in \{0, 1, 2\}$ .

$$s_2 = s_3 - a_3x_3 = s_3 - 5x_3.$$

$$f_3(s_3) = \max_{x_3}(c_3x_3 + f_2(s_2)) = \max_{x_3}(2x_3 + f_2(s_3 - 5x_3))$$

då  $x_3 \leq \lfloor s_3/a_3 \rfloor = \lfloor s_3/5 \rfloor$  dvs.  $x_3 = 0$  om  $s_3 < 5$  och  $x_3 \leq 1$  om  $s_3 < 10$ .

Om  $x_3 = 0$  fås  $f_2(s_3)$ .

## Dynamisk programmering: Kappsäcksproblem: Exempel

Steg 3 ( $x_3$ ):  $0 \leq s_3 \leq 9$ ,  $x_3 \in \{0, 1, 2\}$ .

$$s_2 = s_3 - a_3x_3 = s_3 - 5x_3.$$

$$f_3(s_3) = \max_{x_3}(c_3x_3 + f_2(s_2)) = \max_{x_3}(2x_3 + f_2(s_3 - 5x_3))$$

då  $x_3 \leq \lfloor s_3/a_3 \rfloor = \lfloor s_3/5 \rfloor$  dvs.  $x_3 = 0$  om  $s_3 < 5$  och  $x_3 \leq 1$  om  $s_3 < 10$ .

Om  $x_3 = 0$  fås  $f_2(s_3)$ .

Om  $x_3 = 1$  fås  $2 + f_2(s_3 - 5)$  (flytta 5 steg åt höger och addera 2).



## Dynamisk programmering: Kappsäcksproblem: Exempel

Steg 3 ( $x_3$ ):  $0 \leq s_3 \leq 9$ ,  $x_3 \in \{0, 1, 2\}$ .

$$s_2 = s_3 - a_3x_3 = s_3 - 5x_3.$$

$$f_3(s_3) = \max_{x_3}(c_3x_3 + f_2(s_2)) = \max_{x_3}(2x_3 + f_2(s_3 - 5x_3))$$

då  $x_3 \leq \lfloor s_3/a_3 \rfloor = \lfloor s_3/5 \rfloor$  dvs.  $x_3 = 0$  om  $s_3 < 5$  och  $x_3 \leq 1$  om  $s_3 < 10$ .

Om  $x_3 = 0$  fås  $f_2(s_3)$ .

Om  $x_3 = 1$  fås  $2 + f_2(s_3 - 5)$  (flytta 5 steg åt höger och addera 2).

Om  $x_3 = 2$  fås  $4 + f_2(s_3 - 10)$  (flytta 10 steg åt höger och addera 4).

## Dynamisk programmering: Kappsäcksproblem: Exempel

Steg 3 ( $x_3$ ):  $0 \leq s_3 \leq 9$ ,  $x_3 \in \{0, 1, 2\}$ .

$$s_2 = s_3 - a_3x_3 = s_3 - 5x_3.$$

$$f_3(s_3) = \max_{x_3}(c_3x_3 + f_2(s_2)) = \max_{x_3}(2x_3 + f_2(s_3 - 5x_3))$$

då  $x_3 \leq \lfloor s_3/a_3 \rfloor = \lfloor s_3/5 \rfloor$  dvs.  $x_3 = 0$  om  $s_3 < 5$  och  $x_3 \leq 1$  om  $s_3 < 10$ .

Om  $x_3 = 0$  fås  $f_2(s_3)$ .

Om  $x_3 = 1$  fås  $2 + f_2(s_3 - 5)$  (flytta 5 steg åt höger och addera 2).

Om  $x_3 = 2$  fås  $4 + f_2(s_3 - 10)$  (flytta 10 steg åt höger och addera 4).

$x_3 \backslash s_3$	0	1	2	3	4	5	6	7	8	9
0	0	0	3	4	6	7	8	10	11	11
1	-	-	-	-	-	2	2	5	6	8
2	-	-	-	-	-	-	-	-	-	-
$f_3(s_3)$	0	0	3	4	6	7	8	10	11	11
$\hat{x}_3(s_3)$	0	0	0	0	0	0	0	0	0	0

## Dynamisk programmering: Kappsäcksproblem: Exempel

Steg 4 ( $x_4$ ):  $0 \leq s_4 \leq 9$ ,  $x_4 \in \{0, 1, 2\}$ .

## Dynamisk programmering: Kappsäcksproblem: Exempel

Steg 4 ( $x_4$ ):  $0 \leq s_4 \leq 9$ ,  $x_4 \in \{0, 1, 2\}$ .

$$s_3 = s_4 - a_4 x_4 = s_4 - 2x_4.$$

## Dynamisk programmering: Kappsäcksproblem: Exempel

Steg 4 ( $x_4$ ):  $0 \leq s_4 \leq 9$ ,  $x_4 \in \{0, 1, 2\}$ .

$$s_3 = s_4 - a_4 x_4 = s_4 - 2x_4.$$

$$f_4(s_4) = \max_{x_4} (c_4 x_4 + f_3(s_3)) = \max_{x_4} (5x_4 + f_3(s_4 - 2x_4))$$

då  $x_4 \leq \lfloor s_4/a_4 \rfloor = \lfloor s_4/2 \rfloor$  dvs.  $x_4 = 0$  om  $s_4 < 2$  och  $x_4 \leq 1$  om  $s_4 < 4$ .

## Dynamisk programmering: Kappsäcksproblem: Exempel

Steg 4 ( $x_4$ ):  $0 \leq s_4 \leq 9$ ,  $x_4 \in \{0, 1, 2\}$ .

$$s_3 = s_4 - a_4 x_4 = s_4 - 2x_4.$$

$$f_4(s_4) = \max_{x_4} (c_4 x_4 + f_3(s_3)) = \max_{x_4} (5x_4 + f_3(s_4 - 2x_4))$$

då  $x_4 \leq \lfloor s_4/a_4 \rfloor = \lfloor s_4/2 \rfloor$  dvs.  $x_4 = 0$  om  $s_4 < 2$  och  $x_4 \leq 1$  om  $s_4 < 4$ .

Om  $x_4 = 0$  fås  $f_3(s_4)$ .

## Dynamisk programmering: Kappsäcksproblem: Exempel

Steg 4 ( $x_4$ ):  $0 \leq s_4 \leq 9$ ,  $x_4 \in \{0, 1, 2\}$ .

$$s_3 = s_4 - a_4 x_4 = s_4 - 2x_4.$$

$$f_4(s_4) = \max_{x_4} (c_4 x_4 + f_3(s_3)) = \max_{x_4} (5x_4 + f_3(s_4 - 2x_4))$$

då  $x_4 \leq \lfloor s_4/a_4 \rfloor = \lfloor s_4/2 \rfloor$  dvs.  $x_4 = 0$  om  $s_4 < 2$  och  $x_4 \leq 1$  om  $s_4 < 4$ .

Om  $x_4 = 0$  fås  $f_3(s_4)$ .

Om  $x_4 = 1$  fås  $5 + f_3(s_4 - 2)$  (flytta 2 steg åt höger och addera 5).

## Dynamisk programmering: Kappsäcksproblem: Exempel

Steg 4 ( $x_4$ ):  $0 \leq s_4 \leq 9$ ,  $x_4 \in \{0, 1, 2\}$ .

$$s_3 = s_4 - a_4 x_4 = s_4 - 2x_4.$$

$$f_4(s_4) = \max_{x_4} (c_4 x_4 + f_3(s_3)) = \max_{x_4} (5x_4 + f_3(s_4 - 2x_4))$$

då  $x_4 \leq \lfloor s_4/a_4 \rfloor = \lfloor s_4/2 \rfloor$  dvs.  $x_4 = 0$  om  $s_4 < 2$  och  $x_4 \leq 1$  om  $s_4 < 4$ .

Om  $x_4 = 0$  fås  $f_3(s_4)$ .

Om  $x_4 = 1$  fås  $5 + f_3(s_4 - 2)$  (flytta 2 steg åt höger och addera 5).

Om  $x_4 = 2$  fås  $10 + f_3(s_4 - 4)$  (flytta 4 steg åt höger och addera 10).



## Dynamisk programmering: Kappsäcksproblem: Exempel

Steg 4 ( $x_4$ ):  $0 \leq s_4 \leq 9$ ,  $x_4 \in \{0, 1, 2\}$ .

$$s_3 = s_4 - a_4 x_4 = s_4 - 2x_4.$$

$$f_4(s_4) = \max_{x_4} (c_4 x_4 + f_3(s_3)) = \max_{x_4} (5x_4 + f_3(s_4 - 2x_4))$$

då  $x_4 \leq \lfloor s_4/a_4 \rfloor = \lfloor s_4/2 \rfloor$  dvs.  $x_4 = 0$  om  $s_4 < 2$  och  $x_4 \leq 1$  om  $s_4 < 4$ .

Om  $x_4 = 0$  fås  $f_3(s_4)$ .

Om  $x_4 = 1$  fås  $5 + f_3(s_4 - 2)$  (flytta 2 steg åt höger och addera 5).

Om  $x_4 = 2$  fås  $10 + f_3(s_4 - 4)$  (flytta 4 steg åt höger och addera 10).

$x_4 \backslash s_4$	0	1	2	3	4	5	6	7	8	9
0	0	0	3	4	6	7	8	10	11	11
1	-	-	5	5	8	9	11	12	13	15
2	-	-	-	-	10	10	13	14	16	17
$f_4(s_4)$	0	0	5	5	10	10	13	14	16	17
$\hat{x}_4(s_4)$	0	0	1	1	2	2	2	2	2	2

## Dynamisk programmering: Kappsäcksproblem: Exempel

Steg 5 ( $x_5$ ):  $0 \leq s_5 \leq 9$ ,  $x_5 \in \{0, 1, 2\}$ .

## Dynamisk programmering: Kappsäcksproblem: Exempel

Steg 5 ( $x_5$ ):  $0 \leq s_5 \leq 9$ ,  $x_5 \in \{0, 1, 2\}$ .

$$s_4 = s_5 - a_5 x_5 = s_5 - 3x_5.$$

## Dynamisk programmering: Kappsäcksproblem: Exempel

Steg 5 ( $x_5$ ):  $0 \leq s_5 \leq 9$ ,  $x_5 \in \{0, 1, 2\}$ .

$$s_4 = s_5 - a_5 x_5 = s_5 - 3x_5.$$

$$f_5(s_5) = \max_{x_5} (c_5 x_5 + f_4(s_4)) = \max_{x_5} (6x_5 + f_4(s_5 - 3x_5))$$

då  $x_5 \leq \lfloor s_5/a_5 \rfloor = \lfloor s_5/3 \rfloor$  dvs.  $x_5 = 0$  om  $s_5 < 3$  och  $x_5 \leq 1$  om  $s_5 < 6$ .

## Dynamisk programmering: Kappsäcksproblem: Exempel

Steg 5 ( $x_5$ ):  $0 \leq s_5 \leq 9$ ,  $x_5 \in \{0, 1, 2\}$ .

$$s_4 = s_5 - a_5 x_5 = s_5 - 3x_5.$$

$$f_5(s_5) = \max_{x_5} (c_5 x_5 + f_4(s_4)) = \max_{x_5} (6x_5 + f_4(s_5 - 3x_5))$$

då  $x_5 \leq \lfloor s_5/a_5 \rfloor = \lfloor s_5/3 \rfloor$  dvs.  $x_5 = 0$  om  $s_5 < 3$  och  $x_5 \leq 1$  om  $s_5 < 6$ .

Om  $x_5 = 0$  fås  $f_4(s_5)$ .

## Dynamisk programmering: Kappsäcksproblem: Exempel

Steg 5 ( $x_5$ ):  $0 \leq s_5 \leq 9$ ,  $x_5 \in \{0, 1, 2\}$ .

$$s_4 = s_5 - a_5 x_5 = s_5 - 3x_5.$$

$$f_5(s_5) = \max_{x_5} (c_5 x_5 + f_4(s_4)) = \max_{x_5} (6x_5 + f_4(s_5 - 3x_5))$$

då  $x_5 \leq \lfloor s_5/a_5 \rfloor = \lfloor s_5/3 \rfloor$  dvs.  $x_5 = 0$  om  $s_5 < 3$  och  $x_5 \leq 1$  om  $s_5 < 6$ .

Om  $x_5 = 0$  fås  $f_4(s_5)$ .

Om  $x_5 = 1$  fås  $6 + f_1(s_2 - 3)$  (flytta 3 steg åt höger och addera 6).

## Dynamisk programmering: Kappsäcksproblem: Exempel

Steg 5 ( $x_5$ ):  $0 \leq s_5 \leq 9$ ,  $x_5 \in \{0, 1, 2\}$ .

$$s_4 = s_5 - a_5 x_5 = s_5 - 3x_5.$$

$$f_5(s_5) = \max_{x_5} (c_5 x_5 + f_4(s_4)) = \max_{x_5} (6x_5 + f_4(s_5 - 3x_5))$$

då  $x_5 \leq \lfloor s_5/a_5 \rfloor = \lfloor s_5/3 \rfloor$  dvs.  $x_5 = 0$  om  $s_5 < 3$  och  $x_5 \leq 1$  om  $s_5 < 6$ .

Om  $x_5 = 0$  fås  $f_4(s_5)$ .

Om  $x_5 = 1$  fås  $6 + f_1(s_2 - 3)$  (flytta 3 steg åt höger och addera 6).

Om  $x_5 = 2$  fås  $12 + f_1(s_2 - 6)$  (flytta 6 steg åt höger och addera 12).

## Dynamisk programmering: Kappsäcksproblem: Exempel

Steg 5 ( $x_5$ ):  $0 \leq s_5 \leq 9$ ,  $x_5 \in \{0, 1, 2\}$ .

$$s_4 = s_5 - a_5 x_5 = s_5 - 3x_5.$$

$$f_5(s_5) = \max_{x_5} (c_5 x_5 + f_4(s_4)) = \max_{x_5} (6x_5 + f_4(s_5 - 3x_5))$$

då  $x_5 \leq \lfloor s_5/a_5 \rfloor = \lfloor s_5/3 \rfloor$  dvs.  $x_5 = 0$  om  $s_5 < 3$  och  $x_5 \leq 1$  om  $s_5 < 6$ .

Om  $x_5 = 0$  fås  $f_4(s_5)$ .

Om  $x_5 = 1$  fås  $6 + f_1(s_2 - 3)$  (flytta 3 steg åt höger och addera 6).

Om  $x_5 = 2$  fås  $12 + f_1(s_2 - 6)$  (flytta 6 steg åt höger och addera 12).

$x_5 \backslash s_5$	0	1	2	3	4	5	6	7	8	9
0	0	0	5	5	10	10	13	14	16	17
1	-	-	-	6	6	11	11	16	16	19
2	-	-	-	-	-	-	12	12	17	17
$f_5(s_5)$	0	0	5	6	10	11	13	16	17	19
$\hat{x}_5(s_5)$	0	0	0	1	0	1	0	1	2	1



# Dynamisk programmering: Kappsäcksproblem: Exempel

Steg 6 ( $x_6$ ):  $s_6 = 9$ ,  $x_6 \in \{0, 1, 2\}$ .

## Dynamisk programmering: Kappsäcksproblem: Exempel

Steg 6 ( $x_6$ ):  $s_6 = 9$ ,  $x_6 \in \{0, 1, 2\}$ .

$$s_5 = s_6 - a_6 x_6 = s_6 - 4x_2.$$

## Dynamisk programmering: Kappsäcksproblem: Exempel

Steg 6 ( $x_6$ ):  $s_6 = 9$ ,  $x_6 \in \{0, 1, 2\}$ .

$$s_5 = s_6 - a_6 x_6 = s_6 - 4x_6.$$

$$f_6(s_6) = \max_{x_6} (c_6 x_6 + f_5(s_5)) = \max_{x_6} (5x_6 + f_5(s_6 - 4x_6))$$

då  $x_6 \leq \lfloor s_6/a_6 \rfloor = \lfloor s_6/4 \rfloor$  dvs.  $x_6 = 0$  om  $s_6 < 4$  och  $x_6 \leq 1$  om  $s_6 < 8$ .

## Dynamisk programmering: Kappsäcksproblem: Exempel

Steg 6 ( $x_6$ ):  $s_6 = 9$ ,  $x_6 \in \{0, 1, 2\}$ .

$$s_5 = s_6 - a_6 x_6 = s_6 - 4x_6.$$

$$f_6(s_6) = \max_{x_6} (c_6 x_6 + f_5(s_5)) = \max_{x_6} (5x_6 + f_5(s_6 - 4x_6))$$

då  $x_6 \leq \lfloor s_6/a_6 \rfloor = \lfloor s_6/4 \rfloor$  dvs.  $x_6 = 0$  om  $s_6 < 4$  och  $x_6 \leq 1$  om  $s_6 < 8$ .

Om  $x_6 = 0$  fås  $f_5(s_6)$ .

## Dynamisk programmering: Kappsäcksproblem: Exempel

Steg 6 ( $x_6$ ):  $s_6 = 9$ ,  $x_6 \in \{0, 1, 2\}$ .

$$s_5 = s_6 - a_6 x_6 = s_6 - 4x_6.$$

$$f_6(s_6) = \max_{x_6} (c_6 x_6 + f_5(s_5)) = \max_{x_6} (5x_6 + f_5(s_6 - 4x_6))$$

då  $x_6 \leq \lfloor s_6/a_6 \rfloor = \lfloor s_6/4 \rfloor$  dvs.  $x_6 = 0$  om  $s_6 < 4$  och  $x_6 \leq 1$  om  $s_6 < 8$ .

Om  $x_6 = 0$  fås  $f_5(s_6)$ .

Om  $x_6 = 1$  fås  $5 + f_5(s_6 - 4)$  (flytta 4 steg åt höger och addera 5).

## Dynamisk programmering: Kappsäcksproblem: Exempel

Steg 6 ( $x_6$ ):  $s_6 = 9$ ,  $x_6 \in \{0, 1, 2\}$ .

$$s_5 = s_6 - a_6 x_6 = s_6 - 4x_6.$$

$$f_6(s_6) = \max_{x_6} (c_6 x_6 + f_5(s_5)) = \max_{x_6} (5x_6 + f_5(s_6 - 4x_6))$$

då  $x_6 \leq \lfloor s_6/a_6 \rfloor = \lfloor s_6/4 \rfloor$  dvs.  $x_6 = 0$  om  $s_6 < 4$  och  $x_6 \leq 1$  om  $s_6 < 8$ .

Om  $x_6 = 0$  fås  $f_5(s_6)$ .

Om  $x_6 = 1$  fås  $5 + f_5(s_6 - 4)$  (flytta 4 steg åt höger och addera 5).

Om  $x_6 = 2$  fås  $10 + f_5(s_6 - 8)$  (flytta 8 steg åt höger och addera 10).

## Dynamisk programmering: Kappsäcksproblem: Exempel

Steg 6 ( $x_6$ ):  $s_6 = 9$ ,  $x_6 \in \{0, 1, 2\}$ .

$$s_5 = s_6 - a_6 x_6 = s_6 - 4x_6.$$

$$f_6(s_6) = \max_{x_6} (c_6 x_6 + f_5(s_5)) = \max_{x_6} (5x_6 + f_5(s_6 - 4x_6))$$

då  $x_6 \leq \lfloor s_6/a_6 \rfloor = \lfloor s_6/4 \rfloor$  dvs.  $x_6 = 0$  om  $s_6 < 4$  och  $x_6 \leq 1$  om  $s_6 < 8$ .

Om  $x_6 = 0$  fås  $f_5(s_6)$ .

Om  $x_6 = 1$  fås  $5 + f_5(s_6 - 4)$  (flytta 4 steg åt höger och addera 5).

Om  $x_6 = 2$  fås  $10 + f_5(s_6 - 8)$  (flytta 8 steg åt höger och addera 10).

$x_6 \backslash s_6$	0	1	2	3	4	5	6	7	8	9
0	0	0	5	6	10	11	13	16	17	19
1	-	-	-	-	5	5	10	11	15	16
2	-	-	-	-	-	-	-	-	10	10
$f_6(s_6)$	0	0	5	6	10	11	13	16	17	19
$\hat{x}_6(s_6)$	0	0	0	0	0	0	0	0	0	0

## Dynamisk programmering: Kappsäcksproblem: Exempel

Steg 6 ( $x_6$ ):  $s_6 = 9$ ,  $x_6 \in \{0, 1, 2\}$ .

$$s_5 = s_6 - a_6 x_6 = s_6 - 4x_6.$$

$$f_6(s_6) = \max_{x_6} (c_6 x_6 + f_5(s_5)) = \max_{x_6} (5x_6 + f_5(s_6 - 4x_6))$$

då  $x_6 \leq \lfloor s_6/a_6 \rfloor = \lfloor s_6/4 \rfloor$  dvs.  $x_6 = 0$  om  $s_6 < 4$  och  $x_6 \leq 1$  om  $s_6 < 8$ .

Om  $x_6 = 0$  fås  $f_5(s_6)$ .

Om  $x_6 = 1$  fås  $5 + f_5(s_6 - 4)$  (flytta 4 steg åt höger och addera 5).

Om  $x_6 = 2$  fås  $10 + f_5(s_6 - 8)$  (flytta 8 steg åt höger och addera 10).

$x_6 \backslash s_6$	9
0	19
1	16
2	10
$f_6(s_6)$	19
$\hat{x}_6(s_6)$	0



# Dynamisk programmering: Kappsäcksproblem: Exempel

Nysta upp lösningen.

# Dynamisk programmering: Kappsäcksproblem: Exempel

Nysta upp lösningen.

Nu behövs  $\hat{x}_k(s_k)$  och överföringsfunktionen  $s_{k-1} = s_k - a_k x_k$ .

# Dynamisk programmering: Kappsäcksproblem: Exempel

Nysta upp lösningen.

Nu behövs  $\hat{x}_k(s_k)$  och överföringsfunktionen  $s_{k-1} = s_k - a_k x_k$ .

$$2x_1 + 3x_2 + 5x_3 + 2x_4 + 3x_5 + 4x_6 \leq 9$$

## Dynamisk programmering: Kappsäcksproblem: Exempel

Nysta upp lösningen.

Nu behövs  $\hat{x}_k(s_k)$  och överföringsfunktionen  $s_{k-1} = s_k - a_k x_k$ .

$$2x_1 + 3x_2 + 5x_3 + 2x_4 + 3x_5 + 4x_6 \leq 9$$

Börja bakifrån:

# Dynamisk programmering: Kappsäcksproblem: Exempel

Nysta upp lösningen.

Nu behövs  $\hat{x}_k(s_k)$  och överföringsfunktionen  $s_{k-1} = s_k - a_k x_k$ .

$$2x_1 + 3x_2 + 5x_3 + 2x_4 + 3x_5 + 4x_6 \leq 9$$

Börja bakifrån:

$x_6 \backslash s_6$	0	1	2	3	4	5	6	7	8	9
0	0	0	5	6	10	11	13	16	17	19
1	-	-	-	-	5	5	10	11	15	16
2	-	-	-	-	-	-	-	-	10	10
$f_6(s_6)$	0	0	5	6	10	11	13	16	17	19
$\hat{x}_6(s_6)$	0	0	0	0	0	0	0	0	0	0

$$s_6 = 9$$

# Dynamisk programmering: Kappsäcksproblem: Exempel

Nysta upp lösningen.

Nu behövs  $\hat{x}_k(s_k)$  och överföringsfunktionen  $s_{k-1} = s_k - a_k x_k$ .

$$2x_1 + 3x_2 + 5x_3 + 2x_4 + 3x_5 + 4x_6 \leq 9$$

Börja bakifrån:

$x_6 \backslash s_6$	0	1	2	3	4	5	6	7	8	9
0	0	0	5	6	10	11	13	16	17	19
1	-	-	-	-	5	5	10	11	15	16
2	-	-	-	-	-	-	-	-	10	10
$f_6(s_6)$	0	0	5	6	10	11	13	16	17	19
$\hat{x}_6(s_6)$	0	0	0	0	0	0	0	0	0	0

$$s_6 = 9 \Rightarrow \hat{x}_6(9) = 0$$

# Dynamisk programmering: Kappsäcksproblem: Exempel

Nysta upp lösningen.

Nu behövs  $\hat{x}_k(s_k)$  och överföringsfunktionen  $s_{k-1} = s_k - a_k x_k$ .

$$2x_1 + 3x_2 + 5x_3 + 2x_4 + 3x_5 + 4x_6 \leq 9$$

Börja bakifrån:

$x_5 \backslash s_5$	0	1	2	3	4	5	6	7	8	9
0	0	0	5	5	10	10	13	14	16	17
1	-	-	-	6	6	11	11	16	16	19
2	-	-	-	-	-	-	12	12	17	17
$f_5(s_5)$	0	0	5	6	10	11	13	16	17	19
$\hat{x}_5(s_5)$	0	0	0	1	0	1	0	1	2	1

$$s_6 = 9 \Rightarrow \hat{x}_6(9) = 0 \Rightarrow s_5 = s_6 = 9$$

# Dynamisk programmering: Kappsäcksproblem: Exempel

Nysta upp lösningen.

Nu behövs  $\hat{x}_k(s_k)$  och överföringsfunktionen  $s_{k-1} = s_k - a_k x_k$ .

$$2x_1 + 3x_2 + 5x_3 + 2x_4 + 3x_5 + 4x_6 \leq 9$$

Börja bakifrån:

$x_5 \backslash s_5$	0	1	2	3	4	5	6	7	8	9
0	0	0	5	5	10	10	13	14	16	17
1	-	-	-	6	6	11	11	16	16	19
2	-	-	-	-	-	-	12	12	17	17
$f_5(s_5)$	0	0	5	6	10	11	13	16	17	19
$\hat{x}_5(s_5)$	0	0	0	1	0	1	0	1	2	1

$$s_6 = 9 \Rightarrow \hat{x}_6(9) = 0 \Rightarrow s_5 = s_6 = 9 \Rightarrow \hat{x}_5(9) = 1$$



# Dynamisk programmering: Kappsäcksproblem: Exempel

Nysta upp lösningen.

Nu behövs  $\hat{x}_k(s_k)$  och överföringsfunktionen  $s_{k-1} = s_k - a_k x_k$ .

$$2x_1 + 3x_2 + 5x_3 + 2x_4 + 3x_5 + 4x_6 \leq 9$$

Börja bakifrån:

$x_4 \backslash s_4$	0	1	2	3	4	5	6	7	8	9
0	0	0	3	4	6	7	8	10	11	11
1	-	-	5	5	8	9	11	12	13	15
2	-	-	-	-	10	10	13	14	16	17
$f_4(s_4)$	0	0	5	5	10	10	13	14	16	17
$\hat{x}_4(s_4)$	0	0	1	1	2	2	2	2	2	2

$$s_6 = 9 \Rightarrow \hat{x}_6(9) = 0 \Rightarrow s_5 = s_6 = 9 \Rightarrow \hat{x}_5(9) = 1 \Rightarrow s_4 = s_5 - 3 = 6$$

# Dynamisk programmering: Kappsäcksproblem: Exempel

Nysta upp lösningen.

Nu behövs  $\hat{x}_k(s_k)$  och överföringsfunktionen  $s_{k-1} = s_k - a_k x_k$ .

$$2x_1 + 3x_2 + 5x_3 + 2x_4 + 3x_5 + 4x_6 \leq 9$$

Börja bakifrån:

$x_4 \backslash s_4$	0	1	2	3	4	5	6	7	8	9
0	0	0	3	4	6	7	8	10	11	11
1	-	-	5	5	8	9	11	12	13	15
2	-	-	-	-	10	10	13	14	16	17
$f_4(s_4)$	0	0	5	5	10	10	13	14	16	17
$\hat{x}_4(s_4)$	0	0	1	1	2	2	2	2	2	2

$$\begin{aligned} s_6 = 9 &\Rightarrow \hat{x}_6(9) = 0 \Rightarrow s_5 = s_6 = 9 \Rightarrow \hat{x}_5(9) = 1 \Rightarrow s_4 = s_5 - 3 = 6 \\ &\Rightarrow \hat{x}_4(6) = 2 \end{aligned}$$

# Dynamisk programmering: Kappsäcksproblem: Exempel

Nysta upp lösningen.

Nu behövs  $\hat{x}_k(s_k)$  och överföringsfunktionen  $s_{k-1} = s_k - a_k x_k$ .

$$2x_1 + 3x_2 + 5x_3 + 2x_4 + 3x_5 + 4x_6 \leq 9$$

Börja bakifrån:

$x_3 \backslash s_3$	0	1	2	3	4	5	6	7	8	9
0	0	0	3	4	6	7	8	10	11	11
1	-	-	-	-	-	2	2	5	6	8
2	-	-	-	-	-	-	-	-	-	-
$f_3(s_3)$	0	0	3	4	6	7	8	10	11	11
$\hat{x}_3(s_3)$	0	0	0	0	0	0	0	0	0	0

$$\begin{aligned} s_6 = 9 &\Rightarrow \hat{x}_6(9) = 0 \Rightarrow s_5 = s_6 = 9 \Rightarrow \hat{x}_5(9) = 1 \Rightarrow s_4 = s_5 - 3 = 6 \\ &\Rightarrow \hat{x}_4(6) = 2 \Rightarrow s_3 = s_4 - 2 * 2 = 2 \end{aligned}$$

# Dynamisk programmering: Kappsäcksproblem: Exempel

Nysta upp lösningen.

Nu behövs  $\hat{x}_k(s_k)$  och överföringsfunktionen  $s_{k-1} = s_k - a_k x_k$ .

$$2x_1 + 3x_2 + 5x_3 + 2x_4 + 3x_5 + 4x_6 \leq 9$$

Börja bakifrån:

$x_3 \backslash s_3$	0	1	2	3	4	5	6	7	8	9
0	0	0	3	4	6	7	8	10	11	11
1	-	-	-	-	-	2	2	5	6	8
2	-	-	-	-	-	-	-	-	-	-
$f_3(s_3)$	0	0	3	4	6	7	8	10	11	11
$\hat{x}_3(s_3)$	0	0	0	0	0	0	0	0	0	0

$$\begin{aligned} s_6 = 9 &\Rightarrow \hat{x}_6(9) = 0 \Rightarrow s_5 = s_6 = 9 \Rightarrow \hat{x}_5(9) = 1 \Rightarrow s_4 = s_5 - 3 = 6 \\ &\Rightarrow \hat{x}_4(6) = 2 \Rightarrow s_3 = s_4 - 2 * 2 = 2 \Rightarrow \hat{x}_3(2) = 0 \end{aligned}$$

# Dynamisk programmering: Kappsäcksproblem: Exempel

Nysta upp lösningen.

Nu behövs  $\hat{x}_k(s_k)$  och överföringsfunktionen  $s_{k-1} = s_k - a_k x_k$ .

$$2x_1 + 3x_2 + 5x_3 + 2x_4 + 3x_5 + 4x_6 \leq 9$$

Börja bakifrån:

$x_2 \backslash s_2$	0	1	2	3	4	5	6	7	8	9
0	0	0	3	3	6	6	6	6	6	6
1	-	-	-	4	4	7	7	10	10	10
2	-	-	-	-	-	-	8	8	11	11
$f_2(s_2)$	0	0	3	4	6	7	8	10	11	11
$\hat{x}_2(s_2)$	0	0	0	1	0	1	2	1	2	2

$$\begin{aligned} s_6 = 9 &\Rightarrow \hat{x}_6(9) = 0 \Rightarrow s_5 = s_6 = 9 \Rightarrow \hat{x}_5(9) = 1 \Rightarrow s_4 = s_5 - 3 = 6 \\ &\Rightarrow \hat{x}_4(6) = 2 \Rightarrow s_3 = s_4 - 2 * 2 = 2 \Rightarrow \hat{x}_3(2) = 0 \Rightarrow s_2 = s_3 = 2 \end{aligned}$$

# Dynamisk programmering: Kappsäcksproblem: Exempel

Nysta upp lösningen.

Nu behövs  $\hat{x}_k(s_k)$  och överföringsfunktionen  $s_{k-1} = s_k - a_k x_k$ .

$$2x_1 + 3x_2 + 5x_3 + 2x_4 + 3x_5 + 4x_6 \leq 9$$

Börja bakifrån:

$x_2 \backslash s_2$	0	1	2	3	4	5	6	7	8	9
0	0	0	3	3	6	6	6	6	6	6
1	-	-	-	4	4	7	7	10	10	10
2	-	-	-	-	-	-	8	8	11	11
$f_2(s_2)$	0	0	3	4	6	7	8	10	11	11
$\hat{x}_2(s_2)$	0	0	0	1	0	1	2	1	2	2

$$\begin{aligned} s_6 = 9 &\Rightarrow \hat{x}_6(9) = 0 \Rightarrow s_5 = s_6 = 9 \Rightarrow \hat{x}_5(9) = 1 \Rightarrow s_4 = s_5 - 3 = 6 \\ &\Rightarrow \hat{x}_4(6) = 2 \Rightarrow s_3 = s_4 - 2 * 2 = 2 \Rightarrow \hat{x}_3(2) = 0 \Rightarrow s_2 = s_3 = 2 \\ &\Rightarrow \hat{x}_2(2) = 0 \end{aligned}$$

# Dynamisk programmering: Kappsäcksproblem: Exempel

Nysta upp lösningen.

Nu behövs  $\hat{x}_k(s_k)$  och överföringsfunktionen  $s_{k-1} = s_k - a_k x_k$ .

$$2x_1 + 3x_2 + 5x_3 + 2x_4 + 3x_5 + 4x_6 \leq 9$$

Börja bakifrån:

$x_1 \backslash s_1$	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	-	-	3	3	3	3	3	3	3	3
2	-	-	-	-	6	6	6	6	6	6
$f_1(s_1)$	0	0	3	3	6	6	6	6	6	6
$\hat{x}_1(s_1)$	0	0	1	1	2	2	2	2	2	2

$$\begin{aligned} s_6 = 9 &\Rightarrow \hat{x}_6(9) = 0 \Rightarrow s_5 = s_6 = 9 \Rightarrow \hat{x}_5(9) = 1 \Rightarrow s_4 = s_5 - 3 = 6 \\ &\Rightarrow \hat{x}_4(6) = 2 \Rightarrow s_3 = s_4 - 2 * 2 = 2 \Rightarrow \hat{x}_3(2) = 0 \Rightarrow s_2 = s_3 = 2 \\ &\Rightarrow \hat{x}_2(2) = 0 \Rightarrow s_1 = s_2 = 2 \end{aligned}$$

# Dynamisk programmering: Kappsäcksproblem: Exempel

Nysta upp lösningen.

Nu behövs  $\hat{x}_k(s_k)$  och överföringsfunktionen  $s_{k-1} = s_k - a_k x_k$ .

$$2x_1 + 3x_2 + 5x_3 + 2x_4 + 3x_5 + 4x_6 \leq 9$$

Börja bakifrån:

$x_1 \backslash s_1$	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	-	-	3	3	3	3	3	3	3	3
2	-	-	-	-	6	6	6	6	6	6
$f_1(s_1)$	0	0	3	3	6	6	6	6	6	6
$\hat{x}_1(s_1)$	0	0	1	1	2	2	2	2	2	2

$$\begin{aligned} s_6 = 9 &\Rightarrow \hat{x}_6(9) = 0 \Rightarrow s_5 = s_6 = 9 \Rightarrow \hat{x}_5(9) = 1 \Rightarrow s_4 = s_5 - 3 = 6 \\ &\Rightarrow \hat{x}_4(6) = 2 \Rightarrow s_3 = s_4 - 2 * 2 = 2 \Rightarrow \hat{x}_3(2) = 0 \Rightarrow s_2 = s_3 = 2 \\ &\Rightarrow \hat{x}_2(2) = 0 \Rightarrow s_1 = s_2 = 2 \Rightarrow \hat{x}_1(2) = 1 \end{aligned}$$



# Dynamisk programmering: Kappsäcksproblem: Exempel

Nysta upp lösningen.

Nu behövs  $\hat{x}_k(s_k)$  och överföringsfunktionen  $s_{k-1} = s_k - a_k x_k$ .

$$2x_1 + 3x_2 + 5x_3 + 2x_4 + 3x_5 + 4x_6 \leq 9$$

Börja bakifrån:

$x_1 \backslash s_1$	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	-	-	3	3	3	3	3	3	3	3
2	-	-	-	-	6	6	6	6	6	6
$f_1(s_1)$	0	0	3	3	6	6	6	6	6	6
$\hat{x}_1(s_1)$	0	0	1	1	2	2	2	2	2	2

$$\begin{aligned} s_6 = 9 &\Rightarrow \hat{x}_6(9) = 0 \Rightarrow s_5 = s_6 = 9 \Rightarrow \hat{x}_5(9) = 1 \Rightarrow s_4 = s_5 - 3 = 6 \\ &\Rightarrow \hat{x}_4(6) = 2 \Rightarrow s_3 = s_4 - 2 * 2 = 2 \Rightarrow \hat{x}_3(2) = 0 \Rightarrow s_2 = s_3 = 2 \\ &\Rightarrow \hat{x}_2(2) = 0 \Rightarrow s_1 = s_2 = 2 \Rightarrow \hat{x}_1(2) = 1 \Rightarrow s_0 = s_1 - 2 = 0 \end{aligned}$$

# Dynamisk programmering: Kappsäcksproblem: Exempel

Nysta upp lösningen.

Nu behövs  $\hat{x}_k(s_k)$  och överföringsfunktionen  $s_{k-1} = s_k - a_k x_k$ .

$$2x_1 + 3x_2 + 5x_3 + 2x_4 + 3x_5 + 4x_6 \leq 9$$

Börja bakifrån:

$x_1 \backslash s_1$	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	-	-	3	3	3	3	3	3	3	3
2	-	-	-	-	6	6	6	6	6	6
$f_1(s_1)$	0	0	3	3	6	6	6	6	6	6
$\hat{x}_1(s_1)$	0	0	1	1	2	2	2	2	2	2

$$s_6 = 9 \Rightarrow \hat{x}_6(9) = 0 \Rightarrow s_5 = s_6 = 9 \Rightarrow \hat{x}_5(9) = 1 \Rightarrow s_4 = s_5 - 3 = 6$$

$$\Rightarrow \hat{x}_4(6) = 2 \Rightarrow s_3 = s_4 - 2 * 2 = 2 \Rightarrow \hat{x}_3(2) = 0 \Rightarrow s_2 = s_3 = 2$$

$$\Rightarrow \hat{x}_2(2) = 0 \Rightarrow s_1 = s_2 = 2 \Rightarrow \hat{x}_1(2) = 1 \Rightarrow s_0 = s_1 - 2 = 0$$

Optimal lösning:  $x_1 = 1, x_2 = 0, x_3 = 0, x_4 = 2, x_5 = 1, x_6 = 0$ .

# Dynamisk programmering: Kappsäcksproblem: Exempel

Nysta upp lösningen.

Nu behövs  $\hat{x}_k(s_k)$  och överföringsfunktionen  $s_{k-1} = s_k - a_k x_k$ .

$$2x_1 + 3x_2 + 5x_3 + 2x_4 + 3x_5 + 4x_6 \leq 9$$

Börja bakifrån:

$x_1 \backslash s_1$	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	-	-	3	3	3	3	3	3	3	3
2	-	-	-	-	6	6	6	6	6	6
$f_1(s_1)$	0	0	3	3	6	6	6	6	6	6
$\hat{x}_1(s_1)$	0	0	1	1	2	2	2	2	2	2

$$s_6 = 9 \Rightarrow \hat{x}_6(9) = 0 \Rightarrow s_5 = s_6 = 9 \Rightarrow \hat{x}_5(9) = 1 \Rightarrow s_4 = s_5 - 3 = 6$$

$$\Rightarrow \hat{x}_4(6) = 2 \Rightarrow s_3 = s_4 - 2 * 2 = 2 \Rightarrow \hat{x}_3(2) = 0 \Rightarrow s_2 = s_3 = 2$$

$$\Rightarrow \hat{x}_2(2) = 0 \Rightarrow s_1 = s_2 = 2 \Rightarrow \hat{x}_1(2) = 1 \Rightarrow s_0 = s_1 - 2 = 0$$

Optimal lösning:  $x_1 = 1, x_2 = 0, x_3 = 0, x_4 = 2, x_5 = 1, x_6 = 0$ .

Optimala målfunktionsvärdet ges av  $f_6(9) = 19$ .

# Dynamisk programmering: Kappsäcksproblem: Exempel

Nysta upp lösningen.

Nu behövs  $\hat{x}_k(s_k)$  och överföringsfunktionen  $s_{k-1} = s_k - a_k x_k$ .

$$2x_1 + 3x_2 + 5x_3 + 2x_4 + 3x_5 + 4x_6 \leq 9$$

Börja bakifrån:

$x_1 \setminus s_1$	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	-	-	3	3	3	3	3	3	3	3
2	-	-	-	-	6	6	6	6	6	6
$f_1(s_1)$	0	0	3	3	6	6	6	6	6	6
$\hat{x}_1(s_1)$	0	0	1	1	2	2	2	2	2	2

$$s_6 = 9 \Rightarrow \hat{x}_6(9) = 0 \Rightarrow s_5 = s_6 = 9 \Rightarrow \hat{x}_5(9) = 1 \Rightarrow s_4 = s_5 - 3 = 6$$

$$\Rightarrow \hat{x}_4(6) = 2 \Rightarrow s_3 = s_4 - 2 * 2 = 2 \Rightarrow \hat{x}_3(2) = 0 \Rightarrow s_2 = s_3 = 2$$

$$\Rightarrow \hat{x}_2(2) = 0 \Rightarrow s_1 = s_2 = 2 \Rightarrow \hat{x}_1(2) = 1 \Rightarrow s_0 = s_1 - 2 = 0$$

Optimal lösning:  $x_1 = 1, x_2 = 0, x_3 = 0, x_4 = 2, x_5 = 1, x_6 = 0$ .

Optimala målfunktionsvärdet ges av  $f_6(9) = 19$ .

Inget slack i bivillkoret, ty  $s_0 = 0$ .

# Dynamisk programmering: Kappsäcksproblem

Detta angreppssätt kan användas på många typer av “kappsäckar”.

# Dynamisk programmering: Kappsäcksproblem

Detta angreppssätt kan användas på många typer av “kappsäckar”.

Dvs. där man har ett “lager” som fylls på eller töms vid olika tidpunkter.

# Dynamisk programmering: Kappsäcksproblem

Detta angreppssätt kan användas på många typer av “kappsäckar”.

Dvs. där man har ett “lager” som fylls på eller töms vid olika tidpunkter.

Även på kontinuerliga “kappsäckar”, om man diskretiserar.

# Dynamisk programmering: Kappsäcksproblem

Detta angreppssätt kan användas på många typer av “kappsäckar”.

Dvs. där man har ett “lager” som fylls på eller töms vid olika tidpunkter.

Även på kontinuerliga “kappsäckar”, om man diskretiserar.

Man inför då ett tillstånd för varje möjlig nivå i kappsäcken/lagret.



# Dynamisk programmering: Kappsäcksproblem

Detta angreppssätt kan användas på många typer av “kappsäckar”.

Dvs. där man har ett “lager” som fylls på eller töms vid olika tidpunkter.

Även på kontinuerliga “kappsäckar”, om man diskretiserar.

Man inför då ett tillstånd för varje möjlig nivå i kappsäcken/lagret.

Och bångar, framåt i tiden, för varje möjlig ändring.

# Dynamisk programmering: Variationer

*Flera tillståndsvariabler.*

## Dynamisk programmering: Variationer

*Flera tillståndsvariabler. T.ex. flera kappsäcksvillkor.*

## Dynamisk programmering: Variationer

*Flera tillståndsvariabler.* T.ex. flera kappsäcksvillkor.

Måste undersöka samtliga kombinationer av tillstånden.

## Dynamisk programmering: Variationer

*Flera tillståndsvariabler.* T.ex. flera kappsäcksvillkor.

Måste undersöka samtliga kombinationer av tillstånden.

*Flera styrvariabler.*

## Dynamisk programmering: Variationer

*Flera tillståndsvariabler.* T.ex. flera kappsäcksvillkor.

Måste undersöka samtliga kombinationer av tillstånden.

*Flera styrvariabler.* Även här måste samtliga kombinationer av de möjliga styrningarna undersökas.

## Dynamisk programmering: Variationer

*Flera tillståndsvariabler.* T.ex. flera kappsäcksvillkor.

Måste undersöka samtliga kombinationer av tillstånden.

*Flera styrvariabler.* Även här måste samtliga kombinationer av de möjliga styrningarna undersökas.

Båda jobbigare än att öka antalet steg.

## Dynamisk programmering: Variationer

*Flera tillståndsvariabler.* T.ex. flera kappsäcksvillkor.

Måste undersöka samtliga kombinationer av tillstånden.

*Flera styrvariabler.* Även här måste samtliga kombinationer av de möjliga styrningarna undersökas.

Båda jobbigare än att öka antalet steg.

Tillståndsvariabeln måste vara diskret, men styrvariabeln kan vara kontinuerlig.



## Dynamisk programmering: Variationer

*Flera tillståndsvariabler.* T.ex. flera kappsäcksvillkor.

Måste undersöka samtliga kombinationer av tillstånden.

*Flera styrvariabler.* Även här måste samtliga kombinationer av de möjliga styrningarna undersökas.

Båda jobbigare än att öka antalet steg.

Tillståndsvariabeln måste vara diskret, men styrvariabeln kan vara kontinuerlig.

Ibland kan optimeringen göras analytiskt i ett steg.

## Dynamisk programmering: Variationer

*Flera tillståndsvariabler.* T.ex. flera kappsäcksvillkor.

Måste undersöka samtliga kombinationer av tillstånden.

*Flera styrvariabler.* Även här måste samtliga kombinationer av de möjliga styrningarna undersökas.

Båda jobbigare än att öka antalet steg.

Tillståndsvariabeln måste vara diskret, men styrvariabeln kan vara kontinuerlig.

Ibland kan optimeringen göras analytiskt i ett steg.

Strukturen i Dynamisk programmering beror inte på hur delproblemen löses för ett steg.

## Dynamisk programmering: Slump

Ibland hamnar man inte alltid i det tillstånd man vill komma till, utan kan med viss sannolikhet hamna i ett annat.

## Dynamisk programmering: Slump

Ibland hamnar man inte alltid i det tillstånd man vill komma till, utan kan med viss sannolikhet hamna i ett annat.

Exempelvis kanske försäljningen inte blir exakt vad man förväntade sig.

## Dynamisk programmering: Slump

Ibland hamnar man inte alltid i det tillstånd man vill komma till, utan kan med viss sannolikhet hamna i ett annat.

Exempelvis kanske försäljningen inte blir exakt vad man förväntade sig.

Detta kan hanteras av **stokastisk** dynamisk programmering.

## Dynamisk programmering: Slump

Ibland hamnar man inte alltid i det tillstånd man vill komma till, utan kan med viss sannolikhet hamna i ett annat.

Exempelvis kanske försäljningen inte blir exakt vad man förväntade sig.

Detta kan hanteras av **stokastisk** dynamisk programmering.

Man måste då använda bakåtrekursion.

## Dynamisk programmering: Slump

Ibland hamnar man inte alltid i det tillstånd man vill komma till, utan kan med viss sannolikhet hamna i ett annat.

Exempelvis kanske försäljningen inte blir exakt vad man förväntade sig.

Detta kan hanteras av **stokastisk** dynamisk programmering.

Man måste då använda bakåtrekursion.

I framåtrekursion står man i ett visst tillstånd, och vill finna bästa vägen dit (från förra steget).

## Dynamisk programmering: Slump

Ibland hamnar man inte alltid i det tillstånd man vill komma till, utan kan med viss sannolikhet hamna i ett annat.

Exempelvis kanske försäljningen inte blir exakt vad man förväntade sig.

Detta kan hanteras av **stokastisk** dynamisk programmering.

Man måste då använda bakåtrekursion.

I framåtrekursion står man i ett visst tillstånd, och vill finna bästa vägen dit (från förra steget).

I bakåtrekursion står man i ett visst tillstånd, och vill finna bästa vägen därifrån (till nästa steg).



## Dynamisk programmering: Slump

Ibland hamnar man inte alltid i det tillstånd man vill komma till, utan kan med viss sannolikhet hamna i ett annat.

Exempelvis kanske försäljningen inte blir exakt vad man förväntade sig.

Detta kan hanteras av **stokastisk** dynamisk programmering.

Man måste då använda bakåtrekursion.

I framåtrekursion står man i ett visst tillstånd, och vill finna bästa vägen dit (från förra steget).

I bakåtrekursion står man i ett visst tillstånd, och vill finna bästa vägen därifrån (till nästa steg).

Om man vet vilket tillstånd man befinner sig i, och vet vart man vill gå, kan man räkna ut sannolikheten att man hamnar där man vill, eller i något annat tillstånd i nästa steg.

## Dynamisk programmering: Slump

Ibland hamnar man inte alltid i det tillstånd man vill komma till, utan kan med viss sannolikhet hamna i ett annat.

Exempelvis kanske försäljningen inte blir exakt vad man förväntade sig.

Detta kan hanteras av **stokastisk** dynamisk programmering.

Man måste då använda bakåtrekursion.

I framåtrekursion står man i ett visst tillstånd, och vill finna bästa vägen dit (från förra steget).

I bakåtrekursion står man i ett visst tillstånd, och vill finna bästa vägen därifrån (till nästa steg).

Om man vet vilket tillstånd man befinner sig i, och vet vart man vill gå, kan man räkna ut sannolikheten att man hamnar där man vill, eller i något annat tillstånd i nästa steg.

(Detta fungerar inte åt andra hållet.)

# Dynamisk programmering: Framåtrekursion

I steg  $k$ :

Noden i  $S_k$  kallas **tillstånd**,  $s_k$ .

Noden man kommer från kallas **styrning**,  $x_k$ .

**Överföringsfunktion**: kopplingen mellan olika steg,  $s_{k-1} = T_k(x_k, s_k)$ .

**Målfunktionsvärde**,  $f_k(s_k)$ . (Kostnaden från startnod till  $s_k$ .)

**Rekursiv målfunktion**:  $f_k(s_k) = \min_{x_k} (c_k(x_k, s_k) + f_{k-1}(s_{k-1}))$

**Begränsningar** på tillstånd,  $s_k \in S_k$ , och styrning,  $x_k \in X_k$ .

**Randvillkor** på  $s_0$ ,  $s_N$ , samt  $f_0(s_0)$ .

# Dynamisk programmering: Bakåtrekursion

I steg  $k$ :

Noden i  $S_k$  kallas **tillstånd**,  $s_k$ .

Noden man går till kallas **styrning**,  $x_k$ .

**Överföringsfunktion**: kopplingen mellan olika steg,  $s_{k+1} = T_k(x_k, s_k)$ .

**Målfunktionsvärde**,  $f_k(s_k)$ . (Kostnaden från  $s_k$  till slutnod.)

**Rekursiv målfunktion**:  $f_k(s_k) = \min_{x_k} (c_k(x_k, s_k) + f_{k+1}(s_{k+1}))$

**Begränsningar** på tillstånd,  $s_k \in S_k$ , och styrning,  $x_k \in X_k$ .

**Randvillkor** på  $s_0$ ,  $s_N$ , samt  $f_N(s_N)$ .

# Dynamisk programmering: Stokastisk bakåtrekursion

Vi står i nod  $i \in S_k$

## Dynamisk programmering: Stokastisk bakåtrekursion

Vi står i nod  $i \in S_k$  och siktar mot nod  $j \in S_{k+1}$

## Dynamisk programmering: Stokastisk bakåtrekursion

Vi står i nod  $i \in S_k$  och siktar mot nod  $j \in S_{k+1}$

dvs. vill använda båge  $(i, j)$ .

## Dynamisk programmering: Stokastisk bakåtrekursion

Vi står i nod  $i \in S_k$  och siktar mot nod  $j \in S_{k+1}$

dvs. vill använda båge  $(i, j)$ .

Var hamnar vi?



## Dynamisk programmering: Stokastisk bakåtrekursion

Vi står i nod  $i \in S_k$  och siktar mot nod  $j \in S_{k+1}$

dvs. vill använda båge  $(i, j)$ .

Var hamnar vi?

I nod  $j$  med sannolikhet  $p_{ij}$

## Dynamisk programmering: Stokastisk bakåtrekursion

Vi står i nod  $i \in S_k$  och siktar mot nod  $j \in S_{k+1}$

dvs. vill använda båge  $(i, j)$ .

Var hamnar vi?

I nod  $j$  med sannolikhet  $p_{ij}$  (som beror på tillstånd och styrning).

## Dynamisk programmering: Stokastisk bakåtrekursion

Vi står i nod  $i \in S_k$  och siktar mot nod  $j \in S_{k+1}$

dvs. vill använda båge  $(i, j)$ .

Var hamnar vi?

I nod  $j$  med sannolikhet  $p_{ij}$  (som beror på tillstånd och styrning).

Exempel: Vi står i nod 1. I nästa steg finns noderna 2, 3, och 4.

## Dynamisk programmering: Stokastisk bakåtrekursion

Vi står i nod  $i \in S_k$  och siktar mot nod  $j \in S_{k+1}$

dvs. vill använda båge  $(i, j)$ .

Var hamnar vi?

I nod  $j$  med sannolikhet  $p_{ij}$  (som beror på tillstånd och styrning).

Exempel: Vi står i nod 1. I nästa steg finns noderna 2, 3, och 4.

Vi siktar mot nod 2,

## Dynamisk programmering: Stokastisk bakåtrekursion

Vi står i nod  $i \in S_k$  och siktar mot nod  $j \in S_{k+1}$

dvs. vill använda båge  $(i, j)$ .

Var hamnar vi?

I nod  $j$  med sannolikhet  $p_{ij}$  (som beror på tillstånd och styrning).

Exempel: Vi står i nod 1. I nästa steg finns noderna 2, 3, och 4.

Vi siktar mot nod 2, och hamnar i nod 2 med sannolikhet 0.7,

## Dynamisk programmering: Stokastisk bakåtrekursion

Vi står i nod  $i \in S_k$  och siktar mot nod  $j \in S_{k+1}$

dvs. vill använda båge  $(i, j)$ .

Var hamnar vi?

I nod  $j$  med sannolikhet  $p_{ij}$  (som beror på tillstånd och styrning).

Exempel: Vi står i nod 1. I nästa steg finns noderna 2, 3, och 4.

Vi siktar mot nod 2, och hamnar i nod 2 med sannolikhet 0.7, i nod 3 med sannolikhet 0.2, och i nod 4 med sannolikhet 0.1.

## Dynamisk programmering: Stokastisk bakåtrekursion

Vi står i nod  $i \in S_k$  och siktar mot nod  $j \in S_{k+1}$

dvs. vill använda båge  $(i, j)$ .

Var hamnar vi?

I nod  $j$  med sannolikhet  $p_{ij}$  (som beror på tillstånd och styrning).

Exempel: Vi står i nod 1. I nästa steg finns noderna 2, 3, och 4.

Vi siktar mot nod 2, och hamnar i nod 2 med sannolikhet 0.7, i nod 3 med sannolikhet 0.2, och i nod 4 med sannolikhet 0.1.

Den förväntade kostnaden för att komma till nästa steg blir då  $0.7c_{12} + 0.2c_{13} + 0.1c_{14}$ .

## Dynamisk programmering: Stokastisk bakåtrekursion

Vi står i nod  $i \in S_k$  och siktar mot nod  $j \in S_{k+1}$

dvs. vill använda båge  $(i, j)$ .

Var hamnar vi?

I nod  $j$  med sannolikhet  $p_{ij}$  (som beror på tillstånd och styrning).

Exempel: Vi står i nod 1. I nästa steg finns noderna 2, 3, och 4.

Vi siktar mot nod 2, och hamnar i nod 2 med sannolikhet 0.7, i nod 3 med sannolikhet 0.2, och i nod 4 med sannolikhet 0.1.

Den förväntade kostnaden för att komma till nästa steg blir då  $0.7c_{12} + 0.2c_{13} + 0.1c_{14}$ .

Det betyder också att vägen ska fortsätta från nod 2 med sannolikhet 0.7, från nod 3 med sannolikhet 0.2 och från nod 4 med sannolikhet 0.1.



## Dynamisk programmering: Stokastisk bakåtrekursion

Vi står i nod  $i \in S_k$  och siktar mot nod  $j \in S_{k+1}$

dvs. vill använda båge  $(i, j)$ .

Var hamnar vi?

I nod  $j$  med sannolikhet  $p_{ij}$  (som beror på tillstånd och styrning).

Exempel: Vi står i nod 1. I nästa steg finns noderna 2, 3, och 4.

Vi siktar mot nod 2, och hamnar i nod 2 med sannolikhet 0.7, i nod 3 med sannolikhet 0.2, och i nod 4 med sannolikhet 0.1.

Den förväntade kostnaden för att komma till nästa steg blir då  $0.7c_{12} + 0.2c_{13} + 0.1c_{14}$ .

Det betyder också att vägen ska fortsätta från nod 2 med sannolikhet 0.7, från nod 3 med sannolikhet 0.2 och från nod 4 med sannolikhet 0.1.

Vi ska därför addera  $0.7f_2(2) + 0.2f_2(3) + 0.1f_2(4)$  till  $f_1(s_1)$ .

# Dynamisk programmering: Stokastisk bakåtrekursion

Vi använder alltså väntevärden:

$$f_k(s_k) = \min_{x_k} (E(c_k(x_k, s_k) + f_{k+1}(s_{k+1})))$$

# Dynamisk programmering: Stokastisk bakåtrekursion

Vi använder alltså väntevärden:

$$f_k(s_k) = \min_{x_k} (E(c_k(x_k, s_k) + f_{k+1}(s_{k+1})))$$

I exemplet:

$$f_1(s_1) = \min ((0.7(c_{12} + f_2(2)) + 0.2(c_{13} + f_2(3)) + 0.1(c_{14} + f_2(4))), \dots).$$

# Dynamisk programmering: Stokastisk bakåtrekursion

Vi använder alltså väntevärden:

$$f_k(s_k) = \min_{x_k} (E(c_k(x_k, s_k) + f_{k+1}(s_{k+1})))$$

I exemplet:

$$f_1(s_1) = \min ((0.7(c_{12} + f_2(2)) + 0.2(c_{13} + f_2(3)) + 0.1(c_{14} + f_2(4))), \dots).$$

Uppnystningen av lösningen blir osäker. Vi kanske inte hamnar där vi vill.

# Dynamisk programmering: Stokastisk bakåtrekursion

Vi använder alltså väntevärden:

$$f_k(s_k) = \min_{x_k} (E(c_k(x_k, s_k) + f_{k+1}(s_{k+1})))$$

I exemplet:

$$f_1(s_1) = \min ((0.7(c_{12} + f_2(2)) + 0.2(c_{13} + f_2(3)) + 0.1(c_{14} + f_2(4))), \dots).$$

Uppnystningen av lösningen blir osäker. Vi kanske inte hamnar där vi vill.

Det räcker inte att ta fram den mest sannolika vägen, vi måste även veta hur vi ska fortsätta om vi hamnar fel.

# Dynamisk programmering: Stokastisk bakåtrekursion

Vi använder alltså väntevärden:

$$f_k(s_k) = \min_{x_k} (E(c_k(x_k, s_k) + f_{k+1}(s_{k+1})))$$

I exemplet:

$$f_1(s_1) = \min ((0.7(c_{12} + f_2(2)) + 0.2(c_{13} + f_2(3)) + 0.1(c_{14} + f_2(4))), \dots).$$

Uppnystningen av lösningen blir osäker. Vi kanske inte hamnar där vi vill.

Det räcker inte att ta fram den mest sannolika vägen, vi måste även veta hur vi ska fortsätta om vi hamnar fel.

Slutsats: Ta fram optimal styrning från *alla* tillstånd.

# Dynamisk programmering: Stokastisk bakåtrekursion

Vi använder alltså väntevärden:

$$f_k(s_k) = \min_{x_k} (E(c_k(x_k, s_k) + f_{k+1}(s_{k+1})))$$

I exemplet:

$$f_1(s_1) = \min ((0.7(c_{12} + f_2(2)) + 0.2(c_{13} + f_2(3)) + 0.1(c_{14} + f_2(4))), \dots).$$

Uppnystningen av lösningen blir osäker. Vi kanske inte hamnar där vi vill.

Det räcker inte att ta fram den mest sannolika vägen, vi måste även veta hur vi ska fortsätta om vi hamnar fel.

Slutsats: Ta fram optimal styrning från *alla* tillstånd.

Jämför billigaste väg-träd.