ISSN 0280-5316
ISRN LUTFD2/TFRT--5680--SE

Optimering av Körtider i Realtids-baserat Lagerstyrningssystem

Markus Öberg

Institutionen för Reglerteknik
Lunds Tekniska Högskola
November 2001

Department of Automatic Control
Lund Institute of Technology
Box 118
SE-221 00 Lund Sweden

Document name
MASTER THESES

Date of issue
November 2001

Document Number
ISRN LUTFD2/TFRT--5680—SE

Markus Öberg

Supervisor
Per Hagander, LTH
Björn Scholtz, MA System

Sponsoring organization

Title and subtitle
Optimering av körtider i realtidsbaserat lagerstyrningssystem. (Optimizing operation times in a storage control system)

Abstract
A storage control system named Astro is developed and sold by MA-system in Lund. This system is used for controlling the flow of material, and a central part is to assign tasks to the different trucks in the system. Every task takes a certain time to execute, and the trucks drive a specific total time in the storage. This total time depends on in which order the tasks are executed. A favorable order decreases the time the trucks drive without load, thus increasing the utility-factor.

Some kind of long term planning when assigning tasks is necessary to accomplish a high utility-factor. This is achieved by applying an optimization technique named Simulated Annealing on a model of a storage system. The resulting utility-factor has been compared with the utility-factor obtained by the task-assigning rule used today, which is to always execute the task with shortest ready-time first. The difference in utility factors has increased the capacity with 15%, or decreased the need of trucks by 23%.

The big increase in capacity, even though it is theoretical, suggests further investigating the introduction of an optimization technique. There are many suitable techniques, which all have in common the need for information of how long time it takes to travel from one place in the storage to another. The creation of this time-information is necessary, and the first natural step in implementing an optimization technique.

Keywords

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280-5316

ISBN

Language
Swedish

Number of pages
55

Recipient's notes

Security classification

Sammanfattning

Lundaföretaget MA-system utvecklar ett lagerstyrningssystem, Astro. Detta system styr materialflöden, och ett centralt moment är att truckar av olika typer tilldelas uppdrag som skall utföras. Varje uppdrag innebär en viss tidsåtgång, och tillsammans kör truckarna en viss sträcka i lagret. Sträckan truckarna kör beror på i vilken ordning uppdragen utförs. Genom att för alla truckar hitta en gynnsam körväg kan tiden truckarna kör tomma minskas, och utnyttjandegraden på så vis ökas.

Skall en hög utnyttjandegrad uppnås krävs framförhållning vid val och tilldelning av uppdrag. För att uppnå detta har en optimeringsteknik kallad Simulated Annealing utförts på en modell av ett lagersystem. Denna optimeringsteknik har styrt tilldelningen av uppdrag till truckarna i lagermodellen. Den totala utnyttjandegraden har jämförts med den metod som idag tillämpas för uppdragstilldelning. Skillnaden i utnyttjandegrad om Simulated Annealing används har i modellen inneburit att truckbehovet minskat med 23%, eller motsvarande att kapaciteten ökat med 15%.

Den stora skillnad i truckbehov som åstadkommits, om än teoretiskt, talar för att vidare undersöka möjligheterna att införa en optimeringsmetod. Det finns många passande tekniker, och gemensamt för samtliga är att de behöver information om hur lång tid det tar att färdas från en plats till en annan i lagret. Skapandet av denna tidsinformation är högst väsentlig, samt första naturliga steg vid införande av en utnyttjandegradsoptimering.

Sökord

heuristisk sökning, körtid, lagerstyrningssystem, optimering, simulated annealing, traveling salesman, vehicle routing

Innehållsförteckning

1 Introduktion

Lundaföretaget MA-system utvecklar lagerstyrningssystemet Astro. Detta system styr materialflöden och ett centralt moment är att truckar av olika typer tilldelas uppdrag som skall utföras. Varje uppdrag innebär en viss tidsåtgång vilken beror på en mängd faktorer, till exempel vilken typ av truck som används, var i lagret uppdraget startar och slutar, lagrets principiella utseende, truckförarens skicklighet med mera. Uppdragen skapas och tilldelas en viss truck kontinuerligt och systemet arbetar i realtid.

De många krav och bivillkor som ställs på uppdragshanteringen ger ett komplext system med ett oerhört stort antal möjliga lösningar. Till exempel har en truck vilken skall utföra 10 uppdrag, 10! = 3628800 olika möjligheter på i vilken ordning dessa uppdrag skall utföras. Alla dessa olika möjligheter kan samtliga ses som en unik lösning, och varje lösning innebär en viss tidsåtgång.

I dagsläget har MA-system till SKF levererat lagerstyrningssystem för lager med 100000 pall- och plockplatser där mer än 12000 uppdrag utförs varje dag. Enkel överslagsräkning säger att det inom rimlig tid är omöjligt att matematiskt finna den ordning med vilken dessa uppdrag skall utföras för att tidsåtgången skall bli optimal.

1.1 Problemformulering

Den tid det tar för ett visst antal truckar att utföra ett visst antal uppdrag skall minskas. Metoden för detta skall vara generell och vara oberoende av lagrets utformning, antal truckar och antal uppdrag som skall utföras. Dessutom skall den arbeta i realtid och kunna hanteras av en PC med en i dagsläget normal beräkningskapacitet. Examensarbetet skall byggas på teoretiska principer och ge ett lösningsförlag för problemet. Faller detta väl ut skall det kunna ligga till grund för en uppdragshanteringsmodul i Astro.

1.2 Syfte

Kan den tid det tar att utföra ett visst antal uppdrag minskas, så minskar även behovet av antalet truckar i systemet, och därmed kostnader i form av truck, personal etc. Resultatet blir att MA-system kan leverera ett billigare och konkurrenskraftigare system.

1.3 Metod

För att inte påverkas av den metod som i dagsläget används, studeras inte denna. All information om systemet ges muntligen från Björn Scholtz, eller hämtas från MA-systems hemsida.

Genom att studera de vetenskapliga metoder som i dagsläget finns för lösning av detta eller liknande problem, arbetas ett lösningsförslag fram och utvärderas. Lösningen jämförs med en synnerligen enkel modell för tidsoptimering, samt med en modell av hur systemet fungerar idag. För att praktiskt lösa detta problem skrivs ett program, vilket både modellerar, simulerar och optimerar systemet. Programmet är skrivet i C, vilket är det språk som Astro är programmerat i.

1.3.1 Primärdata

Per Hagander vid Institutionen för Reglerteknik har presenterat tekniker för lösning av problem liknande detta, samt under hela arbetet bidragit med många infallsvinklar.

Björn Scholtz vid MA-system har presenterat och förklarat problemet, samt under hela arbetet svarat på uppkomna frågor och bidragit med många infallsvinklar.

1.3.2 Sekundärdata

Genom att använda de sökverktyg som UB2 i Lund tillhandahåller, har ett flertal böcker och publicerade artiklar av intresse hittats. Sökord som har används är optimering, körsträcka körtid, lager, truck, heuristisk, simulated annealing, tabu search samt kombinationer av dessa ord och deras engelska eller svenska motsvarigheter. Allt material av intresse har varit skrivet på engelska och är utgivet under senare delen av 90-talet. Studier som direkt kan tillämpas på detta problem har inte påträffats.

1.4 Avgränsningar

Endast en av många existerande metoder för lösning av denna typ av optimeringsproblem används och utvärderas. De olika metoderna, vilka senare kort nämns, är omfattande och det finns mängder av litteratur som behandlar var och en. Att för detta problem utförligt ta upp och tillämpa mer än en, ligger utanför arbetets omfattning.

1.5 Källkritik

Även om samtliga artiklar och den litteratur som använts har haft flera sökord gemensamma med denna uppsats, har inte något material eller exempel visat sig vara direkt applicerbart på detta problem. Det genomgångna materialet har mer visat om en viss riktning eller metod är lämplig eller ej. Vad som i all litteratur genomgående har påpekats, är att varje problem är unikt och att det inte finns någon universell lösningsmetod, utan faktiskt flera olika metoder för att lösa samma problem. Vad som är av större vikt är problemet i sig samt i vilket sammanhang en lösning skall användas.

Den information som har hämtats från MA-systems hemsida riktar sig främst mot potentiella kunder och beskiver Astro med syfte att marknadsföra systemet. Argument varför Astro är bra ges inte, utan en så objektiv ställning som möjligt intas i detta arbete.

1.6 Förkortningar

2-i = 2-interchange
CFC = Closest First with time Constraints
SA = Simulated Annealing
SF = Shortest time First

Engelska benämningar används för flera begrepp. Framförallt för att dessa benämningar är de mest kända, använda och accepterade. Dessutom har ingen enhetlig svensk namngivning inom området påträffats.

1.7 Disposition

Kapitel 1 Introduktion, introducerar examensarbetet och ger information om rapporten.

Kapitel 2 Systembeskrivning, beskiver kort Astro, det materialhanteringssystem som MA-system utvecklar och levererar. Bakgrunden till detta exmensarbetes problemformulering beskrivs och förklaras.

Kapitel 3 Optimeringsteorier, behandlar en för detta optimeringsproblem lämplig lösningsmetod. Andra tekniker av intresse nämns mycket kortfattat. Även en enkel lösningsmetod beskrivs, samt den metod som idag kan anses vara den som används.

Kapitel 4 Modellering av systemet, modellerar systemet och sätter det i samband med vald lösningsmetod respektive valda metoder för utvärdering.

Kapitel 5 Resultat och utvärdering, presenterar resultatet av optimeringen i jämförelse med en enkel lösningsmetod samt den metod som idag används.

Appendix A - SA-variabler, visar hur Simulated Annealings olika variabler påverkar resultatet.

Appendix B - Källkod, innehåller den programkod som skrivits för att kunna lösa problemet.

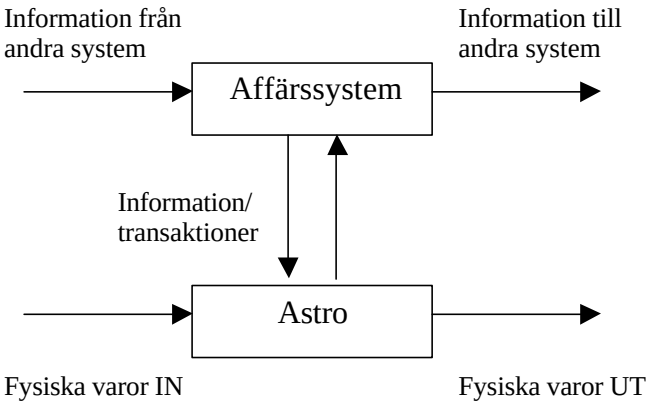
2 Systembeskrivning

All information om Astro kommer muntligen från Björn Scholtz eller från MA-systems hemsida. I stora drag beskrivs här hur systemet fungerar. Flera funktioner har utelämnats och syftet med beskrivningen är att ge sådan bakgrundsinformation att problemställningen tydliggörs och är uppenbar.

Benämningarna ”order” respektive ”uppdrag” används synonymt, och kan förenklat sägas vara ett enskilt arbetsmoment som en truck i lagret utför. Varje uppdrag/order startar på en viss punkt i lagret och slutar vanligtvis på en annan, och tar en viss tid att utföra.

2.1 Astro

Astro är ett generellt lagerstyrningssystem för många olika branscher inom tillverkningsindustri och handel. Några av företagen som använder systemet är ABB, Ericsson, Ikea, Pripps-Ringnes och SKF. Med hjälp av ett affärssystem överförs orderinformation till Astro, som sedan behandlar och optimerar denna information så att materialhanteringen blir mer effektiv, se figur 2.1.



Figur 2.1 Informations och materialflöde för Astro.

I systemet är det är enkelt att ta bort och lägga till order, allt arbete och information sker och behandlas i realtid. Kund- och packorder sänds automatiskt från affärssystemet till Astro som delar in i fullpallar och plockartiklar, och fördelar dessa på olika uppdragsköer. Via monitorer kan arbetet följas i realtid och avvikelser från plan kan snabbt korrigeras.

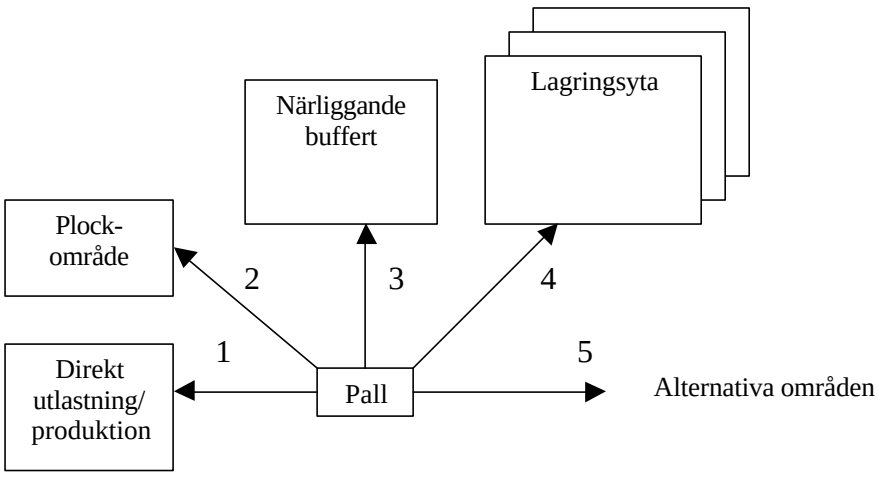
Optimering sker i realtid och oavsett när en order registreras, plockas vanligtvis den högst prioriterade ordern först. Prioriteringsgraden styrs av när en order måste vara klar, det vill säga den order som har lägst färdigtid utförs först. En order kan till exempel vara ett uppdrag där en viss lastpall skall köras till utlastning för vidare extern transport med lastbil, eller att en viss artikel skall köras till rätt maskin i produktionen för vidare bearbetning.

Det finns en primitiv styrning kallad Multicycle, med vars hjälp kan man tvinga truckarna att följa ett visst mönster, till exempel:

1. Tag först en inlagringspall och lagra in den
2. Tag därefter en pall som skall ut från lagret
3. Börja om från 1.

På detta sätt kan truckarna tvingas köra en cykel som i många fall blir lite effektivare än om man alltid skulle ta den pall med lägst färdigtid först. Vad som då erhålls är en bättre utnyttjandegrad av truckar, som i sin tur bestäms av det faktiska tidsförhållandet mellan tomkörningar och körningar med någon typ av artikel. Multicycle är dock väldigt trubbigt och jobbigt att konfigurera.

Den optimering som avgör var en viss pall som ankommer till lagret skall dirigeras, illustreras i figur 2.2. Astro finner det ställe där den för ögonblicket behövs mest, siffrorna i figuren anger prioriteringsgrad och FIFO (First In First Out) är grunden i all hantering.



Figur 2.2 Inläggningsprioritet i Astro.

Tekniskt är Astro är en klient-server lösning där man arbetar från en egen PC. Användargränssnittet är grafiskt, skrivet i Java och exekveras i en webbläsare. Själva systemet är programmerat i C och körs på Oracle eller Microsoft SQL server. Via radiolänk sänds information till truckförarna om vilka uppdrag som skall utföras. Uppdragen presenteras som en lista på skärmar i truckarna, samma lista för samtliga truckar, och när en truck påbörjar ett uppdrag, försvinner det från listan.

Kortfattat är grundidén bakom Astro enkelhet, information i realtid samt kontroll av material och informationsflödet. Genom att bland annat använda Multicycle och inläggningsprioritet uppnås en viss grad av optimering.

2.2 Förbättringsmöjligheter

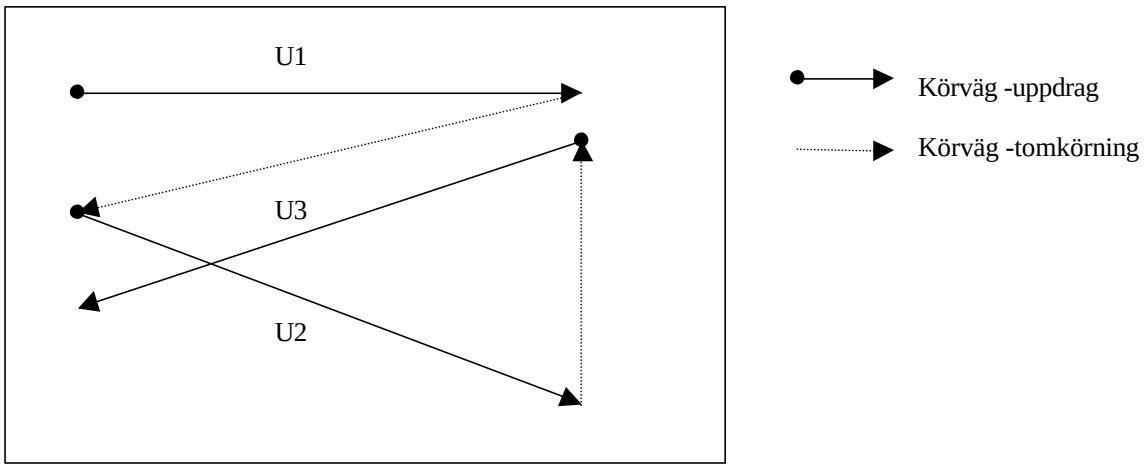
Naturligtvis kan ett system av denna typen ständigt förbättras och utvecklas på väldigt många punkter. Att finna en optimal lösning eller att tala om optimering går inte i ordens strikta betydelse. Genom att använda olika tekniker och regler för styrning uppnås däremot ett

fungerade system som är kommersiellt gångbart i praktiken, vilket ofta innebär olika former av kompromisser. Ordet optimering kan i detta sammanhang ses som en förbättring eller effektivisering vilken ger systemet någon form av mervärde.

En faktor som bör vara så hög som möjligt är truckarnas tidigare nämnda utnyttjandegrad. Kör trucken tom 55% av sin arbetstid, så är utnyttjandegraden 100%-55% = 45%. I ett system som Astro där order uppstår och skall utföras, är det inte intressant hur många truckar som utför dessa order, det viktiga är att uppdragen utförs i rätt tid och med rätt kvalitet. En hög truck-utnyttjningsgrad ger ett mindre behov av truckar. Och eftersom truckar är en kostnad bör deras antal hållas så låg som möjligt.

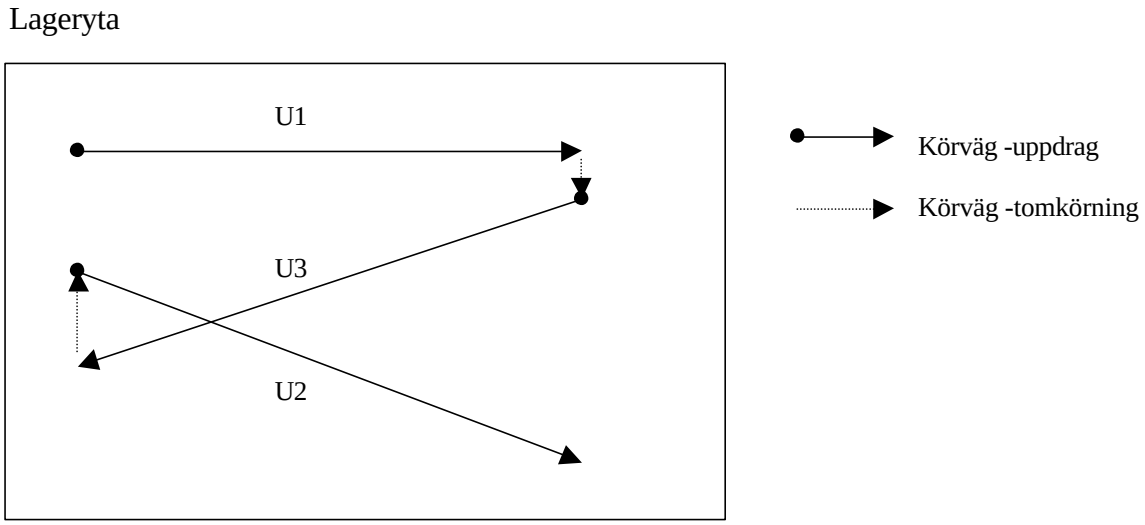
Idag utförs uppdragen så gott som alltid i prioriteringsordning med avseende på uppdragens sista färdigtid. Med hjälp av figur 2.3 illustreras en situation som kan uppstå vid denna styrningsmetod, numera kallad ”kortast tid först”.

Uppdrag	Sista färdigtid
U1	13.00
U2	14.00
U3	15.00



Figur 2.3 Uppdragskörning i ett lager, alternativ 1 - kortast tid först.

Det är uppenbart att en stor andel tomkörning sker. Ett annat sätt att utföra samma tre uppdrag på visas i figur 2.4.



Figur 2.4 Uppdragskörning i ett lager, alternativ 2.

Under förutsättning att en längre körväg ger en ökad tidsåtgång, så ger alternativ 2 en effektivare utnyttjandegrad av den truck som utför uppdragen. Det kan finnas andra faktorer som beror på i vilken ordning uppdragen utförs, och som kan påverka tidsåtgången, men dessa bortses från tills vidare. Problemet är en variant av det klassiska Traveling Salesman problemet, där det gäller att finna den kortaste färdvägen mellan ett antal punkter som alla måste besökas. Finns det en så god tidsmarginal att dessa tre uppdrag kan utföras enligt båda alternativen utan att någon försening inträffar, så illustrerar alternativ 2 alltså bästa körordning.

Något som är mycket viktigt är att varje uppdrag blir färdigt i tid, och för att uppnå detta kan det vara omöjligt att välja en körväg som ger en optimal utnyttjandegrad för den truck som används. Är dock samtliga uppdrag försenade, ger alternativ 2 en lägre genomsnittlig förseningstid för uppdragen. Det ska påpekas att det illustrerade exemplet är mycket litet, problemet får en helt annan dimension när 20 truckar arbetar samtidigt i ett lager där det en viss tidpunkt finns 500 uppdrag att utföra. Det kan vara så att samtliga truckar kan utföra vilket som helst av dessa 500 uppdrag, och kombinationen av de körvägar truckarna kan ta blir oerhört stor.

Sammanfattningsvis gäller det att för samtliga truckar få en så hög utnyttjningsgrad som möjligt, samtidigt som varje uppdrag måste bli färdigt i tid. I vilken ordning uppdragen utförs, och av vilken truck, kan då spela en stor roll. En metod och teknik för detta skall i det här arbetet presenteras. Verktyget Multicycle är i detta sammanhang inte tillräckligt.

3 Optimeringsteorier

Det finns flera olika metoder för att utföra en optimering när sökområdet och antalet alternativa lösningar är väldigt stort. Vad som framförallt styr valet av teknik är problemet i sig, och en metod, Simulated Annealing (SA), har upplevts som den mest lämpliga i detta fall. Varför motiveras i avsnitt 4.5, där även tekniken sätts i sammanhang med Astro och problemet ifråga. Två andra tekniker nämns i detta kapitel mycket kort, helt enkelt därför att de är väldigt kända och båda skulle ha kunnat löst problemet. Kommer dessutom fortsatt arbete inom detta problemområde att ske, bör dessa två vara beskrivna för att ge andra infallsvinklar. Ytterligare två tekniker för uppdragstilldelning presenteras. Ingen av dessa är någon egentlig optimeringsteknik, utan kan mer ses som ett system med regler för bestämmande av vilket uppdrag som härnäst skall utföras och av vilken truck. En av dessa liknar den princip som idag används, och tas upp i jämförelsesyfte.

3.1 Simulated Annealing

Simulated Annealing, också känt som Monte Carlo Annealing, statistical cooling, probabilistic hill-climbing, stochastic relaxation och probabilistic exchange algorithm, är en teknik som blev populär för ungefär tio år sedan, och har sedan dess visat sig vara en effektiv metod för att lösa en mängd olika problem av vitt skild karaktär. Tekniken är enkel men kan visa sig vara svår att tillämpa beroende på hur systemet modelleras. Detta är inget unikt för SA utan gäller för i princip alla optimeringstekniker, vilket betonas i framförallt [Michalewicz Z.].

3.1.1 Inledning

Tekniken fungerar på så sätt att man sätter en tid t till 0, initialiserar en temperatur T och skapar en lösning vars enda krav är att den är genomförbar. Ofta skapas denna initiallösning slumpmässigt. Utifrån denna initiallösning v_c väljs en ny lösning v_n och därefter jämförs dessa med hjälp av en värderingsfunktion eval(), vilket ger en differens eval(v_c) - eval(v_n). Den nya lösningen accepteras om den innebär en förbättring, eller om $e^{(eval(vc) - eval(vn)) / T} > R$, där R är ett slumpmässigt tal mellan 0 och 1. Om den nya lösningen accepteras, ersätts den gamla av den nya, som då blir den befintliga lösningen. Utifrån denna väljs återigen en ny lösning vilken jämförs med den befintliga osv. T är initialt hög vilket ger ett mer slumpmässigt val av nya lösningar, för att efterhand sjunka och till slut medföra att en sämre lösning i princip aldrig accepteras. Detta förfarande, att sämre lösningar kan accepteras, är centralt för SA och används för att inte fastna i lokala optima. Ett problem som detta har oerhört många lokala optima men bara ett globalt. Det gäller att finna detta globala optima eller något av de bästa lokala.

Strukturen beskrivs i figur 3.1. där problemet är ett minimeringsproblem, precis som arbetets uppgift; att minimera den tid det tar för ett visst antal truckar att utföra ett visst antal uppdrag.

```
procedure simulated annealing
begin
   t = 0
   initialize T
   select a current point vc at random
   evaluate vc
   repeat
      repeat
         select a new point vn in the neighborhood of vc
         if eval(vn) < eval(vc)
            then vc = vn
         else if e (eval(vc)-eval(vn)) / T > random(0,1)
            then vc = vn
      until(termination condition)
      T = g(T,t)
      t = t + 1
   until(halting condition)
end
```

Figur 3.1 Simulated Annealing - struktur

Idéerna som lagt grunden för SA kommer från den process ett smält material genomgår vid nedkylning. Om en fast substans upphettas över dess smältpunkt och därefter tillåts svalna och återta sin fasta form, så är substansens kristallstruktur beroende på under hur lång tid avsvalningen sker. Kristallstrukturen är avgörande för vilka egenskaper och vilken kvalité materialet får. Den inre loopen i SA, lokal sökning, representerar det faktum att temperaturförändringar sker språngvis. Materialet genomgår strukturförändring, en liten nedkylning sker, för att återigen genomgå strukturförändring, nedkylning osv. Analogin mellan denna fysikaliska process och ett optimeringsproblem är uppenbar, och de ekvivalenta termerna beskrivs i tabell 3.1.

Fysikaliskt system	Optimeringsproblem
tillstånd	möjlig lösning
energi	utvärderingsfunktion
grundtillstånd	optimal lösning
snabb avkylning	lokal sökning
temperatur	kontrollparameter T
försiktig avkylning	simulated annealing

Tabell 3.1 Analogi mellan fysikaliskt system och optimeringsproblem.

Sannolikheten för acceptans p av en sämre lösning som en funktion av variabeln T illustreras i tabell 3.2, och p som en funktion av differensen eval(v_c)-eval(v_n) visas i tabell 3.3. Problemet är ett minimeringsproblem, och ger den nya lösningen v_n att eval(v_c) - eval(v_n) < 0, så är alltså v_n sämre än den befintliga lösningen v_c, men accepteras ändå med sannolikheten p, vilken beräknas enligt:

$$p = e^{(eval(vc) - eval(vn)) / T}$$

T	$p = e^{-13/T}$
1	0.000002
5	0.07
10	0.27
20	0.52
50	0.77
10^{10}	1.00

Tabell 3.2 Sannolikhet p för acceptans som en funktion av T då $eval(v_c) - eval(v_n) = -13$.

$eval(v_c) - eval(v_n)$	$p = e^{(eval(vc) - eval(vn)) / 10}$
-1	0.90
-7	0.50
-13	0.27
-25	0.08
-100	0.00004

Tabell 3.3 Sannolikhet p för acceptans som en funktion av $eval(v_c) - eval(v_n)$ då T = 10.

Det är tydligt att vid ett högt T finns en högre acceptans för en sämre lösning, och att denna minskar då T avtar. Vidare framgår att ju sämre en lösning är i relation till den befintliga, desto lägre är sannolikheten för acceptans.

Precis som andra sökalgoritmer behöver SA svar på följande problemspecifika och mycket viktiga frågor [Michalewicz Z.]:

- Hur definieras en lösning?
- Vilka angränsande lösningar har en befintlig lösning?
- Hur värderas kvalitén av en lösning?
- Hur bestäms initiallösningen?

Svaren på dessa frågor ger själva strukturen av en lösning med angränsande lösningar, en utvärderingsfunktion och en initieringsfunktion. Detta räcker emellertid inte, utan svar krävs också på:

- Hur bestäms starttemperatur T?
- Hur förändras T med tiden, g(T,t)?
- Hur bestäms stoppkriterie för inre loop (termination condition)?
- Hur bestäms stoppkriterie för yttre loop (halting condition)?

3.1.2 Variabler

Avsvalningshastighet g(T,t) och initialtemperatur T, liksom hur många gånger SA's inre loop skall köras, väljs ibland slumpmässigt. Variablerna anpassas därefter manuellt för att bästa lösning skall uppnås [Rayward-Smith,V.J et al.]. Dock finns det förfaranden som ofta ger bra resultat och som kan vara lämpliga att inleda med.

Initial temperatur
Som tidigare nämnts skall temperaturen i början (T_0) vara så hög att ett stort antal nya lösningar accepteras. I praktiken måste då kvalitetsskillnaden mellan två närliggande lösningar vara känd, vilket inte alltid är fallet. Ett bra sätt att lösa problemet består i att välja vad som verkar vara ett högt T_0, detta kan variera oerhört beroende av vilket typ av problem som löses. Därefter räknas acceptansgraden under programmets körning, vilken för de flesta fall bör ligga mellan 40% och 60% [Rayward-Smith,V.J et al.].

Avsvalningshastighet
Detta är en av de mest viktiga faktorerna vid praktisk tillämpning, och det finns i princip två typer av avsvalningsmetoder, vilka har likheter med homogena och inhomogena Markovkedjor. I det homogena fallet sker nedsvalning vid en fix temperatur tills lokal jämvikt råder. Att avgöra när jämvikt inträffar innebär ytterligare ett problem som måste definieras och lösas. När väl jämviktstillstånd inträffat sänks temperaturen och proceduren upprepas åter. Antal itereringar vid varje temperatursteg kan vara väldigt stort, men å andra sidan kan det också vara litet och orsaka snabba temperatursänkningar. I det inhomogena fallet reduceras temperaturen efter ett bestämt antal itereringar, vilket är både mindre komplicerat och i praktiken mer använt [Rayward-Smith,V.J et al.].

För båda fall måste formen på avsvalningskurvan bestämmas. Två populära metoder finns, där den första, α, beskrivs enligt:

$T = \alpha * T$, där α är en konstant nära 1 (vanligt mellan 0.9 och 0.99)

Den andra, β, enligt:

$T = T / (1 + \beta * T)$, där β är en positiv konstant nära 0

Sluttemperatur
Teoretiskt skall temperaturen sänkas tills sluttemperaturen (T_f) når ett värde mycket nära 0, men i praktiken är det tillräckligt om den sänks tills det inte längre sker några förbättringar. När detta inträffar kan avgöras med hjälp av att acceptansgraden beräknas, precis som vid bestämmande av initialtemperatur. En annan metod är att temperaturen sänks till en bestämd nivå då man vet att inga sämre lösningar kan accepteras.

Antal itereringar
Antal itereringar N_{it} i SA's inre loop kan lätt bestämmas med hjälp av de tre ovan nämnda variablerna T_0, T_f och α eller β, om man använder en inhomogen avsvalningsmetod. I det homogena fallet beror N_{it} även på definitionen av lokal jämvikt. I de inhomogena fallen α och β beräknas N_{it} enligt:

$$N_{it} = (\log T_f - \log T_0) / \log \alpha$$

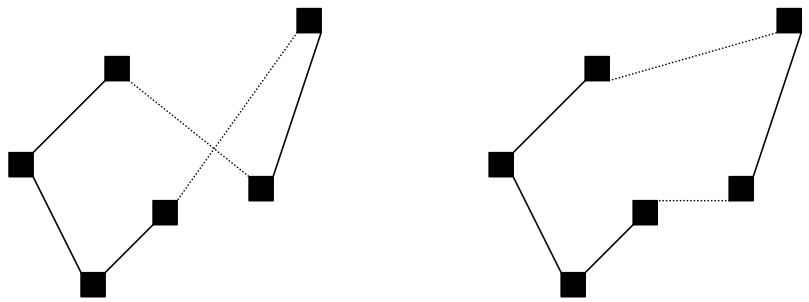
respektive

$$N_{it} = (T_0 - T_f) / (\beta * T_0 * T_f)$$

3.1.3 Andra faktorer och avväganden

Problemspecifika avgöranden
Det viktigaste avgörandet för ett problem består i att definiera angränsande lösningar. Teoretiskt skall varje lösning kunna nås från samtliga andra lösningar, vilket vanligtvis lätt uppnås. När det finns begränsningar i möjligheten att använda en lösning, på engelska constraints on feasibility, kan det bli nödvändigt att i utvärderingsfunktionen införa strafftillägg där eval()-funktionen får lösningen att se sämre ut än vad den egentligen är. Begränsningar och strafftillägg kan vara komplicerade, men absolut nödvändiga när det finns många olika krav på en lösning.

Det är ofta fördelaktigt att definiera angränsande lösningar så att dessa kan utvärderas med få operationer, vilket ger en snabbare optimering. Till exempel så är det för traveling salesman problemet när λ-utbyte används, inte nödvändigt att utvärdera hela turen utan bara effekten av de ändrade länkarna. λ-utbyte innebär att ett antal, λ, länkar mellan städer bryts och därefter sätts in mellan andra städer. Se figur 3.2 där ett 2-utbyte illustreras. 2-utbyte kommer i fortsättningen att kallas för 2-interchange (2-i), vilken är den benämning som dominerar i litteraturen.



Figur 3.2 En tur (a) före och (b) efter att 2-utbyte är tillämpat. Streckade linjer indikerar förändrade färdvägar.

Acceptansfunktionen
Denna funktion är exponentiell:

$$e^{(eval(vc) - eval(vn)) / T} > R$$

Det har påpekats att det kräver mycket beräkningsarbete för en dator när denna likhet utvärderas. Att approximera med 1 - eval(vn) – eval(vc) / T sparar ofta mycket beräkningstid utan att kvalitén försämras [Ansari N. Hou E.].

Reannealing
Detta är en teknik som med framgång har använts. En variant av reannealing består i att spara den temperatur som ger bäst lösning, och när hela simuleringen är färdig starta den på nytt i närheten av denna bästa temperatur och genomföra en mer noggrann sökning i detta område.

En annan teknik går ut på att sänka och höja temperaturen, beroende om en förbättring av befintlig lösning sker eller inte. Till exempel om en förbättring av en lösning sker, sänks temperaturen med:

$T = T / (1 + \beta*T)$, β är en konstant nära 0

Motsvarande om en försämring av en lösning sker, höjs temperaturen med:

$T = T / (1 - \gamma*T)$, γ är en konstant nära 0

Förbjudna områden
Problemspecifik information kan avgöra om det finns en möjlighet att en ny lösning senare kan leda till en bättre lösning, eller om denna möjlighet är försumbar och kan ignoreras. Som exempel kan nämnas att för en lösning av traveling salesman problemet tillåts endast de fyra närmaste punkterna för varje punkt att övervägas som ett led i en lösning, alla andra tillhör ett förbjudet område. Anledningen för att göra detta är att en avsevärt förbättrad beräkningstid uppnås genom att sökområdet minskar, utan att lösningskvalitén riskerar försämras.

3.2 Evolutionary Algorithm

Under 1950-talet uppfanns en typ av algoritm som idag går under namnet Evolutionary eller Genetic algoritm, vilken är designad med den naturliga urvalsprocessen i åtanke. Tekniken har under framförallt 90-talet fått mycket uppmärksamhet, och används precis som SA för att lösa komplexa optimeringsproblem med stora sökområden. Evolutionary Algoritm arbetar med en population av lösningar, P(t), och utifrån dessa lösningar, parents, skapas nya lösningar, children, enligt bestämda metoder. De nya lösningar som visar sig vara bäst används sedan för att skapa nya children osv. Strukturen illustreras i figur 3.3.

```
procedure evolutionary algorithm
begin
   t = 0
   initialize P(t)
   evaluate P(t)
   while (not termination condition) do
   begin
      t = t + 1
      select P(t) from P(t-1)
      alter P(t)
      evaluate P(t)
   end
end
```

Figur 3.3 Evolutionary Algorithm – struktur.

För att utveckla en Evolutionary algoritm krävs ett antal avgöranden, vilka beskrivs nedan.

Representation
En lösning representeras vanligen med hjälp av en binär sträng eller med en vektor med hel- eller decimaltal. Detta görs helt enkelt för att de genetiska operatorer som används utförs lättast på strängar [Ansari N. Hou E.]. Traveling salesman problemet representeras till exempel ofta av en heltalsvektor där varje element motsvaras av en stad. Alla städer har ett

unikt heltalsindex som måste vara med i vektorn, och färden beskrivs av den ordning varvid heltalen (städerna) ligger i vektorn.

Skapande av initialpopulation
Det finns två huvudmetoder för att skapa initiallösningar. Den första skapar lösningar fullständigt slumpmässigt medan den andra använder den förhandsinformation som för problemet finns tillgänglig. En kombination är också möjlig. Är till exempel den kortaste vägen mellan några av punkterna i ett traveling salesman problem känd, implementeras dessa medan resten blir slumpmässigt vald.

Genetiska operatorer
Fyra genetiska operationer är vanliga: urval, kombination, mutering och invertering. Det är inte nödvändigt att använda samtliga eftersom varje operation fungerar oberoende av de andra, men för att algoritmen skall bli effektiv bör de tre första användas [Karaboga D and Pham D. T.]

Urval fungerar som det låter. En lösning ger upphov till en ny som är exakt likadan. Ju bättre den är desto fler kopior får den, antigen proportionerligt eller med en viss sannolikhet som då beror på dess kvalité. En dålig lösning får ingen eller har en mycket liten chans att få en kopia.

Kombination innebär att man med hjälp av två eller flera lösningar skapar en ny. Här är kvalitén på ursprungslösningarna avgörande för hur många nya som skapas. För att kunna använda denna operator kan det vara nödvändigt att dela upp en lösning i dellösningar.

Att skapa en ny lösning utifrån en enda kallas mutation. För Traveling Salesman problemet kan till exempel två slumpmässigt valda städer byta plats.

Liksom mutering utförs invertering på en lösning. För Traveling Salesman problemet där representationen är en vektor, väljs ett antal element vilka sedan får omvänd ordning, först blir sist osv.

Kontrollparametrar
Viktiga kontrollparametrar är antal lösningar i populationen, kombinationsfrekvens och muteringsfrekvens. Huvudriktlinjer är att en stor population ökar sannolikheten för att en global lösning påträffas, samtidigt som beräkningstiden stiger. Kombinationsfrekvensen är viktig för hur snabb konvergensen mot en lösning är. Mutationsfrekvensen hjälper evolutions-algoritmen att finna nya sökområden så att lokala optima kan undvikas, samtidigt ökar risken för instabilitet.

Utvärderingsfunktion
Som gränssnitt mellan evolutions-algoritm och optimeringsproblem används en utvärderingsfunktion, vilken beräknar kvalitén av en lösning.

3.3 Tabu search

Tabu search är en typ av iterativ sökning och karakteriseras genom användandet av ett flexibelt minne. Huvudprincipen består i att utifrån en befintlig lösning välja ett antal nya lösningar, och av dessa välja den bästa som sedan ersätter den befintliga, även om den befintliga kanske är den hittills bästa. Med hjälp av en tabu lista lagras vilka lösningar som

accepterats som nya. Detta görs för att undvika cyklingsproblem som kan uppstå genom att sämre lösningar kan accepteras. Med jämna intervall uppdateras den bästa globala lösningen hittills. Strukturen för Tabu Search beskrivs i figur 3.4.

```
procedure tabu search
begin
   tries = 0
   repeat
     initialize a solution (optional when tries >= 1)
     count = 0
     repeat
       identify a set of solutions
       select the best one
       replace the current solution
       update tabu list and other vaiables
       count = count + 1
     until count = ITER
     tries = tries + 1
     if the current solution is the best so far
       then update best global solution
   until tries = MAX-TRIES
end
```

Figur 3.4 Tabu Search – struktur.

Uppdateringen och kontrollen av tabu listan styrs med hjälp av olika så kallade strategier. Dessa strategier är det viktigaste delarna av algoritmen och nedan beskrivs de mest dominerande [Glover F. and Laguna M.].

Forbidding Strategy
Denna strategi används för att avgöra vilka lösningar som inte får accepteras, vilket bland annat innebär att i tabu listan lagras ett visst antal tidigare valda lösningar. Storleken på listan måste avvägas så att risken för cykling blir så liten som möjligt samtidigt som beräkningseffektiviteten bibehålls. När listan är full ersätts den äldsta lösningen med den senaste enligt FIFO-principen.

Aspiration Criteria och Tabu Restrictions
För att inte göra en lovande lösning där cykling kan undgås förbjuden, används Aspiration Criteria. Det krävs då en funktion för utvärdering av lösningen vars resultat kan hindra inträde i tabu listan. Även en funktion som övervakar cykling är nödvändig. Kortfattat så används Aspiration Criteria som en guide för sökningen. Tabu Restrictions däremot används för att begränsa sökområdet, och medför praktiskt att en lösning med alltför låg kvalitet inte kan accepteras.

Learning Strategy
Denna strategi sparar ett visst antal bra lösningar under algoritmens körning, vilka används för att nya lösningar med liknande egenskaper senare skall kunna väljas. Hur många och hur länge en bra lösning skall sparas måste avgöras, liksom hur jämförelsen mellan lösningar skall ske.

Short-Term Strategy

Samverkan mellan samtliga strategier styrs av Short-Term Strategy. Till exempel måste algoritmen kunna avgöra när Learning Strategy går före Aspiration Criteira.

3.4 Kortast tid först

Som tidigare nämnts är denna teknik ingen optimeringsteknik, utan snarare ett problemspecifikt mönster för hur uppdragen skall köras. Genom att första lediga truck utför det uppdrag som har lägst färdigtid, eller det uppdrag som är mest försenat om försening råder, så erhålls en lösning av problemet. Alla uppdrag blir utförda och detta kan ses som en lösning. Hade alla missförstånd och fel som normalt kan uppkomma i ett lager kunnat undvikas, så blir tomkörningstiden lika stor som körningen med gods. I grova drag är det denna princip som idag används.

3.5 Närmsta uppdrag först, med tidsbegränsning

Detta är heller ingen optimeringsteknik utan en metod där första lediga truck utför det uppdrag som ligger närmast. Med närmast menas i detta fallet det uppdrag vars startpunkt på kortast tid kan nås från truckens befintliga position. Med tidsbegränsning innebär att detta närmsta uppdrag måste tillhöra en grupp av uppdrag som samtliga har en färdigtid <= kortast färdigtid + Δt. Är $\Delta t = 0$ blir denna metod identisk med metoden Kortast tid först.

4 Modellering av systemet

Hur systemet skall eller bör modelleras beror till stor del på vilken lösningsteknik som kommer att användas. Problemets karaktär gör det dock mer eller mindre naturlig att använda en viss representation i modelleringen, vilket sedan medför att en viss optimeringsteknik kan vara mer lämplig. Det skall påpekas att i detta skede har strukturer valts så att en praktisk tillämpning av en lösningsmetod senare skall kunna vara möjlig.

Naturliga fysiska komponenter i detta problem är lager, truckarna som arbetar i lagret samt de uppdrag som truckarna utför. Dessa tre komponenter är nödvändiga för att en lösning skall erhållas, och kan dessutom sägas ha ett naturligt modelleringsutseende. Lösningen är sedan den ordning med vilken uppdragen utförs i samt av vilken truck. Nedan beskrivs datastrukturer för truckar, uppdrag, lager och lösning. Dessa har arbetats fram relativt oberoende av optimeringsmetod. Den variabel som används samt hur tilldelning av värde sker beskrivs. Även de konstanter som måste bestämmas förklaras.

De optimeringstekniker som har använts kan ses som länken mellan lösningen och de tre fysiska komponenterna. Själva lösningsstrategin beskrivs, liksom ingående variabler, konstanter och strukturer. Simulated Annealing (SA) använder mer komplicerade samverkande variabler, och är därför mer ingående beskrivna än övriga.

Syftet med detta avsnitt är att ge en bakgrund och bild av hur det modellerade systemet fungerar. Truckar, uppdrag, lager och lösning skrivs under optimeringen ut i filer. Fullständig beräkningsmetodik kan ses i programkoden, appendix B.

4.1 Truckar

Truckarna har den enklaste strukturen av samtliga variabler. Ingen hänsyn tas i truckstrukturen till faktorer som beror på truckföraren. Inte heller det faktum att vissa truckar kanske inte får utföra uppdrag i vissa delar av lagret är medtaget. Dessa faktorer tillsammans med att truckar i sig kan ha olika effektivitet, återspeglas istället i lagerstrukturen.

Struktur

```
struct truck_struct
{
   int id;          //unikt id
   int speed;       //hastighet [m/s]
   int location;    //truckens läge i lagret: 1, 2, .., NUM_PLACE_TOT
};
```

Variabel

```
struct truck_struct trucks[NUM_TRUCKS];
                   //den globala variabeln trucks. en vektor innehållande
                   //lagrets samtliga truckar
```

Konstanter

NUM_PLACE_TOT = antal platser i lagret
NUM_TRUCKS = antal truckar i lagret

TRUCK_SPEED = truckarnas hastighet

Tilldelning

`trucks[i].id` = i = 1, 2, .., NUM_TRUCKS
`trucks[i].speed` = TRUCK_SPEED
`trucks[i].location` = initialt Random(1, NUM_PLACE_TOT), annars den plats trucken kommer att avsluta pågående uppdrag. Random ger ett slumpmässigt tal inom angivet intervall.

4.2 Lager

Lager kan se ut på oerhört många sätt och fungera enligt många olika principer. Dessutom kan de förändras på mycket kort tid och kan därför inte ses som statiska. Det är därför inte rimligt att utgå från ett lagers fysiska utseende vid modellering. Vad som ytterligare komplicerar problemet är att lagrets utseende avgör eller avgörs av vilka truckar som används. Enligt tidigare skall lagerstrukturen även återspegla begränsningar och individuella skillnader för de truckar som används. För att tillgodose dessa krav används en nedan beskriven struktur.

Struktur

```
struct storage_struct
{
   int time[1+ NUM_TRUCKS][ NUM_PLACE_TOT][ NUM_PLACE_TOT];
                    //första matrisindex = 0: avstånd [m] mellan två
                    // punkter i lagret
                    //första matrisindex > 0: tid [s] det tar för en truck
                    //att köra  mellan två punkter i lagret
};
```

Variabel

```
struct storage_struct storage;
                    //den globala variabeln storage. en tredimensionell
                    //matris innehållande samtliga truckars tider för att
                    //färdas från en punkt till en annan
```

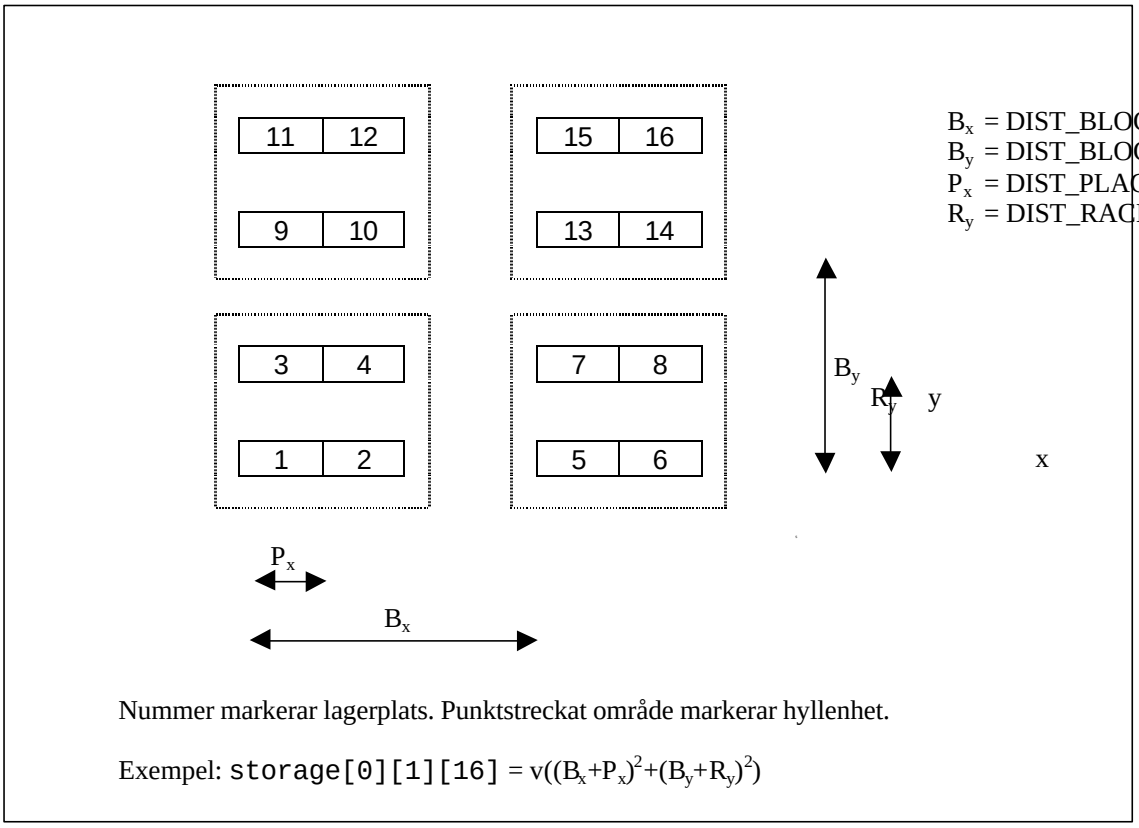
Konstanter

NUM_BLOCK_X = antal hyllenheter i x-led
NUM_BLOCK_Y = antal hyllenheter i y-led
DIST_BLOCK_X = avstånd [m] mellan hyllenheter i x-led
DIST_BLOCK_Y = avstånd [m] mellan hyllenheter i y-led
DIST_RACK_Y = avstånd [m] mellan hyllor i y-led
NUM_PLACE_X = antal platser i x-led
DIST_PLACE_X = avstånd [m] mellan platser i x-led
NUM_PLACE_TOT = antal platser i lagret
= NUM_BLOCK_X*NUM_BLOCK_Y*2*NUM_PLACE_X
NUM_BLOCKS = antal hyllenheter
= NUM_BLOCK_X*NUM_BLOCK_Y

Tilldelning

`storage[0][j][k]` = se figur 4.1 nedan
`storage[i][j][k]` = storage[0][j][k] / trucks[i].speed

i = 1, 2, .., NUM_TRUCKS
j = k = 1, 2, .., NUM_PLACE_TOT



Figur 4.1 Lageruppbyggnad.

Övrigt

De avstånd och tider som finns i lagermatrisen baseras på färdsträckor fågelvägen mellan två punkter, och tar inte hänsyn till väggar eller andra orsaker som omöjliggör en sådan körväg. Syftet med lagermatrisen är som tidigare påpekats endast att ge optimeringen data för beräkningar och jämförelser. Att lagret inte ser fullständigt verklighetstroget ut är för utvärdering av optimeringsteknik inte relevant. Hur lagermatrisen i verkligen kan skapas beskrivs i kapitel 5, där ingen lageruppbyggnad enligt figur 4.1 är nödvändig. Hur matrisen avläses tydliggörs enligt nedan:

`storage[0][4][12]` = avstånd i meter från plats 4 till 12
`storage[2][4][12]` = tid i sekunder det tar för truck nummer 2 att åka från plats 4 till 12, negativt värde anger att truck nummer 2 ej får åka från plats 4 till 12
`storage[2][12][4]` = tid i sekunder det tar för truck nummer 2 att åka från plats 12 till 4, negativt värde anger att truck nummer 2 ej får åka från plats 12 till 4

4.3 Uppdrag

Uppdragen i optimeringen lagras i en länkad lista. Varje uppdrag är unikt och dess viktigaste data är var det startar respektive slutar samt när det skall vara färdigt. Listan sorteras efter den ordning i vilken uppdragen skall vara färdiga. Det uppdrag som har kortast tid på sig för att bli färdigt ligger först. För att uppnå effekter som avdelning för inkommande och utgående gods, vilka har en hög uppdragsfrekvens, kan man i optimeringen låta ett stort antal uppdrag uppstå med samma start eller slutpunkt. Detta för att ge lagersystemet ett mer naturligt utseende.

Uppdragsstrukturen används även av lösningsstrukturen, därför verkar en del variabler vara onödiga. Till exempel är från början inte uppdragets verkliga färdigtid känd.

Struktur

```
struct task_struct
{
   struct task_struct *next_ptr;
                    //pekare till nästa uppdrag
   int id;          //unikt id
   int from;        //plats där uppdraget startar
   int to;          //plats där uppdraget slutar
   int from_block;  //hyllenhet där uppdraget startar
   int to_block;    //hyllenhet där uppdraget slutar
   int last_time;   //uppdragets sista färdigtid
   int ready_time;  //uppdragets verkliga färdigtid
   int late_time;   //uppdragets förseningstid
};
```

Variabel

```
struct task_struct *tasks_ptr;
                    //den globala variabeln tasks_ptr. en pekare på det
                    //första uppdraget i uppdragslistan
```

Konstanter

TOT_NUM_TASKS	=	totalt antal uppdrag som skapas
IN_PERCENT	=	sannolikhet att uppdraget startar på den första lagerplatsen
OUT_PERCENT	=	sannolikhet att uppdraget slutar på den sista lagerplatsen
MIN_LAST_TIME	=	minsta sista färdigtid
MAX_LAST_TIME	=	största sista färdigtid

Tilldelning

`(*tasks_ptr).next_ptr`	=	nästföljande uppdrag
`(*tasks_ptr).id`	=	1, 2, .., TOT_NUM_TASKS
`(*tasks_ptr).from`	=	1 (första lagerplatsen) med en sannolikhet av IN_PERCENT, annars Random(2, NUM_PLACE_TOT)
`(*tasks_ptr).to`	=	NUM_PLACE_TOT med en sannolikhet av OUT_PERCENT, annars Random(1, NUM_PLACE_TOT-1)
`(*tasks_ptr).from_block`	=	Find_Block(`(*tasks_ptr).from`), Find_Block ger den hyllenhet platsen tillhör

```
(*tasks_ptr).to_block   = Find_Block((*tasks_ptr).to)
(*tasks_ptr).last_time  = Random(MIN_LAST_TIME, MAX_LAST_TIME)
(*tasks_ptr).ready_time = beräknas i lösningen
(*tasks_ptr).late_time  = beräknas i lösningen enligt
                          (*tasks_ptr).last_time - (*tasks_ptr).ready_time
```

4.4 Lösning

Denna struktur har en länkad lista för varje truck innehållande de uppdrag som trucken utfört. När ett uppdrag under optimeringen är färdigt lagras det sist i den lista det tillhör. När det antal uppdrag som skall utföras är färdiga beräknas några variabler som beskiver hur effektivt uppdragen utförts.

Struktur

```
struct result_struct
{
  struct task_struct *task_ptr[NUM_TRUCKS];
                  //pekare till en lista med de uppdrag en truck utfört
  int total_travel_time;
                  //total arbetstid för samtliga truckar
  int prcnt_load; //andel av total arbetstid då körning med artikel skett
  int num_late;   //antal försenade uppdrag
  int aver_late;  //genomsnittlig förseningstid för försenade uppdrag
};
```

Variabel

```
struct result_struct result;
                  //den globala variabeln result. innehåller resultatdata
                  //och pekare till de uppdrag varje truck utfört
```

Tilldelning

Se figur 4.2 för hur tilldelning sker.

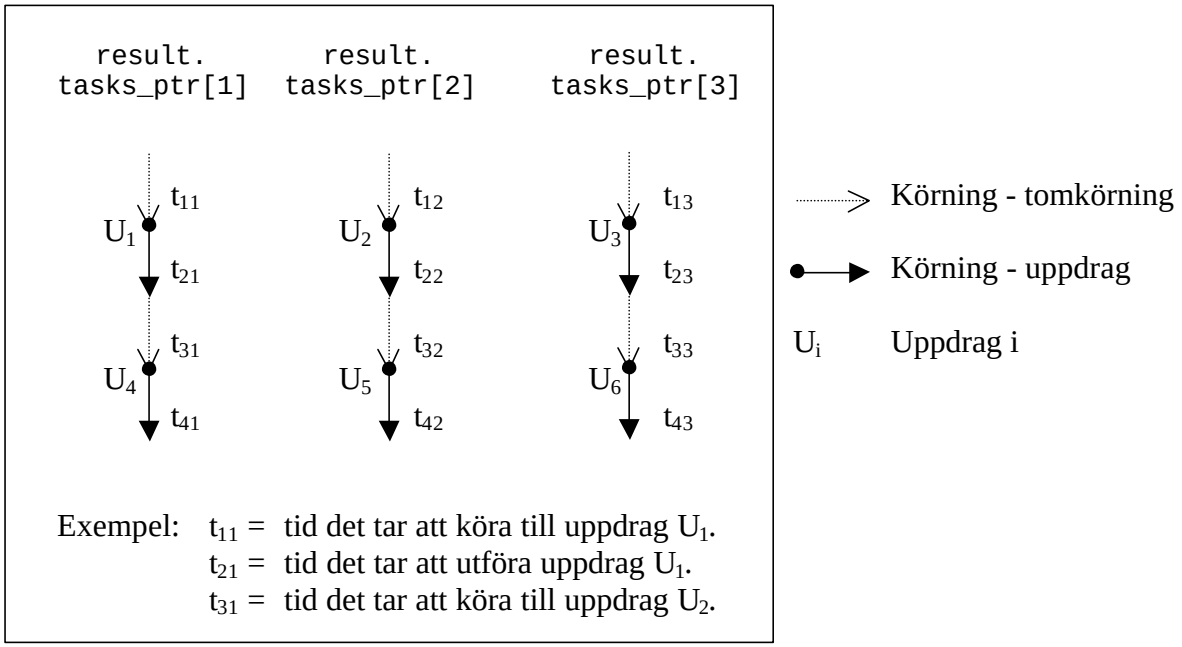
`result.tasks_ptr[i]` = första utförda uppdrag

`result.total_travel_time` = $\Sigma[i=1, ..,4] (\Sigma[j=1,..3] t_{i,j})$

`result.prcnt_load` = $\Sigma[i=2,4] (\Sigma[j=1,..3] t_{i,j})$ / `result.total_travel_time`

`result.num_late` = Σ antal $t_{i,j} < 0$

`result.aver_late` = $\Sigma[i=1, ..,4] (\Sigma[j=1,..3] t_{i,j})$ / `result.num_late`, för $t_{i,j} < 0$



Figur 4.2 Resultatberäkning – 3 truckar utför 6 uppdrag.

Övrigt

Den variabel som är av störst intresse är `result.prcnt_load`, vilken är detsamma som truckarnas totala utnyttjandegrad som nämndes under avsnittet förbättringsmöjligheter. Ju högre värde desto bättre lösning. En mycket viktig faktor i ett lager är att inga uppdrag utförs för sent. För att ha kontroll över förseningar används `result.num_late` och `result.aver_late_time`.

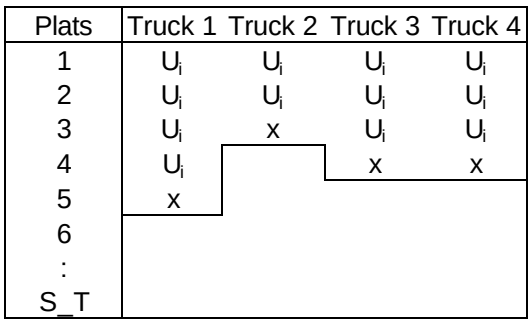
4.5 Val av teknik

Anledningen till att SA har valts som optimeringsteknik är framförallt enkelhet, att algoritmen är välkänd och lätt att förstå och att 2-interchange lätt kan tillämpas. Användbarhet måste prioriteras för att metoden överhuvudtaget skall användas och det är därför inte realistiskt att utveckla en alltför avancerad metod. Viktigt är också att metoden skall kunna arbeta i realtid, varför man inte kan räkna med att finna den verkligt optimala lösningen. Det finns det helt enkelt inte beräkningstid för.

4.5.1 Simulated Annealing

Vad som framförallt krävs av de variabler och strukturer som SA använder är att beräkningsoperationer skall kunna utföras snabbt. Matris eller vektorform där data snabbt kan hämtas med hjälp av index har därför använts. Med tanke på beräkningshastigheten har heltal (int) genomgående används istället för flyttal (float), vilket ger snabbare beräkningar.

Figur 4.3 illustrerar grovt hur den variabel eller struktur som SA arbetar med ser ut. Den kan inte innehålla samtliga uppdrag som väntar på att utföras på grund av att detta hade gett ett alltför stort sökområde. Optimeringen utförs på ett användardefinierat genomsnittligt antal lösningar per truck. Detta är i sig en klar begränsning men nödvändig med tanke på att tekniken skall arbeta i realtid.



U_i = Uppdrag i
i = 1, .., TOT_NUM_TASKS
x = Tom plats för uppdragsbyte
S_T = START_TASKS_IN_SA
(Antal uppdrag i SA-variabeln)

Figur 4.3 Optimering – 4 truckar, 3 uppdrag per truck (genomsnitt).

Varje uppdrag innehåller unik information. Bland annat id, var uppdraget börjar respektive slutar, när det skall vara färdigt, tid det tar att köra till uppdraget från föregående, tid det tar att utföra själva uppdraget samt tid det tar att köra till nästa uppdrag. Dessa tider beror på vilken truck som utför uppdraget. Samtliga uppdrag tar en viss tid att utföra och kan medföra en viss förseningstid. Se `sa_struct` nedan för utförligare beskrivning.

För att optimera den tid det tar att utföra uppdragen tillämpas 2-interchange, vilket innebär att effekten av att två uppdrag byter plats beräknas. Ett uppdrag tillåts också flyttas till en tom plats. Innebär platsbytet en kvalitetsförbättring så utförs bytet. Innebär bytet en försämring så sker ett platsbyte med en viss sannolikhet. Se `interchange_struct` nedan för utförligare beskrivning. Kvalitén, eller rättare sagt kvalitetsförändringen, beräknas enligt:

```
ic.diff_quality  = (ic.diff_total_time * TOTAL_TIME_FACT
                 + ic.diff_max * MAX_TIME_FACT
                 + ic.diff_total_late * TOTAL_LATE_FACT)
                 / (MAX_TIME_FACT + TOTAL_TIME_FACT + TOTAL_LATE_FACT)
```

Att kvalitén inte enbart beror på den totala tid det tar att utföra ett antal uppdrag beror på att det kan uppstå situationer där ett uppdrag ständigt ”halkar ner” i SA’s struktur. För att tvinga upp försenade uppdrag krävs ett hänsynstagande av försenade uppdrag. Detta gynnar inte den totala färdigtiden men är alltså nödvändig. Att en maxtid finns med i kvalitetsbedömningen är för att en jämn fördelning av uppdragen på de olika truckarna skall erhållas.

Observera att negativt värde på kvalitetsförändringen innebär en förbättring och ett positivt en försämring. Sannolikheten för platsbyte som innebär en försämring beräknas enligt:

```
p = exp(-ic.diff_quality/T)
```

Innebär ett platsbyte varken en försämring eller förbättring, accepteras den nya med en användardefinierad sannolikhet.

Multiplikationsfaktorerna `TOTAL_TIME_FACT`, `MAX_TIME_FACT` och `TOTAL_LATE_FACT` används för att i kvalitetsbedömningen få en lämplig avvägning mellan totala körtiden, maximala tiden samt totala förseningstiden.

Ett uppdrag är färdigt när SA's båda loopar är utförda, och plockas då ut ur SA-variabeln och sätts in i lösningen. I samma ögonblick hämtas det ett nytt uppdrag från uppdragslistan och sätts in i SA-variabeln, temperaturen återfår sitt startvärde osv. Temperaturförändringen beräknas enligt α-metoden, vilken beskrevs i avsnitt 3.1.2.

Strukturer

```
struct sa_truck_struct
{
   int truck;       //id nummer för truck
   int time;        //en tid för trucken[s]
};
struct sa_struct
{
   int index;       //unikt id nummer
   int total_time;  //total tid att utföra uppdragen i SA-structure [s]
   int total_late;  //total förseningstid för uppdragen i SA-structure [s]
   int quality;     //kvalitén på SA-structure
   struct sa_truck_struct max_truck;
                    //den truck som sist är färdig med sina uppdrag i SA-
                    //structure
   struct sa_truck_struct first_truck;
                    //den truck som först är färdig med pågående uppdrag
   int num_tasks[NUM_TRUCKS];
                    //hur många uppdrag varje truck tilldelats
   int id[NUM_TRUCKS][START_TASKS_IN_SA];
                    //id nummer för uppdrag
   int from[NUM_TRUCKS][START_TASKS_IN_SA];
                    //plats där uppdraget startar
   int to[NUM_TRUCKS][START_TASKS_IN_SA];
                    //plats där uppdraget slutar
   int from_block[NUM_TRUCKS][START_TASKS_IN_SA];
                    //hyllenhet där uppdraget startar
   int to_block[NUM_TRUCKS][START_TASKS_IN_SA];
                    //hyllenhet där uppdraget slutar
   int before_time[NUM_TRUCKS][START_TASKS_IN_SA];
                    //tid att färdas från föregående uppdrag [s]
   int own_time[NUM_TRUCKS][START_TASKS_IN_SA];
                    //tid att utföra själva uppdraget [s]
   int after_time[NUM_TRUCKS][START_TASKS_IN_SA];
                    //tid att färdas till nästa uppdrag [s]
   int last_time[NUM_TRUCKS][START_TASKS_IN_SA];
                    //uppdragets sista färdigtid [s]
   int ready_time[NUM_TRUCKS][START_TASKS_IN_SA];
                    //uppdragets verkliga färdigtid [s]
   int late_time[NUM_TRUCKS][START_TASKS_IN_SA];
                    //uppdragets förseningstid
};
struct change_struct
{
   int truck;       //id nummer för truck
   int task;        //id nummer för uppdrag
   int before_time; //tid att färdas från föregående uppdrag [s]
   int own_time;    //tid att utföra själva uppdraget [s]
   int after_time;  //tid att färdas till nästa uppdrag [s]
   int ready_time;  //uppdragets verkliga färdigtid [s]
   int diff_time;   //hur mycket tidigare/senare uppdraget kommer att vara
                    //färdiga då 2-interchange (2-i) utförs
};
```

```
struct interchange_struct
{
   int diff_max;    //effekt på maxtiden då 2-i utförs
   int diff_total_time;
                    //effekt på totala tiden då 2-i utförs
   int diff_total_late;
                    //effekt på total förseningstid då 2-i utförs
   int diff_quality;//effekt på kvalitén då 2-i utförs
   struct change_struct i1;
                    //första uppdraget i 2-i
   struct change_struct i2;
                    //andra uppdraget i 2-i
};
```

Variabel

```
struct sa_struct sa[2];
                    //den globala variabeln sa[2]. innehåller de uppdrag och
                    //truckar som SA utförs på
struct interchange_struct ic;
                    //den globala variabeln 2-i. används av SA
```

Konstanter

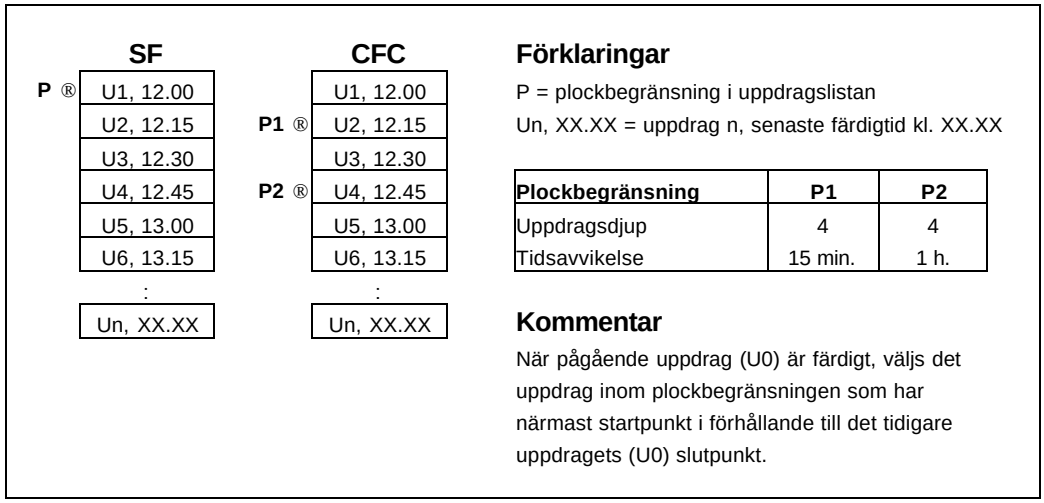
Flera konstanter styr SA, till exempel starttemperatur, temperatursänkning med mera. Vilka värde dessa har haft anges under avsnittet resultat och utvärdering.

NUM_TASKS_PER_TRUCK	=	antal uppdrag per truck i SA-structure (i genomsnitt)
MAX_TIME_FACT	=	multiplikationsfaktor för `sa[i].max_truck.time`
TOTAL_TIME_FACT	=	multiplikationsfaktor för `sa[i].total_time`
TOTAL_LATE_FACT	=	multiplikationsfaktor för `sa[i].total_late`
INNER_LOOP	=	inre loop i SA
OUTER_LOOP	=	yttre loop i SA
START_TEMP	=	starttemperatur för SA
ALPHA	=	temperatursänkningsfaktor i SA
START_TASKS_IN_SA	=	NUM_TASKS_PER_TRUCK*NUM_TRUCKS+1

i = 1, 2

4.5.2 Kortast tid först - Närmsta uppdrag först, med tidsbegränsning

Tekniken ”Kortast tid först” (SF) innebär att när en truck är färdig med ett uppdrag påbörjas det uppdrag som ligger först i uppdragslistan, det vill säga det uppdrag som har kortast tid på sig att bli färdigt. ”Närmsta uppdrag först, med tidsbegränsning” (CFC) innebär att det uppdrag vars startpunkt det tar kortast tid att köra till påbörjas först. Detta under förutsättning att uppdraget tillhör ett bestämt antal av de första uppdragen och att dess sista färdigtid maximalt avviker en viss absolut tidsenhet från det uppdrag som har kortast färdigtid. Teknikerna för uppdragstilldelning illustreras i figur 4.4.



Figur 4.4 SF-CFC, uppdragstilldelning.

Observera att om uppdragsdjup eller tidsavvikelse sätts till 0 för CFC, så är CFC ekvivalent med SF. SF är alltså ett specialfall av CFC och samma struktur och variabel har därför använts. Det är också möjligt att sätta antingen uppdragsdjup eller tidsavvikelse till oändlighet så att enbart en plockbegränsning definierar uppdragsdjupet för CFC.

Struktur

```
struct sf_cfc_struct
{
   int truck_location[NUM_TRUCKS];
                  //plats i lagret där trucken kommer att avsluta pågående
                  //uppdrag
   struct task_struct task[NUM_TRUCKS];
                  //id nummer för pågående uppdrag
   struct opt_truck_struct first_truck;
                  //den truck som först är färdig med pågående uppdrag
};
```

Variabel

```
struct sf_cfc_struct sf_cfc;
                  //den globala variabeln sf_cfc. innehåller truckarna och
                  //de uppdrag de utför, samt den truck som först blir
                  //färdig
```

Konstanter

TASK_DEPTH = maximalt plockdjup i globala variabeln `tasks_ptr` när nytt uppdrag väljs

TIME_DEVIATION = maximal tidsavvikelse valt uppdrag får ha i relation till det första uppdraget i `tasks_ptr`

5 Resultat och utvärdering

För att utvärdera hur effektiv en optimeringsmetod är har följande värden beräknats och redovisats:

- Total tid att utföra samtliga uppdrag
- Utnyttjandegrad [%]
- Antal försenade uppdrag
- Genomsnittlig förseningstid för försenade uppdrag
- Beräkningstid (total) för datorn [s]

Den totala tiden att utföra samtliga uppdrag samt den genomsnittliga förseningstiden för försenade uppdrag anges utan tidsenhet. Tiderna är rent teoretiska men står i proportion till varandra och den tid det tar att köra från en plats i lagret till en annan. Beräkningstiden för datorn är reell och anges i sekunder.

Den resultatsiffra som är mest intressant är utnyttjandegraden. Är den hög erhålls en låg total tid och få försenade uppdrag. Antal försenade uppdrag är dock en mycket väsentlig kvalitetsfaktor som bör övervakas och redovisas, liksom total tid.

I de fall då Simulated Annealing (SA) används som optimeringsteknik redovisas även initial acceptansgrad för sämre lösningar [%] samt slutlig acceptansgrad för sämre lösningar [%], vilket krävs för att kunna uppskatta om vald starttemperatur och temperaturförändring är lämplig.

5.1 Testfall

Optimeringarna har utförts på tre olika fall för jämförelse mellan metoderna samt ett fall för utvärdering av olika variablers inverkan. De fall som har utvärderats ser ut enligt följande:

Variabelinställning
Ett lager med 120 platser där 3 truckar utför 250 uppdrag. Detta fall har använts för att komma fram till en lämplig inställning av SA's variabler.

Litet lager
Ett lager med 120 platser där 3 truckar utför 750 uppdrag.

Medelstort lager
Ett lager med 480 platser där 12 truckar utför 3000 uppdrag.

Stort lager
Ett lager med 480 platser där 36 truckar utför 9000 uppdrag.

Det kan påpekas att ett stort lager i verkligheten har mycket fler än 480 platser. För att datamängden inte skall bli för stor kan ett lager med till exempel 40000 platser inte användas. I detta sammanhang kan det tänkas att varje plats egentligen refererar till 80 platser bredvid varandra.

Användardefinierbara konstanter för lager, truckar och uppdrag har tilldelats värden enligt tabell 5.1.

Lagerkonstanter	Variabel-inställning	Litet lager	Medelstort lager	Stort lager
NUM_BLOCK_X	2	2	4	4
NUM_BLOCK_Y	3	3	6	6
DIST_BLOCK_X	45	45	45	45
DIST_BLOCK_Y	20	20	20	20
DIST_RACK_Y	5	5	5	5
NUM_PLACE_X	10	10	10	10
DIST_PLACE_X	4	4	4	4

Truckkonstanter	Variabel-inställning	Litet lager	Medelstort lager	Stort lager
NUM_TRUCKS	3	3	12	36
TRUCK_SPEED	3	3	3	3

Uppdragskonstanter	Variabel-inställning	Litet lager	Medelstort lager	Stort lager
DO_NUM_TASKS	250	750	3000	9000
MIN_LAST_TIME	100	100	100	100
MAX_LAST_TIME	2550	7000	15000	15000
IN_PERCENT	25	25	25	25
OUT_PERCENT	25	25	25	25

Tabell 5.1 Inställning för lager, truckar och uppdrag.

5.2 Variabelinställning

Simulated Annealing
SA har en mängd variabler som skall bestämmas. Genom att variera en variabel i taget har en bild skapats av hur resultatet påverkas. Efter varje undersökning av respektive variabel återställs den till ett värde enligt tabell 5.2, innan nästa variabel undersöks. Hur variablerna påverkar resultatet redovisas i appendix A.

Grundinställningen av variablerna har delvis uppkommit experimentellt. NUM_TASKS_PER_TRUCK, INNER_LOOP och OUTER_LOOP har fått sina värden av att en passande beräkningstid erhålls. MAX_TIME_FACT, TOTAL_TIME_FACT och TOTAL_LATE_FACT har tilldelats värden för att erhålla en så hög utnyttjandegrad som möjligt. START_TEMP och ALPHA ger tillsammans lämplig initial och slutlig acceptans för sämre lösningar. Resultatet från variabelgrundinställningen visas i tabell 5.3.

Variabel	Värde
NUM_TASKS_PER_TRUCK	6
MAX_TIME_FACT	10
TOTAL_TIME_FACT	200
TOTAL_LATE_FACT	20
INNER_LOOP	100
OUTER_LOOP	100
START_TEMP	20
ALPHA	0.95

Tabell 5.2 Variabelgrundinställning för SA.

Resultat	Värde
Total tid för samtliga uppdrag [tidsenhet]	5819
Utnyttjandegrad [%]	63
Antal försenade uppdrag	33
Genomsnittlig förseningstid	27
Beräkningstid [s]	15
Initial acceptansgrad för sämre lösningar [%]	61
Slutlig acceptansgrad för sämre lösningar [%]	0

Tabell 5.3 Resultat - Variabelgrundinställning för SA.

Efter att ha studerat resultatet av de olika variablernas inverkan väljs en inställning enligt tabell 5.4. Denna inställning är den som gett bäst utnyttjandegrad för testfallet variabelinställning och resultatet visas i tabell 5.5. Att denna inställning alltid för samtliga testfall är den bästa är inte sannolikt, men att för varje fall mer eller mindre experimentellt finna optimal inställning är tidsödande och innebär heller inte någon avsevärd resultatförbättring.

Variabel	Värde
NUM_TASKS_PER_TRUCK	7
MAX_TIME_FACT	5
TOTAL_TIME_FACT	200
TOTAL_LATE_FACT	5
INNER_LOOP	200
OUTER_LOOP	200
START_TEMP	15
ALPHA	0.96

Tabell 5.4 Bästa funna variabelinställning för SA.

Resultat	Värde
Total tid för samtliga uppdrag [tidsenhet]	5568
Utnyttjandegrad [%]	66
Antal försenade uppdrag	37
Genomsnittlig förseningstid	48
Beräkningstid [s]	58
Initial acceptansgrad för sämre lösningar [%]	44
Slutlig acceptansgrad för sämre lösningar [%]	0

Tabell 5.5 Resultat - Bästa funna variabelinställning för SA.

Kortast tid först
Denna metod (SF) har ingen variabelinställning. Teoretiskt skall denna teknik ge en utnyttjandegrad på 50%. Eventuella avvikelser från detta värde beror på avrundningsfel i uträkningarna.

Närmsta uppdrag först, med tidsbegränsning
För att effektivt kunna jämföra denna teknik (CFC) med SA har enbart uppdragsdjupet använts som plockbegränsning. Uppdragsdjupet sätts till samma värde som antalet uppdrag i SA-strukturen. På så sätt tillåts båda metoderna ha samma framförhållning vid val av uppdrag som skall utföras. Teoretisk kan man i ett ändligt lager erhålla en utnyttjandegrad på 100% om man har ett obegränsat antal uppdrag att välja på och om uppdragsdjupet täcker samtliga uppdrag. När ett uppdrag avslutas på en plats finns då alltid ett som påbörjas på samma plats.

Enligt tidigare beräknas alltså uppdragsdjupet enligt:

NUM_TASKS_PER_TRUCK*NUM_TRUCKS+1

Detta ger för de olika testfallen uppdragsdjup enligt tabell 5.6

Variabel	Litet lager	Medelstort lager	Stort lager
Plockdjup	22	85	253

Tabell 5.6 CFC – Plockdjupsinställning.

5.3 Resultat

Resultaten optimeringarna ger för respektive testfall visas i tabell 5.7. Den observante kan se att utnyttjandegraden multiplicerat med total tid inte blir exakt samma för de olika lagerna. Detta beror på att heltalsräkningar genomgående gjorts, vilket har inneburit avrundningar, och skillnaderna ligger inom denna felmarginal.

Lagerstorlek	Litet			Medelstort			Stort		
Resultat	**SF**	**CFC**	**SA**	**SF**	**CFC**	**SA**	**SF**	**CFC**	**SA**
Total tid för samtliga uppdrag	21798	18396	16791	191552	164275	148922	573690	490385	445827
Utnyttjandegrad [%]	49	58	64	49	57	64	49	57	64
Antal försenade uppdrag	597	169	0	2629	675	20	8405	2077	75
Genomsnittlig förseningstid	412	213	0	839	775	66	705	641	57
Beräkningstid [s]	0	0	176	0	0	918	0	0	7030
Initial acceptansgrad för sämre lösningar [%]	-	-	44	-	-	19	-	-	15
Slutlig acceptansgrad för sämre lösningar [%]	-	-	0	-	-	0	-	-	0

Tabell 5.7 Resultat – Samtliga optimeringstekniker.

För att illustrera vad en högre utnyttjandegrad i praktiken innebär beräknas det minskade behovet av truckar. Resultatet av detta visas i tabell 5.8. Metoden kortast tid först representerar här ett truckbehov på 100%.

	SF	**CFC**	**SA**
Utnyttjandegrad [%]	49	57	64
Truckbehov [%]	100	86	77
Minskat truckbehov [%]	0	14	23

Tabell 5.8 Resultat – Minskat truckbehov.

Det framgår tydligt att det för den modell som i detta arbete utvecklats är optimering med hjälp av SA klart bäst. Beräkningstiden det tar att utföra optimeringarna är i samtliga fall mycket låg, mindre än 1 sekund per uppdrag. Ett minskat truckbehov kan även omsättas i en kapacitetsökning, vilket för CFC och SA i relation till SF skulle innebära ökningar på 8% respektive 15 %

5.4 Varför skall en optimeringsteknik användas

Utnyttjandegraden ökar
Den mest positiva effekten av en optimering är att utnyttjandegraden ökar markant. Detta kan också tolkas som om truckbehovet minskar eller att hanteringskapaciteten ökar. Naturligtvis är resultaten i detta arbete teoretiska och förenklingar har gjorts, vilket är en brist. Å andra sidan finns det möjligheter att förfina och utveckla optimeringstekniken ytterligare. Sammantaget finns det inget som pekar på att en avsevärd ökning av utnyttjandegraden inte kan uppnås.

Goda grundförutsättningar
Idag används ingen optimering, vilket gör grundförutsättningarna för införandet av en mycket goda. Det är i princip omöjligt att inte åstadkomma en förbättring.

Kommer att tillämpas i framtiden
Många speditionsfirmor använder sig idag av optimeringstekniker för att bestämma lämpliga körvägar. Det är högst troligt att samma utveckling kommer att ske inom området lagerhantering. Så frågan om en optimeringsteknik skall utvecklas nu eller senare, bör ställas.

Flexibiliteten finns
En optimeringsmetod behöver inte vara styrande. Den kan arbeta som en rådgivare där det uppdrag som härnäst bör utföras presenteras. Att införa en optimering behöver därför inte tillföra något osäkerhetsmoment även om det är ny teknik det handlar om. Vidare kan optimeringstekniker kontinuerligt utvecklas, förfinas eller bytas ut. Den viktigaste komponenten, data om hur lång tid det tar att åka från en punkt till en annan, är oberoende av optimeringsteknik.

5.5 Hur kan en optimeringsteknik implementeras

Lagermatris
Gemensamt för alla optimeringstekniker som skall lösa denna typ av problem är att information om hur lång tid det tar att åka från en punkt till en annan måste finnas tillgänglig. Denna information kan i realiteten skapas genom att varje uppdrag registreras när det påbörjas och avslutas. På så sätt erhålls en sekvens med tider vilka samtliga är förknippade med fysiska enheter, vilka kan vara allt ifrån en enskild plats till en hel lagersektion. Med hjälp av en sådan obruten sekvens kan tider det tar att åka från en punkt till en annan beräknas för att sedan lagras i en form av tidsmatris. Detta är en process som kan arbeta osynligt i bakgrunden utan att en användare märker något.

Svårigheten med lagermatrisen består i att definiera vilka platser i lagret som skall tillhöra en specifik plats i lagermatrisen. För ett stort lager måste flera platser samlas ihop till större enheter, annars blir den nödvändiga datamängden för stor. Att skapa en prototyp för lagermatrisen samt undersöka hur denna verkligen kan skapas är ett av de viktigaste stegen i ett fortsatt arbete inom detta område.

Optimeringsteknik
Har en tidsmatris väl skapats så kan ett gränssnitt mot en optimeringsmetod hantera informationsutbytet. På så sätt kan sedan en optimeringsmetod uppdateras eller helt ersättas med en annan.

Under informationsutbytet är det möjligt att manipulera den hämtade tiden och på så sätt gynna eller missgynna ett uppdrag i optimeringen. Detta kan vara intressant vid försenade uppdrag samt för uppdrag som utförs i lagrets periferi.

Vad som är oerhört viktigt att påpeka är att SA är endast en metod för att lösa detta typ av problem. De resultat som framkommit kan med mycket stor sannolikhet överträffas av både annorlunda utformade SA-metoder samt av helt andra optimeringstekniker. Dessutom pågår det en ständig forskning inom området och nya rön tillkommer ständigt. Det kan därför vara önskvärt att inte binda sig vid en viss teknik vid ett införande av en optimeringsmetod. Att börja med en metod som SFC är enkelt och kan tjäna som utvärdering för fortsatt arbete.

5.6 Nackdelar med en optimeringsteknik

Kostnaden
Att införa ny teknik innebär alltid en viss osäkerhet om huruvida investeringen blir lyckad eller ej. Resurser är alltid begränsade och det kan finnas mer prioriterade områden.

Resultaten uppkommer efterhand
Det säger sig själv att ett minskad truckbehov eller en ökad kapacitet uppkommer efter hand eftersom en tidsmatris tar tid att skapa. Optimering kan alltså initialt inte användas, varken i ett nytt eller gammalt lagersystem.

Tidsberoendet
En nackdel är att ju längre tid en optimering har på sig att utvärdera ett och samma tillstånd, ju bättre blir resultatet. Det går alltså inte att ge ett säkert besked på hur mycket utnyttjandegraden kan ökas. Varje unikt lager får sin unika ökning av utnyttjandegraden.

6 Källförteckning

http://www.masystem.com

Ansari N. Hou E (1997). *Computational Intelligence for Optimization*. Kluwer Academic Publishers, Norwell, Massachusetts.

Gendreau M. (1994). A Tabu Search Heuristic for the Vehicle Routing Problem. *Management Science*, Vol. 40, sid. 1276-1290.

Glover F. and Laguna M. (1997). *Tabu Search*. Kluwer Academic Publishers, Norwell, Massachusetts.

Karaboga D and Pham D. T. (2000). *Intelligent Optimization Techniques*. Springer-Verlag, Berlin

Michalewicz Z. (2000). *How to solve it: Modern Heuristics*. Springer-Verlag, Berlin.

Rayward-Smith,V.J et al. (1996). *Modern Heuristic Search Methods*. John Wiley & Sons, Chichester, England.

Rego C. (1998). A Subpath Ejection Method for the Vehicle Routing Problem. *Management Science*, Vol. 44, sid. 1447-1459.

Sariklis D. and Powell S. (2000). A heuristic method for the open vehicle routing problem. *Journal of the Operational Research Society*, Vol.51, sid. 564-573.

Taillard E. D. (1996). Vehicle Routing with Multiple Use of Vehicles. *Journal of the Operational Research Society*, Vol.47, sid. 1065-1070.

Följande mjukvara har använts:

- Bloodshed Dev-C++. Freeware nerladdat från http://bloodshed.net
- Microsoft® Word 2000
- Microsoft® Excel 2000

Appendix A – SA-variabler

NUM_TASKS_PER_TRUCK	4	5	6	7	8
Total tid för samtliga uppdrag	5870	5792	5819	5676	5755
Utnyttjandegrad [%]	63	63	63	65	64
Antal försenade uppdrag	31	33	33	18	54
Genomsnittlig förseningstid	24	23	27	24	31
Beräkningstid [s]	14	15	15	16	17
Initial acceptansgrad för sämre lösningar [%]	64	61	61	58	56
Slutlig acceptansgrad för sämre lösningar [%]	0	0	0	0	0

MAX_TIME_FACT	1	5	10	15	20
Total tid för samtliga uppdrag	5788	5739	5819	5740	5755
Utnyttjandegrad [%]	63	64	63	64	63
Antal försenade uppdrag	48	24	33	29	26
Genomsnittlig förseningstid	25	26	27	21	30
Beräkningstid [s]	15	15	15	15	15
Initial acceptansgrad för sämre lösningar [%]	60	60	61	60	62
Slutlig acceptansgrad för sämre lösningar [%]	0	0	0	0	0

TOTAL_TIME_FACT	50	100	200	400	600
Total tid för samtliga uppdrag	6015	5814	5819	5778	5748
Utnyttjandegrad [%]	61	63	63	64	64
Antal försenade uppdrag	113	33	33	36	40
Genomsnittlig förseningstid	41	19	27	37	44
Beräkningstid [s]	15	15	15	15	15
Initial acceptansgrad för sämre lösningar [%]	54	58	61	60	61
Slutlig acceptansgrad för sämre lösningar [%]	0	0	0	0	0

TOTAL_LATE_FACT	5	10	20	30	40
Total tid för samtliga uppdrag	5677	5724	5819	5856	5820
Utnyttjandegrad [%]	64	64	63	63	63
Antal försenade uppdrag	50	33	33	59	32
Genomsnittlig förseningstid	55	29	27	22	17
Beräkningstid [s]	15	15	15	15	15
Initial acceptansgrad för sämre lösningar [%]	61	61	61	60	57
Slutlig acceptansgrad för sämre lösningar [%]	0	0	0	0	0

INNER_LOOP	10	50	100	200	400
Total tid för samtliga uppdrag	5826	5832	5819	5708	5782
Utnyttjandegrad [%]	63	63	63	64	63
Antal försenade uppdrag	68	47	33	24	32
Genomsnittlig förseningstid	32	26	27	22	24
Beräkningstid [s]	1	7	15	29	59
Initial acceptansgrad för sämre lösningar [%]	52	59	61	60	61
Slutlig acceptansgrad för sämre lösningar [%]	0	0	0	0	0

OUTER_LOOP	**10**	**50**	**100**	**200**	**400**
Total tid för samtliga uppdrag	6817	5957	5819	5729	5748
Utnyttjandegrad [%]	54	61	63	64	64
Antal försenade uppdrag	179	76	33	32	30
Genomsnittlig förseningstid	194	46	27	21	22
Beräkningstid [s]	2	9	15	28	53
Initial acceptansgrad för sämre lösningar [%]	60	60	61	60	61
Slutlig acceptansgrad för sämre lösningar [%]	46	3	0	0	0

START_TEMP	**5**	**15**	**25**	**35**	**45**
Total tid för samtliga uppdrag	5812	5830	5819	5790	5763
Utnyttjandegrad [%]	63	63	63	64	64
Antal försenade uppdrag	45	38	33	38	27
Genomsnittlig förseningstid	27	26	27	26	24
Beräkningstid [s]	12	14	15	16	16
Initial acceptansgrad för sämre lösningar [%]	14	44	61	69	74
Slutlig acceptansgrad för sämre lösningar [%]	0	0	0	0	0

ALPHA	**0.93**	**0.94**	**0.95**	**0.96**	**0.97**
Total tid för samtliga uppdrag	5779	5777	5819	5738	5862
Utnyttjandegrad [%]	63	64	63	64	63
Antal försenade uppdrag	44	31	33	32	59
Genomsnittlig förseningstid	27	21	27	25	35
Beräkningstid [s]	14	14	15	16	17
Initial acceptansgrad för sämre lösningar [%]	60	60	61	60	60
Slutlig acceptansgrad för sämre lösningar [%]	0	0	0	0	1

Appendix B – Källkod

```
/*****************************************************************************\
*****************************************************************************
* Name:
*     Optimize
*
* Purpose:
*     Minimize the time it takes for trucks perform a number of tasks in
*     a real time storage. Three techniques are used;
*     Simulated Annealing (SA, sa), "Shortest time First" (SF, sf) and
*     "Closest First with time Constraints" (CFC, cfc). SF and CFC uses
*     the same procedures
*
* Usage:
*     Set desired values for all constants.
*     Compile and run program.
*     View the result in the "result.doc" file. User must know in what
*     directory output-files are created and found.
*
* Other:
*     Procedures used when developing the program are included. (For example
*     printing different variables)
*
*****************************************************************************
\*****************************************************************************/

/*****************************************************************************\
* Section:
*     Include files
\*****************************************************************************/
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <stdio.h>

/*****************************************************************************\
* Section:
*     Constants
\*****************************************************************************/
//-Storage-USER-DEFINED-------------------------------------------------------
#define NUM_BLOCK_X         4
                                //number of storage blocks in x-direction
#define NUM_BLOCK_Y         6
                                //number of storage blocks in y-direction
#define DIST_BLOCK_X        45
                                //distance between storage blocks in x-dir. [m]
#define DIST_BLOCK_Y        20
                                //distance between storage blocks in y-dir. [m]
#define DIST_RACK_Y         5
                                //distance between the two racks in a block [m]
#define NUM_PLACE_X         10
                                //number of places in a rack
#define DIST_PLACE_X        4
                                //distance between places in a rack [m]
//-Storage--------------------------------------------------------------------
#define NUM_PLACE_TOT       NUM_BLOCK_X*NUM_BLOCK_Y*2*NUM_PLACE_X
                                //total number of places in the storage
#define NUM_BLOCKS          NUM_BLOCK_X*NUM_BLOCK_Y
                                //total number of blocks in the storage
//-Trucks-USER-DEFINED--------------------------------------------------------
#define NUM_TRUCKS          12
                                //number of trucks in the storage
#define TRUCK_SPEED         3
                                //minimum truck speed [m/s]
#define MIN_SPEED           3
                                //minimum truck speed [m/s]
#define MAX_SPEED           3
                                //maximum truck speed [m/s]
//-Tasks-USER-DEFINED---------------------------------------------------------
#define DO_NUM_TASKS        3000
                                //number of tasks to do
#define MIN_LAST_TIME       100
                                //minimum last time for the tasks
#define MAX_LAST_TIME       15000
                                //maximum last time for the tasks
#define IN_PERCENT          25
                                //chance a task begins on the first place in the
                                //storage [%]
#define OUT_PERCENT         25
                                //chance a task ends on the last place in the
                                //storage [%]
//-Tasks----------------------------------------------------------------------
#define TOT_NUM_TASKS       DO_NUM_TASKS + TASK_DEPTH + START_TASKS_IN_SA+50
                                //total number tasks in to create
//-SA-structure-USER-DEFINED--------------------------------------------------
#define NUM_TASKS_PER_TRUCK 7
                                //number of tasks/truck in the SA-structure
#define MAX_TIME_FACT       5
                                //factor for the max time in SA
#define TOTAL_TIME_FACT     200
                                //factor for the total time in SA
#define TOTAL_LATE_FACT     5
                                //factor for the total late time in SA
#define INNER_LOOP          200
                                //the inner loop in SA
#define OUTER_LOOP          200
                                //the outer loop in SA
#define START_TEMP          15
                                //start temperature in SA
#define ALPHA               0.96
                                //temperature modification factor in SA
//-SA-structure---------------------------------------------------------------
#define START_TASKS_IN_SA  NUM_TASKS_PER_TRUCK*NUM_TRUCKS+1
                                //number of tasks in the SA-structure
//-SF-CFC-structure-USER-DEFINED----------------------------------------------
#define TASK_DEPTH          85
                                //maximum task depth when selecting task to do
#define TIME_DEVIATION      100000000000000000
                                //how much the "last_time" may differ for the
                                //lowest "last_time", an absolute value
//-Other-USER-DEFINED---------------------------------------------------------
#define RUN_MODE            2
                                //RUN_MODE = 1: runs SA
                                //RUN_MODE = 2:
                                //              TIME_DEVIATION = 0 : runs SF
                                //              TIME_DEVIATION > 0 : runs CFC
//-Other----------------------------------------------------------------------
#define BIG_NUMBER          1000000000
                                //just a big number
```

```
/*****************************************************************************\
* Section:
*     Structures
\*****************************************************************************/
//-Storage-------------------------------------------------------------------
struct storage_struct
{
  int time[1+NUM_TRUCKS][NUM_PLACE_TOT][NUM_PLACE_TOT];
                              //[i=0]: time[i][j][k]=distance from place
                              //j to k [m]
                              //[i>0]: time[i][j][k]=time it takes for truck i
                              //to travel from place j to k [s]
};
//-Trucks--------------------------------------------------------------------
struct truck_struct
{
  int id;                     //unique id-number
  int speed;                  //speed [m/s]
  int location;               //location of truck (storage place)
};
//-Tasks---------------------------------------------------------------------
struct task_struct
{
  struct task_struct *next_ptr; //pointer to next task
  int    id;                  //unique id-number
  int    from;                //place in storage where the task start
  int    to;                  //place in storage where the task end
  int    from_block;          //block in storage where the task start
  int    to_block;            //block in storage where the task end
  int    last_time;           //last (=desired) ready time [s]
  int    ready_time;          //ready time [s]
  int    late_time;           //how late the task was [s]
};
//-Result--------------------------------------------------------------------
struct result_struct
{
  struct task_struct *task_ptr[NUM_TRUCKS];
                              //pointer to list with tasks performed by every
                              //truck
  int    total_travel_time;   //total work/travel-time for the trucks [h]
  int    prcnt_load;          //part of travel time with load [%]
  int    prcnt_empty;         //part of travel time without load [%]
  int    num_late;            //Number of tasks not ready in time
  int    aver_late;           //average late time for those task who were
                              //late [s]
};
//-SA-structure--------------------------------------------------------------
struct opt_truck_struct
{
  int truck;                  //truck id-number
  int time;                   //time for the truck[s]
};
struct sa_struct
{
  int    index;               //unique id-number
  int    total_time;          //total time it takes to do the tasks in
                              //SA-structure [s]
  int    total_late;          //total late time for the task in
                              //SA-structure [s]
  int    quality;             //the quality of the SA-structure
  struct opt_truck_struct max_truck;
                              //the last truck to be ready
  struct opt_truck_struct first_truck;
                              //the first truck to be ready
  int    num_tasks[NUM_TRUCKS]; //how many tasks assigned to every truck
  int    id[NUM_TRUCKS][START_TASKS_IN_SA];
                              //the task id
  int    from[NUM_TRUCKS][START_TASKS_IN_SA];
                              //place where task start
  int    to[NUM_TRUCKS][START_TASKS_IN_SA];
                              //place where task end
  int    from_block[NUM_TRUCKS][START_TASKS_IN_SA];
                              //block where task start
  int    to_block[NUM_TRUCKS][START_TASKS_IN_SA];
                              //block where task end
  int    before_time[NUM_TRUCKS][START_TASKS_IN_SA];
                              //time to travel from previous task [s]
  int    own_time[NUM_TRUCKS][START_TASKS_IN_SA];
                              //time it takes to do the task [s]
  int    after_time[NUM_TRUCKS][START_TASKS_IN_SA];
                              //time to travel to next task [s]
  int    last_time[NUM_TRUCKS][START_TASKS_IN_SA];
                              //desired last time for task [s]
  int    ready_time[NUM_TRUCKS][START_TASKS_IN_SA];
                              //when task will be ready [s]
  int    late_time[NUM_TRUCKS][START_TASKS_IN_SA];
                              //haw late the ready task will be
};
struct change_struct
{
  int truck;                  //truck id-number
  int task;                   //task id-number
  int before_time;            //time to travel from previous task [s]
  int own_time;               //time it takes to do the task [s]
  int after_time;             //time to travel to next task [s]
  int ready_time;             //when task will be ready [s]
  int diff_time;              //how much sooner/later the task will be ready
                              //when a 2-intercahnce (2i) is made
};
struct interchange_struct
{
  int    diff_max;            //effect on the max-time when a 2i is made
  int    diff_total_time;     //effect on the total time when a 2i is made
  int    diff_total_late;     //eff.  on the total late-time when a 2i is made
  int    diff_quality;        //effect on the quality when a 2i is made
  struct change_struct i1;    //the first task of the 2i
  struct change_struct i2;    //the second task of the 2i
};
//-SF-CFC-structure-----------------------------------------------------------
struct sf_cfc_struct
{
  int                   truck_location[NUM_TRUCKS];
                              //place for the last delivered task
  struct task_struct    task[NUM_TRUCKS];
                              //the task id
  struct opt_truck_struct first_truck;
                              //the first truck to be ready
};
```

```
/*****************************************************************************\
* Section:
*     Global variables
\*****************************************************************************/
struct storage_struct     storage;           //the storage
struct truck_struct       trucks[NUM_TRUCKS]; //the trucks
struct task_struct        *tasks_ptr;        //the tasks
struct result_struct      result;            //the result
struct sa_struct          sa[2];             //the variable used by SA
struct interchange_struct ic;                //the 2-interchange-variabele
struct sf_cfc_struct      sf_cfc;            //the variable used by SFCFC
int    worse_sugg_start;                 //number of worse solutions suggested
int    worse_sugg_end;                   //number of worse solutions suggested
int    worse_start;                      //worse solutions accepted at the start
int    worse_end;                        //worse solutions accepted at the end
int    simulation_time;                  //the simulation-time
int    real_world_time;                  //the "real-world"-time
char   line[15];                         //a string for inputs

/*****************************************************************************\
* Section:
*     main program
\*****************************************************************************/
int main(void)
{
    simulation_time = 0;
    tasks_ptr = NULL;
    //-------------------------------------------------------------------------
    CreateTasks(TOT_NUM_TASKS, MIN_LAST_TIME, MAX_LAST_TIME);
    CreateTrucks();
    CreateStorage();

//    PrintTasks();
//    PrintTasksToFile();
//    PrintTrucks();
    PrintTrucksToFile();
//    PrintStorage();
//    PrintStorageToFile();
    //-------------------------------------------------------------------------
    real_world_time = clock();
    if((RUN_MODE == 1) || (RUN_MODE == 2))
    {
        if(RUN_MODE == 1)
            SimulatedAnnealing();
        if(RUN_MODE == 2)
            ShortestClosest();
        real_world_time = (clock()-real_world_time)/CLOCKS_PER_SEC;
        PrintResultToFile();
        printf("\n\nOpen the file \"result.doc\"!\nPress enter to exit");
    }
    else
        printf("\n\nUnvalid run mode\nPress enter to exit");
    //-------------------------------------------------------------------------
//    fgets(line, sizeof(line), stdin);
    return 0;
}

/*****************************************************************************\
* Section:
*     procedures/functions for storage
\*****************************************************************************/
/*_____________________________________________________________________________
Name:    Find_Block
Purpose: find the block-number for a place
I/O:
_____________________________________________________________________________*/
int Find_Block(int ind)
{
    div_t result;

    result = div(ind, 2*NUM_PLACE_X);
    return(result.quot);
}
/*_____________________________________________________________________________
Name:    Find_Rack
Purpose: find the rack-number for a place
I/O:
_____________________________________________________________________________*/
int Find_Rack(int ind)
{
    div_t result;

    result = div(ind, 2*NUM_PLACE_X);
    if(result.rem <= NUM_PLACE_X-1)
        return(0);
    else
        return(1);
}
/*_____________________________________________________________________________
Name:    Find_Place
Purpose: find the place-number within a rack
I/O:
_____________________________________________________________________________*/
int Find_Place(int ind)
{
    div_t result;

    result = div(ind, 2*NUM_PLACE_X);
    if(result.rem < NUM_PLACE_X)
        return(result.rem);
    else
        return(result.rem - NUM_PLACE_X);
}
/*_____________________________________________________________________________
Name:    Dist_X
Purpose: find x-distance for a place
I/O:
_____________________________________________________________________________*/
int Dist_X(int ind)
{
    div_t result;

    result = div(Find_Block(ind), NUM_BLOCK_X);
    return(result.rem*DIST_BLOCK_X + Find_Place(ind)*DIST_PLACE_X);
}
/*_____________________________________________________________________________
Name:    Dist_Y
Purpose: find y-distance for a place
I/O:
_____________________________________________________________________________*/
int Dist_Y(int ind)
{
    div_t result;

    result = div(Find_Block(ind), NUM_BLOCK_X);
    return(result.quot*DIST_BLOCK_Y + Find_Rack(ind)*DIST_RACK_Y);
}
/*_____________________________________________________________________________
Name:    CreateStorage
Purpose: create storage(global)
I/O:
_____________________________________________________________________________*/
```

```
void CreateStorage(void)
{
    int i, j, k, x1, y1, x2, y2;

    for(j=0;j<NUM_PLACE_TOT;++j)
    {
        x1 = Dist_X(j);
        y1 = Dist_Y(j);
        for(k=0;k<NUM_PLACE_TOT;++k)
        {
            x2 = Dist_X(k);
            y2 = Dist_Y(k);
            storage.time[0][j][k] = sqrt( (x1-x2)*(x1-x2)+(y1-y2)*(y1-y2) );
        }
    }
    for(i=1;i<1+NUM_TRUCKS;++i)
        for(j=0;j<NUM_PLACE_TOT;++j)
        {
            for(k=0;k<NUM_PLACE_TOT;++k)
            {
                storage.time[i][j][k] = storage.time[0][j][k] / trucks[i-1].speed;
            }
            printf("\n%d/%d  (%d/%d)",j,i,NUM_PLACE_TOT,NUM_TRUCKS);
        }
    printf("\nStorage created");
}
/*_____________________________________________________________________________
Name:    PrintStorage
Purpose: print storage(global) to screen
I/O:
_____________________________________________________________________________*/
void PrintStorage(void)
{
    int i, j, k;

    for(i=0;i<1+NUM_TRUCKS;++i)
    {
        if(i == 0)
            printf("\nDistance[m] for Storage");
        else
            printf("\n\nTime[s] for Truck %d", i);
        fgets(line, sizeof(line), stdin);
        for(j=0;j<NUM_PLACE_TOT;++j)
            if(j == 0)
                printf("\n\t[%d]", j);
            else
                printf(" [%d]", j);
        for(j=0;j<NUM_PLACE_TOT;++j)
        {
            printf("\n");
            for(k=0;k<NUM_PLACE_TOT;++k)
            {
                if(k == 0)
                {
                    printf("[%d]\t", j);
                    printf("%d  ", storage.time[i][j][k]);
                }
                else if(k < 9)
                    printf("%d  ", storage.time[i][j][k]);
                else
                    printf("%d   ", storage.time[i][j][k]);
            }
        }
    }
}
/*_____________________________________________________________________________
Name:    PrintStorageToFile
Purpose: print storage(global) to the file "storage.doc"
I/O:
_____________________________________________________________________________*/
void PrintStorageToFile(void)
{
    const char FILE_NAME[] = "storage.doc";
    int i, j, k;
    FILE *storage_file;

    storage_file = fopen(FILE_NAME, "wb");
    if(storage_file == NULL)
    {
        printf("\nCannot open %s", FILE_NAME);
        exit(8);
    }
    for(i=0;i<1+NUM_TRUCKS;++i)
    {
        if(i == 0)
            fprintf(storage_file, "\nDistance[m] for Storage");
        else
            fprintf(storage_file, "\n\nTime[s] for Truck %d", i);
        for(j=0;j<NUM_PLACE_TOT;++j)
            if(j == 0)
                fprintf(storage_file, "\n\t[%d]", j);
            else
                fprintf(storage_file, " [%d]", j);
        for(j=0;j<NUM_PLACE_TOT;++j)
        {
            fprintf(storage_file, "\n");
            for(k=0;k<NUM_PLACE_TOT;++k)
            {
                if(k == 0)
                {
                    fprintf(storage_file, "[%d]\t", j);
                    fprintf(storage_file, "%d  ", storage.time[i][j][k]);
                }
                else if(k < 9)
                    fprintf(storage_file, "%d  ", storage.time[i][j][k]);
                else
                    fprintf(storage_file, "%d   ", storage.time[i][j][k]);
            }
        }
    }
    fclose(storage_file);
}

/*****************************************************************************\
* Section:
*     procedures/functions for trucks
\*****************************************************************************/
/*_____________________________________________________________________________
Name:    CreateTrucks
Purpose: create trucks[NUM_TRUCKS](global)
I/O:
_____________________________________________________________________________*/
void CreateTrucks(void)
{
    int i;

    for(i=0;i<NUM_TRUCKS;++i)
    {
      trucks[i].id = i;
      trucks[i].speed = TRUCK_SPEED;//Random(MIN_SPEED, MAX_SPEED);
      trucks[i].location = Random(0, NUM_PLACE_TOT-1);
    }
```

```
    printf("\nTrucks created");
}
/*_____________________________________________________________________________
Name:    PrintTrucks
Purpose: print trucks[NUM_TRUCKS](global) on screen
I/O:
_____________________________________________________________________________*/
void PrintTrucks(void)
{
    int i;

    printf("\nTrucks:");
    for(i=0;i<NUM_TRUCKS;++i)
        printf("\n\nid: %d  speed: %d  location: %d",
               trucks[i].id, trucks[i].speed,trucks[i].location);
    printf("\nPrintTrucks done, press enter");
    fgets(line, sizeof(line), stdin);
}
/*_____________________________________________________________________________
Name:    PrintTrucksToFile
Purpose: print trucks[NUM_TRUCKS](global) to the file "trucks.doc"
I/O:
_____________________________________________________________________________*/
void PrintTrucksToFile(void)
{
    int i;
    const char FILE_NAME[] = "trucks.doc";
    FILE *trucks_file;

    trucks_file = fopen(FILE_NAME, "wb");
    if(trucks_file == NULL)
    {
        printf("\nCannot open %s", FILE_NAME);
        exit(8);
    }
    fprintf(trucks_file, "\nTrucks:");
    for(i=0;i<NUM_TRUCKS;++i)
        fprintf(trucks_file, "\n\nid: %d  speed: %d  location: %d",
               trucks[i].id, trucks[i].speed, trucks[i].location);
    fclose(trucks_file);
}

/*****************************************************************************\
* Section:
*     procedures/functions for tasks
\*****************************************************************************/
/*_____________________________________________________________________________
Name:    CreateTasks
Purpose: create all tasks in *tasks_ptr(global) to be processed
I/O:
_____________________________________________________________________________*/
void CreateTasks(int nbr_tasks, int min_time, int max_time)
{
    int i;
    struct task_struct *before_ptr, *after_ptr, *new_task_ptr;

    for(i=0;i<nbr_tasks;++i)
    {
        printf("\nno. %d (%d)", i, nbr_tasks);
        new_task_ptr = malloc(sizeof(struct task_struct));
        (*new_task_ptr).next_ptr = NULL;
        (*new_task_ptr).id = i+1;
        if(Random(1, 100) <= IN_PERCENT)
            (*new_task_ptr).from = 0;
        else
            (*new_task_ptr).from = Random(1, NUM_PLACE_TOT-1);
        if(Random(1, 100) <= OUT_PERCENT)
            (*new_task_ptr).to = NUM_PLACE_TOT-1;
        else
            (*new_task_ptr).to = Random(0, NUM_PLACE_TOT-2);
        (*new_task_ptr).from_block = Find_Block((*new_task_ptr).from);
        (*new_task_ptr).to_block = Find_Block((*new_task_ptr).to);
        (*new_task_ptr).last_time = Random(min_time, max_time);
        (*new_task_ptr).ready_time = 0;
        (*new_task_ptr).late_time = 0;
        if(tasks_ptr == NULL)
            tasks_ptr = new_task_ptr;
        else
        {
            before_ptr = tasks_ptr;
            after_ptr = (*before_ptr).next_ptr;
            while(1)
            {
                if(after_ptr == NULL)
                    break;
                if((*after_ptr).last_time >= (*new_task_ptr).last_time)
                    break;
                after_ptr = (*after_ptr).next_ptr;
                before_ptr = (*before_ptr).next_ptr;
            }
            if((*tasks_ptr).last_time > (*new_task_ptr).last_time)
            {
                (*new_task_ptr).next_ptr = tasks_ptr;
                tasks_ptr = new_task_ptr;
            }
            else
            {
                (*before_ptr).next_ptr = new_task_ptr;
                (*new_task_ptr).next_ptr = after_ptr;
            }
        }
    }
    printf("\nTasks created");
}
/*_____________________________________________________________________________
Name:    DeleteTask
Purpose: delete a task from *tasks_ptr(global)
I/O:
_____________________________________________________________________________*/
void DeleteTask(int task_id)
{
    struct task_struct *before_ptr, *after_ptr;

    if((*tasks_ptr).id == task_id)
    {
        before_ptr = tasks_ptr;
        tasks_ptr = (*tasks_ptr).next_ptr;
        free(before_ptr);
        before_ptr = NULL;
    }
    else
    {
        before_ptr = tasks_ptr;
        after_ptr = (*before_ptr).next_ptr;
        while(1)
        {
            if((*after_ptr).id == task_id)
                break;
            else
            {
                before_ptr = after_ptr;
                after_ptr = (*before_ptr).next_ptr;
```

```
            }
        }
        (*before_ptr).next_ptr = (*after_ptr).next_ptr;
        free(after_ptr);
        after_ptr = NULL;
    }
}
/*_____________________________________________________________________
Name:    PrintTasks
Purpose: print *tasks_ptr(global) on the screen
I/O:
_____________________________________________________________________*/
void PrintTasks(void)
{
    struct task_struct *current_ptr;

    printf("\nTasks waiting:");
    if(tasks_ptr != NULL)
    {
        current_ptr = tasks_ptr;
        while(current_ptr != NULL)
        {
            printf("\nlast_time: %d\tid: %d\tfrom: [%d][%d]\tto: [%d][%d]",
                   (*current_ptr).last_time, (*current_ptr).id,
                   (*current_ptr).from, (*current_ptr).from_block,
                   (*current_ptr).to, (*current_ptr).to_block);
            current_ptr = (*current_ptr).next_ptr;
        }
    }
    else
        printf("\nNo Tasks exist");
    printf("\nPrintTasks done, press enter");
    fgets(line, sizeof(line), stdin);
}
/*_____________________________________________________________________
Name:    PrintTasksToFile
Purpose: print *tasks_ptr(global) to the file "tasks.doc"
I/O:
_____________________________________________________________________*/
void PrintTasksToFile(void)
{
    struct task_struct *current_ptr;
    const char FILE_NAME[] = "tasks.doc";
    FILE *tasks_file;

    tasks_file = fopen(FILE_NAME, "wb");
    if(tasks_file == NULL)
    {
        printf("\nCannot open %s", FILE_NAME);
        exit(8);
    }
    fprintf(tasks_file, "\nTasks waiting:");
    if(tasks_ptr != NULL)
    {
        current_ptr = tasks_ptr;
        while(current_ptr != NULL)
        {
            fprintf(tasks_file, "\nlast_time: %d\tid: %d\tfrom: [%d][%d]\tto: [%d][%d]",
                   (*current_ptr).last_time, (*current_ptr).id,
                   (*current_ptr).from, (*current_ptr).from_block,
                   (*current_ptr).to, (*current_ptr).to_block);
            current_ptr = (*current_ptr).next_ptr;
        }
    }
    else
        fprintf(tasks_file, "\nNo Tasks exist");
    fclose(tasks_file);
}

/*****************************************************************************\
* Section:
*     procedures/functions for result
\*****************************************************************************/
/*_____________________________________________________________________
Name:    PutTaskInResultSa
Purpose: put a task in result(global) from sa[2](global)
I/O:
_____________________________________________________________________*/
void PutTaskInResultSa(int s_1)
{
    struct task_struct *new_task_ptr, *current_ptr;

    new_task_ptr = malloc(sizeof(struct task_struct));
    (*new_task_ptr).next_ptr   = NULL;
    (*new_task_ptr).id         = sa[s_1].id[sa[s_1].first_truck.truck][0];
    (*new_task_ptr).from       = sa[s_1].from[sa[s_1].first_truck.truck][0];
    (*new_task_ptr).to         = sa[s_1].to[sa[s_1].first_truck.truck][0];
    (*new_task_ptr).from_block = sa[s_1].from_block[sa[s_1].first_truck.truck][0];
    (*new_task_ptr).to_block   = sa[s_1].to_block[sa[s_1].first_truck.truck][0];
    (*new_task_ptr).last_time  = sa[s_1].last_time[sa[s_1].first_truck.truck][0];
    (*new_task_ptr).ready_time = sa[s_1].ready_time[sa[s_1].first_truck.truck][0];
    (*new_task_ptr).late_time  = sa[s_1].late_time[sa[s_1].first_truck.truck][0];
    current_ptr = result.task_ptr[sa[s_1].first_truck.truck];
    if(current_ptr == NULL)
        result.task_ptr[sa[s_1].first_truck.truck] = new_task_ptr;
    else
    {
        while(1)
        {
            if((*current_ptr).next_ptr == NULL)
                break;
            current_ptr = (*current_ptr).next_ptr;
        }
        (*current_ptr).next_ptr = new_task_ptr;
    }
}
/*_____________________________________________________________________
Name:    PutTaskInResultSfCfc
Purpose: put a task in result(global) from sf_cfc(global)
I/O:
_____________________________________________________________________*/
void PutTaskInResultSfCfc(int truck)
{
    struct task_struct *new_task_ptr, *current_ptr;

    new_task_ptr = malloc(sizeof(struct task_struct));
    (*new_task_ptr).next_ptr   = NULL;
    (*new_task_ptr).id         = sf_cfc.task[truck].id;
    (*new_task_ptr).from       = sf_cfc.task[truck].from;
    (*new_task_ptr).to         = sf_cfc.task[truck].to;
    (*new_task_ptr).from_block = sf_cfc.task[truck].from_block;
    (*new_task_ptr).to_block   = sf_cfc.task[truck].to_block;
    (*new_task_ptr).last_time  = sf_cfc.task[truck].last_time;
    (*new_task_ptr).ready_time = sf_cfc.task[truck].ready_time;
    (*new_task_ptr).late_time  = sf_cfc.task[truck].late_time;

    current_ptr = result.task_ptr[truck];
    if(current_ptr == NULL)
        result.task_ptr[truck] = new_task_ptr;
    else
    {
        while(1)
```

```
        {
            if((*current_ptr).next_ptr == NULL)
                break;
            current_ptr = (*current_ptr).next_ptr;
        }
        (*current_ptr).next_ptr = new_task_ptr;
    }
}
/*_____________________________________________________________________________
Name:    EvaluateResult
Purpose: evaluates result(global)
I/O:
_____________________________________________________________________________*/
void EvaluateResult(void)
{
    int i, total_travel_time, total_late_time, load_time;
    struct task_struct *current_ptr, *next_ptr;

    total_travel_time = 0;
    total_late_time = 0;
    load_time = 0;
    for(i=0;i<NUM_TRUCKS;++i)
    {
        current_ptr = result.task_ptr[i];
        while(current_ptr != NULL)
        {
            load_time += storage.time[1+i][(*current_ptr).from][(*current_ptr).to];
            if((*current_ptr).late_time > 0)
            {
                total_late_time += (*current_ptr).late_time;
                ++result.num_late;
            }
            if((*current_ptr).next_ptr != NULL)
                next_ptr = (*current_ptr).next_ptr;
            else
                total_travel_time += (*current_ptr).ready_time;
            current_ptr = (*current_ptr).next_ptr;
        }
    }
    result.total_travel_time = total_travel_time;
    result.prcnt_load = (float)load_time/(float)result.total_travel_time * (float)100;
    result.prcnt_empty = 100 - result.prcnt_load;
    if(result.num_late != 0)
        result.aver_late = total_late_time/result.num_late;
    else
        result.aver_late = 0;
}
/*_____________________________________________________________________________
Name:    PrintResultToFile
Purpose: evaluates and prints result(global) to the file
         "result.doc"
I/O:
_____________________________________________________________________________*/
void PrintResultToFile(void)
{
    int i;
    struct task_struct *current_ptr;
    const char FILE_NAME[] = "result.doc";
    FILE *result_file;

    result_file = fopen(FILE_NAME, "wb");
//    fprintf(result_file, "\n\n*USER DEFINED VARIABLES***********************");
//-Results-from-optimization ---------------------------------------------------
    EvaluateResult();
//    fprintf(result_file, "\n\n\n*RESULT*************************************");
    fprintf(result_file, "\n%d",result.total_travel_time);
    fprintf(result_file, "\n%d",result.prcnt_load);
//    fprintf(result_file, "\nPart of total travel time WITHOUT load [o/o]: %d",
//                                                          result.prcnt_empty);
    fprintf(result_file, "\n%d",result.num_late);
    fprintf(result_file, "\n%d",result.aver_late);
    fprintf(result_file, "\n%d",real_world_time);
    if(RUN_MODE == 1)
    {
    fprintf(result_file,
"\n%d",(int)((float)worse_start/(float)worse_sugg_start*(float)100));
    fprintf(result_file, "\n%d",(int)((float)worse_end/(float)worse_sugg_end*(float)100));
    }
/*    EvaluateResult();
    fprintf(result_file, "\n\n\n*RESULT*************************************");
    fprintf(result_file, "\n\nTotal travel time for all trucks [h]:       \t%d",
                                                    result.total_travel_time);
    fprintf(result_file, "\nPart of total travel time WITH load [o/o]:  \t%d",
                                                    result.prcnt_load);
//    fprintf(result_file, "\nPart of total travel time WITHOUT load [o/o]: %d",
//                                                    result.prcnt_empty);
    fprintf(result_file, "\nNumber of tasks not ready in time:          \t%d",
                                                     result.num_late);
    fprintf(result_file, "\nAverage late time for late tasks:           \t%d",
                                                    result.aver_late);
    fprintf(result_file, "\nProcessing time:                            \t%d",
                                                     real_world_time);
    if(RUN_MODE == 1)
    {
    fprintf(result_file, "\nWorse accepted start [o/o]:                 \t%d",
                 (int)((float)worse_start/(float)worse_sugg_start*(float)100));
    fprintf(result_file, "\nWorse accepted end [o/o]:                   \t%d",
                     (int)((float)worse_end/(float)worse_sugg_end*(float)100));
    }*/
//-SA-structure-USER-DEFINED---------------------------------------------------
/*    if(RUN_MODE == 1)
    {
    fprintf(result_file, "\n\n-Simulated Annealing-------------------------");
    fprintf(result_file, "\nNUM_TASKS_PER_TRUCK:   %d", NUM_TASKS_PER_TRUCK);
    fprintf(result_file, "\nMAX_TIME_FACT:         %d", MAX_TIME_FACT);
    fprintf(result_file, "\nTOTAL_TIME_FACT:       %d", TOTAL_TIME_FACT);
    fprintf(result_file, "\nTOTAL_LATE_FACT:       %d", TOTAL_LATE_FACT);
    fprintf(result_file, "\nINNER_LOOP:            %d", INNER_LOOP);
    fprintf(result_file, "\nOUTER_LOOP:            %d", OUTER_LOOP);
    fprintf(result_file, "\nSTART_TEMP:            %d", START_TEMP);
    fprintf(result_file, "\nALPHA:                 %f", ALPHA);
    }
//-SF-CFC-structure-USER-DEFINED-----------------------------------------------
    if(RUN_MODE == 2)
    {
    fprintf(result_file, "\n\n-Shortest time First---------------------------");
    fprintf(result_file, "\n-Closest First with time Constraints-----------");
    fprintf(result_file, "\nTASK_DEPTH:            %d", TASK_DEPTH);
    fprintf(result_file, "\nTIME_DEVIATION [o/o]:  %d", TIME_DEVIATION);
    }
//-Storage-USER-DEFINED--------------------------------------------------------
    fprintf(result_file, "\n\n-Storage----------------------------------------");
    fprintf(result_file, "\nNUM_BLOCK_X:           %d", NUM_BLOCK_X);
    fprintf(result_file, "\nNUM_BLOCK_Y:           %d", NUM_BLOCK_Y);
    fprintf(result_file, "\nDIST_BLOCK_X [m]:      %d", DIST_BLOCK_X);
    fprintf(result_file, "\nDIST_BLOCK_Y [m]:      %d", DIST_BLOCK_Y);
    fprintf(result_file, "\nDIST_RACK_Y [m]:       %d", DIST_RACK_Y);
    fprintf(result_file, "\nNUM_PLACE_X:           %d", NUM_PLACE_X);
    fprintf(result_file, "\nDIST_PLACE_X [m]:      %d", DIST_PLACE_X);
//-Trucks-USER-DEFINED---------------------------------------------------------
    fprintf(result_file, "\n\n-Trucks------------------------------------------");
    fprintf(result_file, "\nNUM_TRUCKS:            %d", NUM_TRUCKS);
```

```
    fprintf(result_file, "\nMIN_SPEED [m/s]:       %d", MIN_SPEED);
    fprintf(result_file, "\nMAX_SPEED [m/s]:       %d", MAX_SPEED);
//-Tasks-USER-DEFINED--------------------------------------------------------
    fprintf(result_file, "\n\n-Tasks----------------------------------------");
    fprintf(result_file, "\nDO_NUM_TASKS:          %d", DO_NUM_TASKS);
    fprintf(result_file, "\nMIN_LAST_TIME [s]:     %d", MIN_LAST_TIME);
    fprintf(result_file, "\nMAX_LAST_TIME [s]:     %d", MAX_LAST_TIME);
    fprintf(result_file, "\nIN_PERCENT [s]:        %d", IN_PERCENT);
    fprintf(result_file, "\nOUT_PERCENT [s]:       %d", OUT_PERCENT);
//-Tasks-done-by-truck-------------------------------------------------------
/*    fprintf(result_file, "\n\nTasks done by trucks:");
    for(i=0;i<NUM_TRUCKS;++i)
    {
        fprintf(result_file, "\n\n-Truck number %d-------------------------------",
                                                                        (i+1));
        current_ptr = result.task_ptr[i];
        while(current_ptr != NULL)
        {
            fprintf(result_file, "\nid: %d\tfrom: [%d][%d]\tto: [%d][%d]",
                   (*current_ptr).id,
                   (*current_ptr).from, (*current_ptr).from_block,
                   (*current_ptr).to, (*current_ptr).to_block);
            fprintf(result_file, "\tlast_t: %d\tready_t: %d\tlate_t: %d",
                   (*current_ptr).last_time, (*current_ptr).ready_time,
                   (*current_ptr).late_time);
            current_ptr = (*current_ptr).next_ptr;
        }
    }
    fclose(result_file);*/
}

/*****************************************************************************\
* Section:
*     procedures/functions for SA-var
\*****************************************************************************/
/*_____________________________________________________________________________
Name:    ResetSAvar
Purpose: reset sa[2](global)
I/O:
_____________________________________________________________________________*/
void ResetSAvar(int s_1)
{
    int i, j;

    sa[s_1].index = -1;
    sa[s_1].total_late = 0;
    sa[s_1].quality = 0;
    sa[s_1].max_truck.truck = -1;
    sa[s_1].max_truck.time = 0;
    sa[s_1].first_truck.truck = -1;
    sa[s_1].first_truck.time = BIG_NUMBER;
    for(i=0;i<NUM_TRUCKS;++i)
        for(j=0;j<START_TASKS_IN_SA;++j)
        {
            sa[s_1].id[i][j] = -1;
            sa[s_1].from[i][j] = -1;
            sa[s_1].to[i][j] = -1;
            sa[s_1].from_block[i][j] = -1;
            sa[s_1].to_block[i][j] = -1;
            sa[s_1].last_time[i][j] = 0;
            sa[s_1].before_time[i][j] = 0;
            sa[s_1].own_time[i][j] = 0;
            sa[s_1].after_time[i][j] = 0;
            sa[s_1].ready_time[i][j] = 0;
            sa[s_1].late_time[i][j] = 0;
        }
}
/*_____________________________________________________________________________
Name:    PutNewTaskInSAvar
Purpose: put a new task in  sa[2](global)
I/O:
_____________________________________________________________________________*/
void PutNewTaskInSAvar(int s_1)
{
    int truck, task;

    truck = Random(0, NUM_TRUCKS-1);
    task = sa[s_1].num_tasks[truck];
    sa[s_1].id[truck][task]         = (*tasks_ptr).id;
    sa[s_1].from[truck][task]       = (*tasks_ptr).from;
    sa[s_1].to[truck][task]         = (*tasks_ptr).to;
    sa[s_1].from_block[truck][task] = (*tasks_ptr).from_block;
    sa[s_1].to_block[truck][task]   = (*tasks_ptr).to_block;
    sa[s_1].last_time[truck][task]  = (*tasks_ptr).last_time;
    //-------------------------------------------------------------------------
    if(task == 0)
    {
        sa[s_1].before_time[truck][task] =
            storage.time[1+truck][trucks[truck].location][sa[s_1].from[truck][task]];
        sa[s_1].own_time[truck][task] =
            storage.time[1+truck][sa[s_1].from[truck][task]][sa[s_1].to[truck][task]];
        sa[s_1].ready_time[truck][task] = sa[s_1].before_time[truck][task]
          + sa[s_1].own_time[truck][task]+simulation_time;
    }
    else
    {
        sa[s_1].before_time[truck][task] =
            storage.time[1+truck][sa[s_1].to[truck][task-1]][sa[s_1].from[truck][task]];
        sa[s_1].own_time[truck][task] =
            storage.time[1+truck][sa[s_1].from[truck][task]][sa[s_1].to[truck][task]];
        sa[s_1].ready_time[truck][task] = sa[s_1].ready_time[truck][task-1]
          + sa[s_1].before_time[truck][task] + sa[s_1].own_time[truck][task];
        sa[s_1].after_time[truck][task-1] = sa[s_1].before_time[truck][task];
    }
    sa[s_1].late_time[truck][task] = sa[s_1].ready_time[truck][task]
          - sa[s_1].last_time[truck][task];
    //-------------------------------------------------------------------------
    ++sa[s_1].num_tasks[truck];
    DeleteTask((*tasks_ptr).id);
}
/*_____________________________________________________________________________
Name:    DeleteTaskFromSAvar
Purpose: delete a task in sa[2](global)
I/O:
_____________________________________________________________________________*/
void DeleteTaskFromSAvar(int s_1)
{
    int i;

    trucks[sa[s_1].first_truck.truck].location = sa[s_1].to[sa[s_1].first_truck.truck][0];
    simulation_time = sa[s_1].ready_time[sa[s_1].first_truck.truck][0];
    for(i=0;i<sa[s_1].num_tasks[sa[s_1].first_truck.truck];++i)
    {
        sa[s_1].id[sa[s_1].first_truck.truck][i]                                    =
sa[s_1].id[sa[s_1].first_truck.truck][i+1];
        sa[s_1].from[sa[s_1].first_truck.truck][i]                                  =
sa[s_1].from[sa[s_1].first_truck.truck][i+1];
        sa[s_1].to[sa[s_1].first_truck.truck][i]                                    =
sa[s_1].to[sa[s_1].first_truck.truck][i+1];
        sa[s_1].from_block[sa[s_1].first_truck.truck][i]                            =
sa[s_1].from_block[sa[s_1].first_truck.truck][i+1];
```

```
        sa[s_1].to_block[sa[s_1].first_truck.truck][i]
sa[s_1].to_block[sa[s_1].first_truck.truck][i+1];
        sa[s_1].last_time[sa[s_1].first_truck.truck][i]
sa[s_1].last_time[sa[s_1].first_truck.truck][i+1];
        sa[s_1].before_time[sa[s_1].first_truck.truck][i]
sa[s_1].before_time[sa[s_1].first_truck.truck][i+1];
        sa[s_1].own_time[sa[s_1].first_truck.truck][i]
sa[s_1].last_time[sa[s_1].first_truck.truck][i+1];
        sa[s_1].after_time[sa[s_1].first_truck.truck][i]
sa[s_1].after_time[sa[s_1].first_truck.truck][i+1];
        sa[s_1].ready_time[sa[s_1].first_truck.truck][i]
sa[s_1].ready_time[sa[s_1].first_truck.truck][i+1];
        sa[s_1].late_time[sa[s_1].first_truck.truck][i]
sa[s_1].late_time[sa[s_1].first_truck.truck][i+1];
    }
    --sa[s_1].num_tasks[sa[s_1].first_truck.truck];
    CalcMaxTime(s_1);
    CalcTotalTime(s_1);
    CalcTotalLate(s_1);
    CalcQuality(s_1);
    CalcFirstReady(s_1);
}
/*_____________________________________________________________________________
Name:    CopyTaskInSAvar
Purpose: make the to tasks in sa[2](global) same
I/O:
_____________________________________________________________________________*/
void CopyTaskInSAvar(int s_1, int truck_1, int task_1, int truck_2, int task_2)
{
    sa[s_1].id[truck_1][task_1]         = sa[s_1].id[truck_2][task_2];
    sa[s_1].from[truck_1][task_1]       = sa[s_1].from[truck_2][task_2];
    sa[s_1].to[truck_1][task_1]         = sa[s_1].to[truck_2][task_2];
    sa[s_1].from_block[truck_1][task_1] = sa[s_1].from_block[truck_2][task_2];
    sa[s_1].to_block[truck_1][task_1]   = sa[s_1].to_block[truck_2][task_2];
    sa[s_1].last_time[truck_1][task_1]  = sa[s_1].last_time[truck_2][task_2];
}
/*_____________________________________________________________________________
Name:    UpdateNumTasksPerTruck
Purpose: updates number of tasks/truck in  sa[2](global)
I/O:
_____________________________________________________________________________*/
void UpdateNumTasksPerTruck(int s_1)
{
    int i, j;

    for(i=0;i<NUM_TRUCKS;++i)
        sa[s_1].num_tasks[i] = 0;
    for(i=0;i<NUM_TRUCKS;++i)
    {
        j = 0;
        while(sa[s_1].id[i][j] != -1)
        {
            ++sa[s_1].num_tasks[i];
            ++j;
        }
    }
}
/*_____________________________________________________________________________
Name:    CalcMaxTime
Purpose: calculate maxtime in sa[2](global)
I/O:
_____________________________________________________________________________*/
void CalcMaxTime(int s_1)
{
    int i;

    sa[s_1].max_truck.time = -1;
    for(i=0;i<NUM_TRUCKS;++i)
        if(sa[s_1].max_truck.time < sa[s_1].ready_time[i][sa[s_1].num_tasks[i]-1])
        {
            sa[s_1].max_truck.truck = i;
            sa[s_1].max_truck.time = sa[s_1].ready_time[i][sa[s_1].num_tasks[i]-1];
        }
}
/*_____________________________________________________________________________
Name:    CalcFirstReady
Purpose: calculate first truck ready in sa[2](global)
I/O:
_____________________________________________________________________________*/
void CalcFirstReady(int s_1)
{
        int i;

        sa[s_1].first_truck.time = BIG_NUMBER;
        for(i=0;i<NUM_TRUCKS;++i)
            if( (sa[s_1].ready_time[i][0] != 0) &&
                (sa[s_1].ready_time[i][0] < sa[s_1].first_truck.time) )
            {
                sa[s_1].first_truck.truck = i;
                sa[s_1].first_truck.time = sa[s_1].ready_time[i][0];
            }
        if(sa[s_1].first_truck.time == BIG_NUMBER)
        {
            sa[s_1].first_truck.time = 0;
            sa[s_1].first_truck.truck = -1;
        }
}
/*_____________________________________________________________________________
Name:    CalcTotalTime
Purpose: calculate total time in sa[2](global)
I/O:
_____________________________________________________________________________*/
void CalcTotalTime(int s_1)
{
    int i;

    sa[s_1].total_time = 0;
    for(i=0;i<NUM_TRUCKS;++i)
            sa[s_1].total_time += sa[s_1].ready_time[i][sa[s_1].num_tasks[i]-1];
}
/*_____________________________________________________________________________
Name:    CalcTotalLate
Purpose: calculate total latetime in sa[2](global)
I/O:
_____________________________________________________________________________*/
void CalcTotalLate(int s_1)
{
    int i, j;

    sa[s_1].total_late = 0;
    for(i=0;i<NUM_TRUCKS;++i)
        for(j=0;j<sa[s_1].num_tasks[i];++j)
            sa[s_1].total_late += sa[s_1].late_time[i][j];
}
/*_____________________________________________________________________________
Name:    CalcQuality
Purpose: calculate quality in sa[2](global)
I/O:
_____________________________________________________________________________*/
void CalcQuality(int s_1)
{
    sa[s_1].quality = sa[s_1].max_truck.time*MAX_TIME_FACT
```

```
                + sa[s_1].total_time*TOTAL_TIME_FACT
                + sa[s_1].total_late*TOTAL_LATE_FACT;
}
/*____________________________________________________________________________
Name:    InitSAvar
Purpose: initialize sa[2](global)
I/O:
____________________________________________________________________________*/
void InitSAvar(int s_1)
{
    int i;

    ResetSAvar(s_1);
    sa[s_1].index = s_1;
    for(i=0;i<START_TASKS_IN_SA;++i)
        PutNewTaskInSAvar(s_1);
    UpdateNumTasksPerTruck(s_1);
    CalcMaxTime(s_1);
    CalcFirstReady(s_1);
    CalcTotalTime(s_1);
    CalcTotalLate(s_1);
    CalcQuality(s_1);
}
/*____________________________________________________________________________
Name:    CopySAvar
Purpose: make the to elements in sa[2](global) same
I/O:
____________________________________________________________________________*/
void CopySAvar(int s_1, int s_2)
{
    int i, j;

    sa[s_2].index             = s_2;
    sa[s_2].total_time        = sa[s_1].total_time;
    sa[s_2].total_late        = sa[s_1].total_late;
    sa[s_2].quality           = sa[s_1].quality;
    sa[s_2].max_truck.truck   = sa[s_1].max_truck.truck;
    sa[s_2].max_truck.time    = sa[s_1].max_truck.time;
    sa[s_2].first_truck.truck = sa[s_1].first_truck.truck;
    sa[s_2].first_truck.time  = sa[s_1].first_truck.time;
    for(i=0;i<NUM_TRUCKS;++i)
        for(j=0;j<START_TASKS_IN_SA;++j)
        {
                sa[s_2].id[i][j]          = sa[s_1].id[i][j];
                sa[s_2].from[i][j]        = sa[s_1].from[i][j];
                sa[s_2].to[i][j]          = sa[s_1].to[i][j];
                sa[s_2].from_block[i][j]  = sa[s_1].from_block[i][j];
                sa[s_2].to_block[i][j]    = sa[s_1].to_block[i][j];
                sa[s_2].last_time[i][j]   = sa[s_1].last_time[i][j];
                sa[s_2].before_time[i][j] = sa[s_1].before_time[i][j];
                sa[s_2].own_time[i][j]    = sa[s_1].own_time[i][j];
                sa[s_2].after_time[i][j]  = sa[s_1].after_time[i][j];
                sa[s_2].ready_time[i][j]  = sa[s_1].ready_time[i][j];
                sa[s_2].late_time[i][j]   = sa[s_1].late_time[i][j];
        }
}
/*____________________________________________________________________________
Name:    PrintSAvar
Purpose: print sa[2](global) on screen
I/O:
____________________________________________________________________________*/
void PrintSAvar(int s_1, int print_ver)
{
    int i, j;

    printf("\nSol. no. %d\tmax: %d (truck %d)\tfirst: %d (truck %d)",
           sa[s_1].index, sa[s_1].max_truck.time, sa[s_1].max_truck.truck,
           sa[s_1].first_truck.time, sa[s_1].first_truck.truck);
    printf("\ntotal_time: %d\ttotal_late: %d\t\tquality: %d",
           sa[s_1].total_time, sa[s_1].total_late, sa[s_1].quality);
    for(i=0;i<NUM_TRUCKS;++i)
    {
        printf("\nTruck no: %d (from place: %d)", i, trucks[i].location);
        for(j=0;j<START_TASKS_IN_SA;++j)
            if(sa[s_1].id[i][j] != -1)
                if(print_ver == 1)
                    printf("\nid: %d\t%d[%d]->%d[%d]\tlast_t: %d\tready_t: %d\tlate_t: %d",
                           sa[s_1].id[i][j], sa[s_1].from[i][j],
                           sa[s_1].from_block[i][j], sa[s_1].to[i][j],
                           sa[s_1].to_block[i][j], sa[s_1].last_time[i][j],
                           sa[s_1].ready_time[i][j], sa[s_1].late_time[i][j]);
                else if(print_ver == 2)
                    printf("\nid:          %d\t%d[%d]->%d[%d]\tlast_t:          %d\tready_t:
%d\t[%d][%d][%d]",
                           sa[s_1].id[i][j], sa[s_1].from[i][j],
                           sa[s_1].from_block[i][j], sa[s_1].to[i][j],
                           sa[s_1].to_block[i][j], sa[s_1].last_time[i][j],
                           sa[s_1].ready_time[i][j],
                           sa[s_1].before_time[i][j],                sa[s_1].own_time[i][j],
sa[s_1].after_time[i][j]);
    }
    printf("\nPrintSAvar done, press enter");
    fgets(line, sizeof(line), stdin);
}
/*____________________________________________________________________________
Name:    PrintSAvarToFile
Purpose: print sa[2](global) to the file "SAvar.doc"
I/O:
____________________________________________________________________________*/
void PrintSAvarToFile(int s_1, int print_ver)
{
    int i, j;
    struct task_struct *current_ptr;
    const char FILE_NAME[] = "SAvar.doc";
    FILE *solution_file;

    solution_file = fopen(FILE_NAME, "wb");
    if(solution_file == NULL)
    {
        printf("\nCannot open %s", FILE_NAME);
        exit(8);
    }
    fprintf(solution_file, "\nSol. no. %d\nmax: %d(truck %d)  total_late: %d  quality: %d",
           sa[s_1].index, sa[s_1].max_truck.time, sa[s_1].max_truck.truck,
           sa[s_1].total_late, sa[s_1].quality);
    fprintf(solution_file, "\nfirst_ready\ttime: %d  truck: %d",
           sa[s_1].first_truck.time, sa[s_1].first_truck.truck);
    for(i=0;i<NUM_TRUCKS;++i)
    {
        fprintf(solution_file, "\n\nTruck no: %d", j);
        for(j=0;j<START_TASKS_IN_SA;++j)
            if(sa[s_1].id[i][j] != -1)
                if(print_ver == 1)
                    fprintf(solution_file,  "\nid:  %d\t%d[%d]->%d[%d]\tlast_t:  %d\tready_t:
%d\tlate_t: %d",
                           sa[s_1].id[i][j], sa[s_1].from[i][j],
                           sa[s_1].from_block[i][j], sa[s_1].to[i][j],
                           sa[s_1].to_block[i][j], sa[s_1].last_time[i][j],
                           sa[s_1].ready_time[i][j], sa[s_1].late_time[i][j]);
                else if(print_ver == 2)
                    fprintf(solution_file,  "\nid:  %d\t%d[%d]->%d[%d]\tlast_t:  %d\tready_t:
%d\t[%d][%d][%d]",
```

```
                            sa[s_1].id[i][j], sa[s_1].from[i][j],
                            sa[s_1].from_block[i][j], sa[s_1].to[i][j],
                            sa[s_1].to_block[i][j], sa[s_1].last_time[i][j],
                            sa[s_1].ready_time[i][j],
                            sa[s_1].before_time[i][j],                sa[s_1].own_time[i][j],
sa[s_1].after_time[i][j]);
    }
    fclose(solution_file);
}

/*****************************************************************************\
* Section:
*     procedures/functions for Simulated Annealing
\*****************************************************************************/
/*___________________________________________________________________________
Name:    Suggested2iChange
Purpose: the suggested 2-interchange ic(global)
I/O:
___________________________________________________________________________*/
void Suggested2iChange(int s_1)
{
    ic.i1.truck = Random(0, NUM_TRUCKS-1);
    ic.i1.task =  Random(1, sa[s_1].num_tasks[ic.i1.truck]);
    ic.i2.truck = Random(0, NUM_TRUCKS-1);
    ic.i2.task =  Random(1, sa[s_1].num_tasks[ic.i2.truck]);
    Calc2iTimes(s_1);
    Calc2iDifferences(s_1);
}
/*___________________________________________________________________________
Name:    Calc2iTimes
Purpose: calculate 2-interchange times ic(global)
I/O:
___________________________________________________________________________*/
void Calc2iTimes(int s_1)
{
    if(sa[s_1].id[ic.i1.truck][ic.i1.task] != -1)
    {
        ic.i1.before_time =
            storage.time[1+ic.i2.truck][sa[s_1].to[ic.i2.truck][ic.i2.task-
1]][sa[s_1].from[ic.i1.truck][ic.i1.task]];
        ic.i1.own_time =

storage.time[1+ic.i2.truck][sa[s_1].from[ic.i1.truck][ic.i1.task]][sa[s_1].to[ic.i1.truck][i
c.i1.task]];
        ic.i1.ready_time = sa[s_1].ready_time[ic.i2.truck][ic.i2.task-1]
                         + ic.i1.before_time
                         + ic.i1.own_time;
        if(sa[s_1].id[ic.i2.truck][ic.i2.task+1] != -1)
            ic.i1.after_time =

storage.time[1+ic.i2.truck][sa[s_1].to[ic.i1.truck][ic.i1.task]][sa[s_1].from[ic.i2.truck][i
c.i2.task+1]];
        else
            ic.i1.after_time = 0;
    }
    else
    {
        if(sa[s_1].id[ic.i2.truck][ic.i2.task+1] != -1)
        {
            ic.i1.before_time =
                storage.time[1+ic.i2.truck][sa[s_1].to[ic.i2.truck][ic.i2.task-
1]][sa[s_1].from[ic.i2.truck][ic.i2.task+1]];
            ic.i1.ready_time = sa[s_1].ready_time[ic.i2.truck][ic.i2.task-1]
                             + ic.i1.before_time
                             + sa[s_1].own_time[ic.i2.truck][ic.i2.task+1];
        }
        else
            ic.i1.before_time = ic.i1.ready_time = 0;
        ic.i1.own_time = ic.i1.after_time = 0;
    }
    //---------------------------------------------------------------------------
    if(sa[s_1].id[ic.i2.truck][ic.i2.task] != -1)
    {
        ic.i2.before_time =
            storage.time[1+ic.i1.truck][sa[s_1].to[ic.i1.truck][ic.i1.task-
1]][sa[s_1].from[ic.i2.truck][ic.i2.task]];
        ic.i2.own_time =

storage.time[1+ic.i1.truck][sa[s_1].from[ic.i2.truck][ic.i2.task]][sa[s_1].to[ic.i2.truck][i
c.i2.task]];
        ic.i2.ready_time = sa[s_1].ready_time[ic.i1.truck][ic.i1.task-1]
                         + ic.i2.before_time
                         + ic.i2.own_time;
        if(sa[s_1].id[ic.i1.truck][ic.i1.task+1] != -1)
            ic.i2.after_time =

storage.time[1+ic.i1.truck][sa[s_1].to[ic.i2.truck][ic.i2.task]][sa[s_1].from[ic.i1.truck][i
c.i1.task+1]];
        else
            ic.i2.after_time = 0;
    }
    else
    {
        if(sa[s_1].id[ic.i1.truck][ic.i1.task+1] != -1)
        {
            ic.i2.before_time =
                storage.time[1+ic.i1.truck][sa[s_1].to[ic.i1.truck][ic.i1.task-
1]][sa[s_1].from[ic.i1.truck][ic.i1.task+1]];
            ic.i2.ready_time = sa[s_1].ready_time[ic.i1.truck][ic.i1.task-1]
                             + ic.i2.before_time
                             + sa[s_1].own_time[ic.i1.truck][ic.i1.task+1];
        }
        else
            ic.i2.before_time = ic.i2.ready_time = 0;
        ic.i2.own_time = ic.i2.after_time = 0;
    }
    if(ic.i1.truck == ic.i2.truck)
        ModifyTaskTimes(s_1);
    else
    {
        ic.i1.diff_time = ic.i1.before_time
                        + ic.i1.own_time
                        + ic.i1.after_time
                        - sa[s_1].before_time[ic.i2.truck][ic.i2.task]
                        - sa[s_1].own_time[ic.i2.truck][ic.i2.task]
                        - sa[s_1].after_time[ic.i2.truck][ic.i2.task];
        ic.i2.diff_time = ic.i2.before_time
                        + ic.i2.own_time
                        + ic.i2.after_time
                        - sa[s_1].before_time[ic.i1.truck][ic.i1.task]
                        - sa[s_1].own_time[ic.i1.truck][ic.i1.task]
                        - sa[s_1].after_time[ic.i1.truck][ic.i1.task];
    }
}
/*___________________________________________________________________________
Name:    Calc2iDifferences
Purpose: calculate the effect of the 2-interchange ic(global)
I/O:
___________________________________________________________________________*/
void Calc2iDifferences(int s_1)
{
    ic.diff_total_time = ic.i1.diff_time + ic.i2.diff_time;
```

```
    //------------------------------------------------------------------------
    ic.diff_max = 0;
    if(ic.i1.truck == sa[s_1].max_truck.truck)
        ic.diff_max += ic.i2.diff_time;
    if(ic.i2.truck == sa[s_1].max_truck.truck)
        ic.diff_max += ic.i1.diff_time;
    //------------------------------------------------------------------------
    ic.diff_total_late = Calc2iDiffTotalLate(s_1);
    ic.diff_quality = (ic.diff_total_time * TOTAL_TIME_FACT
                    + ic.diff_max * MAX_TIME_FACT
                    + ic.diff_total_late * TOTAL_LATE_FACT)
                    / (MAX_TIME_FACT + TOTAL_TIME_FACT + TOTAL_LATE_FACT);
}
/*____________________________________________________________________________
Name:    Calc2iDiffTotalLate
Purpose: calculate the late-effect of the 2-interchange ic(global)
I/O:
____________________________________________________________________________*/
int Calc2iDiffTotalLate(int s_1)
{
    int result, pos_res;

    if((sa[s_1].id[ic.i1.truck][ic.i1.task] != -1) &&
       (sa[s_1].id[ic.i2.truck][ic.i2.task] != -1))
        result         =         (          (max(sa[s_1].last_time[ic.i1.truck][ic.i1.task],
sa[s_1].last_time[ic.i2.truck][ic.i2.task])
                  - sa[s_1].last_time[ic.i1.truck][ic.i1.task])
                * (ic.i1.ready_time - sa[s_1].ready_time[ic.i1.truck][ic.i1.task])
                +                           (max(sa[s_1].last_time[ic.i1.truck][ic.i1.task],
sa[s_1].last_time[ic.i2.truck][ic.i2.task])
                  - sa[s_1].last_time[ic.i2.truck][ic.i2.task])
                * (ic.i2.ready_time - sa[s_1].ready_time[ic.i2.truck][ic.i2.task]) );
    else if((sa[s_1].id[ic.i1.truck][ic.i1.task] != -1) &&
            (sa[s_1].id[ic.i1.truck][ic.i1.task+1] != -1) &&
            (sa[s_1].id[ic.i2.truck][ic.i2.task] == -1))
        result         =         (          (max(sa[s_1].last_time[ic.i1.truck][ic.i1.task],
sa[s_1].last_time[ic.i1.truck][ic.i1.task+1])
                  - sa[s_1].last_time[ic.i1.truck][ic.i1.task])
                * (ic.i1.ready_time - sa[s_1].ready_time[ic.i1.truck][ic.i1.task])
                +                           (max(sa[s_1].last_time[ic.i1.truck][ic.i1.task],
sa[s_1].last_time[ic.i1.truck][ic.i1.task+1])
                  - sa[s_1].last_time[ic.i1.truck][ic.i1.task+1])
                * (ic.i2.ready_time - sa[s_1].ready_time[ic.i1.truck][ic.i1.task+1]) );
    else if((sa[s_1].id[ic.i1.truck][ic.i1.task] == -1) &&
            (sa[s_1].id[ic.i2.truck][ic.i2.task] != -1) &&
            (sa[s_1].id[ic.i2.truck][ic.i2.task+1] != -1))
        result         =         (          (max(sa[s_1].last_time[ic.i2.truck][ic.i2.task],
sa[s_1].last_time[ic.i2.truck][ic.i2.task+1])
                  - sa[s_1].last_time[ic.i2.truck][ic.i2.task])
                * (ic.i2.ready_time - sa[s_1].ready_time[ic.i2.truck][ic.i2.task])
                +                           (max(sa[s_1].last_time[ic.i2.truck][ic.i2.task],
sa[s_1].last_time[ic.i2.truck][ic.i2.task+1])
                  - sa[s_1].last_time[ic.i2.truck][ic.i2.task+1])
                * (ic.i1.ready_time - sa[s_1].ready_time[ic.i2.truck][ic.i2.task+1]) );
    else if((sa[s_1].id[ic.i1.truck][ic.i1.task] != -1) &&
            (sa[s_1].id[ic.i2.truck][ic.i2.task]                ==                -1))//&&
(sa[s_1].id[ic.i1.truck][ic.i1.task+1] == -1)
        result = ( (sa[s_1].last_time[ic.i1.truck][ic.i1.task])
                * (ic.i1.ready_time - sa[s_1].ready_time[ic.i1.truck][ic.i1.task]) );
    else if((sa[s_1].id[ic.i1.truck][ic.i1.task] == -1) &&
            (sa[s_1].id[ic.i2.truck][ic.i2.task]                !=                -1))//&&
(sa[s_1].id[ic.i2.truck][ic.i2.task+1] == -1)
        result = ( (sa[s_1].last_time[ic.i2.truck][ic.i2.task])
                * (ic.i2.ready_time - sa[s_1].ready_time[ic.i2.truck][ic.i2.task]) );
    else
        result = 0;

    if(result >= 0)
        pos_res = 1;
    else
        pos_res = 0;
    if( ((sa[s_1].id[ic.i1.truck][ic.i1.task] != -1) &&
         (sa[s_1].id[ic.i1.truck][ic.i1.task+1] == -1) &&
         (sa[s_1].id[ic.i2.truck][ic.i2.task] == -1)) ||
        ((sa[s_1].id[ic.i2.truck][ic.i2.task] != -1) &&
         (sa[s_1].id[ic.i2.truck][ic.i2.task+1] == -1) &&
         (sa[s_1].id[ic.i1.truck][ic.i1.task] == -1)) )
        result = sqrt(abs(result));
    result = sqrt(abs(result));
    if(pos_res == 1)
        return(result);
    else
        return(-result);
}
/*____________________________________________________________________________
Name:    ModifyTaskTimes
Purpose: calculate basic late-effects of the 2-interchange ic(global)
I/O:
____________________________________________________________________________*/
void ModifyTaskTimes(int s_1)
{
    if(ic.i1.task - ic.i2.task == 1)
    {
        if(sa[s_1].id[ic.i1.truck][ic.i1.task] != -1)
        {
            ic.i1.before_time =
                storage.time[1+ic.i2.truck][sa[s_1].to[ic.i2.truck][ic.i2.task-
1]][sa[s_1].from[ic.i1.truck][ic.i1.task]];
            ic.i1.after_time = ic.i2.before_time =

storage.time[1+ic.i2.truck][sa[s_1].to[ic.i1.truck][ic.i1.task]][sa[s_1].from[ic.i2.truck][i
c.i2.task]];
            ic.i1.ready_time = sa[s_1].ready_time[ic.i2.truck][ic.i2.task-1]
                             + ic.i1.before_time
                             + ic.i1.own_time;
            ic.i2.ready_time = ic.i1.ready_time
                             + ic.i2.before_time
                             + ic.i2.own_time;
        }
        else
        {
            ic.i2.before_time = sa[s_1].before_time[ic.i2.truck][ic.i2.task];
            ic.i2.ready_time = sa[s_1].ready_time[ic.i2.truck][ic.i2.task];
        }
    }
    //------------------------------------------------------------------------
    else if(ic.i1.task - ic.i2.task == -1)
    {
        if(sa[s_1].id[ic.i2.truck][ic.i2.task] != -1)
        {
            ic.i2.before_time =
                storage.time[1+ic.i1.truck][sa[s_1].to[ic.i1.truck][ic.i1.task-
1]][sa[s_1].from[ic.i2.truck][ic.i2.task]];
            ic.i2.after_time = ic.i1.before_time =

storage.time[1+ic.i1.truck][sa[s_1].to[ic.i2.truck][ic.i2.task]][sa[s_1].from[ic.i1.truck][i
c.i1.task]];
            ic.i2.ready_time = sa[s_1].ready_time[ic.i1.truck][ic.i1.task-1]
                             + ic.i2.before_time
                             + ic.i2.own_time;
            ic.i1.ready_time = ic.i2.ready_time
                             + ic.i1.before_time
```

```
                            + ic.i1.own_time;
        }
        else
        {
            ic.i1.before_time = sa[s_1].before_time[ic.i1.truck][ic.i1.task];
            ic.i1.ready_time = sa[s_1].ready_time[ic.i1.truck][ic.i1.task];
        }
    }
    //-------------------------------------------------------------------------
    ic.i1.diff_time = ic.i1.before_time
                    + ic.i1.own_time
                    + ic.i1.after_time
                    - sa[s_1].before_time[ic.i2.truck][ic.i2.task]
                    - sa[s_1].own_time[ic.i2.truck][ic.i2.task]
                    - sa[s_1].after_time[ic.i2.truck][ic.i2.task];
    ic.i2.diff_time = ic.i2.before_time
                    + ic.i2.own_time
                    + ic.i2.after_time
                    - sa[s_1].before_time[ic.i1.truck][ic.i1.task]
                    - sa[s_1].own_time[ic.i1.truck][ic.i1.task]
                    - sa[s_1].after_time[ic.i1.truck][ic.i1.task];
    //-------------------------------------------------------------------------
    if(ic.i1.task - ic.i2.task > 1)
        ic.i2.ready_time += ic.i1.diff_time;
    else if(ic.i1.task - ic.i2.task < -1)
        ic.i1.ready_time += ic.i2.diff_time;
}
/*_____________________________________________________________________________
Name:    CalcOneReadyLateInSAvar
Purpose: calculate ready and late times for one task in sa[2](global)
I/O:
_____________________________________________________________________________*/
void CalcOneReadyLateInSAvar(int s_1, int truck, int task)
{
    int i;

    sa[s_1].late_time[truck][task] = sa[s_1].ready_time[truck][task]
                                   - sa[s_1].last_time[truck][task];
    for(i=task;i<sa[s_1].num_tasks[truck];++i)
    {
        sa[s_1].ready_time[truck][i] = sa[s_1].ready_time[truck][i-1]
                                     + sa[s_1].before_time[truck][i]
                                     + sa[s_1].own_time[truck][i];
        sa[s_1].late_time[truck][i]  = sa[s_1].ready_time[truck][i]
                                     - sa[s_1].last_time[truck][i];
    }
}
/*_____________________________________________________________________________
Name:    ChangeSAvar
Purpose: do the 2-interchange ic(global) on sa[2](global)
I/O:
_____________________________________________________________________________*/
void ChangeSAvar(int s_1)
{
    int i;
    int temp1, temp2, temp3, temp4, temp5, temp6;

     if((sa[s_1].id[ic.i1.truck][ic.i1.task] != -1) && (sa[s_1].id[ic.i2.truck][ic.i2.task]
!= -1))
    {
        temp1  = sa[s_1].id[ic.i1.truck][ic.i1.task];
        temp2  = sa[s_1].from[ic.i1.truck][ic.i1.task];
        temp3  = sa[s_1].to[ic.i1.truck][ic.i1.task];
        temp4  = sa[s_1].from_block[ic.i1.truck][ic.i1.task];
        temp5  = sa[s_1].to_block[ic.i1.truck][ic.i1.task];
        temp6  = sa[s_1].last_time[ic.i1.truck][ic.i1.task];

        CopyTaskInSAvar(s_1, ic.i1.truck, ic.i1.task, ic.i2.truck, ic.i2.task);
        sa[s_1].before_time[ic.i1.truck][ic.i1.task]   = ic.i2.before_time;
        sa[s_1].own_time[ic.i1.truck][ic.i1.task]      = ic.i2.own_time;
        sa[s_1].after_time[ic.i1.truck][ic.i1.task]    = ic.i2.after_time;
         sa[s_1].ready_time[ic.i1.truck][ic.i1.task]    = ic.i2.ready_time; //not necessary
(all ready_time's)
        //
        sa[s_1].after_time[ic.i1.truck][ic.i1.task-1]                                  =
sa[s_1].before_time[ic.i1.truck][ic.i1.task];
        sa[s_1].before_time[ic.i1.truck][ic.i1.task+1]                                 =
sa[s_1].after_time[ic.i1.truck][ic.i1.task];
        //-----------------------------------------------------------------------
        sa[s_1].id[ic.i2.truck][ic.i2.task]            = temp1;
        sa[s_1].from[ic.i2.truck][ic.i2.task]          = temp2;
        sa[s_1].to[ic.i2.truck][ic.i2.task]            = temp3;
        sa[s_1].from_block[ic.i2.truck][ic.i2.task]    = temp4;
        sa[s_1].to_block[ic.i2.truck][ic.i2.task]      = temp5;
        sa[s_1].last_time[ic.i2.truck][ic.i2.task]     = temp6;
        sa[s_1].before_time[ic.i2.truck][ic.i2.task]   = ic.i1.before_time;
        sa[s_1].own_time[ic.i2.truck][ic.i2.task]      = ic.i1.own_time;
        sa[s_1].after_time[ic.i2.truck][ic.i2.task]    = ic.i1.after_time;
        sa[s_1].ready_time[ic.i2.truck][ic.i2.task]    = ic.i1.ready_time;
        //
        sa[s_1].after_time[ic.i2.truck][ic.i2.task-1]                                  =
sa[s_1].before_time[ic.i2.truck][ic.i2.task];
        sa[s_1].before_time[ic.i2.truck][ic.i2.task+1]                                 =
sa[s_1].after_time[ic.i2.truck][ic.i2.task];
    }
    //-------------------------------------------------------------------------
    if( (sa[s_1].id[ic.i1.truck][ic.i1.task] != -1) &&
        (sa[s_1].id[ic.i2.truck][ic.i2.task] == -1) &&
        (ic.i1.task != sa[s_1].num_tasks[ic.i1.truck]-1) )
    {
        CopyTaskInSAvar(s_1, ic.i2.truck, ic.i2.task, ic.i1.truck, ic.i1.task);
        sa[s_1].before_time[ic.i2.truck][ic.i2.task]   = ic.i1.before_time;
        sa[s_1].own_time[ic.i2.truck][ic.i2.task]      = ic.i1.own_time;
        sa[s_1].after_time[ic.i2.truck][ic.i2.task]    = ic.i1.after_time;
        sa[s_1].ready_time[ic.i2.truck][ic.i2.task]    = ic.i1.ready_time;
        //
        sa[s_1].after_time[ic.i2.truck][ic.i2.task-1]                                  =
sa[s_1].before_time[ic.i2.truck][ic.i2.task];
        //-----------------------------------------------------------------------
        for(i=ic.i1.task;i<sa[s_1].num_tasks[ic.i1.truck]+1;++i)
        {
            CopyTaskInSAvar(s_1, ic.i1.truck, i, ic.i1.truck, i+1);
            if(i == ic.i1.task)
            {
                sa[s_1].before_time[ic.i1.truck][i]  = ic.i2.before_time;
                sa[s_1].ready_time[ic.i1.truck][i]  = ic.i2.ready_time;
                //
                sa[s_1].after_time[ic.i1.truck][i-1] = ic.i2.before_time;
            }
            else
                sa[s_1].before_time[ic.i1.truck][i] = sa[s_1].before_time[ic.i1.truck][i+1];
            sa[s_1].own_time[ic.i1.truck][i]   = sa[s_1].own_time[ic.i1.truck][i+1];
            sa[s_1].after_time[ic.i1.truck][i] = sa[s_1].after_time[ic.i1.truck][i+1];
        }
        ++sa[s_1].num_tasks[ic.i2.truck];
        --sa[s_1].num_tasks[ic.i1.truck];
    }
    if( (sa[s_1].id[ic.i1.truck][ic.i1.task] == -1) &&
        (sa[s_1].id[ic.i2.truck][ic.i2.task] != -1) &&
        (ic.i2.task != sa[s_1].num_tasks[ic.i2.truck]-1) )
    {
        CopyTaskInSAvar(s_1, ic.i1.truck, ic.i1.task, ic.i2.truck, ic.i2.task);
```

```
        sa[s_1].before_time[ic.i1.truck][ic.i1.task]   = ic.i2.before_time;
        sa[s_1].own_time[ic.i1.truck][ic.i1.task]      = ic.i2.own_time;
        sa[s_1].after_time[ic.i1.truck][ic.i1.task]    = ic.i2.after_time;
        sa[s_1].ready_time[ic.i1.truck][ic.i1.task]    = ic.i2.ready_time;
        //
        sa[s_1].after_time[ic.i1.truck][ic.i1.task-1]                                    =
sa[s_1].before_time[ic.i1.truck][ic.i1.task];
        //---------------------------------------------------------------------
        for(i=ic.i2.task;i<sa[s_1].num_tasks[ic.i2.truck]+1;++i)
        {
            CopyTaskInSAvar(s_1, ic.i2.truck, i, ic.i2.truck, i+1);
            if(i == ic.i2.task)
            {
                sa[s_1].before_time[ic.i2.truck][i]  = ic.i1.before_time;
                sa[s_1].ready_time[ic.i2.truck][i]  = ic.i1.ready_time;
                //
                sa[s_1].after_time[ic.i2.truck][i-1] = ic.i1.before_time;
            }
            else
                sa[s_1].before_time[ic.i2.truck][i] = sa[s_1].before_time[ic.i2.truck][i+1];
            sa[s_1].own_time[ic.i2.truck][i]   = sa[s_1].own_time[ic.i2.truck][i+1];
            sa[s_1].after_time[ic.i2.truck][i] = sa[s_1].after_time[ic.i2.truck][i+1];
        }
        ++sa[s_1].num_tasks[ic.i1.truck];
        --sa[s_1].num_tasks[ic.i2.truck];
    }
    CalcOneReadyLateInSAvar(s_1, ic.i1.truck, ic.i1.task);
    CalcOneReadyLateInSAvar(s_1, ic.i2.truck, ic.i2.task);
    CalcMaxTime(s_1);
    CalcTotalTime(s_1);
    CalcTotalLate(s_1);
    CalcQuality(s_1);
}
/*____________________________________________________________________________
Name:    WorseAccepted
Purpose: return if a worse SAvar is accepted
I/O:
____________________________________________________________________________*/
int WorseAccepted(int diff_quality, float T)
{
    if( ((float)Random(0,1000000) < exp(-(float)diff_quality/(float)T)*(float)1000000) &&
        (T > 0) )
        return(1);
    else
        return(0);
}
/*____________________________________________________________________________
Name:    FunctionT
Purpose: return the new temperature
I/O:
____________________________________________________________________________*/
float FunctionT(float T)
{
    return(ALPHA*T);
}
/*____________________________________________________________________________
Name:    FunctionT_2
Purpose: return the new temperature
I/O:
____________________________________________________________________________*/
float FunctionT_2(int t, int T0, int outer_loop, int t_pow)
{
    return((float)abs(   (float)T0   /(float)my_pow(outer_loop,t_pow)   *   (float)my_pow((t-
outer_loop),t_pow) ));
}
/*____________________________________________________________________________
Name:    SimulatedAnnealing
Purpose: the Simulated Annealing main procedure
I/O:
____________________________________________________________________________*/
void SimulatedAnnealing(void)
{
    int s_1 = 0;
    int s_2 = 1;
    int t;
    float T;
    int i, j, k;

    worse_sugg_start = 0;
    worse_sugg_end = 0;
    worse_start = 0;
    worse_end = 0;
    InitSAvar(s_1);
    for(i=0;i<DO_NUM_TASKS;++i)
    {
        t = 0;
        T = START_TEMP;
        sa[s_2].quality = BIG_NUMBER;
        for(j=0;j<OUTER_LOOP;++j)
        {
            for(k=0;k<INNER_LOOP;++k)
            {
                Suggested2iChange(s_1);
                if(ic.diff_quality < 0)
                {
                    ChangeSAvar(s_1);
                    if(sa[s_1].quality < sa[s_2].quality)
                        CopySAvar(s_1, s_2);
                }
                else if(ic.diff_quality == 0 && Random(1,100) < 50)
                {
                    ChangeSAvar(s_1);
                    if(sa[s_1].quality < sa[s_2].quality)
                        CopySAvar(s_1, s_2);
                }
                else if(ic.diff_quality > 0)
                {
                    if(j == 0)
                        ++worse_sugg_start;
                    if(j == OUTER_LOOP - 1)
                        ++worse_sugg_end;
                    if(WorseAccepted(ic.diff_quality, T) == 1)
                    {
                        if(j == 0)
                            ++worse_start;
                        if(j == OUTER_LOOP - 1)
                            ++worse_end;
                        ChangeSAvar(s_1);
                        if(sa[s_1].quality < sa[s_2].quality)
                            CopySAvar(s_1, s_2);
                    }
                }
            }//for(k=0;k<INNER_LOOP;++k)
            ++t;
            T = FunctionT(T);
        }//for(j=0;j<OUTER_LOOP;++j)
//        CopySAvar(s_2, s_1);
        UpdateNumTasksPerTruck(s_1);
        PutTaskInResultSa(s_1);
        DeleteTaskFromSAvar(s_1);
        PutNewTaskInSAvar(s_1);
        printf("\nTask no. %d (of %d)", (i+1), DO_NUM_TASKS);
```

```
    }//for(i=0;i<DO_NUM_TASKS;++i)
    printf("\n\nSimulatedAnnealing...");
}

/*************************************************************************\
* Section:
*     procedures/functions for SF_CFC-var, "Shortest time First" and
*     "Closest First with time Constraints"
\*************************************************************************/
/*_____________________________________________________________________
Name:     ShortestClosest
Purpose:  the ShortstFirst/ClosestWithConstraints main procedure
I/O:
_____________________________________________________________________*/
void ShortestClosest(void)
{
    int a;

    InitSFCFCvar(TASK_DEPTH, TIME_DEVIATION);
    for(a=0;a<DO_NUM_TASKS;++a)
    {
        PutTaskInResultSfCfc(sf_cfc.first_truck.truck);
        sf_cfc.truck_location[sf_cfc.first_truck.truck]
sf_cfc.task[sf_cfc.first_truck.truck].to;
        simulation_time = sf_cfc.first_truck.time;
        GetNextTaskSfCfc(sf_cfc.first_truck.truck, TASK_DEPTH, TIME_DEVIATION);
        GetFirstTruckSfCfc();
    }
    printf("\n\nShortestClosest...");
}
/*_____________________________________________________________________
Name:    GetFirstTruckSfCfc
Purpose: get the truck who is first ready
I/O:
_____________________________________________________________________*/
void GetFirstTruckSfCfc(void)
{
    int truck;

    sf_cfc.first_truck.time = BIG_NUMBER;
    for(truck=0;truck<NUM_TRUCKS;++truck)
        if(sf_cfc.task[truck].ready_time < sf_cfc.first_truck.time)
        {
            sf_cfc.first_truck.time = sf_cfc.task[truck].ready_time;
            sf_cfc.first_truck.truck = truck;
        }
}
/*_____________________________________________________________________
Name:    InitSFCFCvar
Purpose: initialize sf_cfc(global)
I/O:
_____________________________________________________________________*/
void InitSFCFCvar(int depth, int diff)
{
    int i;

    for(i=0;i<NUM_TRUCKS;++i)
    {
        sf_cfc.truck_location[i] = trucks[i].location;
        GetNextTaskSfCfc(i, depth, diff);
    }
    GetFirstTruckSfCfc();
}
/*_____________________________________________________________________
Name:    GetNextTaskSfCfc
Purpose: get next task and put it in sf_cfc(global)
I/O:
_____________________________________________________________________*/
void GetNextTaskSfCfc(int truck, int depth, int diff)
{
    int i, closest;
    struct task_struct *current_ptr, *best_ptr;

    closest = BIG_NUMBER;
    current_ptr = tasks_ptr;
    for(i=0;i<depth;++i)
    {
        if(diff <= 0)
        {
            if(storage.time[1+truck][sf_cfc.truck_location[truck]][(*current_ptr).from]    <
closest)
            {
                closest                                                                    =
storage.time[1+truck][sf_cfc.truck_location[truck]][(*current_ptr).from];
                best_ptr = current_ptr;
            }
            current_ptr = (*current_ptr).next_ptr;
        }
        else
        {
            if((*current_ptr).last_time - (*tasks_ptr).last_time <= diff)
            {
                if(storage.time[1+truck][sf_cfc.truck_location[truck]][(*current_ptr).from]
< closest)
                {
                    closest                                                                =
storage.time[1+truck][sf_cfc.truck_location[truck]][(*current_ptr).from];
                    best_ptr = current_ptr;
                }
                current_ptr = (*current_ptr).next_ptr;
            }
            else
                i = BIG_NUMBER;
        }
    }
    sf_cfc.task[truck].next_ptr   = NULL;
    sf_cfc.task[truck].id         = (*best_ptr).id;
    sf_cfc.task[truck].from       = (*best_ptr).from;
    sf_cfc.task[truck].to         = (*best_ptr).to;
    sf_cfc.task[truck].from_block = (*best_ptr).from_block;
    sf_cfc.task[truck].to_block   = (*best_ptr).to_block;
    sf_cfc.task[truck].last_time  = (*best_ptr).last_time;
    sf_cfc.task[truck].ready_time                                                          =
storage.time[1+truck][sf_cfc.truck_location[truck]][(*best_ptr).from]
                                  +
storage.time[1+truck][sf_cfc.task[truck].from][sf_cfc.task[truck].to]
                                  + simulation_time;
    sf_cfc.task[truck].late_time  = sf_cfc.task[truck].ready_time
                                  - sf_cfc.task[truck].last_time;
    DeleteTask(sf_cfc.task[truck].id);
}

/*************************************************************************\
* Section:
*     procedures/functions for basic purpuses
\*************************************************************************/
/*_____________________________________________________________________
Name:    Random
Purpose: return a random number within an interval
I/O:
_____________________________________________________________________*/
int Random(int min, int max)
```

```
{
    int length, round;
    double decimal;

    length = abs(max-min)+1;
    decimal = (double)length/(double)32757*(double)rand()+(double)1;
    round = (int)decimal+min-1;
    if (round > max) //round is sometimes>max due to rounding errors (very rare)
        return(round-1);
    else
        return(round);
}
/*_____________________________________________________________________________
Name:    max
Purpose: return max
I/O:
_____________________________________________________________________________*/
int max(int a, int b)
{
    if(a >= b)
        return(a);
    else
        return(b);
}
/*_____________________________________________________________________________
Name:    min
Purpose: return min
I/O:
_____________________________________________________________________________*/
int min(int a, int b)
{
    if(a <= b)
        return(a);
    else
        return(b);
}
/*_____________________________________________________________________________
Name:    pow
Purpose: return power of
I/O:
_____________________________________________________________________________*/
int my_pow(int a, int b)
{
    int i, result;

    result = 1;
    for(i=0;i<b;++i)
        result = result*a;
    return(result);
}
```