FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Gevorderde algoritmen

Bert De Saffel

Master in de Industriële Wetenschappen: Informatica Academiejaar 2018–2019

Gecompileerd op 30 november 2019

UNIVERSITEIT
GENT

Inhoudsopgave

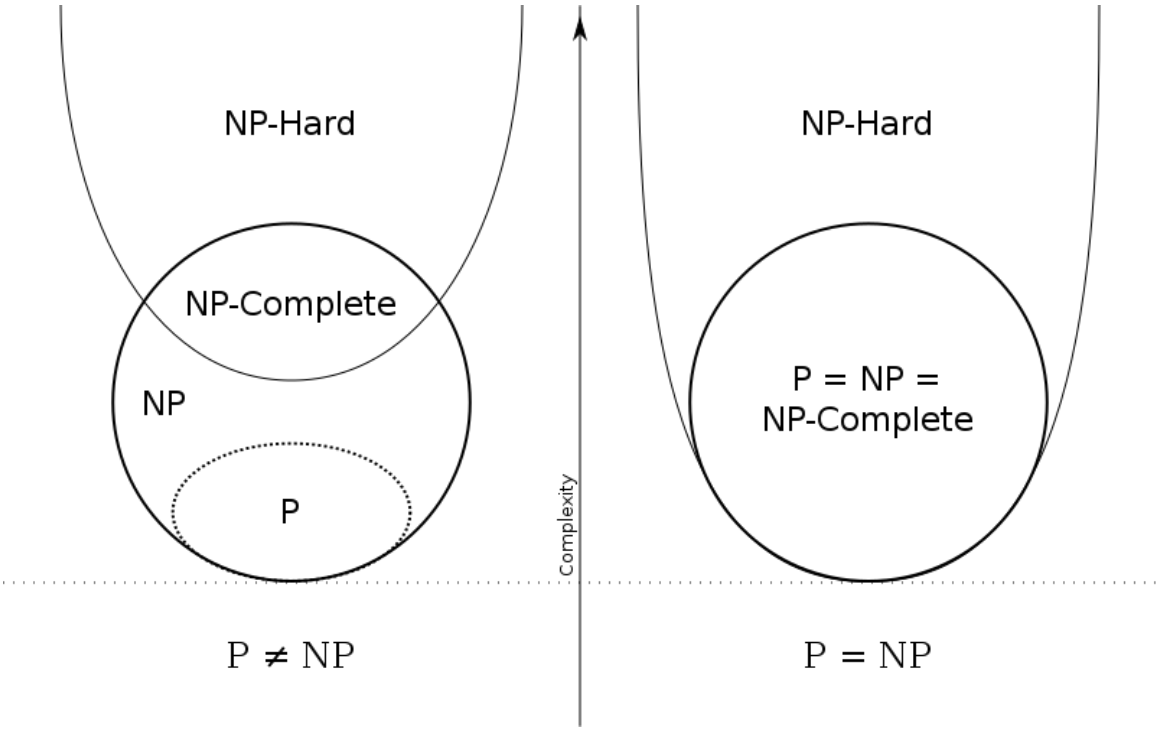
Deel I

Hardnekkige problemen

Hoofdstuk 1

NP

1.1 Complexiteit: P en NP



Figuur 1.1: De linkse deelfiguur toont de verschillende complexiteitsklassen indien $P \neq NP$. De rechtse deelfiguur toont hetzelfde indien $P = NP$.

- Alle besproken algoritmen hebben een efficiënte oplossing.
- Hun uitvoeringstijd wordt begrensd door een **veelterm** zoals $O(n^2)$ of $O(n^2m)$.
- Sommige problemen hebben hebben geen efficiënte oplossing.
- Problemen worden onderverdeeld in **complexiteitsklassen**.
    - Beperking tot **beslissingsproblemen**, waarbij de uitvoer *ja* of *nee* is.

- Niet echt een beperking omdat elk probleem als een beslissingsproblemen kan geformuleerd worden.

1.1.1 Complexiteitsklassen

- De klasse **P** (**P**olynomiaal) bevat alle problemen waarvan de uitvoeringstijd begrensd wordt door een veelterm.
  - Op een realistisch computermodel.
    - Heeft een polynomiale bovengrens voor het werk dat in één tijdseenheid kan verricht worden.
  - Met een redelijke voorstelling van de invoergegevens (geen overbodige informatie, compact, ...).
  - Al de problemen in **P** worden als efficiënt oplosbaar beschouwd.
  - ! Waarom een veelterm? $O(n^{100})$ kan nauwelijks efficiënt genoemd worden.
    1. Meestal is de graad van de veelterm beperkt tot twee of drie.
    2. Veeltermen vormen de kleinste klasse functies die kunnen gecombineerd worden, en opnieuw een veelterm opleveren.
       - Men noemt dit een **gesloten klasse**.
       - Efficiënte algoritmen voor eenvoudigere problemen kunnen dus gecombineerd worden tot een efficiënt algoritme voor een complex probleem.
    3. De efficiëntiemaat blijft onafhankelijk van het computermodel.
- De klasse **NP** (**N**iet-deterministisch **P**olynomiaal) bevat alle problemen die door een niet-deterministische computer in polynomiale tijd kunnen opgelost worden en waarvan de oplossing kan gecontroleerd worden in polynomiale tijd.
  - Een niet-deterministische computer bevat hypothetisch een oneindig aantal processoren, waarvan er op tijdstap t er k kunnen aangesproken van worden. De processoren werken niet samen, maar kunnen wel hun deel van het probleem oplossen.
  - Elk probleem uit **P** behoort tot **NP**.
  - Niet geweten of er problemen zitten in **NP** die niet tot **P** behoren → **P** vs **NP** probleem (Figuur 1.1).
  - Wel geweten dat er problemen zijn die niet in **NP** zitten, en dus ook niet in **P**.
- De klasse **NP-hard** bevat alle problemen die minstens even zwaar zijn als elk **NP**-probleem.
  - Een probleem X dat gereduceerd kan worden naar een probleem Y betekent dat Y minstens even zwaar is als X.
- De klasse **NP-compleet** bevat alle problemen die **NP-hard** zijn, maar toch nog in **NP** zitten.
  - Als er één **NP-compleet** probleem bestaat die efficiënt oplosbaar zou zijn (en dus in **P** behoort), dan zouden alle problemen uit **NP** ook efficiënt oplosbaar zijn, zodat **P** = **NP**.
  - NP-complete problemen kunnen op verschillende manieren aangepakt worden:
    - Backtracking en snoeien.
    - Speciale gevallen oplossen met efficiënte algoritmen.
    - Het kan zijn dat de gemiddelde uitvoeringstijd toch goed is.
    - Gebruik een benaderend algoritme.
    - Maak gebruik van heuristieken.

1.2 NP-complete problemen

- Overzicht van belangrijke NP-complete (optimalisatie)problemen.
- Om na te gaan of een probleem NP-compleet is, moet het herleid kunnen worden naar een basisvorm.

1.2.1 Het basisprobleem: SAT (en 3SAT)

- Gegeven:
  - Een verzameling logische variabelen $\mathcal{X} = \{x_1, \ldots, x_{|\mathcal{X}|}\}$.
  - Een verzameling logische uitspraken $\mathcal{F} = \{f_1, \ldots, f_{|\mathcal{F}|}\}$.
  - Elke uitspraak bestaat uit automaire uitspraken (atomen) samengevoegd met OF-operaties:

$$f_1 = x_2 \vee \overline{x_5} \vee x_7 \vee x_8$$

- Gevraagd:
  - Hoe moeten de waarden toegekend worden aan de variabelen uit $\mathcal{X}$ zodat elke uitspraak in $\mathcal{F}$ waar is?
- Elk NP-compleet probleem is reduceerbaar tot SAT.
- Een uitspraak met meer dan drie atomen kan herleidt worden naar een reeks uitspraken met elk drie atomen:

$$\begin{aligned} f_1 &= x_2 \vee \overline{x_5} \vee x_n \\ f_1' &= \overline{x_n} \vee x_7 \vee x_8 \end{aligned}$$

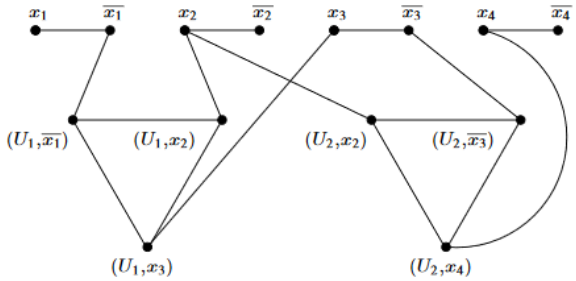
1.2.2 Vertex Cover

- Gegegeven:
  - Een ongerichte graaf.
- Gevraagd:
  - Hoe kan de kleinste groep knopen bepaald worden die minsten één eindknoop van elke verbinding bevat.
- Voorbeeld:
  - 3SAT kan gereduceerd worden tot vertex cover.
    - Voor elke logische variabele x_i worden er twee knopen gemaakt: voor x_i en $\overline{x}_i$. Deze knopen zijn verbonden.
    - Voor elke uitspraak worden er drie knopen gemaakt, één voor elk atoom. Deze drie knopen worden verbonden. Elk atoom wordt ook verbonden met de knopen voor de individuele logische variabelen.
  - Voor elke logische variabele moet zeker één van de twee knopen opgenomen worden.
  - Voor elke uitspraak moeten zeker twee van de drie knopen opgenomen worden.
  - Minstens $|\mathcal{X}| + 2|\mathcal{F}|$ knopen.

- Stel volgende uitspraken

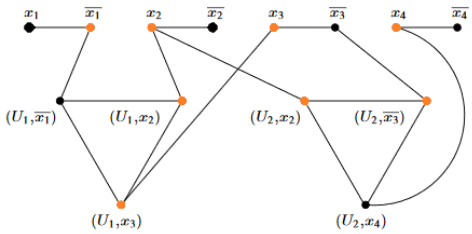
$$U_1 = \overline{x}_1 \vee x_2 \vee x_3$$
$$U_2 = x_2 \vee \overline{x}_3 \vee x_4$$

Bijhorende graaf:



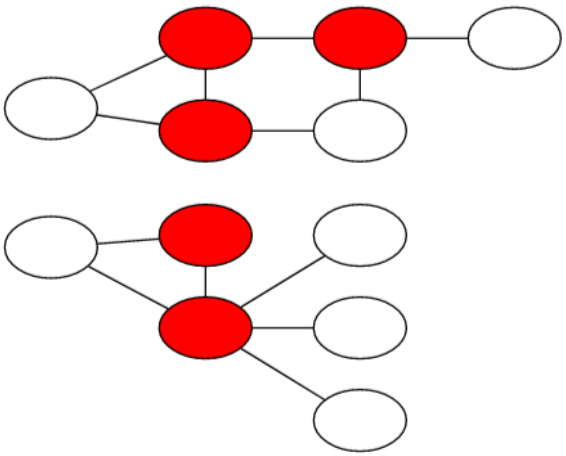
Figuur 1.2: Een graaf voor het 3SAT-probleem.

Voorbeeld van een minimale vertex cover:



Figuur 1.3: Een minimale vertex cover van de graaf op figuur 1.2

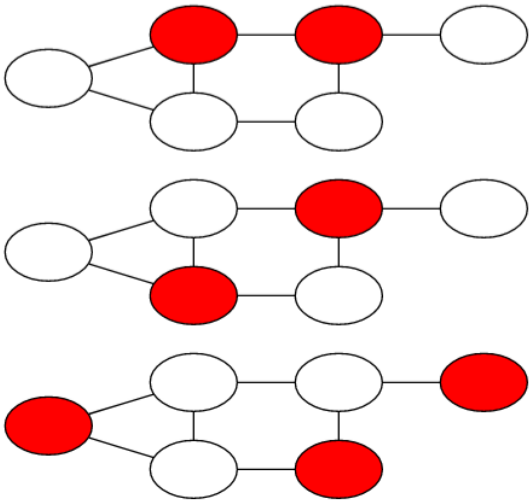
- Andere voorbeelden van minimale vertex covers:



Figuur 1.4: De minimale vertex cover van twee verschillende grafen.

1.2.3 Dominating set

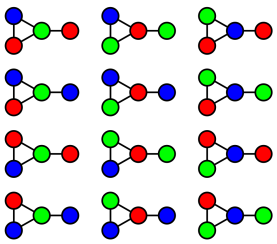
- Gegeven:
  - Een ongerichte graaf.
- Gevraagd:
  - Een kleinste groep knopen zodat elke andere knoop met minstens een van de knopen uit de groep verbonden is.
- Voorbeelden:



Figuur 1.5: Voorbeelden van dominating sets.

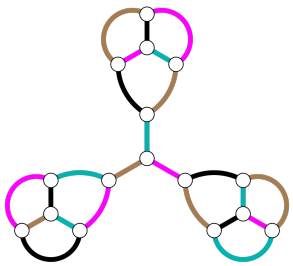
1.2.4 Graph Coloring

- **Vertex coloring**:
  - Gegeven:
    - Een ongerichte graaf.
  - Gevraagd:
    - Kleur de knopen met een minimum aantal kleuren, zodat de eindknopen van elke verbinding een verschillende kleur hebben.
  - Voorbeeld:



Figuur 1.6: De knopen van een graaf kunnen op meerdere manieren gekleurd worden.

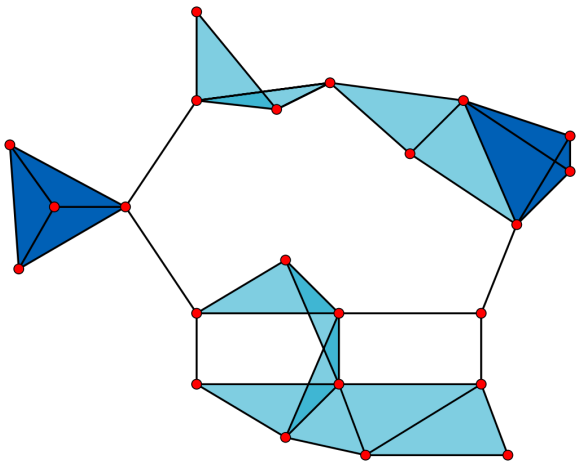
- **Edge coloring**:
  - Gegeven:
    - Een ongerichte graaf.
  - Gevraagd:
    - Kleur de verbindingen met een minimum aantal kleuren, zodat verbindingen met dezelfde eindknoop een verschillende kleur hebben.
  - Voorbeeld:



Figuur 1.7: Elke knoop heeft graad 3 dus zijn er hoogstens 4 kleuren nodig om de verbindingen te kleuren.

1.2.5 Clique

- Gegeven:
  - Een ongerichte graaf.
- Gevraagd:
  - De grootste groep knopen die allemaal met elkaar verbonden zijn.
- Voorbeeld:

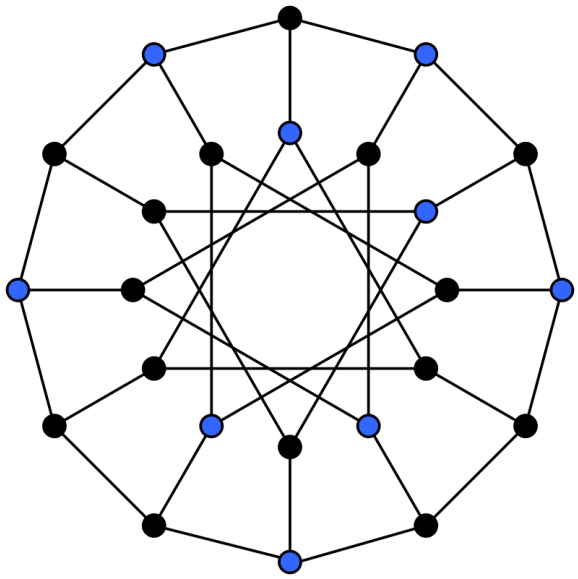


Figuur 1.8: Deze graaf bevat 23 1-knoop cliques (de knopen), 49 2-knoop cliques (door de verbindingen), 19 3-knoop cliques (lichtblauwe driehoeken) en 2 4-knoop cliques (donkerblauwe oppervlakken). In deze graaf zijn de twee 4-knoop de grootste groep van knopen.

- Het SAT probleem kan herleidt worden tot clique.
  - Voor elk voorkomen van een atoom in een uitspraak wordt twee knopen (F_i, x_j) en $(F_i, \overline{x_j})$ aangemaakt.
  - Er is een verbinding tussen knopen die horen bij verschillende uitspraken als de atomen elkaar niet uitsluiten:
    - (F_i, x_k) met (F_j, x_l) voor alle k en l.
    - $(F_i, \overline{x_k})$ met $(F_i, \overline{x_l})$ voor alle k en l.
    - (F_i, x_k) met $(F_i, \overline{x_l})$ als $k \neq l$.
- Een clique kan hoogstens $|\mathcal{F}|$ elementen bevatten.

1.2.6 Independent set

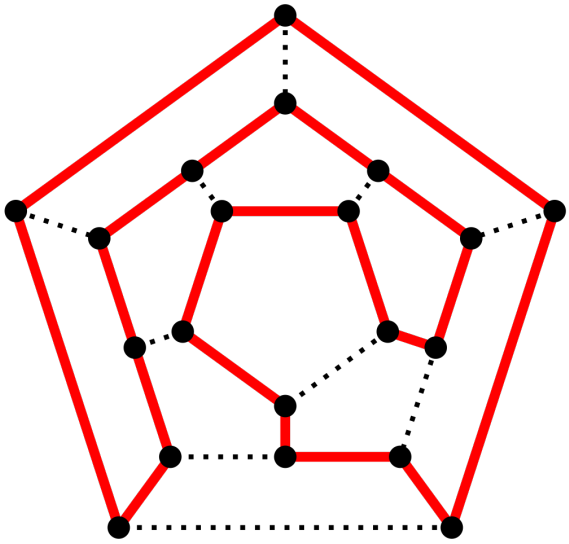
- Gegeven:
  - Een ongerichte graaf.
- Gevraagd:
  - De grootste groep knopen zonder gemeenschappelijke verbindingen.
- Voorbeeld:



Figuur 1.9: De blauwe knopen vormen de independent set.

1.2.7 Hamilton path

- Gegeven:
  - Een al dan niet gerichte graaf.
- Gevraagd:
  - Bestaat er een circuit dat elke knoop eenmaal bevat?
- Voorbeeld:



Figuur 1.10: Een tweedimensionale weergave van een Hamiltonpad door een dodecaëder. De zwarte stippelijnen zijn ook verbindingen.

1.2.8 Minimum cover

- Gegeven:
  - Een verzameling $\mathcal{S}$.
  - Een collectie deelverzamelingen $\mathcal{C}$ van $\mathcal{S}$.
- Gevraagd:
  - De kleinste deelverzameling $\mathcal{C}'$ van $\mathcal{C}$ zodat elk element van $\mathcal{S}$ tot minstens één van de deelverzamelingen uit $\mathcal{C}'$ behoort.
- Voorbeeld:
  - Dit kan efficiënt opgelost worden als de deelverzamelingen in $\mathcal{C}$ niet meer dan twee elementen bevatten.
  - Als elke $C \in \mathcal{C}$ exact twee elementen heeft is het equivalent met edge cover.
  - Als er $C \in \mathcal{C}$ zijn met slechts één element kunnen deze geschrapt worden en dan edge cover oplossen.
  - SAT kan ook herleidt worden tot minimum cover.
    - Neem $\mathcal{S} = \mathcal{F} \cup \mathcal{X}$
    - Voor elke logische variabele x_i zijn er twee elementen van C

$$C_{x_i} = \{x_i\} \cup \{F \in \mathcal{F} : x_i \in F\}$$
$$C_{\overline{x_i}} = \{x_i\} \cup \{F \in \mathcal{F} : \overline{x_i} \in F\}$$

1.2.9 Subset sum

- Gegeven:
  - Een verzameling elementen met elk een positieve gehele grootte.
  - Een positief geheel getal k.

- Gevraagd:
  - Een deelverzameling van die elementen, zodat de som van hun grootten gelijk is aan k.

1.2.10 Partition

- Gegeven:
  - Een zak van positieve gehele getallen.
- Gevraagd:
  - Kan die opgesplitst worden in twee deelverzamelingen, zodat de som van de grootten van de ene gelijk is aan de som van de grootten van de andere.

1.2.11 Travelling salesman problem

- Gegeven:
  - Een aantal steden, en de afstand tussen elk paar steden.
- Gevraagd:
  - Het kortste circuit dat elke stad eenmaal bevat.

1.2.12 Longest path

- Gegeven:
  - Een gerichte of ongerichte gewogen graaf.
  - Twee knopen s en t.
- Gevraagd:
  - De langste weg zonder lussen van s naar t.

1.2.13 Bin packing

- Gegeven:
  - Een verzameling van n objecten met afmetingen $s_1, \cdots, s_n$.
  - Een verzameling van m bakken met capaciteiten $c_1, \cdots, c_m$.
- Gevraagd:
  - Sla alle objecten op in zo weinig mogelijk bakken

1.2.14 Knapsack

- Gegeven:
  - Een verzameling van n objecten met elk een afmeting.
  - Een knapzak met een zekere capaciteit.
- Gevraagd:
  - Steek objecten in de zak zonder zijn capaciteit te overschrijden, zodat hun totale waarde zo groot mogelijk is.

Hoofdstuk 2

Metaheuristieken

- **Heuristieken** zijn vuistregels bij het zoeken naar een oplossing van een probleem.
- Garanderen niet dat er een oplossing gevonden wordt, maar versnelt wel de zoektocht ernaar.

2.1 Combinatorische optimisatie

- Abstracte representatie van de problemen nodig.
  - Het zijn **optimisatie**-problemen.
  - Voor een verzameling $\mathcal{S}$ moet hieruit de beste gekozen worden.
  - De verzameling $\mathcal{S}$ is een eindige verzameling van strings over een eindig alfabet.
  - Het beste individu wordt bepaald door een evaluatiefunctie f.
  - De beste waarde komt overeen met de kleinste waarde voor f.
- Voorbeeld bij het Travelling Salesman probleem:
  - Het alfabet is hier de verzameling verbindingen.
  - De verzameling $\mathcal{S}$ bestaat uit een reeks verbindingen zo dat elke verbinding vertrekt uit het eindpunt van de vorige en zo dat alle steden bezocht worden.
  - Het beste individu is die met de kleinste lengte.
- Vaak zijn alle strings in $\mathcal{S}$ even lang.
  - Elementen van $\mathcal{S}$ kunnen dan beschreven worden door variabelen.
  - Elke letter $s \in \mathcal{S}$ geeft de waarde aan van een variabele.
  - Voorbeeld bij het lessenroosterprobleem:
    - Er zijn een aantal lessen, een aantal docenten, een aantal klaslokalen, een aantal mogelijke tijdsslots en een aantal groepen studenten.
    - Er moet een lessenrooster opgesteld worden zodanig dat er minimale negatieve elementen zijn (springuren, onevenwichtige verdelingen van de studenten, ...).
    - Elke string van $\mathcal{S}$ is even lang en beschrijft e en lessenrooster zonder conflicten door aan elke les een lokaal en een tijdsslot toe te kennen.
    - $\mathcal{S}$ heeft twee letters: de eerste duidt de tijdsslot aan en de tweede het lokaal.
- In principe zijn zulke problemen oplosbaar met backtracking.

- Maar voor problemen die te groot zijn voor backtracking, zijn metaheuristieken toch nodig.
- Er wordt daarom eerst een steekproef genomen uit de totale verzameling $\mathcal{S}$ en voor elk individu wordt de $f-$waarde berekend.
- Het individu met de beste $f-$waarde wordt altijd bijgehouden.
- Het zoeken stop na een bepaald criterium, zoals bijvoorbeeld een tijdslimiet of als het gevonden individu reeds goed genoeg is.
- De verschillende metaheuristieken verschillen in de manier waarop individuen worden uitgekozen om te vergelijken.
- Elke metaheuristiek leent zich ertoe om lokale optimalisaties door te voeren.

2.2 Vooronderstellingen

- Bij de keuze van methodieken moet er eerst nagegaan worden of het probleem aan een aantal voorwaarden voldoet. Het ontbreken van een bepaalde voorwaarde kan bijvoorbeeld een metaheuristiek uitsluiten. Er zijn **vier** gebieden waarin onderscheidt tussen de verschillende optimisatietechnieken gemaakt kan worden.

  1. **Het moet zinvol zijn om op zoek te gaan naar betere individuen in de buurt van een gegeven individu.** Het kan zijn dat we een individu van de grond af opbouwen (zeker bij het eerste individu), maar er moeten ook pogingen ondernomen worden om een reeds bestaand individu te verbeteren.
  2. **Soms is het niet zinvol om bij het opbouwen van een nieuw individu van $\mathcal{S}$ uit te gaan van reeds bekende individuen.** Een nieuw individu van de grond opbouwen kan op twee manieren:
     (a) Het individu wordt opgebouwd uit componenten. Deze componenten komen overeen met letters van de strings in $\mathcal{S}$. Dit gebeurt at random, eventueel met een bepaalde heuristiek die zeer slechte individuen uitsluit.
     (b) Het individu wordt rechtstreeks aangemaakt.

     In het travelling salesman probleem kan een individu beschouwd worden als een pad in een graaf. Via de eerste methode is het pad leeg en worden verbindingen toegevoegd. In het tweede geval is het individu een permutatie van de verzameling verbindingen.
  3. **Een heuristiek moet aangeven dat een bepaalde keuze beter is dan de andere.** Bij het opbouwen van de componenten van een individu zijn sommige keuzes niet meer zinvol. Die mogen dan zeker niet meer gekozen worden. Er zijn twee mogelijkheden om dit te realiseren.
     (a) **Shortlisting.** Eerst worden de beste kandidaten geselecteerd met behulp van een bepaalde cutoff. Daarna wordt hieruit gekozen met gelijke waarschijnlijkheid.
     (b) **Gewogen keuze.** Elke mogelijkheid krijgt een een gewicht w_i. De kans $p(i)$ dat i gekozen wordt is dan
     $$p(i) = \frac{w_i}{\sum_j w_j}$$
  4. **Er zijn drie fundamentele methoden om een nieuw individu $\mathcal{S}$ te kiezen uitgaande van andere individuen.**
     (a) Kleine wijzigingen aanbrengen in een reeds bekeken individu s van $\mathcal{S}$.
     (b) Kruisen van twee bekeken individuen.
     (c) Het vermengen van een grote hoeveelheid al bekeken individuen.

2.3 Lokaal versus globaal zoeken

- Bij het opbouwen van nieuwe individuen van $\mathcal{S}$ moet het zinvol zijn om een individu van $\mathcal{S}$ te verbeteren door kleine aanpassingen.
- Een kleine aanpassing hangt af van het probleem.
  - Voor elke $s \in \mathcal{S}$ is er een **omgeving** $\mathcal{N}(s)$: de omgeving van s die bestaat uit s zelf en alle strings uit $\mathcal{S}$ die door één kleine wijziging uit s bekomen kunnen worden.
- Een goed individu kan zeker in een slecht individu omgevormd worden, maar vaker blijft het een goed individu.
- Lokale verbeteringen kunnen iteratief toegepast worden:
  - Beginnend van een individu $s_0 \in \mathcal{S}$
  - Genereer een rij van individuen $s_1, \cdots, s_n \in \mathcal{S}$.
  - Hierbij geldt dat voor alle $i = 0, \cdots, n-1, s - i + 1 \in \mathcal{N}(s_i)$ en $f(s_{i+1}) < f(s_i)$.
  - Dit eindigt met een lokaal minimum, een individu s_n zodat $\forall s \in \mathcal{N}(s_n) : f(s) \geq f(s_n)$.
- Het vinden van zo een rij heet **lokale optimisatie**.
- Zo een rij vinden kan op meerdere manieren:
  1. **Steepest hill climbing.** (of steepest slope descending)
     Soms is het gemakkelijk om een lokaal minimum te vinden omdat $\mathcal{N}(s)$ weinig individuen bevat.
  2. **Randomisatie.**
     Veranderingen worden random gekozen.
- Er kunnen veel lokale minima zijn in $\mathcal{S}$, elk met een verschillende $f-$waarde.
- Er moet een manier zijn om te ontsnappen aan een lokaal minimum $\rightarrow$ **exploratie**.
  1. Exploratie door helemaal opnieuw te beginnen.
  2. Exploratie door grotere wijzigingen aan te brengen aan gekende individuen.

2.4 Methodes zonder recombinatie

- Random zoeken kan efficiënter gemaakt worden.
  1. Lokaal zoeken verbetert vaak random aangemaakt individuen.
  2. Heuristieken gebruiken om het aanmaken van de individuen te sturen.

2.4.1 Simulated Annealing

- Tijdens lokale optimisatie maakt het gebruik van een kans op een individu s', ook al is s' slechter dan s.
- De kans hangt af van:
  - De evaluatiewaarden $f(s)$ en $f(s')$.
  - Een 'temperatuur' T.

- De kans wordt groter indien T kleiner is of indien $f(s') - f(s)$ kleiner is (Boltzmanndistributie).

$$\rho(T, f(s'), f(s)) = \exp\left(\frac{f(s') - f(s)}{T}\right)$$

- T wordt vrij hoog gekozen, zodat exploratie bevorderd wordt en wordt iteratief gedaald tot bijna nul.
- Dit algoritme vindt zeer waarschijnlijk een globaal optimum als er een $\Gamma > 0$ bestaat waarvoor

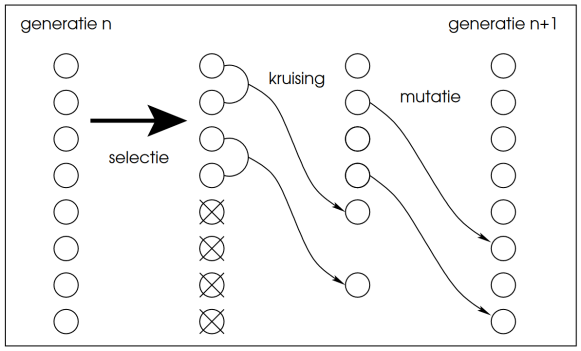
$$\sum_{k=1}^{\infty} \exp\left(\frac{-\Gamma}{T_k}\right) = \infty$$

2.4.2 Tabu Search

- Houdt een taboelijst bij: een lijst van al gecontroleerde individuen die niet opnieuw mogen bezocht worden.
- Alle gecontroleerde individuen bijhouden is niet efficiënt, daarom maar een beperkt aantal.
- De lijst bevorderd exploratie: bij een grote lijst zal een nieuwe individu veel verder in de omgeving $\mathcal{N}(s)$ liggen.
- De taboelijst kan ook eigenschappen bijhouden en geen individuen.

2.5 Genetische algoritmen

- Meeste genetische algoritmen maken gebruik van **kruising**.
- Kruising is het combineren van twee individuen.
- Enkel zinvol als een kruising een beter individu *kan* opleveren.
- Kruising combineerd meestal de componenten van beide individuen die goed kunnen zijn.
- Handig als één van de individuen een slechte $f-$waarde heeft.
- Twee manieren om te kruisen:
  1. **Gerichte kruising.**
     Enkel haalbaar als er een algemeen idee is van de structuur van individuen.
  2. **Blinde kruising.**
     Gerandomiseerde kruising van componenten.
- Er is een **populatie nodig**.
- Het doel is dan om de algemen populatie te verbeteren.
- Men spreekt van **generaties**. De overstap van een generatie naar een andere gebeurt als volgt (figuur 2.1):
  1. De populatie wordt uitgedund zodat enkel de beste individuen overblijven.
  2. De populatie wordt terug aangevuld op basis van de overblijvende individuen door kruising en mutatie.
- Twee verfijningen:



Figuur 2.1: Overgang naar een volgende generatie.

- Twee componenten kunnen elkaar positief beïnvloeden. Het is dan interessant om deze componenten bij elkaar te houden. Dit kan ingebouwd worden bij gerichte kruising. Er kan ook een maat voor gelijkaardigheid ingevoerd worden.
- Kruising moet niet enkel zorgen voor betere individuen, maar ook voor diversificatie en zo voor exploratie. Zo voert men **niching** in, dat de fitheid van individuen verminderd naarmate er meer gelijkaardige individuen in de populatie zitten.

2.6 Vermenging

- Niet twee individuen kruisen, maar de globale eigenschappen van de hele populatie wordt in beschouwing genomen.
  1. **Componentniveau.** Hierbij gaat men ervan uit dat het succes van een individu afhangt van de individuele componenten die ze bevat.
  2. **Combinatieniveau.** Hierbij gaat men ervan uit dat het succes van een individu afhangt van de combinatie van componenten die ze bevat.

2.6.1 Recombinatie op componentniveau

- Er wordt een $f-$waarde toegekend niet enkel aan de individuen, maar ook aan de componenten zelf.
- Het construeren van een nieuwe populatie gebeurt pseudorandom waarbij gewicht w_i van component i dient als gewicht voor de gewogen keuze.
- De gewogen random keuze kiest uit een verzameling $\mathcal{V}$ van componenten een component i met waarschijnlijkheid

$$p(i) = \frac{w_i}{\sum_{j \in V} w_j}$$

- Twee belangrijke metaheuristieken die gebruik maken van dit soort recombinatie. Ze verschillen in de manier waarop gewichten aan componenten toegekend worden.
  - **Gene Pool Recombiniation (GPR).**
    Elk component krijgt een gewicht evenredig met het aantal individuen in de overblijvende populatie die het component bevatten. Het gewicht van een component is dan enkel afhankelijk van de vorige generatie.

- **Ant Colony Optimisation (ACO).**
  Elke s krijgt eerst een kwaliteitswaarde $F(s)$, die groter is naarmate $f(s)$ kleiner is. Een nieuw gewicht voor een component i is dan

$$w_i = (1-\rho)w_i + \sum_{s \text{ bevat } i} F(s)$$

  met ρ de vergeetfactor. Het gewicht van meerdere generaties zal invloed hebben op het huidige gewicht.

2.6.2 Recombinatie op combinatieniveau

- Er zijn twee voorwaarden om dergerlijke methoden te gebruiken:
  1. Opeenvolgende letters in het individu moeten een sterk onderling verband vertonen.
  2. Een deelstring van een goed individu gebruiken in een ander individu verhoogt de kans dat het andere individu ook goed is.
- De componenten van de individuen worden de knopen van een ongerichte gewogen graaf.
- Als twee componenten samen kunnen voorkomen is er een verbinding tussen de overeenkomstige knopen in de graaf.
- Er zijn nu een aantal **agents** die proberen een individu uit $\mathcal{S}$ te construeren. Dit wordt voorgesteld door een pad in de graaf.
  1. Op een bepaald punt bepaalt de agent of het pad een individu definieert. Zo ja dan stopt het.
  2. Anders worden de componenten bepaalt die nog in aanmerking komen.
  3. Als er geen mogelijkheden zijn wordt er gestopt, anders wordt er een gewogen keuze gemaakt tussen de verbindinge.
- Nadat elke agent geprobeerd heeft om een individu te construeren, worden de gewichten aangepast.
- Elke agent die een individu s heeft gevonden, past het gewicht aan van elke verbinding dat het gebruikt heeft met $F(s)$

$$\tau_{ij} = (1-\rho)\tau_{ij} + \sum_{s \in A_{ij}} F(s)$$

  Hierbij is A_{ij} de verzameling individuen die verbinding v_{ij} gebruikt hebben.
- Meestal zijn alle τ_{ij} eerst gelijk, maar andere beginwaarden zijn ook mogelijk.