BA

stichting
mathematisch
centrum

MC

AFDELING MATHEMATISCHE BESLISKUNDE BN 16/72 OKTOBER

BA

J.K. LENSTRA
BRANCH-AND-BOUND ALGORITHMEN
VOOR HET HANDELSREIZIGERSPROBLEEM

Voorlopige uitgave

2e boerhaavestraat 49 amsterdam

Printed at the Mathematical Centre, 49, 2e Boerhaavestraat, Amsterdam.

The Mathematical Centre, founded the 11-th of February 1946, is a non-profit institution aiming at the promotion of pure mathematics and its applications. It is sponsored by the Netherlands Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O), by the Municipality of Amsterdam, by the University of Amsterdam, by the Free University at Amsterdam, and by industries.

Inhoud

1. Het handelsreizigersprobleem

Een handelsreiziger wenst vanuit een bepaalde stad n - 1 gegeven steden elk precies éénmaal te bezoeken en tenslotte naar de eerste stad terug te keren. Hij kent de afstand tussen elk geordend paar steden, en vraagt zich af hoe hij moet reizen om de totaal af te leggen afstand zo klein mogelijk te houden.

Het probleem kan ook als volgt worden geformuleerd: bepaal bij een gegeven n×n-matrix $C = (c_{ij})$ een cyclische permutatie $(1,i_2,\ldots,i_n)$ van de getallen $(1,2,\ldots,n)$ zó dat

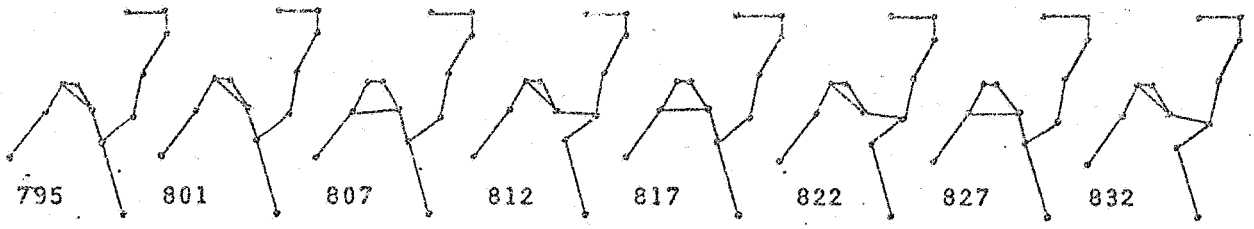
$$c_{1i_2} + c_{i_2i_3} + \ldots + c_{i_n1}$$

minimaal is.

Dit probleem heeft (n - 1)! toegelaten oplossingen, waaronder één of meer optimale. Een algorithme die een optimum bepaalt in een tijd t_n met $t_n = O(n^c)$ voor $n \to \infty$ is tot nu toe niet gevonden. Voor een overzicht van het onderzoek op dit terrein zij verwezen naar [2].

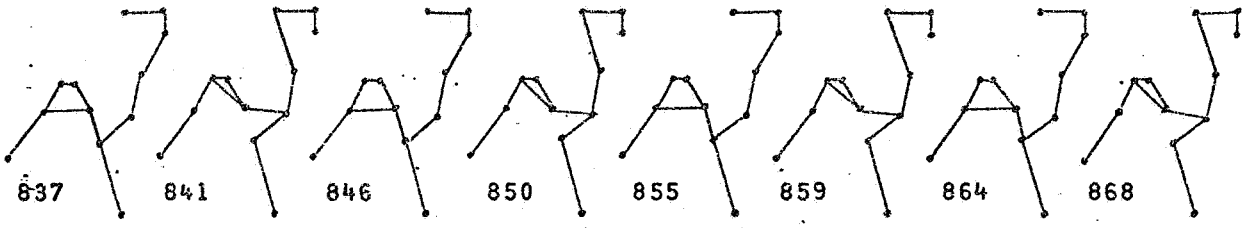
De beste resultaten zijn verkregen met de branch-and-bound algorithmen van EASTMAN [6,7] & SHAPIRO [17], LITTLE, MURTY, SWEENEY & KAREL [15] en HELD & KARP [9,10].

Een branch-and-bound procedure splitst de verzameling van toegelaten oplossingen van een probleem in deelverzamelingen, en berekent een ondergrens voor de waarde van de oplossingen in elke gegenereerde deelverzameling. Door dit proces weer op deelverzamelingen toe te passen ontstaat een boom waarvan elke tak met een deelverzameling en elke vertakking met een splitsing correspondeert. Bij het handelsreizigerspropleem vindt vertakking gewoonlijk plaats door kanten verplicht te stellen of te verbieden.



Eastman & Shapiro beginnen met oplossing van het met het handelsreizigersprobleem corresponderende toewijzingsprobleem. In het vertakkingsproces wordt voor elke tak een ondergrens bepaald door een toewijzingsprobleem op te lossen. Als deze oplossing cyclisch is wordt er niet verder vertakt; in het andere geval wordt voor elke kant van de kortste cykel in de oplossing een nieuwe tak toegevoegd, waarin deze kant wordt verboden. Zodra een cyclische oplossing wordt gevonden met een lengte die niet groter is dan de ondergrenzen van de andere takken is het probleem opgelost.

De algorithme van Little e.a. wordt in §2 beschreven, die van Held en Karp in §3.



2. De algorithme van Little, Murty, Sweeney en Karel

In deze branch-and-bound methode speelt het begrip reductie een belangrijke rol bij het bepalen van de ondergrenzen.

Iedere route bevat één en slechts één element uit elke rij. Dit betekent dat als we van elk element van een rij in de afstandsmatrix een constante h aftrekken de lengte van iedere route met h afneemt. Hierbij blijft een optimale route dus optimaal.

Wanneer we het kleinste element van een rij van elk element van die rij aftrekken zeggen we dat de rij wordt gereduceerd. Een matrix met uitsluitend niet-negatieve elementen en in elke rij en kolom minstens één nul heet een gereduceerde matrix; deze kan worden verkregen door bijvoorbeeld eerst alle rijen en dan alle kolommen te reduceren. Aangezien een gereduceerde matrix geen negatieve elementen bevat is de som van de reductieconstanten een ondergrens voor de lengte van elke route onder de oorspronkelijke matrix.

Het vertakkingsproces kan als volgt worden beschreven.

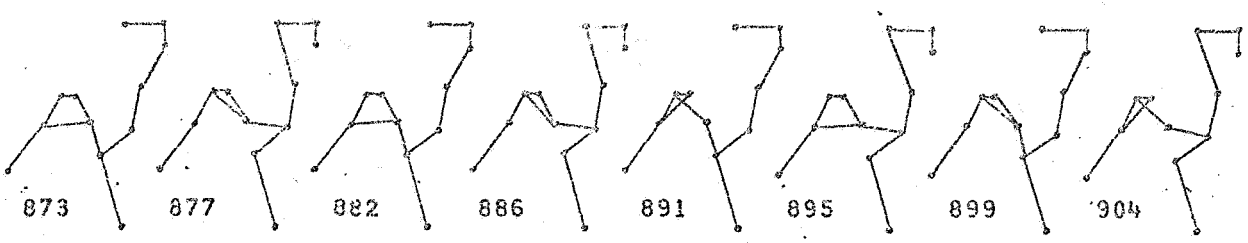
We beginnen met de oorspronkelijke matrix waarbij een ondergrens 0 hoort; de bovengrens, i.e. de lengte van de kortste tot nu toe gevonden route, stellen we op ∞.

Een tak komt alleen voor verdere behandeling in aanmerking als de corresponderende ondergrens onder de bovengrens ligt. De bij deze tak behorende matrix wordt eerst gereduceerd. Als de ondergrens b hierna niet meer kleiner is dan de bovengrens wordt de tak verlaten. Anders kiezen we een kant (s,t) die, naar het schijnt, het slechtst gemist kan worden. Hiertoe maximaliseren we de prijs d_{ij} die we moeten betalen als we niet van i naar j gaan:

$$d_{ij} = \min_{k \neq j} c_{ik} + \min_{k \neq i} c_{kj} ,$$

$$d_{st} = \max_{i,j} d_{ij} .$$

Merk op dat we alleen kanten (i,j) met $c_{ij} = 0$ hoeven te onderzoeken,



daar anders $d_{ij} = 0$. Vervolgens bepalen we de kant (u,v), waarbij v het beginpunt en u het eindpunt is van de langste rij op elkaar aansluitende kanten die (s,t) bevat en verder alleen uit verplichte kanten bestaat.

Als dit proces plaats vindt in een tak waarin reeds n - 2 kanten verplicht zijn gesteld leidt deze tak niet tot verdere vertakking; de n - 2 verplichte kanten vormen samen met de kanten (s,t) en (u,v) een nieuwe, kortere, route van lengte b. De bovengrens wordt gelijk gesteld aan b.

In het andere geval krijgt de boom twee nieuwe takken:

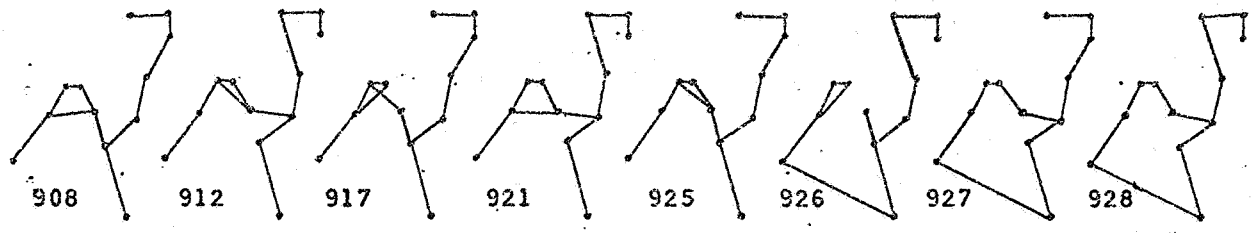
1. een "rechtertak", waarin we de kant (s,t) verplicht stellen, (u,v) verbieden, en rij s en kolom t uit de matrix verwijderen; de ondergrens van deze tak is b;
2. een "linkertak", waarin (s,t) wordt verboden; de ondergrens bedraagt $b + d_{st}$.

Het probleem is opgelost zodra voor geen enkele nog niet onderzochte tak de bijbehorende ondergrens kleiner is dan de bovengrens.

Little noemt verscheidene methoden om een tak voor nader onderzoek te selecteren. Het ligt voor de hand steeds de tak met de kleinste ondergrens te kiezen; dit leidt tot een boom met een minimaal aantal takken. Een andere, door ons gevolgde,methode is: kies steeds eerst de rechtertak, vervolgens alle daaruit voortkomende takken, en dan pas de linkertak. Deze methode maakt het mogelijk de algorithme door middel van een recursieve procedure te beschrijven, vermindert de benodigde hoeveelheid geheugenruimte en heeft voordelen t.a.v. de in elke tak noodzakelijke matrixreductie. We riskeren echter een toename van de rekentijd, vooral als de eerste route, die we door vertakking naar rechts vinden, tamelijk slecht is.

In het branch-and-bound proces komen vier soorten takken voor:

1. takken die nooit worden gekozen;
2. takken die door reductie worden geëlimineerd;
3. takken die tot verdere vertakking leiden;
4. takken waarin een route wordt voltooid.



Als het handelsreizigersprobleem symmetrisch is kan bij elke route T een andere even lange route worden gevonden door T in tegengestelde richting te doorlopen. Van elk dergelijk tweetal routes kan er één worden uitgesloten door de takken aan de uiterste linkerkant van de boom te veranderen.

Stel dat een tak X waarin geen verplichte kanten voorkomen wordt gesplitst in Y, met (i,j) verplicht, en $\overline{Y}$, met (i,j) verboden. De omgekeerden van de routes in Y bevatten de kant (j,i) en behoren, voor zover ze in X zitten, tot $\overline{Y}$; ze kunnen worden uitgesloten door in $\overline{Y}$ naast (i,j) ook (j,i) te verbieden.

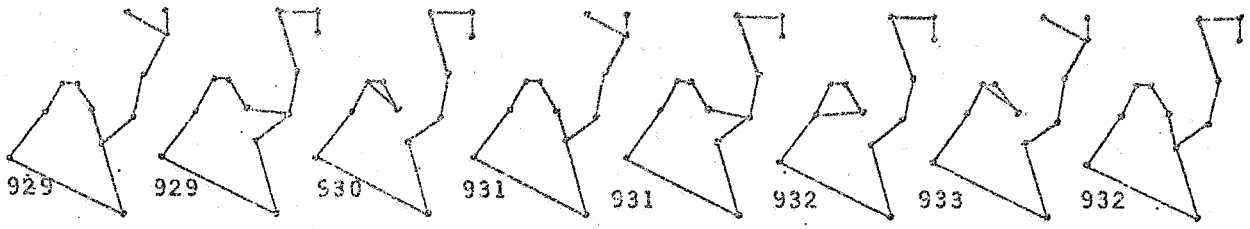
Zodra van een route één kant verplicht is gesteld wordt op deze wijze de tegengestelde route verboden.

Wanneer we bij het onderzoek van een tak een kant (s,t) tegenkomen met een boete d_{st} die minstens gelijk is aan het verschil tussen boven- en ondergrens kunnen we tot vertakking overgaan; er ontstaat dan een linkertak die zeker nooit voor nadere behandeling in aanmerking komt. Pas als er niet zo'n kant blijkt te bestaan behoeven we een kant te bepalen waarvoor de boete maximaal is.

Deze wijziging van de algorithme blijkt de oorspronkelijke rekentijd voor problemen van 20, 25 en 30 steden met 60%, 70% resp. 75% te verminderen.

Toepassing van het hier gevolgde principe: zoek niet het best bereikbare, maar neem het eerste dat voldoende is, geeft ook in andere gevallen een zeer aanzienlijke tijdsbesparing; vergelijk bijvoorbeeld opmerking 2 op pag.14 en Lin [14], 8.3.

In §6 staat de tekst van een ALGOL 60-procedure die handelsreizigersproblemen volgens de hier beschreven algorithme oplost. Deze procedure bleek aanzienlijk efficiënter dan de in [11] en [18] gegeven ALGOL 60programma's voor de methode van Little. Een overzicht van de rekenresultaten vindt men in §7.



3. De algorithme van Held en Karp

Bij het symmetrische handelsreizigersprobleem kunnen we de n steden beschouwen als de hoekpunten van een volledige ongerichte graph; de lengte van kant (i,j) wordt gegeven door c_{ij}.

Een boom is een samenhangende graph zonder cykels. Een 1-boom is een boom over de hoekpunten 2,3,...,n samen met twee kanten aan hoekpunt 1. Een 1-boom bestaat dus uit n kanten en heeft precies één cykel, die hoekpunt 1 bevat.

Held en Karp maken gebruik van de volgende relaties tussen routes en 1-bomen.

a. Het symmetrische handelsreizigersprobleem wordt als zeer moeilijk beschouwd; een minimale 1-boom is eenvoudig te bepalen met behulp van de in §4 beschreven constructie van een minimale boom.

b. Een route is een 1-boom waarin elk hoekpunt graad twee heeft.

c. Als een minimale 1-boom een route is, is het een optimale route.

d. Zij $\pi = (\pi_1,\pi_2,\ldots,\pi_n)$ een reële n-vector. De transformatie

$$c_{ij} := c_{ij} + \pi_i + \pi_j$$

kan de minimale 1-boom veranderen; iedere optimale route blijft echter optimaal, omdat elke route $2\Sigma\pi_i$ langer wordt.

Het is duidelijk dat onder de matrix $(c_{ij} + \pi_i + \pi_j)$ een optimale route niet korter is dan een minimale 1-boom, d.w.z.

$$C^* + 2\Sigma\pi_i \geq \min_k(c_k + \Sigma\pi_i d_{ik}) \qquad (1)$$

Hierbij geldt:

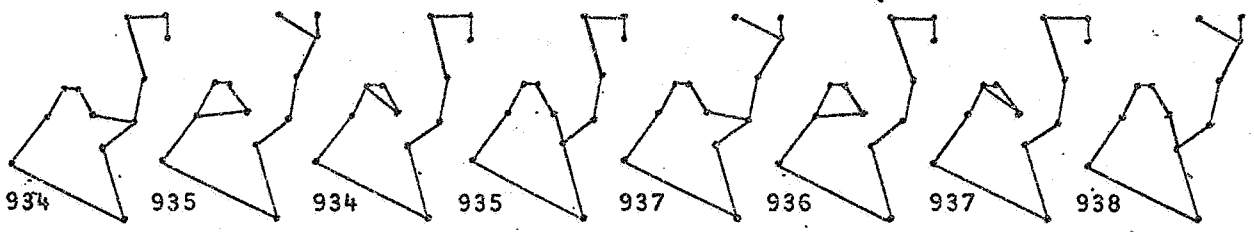
C^* = de lengte van een kortste route onder de oorspronkelijke matrix;

k indiceert de 1-bomen;

c_k = de lengte van de k-de 1-boom onder de oorspronkelijke matrix;

d_{ik} = de graad van hoekpunt i in de k-de 1-boom;

alle sommaties lopen van 1 tot en met n.



We definiëren

$$v_{ik} = d_{ik} - 2;$$

$$w(\pi) = \min_k (c_k + \Sigma \pi_i v_{ik}).$$

Nu is (1) equivalent met

$$C^* \geq w(\pi).$$

Indien we een π^* vinden met

$$C^* = w(\pi^*) = \max_\pi w(\pi)$$

hebben we een situatie waarin een minimale 1-boom een route is; het probleem is dan opgelost. Zo'n π^* hoeft echter niet te bestaan. Held en Karp bewijzen in [9] dat

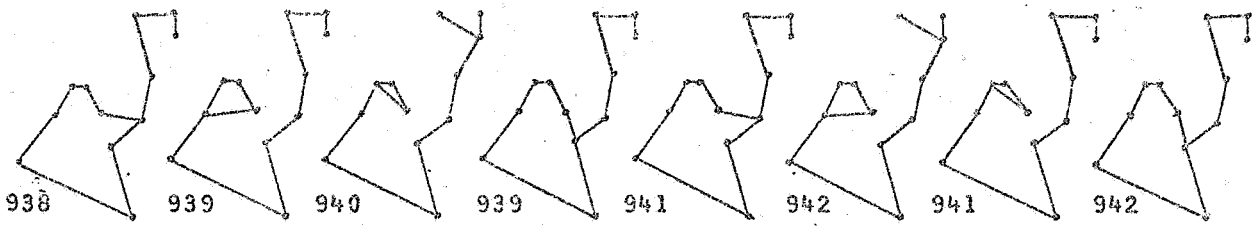
$$C^* = \max_\pi w(\pi)$$

dan en slechts dan als er een geheeltallige optimale oplossing bestaat van een bekend lineair programmeringsprobleem, namelijk het probleem dat wordt verkregen door in een formulering van het handelsreizigersprobleem als geheeltallig l.p.-probleem de integer-restricties weg te laten.

Om $\max_\pi w(\pi)$ te berekenen of te benaderen gebruiken Held en Karp de volgende "ascent"-methode:

1. $\pi_i := 0$, $1 \leq i \leq n$.
2. Construeer een onder $(c_{ij} + \pi_i + \pi_j)$ minimale 1-boom.
   De index van deze 1-boom is $k(\pi)$.
3. $\pi_i := \pi_i + v_{ik(\pi)}$, $1 \leq i \leq n$.
4. Ga naar 2.

Het is met deze methode dat bovenstaand 12-stedenprobleem in 155 iteraties wordt opgelost.



In [10] wordt het verband onderzocht tussen de relaxatiemethode voor lineaire ongelijkheden en het iteratieschema

$$\pi_i := \pi_i + t_m v_{ik(\pi)},\ 1 \leq i \leq n,$$

waarbij m het aantal reeds verrichte iteratiestappen is. Hieruit volgen regels voor de keuze van t_m die de voorkeur verdienen boven de regel:

$$t_m = 1 \text{ voor alle } m.$$

Verder kunnen we een zodanige beginvector π_0 kiezen dat $w(\pi_0)$ niet kleiner is dan de waarde van de optimale oplossing van het toewijzingsprobleem.

Van deze mogelijkheden is geen gebruik gemaakt bij de in [10] en in §7 weergegeven berekeningen; volgens Held en Karp kan er een aanzienlijke tijdsbesparing van worden verwacht.

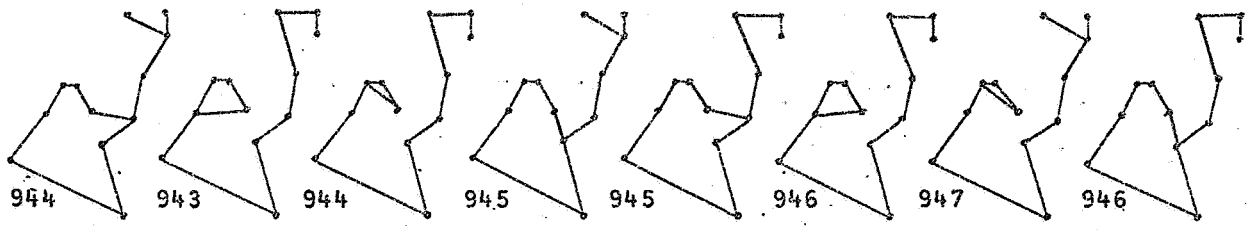
De ascent-methode hoeft $\max_{\pi} w(\pi)$ niet te bereiken, en bovendien kan dit maximum kleiner zijn dan C^*. We combineren daarom de ascent-methode met een branch-and-bound procedure, die we hieronder beschrijven.

Elke tak kunnen we aangeven met (X,Y,π); hierbij is X een verzameling verplicht gestelde kanten, Y een verzameling verboden kanten, en π een reële n-vector. We definiëren $w_{X,Y}(\pi)$ als de lengte van een onder $(c_{ij} + \pi_i + \pi_j)$ minimale 1-boom die alle kanten uit X en geen kanten uit Y bevat.

De bovengrens wordt in het begin op ∞ gesteld, en we nemen als eerste tak $(\emptyset,\emptyset,\vec{0})$.

In een willekeurige tak (X,Y,π) voeren we een ascent-proces uit, met π als beginvector. Dit proces heeft drie mogelijke uitkomsten:

1. Er wordt een minimale 1-boom geconstrueerd met een lengte die niet kleiner is dan de bovengrens. De tak wordt dan verlaten.
2. Er wordt een minimale 1-boom geconstrueerd die een route is. We stellen dan de bovengrens gelijk aan de lengte van deze route, en de tak wordt verlaten.



3. Er ontstaat in p opeenvolgende iteraties geen verhoging van de ondergrens $w_{X,Y}(\pi)$. We breken dan het ascent-proces af en bepalen een vector π' waarvoor $w_{X,Y}(\pi)$ zijn maximum bereikte. De niet-verplichte kanten van een onder $(c_{ij} + \pi'_i + \pi'_j)$ minimale 1-boom worden nu geordend volgens het bedrag waarmee $w_{X,Y}(\pi')$ zou toenemen als de betreffende kant zou worden verboden. Zo krijgen we een rij kanten $(e_1, e_2, \ldots, e_k)$ met

$$w_{X,Y\cup\{e_1\}}(\pi') \geq w_{X,Y\cup\{e_2\}}(\pi') \geq \ldots \geq w_{X,Y\cup\{e_k\}}(\pi') \geq w_{X,Y}(\pi').$$

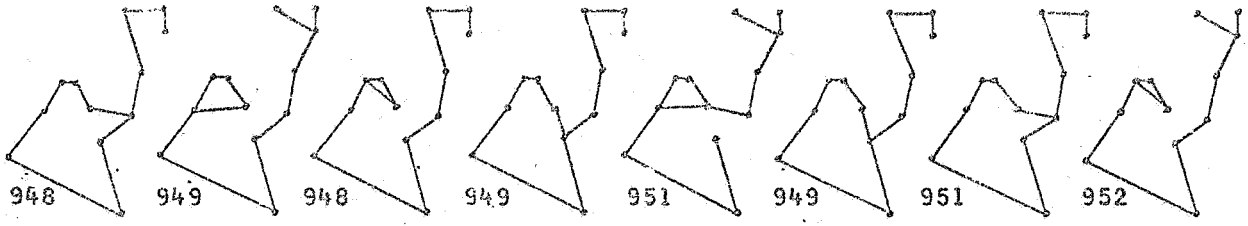
We definiëren

$$\begin{aligned} X_1 &= X & Y_1 &= Y\cup\{e_1\} \\ X_2 &= X\cup\{e_1\} & Y_2 &= Y\cup\{e_2\} \\ X_3 &= X\cup\{e_1,e_2\} & &\vdots \\ &\vdots & Y_{q-1} &= Y\cup\{e_{q-1}\} \\ X_q &= X\cup\{e_1,e_2,\ldots,e_{q-1}\} & Y_q &= Y\cup A_r\cup A_s \end{aligned}$$

Hierbij is q de kleinste index waarvoor een stad r bestaat zó dat X niet twee kanten aan r bevat maar X_q wèl. Het is mogelijk dat een tweede stad, s, dezelfde eigenschap heeft; als dit niet het geval is stellen we s = 0. Voor r > 0 bestaat A_r uit alle kanten aan r die niet tot X_q behoren; $A_0 = \emptyset$. We vormen nu q nieuwe takken (X_i, Y_i, π'). Bij elke tak (X_i, Y_i, π') met $1 \leq i \leq q-1$ behoort een ondergrens $w_{X,Y\cup\{e_i\}}(\pi')$; de ondergrens van (X_q, Y_q, π') bedraagt $w_{X,Y}(\pi')$.

Voor elke tak (X,Y,π) geldt dat als X twee kanten aan een stad r bevat alle andere kanten aan r tot Y behoren. Immers, dit geldt voor de eerste tak $(\emptyset,\emptyset,\vec{0})$, en door de toegepaste vertakkingsstrategie ook voor alle daaruit voortkomende takken.

Zodra voor geen enkele nog niet onderzochte tak de bijbehorende ondergrens kleiner is dan de bovengrens is het probleem opgelost.

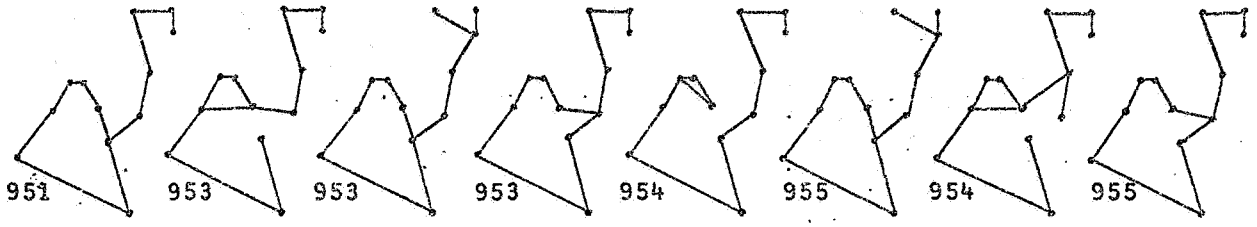


Held en Karp selecteren steeds de tak met de kleinste ondergrens. Wij hebben de algorithme beschreven door middel van een recursieve procedure, waarin de takken van rechts naar links worden behandeld, d.w.z. eerst (X_q, Y_q, π'), dan alle daaruit voortkomende takken, vervolgens (X_{q-1}, Y_{q-1}, π'), enz.

In het branch-and-bound proces komen vier soorten takken voor:

1. takken die nooit worden gekozen;
2. takken die in het ascent-proces worden geëlimineerd;
3. takken die tot verdere vertakking leiden;
4. takken waarin een route wordt gevonden.

De tekst van een ALGOL 60-procedure die symmetrische handelsreizigers-problemen volgens bovenstaande algorithme oplost vindt men in §6. Indien de gebruiker van de procedure dat wenst wordt de beginwaarde van de bovengrens in het vertakkingsproces gelijk aan de lengte van een 3-opt route, geconstrueerd volgens de in §5 beschreven methode van Lin. Het ascent-proces in de eerste tak wordt pas afgebroken als er in 2p opeenvolgende iteraties geen betere ondergrens is verkregen. De keuze p = 10 bleek voor de meeste problemen meer vertakking maar toch een aanzienlijk kortere rekentijd te geven dan de keuze p = 25. Voor een overzicht van de overige rekenresultaten zij verwezen naar §7.



4. Constructie van een minimale boom

We beschouwen een volledige ongerichte graph met n hoekpunten; de lengte van elke kant is gegeven. We zoeken een boom op de n punten van minimale totale lengte. De volgende constructie is ontleend aan DIJKSTRA [5].

Tijdens de constructie worden de kanten onderverdeeld in drie verzamelingen:

S1. de kanten die al zijn toegewezen aan de boom; zij vormen een deelboom;

S2. de kanten waaruit we de volgende kant die aan S1 wordt toegevoegd kiezen;

S3. de overige kanten; zij zijn verworpen of nog niet bekeken.

De punten worden verdeeld in twee verzamelingen:

SA. de punten verbonden door de kanten in S1;

SB. de overige punten; elk van deze punten is met één en slechts één kant uit S2 verbonden aan een punt uit SA.

Bij het begin van de constructie moet gelden:

SA bevat één willekeurig punt als enig element;

S2 bevat alle kanten die in dit punt uitkomen;

S1 is leeg.

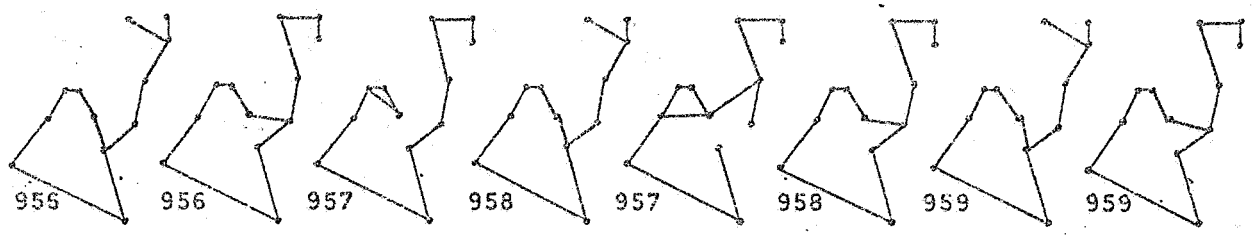
De volgende twee stappen worden nu herhaaldelijk uitgevoerd.

Stap 1. De kortste kant uit S2 wordt uit S2 verwijderd en aan S1 toegevoegd; hierdoor gaat een punt, zeg i, van SB over naar SA.

Stap 2. Voor elk punt b in SB vergelijken we de kant (i,b) met de overeenkomstige kant (a,b) uit S2. Als (i,b) korter is wordt (a,b) verworpen en (i,b) aan S2 toegevoegd; anders wordt (i,b) verworpen.

We herhalen dit proces totdat de verzamelingen S2 en SB leeg zijn. De kanten in S1 vormen dan de gezochte boom.

Deze algorithme, reeds in 1957 door PRIM [16] gepubliceerd, verdient de voorkeur boven de constructies van KRUSKAL [12], omdat rangschikking van alle kanten volgens lengte en controles op cykels en samenhang worden vermeden.



5. De 3-opt methode van Shen Lin

Met een methode van LIN [14] kunnen zeer goede locaal optimale oplossingen van het symmetrische handelsreizigersprobleem worden verkregen.

Lin voert de volgende begrippen in:

a. Een route T heet λ-optimaal (of kortweg λ-opt) als er geen kortere route kan ontstaan door een verzameling van λ kanten van T door een andere verzameling van λ kanten te vervangen.

b. Een route T heet bestand tegen omkering als er geen kortere route kan ontstaan door een rij opeenvolgende steden in T om te keren.

c. Een route T heet bestand tegen omkering en invoeging als er geen kortere route kan ontstaan door een rij opeenvolgende steden in T uit T te verwijderen en, al of niet omgekeerd, tussen twee opeenvolgende overgebleven steden in T in te voegen.

Voor elke route T geldt:

1. T is 1-opt.
2. T is 2-opt $\Leftrightarrow$ T is bestand tegen omkering.
3. T is 3-opt $\Leftrightarrow$ T is bestand tegen omkering en invoeging.
4. T is n-opt $\Leftrightarrow$ T is optimaal
5. T is λ-opt $\Leftrightarrow$ T is κ-opt voor $1 \leq \kappa \leq \lambda$.

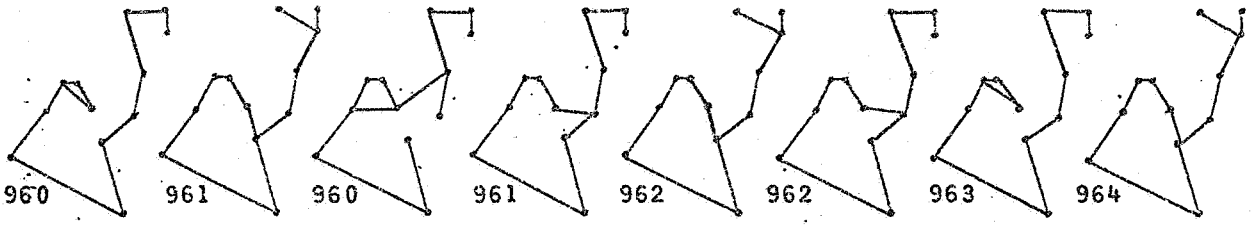
Een route die wel (n-1)-opt maar niet optimaal is heeft met geen enkele optimale route een kant gemeen. Het bestaan van zo'n route lijkt zeer onwaarschijnlijk. Lin bewijst dat het vermoeden

6. T is (n-1)-opt $\Leftrightarrow$ T is optimaal

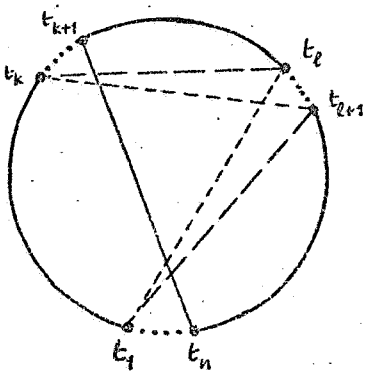
equivalent is met

6'. Gegeven is een graph met n hoekpunten en2n kanten en met de eigenschap dat de verzameling van kanten verdeeld kan worden in twee verzamelingen van n kanten, die elk een Hamilton-circuit vormen. Dan bestaat er nog zo'n verdeling met dezelfde eigenschap.

Voor $n \leq 7$ is 6' eenvoudig te bewijzen.



Onderstaande constructie van een 3-opt route uit een willekeurige beginroute is gebaseerd op eigenschap 3. We gaan op systematische wijze na of de route bestand is tegen omkering en invoeging. Telkens als een verbetering wordt gevonden herstarten we dit proces met de nieuwe route als beginroute; om te voorkomen dat de nieuwe kanten direct voor vervanging in aanmerking komen wordt de verbeterde route eerst geroteerd (zie F). Het proces eindigt met een 3-opt route wanneer geen verdere verbeteringen worden bereikt.



A. Kies een beginroute $(t_1,t_2,\ldots,t_n)$.

B. Voer C uit voor $j = 1,2,\ldots,n$.

C. 1. Voer D uit voor $k = 1,2,\ldots,n-3$.

2. Roteer de route:

$(t_1,t_2,\ldots,t_n) := (t_n,t_1,\ldots,t_{n-1})$.

D. Voer E uit voor $\ell = k+1,k+2,\ldots,n-1$.

E. $c_0 := c_{t_1t_n} + c_{t_kt_{k+1}} + c_{t_\ell t_{\ell+1}}$;

$c_1 := c_{t_{k+1}t_n} + c_{t_1t_\ell} + c_{t_kt_{\ell+1}}$;

$c_2 := c_{t_{k+1}t_n} + c_{t_1t_{\ell+1}} + c_{t_kt_\ell}$;

voer F uit indien $c_1 < c_0$ of $c_2 < c_0$.

F. Verander de route:

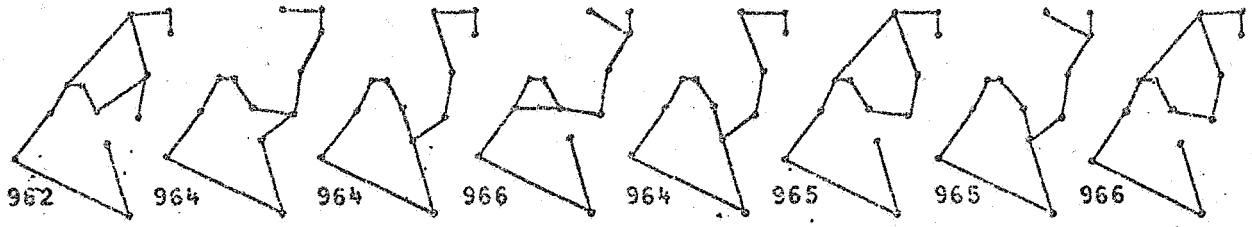
als $c_1 \leq c_2$: $(t_1,t_2,\ldots,t_n) := (t_{\ell+2},\ldots,t_n,t_{k+1},\ldots,t_\ell,t_1,\ldots,t_k,t_{\ell+1})$;

anders: $(t_1,t_2,\ldots,t_n) := (t_{\ell+2},\ldots,t_n,t_{k+1},\ldots,t_\ell,t_k,\ldots,t_1,t_{\ell+1})$.

Begin opnieuw met B.

Deze algorithme is efficiënt in die zin, dat de verwachte rekentijd rechtevenredig is met n^3.

We gaan hier voorbij aan de wijze waarop Lin bij de constructie van de (m+1)ste 3-opt route gebruik maakt van de m reeds verkregen locale optima.



Opmerking 1

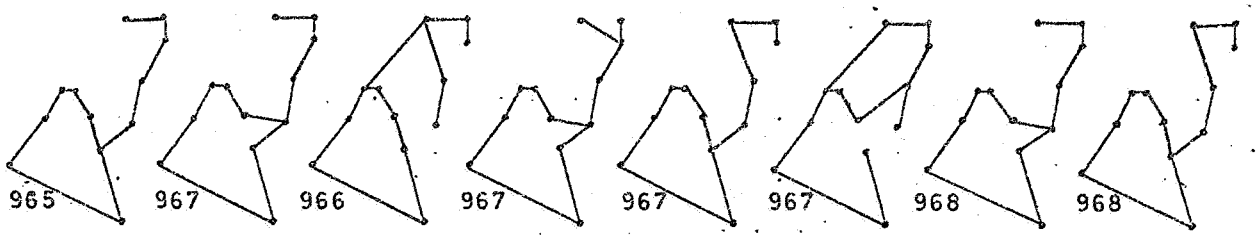
Stel dat een λ-opt route in een tijd t_λ kan worden verkregen, en dat deze route met kans p_λ optimaal is. Als we beschikken over een totale rekentijd t kunnen we $\lfloor t/t_\lambda \rfloor = k_\lambda$ van deze locale optima construeren. De kans dat zich hieronder een optimum bevindt is

$$p_\lambda^* = 1 - (1 - p_\lambda)^{k_\lambda}.$$

Op empirische gronden meent Lin te mogen aannemen dat bij problemen met tussen de 10 en 100 steden p_λ^* voor λ= 3 zijn maximum aanneemt.

Opmerking 2

De algorithme bereikt een aanzienlijke tijdsbesparing door niet steeds naar de grootst mogelijke verbetering te zoeken maar telkens de verbetering te nemen die het eerst optreedt. Dit principe is van algemene toepassing; zie bijvoorbeeld pag.9.



6. De ALGOL 60-procedures

In deze paragraaf vindt men de tekst van de ALGOL 60-procedures

little et al (sym,n,c,nc,nr,nb,nt),
held and karp (a,n,c,b,nc,na,nb,nt).

Een gebruiksaanwijzing voor deze procedures volgt hieronder.

Bij een aanroep moet elke actuele parameter natuurlijk van het type zijn dat voor de overeenkomstige formele parameter in de proceduredeclaratie is gespecificeerd. Steeds is n het aantal steden en c een n×n-matrix waarbij c[i,j] de afstand van i naar j aangeeft.

De aanroep

little et al (sym,n,c,nc,nr,nb,nt)

waarbij

sym = true $\Leftrightarrow$ c[i,j] = c[j,i] voor $1 \leq i,j \leq n$

heeft het volgende effect:

a. Het handelsreizigersprobleem wordt opgelost volgens de in §2 beschreven algorithme.

b. De voltooiing van een route $(i_1,i_2,\ldots,i_n)$ veroorzaakt een aanroep

printtour (rop,ub,v)

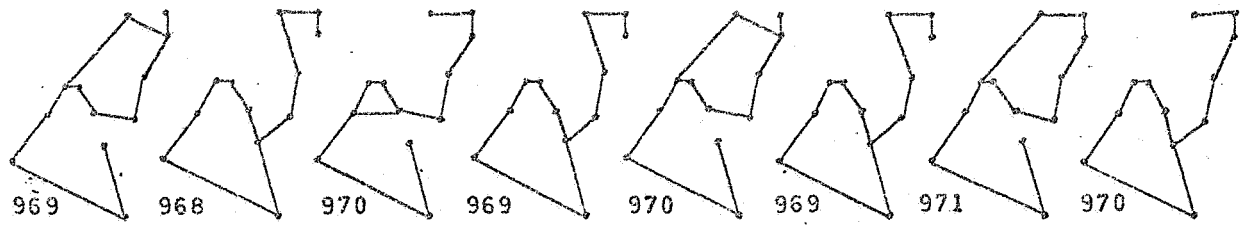
met

ub = de lengte van de route,
$v = i_1$,
$rop[i_j] = i_{j+1}$ voor $1 \leq j \leq n-1$, $rop[i_n] = 0$.

Declaratie van deze outputprocedure:

```
procedure printtour (rop,ub,v); value rop,ub,v;
integer ub,v; integer array rop;
<procedure body>
```

dient buiten die van "little et al" te geschieden.



c. Na uitvoering van de procedure geldt:

de laatst gevonden route is optimaal;

nc = het aantal linkertakken dat niet is gekozen;

nr = het aantal takken dat tijdens de reductie is geëlimineerd;

nb = het aantal takken dat tot verdere vertakking heeft geleid;

nt = het aantal takken waarin een route is voltooid.

De aanroep

held and karp (a,n,c,b,nc,na,nb,nt)

heeft het volgende effect:

a. Het symmetrische handelsreizigersprobleem wordt opgelost volgens de in §3 beschreven algorithme. De parameter p, die de beeindiging van het ascent-proces regelt, wordt bepaald door

$$p = |a|.$$

Indien a > 0 krijgt de bovengrens in het branch-and-bound proces 10^6 als beginwaarde; indien a < 0 wordt deze beginwaarde gelijk aan de lengte van een 3-opt route, geconstrueerd volgens de in §5 beschreven algorithme met (1,2,...,n) als beginroute.

b. De constructie van een 3-opt route veroorzaakt een aanroep

printtour (t,ub,-1);

het vinden van een route $(i_1,i_2,\ldots,i_n)$ met xx verplichte kanten heeft een aanroep

printtour (t,ub,xx)

tot gevolg. In beide gevallen geldt:

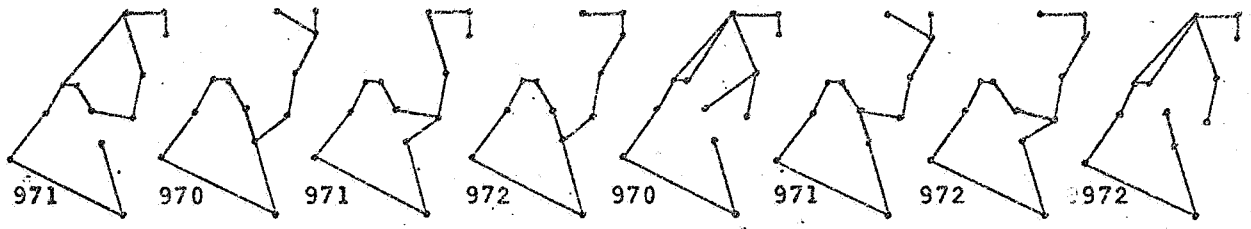
ub = de lengte van de route,

$t[i_j] = i_{j+1}$ voor $1 \leq j \leq n-1$, $t[i_n] = i_1$.

Declaratie van deze outputprocedure:

```
procedure printtour (t,ub,xx); value t,ub,xx;
integer ub,xx; integer array xx;
<procedure body>
```

dient buiten die van "held and karp" te geschieden.



c. Na uitvoering van de procedure geldt:

de laatst gevonden route is optimaal;

b = de hoogste ondergrens die in het eerste ascent-proces is gevonden;

nc = het aantal takken dat niet is gekozen;

na = het aantal takken dat tijdens het ascent-proces is geëlimineerd;

nb = het aantal takken dat tot verdere vertakking heeft geleid;

nt = het aantal takken waarin een route is gevonden.

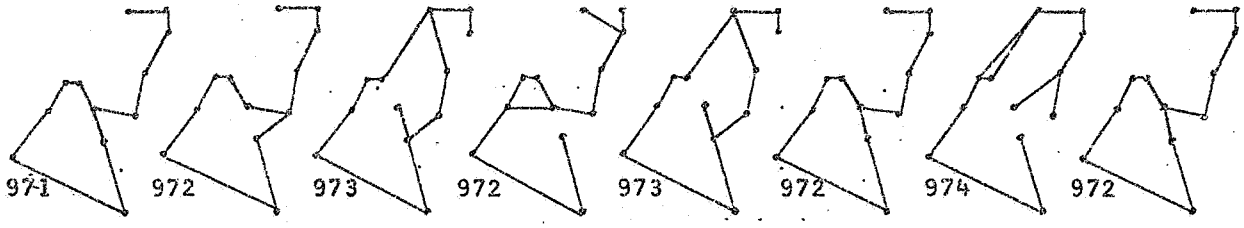
```
PROCEDURE LITTLE ET AL(SYM,N,C,NC,NR,NB,NT); VALUE SYM,N;
BOOLEAN SYM; INTEGER N,NC,NR,NB,NT; INTEGER ARRAY C;
BEGIN INTEGER G, H, I, J, TPS, UPS, LB, UB;
    INTEGER ARRAY ROP, COP, ROWMIN, COLMIN[1:N];

    PROCEDURE NODE(A,Z); VALUE A,Z; BOOLEAN Z; INTEGER A;
    BEGIN INTEGER LLB, S, T, U, V;
        IF Z THEN
        BEGIN
          FOR I:= 1 STEP 1 UNTIL N DO IF ROP[I] = 0 THEN
          BEGIN U:= 10^7;
            FOR J:= 1 STEP 1 UNTIL N DO IF COP[J] = 0 THEN
            BEGIN H:= C[I,J]; IF H < U THEN U:= H END;
            IF U ≠ 0 THEN
            BEGIN LB:= LB + U;
              FOR J:= 1 STEP 1 UNTIL N DO C[I,J]:= C[I,J] - U
            END
          END;
          IF LB ≥ UB THEN BEGIN NR:= NR + 1; GOTO END END
        END  ELSE
        FOR I:= 1 STEP 1 UNTIL N DO
        IF I = UPS ∨ ROP[I] = 0 ∧ C[I,TPS] = U THEN
        BEGIN U:= 10^7;
            FOR J:= 1 STEP 1 UNTIL N DO IF COP[J] = 0 THEN
            BEGIN H:= C[I,J]; IF H < U THEN U:= H END;
            IF U ≠ 0 THEN
            BEGIN H:= LB + U;
                IF H ≥ UB THEN BEGIN NR:= NR + 1; GOTO END END;
                LB:= H;
                FOR J:= 1 STEP 1 UNTIL N DO C[I,J]:= C[I,J] - U
            END
        END;
        G:= UB - LB;
        FOR J:= 1 STEP 1 UNTIL N DO IF COP[J] = 0 THEN
        BEGIN U:= V:= 10^7;
            FOR I:= 1 STEP 1 UNTIL N DO IF ROP[I] = 0 THEN
            BEGIN H:= C[I,J];
                IF H < U THEN BEGIN V:= U; U:= H END ELSE
                IF H < V THEN BEGIN V:= H; IF V=0 THEN GOTO CM END
            END;
            IF U ≠ 0 THEN
            BEGIN H:= LB + U;
                IF H ≥ UB THEN BEGIN NR:= NR + 1; GOTO END END;
                LB:= H; G:= G - U; V:= V - U;
                FOR I:= 1 STEP 1 UNTIL N DO C[I,J]:= C[I,J] - U
            END;
        CM: IF V ≥ G THEN
            BEGIN LLB:= V; T:= J;
                FOR I:= 1 STEP 1 UNTIL N DO IF ROP[I] = 0 THEN
                BEGIN IF C[I,J] = 0 THEN
                    BEGIN S:= I; GOTO LITTLE END
                END
            END       ELSE COLMIN[J]:= V
        END;
```

```
            FOR I:= 1 STEP 1 UNTIL N DO IF ROP[I] = 0 THEN
            BEGIN U:= V:= 10↑7;
                FOR J:= 1 STEP 1 UNTIL N DO IF COP[J] = 0 THEN
                BEGIN H:= C[I,J];
                    IF H < U THEN BEGIN V:= U; U:= H END ELSE
                    IF H < V THEN BEGIN V:= H; IF V=0 THEN GOTO RM END
                END;
            RM: IF V ≥ G THEN
                BEGIN LLB:= V; S:= I;
                    FOR J:= 1 STEP 1 UNTIL N DO IF COP[J] = 0 THEN
                    BEGIN IF C[I,J] = 0 THEN
                        BEGIN T:= J; GOTO LITTLE END
                    END
                END       ELSE ROWMIN[I]:= V
            END;
            LLB:= -1;
            FOR I:= 1 STEP 1 UNTIL N DO IF ROP[I] = 0 THEN
            BEGIN V:= ROWMIN[I];
                FOR J:= 1 STEP 1 UNTIL N DO IF COP[J] = 0 THEN
                BEGIN IF C[I,J] = 0 THEN
                    BEGIN H:= V + COLMIN[J]; IF H > LLB THEN
                        BEGIN LLB:= H; S:= I; T:= J;
                            IF LLB ≥ G THEN GOTO LITTLE
                        END
                    END
                END
            END;
LITTLE:     ROP[S]:= V:= T; COP[T]:= U:= S;
            FOR H:= ROP[U] WHILE H ≠ 0 DO U:= H;
            FOR H:= COP[V] WHILE H ≠ 0 DO V:= H;
            IF A = 2 THEN
            BEGIN NT:= NT + 1; UB:= LB; PRINTTOUR(ROP,UB,V);
                ROP[S]:= COP[T]:= 0
            END A=2  ELSE
            BEGIN NB:= NB + 1; LLB:= LB + LLB; TPS:= T; UPS:= U;
                C[U,V]:= C[U,V] + 10↑7;
                NODE(A + 1,FALSE);
                C[U,V]:= C[U,V] - 10↑7;
                ROP[S]:= COP[T]:= 0;
                IF LLB < UB THEN
                BEGIN BOOLEAN SYMN; SYMN:= SYM ^ A=N;
                    C[S,T]:=          10↑7;
                    IF SYMN THEN C[T,S]:= C[T,S] + 10↑7;
                    NODE(A,TRUE);
                    C[S,T]:= C[S,T] - 10↑7;
                    IF SYMN THEN C[T,S]:= C[T,S] - 10↑7
                END          ELSE NC:= NC + 1
            END A>2;
END:END NODE;

    NC:= NR:= NB:= NT:= LB:= 0; UB:= 10↑7;
    FOR I:= 1 STEP 1 UNTIL N DO
    BEGIN C[I,I]:= 10↑7; ROP[I]:= COP[I]:= 0 END;
    NODE(N,TRUE)
END LITTLE ET AL;
```

```
PROCEDURE HELD AND KARP(A,N,C,B,NC,NA,NB,NT); VALUE A,N;
INTEGER A,N,B,NC,NA,NB,NT; INTEGER ARRAY C;
BEGIN BOOLEAN TOUR; INTEGER G, H, I, J, MXX, LB, UB;
    INTEGER ARRAY X, D, T, V, EE, FF, WW[1:N];

    INTEGER PROCEDURE W3OPTTOUR;
    BEGIN INTEGER G, H, I, J, K, L, N3, N1,
        T1, TN, TK, TKK, TL, TLL, CJ, CK, CL, C1, C2, CKKN;
        FOR I:= 1 STEP 1 UNTIL N DO T[I]:= I; N3:= N-3; N1:= N-1;
LIN:    FOR J:= 1 STEP 1 UNTIL N DO
        BEGIN T1:= T[1]; TN:= T[N]; CJ:= C[T1,TN];
          FOR K:= 1 STEP 1 UNTIL N3 DO
          BEGIN TK:= T[K]; TKK:= T[K + 1]; CK:= CJ + C[TK,TKK];
            CKKN:= C[TKK,TN];
            FOR L:= K + 1 STEP 1 UNTIL N1 DO
            BEGIN TL:= T[L]; TLL:= T[L + 1]; CL:= CK + C[TL,TLL];
              C1:= C[T1,TL] + C[TK,TLL] + CKKN;
              C2:= C[T1,TLL] + C[TK,TL] + CKKN;
              IF C1 < CL ∨ C2 < CL THEN
              BEGIN FOR I:= 1 STEP 1 UNTIL N DO V[I]:= T[I];
                T[N]:= TLL;
                G:= L + 1; H:= N - G;
                FOR I:=      1 STEP 1 UNTIL H DO T[I]:= V[I + G];
                G:= K - H; H:= L - G;
                FOR I:= N - L STEP 1 UNTIL N DO T[I]:= V[I + G];
                FOR I:= 1 STEP 1 UNTIL K DO
                T[IF C1 ≤ C2 THEN H + I ELSE N - I]:= V[I];
                GOTO LIN
              END
            END
          END;
          FOR I:= N STEP -1 UNTIL 2 DO T[I]:= T[I - 1]; T[1]:= TN
        END;
        G:= 0; TN:= T[N]; FOR I:= 1 STEP 1 UNTIL N DO
        BEGIN T1:= T[I]; V[TN]:= T1; G:= G+C[TN,T1]; TN:= T1 END;
        W3OPTTOUR:= G
    END W3OPTTOUR;

    INTEGER PROCEDURE WMIN1TREE;
    BEGIN REAL G; INTEGER H, I, D1, T1, D0, T0; D1:= ₁₀7;
        FOR I:= 2 STEP 1 UNTIL N DO
        BEGIN D[I]:= ₁₀7; T[I]:= 0; H:= C[1,I];
            IF H < D1 THEN
              BEGIN D0:= D1; T0:= T1; D1:= H; T1:= I END
            ELSE IF H < D0 THEN BEGIN D0:= H; T0:= I END
        END;
        D[1]:= D1; T[1]:= T1; D[T0]:= D0; T[T0]:= -1; G:= D1;
PRIM:   T[T0]:= -T[T0]; G:= G + D0; T1:= T0; T0:= 0; D0:= ₁₀7;
        FOR I:= 2 STEP 1 UNTIL N DO IF T[I] ≤ 0 THEN
        BEGIN H:= IF I < T1 THEN C[I,T1] ELSE C[T1,I];
            IF H < D[I] THEN BEGIN D[I]:= H; T[I]:= -T1 END
            ELSE H:= D[I];
            IF H < D0   THEN BEGIN T0  := I; D0  := H   END
        END;
        IF T0 > 0 THEN GOTO PRIM;
        WMIN1TREE:= G + MXX
    END WMIN1TREE;
```

```
        PROCEDURE ADJUST(R,M); VALUE R,M; INTEGER R,M;
        BEGIN FOR I:= R - 1 STEP -1 UNTIL 1 DO
              IF C[R,I] > -5.5 THEN C[R,I]:= C[R,I] + M;
              FOR I:= R + 1 STEP  1 UNTIL N DO
              IF C[I,R] > -5.5 THEN C[I,R]:= C[I,R] + M
        END ADJUST;

        PROCEDURE NODE(XX,PI,FIRST NODE); VALUE XX,PI,FIRST NODE;
        BOOLEAN FIRST NODE; INTEGER XX; INTEGER ARRAY PI;
        BEGIN INTEGER Q1, R, S, K, EK, FK; INTEGER ARRAY PIE[1:N];
            FOR I:= 2 STEP 1 UNTIL N DO
            BEGIN H:= PI[I];
                FOR J:= I - 1 STEP -1 UNTIL 2 DO
                C[J,I]:= C[I,J] + H + PI[J];
                C[1,I]:= C[I,1] + H
            END;
            MXX:= .6 * XX; LB:= -.7;
            S:= IF FIRST NODE THEN 2 * A ELSE A;
ASCENT:     G:= WMIN1TREE;
            IF G > LB THEN
            BEGIN IF FIRST NODE THEN B:= G;
                IF G ≥ UB THEN BEGIN NA:= NA + 1; GOTO END END;
                K:= 0; LB:= G;
                FOR I:= 2 STEP 1 UNTIL N DO PIE[I]:= PI[I]
            END       ELSE
            BEGIN K:= K + 1; IF K = S THEN GOTO HELD END;
            TOUR:= TRUE;
            FOR I:= 1 STEP 1 UNTIL N DO V[I]:= -1;
            FOR I:= 1 STEP 1 UNTIL N DO
            BEGIN J:= T[I]; V[J]:= V[J] + 1 END;
            FOR I:= 2 STEP 1 UNTIL N DO IF V[I] ≠ 0 THEN
            BEGIN TOUR:= FALSE;
                H:= V[I]; PI[I]:= PI[I] + H;
                FOR J:= I - 1 STEP -1 UNTIL 1 DO C[J,I]:= C[J,I] + H;
                FOR J:= I + 1 STEP  1 UNTIL N DO C[I,J]:= C[I,J] + H
            END;
            IF TOUR THEN
            BEGIN UB:= G; PRINTTOUR(T,UB,XX);
                NT:= NT + 1; GOTO END
            END;
            GOTO ASCENT;
HELD:       NB:= NB + 1;
            FOR I:= 2 STEP 1 UNTIL N DO
            BEGIN H:= PIE[I];
                FOR J:= I - 1 STEP -1 UNTIL 2 DO
                C[J,I]:= C[I,J] + H + PIE[J];
                C[1,I]:= C[I,1] + H
            END;
            WMIN1TREE;
```

```
            FOR I:= 1 STEP 1 UNTIL N DO V[I]:= T[I]; K:= 0;
            FOR I:= 1 STEP 1 UNTIL N DO IF C[I,V[I]] > -5.5 THEN
            BEGIN K:= K + 1; P[K]:= I;
                FK:= V[I];
                IF I > FK THEN EK:= I ELSE BEGIN EK:= FK; FK:= I END;
                H:= C[FK,EK]; C[FK,EK]:= H + .6;
                EE[I]:= EK; FF[I]:= FK; WW[I]:= G:= WMIN1TREE;
                FOR J:= K - 1 STEP -1 UNTIL 1 DO
                IF WW[P[J]] ≥ G THEN GOTO NEXTI ELSE
                BEGIN P[J + 1]:= P[J]; P[J]:= I END;
                NEXTI:          C[FK,EK]:= H
            END;
            FOR Q1:= 1 STEP 1 UNTIL K DO
            BEGIN I:= P[Q1]; EK:= EE[I]; FK:= FF[I];
                C[EK,FK]:= C[EK,FK] - .6;
                X[EK]:= G:= X[EK] + 1; X[FK]:= H:= X[FK] + 1;
                IF G = 2 THEN
                BEGIN R:= EK; S:= IF H = 2 THEN FK ELSE 0;
                    GOTO KARP
                END;
                IF H = 2 THEN
                BEGIN R:= FK; S:= 0;
                    GOTO KARP
                END
            END;
KARP:       BEGIN INTEGER ARRAY E, F, W[1:Q1];
                FOR K:= 1 STEP 1 UNTIL Q1 DO
                BEGIN I:= P[K];
                    E[K]:= EE[I]; F[K]:= FF[I]; W[K]:= WW[I]
                END;
                ADJUST(R, .6); IF S > 0 THEN ADJUST(S, .6);
                NODE(XX + Q1,PIE,FALSE);
                ADJUST(R,-.6); IF S > 0 THEN ADJUST(S,-.6);
                FOR K:= Q1 STEP -1 UNTIL 1 DO
                BEGIN EK:= E[K]; FK:= F[K];
                    C[EK,FK]:= C[EK,FK] + 2.6;
                    X[EK]:= X[EK] - 1; X[FK]:= X[FK] - 1;
                    IF W[K] ≥ UB THEN NC:= NC + 1 ELSE
                    NODE(XX + K - 1,PIE,FALSE);
                    C[EK,FK]:= C[EK,FK] -  .6
                END
            END;
END;END NODE;

    IF A > 0 THEN UB:= .6 ELSE
    BEGIN UB:= W3OPTTOUR; PRINTTOUR(V,UB,-1); A:= -A END;
    NC:= NA:= NB:= NT:= 0; FOR I:= 1 STEP 1 UNTIL N DO X[I]:= 0;
    NODE(0,X,TRUE)
END HELD AND KARP;
```



7. Resultaten

In deze paragraaf geven we enige resultaten die zijn behaald met de procedures "little et al" en "held and karp".

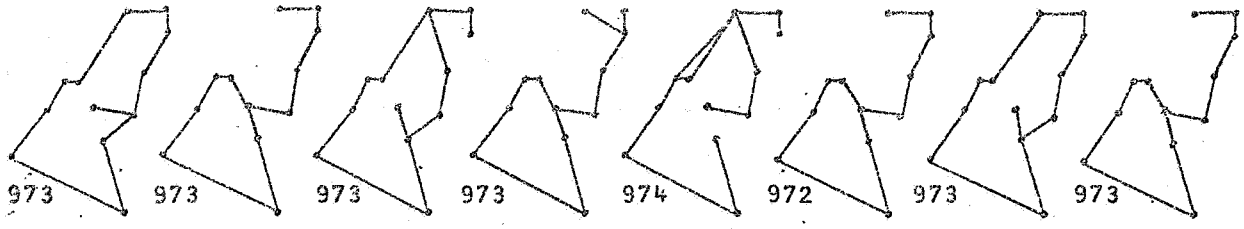
Verscheidene van onze testproblemen zijn aan de literatuur ontleend. De afstandsmatrix van het probleem "Nederland" vindt men in tabel 1. Voor de met "R" aangegeven problemen zijn de c_{ij} aselect getrokken uit een discrete uniforme verdeling over $\{0,1,\ldots,100\}$. Voor de met "RE" aangegeven problemen zijn de n punten aselect in een vierkant met zijde 100 geplaatst; c_{ij} is de Euclidische afstand tussen de punten i en j.

In figuur 1, 3 en 4 staan afbeeldingen van de graphen G9, G12, G15, G16, G22 en de "Tutte-graph". Bij elk van deze graphen kunnen we als volgt een afstandsmatrix definiëren:

$$c_{ij} = \begin{cases} 0 & \text{als de graph de kant } (i,j) \text{ bevat;} \\ 1 & \text{anders.} \end{cases}$$

De graph bevat een Hamilton-circuit dan en slechts dan als het zo verkregen handelsreizigersprobleem een kortste route van lengte 0 heeft. De graph G22 is ontstaan uit Euler's bekende probleem "de zeven bruggen van Koningsbergen" (figuur 2); het bestaan van een tocht waarbij elke brug precies éénmaal wordt overgestoken is equivalent met het bestaan van een Hamilton-circuit in G22. In het probleem "Paard" zijn de steden de velden van het schaakbord, met afstand 0 tussen velden die één paardesprong van elkaar liggen, en 1 anders.

De tabellen 2 en 3 geven gedetailleerde resultaten. De betekenis van enige van de weergegeven grootheden kan men in §6 vinden. De rekentijden zijn gemeten in seconden. De tijden van de methode van Held en Karp voor de problemen "Paard" en "Tutte" zijn geschat met behulp van de in [10] genoemde tijden. Alle berekeningen zijn uitgevoerd op de Electrologica-X8 rekenautomaat van het Mathematisch Centrum.



Op grond van de resultaten komen we tot de volgende conclusies:

a. Voor Euclidische problemen is de methode van Held en Karp aanzienlijk veel beter dan de methode van Little e.a.

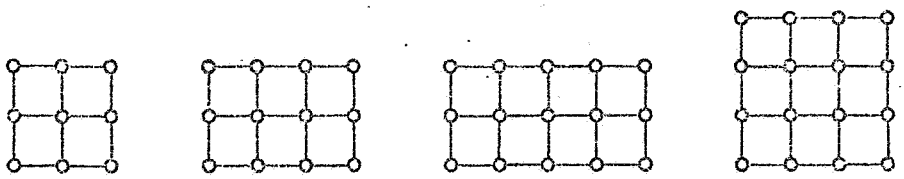
b. Voor Hamilton-circuits is de situatie precies omgekeerd.

c. Voor grote symmetrische random problemen is de methode van Held en Karp te verkiezen boven die van Little; het verschil is echter minder sprekend dan onder a.

d. Voor asymmetrische problemen werkt de methode van Little beter dan voor symmetrische.

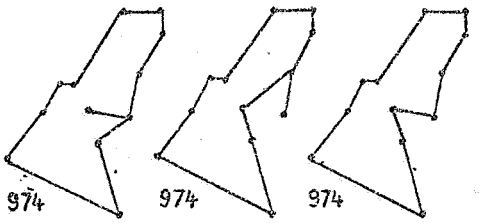
Amsterdam	1	0											
Arnhem	2	95	0										
Assen	3	180	140	0									
Den Haag	4	60	120	220	0								
Groningen	5	198	170	28	250	0							
Haarlem	6	22	115	205	45	204	0						
's-Hertogenbosch	7	90	65	206	105	230	100	0					
Leeuwarden	8	137	160	70	190	58	143	220	0				
Maastricht	9	202	153	290	190	320	217	115	310	0			
Middelburg	10	180	205	320	125	350	170	130	310	185	0		
Utrecht	11	40	65	160	65	190	55	55	175	166	160	0	
Zwolle	12	115	65	74	155	100	130	130	90	220	240	90	0
		1	2	3	4	5	6	7	8	9	10	11	12

Tabel 1. De afstandsmatrix van een 12-stedenprobleem

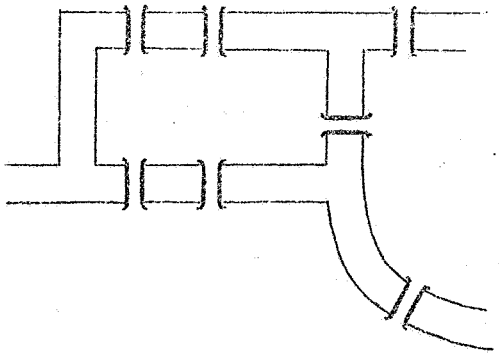


G9 G12 G15 G16

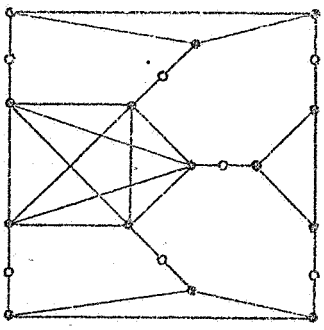
Figuur 1.



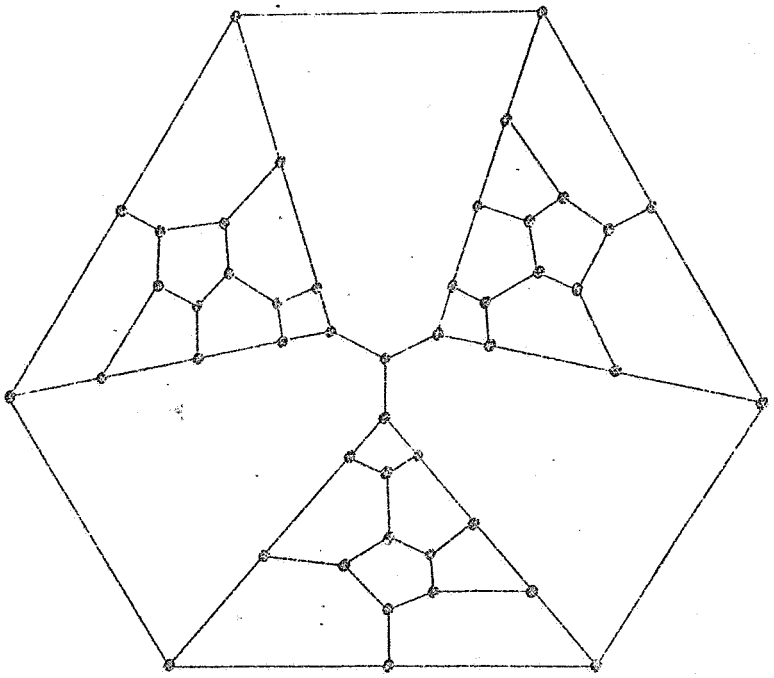
Figuur 0.
Oplossing van het 12-stedenprobleem uit Tabel 1 met behulp van de ascent-methode



Figuur 2. De zeven bruggen van Koningsbergen



Figuur 3. G22
(Het meest linkse hoekpunt is verbonden met alle punten van het type •)



Figuur 4. De Tutte-graph

PROBLEEM	n	C*	LITTLE ET AL nc	nr	nb	nt	tijd	HELD AND KARP b	a = +10 nc	na	nb	nt	tijd	a = -10 lin	nc	na	nb	nt	tijd
EUCLIDISCH																			
Barachet [1]	10	378	18	4	22	1	2	378	7	1	4	1	11	378	0	1	0	0	5
Nederland	12	974	205	79	287	4	32	974	0	0	0	1	17	985	0	0	0	1	19
RE	20	382					>3600	382	14	0	6	1	70	382	0	1	0	0	53
RE	20	365					>3600	364	16	0	9	1	63	365	11	1	4	0	57
RE	20	360					-	359	13	0	6	1	47	370	13	0	6	1	58
H&K [8]	25	1711					>3600	1711	0	0	0	1	122	1723	0	0	0	1	146
RE	25	414					>3600	411	20	5	15	1	198	414	15	6	10	0	197
RE	25	381					-	379	27	0	21	2	194	395	27	0	21	2	227
RE	25	408					-	407	22	5	22	6	248	408	16	1	7	0	127
Dantzig [4]	42	699					-	690	85	36	96	4	3170	699	23	23	27	0	1410
HAMILTON-CIRCUITS																			
G9	9	1	7	3	10	1	1	0	11	4	12	1	15	1	8	5	10	0	14
G12	12	0	10	0	10	1	2	0	17	10	26	3	52	0	0	1	0	0	2
G15	15	1	17	5	22	1	4	0	42	29	66	2	207	1	36	23	53	0	175
G16	16	0	14	0	14	1	3	0	12	3	13	1	53	0	0	1	0	0	6
G22	22	1	403	78	481	1	62	0					-	1					>3400
Tutte	46	1	1652	193	1846	2	612						(±7u)						-
Paard	64	0	62	0	62	1	118						(±4u)						-

Tabel 2. Rekenresultaten voor Euclidische problemen en Hamilton-circuits

PROBLEEM	n	C^*	LITTLE ET AL nc	nr	nb	nt	tijd	HELD AND KARP b	a = +10 nc	na	nb	nt	tijd	a = -10 lin	nc	na	nb	nt	tijd
SYMM.RANDOM																			
R	10	232	33	11	45	2	4	232	0	0	0	1	3	232	0	1	0	0	5
R	10	145	8	3	11	1	2	145	0	0	0	1	2	151	0	0	0	1	3
R	10	227	19	4	24	2	2	227	0	0	0	1	4	234	0	0	0	1	5
R	15	185	74	18	95	4	13	185	9	0	3	1	17	185	0	1	0	0	13
R	15	194	178	41	227	9	27	194	0	0	0	1	12	194	0	1	0	0	16
R	15	139	31	5	38	3	6	139	0	0	0	1	7	139	0	1	0	0	11
Croes [3]	20	246	99	18	117	1	23	246	0	0	0	1	11	252	0	0	0	1	27
R	20	201	191	41	236	5	42	200	17	0	9	1	60	214	17	0	9	1	73
R	20	251	262	52	321	7	59	251	16	0	9	1	64	273	16	0	9	1	76
R	20	196	227	64	293	3	77	194	14	3	11	1	76	225	14	3	11	1	89
R	25	220	1035	210	1253	9	288	219	20	0	11	2	124	221	15	1	6	1	132
R	25	266	1947	398	2347	3	576	258	47	59	93	8	1240	266	38	33	52	0	647
R	25	202	258	54	315	4	92	202	22	0	14	1	126	202	0	1	0	0	40
ASYMM.RANDOM																			
R	10	151	18	4	24	3	2												
R	10	177	8	0	8	1	1												
R	10	145	7	2	9	1	2												
R	20	145	295	49	349	6	52												
R	20	159	18	0	18	1	7												
R	20	177	301	51	357	6	52												
R	30	171	281	43	326	3	86												
R	30	145	487	69	558	3	151												
R	30	138	1211	173	1386	3	352												

Tabel 3. Rekenresultaten voor symmetrische en asymmetrische random problemen

Literatuur

1. Barachet, L.L., "Graphic Solution of the Traveling Salesman Problem," Opns.Res. 5, 841-845 (1957).

2. Bellmore, M. and G.L. Nemhauser, "The Traveling Salesman Problem: A Survey," Opns.Res. 16, 538-558 (1968).

3. Croes, G.A., "A Method for Solving Traveling Salesman Problems," Opns.Res. 6, 791-814 (1958).

4. Dantzig, G.B., D.R. Fulkerson and S.M. Johnson, "Solution of a Large Scale Traveling Salesman Problem," Opns.Res. 2, 393-410 (1954).

5. Dijkstra, E.W., "A Note on Two Problems in Connexion with Graphs," Numerische Mathematik 1, 269-271 (1959).

6. Eastman, W.L., "Linear Programming with Pattern Constraints," Ph.D. Dissertation, Harvard University (1958).

7. ---, "A Solution to the Traveling-Salesman Problem," Econometrica 27, 282 (1959).

8. Held, M. and R.M. Karp, "A Dynamic Programming Approach to Sequencing Problems," SIAM 10, 196-210 (1962).

9. ---, "The Traveling-Salesman Problem and Minimum Spanning Trees," Opns.Res. 18, 1138-1162 (1970).

10. ---, "The Traveling-Salesman Problem and Minimum Spanning Trees: Part II," to appear.

11. Kruseman Aretz, F.E.J., "Waarom langer reizen dan nodig is?," MC-Rapport ZW 1969-009 (1969).

12. Kruskal, J.B., "On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem," Proc.Amer.Math.Soc. 2, 48-50 (1956).

13. Lawler, E.L. and D.E. Wood, "Branch-and-Bound Methods: A Survey," Opns.Res. 14, 699-719 (1966).

14. Lin, S., "Computer Solution of the Traveling Salesman Problem," Bell System Tech.J. 44, 2245-2269 (1965).

15. Little, J.D.C., K.G. Murty, D.W. Sweeney and C. Karel, "An Algorithm for the Traveling Salesman Problem," Opns.Res. 11, 972-989 (1963).

16. Prim, R.C., "Shortest Connection Networks and Some Generalizations," Bell System Tech.J. 36, 1389-1401 (1957).

17. Shapiro, D., "Algorithms for the Solution of the Optimal Cost and Bottleneck Traveling Salesman Problems," Sc.D. Thesis, Washington University, St. Louis (1966).

18. Tijdeman, R., "Het handelsreizigersprobleem, een literatuur-onderzoek," MC-Rapport S 385 (1968).