

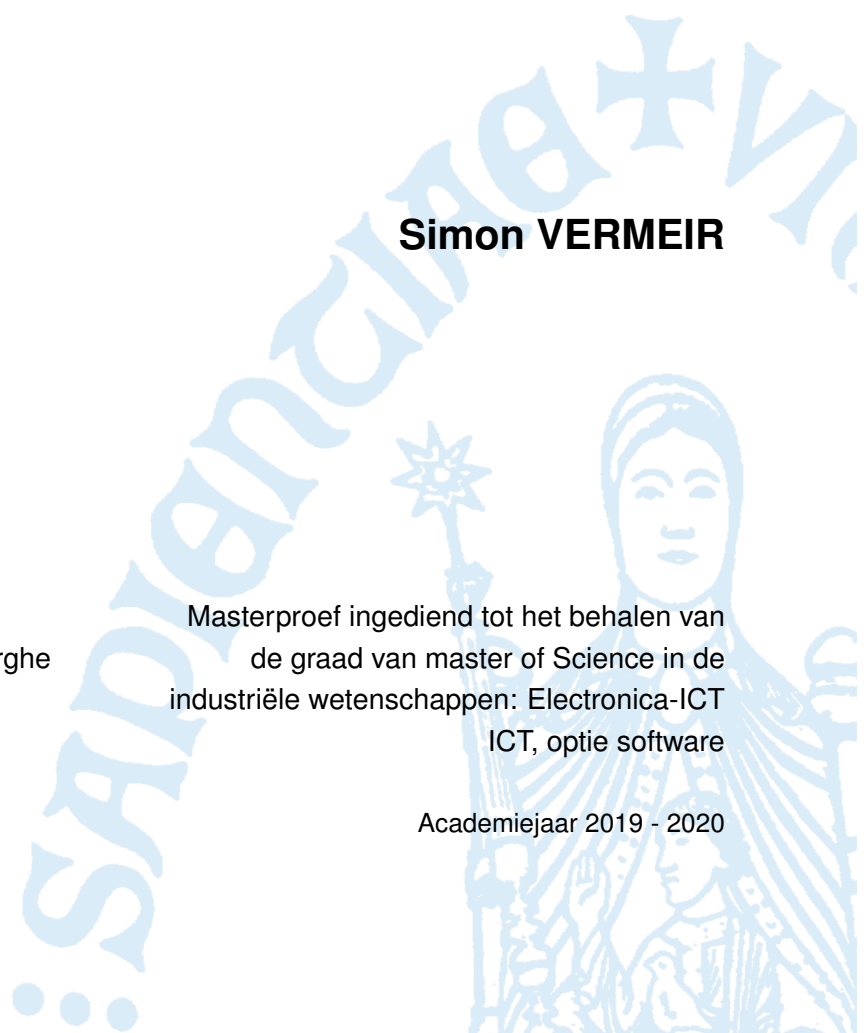
The Job Sequencing and Tool Switching Problem

Simon VERMEIR

Promotor(en): Prof. Dr. Ir.
Greet Vanden Berghe

Masterproef ingediend tot het behalen van
de graad van master of Science in de
industriële wetenschappen: Electronica-ICT
ICT, optie software

Academiejaar 2019 - 2020



©Copyright KU Leuven

Zonder voorafgaande schriftelijke toestemming van zowel de promotor(en) als de auteur(s) is overnemen, kopiëren, gebruiken of realiseren van deze uitgave of gedeelten ervan verboden. Voor aanvragen i.v.m. het overnemen en/of gebruik en/of realisatie van gedeelten uit deze publicatie, kan u zich richten tot KU Leuven Technologicampus Gent, Gebroeders De Smetstraat 1, B-9000 Gent, +32 92 65 86 10 of via e-mail iiw.gent@kuleuven.be.

Voorafgaande schriftelijke toestemming van de promotor(en) is eveneens vereist voor het aanwenden van de in deze masterproef beschreven (originele) methoden, producten, schakelingen en programma's voor industrieel of commercieel nut en voor de inzending van deze publicatie ter deelname aan wetenschappelijke prijzen of wedstrijden.

Voorwoord

Ik zou allereerst graag mijn ouders en zus bedanken om mij te steunen door al die jaren. Ik weet dat het geen gemakkelijke opdracht was om deze richting te voltooien. Zonder hun voorwaardelijke steun was dit nooit gelukt.

Er moet ook dank uitgebracht worden aan alle mensen die mij hielpen om mijn trein op de rails te houden. Trajectbegeleiders, secretariaat, ombuds, opleidingshoofden... Mijn trein had nog al een moeilijk traject en zonder mensen zoals: Annelies Debever, Martin Meganck, Annemie Vostermans, Tamara De Jaeger, Carine Naessens, Ann Van Vossel, Luc Peelman en vele andere was het niet mogelijk geweest.

Een dankwoord is ook toepasselijk voor al de goeie vrienden die ik op deze weg heb leren kennen. Het doet plezier om gelijkgestemde mensen te ontmoeten.

Alle talloze professoren en assistenten verdienen ook een welgemeende merci! De discipline van ingenieur die ze samen aanleerde is nobel, het maakt ons meer bewust van hoe de wereld werkt rond ons en hoe we hierin een rol kunnen spelen.

Ik wens ook Ine Van Cauwelaert te bedanken voor mijn thesis na te lezen en Tony Wauters voor de steeds snelle technische hulp met de rekenserver.

Ten slotte wil ik mijn dankbetuiging uitbrengen aan mijn promotor Greet Vanden Berghe. Naast het aanleren van Operations Research, heeft zij mij geleerd om structuur te brengen in wat ik doe. Om steeds de waarom vraag te durven stellen en te streven naar simpliciteit en helderheid tijdens onderzoek. Haar steeds kritische houding heeft mij bewuster gemaakt. De liefde die ze voor dit vak uitstraalt gaf de kracht om telkens die stap verder te gaan.

Bedankt aan iedereen.

-Simon

Samenvatting

Tijd gaat telkens opnieuw verloren bij het verwisselen van gereedschap. Zo is het bijvoorbeeld een intuïtief gegeven om eerst een muur te schuren en deze pas daarna te verven. Het is niet zinvol om stukje per stukje te verven en te schuren. Hetzelfde fenomeen duikt op binnen de industrie. In moderne productieomgevingen wordt er vaak gebruik gemaakt van flexibele machines. Die kunnen verschillende producten produceren door een reeks bewerkingen uit te voeren (boren, zagen, snijden. . .). Om dat mogelijk te maken beschikken ze over een magazijn waarin verschillende stukken gereedschap geplaatst kunnen worden. Er is voldoende plaats voorhanden om het gereedschap te bewaren voor een type product. Het bewaren van gereedschap voor alle producten is vaak onmogelijk. Daarom is het wisselen van het gereedschap nodig, wat helaas een tijdrovend proces is. Toch kunnen we dit drastisch beperken. Door de producten op een slimme manier te ordenen, en het gereedschap te bewaren dat we later nodig hebben. In de literatuur noemt dit probleem "The Job Sequencing and tool Switching Problem" of "SSP". Hierbij interpreteren we de producten als "jobs" die uitgevoerd moeten worden en het gereedschap als "tools" die nodig zijn. Producten ordenen we door jobs een positie te geven binnen een sequentie. Gereedschap bijhouden gebeurt door op elke positie de gewenste tools te bepalen. Het objectief is, het uitwisselen limiteren of het aantal "switches" minimaliseren. Waarbij de capaciteit van het magazijn een beperking vormt. Een switch is te definiëren als het invoegen of verwisselen van een nieuw tool.

Formeel werd het eerste werk geïntroduceerd door Tang en Denaro. Ze ontbonden het probleem in 2 subproblemen:

- The sequencing problem (SP): De volgorde bepalen waarin de tools uitgevoerd moeten worden. Het sequencing problem is $\mathcal{NP}$ hard.
- The tooling problem (TP): Gegeven de job sequentie bepalen welke tools gewenst zijn op elke positie. Het is oplosbaar in polynomiale tijd.

Bovendien bewezen ze dat gegeven een vaste sequentie (SP) de optimale tool bezetting te vinden is (TP). Dankzij de "Keep Tool Needest Soonest" strategie of "KTNS policy ". Hierbij minimaliseert men het aantal tool switches door de snelst nodige tools te bewaren in het magazijn.

Reeds vele pogingen werden ondernomen om het probleem op te lossen. De moeilijkheid van het SP beperkt exacte methoden al vanaf een bescheiden aantal jobs en tools. Het grote deel van de werken focust zich daarom op heuristische methoden.

Verschillende varianten van het probleem bestaan, in dit werk ligt de focus op het uniforme probleem: tools passen overal in het magazijn, tools hebben een uniforme grootte, jobs hebben dezelfde duur en één machine.

Het werk is gebaseerd op 3 grote peilers. Het ontwikkelen van een ruime, gerichte Local Search, een snelle decoder en een objectief functie die plateaus vermijdt. Een belangrijke fundering voor deze peilers vormen de data structuren. Deze zijn nauwlettend ontwikkeld om nutteloze berekeningen te beperken. Relaties tussen oplossingen en preprocessing staan hier dus centraal.

Er wordt een indirecte representatie van het probleem gebruikt om het op te lossen. Hierbij wordt de zoektocht uitgevoerd op de job sequentie. Telkens wordt een permutatie van deze sequentie bepaald. Dit met behulp van "Ruin & Recreate". Als ruin stap wordt een tool stochastisch geselecteerd. Vervolgens worden alle jobs die deze bevatten verwijderd. De logica hier: deze jobs zijn gerelateerd. Als recreate stap worden ze volgens een greedy insert geplaatst op de beste plaats.

Vervolgens bepaalt de decoder de echte oplossing. Deze stap zorgt voor extra rekencomplexiteit maar maakt de oplossingen ruimte voor de local search kleiner. De decoder is een implementatie van KTNS. De focus lag hier sterk op performantie. Deze heeft namelijk de grootste invloed op rekentijd. De decoder bepaalt op zijn beurt ook het aantal switches. Die gebruikt worden om de oplossing telkens te evalueren.

Het werk wordt vergelekt en geanalyseerd door middel van benchmark instanties die reeds vele malen in de literatuur aanbod kwamen. Zowel kleine als grote instanties. Sommige stochastisch gegenereerd andere gebaseerd op echte use cases.

Inhoudsopgave

Voorwoord	iii
Samenvatting	v
Inhoud	viii
Figurenlijst	x
Tabellenlijst	xi
Lijst met afkortingen	xii
1 Inleiding	1
1.1 Introductie	1
1.2 Industriële significantie	3
1.3 Probleem Omschrijving	4
1.4 Notatie en terminologie	5
1.4.1 Gegeven	5
1.4.2 Oplossing	6
1.4.3 Objectief	9
1.5 Gelijkaardige jobs	10
1.5.1 Samenvatting	10
1.5.2 Toepassingsgebied	12
1.6 Samenvatting	12
2 Literatuur-studie	13
2.1 Introductie	13
2.2 Eigenschappen	13
2.2.1 The sequencing and Tooling problem	13
2.2.2 KTNS	14

2.2.3	1/0 Blocks	18
2.2.4	Modelering traveling salesman problem	19
2.2.5	Dominance property	20
2.2.6	Symmetrie eigenschap	21
2.3	Heuristieken	21
2.4	State of The art	22
2.4.1	Tool Sequencing	22
2.4.2	Tie Breaking	24
2.5	Samenvatting	25
3	Methodologie	26
3.1	Inleiding	26
3.2	Local Search	26
3.3	Constructieve Heuristiek	27
3.4	Besluit	28
4	Evaluatie	29
4.1	Decoder	29
4.2	KTNS	30
4.2.1	Algoritme	30
5	Local Search	32
5.1	Introductie	32
5.2	Swaps	33
5.2.1	Algoritme	33
5.2.2	Local search omgeving	33
5.2.3	Analyse	35
5.3	Ruin en recreate	37
5.3.1	Redenering	37
5.3.2	Methode: SMRI	38
5.3.3	Werking	38
5.3.4	Samenvatting	40
6	Metaheuristiek	41
6.1	Introductie	41
6.2	Steepest Descent	42
6.2.1	Algoritme	42

6.2.2 Analyse	43
6.3 Simulated Annealing	45
6.3.1 Introductie	45
6.3.2 Parameters	46
6.3.3 Algoritme	48
6.3.4 Motivering	48
6.4 Samenvatting	49
7 Resultaten	50
7.1 Introductie	50
7.2 Oplossingsmethode	51
7.2.1 Constructieve heuristiek	51
7.2.2 Meta heuristiek	52
7.3 Local search operator	53
7.3.1 Ruin en recreate configuratie	53
7.3.2 Vergelijking job swaps vs SMRI	54
7.4 State of the art	55
7.4.1 Configuratie	55
7.5 Resultaten	56
8 Conclusie	57
8.1 Samenvatting	57
A Beschrijving van deze masterproef in de vorm van een wetenschappelijk artikel	60
B Poster	67
C Architectuur dataprocessing	69

Lijst van figuren

1.1 Een boormachine	1
1.2 Voorbeeld met een magazijn en meerdere tools per job	2
1.3 Een voorbeeld van een flexibele machine: een CNC machine	3
1.4 Voorbeeld 1: een sequentie	7
1.5 Voorbeeld 1: overgang van A^* en σ naar P^*	8
1.6 Voorbeeld 1: aantal setups en switches	9
2.1 Voorbeeld 1: matrix P voor $\sigma = \{2, 0, 3, 1\}$, $C=3$	15
2.2 Voorbeeld 1: Toewijzen voor job 2, $C=3$	15
2.3 Voorbeeld 1: toewijzen voor job 0, $C=3$	16
2.4 Voorbeeld 1, $C=3$	17
2.5 Voorbeeld 1: Toepassen KTNS, $C=3$	17
2.6 Catanzaro $N=10$, $M=10$, $C=3$: Toepassen KTNS	18
2.7 Voorbeeld 1, $C=3$: 0-block aanwezig	18
2.8 Voorbeeld 1, $C=3$: Geen 0-blocks aanwezig, optimale oplossing.	19
2.9 Sparse vs Dense: $C=3$	20
2.10 Voorbeeld 2: dominance property, $C=2$	21
2.11 Voorbeeld 1: Tool Sequencing, $C=3$	22
2.12 Voorbeeld 1: Tool Graph	23
2.13 Voorbeeld 1: BFS op de Tool Graph	23
2.14 Voorbeeld 1: Greedy job sequencing	24
3.1 Local Search	26
4.1 Werking decoder	29
5.1 Voorbeeld van een job swap. Op een kleine sequentie	33
5.2 Omgeving job swaps op een kleine sequentie	33

5.3 Zoeklandschap: omgeving met swaps (oranje stippelijn)	35
5.4 Zoeklandschap: gewenste omgeving (oranje stippelijn)	36
5.5 SMRI: adhv voorbeeld 1	39
6.1 Oplossingenruimte: probleem steepest descent	43
6.2 Annealing: van een polykristal naar een monokristal [1]	45
7.1 Kost functie voor: Catanzaro instantie: $N = 30, M = 40, C = 17, i = 1$	52
8.1 Finale local search architectuur voor het SSP	57

Lijst van tabellen

1.1 Voorbeeld 1: sets van tools voor de set van jobs	5
1.2 Voorbeeld 1: binaire matrix A	6
1.3 Voorbeeld 1: sets van toegewezen tools voor de set van jobs	6
1.4 Voorbeeld 1: matrix A^*	7
1.5 Formulering samenvatting	11
5.1 SMRI samenvatting	40
6.1 Parameters voor het bepalen van probleem grootte v	47
7.1 Overzicht instanties	50
7.2 Beste SMRI configuraties	53
7.3 Roulette wheel selection: kansen	54
7.4 State of the art werken	55
7.5 Vergelijking Catanzaro dataset	56

Lijst van afkortingen

Afkorting	Betekenis
SSP	The Job Sequencing and Tool Switching Problem
KTNS	Keep Tool Needed Soonest
SP	Sequencing Problem
TP	Tooling Problem
SA	Simulated Annealing
SD	Steepest Descent
RR	Ruin & Recreate

Hoofdstuk 1

Inleiding

1.1 Introductie



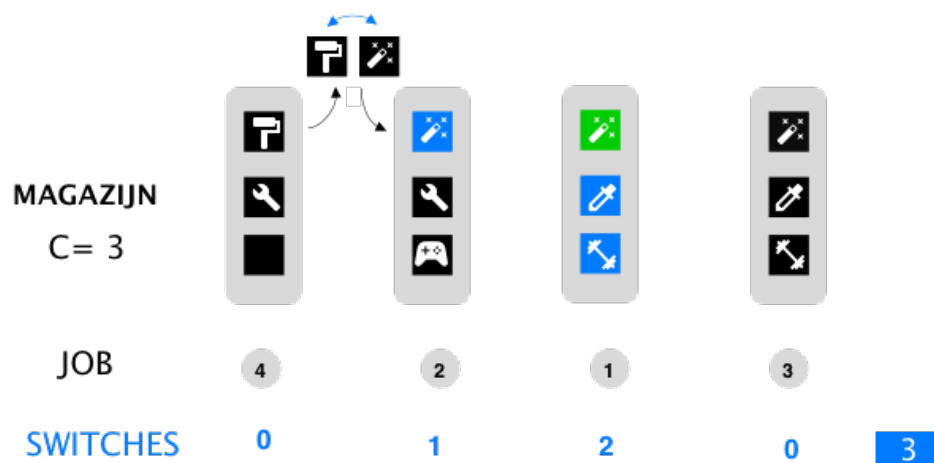
Figuur 1.1: Een boormachine

De opgave is een legplank bevestigen aan de muur. Er zijn vier jobs: twee gaten boren en twee vijzen vastschroeven. Er wordt gevraagd een boormachine te gebruiken waarin verschillende bits passen. Bits zijn tools. Er worden twee tools gebruikt: één voor het boren en één voor het vijzen. De vraag die zich nu stelt is: welke van onderstaande werkschema's het snelst is?

1. boren, tool veranderen, schroeven, tool veranderen, boren, tool veranderen, schroeven
2. alles boren, tool veranderen, alles schroeven.

Het is intuïtief om te kiezen voor optie 2: eerst alles boren, dan alles schroeven. Telkens switchen tussen twee bits is onverstandig. Zelfs met een beperkt aantal gaten en schroeven gaat er veel tijd verloren.

Bedenk nu dat de boormachine plaats heeft voor meerdere tools tegelijk. Laat ons dit een *magazijn* noemen waarin tools passen. Zoals het magazijn van een revolver waar meerdere kogels in passen. Dit maakt het mogelijk om jobs te doen die meerdere tools tegelijk gebruiken. Figuur ?? illustreert een dergelijke situatie.



Figuur 1.2: Voorbeeld met een magazijn en meerdere tools per job

De blauwe tools duiden aan wanneer er een switch plaatsvindt. Van job 4 naar job 2 is er bijvoorbeeld één tool switch. In dit geval wordt de verfborstel vervangen door de toverstaf.

Het groene tool hoort niet bij job 1. Het is blijven zitten voor job 3 die later komt. Dit vermijdt dat het tool er onnodig uitgehaald wordt.

Zwarte tools betekenen dat ze blijven zitten of zijn ingevoegd zonder een verwisseling.

Het oplossen van dit probleem bestaat uit:

1. De sequentie van jobs bepalen.
2. Extra tools aanwijzen voor een job, zodat latere jobs deze tools kunnen gebruiken.

Dit manueel oplossen is vrij arbeidsintensief. Bij grote problemen is het quasi onmogelijk om een goede oplossing te bepalen. Het ontwikkelen van een algoritme lijkt hier dus gepast. Het algoritme moet snel zijn en vooral goede oplossingen vinden. Het is duidelijk dat een goede oplossing vele switches kan vermijden.

1.2 Industriële significantie

Dit probleem is op grote schaal in de industrie aanwezig. Moderne productieomgevingen maken vaak gebruik van flexibele machines. Deze kunnen verschillende producten produceren door een reeks bewerkingen uit te voeren (boren, zagen, snijden...). Een klassiek voorbeeld is de CNC machine (Figuur 1.3). Deze heeft een magazijn waar verschillende tools in passen. Er is voldoende plaats voorhanden om de tools te bewaren voor één type product of job. Het bewaren van alle tools voor alle jobs is vaak onmogelijk. Daarom zijn tool switches nodig.



(a) CNC



(b) magazine

Figuur 1.3: Een voorbeeld van een flexibele machine: een CNC machine

Er is opgemerkt dat het switchen tussen tools lang kan duren; enkele voorbeelden hiervan zijn:

- Tools kunnen erg zwaar zijn (bijvoorbeeld in de metaal industrie).
- Afstellen van tools neemt vaak veel tijd in beslag.
- Wachten op personeel om tools te wisselen.
- ...

Het minimaliseren van tool switches is daarom gewenst, vooral omdat ze soms langer duren dan het uitvoeren van een job.

Wanneer een job uitgevoerd wordt en welke tools in het magazijn zitten heeft een effect op de volledige productieketen. Bij een slecht gekozen werkschema duurt fabricatie langer, een fabriek heeft hierdoor minder capaciteit, wat kan leiden tot onnodige uitbreidingen.

Dit werk heeft als doel de capaciteit van productieomgevingen te verhogen door switches te minimaliseren. Het gaat hier niet over een machine beter maken, een groter magazijn installeren of meer machines plaatsen, maar werk beter organiseren. Dit kost niets maar kan toch een enorme impact hebben.

Dit werk hoopt een belangrijke bijdrage te leveren in dit klassieke probleem binnen operational research.

1.3 Probleem Omschrijving

In de literatuur noemt dit probleem “The Job Sequencing and tool Switching Problem” of “SSP”. Formeel kunnen we het probleem als volgt definiëren:

Probleem

Gegeven een set van N jobs die elk een specifieke set van M tools nodig hebben en een magazijn met een bepaalde capaciteit C . Bepaal op welke positie in de sequentie elke job wordt uitgevoerd en welke tools nodig zijn op die positie. Het objectief is minimaliseer het aantal tool switches.

Vele varianten op dit probleem bestaan:

- Parallelle machines gebruiken.
- Rekening houden met tool slijtage.
- Rekening houden met tool size.
- Tool afhankelijke plaatsen in het magazijn.
- ...

Alhoewel deze variaties interessant zijn focust dit werk op de uniforme variant. Het volgende is een opsomming van het model dat hier gebruikt wordt:

1. Eén machine of één magazijn is beschikbaar.
2. Alle tools van een job passen gelijktijdig in het magazijn.
3. Elk tool neemt één plaats in binnen het magazijn. [10]
4. De tijd voor een tool te verwijderen of toe te voegen is constant. [10]
5. Tools kunnen niet gelijktijdig vervangen worden. [10]
6. Tools verslijten niet. [10]

1.4 Notatie en terminologie

1.4.1 Gegeven

Het aantal jobs is $M \in \mathbb{N}$ en het aantal tools is $N \in \mathbb{N}$. Vaak zullen er meer tools zijn dan jobs, $N \geq M$.

Laat $J = \{0, 1, 2, \dots, M\}$ de set van jobs zijn en $T = \{0, 1, 2, \dots, N\}$ de set van alle tools. De set van tools die vereist zijn voor een job is T_j , deze is een subset van T . Dus $j \in J, T_j \subseteq T$.

Ten slotte is de magazijn-capaciteit $C \in \mathbb{N}$. Alle tools van een job passen gelijktijdig in het magazijn. Dus $C \leq \max(|T_j|), j \in J$.

Tabel 1.1 illustreert een voorbeeld van deze sets. Hierbij is de set van jobs $J = \{0, 1, 2, 3\}$. De tools vereist voor job 2 zijn $T_2 = \{1, 2, 3\}$. De magazijn-capaciteit is $C = 3$

Jobs	0	1	2	3
Tools	0	1	1	1
	3	2	2	
			3	
magazijn-capaciteit = 3				

Tabel 1.1: Voorbeeld 1: sets van tools voor de set van jobs

Een instantie wordt typisch weergegeven door een *tool-job matrix*. Dit is een $M \times N$ binaire matrix A waarbij:

$$\begin{aligned}
 a_{ij} &= 1 && \text{Als tool } i \in T, \text{ vereist is voor job } j \in J \\
 &= 0 && \text{Als tool } i \in T, \text{ niet vereist is voor job } j \in J
 \end{aligned}$$

Elke kolom stelt dus een job voor en elke rij een tool. Tabel 1.2 illustreert de *tool-job matrix* A van voorbeeld 1. Het is duidelijk dat een kolom j overeenstemt met T_j . In het voorbeeld is kolom 2 bijvoorbeeld $[0, 1, 1, 1]$, dit komt overeen met $T_2 = \{1, 2, 3\}$.

	Jobs	0	1	2	3
Tools	0	1	0	0	0
	1	0	1	1	1
	2	0	1	1	0
	3	1	0	1	0

Tabel 1.2: Voorbeeld 1: binaire matrix A

1.4.2 Oplossing

Een oplossing voor het probleem bestaat uit:

- Een sequentie σ . Dit is een permutatie van de set van jobs J .
- De set van **toegewezen** tools voor elke job. Dit is T_j^* en $j \in J$. Deze is een subset van T . Dus $j \in J, T_j^* \subseteq T$.

Tabel 5.1 illustreert een mogelijke oplossing. Een set van toegewezen tools bestaat uit de vereiste tools voor een job en extra toegewezen tools. Deze extra tools zijn om switches te minimaliseren. Daarom is $T_j \subseteq T_j^*, j \in J$. In het voorbeeld is $T_1 = \{1, 2\}$ en $T_1^* = \{1, 2, 3\}$. Extra toegewezen tools zijn onderstreept.

Jobs	0	1	2	3
Tools	0	1	1	1
		3	2	2
		<u>1</u>	<u>3</u>	3
magazijn-capaciteit = 3				

Tabel 1.3: Voorbeeld 1: sets van toegewezen tools voor de set van jobs

Een oplossing voor een probleem stellen we in dit werk voor door $S = \{\sigma, A^*\}$:

- Een sequentie σ . Dit is een permutatie van de set van jobs J .
- Een “Augmented” *tool-job matrix* , A^* :

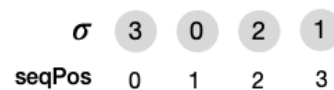
$$a_{ij}^* = \begin{cases} 1 & \text{Als tool } i \in T, \text{ toegewezen is aan job } j \in J \\ 0 & \text{Als tool } i \in T, \text{ niet toegewezen is aan job } j \in J \end{cases}$$

Tabel 1.4 illustreert een voorbeeld van een augmented *tool-job matrix* A^* . Een kolom j stemt nu overeen met T_j^* . Kolom 1 is bijvoorbeeld $[0, 1, 1, 1]$, dit komt overeen met $T_1^* = \{1, 2, 3\}$

Figuur 1.4 illustreert sequentie. Elke positie in de sequentie noemen we een *sequence position* of *seqPos*. Een job wordt dus toegewezen aan een positie in de sequentie σ . In dit voorbeeld is de sequentie $\sigma = \{3, 0, 1, 2\}$

	Jobs	0	1	2	3
Tools	0	1	0	0	0
	1	<u>1</u>	1	1	1
	2	0	1	1	0
	3	1	<u>1</u>	1	0

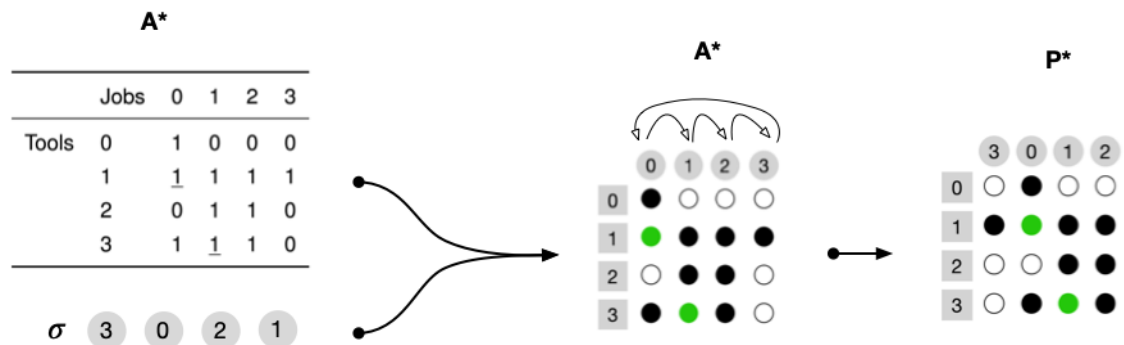
Tabel 1.4: Voorbeeld 1: matrix A^*



Figuur 1.4: Voorbeeld 1: een sequentie

1.4.2.1 Visuele voorstelling

Als alternatief kunnen we ook de kolommen van A^* permuteren aan de hand van de sequentie σ . Dat geeft een visuele voorstelling van het SSP. We noemen deze matrix P^* . Om het duidelijker te maken worden de cijfers vervangen door bolletjes. Een ingekleurd bolletje is een aanwezig tool. Een niet ingekleurd bolletje is een niet aanwezig tool. Vereiste tools zijn zwart gekleurd en extra toegewezen tools groen. Figuur 1.5 maakt dit duidelijk.



Figuur 1.5: Voorbeeld 1: overgang van A^* en σ naar P^*

Door deze voorstelling wordt de evolutie van het magazijn zichtbaar. Het is onmiddellijk duidelijk welke tools nodig zijn op welk moment. De eerste kolom stelt dus de eerste job voor in de sequentie, de 2de kolom de 2 de job in de sequentie enzovoort. Het is ook duidelijk dat er steeds maximaal C tools aanwezig zijn.

Dankzij deze voorstelling is het mogelijk om te zien wanneer een tool wordt ingevoegd en wordt verwijderd.

1.4.3 Objectief

Het objectief is om het aantal tool switches te minimaliseren:

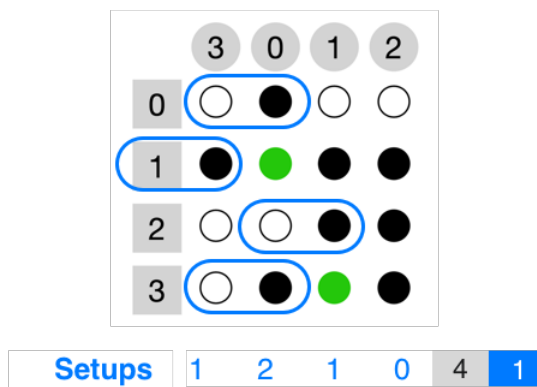
$$MIN(\text{\#aantal switches}) \quad (1.1)$$

- Een tool switch is de combinatie van:
 - Het invoegen van een **niet aanwezig** tool.
 - Het verwijderen van een **aanwezig** tool.

Crama Et. al. [10] gaf aan dat het aantal switches gelijk is aan:

$$\text{\#aantal switches} = \text{\#ingevoegde tools} - C \quad (1.2)$$

In figuur 1.6 is het aantal switches gelijk aan 1 want er zijn 4 tool setups en $C = 3$



Figuur 1.6: Voorbeeld 1: aantal setups en switches

1.5 Gelijkaardige jobs

Het concept van gelijkaardig jobs wordt vaak vermeld. Gelijkaardige jobs zijn jobs die een aantal tools gemeenschappelijk hebben. Hoe meer tools ze gemeenschappelijk hebben hoe sterker gelijkaardig ze zijn, hoe minder hoe zwakker.

In figuur 1.6 is job 1 en job 2 bijvoorbeeld sterk gelijkaardig. Job 3 en job 0 zijn helemaal niet gelijkaardig ze hebben geen enkel tool gemeenschappelijk. Job 3, 1 en 2 zijn zwak gelijkaardig samen, waarbij job 2, 1 meer gelijkaardig zijn.

Het is moeilijk om hier een exacte formulering op te plakken.

1.5.1 Samenvatting

Het volgende deel vat de formulering samen in een overzicht. Bij elk gebruikt symbool kunnen zowel de mogelijke waarden als de betekenis afgelezen worden. Enkele additionele symbolen zijn toegevoegd om het model uit te breiden.

Symbolen	Waarde	Betekenis
N	$\mathbb{N}$	Aantal Jobs
M	$\mathbb{N}$	Aantal Tools
C	$\mathbb{N}$	magazijn capaciteit
J	$\{0, 1, \dots, N\}$	Set van Jobs
J_i	$\{0, 1, \dots, N\}$	Set van Jobs <u>vereist</u> voor Tool i
$^c J_i$	$\{0, 1, \dots, N\}$	Set van Jobs <u>niet vereist</u> voor Tool i
J_i^*	$\{0, 1, \dots, N\}$	Set van Jobs <u>toegewezen</u> aan Tool i
$^c J_i^*$	$\{0, 1, \dots, N\}$	Set van Jobs <u>niet toegewezen</u> aan Tool i
T	$\{0, 1, \dots, M\}$	Set van Tools
T_j	$\{0, 1, \dots, M\}$	Set van Tools <u>vereist</u> voor Job j
$^c T_j$	$\{0, 1, \dots, M\}$	Set van Tools <u>niet vereist</u> voor Job j
T_j^*	$\{0, 1, \dots, M\}$	Set van Tools <u>toegewezen</u> aan Job j
$^c T_j^*$	$\{0, 1, \dots, M\}$	Set van Tools <u>niet toegewezen</u> aan Job j
a_{ij}	$\{0, 1\}$	$a_{ij} = 1$ als tool j toegewezen is aan Job i , 0 anderszids
$A_{ij} = (a_{ij})$		Job Tool Matrix (vereiste tools)
$A_{ij}^* = (a_{ij}^*)$		Augmented Job Tool Matrix (toegewezen tools)
$P_{ij} = (a_{ij})$		Permutated, Job Tool Matrix (vereiste tools)
$P_{ij}^* = (a_{ij}^*)$		Augmented, Permutated, Job Tool Matrix (toegewezen tools)
σ		Job Sequentie
$S = \{\sigma, A_{ij}^*\}$		Oplossing, bevat de Job sequentie en Augmented Job Tool Matrix

Tabel 1.5: Formulering samenvatting

1.5.2 Toepassingsgebied

Bovenstaande abstractie maakt het mogelijk om een reflectie te maken van de echte wereld. Het maakt niet uit over welk type tool het gaat, welk type machine of welke setting. Eenmaal een echt situatie geprojecteerd is op het bovenstaande model kan dezelfde oplossingsmethode gebruikt worden. Bijvoorbeeld: Een magazijn met 2 plaatsen wordt magazijn size $C = 2$. Een zaag, een hamer en een boor zijn tools $T = \{0, 1, 2\}$. Drie stukken metaal die bewerkt worden zijn jobs $J = \{0, 1, 2\}$.

1.6 Samenvatting

Dit hoofdstuk introduceerde het Job Sequencing & Tool Switching Problem aan de hand van een vereenvoudigd voorbeeld. De transitie werd zo gemaakt naar het volledige probleem. Het werd duidelijk dat dit niet enkel een willekeurig abstract probleem is, maar dat het een belangrijke industriële significantie heeft. Vervolgens werd er op een formele wijze een beschrijving en een formulering gemaakt. Aan de hand van deze formulering werden ook enkele visualisaties geïntroduceerd. Deze maken interpretatie mogelijk. Verder maakt de abstractie het mogelijk om het probleem toe te passen op een groot toepassingsgebied. In het volgende hoofdstuk worden enkele interessante eigenschappen uit de literatuur besproken.

Hoofdstuk 2

Literatuur-studie

2.1 Introductie

Het Job Sequencing & Tool Switching Problem is reeds uitgebreid bestudeerd geweest in de literatuur. Het objectief van deze literatuur-studie is niet om elk werk te beschrijven. Een recente *state of the art literature review* door **Calmels** [3] geeft een goede overview van de complete literatuur tot 2017. Het beschrijft verschillende variaties, methoden en resultaten van het probleem. Het doel van deze literatuur-studie is, inzicht geven in het probleem, en verschillende eigenschappen blootleggen die een betekenis kunnen hebben voor dit werk.

2.2 Eigenschappen

2.2.1 The sequencing and Tooling problem

Tang en Denardo introduceerde het probleem als eerste in 1988 [12]. Kort daarna introduceerde **Bard** [11] hetzelfde probleem. Centraal stond het idee om het probleem te ontbinden in 2 subproblemen:

- The sequencing problem (SP): De volgorde bepalen waarin de tools uitgevoerd moeten worden. Het sequencing problem is $\mathcal{NP}$ hard.
- The tooling problem (TP): Gegeven de job-sequentie bepalen welke tools gewenst zijn op elke positie. Het is oplosbaar in *Polynomiale* tijd.

Het is natuurlijk om deze ontbinding hierin te zien. Een oplossing voor dit probleem bestaat immers uit het vinden van een gepaste job sequentie en een gepaste tool configuratie. **Crama** [10] gaf in 1994 een formeel bewijs dat het sequencing problem $\mathcal{NP}$ hard is voor $C \geq 2$. Dit gaf reeds de indicatie dat het vinden van een gepaste sequentie een moeilijk combinatorisch probleem is.

2.2.2 KTNS

Hoewel het “Sequencing Problem” moeilijk op te lossen is, demonstreerden **Tang en Denardo** [12] dat het tooling problem triviaal is. Ze introduceerden de “keep tool needest soonest” policy of “KTNS”, deze betekent het volgende:

- Tools worden enkel toegevoegd als ze vereist zijn.
- Wanneer tools verwijderd moeten worden om plaats te bieden aan nieuwe, verwijder diegene die het minst snel nodig zijn. i.e. 'keep the tools needed soonest'

Door deze regels toe te passen kan elke job zijn vereiste tools gebruiken. Bovendien gebruikt het de resterende plaats in het magazine om tools toe te wijzen die later vereist zijn. Het maakt hierbij de keuze tools bij te houden die het snelst nodig zijn door volgende jobs.

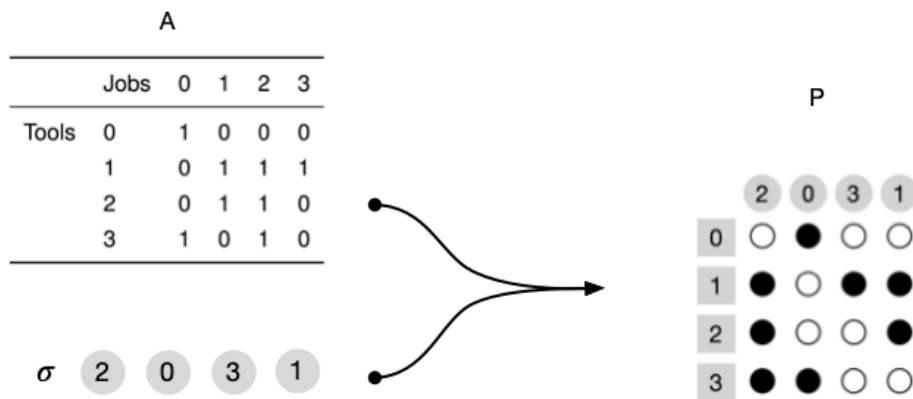
Tang en Denardo bewezen dat voor een sequentie σ de *optimale* tool configuratie gevonden kan worden met KTNS. Dit betekent dat voor de gekozen sequentie de switches minimaal zijn.

Deze belangrijke eigenschap laat toe het probleem te modelleren als het zoeken naar de beste job sequentie. Telkens wanneer een sequentie σ is gevonden kan deze omgezet worden naar een volledige oplossing $S = \{\sigma, A^*\}$. Dus $S = KTNS(\sigma)$

2.2.2.1 Voorbeeld

Omdat KTNS zo een belangrijke rol speelt is het essentieel deze te begrijpen. We gebruiken voorbeeld 1 uit hoofdstuk 1 om KTNS te verduidelijken. KTNS wordt hieronder op een visuele manier toegepast. In hoofdstuk 4, dat draait rond evaluatie, wordt het algoritme voorgesteld. Daar zal de focus vooral zijn op datastructuren en performantie.

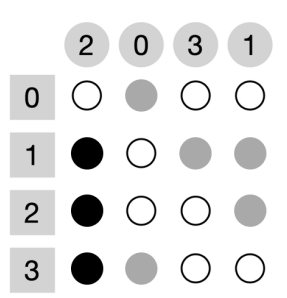
STAP 0



Figuur 2.1: Voorbeeld 1: matrix P voor $\sigma = \{2, 0, 3, 1\}$, $C=3$

Vorm de matrix P aan de hand van de gegeven sequentie $\sigma = \{2, 0, 3, 1\}$. Dit maakt het makkelijker om visueel de juiste tools toe te wijzen. Zie figuur 2.1

STAP 1

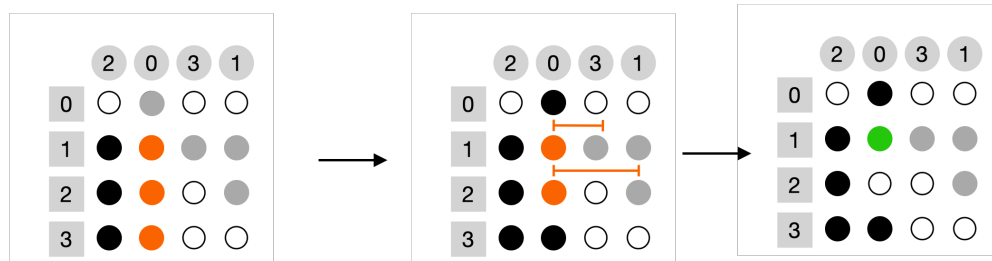


Figuur 2.2: Voorbeeld 1: Toewijzen voor job 2, $C=3$

- Start bij de eerste job in σ : job 2.
- Voeg de **vereiste tools** toe: $T_2 = \{1, 2, 3\}$ dus $T_2^* = \{1, 2, 3\}$, deze kleuren we zwart.

Zie figuur 2.2

STAP 2

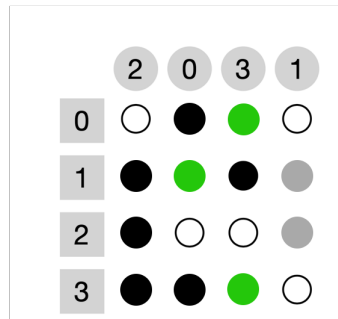


Figuur 2.3: Voorbeeld 1: toewijzen voor job 0, $C=3$

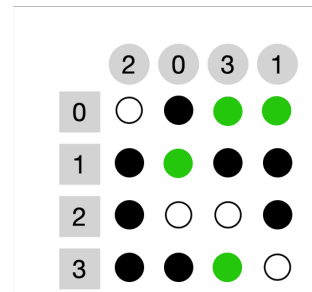
- Ga naar de volgende job in σ : job 0.
- Laat de **toegewezen tools** van de vorige job zitten.
 - $T_2^* = \{1, 2, 3\}$
 - Deze kleuren we oranje.
- Voeg de **vereiste tools** van de huidige job toe
 - $T_0 = \{0, 3\}$
 - Deze kleuren we zwart. Let op: tool 3 was al blijven zitten, omdat het vereist is kleuren we het ook zwart.
- Respecteer de magazijn-capaciteit.
 - De magazijn-capaciteit is $C = 3$. Er zit dus één tool teveel.
 - Hou de tools die het snelst nodig zijn. De vereiste tools blijven altijd zitten.
Tool 1 is nodig in job 3 en tool 2 in job 1. Job 3 komt voor job 1.
Dus hou tool 2, i.e. “keep tool needest soonest”
 - Tool 2 is toegewezen aan de huidige job dankzij KTNS.
 - * In dit werk noemen we dit een KTNS tool. We kleuren het telkens groen. Dit maakt duidelijk dat dit niet vereist is maar toegewezen is om switches te minimaliseren.
 - Uiteindelijk krijgen we de toegewezen tools voor de huidige job: $T_0^* = \{0, 1, 3\}$

Zie figuur 2.3

STAP 3



(a) Toewijzen voor job 3

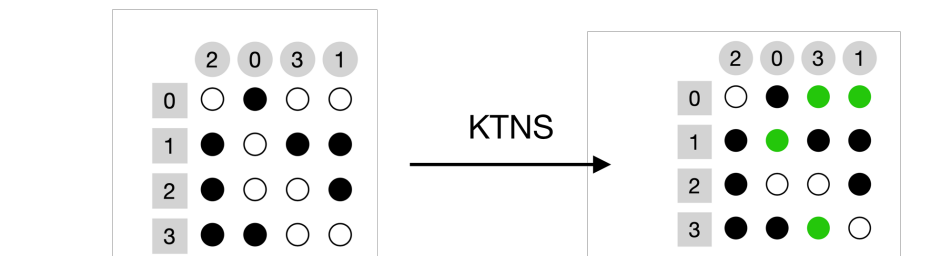


(b) Toewijzen voor job 1

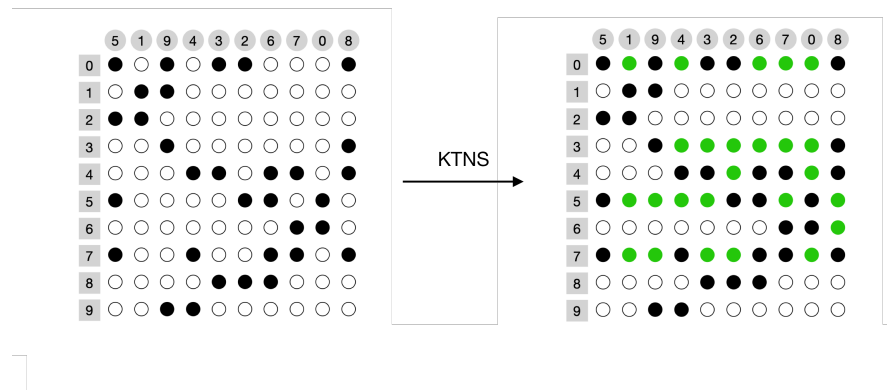
Figuur 2.4: Voorbeeld 1, $C = 3$

- Herhaal stap 2. Tot elke job bezocht is. Zie figuur 2.4

Na het uitvoeren van KTNS krijgen we dus volgend resultaat (zie figuur 2.5):

Figuur 2.5: Voorbeeld 1: Toepassen KTNS, $C=3$

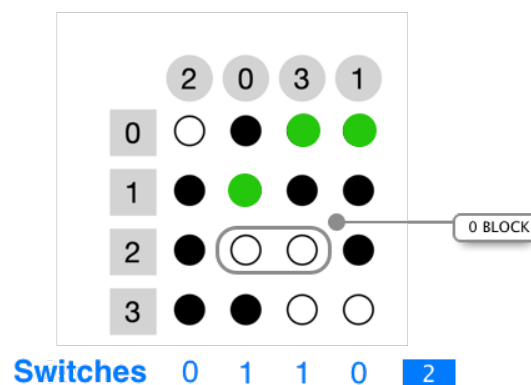
KTNS is een simpele methode. Maar neemt al heel wat stappen bij relatief kleine instanties. Het volgende voorbeeld toont een echte (kleine) instantie waarop KTNS wordt toegepast. Hierop is duidelijk zichtbaar hoe KTNS ons helpt bij het verminderen van job switches.



Figuur 2.6: Catanzaro N=10, M=10, C=3: Toepassen KTNS

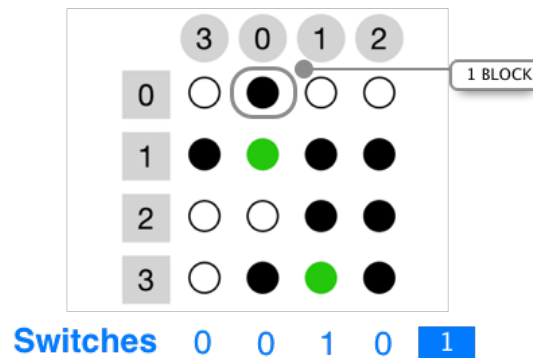
2.2.3 1/0 Blocks

Crama Et. al. [10] brachten het idee naar voren om 0/1 blocks te herkennen in het probleem. Een 0 block is een sequentie van niet toegewezen tools, voorafgegaan en gevolgd door een toegewezen tool. Zoals te zien is op figuur 2.7. Ze merkten op dat het aantal tool switches enkel te reduceren is als we een volledig 0 block kunnen elimineren. Dit betekent dat we het objectief kunnen herformuleren als het reduceren van het aantal 0 blocks. Daarnaast herformuleerden ze KTNS als het invullen van het maximum aantal 0 blocks, terwijl elke job zijn vereiste tools kan gebruiken en de magazijn-capaciteit niet overschreden wordt.



Figuur 2.7: Voorbeeld 1, C=3: 0-block aanwezig

Het herkennen van 0-blocks is interessant omdat een probleem hierdoor een lokale betekenis krijgt. In figuur 2.7, speelt tool 2 bijvoorbeeld een grote rol in het toevoegen van switches. Overigens zijn de betrokken jobs in een 0 block ook zichtbaar.



Figuur 2.8: Voorbeeld 1, $C=3$: Geen 0-blocks aanwezig, optimale oplossing.

Ten slotte geeft het inzicht van de 0-blocks ook weer wat de theoretisch optimale oplossing is. De theoretisch beste oplossing is wanneer we een tool maximaal één keer toevoegen en één keer verwijderen. Een oplossing ziet er dan uit zoals figuur 2.8. Hierbij zijn er geen 0 blocks aanwezig. Deze situatie is natuurlijk niet altijd mogelijk. Het illustreert wel hoe een goede oplossing er uit ziet.

2.2.4 Modelering traveling salesman problem

Tang en Denardo introduceerden was om het SSP te modelleren als een instantie van het Traveling Salesman problem.

TSP

Het traveling salesman problem kan gemodelleerd worden als een undirected graaf waarin elke vertex een stad voorstelt en de edges tussen steden de straten. Het doel is om elke stad éénmaal te bezoeken, waarbij de afgelegde afstand geminimaliseerd wordt.

Tang en Denardo definieerde een graaf $G = (V, E, lb)$ [12]. De vertices V stellen de set van jobs J voor. De edges $\{i, j\} \in E$ stellen job j voor die job i volgt, waar $i, j \in J$. Ten slotte is het gewicht van een edge $\{i, j\}$ het volgende:

$$LB(i, j) = \max(|T_i \cup T_j| - C, 0) \quad (2.1)$$

Dit is een lower bound voor het aantal switches tussen job i en job j . Het probleem bestaat er nu uit alle vertices te bezoeken en de kortste weg af te leggen. De volgorde die gevonden wordt is de

job sequentie. Na het vinden van de job sequentie σ (SP) zijn de toegewezen tools te vinden met KTNS.

Door het probleem zo te modelleren konden **Tang en Denardo** aantonen dat het probleem $\mathcal{NP}$ hard is.

Crama Et. al merkte op dat het oplossen van het probleem met de lower bound van **Tang and Denaro** [12] niet goed werkt als de instantie bestaat uit een sparse *tool-job matrix*.

Er is sprake van een sparse *tool-job matrix* als de tools van jobs weinig plaats innemen in het magazijn. Het verschil zou duidelijk moeten zijn aan de hand van onderstaande figuur [2.9]



Figuur 2.9: Sparse vs Dense: $C = 3$

De lower bound werkt niet goed bij een sparse *tool-job matrix* omdat KTNS nog veel tools kan toevoegen. Bij een goed gekozen sequentie σ zal KTNS misschien veel switches vermijden, bij een andere misschien heel weinig. Daarom is het moeilijk in te schatten hoeveel switches er nu effectief zijn tussen 2 jobs.

Indien het magazijn telkens hoofdzakelijk bestaat uit tools van de huidige job dan werkt de lower bound echter erg goed. Op deze manier kunnen bestaande TSP algoritmen gemakkelijk toegepast worden.

2.2.5 Dominance property

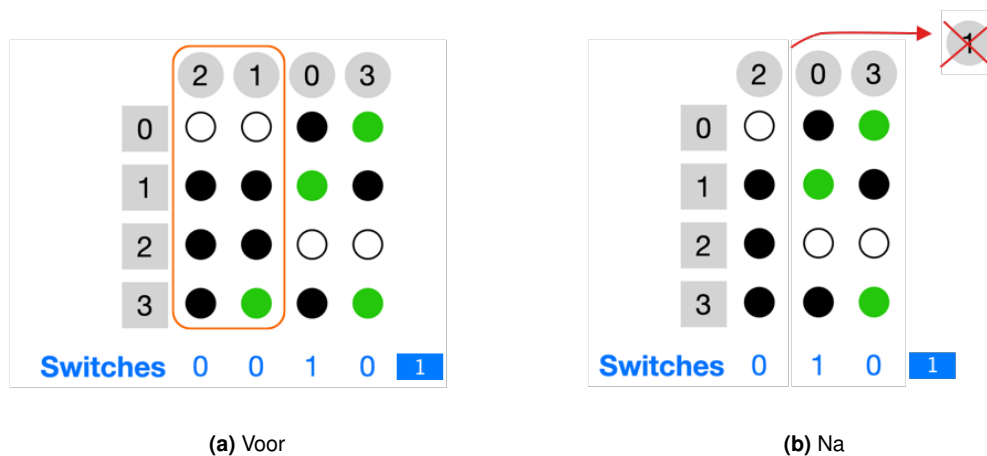
Als de tool set van een job i een subset is van een andere job j , zoals in vergelijking [2.2]:

$$T_i \subseteq T_j \quad i, j \in J \quad (2.2)$$

Dan verandert de optimale oplossing niet wanneer job i weggelaten wordt van de instantie.

Tang and Denardo [12] merkten deze eigenschap op en noemde het "the dominance property".

2.2.5.1 Redenering



Figuur 2.10: Voorbeeld 2: dominance property, $C = 2$

In figuur 2.10 is de tool set van job 1 een subset van job 2. Het is duidelijk dat $T_1 = \{1, 2\} \subseteq T_2 = \{1, 2, 3\}$. De dominance property geldt omdat men job 1 na job 2 kan plaatsen.

Hierbij kunnen de vereiste tools blijven zitten. De overige tools van job 2 kunnen ook blijven dankzij KTNS. Hierdoor zijn er geen switches tussen job 2 en 1.

Deze eigenschap laat toe om instanties te verkleinen, wat ze makkelijk oplosbaar maken. De meeste benchmark instanties bevatten deze type jobs niet omdat ze verwaarloosbaar zijn.

2.2.5.2 Premisse

De plaatsing van gelijkaardige jobs kan een verschil maken op het aantal switches. Het kan interessant zijn om gelijkaardige jobs te ordenen zodat ze een positieve invloed hebben op elkaar.

Bovendien leunt het job grouping problem dicht aan bij het SSP. Het herkennen van relaties tussen jobs zou ons kunnen helpen om het probleem op te lossen. Het is ook een oplossing-strategie die velen zouden toepassen om het probleem manueel op te lossen.

2.2.6 Symmetrie eigenschap

Ghani et al. [9] herkende een symmetrie eigenschap. Ze verklaarden en bewezen dat de inverse van elke job sequentie hetzelfde aantal switches heeft.

2.3 Heuristieken

Vele pogingen zijn ondernomen om het SSP op te lossen. Zowel exacte methoden als heuristische methoden zijn gebruikt.

Het voordeel van een exacte methode (ILP, MIP...) is dat men de optimale oplossing vind. Daartegenover staat dat deze methoden vaak trager zijn dan heuristische. Heuristische proberen de oplossing te benaderen. De optimale oplossing vinden is mogelijk, maar het kan niet bewezen worden dat deze optimaal is.

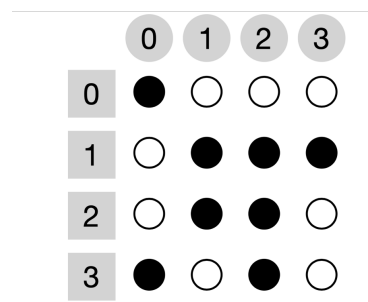
Camels toonde in 2017 aan dat er 25 werken heuristische methoden gebruiken en 10 werken exacte methoden [3]. De moeilijkheid van het SP speelt hier de grootste rol. Exacte methoden hebben geen optimale oplossingen voor meer dan 25 jobs gevonden. Recente werken die State of the Art resultaten behalen behoren allemaal tot de heuristische categorie. Als we de state of the art willen benaderen zal een heuristische methode meer kans maken.

2.4 State of The art

Het volgende deel bespreekt interessante componenten uit State of The Art artikelen die potentie kunnen bieden in het oplossen van dit probleem. De premisse dat hun resultaten als beste kunnen beschouwd worden, en de logica en elegantie die deze methoden vertonen is de motivatie om deze hier in de kijker te zetten.

2.4.1 Tool Sequencing

Paiva et al. [5] stelde een aantrekkelijke constructieve heuristiek voor (initiële oplossing). We zullen deze uitleggen met behulp van voorbeeld 1 (zie figuur 2.11):

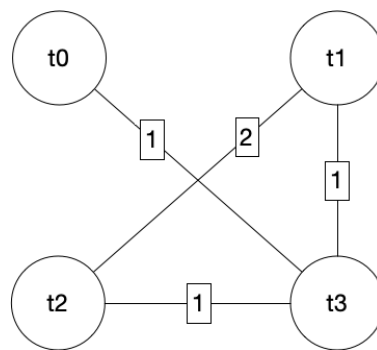


Figuur 2.11: Voorbeeld 1: Tool Sequencing, C=3

STAP 1: Stel de tool graph op.

Ze definieerden een tool Graph als een *weighted undirected graph* $G = (V, E)$, waar de vertices V de set van tools T voorstellen, de edges stellen een paar van tools voor $\{i, j\} \in E$. Het gewicht van de edges is de hoeveelheid jobs die beide tool i en j nodig hebben. Als een edge gewicht 0 heeft wordt deze niet toegevoegd.

Toegepast op het voorbeeld ziet dit er zo uit (zie figuur 4.1):

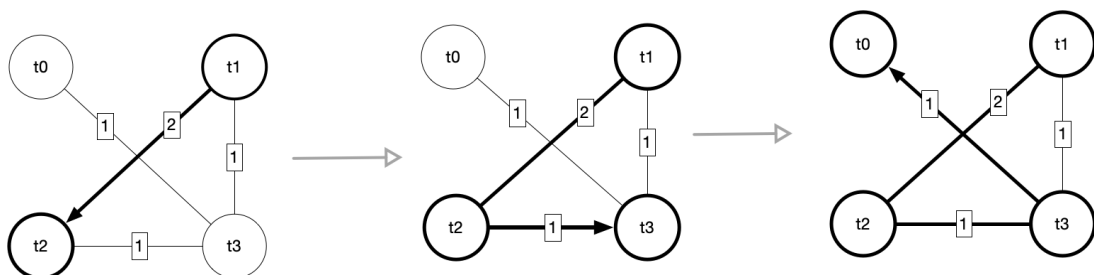


Figuur 2.12: Voorbeeld 1: Tool Graph

Tool 1 en 2 zijn beide aanwezig voor 2 jobs bijvoorbeeld.

Fascinerend genoeg is deze graaf niet afhankelijk van de job sequentie. Deze overweegt alle jobs tegelijk [5].

STAP 2: Voer BFS uit en stel de tool sequence op



Figuur 2.13: Voorbeeld 1: BFS op de Tool Graph

De volgende stap is om Breadth First Search toe te passen op de graph, met een extra criterium. De hoogste gewichten van edgen bezoekt men eerst. Indien een vertex wordt bezocht dan voegt men deze toe aan de tool sequence. De BFS wordt gestart bij een random tool. In figuur 4.1 wordt dit stap per stap uitgevoerd.

Zo kan men de tool sequence $\phi = \{1, 2, 3, 0\}$ vinden.

STAP 3: Voer greedy job sequencing uit en stel de job sequence op

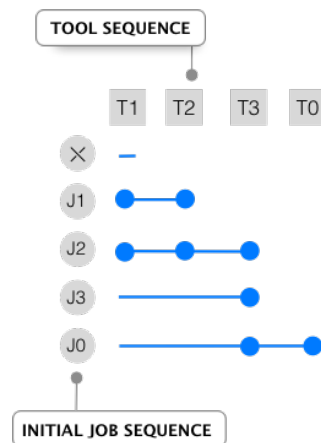
Nu voeren we greedy job sequencing uit om de job sequence te vinden. Hierbij wordt de tool sequence σ opgebouwd. Er wordt een set A van beschikbare tools steeds vergroot door de tool sequence ϕ af te gaan. Stap per stap wordt een tool toegevoegd.

Telkens wanneer een tool toegevoegd is kijken we of er jobs zijn waarbij $T_i \subseteq A$ met $i \in J$. Als dit het geval is voegen we de job toe aan σ .

Wanneer er meerdere jobs zijn waarvoor $T_i \subseteq A$ geldt, dan voegen we eerst de job toe die het laagst

aantal switches zal veroorzaken. Hierbij voeren we KTNS uit op de partiële sequence σ en bepalen daarna het aantal switches.

We verwijzen naar **Paiva Et. al.** [5] voor meer details en het algoritme. Figuur 2.14 illustreert dit visueel op het voorbeeld. De gevonden job sequence is $\sigma = \{1, 2, 3, 0\}$.



Figuur 2.14: Voorbeeld 1: Greedy job sequencing

Deze methode stelt vast hoe jobs gerelateerd zijn aan elkaar, door eerst een tool sequence op te stellen.

2.4.2 Tie Breaking

Mecler et al. [4] herkende dat vele oplossingen op het SSP dezelfde oplossing hebben, of slechts verschillen met een gering aantal switches. Het zoeklandschap heeft vrij veel plateaus. Als we alle oplossingen zouden interpreteren als een gebergte dan zijn goede oplossingen hoge bergen en slechte oplossingen lage bergen. Het probleem met plateaus is dat alles op elkaar lijkt. Daardoor is het moeilijk om uit het plateau te ontsnappen en de bergen te ontdekken.

Mecler et al. [4] stellen een nieuwe objectief functie voor. Ze noemen het een **Tie Breaking Objective**. Ze baseren de objectief functie op het idee van de 0/1 blocks van **Crama Et. al.** [10]. Ze stellen dat tijdens het zoeken naar een oplossing het gemakkelijker is om korte zero blocks te reduceren.

2.4.2.1 Formulering

- Laat $k_j(S)$ de hoeveelheid 0 blocks zijn voor elk tool $j \in T$, voor een oplossing S .
- Laat $b_j^x(S)$ de lengte zijn van het x de 0 block, waarbij $x \in k_j(S)$, voor een oplossing S .

$$\min(\sum_{j=1}^M \sum_{x=1}^{k_j(S)} \sqrt{b_j^x(S)}) \quad (2.3)$$

Door het gebruik van de vierkantswortel zullen oplossingen met korte 0 blocks en lange 0 blocks beter scoren. Dit is het geval omdat: korte makkelijker te reduceren zijn en lange duiden op weinig tool switches.

2.5 Samenvatting

In dit hoofdstuk zijn eigenschappen besproken die nuttig zijn in het oplossen van het SSP. Het probleem opsplitsen in het Sequencing Problem en Tooling Problem is nuttig. Hierdoor is KTNS inzetbaar, een policy die optimaal werkt voor het tooling problem.

De mogelijkheid om het SSP te modelleren als een routing problem maakt het mogelijk om ook oplossingsmethoden uit die literatuur te overwegen.

De dominance property illustreerde de samenhang van jobs en het effect dat ze hebben op switches. De invloed van zero blocks en hun lokale aart maken ze een goede kandidaat om ze systematisch weg te werken. Beide eigenschappen zijn te overwegen tijdens het oplossen.

Het is belangrijk rekening te houden met het platte zoeklandschap van de normale objectief functie, het gebruiken van een objectief functie die plateaus tegengaat kan nuttig zijn. Het volgende hoofdstuk introduceert de oplossingsmethode die ontwikkeld is.

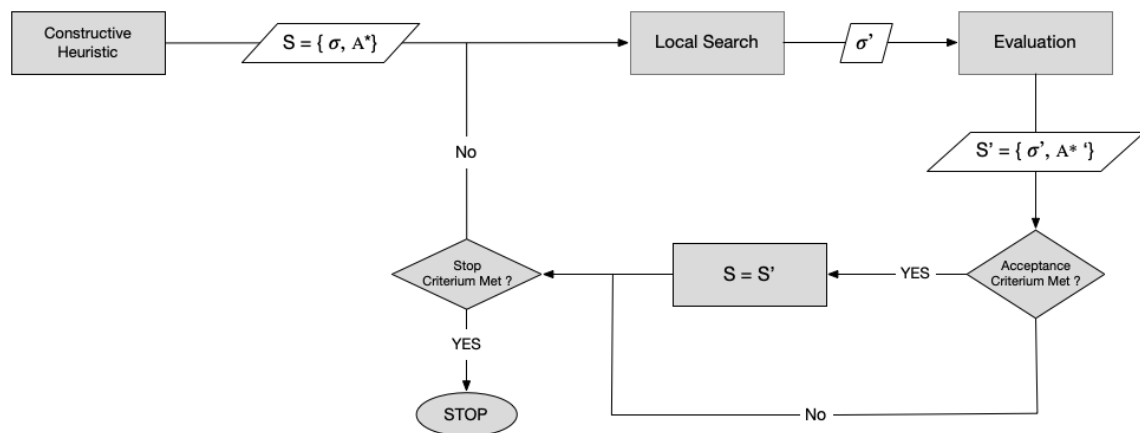
Hoofdstuk 3

Methodologie

3.1 Inleiding

Heuristieken zijn voor het oplossen van het SSP aangewezen als we de “State Of The Art” willen benaderen. De opdracht was los dit op met Local Search. Dit onderzoek was een afwisselingen van implementeren, experimenten en analyseren. Verschillende strategieën werden toegepast. Eerst om inzicht te krijgen, nadien om goede resultaten te krijgen. De volgende hoofdstukken trachten de werking en de inzichten weer te geven.

3.2 Local Search



Figuur 3.1: Local Search

Figuur [3.1](#) illustreert de werking van local search. Bij Local Search gaan we op zoek naar een betere oplossing door een bestaande oplossing (lokaal) aan te passen. Het aanpassen van deze oplossing kan door gebruik te maken van local search operatoren.

De operator die je gebruikt als je jezelf verplaatst is bijvoorbeeld links, rechts, vooruit en achterwaarts een stap zetten. Op dezelfde manier willen we met ons algoritme naar een oplossing gaan

waar het aantal switches het laagste is. De operator zal dus de huidige oplossing aanpassen naar een nieuwe oplossing.

Na het aanpassen wordt een oplossing geëvalueerd. In ons geval is het evalueren het bepalen van het aantal switches. Op basis hiervan kan de oplossing al dan niet geaccepteerd worden. Wordt het geaccepteerd? Dan wordt dit de huidige oplossing en passen we deze verder aan. Anders doen we verder met de vorige. Dit zou duidelijk moeten zijn aan de hand van figuur 3.1.

Ten slotte wordt er gekeken of het stop criterium bereikt is. Dit kan bijvoorbeeld zijn wanneer de tijd op is.

3.3 Constructieve Heuristiek

Om een oplossing te kunnen aanpassen is er eerst een oplossing nodig. Hiervoor dient een constructieve heuristiek. Het bepalen van een initiële oplossing. Bij het bepalen van een initiële oplossing kunnen we telkens op zoek gaan naar de job sequentie σ . Daarna bepalen we de *tool-job matrix* A^* met KTNS. Om zo tot de initiële oplossing $S = \{\sigma, A^*\}$ te komen.

Volgende zijn te overwegen voor het vinden van een initiële sequentie σ :

- De originele volgorde van de set van jobs behouden.
 - Een sequentie van 0 tot N .
- Een willekeurige volgorde van jobs gebruiken.
 - Een sequentie 0 tot N . Maar in willekeurige volgorde.
- Het probleem voorstellen en oplossen als een instantie van het TSP.
 - Zoals in **hoofdstuk 2, sectie 2.2.4**
- Het probleem oplossen met behulp van tool sequencing.
 - Zoals in **hoofdstuk 2, sectie 2.4.1**

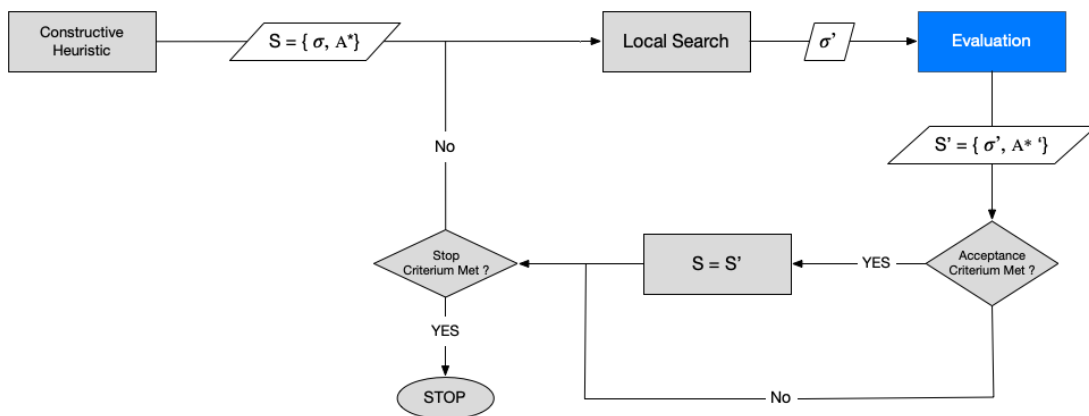
De voorkeur in dit werk ging uit naar de methode van **Paiva Et. al.** [5]. Om een eerlijke vergelijking te maken zijn zowel experimenten uitgevoerd voor *Tool Sequencing*, *Ordered* en *Random*.

3.4 Besluit

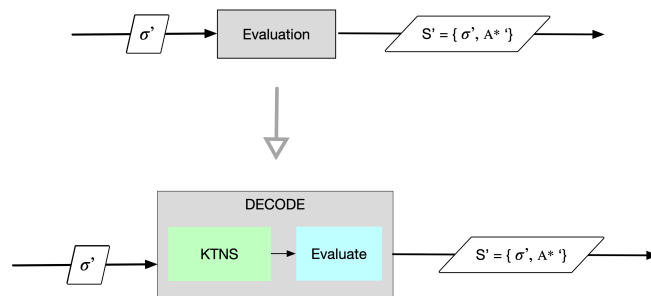
Dit hoofdstuk introduceerde local search en enkele mogelijke constructieve heuristieken. Tijdens het ontwikkelen van de Local Search werden 2 belangrijke peilers zichtbaar. Het ontwikkelen van een ruime gerichte local search en het ontwikkelen van een performante evaluatie. Desalniettemin speelden de andere bouwblokken hier ook natuurlijk een rol in. De volgende hoofdstukken zullen elke bouwblok in detail bespreken. Het volgende hoofdstuk neemt de evaluatiefunctie onder de loep.

Hoofdstuk 4

Evaluatie



4.1 Decoder



Figuur 4.1: Werking decoder

De decoder maakt het mogelijk om een oplossing aan te passen door enkele de job sequentie te veranderen.

De decoder voert telkens twee stappen uit. De eerste stap is een volledige oplossing bepalen. Dit gebeurt met $S = KTNS(\sigma)$, zoals toegelicht in hoofdstuk 2. De tweede stap bepaalt de objectief

waarde van een oplossing. De objectief waarde van een oplossing is een score die uitdrukt hoe goed een oplossing is. Aan de hand van die score wordt met het acceptatiecriterium bepaald of dit de nieuwe huidige oplossing wordt.

Er werden twee mogelijke objectief functies besproken:

- Switches als objectief (zoals in hoofdstuk 1)
- Tie breaking als objectief (zoals in hoofdstuk 2)
 - Als een oplossing geaccepteerd wordt kijken we ook of deze oplossing een verbetering is van de best gevonden oplossing. Als dit het geval is wordt de aangepaste oplossing ook de best gevonden.
Het tie breaking objectief wordt enkel gebruikt om een oplossing al dan niet te accepteren, niet om te kijken of het een verbetering is. Daarvoor gebruiken we steeds switches als objectief.

Omdat de decoder wordt opgeroepen bij elke aanpassing van een oplossing zullen er veel iteraties gebeuren. Uit profiling werd duidelijk dat de decoder de grootste invloed heeft op rekentijd. De implementatie van snelle decoder is daarom van groot belang.

4.2 KTNS

De meeste werken gebruiken het algoritme van **Tang en Denardo** [12] uit 1988 om KTNS uit te voeren. Dit algoritme werkt op een recursieve manier. Tijdens het analyseren van dit algoritme werd het duidelijk dat het eenvoudiger kon.

4.2.1 Algoritme

De werking van het ontwikkelde KTNS algoritme is voorgesteld door middel van pseudocode in [1]. Om de werking duidelijk te maken is er bij elke stap steeds commentaar geplaatst.

Algorithm 1: KTNS

Input: New sequence σ'
Output: New Solution $S' = \{\sigma', A'^*\}$

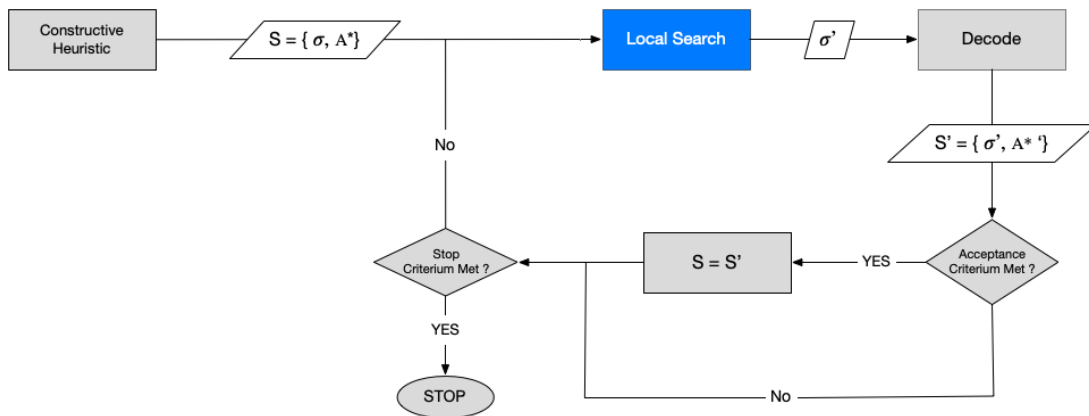
```

1 function KTNS( $\sigma'$ )
    ▷ Zet de oude tool job matrix gelijk aan de nieuwe. De vereiste tools blijven hetzelfde.
2    $A'^* = A^*$ 
    ▷ Verhoog de ktnsId, alle extra aangewezen tools uit de vorige oplossing met de vorige
    ktnsId zullen niet meer als aanwezig beschouwd worden door de evaluatiefunctie.
3    $ktnsId = ktnsId + 1$ 
4   for  $job$  in  $\sigma$  do ▷ Neem elke job in volgorde uit de sequentie
5        $prevJob \leftarrow job - 1$ 
6        $nextJob \leftarrow job + 1$ 
7       if  $job$  not first in  $\sigma$  then
8           ▷ Bepaal hoeveel tools maximaal als extra kunnen aangewezen worden aan deze job.
            $nToolsAdd \leftarrow \max(0, |T_{job}| - C)$ 
           ▷ Kijk welke tools meegenomen worden van de vorige job. Diff bevat de aangewezen
           tools van de vorige job maar niet de vereiste van de huidige. Hierdoor overwegen we
           enkel tools die kunnen blijven zitten uit de vorige job en die potentieel als extra
           kunnen toegevoegd worden. Maak een linkedList aan van deze tools.
9            $diff : LinkedList \leftarrow T_{prevJob}^* \setminus T_{job}$ 
           ▷ Vul de vrije plaatsen in het magazijn op met de snelst nodige tools eerst
10          while  $nToolsAdd > 0$  &  $diff.length > 0$  &  $nextJob < N$  do
11              ▷ Bezoek elk tool in de linkedlist.
              for  $tool \in diff$  do
12                  ▷ Is het tool vereist voor de volgende job.
                  if  $tool \in T_{nextJob}$  then
13                      ▷ Wijs het tool toe aan de huidige job.
                       $A_{tool, job} \leftarrow ktnsId$ 
                      ▷ Verwijder het tool uit de linkedlist. Een linkedList is belangrijker om
                      efficient tools te verwijderen.
14                       $diff.removeTool(tool)$ 
                      ▷ Verminder het aantal toe te voegen tools.
15                       $nToolsAdd \leftarrow nToolsAdd - 1$ 
                      ▷ Ga naar de volgende job als: de huidige job nog plaats heeft voor tools EN
                      alle tools nog niet zijn meegenomen (diff nog niet leeg) EN deze niet de laatste
                      nextJob was. (zie lijn 10)
16                       $nextJob \leftarrow nextJob + 1$ 
              ▷ Vul de overige vrije plaatsen met om het even welk tool uit diff. Dit is het geval
              bij de laatste job of als de volgende jobs de tools uit diff niet bevatten.
17          for  $tool \in diff$  do
18               $A_{tool, job} \leftarrow ktnsId$ 
19               $diff.removeTool(tool)$ 
20               $nToolsAdd \leftarrow nToolsAdd - 1$ 
21              if  $nToolsAdd == 0$  &  $diff.length == 0$  then
22                  break

```

Hoofdstuk 5

Local Search



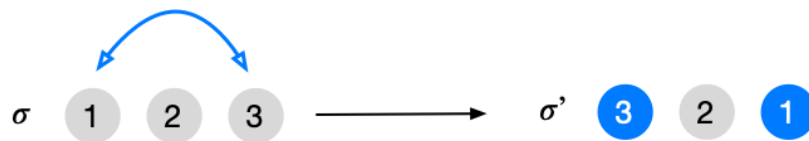
5.1 Introductie

Dit hoofdstuk zal de verschillende Local Search operatoren behandelen die ontwikkeld zijn. De aanzet was om te beginnen bij simpele operatoren om inzicht te krijgen in het probleem. Door tussentijdse resultaten werd het duidelijk dat deze strategie onvoldoende was, een nieuwe methode werd daarom ontwikkeld.

5.2 Swaps

Door de indirecte representatie van het probleem zal de zoektocht steeds gebeuren op de sequentie σ . Dit wilt zeggen dat de oplossing aangepast wordt door de sequentie aan te passen. Nadien wordt de nieuwe sequentie gedecodeerd waardoor een nieuwe oplossing wordt gecreëerd. Dit werd reeds geïllustreerd in hoofdstuk 4 dat evaluatie behandelde.

Een simpele maar intuïtieve manier om een sequentie aan te passen is door gebruik te maken van **job swaps**. Bij een job swap worden twee jobs willekeurig geselecteerd en met elkaar verwisseld. Figuur 5.1 stelt een job swap voor.



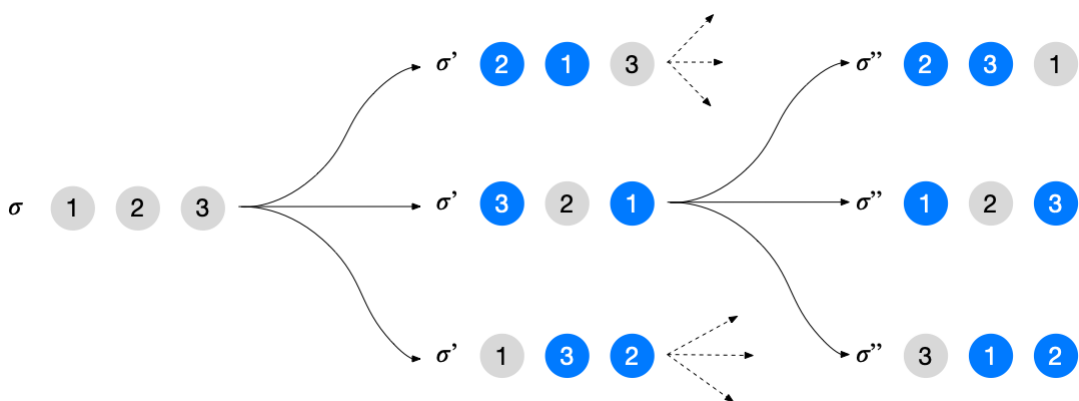
Figuur 5.1: Voorbeeld van een job swap. Op een kleine sequentie

5.2.1 Algoritme

Het volgende algoritme wordt gebruikt om de swaps uit te voeren.

- Kies twee willekeurige jobs uit de huidige sequentie.
- Wissel de twee gekozen jobs om.

5.2.2 Local search omgeving



Figuur 5.2: Omgeving job swaps op een kleine sequentie

De oplossingenruimte is de verzameling van alle mogelijke oplossingen voor een probleem. Als een oplossing wordt aangepast door een local search operator zijn er maar een beperkt aantal

oplossingen bereikbaar met één operatie. De verzameling van oplossingen die bereikbaar zijn noemt de *omgeving* van een oplossing.

In het geval van het SSP gebeurt de operatie op de sequentie. De omgeving van een sequentie $N(\sigma)$ is een verzameling van andere sequenties.

Job swaps maken het mogelijk om de omgeving eenvoudig te illustreren. In figuur 5.2 zijn de sequenties σ' : $\{2, 1, 3\}$, $\{3, 2, 1\}$ en $\{1, 3, 2\}$ bereikbaar met één job swap vanuit sequentie σ . De sequenties σ' zijn de omgeving van σ of $N(\sigma)$. Andere delen van de oplossingenruimte zijn enkel bereikbaar na meerdere operaties. Bijvoorbeeld: $\sigma'' = \{3, 1, 2\}$ is bereikbaar na een swap van job 3 en 1, gevolgd door een swap van job 1 en 2.

Aan figuur 5.2 is ook zichtbaar dat je terug bij dezelfde oplossing terecht kan komen. Bijvoorbeeld: door een swap van job 3 en 1 en vervolgens weer een swap van job 3 en 1.

5.2.3 Analyse

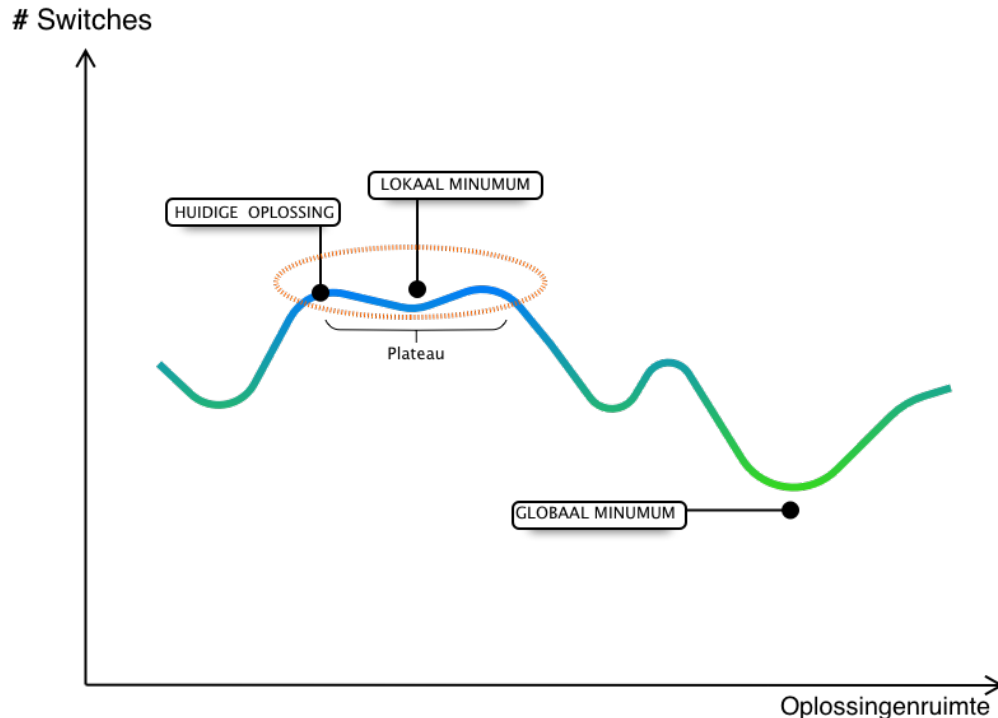
Het voordeel aan swaps is:

- Ze zijn simpel en bovendien snel.
- Ze kunnen de volledige oplossingenruimte verkennen.
- Er moet geen rekening gehouden worden met de staat van de huidige oplossing zoals de *tool-job matrix* A^*
- Ze tonen geen voorkeur voor bepaalde oplossingen.

Tijdens tussentijdse experimenten werd echter duidelijk dat er nadelen zijn:

Swaps verkennen een kleine omgeving.

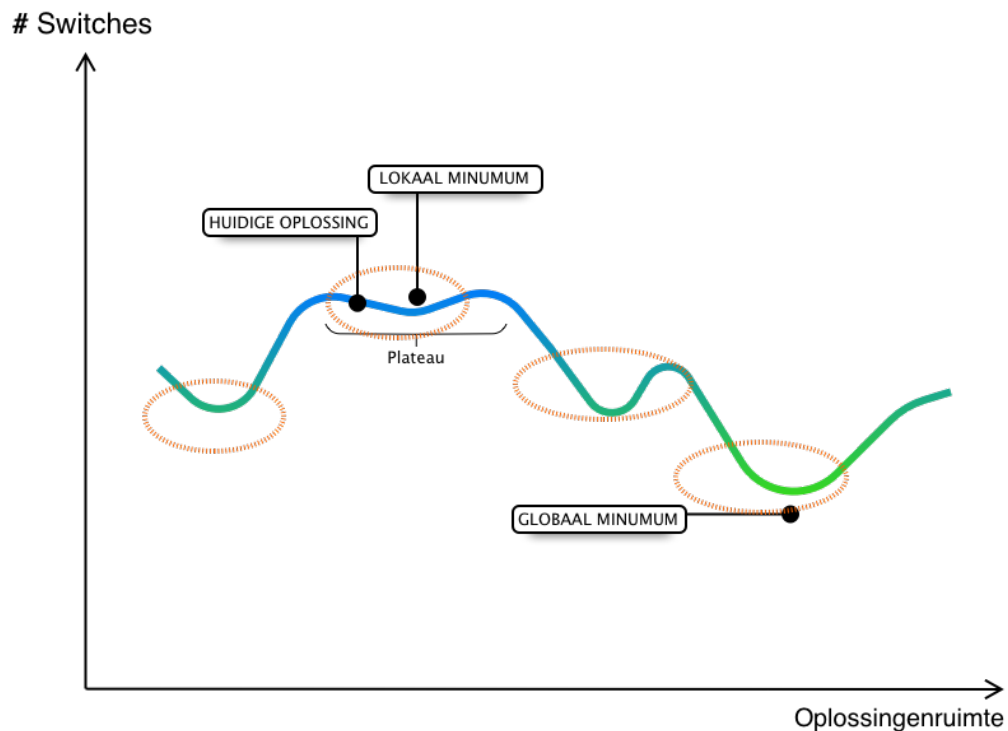
Figuur 6.1 stelt het zoeklandschap voor van een probleem. De y-as stelt het aantal switches voor. Terwijl de x-as de oplossingenruimte voorstelt. De curve geeft hier elke oplossing aan binnen het zoeklandschap. Slechte oplossingen hebben veel switches, waardoor de curve hoog ligt. Goede oplossingen hebben weinig switches, waardoor de curve laag ligt.



Figuur 5.3: Zoeklandschap: omgeving met swaps (oranje stippelijn)

Swaps werken erg lokaal, oplossingen die zich in de omgeving bevinden hebben maar een klein verschil met de huidige oplossing. Dit is zichtbaar op figuur 5.3, oranje stippelijijn. Dit is problematisch als de oplossing zich bevindt in een slechte omgeving. In figuur 5.3 zijn er heel wat goede swaps nodig om te geraken bij de lager gelegen gebieden.

Een operator die een grotere en diversere omgeving kan bereiken heeft meer kans om een goede oplossing te vinden. Figuur ?? illustreert een operator die hierin slaagt.



Figuur 5.4: Zoeklandschap: gewenste omgeving (oranje stippelijijn)

Oscillarence toestanden zijn mogelijk.

Eenmaal een goede oplossing gevonden is het mogelijk dat we tussen 2 dezelfde oplossingen blijven hangen. Dit werd vooral duidelijk tijdens tussentijdse experimenten. Dit is een gevolg van de kleine omgeving die bereikbaar is met swaps.

Een goede oplossing wordt vernietigd.

Door louter willekeurig swaps te doen is het mogelijk dat een goede oplossing wordt vernietigd. Job swaps houden enkel rekening met de sequentie σ en houden geen rekening met de volledige oplossing $S = \{\sigma, A^*\}$. Het aantal switches is afhankelijk van de volledige *tool-job matrix*.

Deze bovenvermelde redenen waren de motivatie om te zoeken naar een nieuwe operator die een ruime omgeving verkent en die doelgericht werkt.

5.3 Ruin en recreate

Ruin en recreate is een local search operator die bestaat uit 2 fases. Een ruin fase $\mathcal{R}^-$ die de oplossing gedeeltelijk vernietigt en een recreate fase $\mathcal{R}^+$ die de oplossing terug opbouwt. De combinatie van verschillende ruin en recreate operatoren is mogelijk.

In het geval van operaties op de job sequentie σ is dit:

- Jobs uit σ halen.
- De uitgehaalde jobs op een bepaalde manier terug in σ plaatsen.

De inspiratiebron om ruin en recreate te gebruiken binnen dit werk is afkomstig uit het artikel **Slack Induction by String Removals for Vehicle Routing Problems (SISRs)** door **Christiaens en Vanden Berghe**[\[2\]](#). Hun oplossing strategie werd toegepast op dit probleem. Een methode is ontwikkeld die geschikt is voor het SSP.

5.3.1 Redenering

Premisse 1: Overweeg gerelateerde jobs samen De "dominance property" werd in hoofdstuk 1, sectie 2.2.5 voorgesteld. Dit leidde tot het inzicht dat gerelateerde jobs een positieve invloed kunnen hebben op elkaar. Jobs die één of meerdere tools gemeenschappelijk hebben plaatst men best in relatie tot mekaar. Zodat KTNS beter werkt en er minder switches zijn.

Premisse 2: Haal genoeg jobs uit de sequentie In sectie 5.2.4 werd een analyse gedaan van de omgeving met swaps. Een limitatie was het verkennen van een kleine omgeving.

Door genoeg jobs uit een sequentie te halen ontstaan er vele mogelijkheden om een nieuwe te vormen. Dit moet wel op slimme manier gebeuren. Een oplossing kappot maken kan ook het verknoeien betekenen van een goede. Daarom is het belangrijk om bij het verwijderen van jobs te zorgen dat deze jobs gerelateerd zijn.

Premisse 3: Verminder het aantal zero blocks In sectie 2.2.3 werden 0-blocks toegelicht. Een optimale oplossing op het SSP heeft geen 0-blocks. Dus tijdens het oplossen willen we het aantal 0-blocks verminderen. Een operator die dit doelgericht doet is hiervoor interessant.

5.3.2 Methode: SMRI

Een systematische aanpak werd ontworpen bij het uitvoeren van Ruin en Recreate. De werking van de ontwikkelde methode wordt stap per stap geïllustreerd aan de hand van een visueel voorbeeld op de volgende pagina.

De visuele manier is net dezelfde manier waarop de methode is ontwikkeld. Door problemen te visualiseren werden bepaalde operaties zichtbaar die nuttig zouden zijn. Het idee is puur creatief en experimenteel tot stand gekomen. Tijdens het ontwerp werd rekening gehouden met de bovenvermelde premissen en met de methode van **SISRs**^[2].

De methode krijgt de naam **SMRI**, wat staat voor Select, Match, Remove, Insert.

5.3.3 Werking

START

Stel de huidige oplossing $S = \{\sigma, A^*\}$ voor door de matrix P^*

	2	0	3	1
0	○	●	●	●
1	●	●	●	●
2	●	○	○	●
3	●	●	●	○

Switches 0 1 0 1

FASE 1
SELECT

Selecteer een willekeurig tool.

	2	0	3	1
0	○	●	●	●
1	●	●	●	●
2	●	○	○	●
3	●	●	●	○

Switches 0 1 0 1

FASE 2
MATCH

Match jobs dat het geselecteerde tool vereist nodig hebben.

	2	0	3	1
0	○	●	●	●
1	●	●	●	●
2	●	○	○	●
3	●	●	●	○

Switches 0 1 0 1

Job kandidaten 2 1

FASE 3
REMOVE

Verwijder de jobs.

	2	0	3	1
0	○	●	●	●
1	●	●	●	●
2	○	○	○	○
3	○	○	○	○

Verwijderde jobs 2 1

FASE 4
INSERT

Voeg de jobs toe in de beste posities.

	2	0	3	1
0	○	●	●	●
1	●	●	●	●
2	○	○	○	○
3	○	○	○	○

INSERT POSITIES

EINDE
 $\text{decode}(\sigma')$

Neem de nieuwe sequentie en decode naar een nieuwe oplossing

$\sigma' = 1 \ 2 \ 0 \ 3$ → $\text{decode}(\sigma')$

	1	2	0	3
0	○	○	●	●
1	●	●	●	●
2	●	○	○	○
3	○	○	○	○

$S' = \{\sigma', A^*\}$

Figuur 5.5: SMRI: adhv voorbeeld 1

5.3.4 Samenvatting

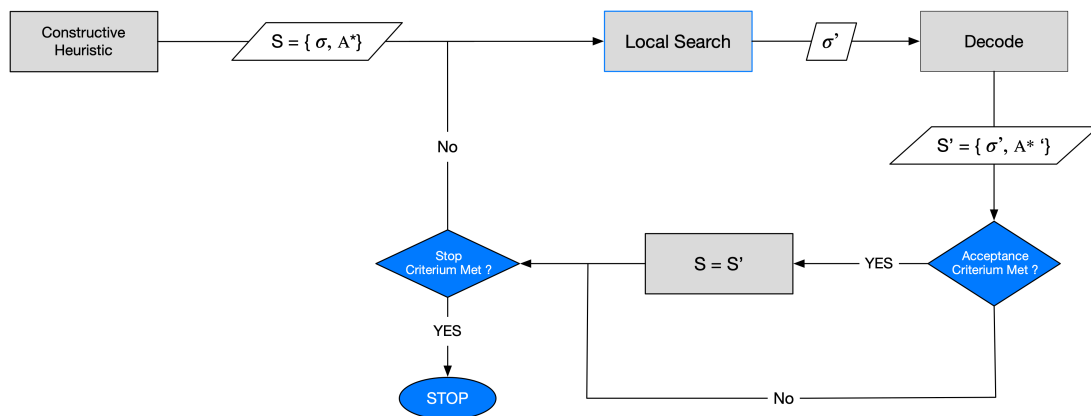
Verschillende variaties werden ontwikkeld bij elke fase. Deze zijn ontwikkeld op basis van de eigenschappen die het SSP bezit. Alle variaties zijn aan de hand van onderstaande tabel toegelicht. Verschillende variaties zijn combineerbaar met elkaar.

Methode	Uitleg
Select fase	
select random	Selecteer één willekeurig tool
select multi random	Selecteer tussen $[1, M[$ verschillende tools random tools.
Match fase	
match vereist tool	Match de jobs die het geselecteerde tool vereist nodig hebben.
match KTNS tool	Match de jobs die het geselecteerde tool aangewezen kregen door KTNS
match 1-block	Match de jobs die betrokken zijn in een 1-block voor het geselecteerde tool.
match 0-block	Match de jobs die betrokken zijn in een 0-block voor het geselecteerde tool.
$\mathcal{R}^-$ Remove fase	
random matches	Verwijder <i>ruinSize</i> aantal jobs die een match hebben, willekeurig, uit de sequentie.
worst matches	Verwijder <i>ruinSize</i> aantal jobs die een match hebben, volgens een worst policy (de jobs die het meeste switches veroorzaken).
maximal matches	Enkel toepasbaar bij select multi. Verwijder de jobs met het maximum aantal matches. Bijvoorbeeld: als job 1 en 2 , 4 matches heeft en job 3 , 1 match dan wordt enkel job 1 en job 2 verwijderd. Job 1 en job 2 hebben het maximaal aantal matches in dit geval. Deze methode zorgt ervoor dat meer gelijkaardige jobs er uit worden gehaald, en clustert als het ware jobs bij elkaar tijdens de zoektocht. Afhangend van het aantal tools dat geselecteerd is in de select fase kan deze clustering sterk of zwak zijn. Bij bijvoorbeeld veel tools zullen de jobs sterk gelijkaardig zijn.
$\mathcal{R}^+$ Insert fase	
Best insert with blinks	Insert de verwijderde jobs terug in de sequentie, pas best insert met blinks toe. Deze methode is dezelfde insert methode die toegepast wordt in SISR's ^[2]

Tabel 5.1: SMRI samenvatting

Hoofdstuk 6

Metaheuristiek



6.1 Introductie

Dit hoofdstuk bespreekt de metaheuristieken die onderzocht zijn voor dit probleem. Een metaheuristiek zal Local Search gaan leiden doorheen het zoeklandschap. Dit gebeurt door het iteratief verbeteren van een oplossing. Een kandidaat oplossing wordt al dan niet geaccepteerd aan de hand van een acceptatiecriterium.

6.2 Steepest Descent

Steepest Descent neemt de beste oplossing in de omgeving van de huidige oplossing. Dit wordt herhaalt tot de oplossing stopt met verbeteren. Deze methode valt goed te combineren met job swaps. De omgeving van een oplossing met job swaps is klein genoeg dat deze volledig verkenbaar is. ¹

6.2.1 Algoritme

- Tijdens de local search: neem de beste oplossing uit de omgeving $N(S)$
- Het acceptatie criterium is: accepteer een betere oplossing dan de huidige.
- Het stopcriterium is: als de oplossing niet meer verbetert dan stopt het zoeken.

Algoritme ² geeft de implementatie van Steepest Descent weer.

Let op: als een oplossing even goed is dan een reeds gevonden oplossing uit de omgeving dan wordt één willekeurig gekozen als de huidige oplossing.

Algorithm 2: Steepest Descent

Input: Initial solution $S = \{\sigma, A^*\}$
Output: Best solution S_{best}

```

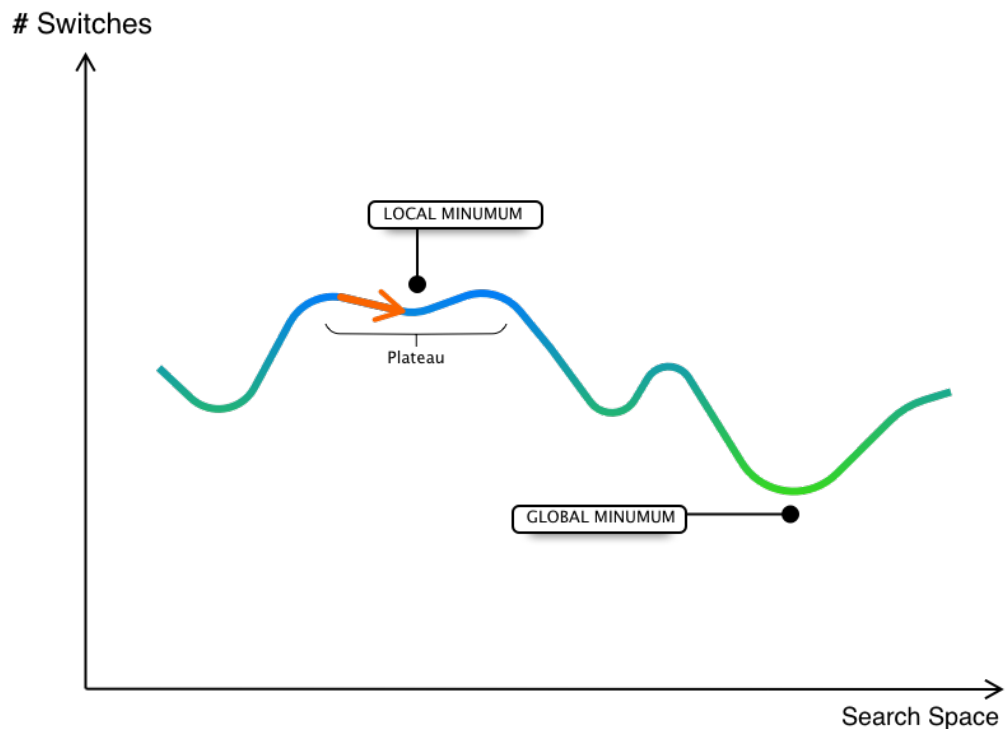
1 function Steepest Descent( $S$ )
2   while  $S < S_{best}$  or first iteration do
3     for  $\forall S' \in N(S)$  do Bezoek alle oplossingen in de omgeving van de huidige oplossing
4        $numberOfBestSolutions \leftarrow 1$ 
5       if  $cost(S') \leq cost(S)$  then
6          $S = S'$ 
7          $numberOfBestSolutions \leftarrow 1$ 
8       else if  $cost(S') == cost(S)$  then
9          $numberOfBestSolutions \leftarrow numberOfBestSolutions + 1$ 
10         $probability \leftarrow \frac{1}{numberOfBestSolutions}$ 
11        if  $random(0, 1) \leq probability$  then
12           $S_{best} \leftarrow S'$ 
13           $numberOfBestSolutions \leftarrow 1$ 
14  return  $S_{best}$ 

```

¹ De betekenis van een omgeving van een oplossing $N(s)$ werd reeds in hoofdstuk 5 besproken.

6.2.2 Analyse

Een voordeel dat Steepest Descent biedt is dat er snel een **lokaal** minimum gevonden wordt. Dit is ook nadelig, als een initiële oplossing een slechte omgeving heeft, dan is het moeilijk om deze te ontsnappen.



Figuur 6.1: Oplossingenruimte: probleem steepest descent

Figuur 6.1, stelt het imaginair zoeklandschap voor van een probleem. De y-as stelt het aantal switches voor. Terwijl de x-as het zoeklandschap voorstelt. De curve geeft hier elke oplossing aan binnen het zoeklandschap. Slechte oplossingen hebben veel switches (de curve ligt hoog). Goede oplossingen hebben weinig switches (de curve ligt laag).

Stel dat huidige oplossing zich bevindt op een plateau, een eigenschap die dit probleem vertoont [4]. Op de figuur 6.1 is dit bijvoorbeeld aan het begin van de oranje pijl.

Bij het uitvoeren van Steepest Descent zal de oplossing steeds verbeteren. De pijl duidt dus de verplaatsing aan van de huidige oplossing naar een betere. In de put van het plateau zal Steepest Descent stoppen, omdat enkel verbeterende oplossingen aanvaardbaar zijn. De switches gaan rechts en links van de put weer omhoog.

De oplossing eindigt in een lokaal minimum. Het globaal minimum ligt lager. Indien steepest descent niet verbeterende oplossingen zou overwegen, waren de betere oplossingen misschien wel bereikbaar geweest.

Ten slotte is Steepest Descent niet geschikt voor de nieuwe Ruin en Recreate methode. Het feit dat men steeds de beste oplossing neemt uit de volledige omgeving is hier de belemmerde factor. Eén van de eigenschappen van de ontwikkelde ruin & recreate is dat deze een grote omgeving kan verkennen. Bij elke oplossing de volledige omgeving gaan verkennen is moeilijk. Bovendien zou het te veel rekentijd vragen.

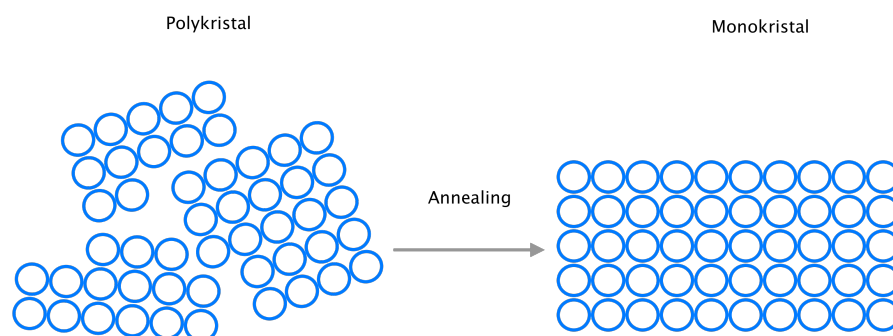
Het is aangewezen om één Ruin en Recreate move te doen en dit steeds te herhalen. Een methode die dit mogelijk maakt is **Simulated Annealing**. Simulated Annealing zal de oplossing ook leiden naar betere oplossingen net zoals **Steepest Descent**.

6.3 Simulated Annealing

6.3.1 Introductie

Kirkpatrick Et. Al. [13] introduceerden Simulated Annealing, een meta heuristiek gebaseerd op het statische model van Annealing.

Annealing of gloeien is een process dat in de metaal industrie gebruikt wordt. Het tracht hierbij onregelmatigheden uit staal te krijgen.



Figuur 6.2: Annealing: van een polykristal naar een monokristal [1]

Figuur 6.2 illustreert annealing. Het doel is om zo weinig mogelijk interne spanningen in een materiaal te hebben. Door een materiaal traag af te koelen kan het van een slechte ongeordende toestand naar een goede geordende toestand geraken.

Het simulated annealing algoritme start bij een initiële temperatuur T_0 en eindigt bij een temperatuur T_1 . Om de temperatuur te laten dalen maakt het algoritme gebruik van een *koel schema*.

Tijdens de zoektocht naar een oplossing accepteert Simulated Annealing zowel verbeterende als niet verbeterende oplossingen. Door dit gedrag kan SA lokale optima ontsnappen. Dit gebeurt aan de hand van een acceptatiecriterium.

In het geval van SA is dit een probaliteitsfunctie: de kans dat een slechte oplossing geaccepteerd wordt, daalt naarmate de temperatuur daalt.

De systematische aanpak door Christiaens en Vanden Berghe [2] uit **Slack Induction by String Removals for Vehicle Routing Problems** is toegepast voor de implementatie van Simulated Annealing. Aanpassingen werden aangebracht waar nodig voor de toepassing op dit probleem.

6.3.2 Parameters

De afkoeling van het systeem gebeurt met een exponentieel koeling schema, zoals in vergelijking [6.1](#):

$$\mathcal{T}_{k+1} = \mathcal{T}_k \cdot c \quad 0 < k < f \quad (6.1)$$

De koeling constante wordt bepaald zoals in vergelijking [6.2](#):

$$c = \left(\frac{\mathcal{T}_1}{\mathcal{T}_0} \right)^{\frac{1}{f}} \quad (6.2)$$

Waarbij $\mathcal{T}_0$ de start temperatuur, $\mathcal{T}_1$ de eind temperatuur, en $\mathcal{T}_0 > \mathcal{T}_1$.

6.3.2.1 Iteraties & probleem grootte

Het aantal iteraties f wordt in vergelijking [6.3](#) bepaald aan de hand van de probleem grootte v .

$$f = it(v) \quad (6.3)$$

$$(6.4)$$

De probleem grootte v is een functie van het aantal jobs N , aantal tools M en de magazine capaciteit C . Het is een getal dat de grootte en moeilijkheid van een probleem voorstelt. Vergelijking [6.5](#) geeft dit weer.

$$v = problemSize(N, M, C) \quad (6.5)$$

$$= \alpha \cdot N + \beta \cdot M + \gamma \cdot C \quad (6.6)$$

De parameters α, β, γ zijn gewichten die bepalen welk effect het aantal switches N , aantal tools M en de magazine capaciteit C hebben op v .

De parameters zijn empirisch bepaald door onderzoek naar resultaten van vorige werken. Hierbij zijn de run times van de verschillende instanties vergeleken om zo een betekenis te geven aan de invloed van N , M en C op run time. Tabel [6.1](#) geeft de gekozen waarden weer en waarom.

Parameter	Waarde	Verklaring
α	1	Het aantal jobs speelt een grote rol.
β	0.7	Het aantal tools speelt een even grote rol maar is meestal meer dan het aantal jobs. Door ze maar voor 70% te laten meetellen zijn deze ongeveer gelijk in de meeste gevallen.
γ	-1	Hoe groter de magazine grootte, hoe makkelijker oplosbaar het probleem. Dit is omdat meer tools in het magazine kunnen blijven dankzij KTNS. Wat het aantal switches verminderd. De magazine capaciteit is omgekeerd evenredig met het aantal jobs en tools. Daarom is het negatief. Door het feit dat de magazine capaciteit vaak veel kleiner is dan het aantal jobs en aantal tools verkleinen we deze niet.

Tabel 6.1: Parameters voor het bepalen van probleem grootte v

De empirische strategie is niet de beste manier om parameters te bepalen. Parameter tuning zou hierbij een beter aanpak zijn. Door bijvoorbeeld gebruik te maken van *IRACE* [6].

De minimale probleem grootte v_{min} wordt bepaald volgens vergelijking [6.7] en de maximale probleem grootte wordt bepaald volgens vergelijking [6.8]. De ingevulde getallen zijn het aantal jobs, aantal tools en de magazijn-capaciteit van de kleinste en grootste instanties die gebruikt zijn in dit werk.

$$v_{max} = \text{problemSize}(70, 105, 40) \quad (6.7)$$

$$v_{min} = \text{problemSize}(8, 10, 10) \quad (6.8)$$

Het maximaal aantal iteraties is $it(v_{max})$ en het minimaal aantal iteraties is $it(v_{min})$, waarbij $it(v_{min}) < it(v_{max})$.

Ten slotte wordt lineaire interpolatie toegepast in vergelijking [6.9] voor het aantal iteraties te vinden. Hierbij wordt de probleem grootte als parameter gebruikt.

$$it(v) = it(v_{min}) + \frac{it(v_{max}) - it(v_{min})}{v_{max} - v_{min}} \cdot (v - v_{min}) \quad (6.9)$$

Door dit toe te passen zal het aantal iteraties liggen tussen $it(v_{min})$ en $it(v_{max})$, afhankelijk van de probleemgrootte v . Hoe moeilijker en groter het probleem hoe meer iteraties er nodig zijn.

6.3.3 Algoritme

Algorithm 3: Simulated Annealing

Input: Initial solution $S = \{\sigma, A^*\}$
Input: Best solution $S_{best} = \{\sigma, A^*\}$

```

1 function SimulatedAnnealing( $S$ )
2   for  $f$  iterations do
3      $\sigma' = \text{ruinAndRecreate}(S)$ 
4      $S' = \text{decode}(\sigma')$ 
5     if  $\text{cost}(S) < \text{cost}(S') - T \ln(\text{random}(0, 1))$  then  $\triangleright$  Accepteer een oplossing met een
        zeker probabiliteit
6        $S \leftarrow S'$ 
7     if  $n\text{Switches}(S') < n\text{Switches}(S_{best})$  then  $\triangleright$  Maak een oplossing de nieuwe beste als
        deze minder switches heeft dan de huidige.
8        $S_{best} \leftarrow S'$ 
9     if  $S' == S_{best}$ , for  $n$  iterations then  $\triangleright$  Stop als de oplossing niet meer veranderd.
10      return  $S_{best}$ 
11     $T \leftarrow T \cdot c$ 
12 return  $S_{best}$ 
  
```

6.3.4 Motivering

Enkele argumenten waarom de keuze ging naar Simulated Annealing:

- Simulated Annealing bezit de vaardigheid om aan lokale optima te ontsnappen.
- Er zijn (relatief) weinig parameters om te tunen.
- Het lijkt een goede partner voor de ontwikkelde Ruin en Recreate.
 - Het artikel **Slack Induction by String Removals for Vehicle Routing Problems** was de inspiratiebron voor de ontwikkelde Ruin en Recreate. **Christiaens en Vanden Berghe**^[2] passen hierbij Simulated Annealing toe. Het behalen van State Of The Art resultaten was hier een motiverende factor om te kiezen voor Simulated Annealing.

6.4 Samenvatting

Dit hoofdstuk onderzoekt Steepest Descent en Simulated annealing als meta heuristieken.

Steepest Descent werd initieel gekozen om vertrouwd te raken met het probleem. Deze is eenvoudig en tevens goed te combineren met job swaps.

Het werd echter duidelijk dat Steepest Descent beperkt is. Ten eerste was het vastlopen in een lokale minima ongewenst gedrag. Ten tweede was het niet de ideale partner voor Ruin en Recreate.

Daarom werd Simulated Annealing als alternatief onderzocht. Simulated Annealing vermijdt lokale optima en is goed combineerbaar met moves die een grote omgeving verkennen.

Het volgende hoofdstuk bespreekt de computationele resultaten die behaald zijn. De eerste fase vergelijkt de verschillende voorgestelde strategieën. Ten slotte zal de tweede fase de beste strategie vergelijken met de state of the art.

Hoofdstuk 7

Resultaten

7.1 Introductie

Voor het SSP bestaan er geen officiële benchmark instanties. Dit maakt het moeilijk om verschillende methoden te gaan vergelijken. Vele werken genereerden hun eigen instanties random. Hierin variëren ze het aantal jobs, tools en magazijn-capaciteit.

Gelukkig worden de laatste jaren enkele beschikbare data sets meer en meer beschouwd als benchmark instanties. Problemen die zich positioneren als state of the art maken vaak gebruik van één of meerdere van deze sets. In dit werk is er gekozen om van deze gebruik te maken. Zo is een vergelijking mogelijk met andere werken.¹

Auteur	# instanties	N		M		C	
		Min	Max	Min	Max	Min	Max
Crama Et. al. ^[10]	160	10	40	10	60	4	30
Yanasse Et. al. ^[8]	1350	8	25	15	25	4	20
Catanzaro Et. al. ^[7]	160	10	40	10	60	4	30

Tabel 7.1: Overzicht instanties

In totaal zijn er 1670 instanties. Elke dataset groepeert instanties. Een groep zijn instanties met hetzelfde aantal jobs, tools en magazine size. Binnen een groep zijn verschillende variaties aanwezig. Elke variatie heeft een andere *tool-job matrix* en is uniek. Resultaten worden weergegeven door het gemiddeld aantal job switches binnen een groep te berekenen.

¹De resultaten zijn niet final. Een revisie van de decoder vind nog plaats alsook aanpassingen aan SMRI. Een bijlage wordt beschikbaar gesteld tijdens de presentatie, deze zal de nieuwe resultaten voorleggen. De datasets van Yanasse en Crama zijn hier nog niet weergegeven bij de state of the art vergelijking.

7.2 Oplossingsmethode

Meer dan 30 GB aan data² is tijdens het experimenteren gegenereerd geweest met deze datasets. Het zou overdreven zijn om hier elk experiment te gaan uitleggen. Daarom is gekozen om de belangrijkste experimenten naar voren te brengen, deze zijn bewijzen waarom een bepaalde methode is gebruikt in de gekozen oplossingsmethode van dit probleem. Wanneer een bepaald onderdeel van het algoritme onderzocht werd zijn de andere parameters en methoden constant gehouden. Dit om het verschil zo eerlijk mogelijk weer te geven. Bij elke variatie gebeurden 3 runs. In de tabellen wordt telkens het beste aantal switches $\mathcal{S}_{best}$, gemiddeld aantal switches $\mathcal{S}_{avg}$ en de tijd t weergegeven. Ten slotte zijn de resultaten hieronder op een selectie van instanties. De selectie bestaat uit zowel kleine als grote instanties.

7.2.1 Constructieve heuristiek

Voor de constructieve heuristiek werden 3 variaties geïmplementeerd:

1. Geordende set van jobs (ORD)
2. Random permutatie van de set van jobs (RAN)
3. Tool sequencing (TSEQ)

Auteur	N	M	C	i	ORD	RAN	TSEQ
yanasse	8	15	10	30	7.93	7.8	7.47
crama	30	40	25	10	52.8	52.6	40.1
catanzaro	15	20	12	10	9.9	9.2	7.7
catanzaro	30	40	17	10	99.8	98.5	88.3
catanzaro	40	60	30	10	143.6	143.6	119.9

Het is onmiddellijk zichtbaar dat de winnende constructieve heuristiek hier Tool Sequencing is.

²Om de vele experimenten mogelijk te maken was een volledige automatisering van experimenten vereist. Een volledig framework werd ontwikkeld om grote batch experimenten mogelijk te maken. De resultaten waren nadien beschikbaar in een dashboard. Het dashboard maakt visualisatie mogelijk van logs mogelijk in grafieken. Visualisatie is ook mogelijk van tussentijdse oplossingen. In bijlage C is er een schematische weergave van dit framework en enkele screenshots

7.2.2 Meta heuristiek

7.2.2.1 Simulated Annealing temperaturen

Simulated annealing begint bij een start temperatuur $\mathcal{T}_0$ en eindigt bij een eind temperatuur $\mathcal{T}_1$. Voor tuning werden dezelfde parameters gebruikt als in **SISRs** door **Christiaens en Vanden Berghe** [2].

$$\mathcal{T}_0 = 100 \quad (7.1)$$

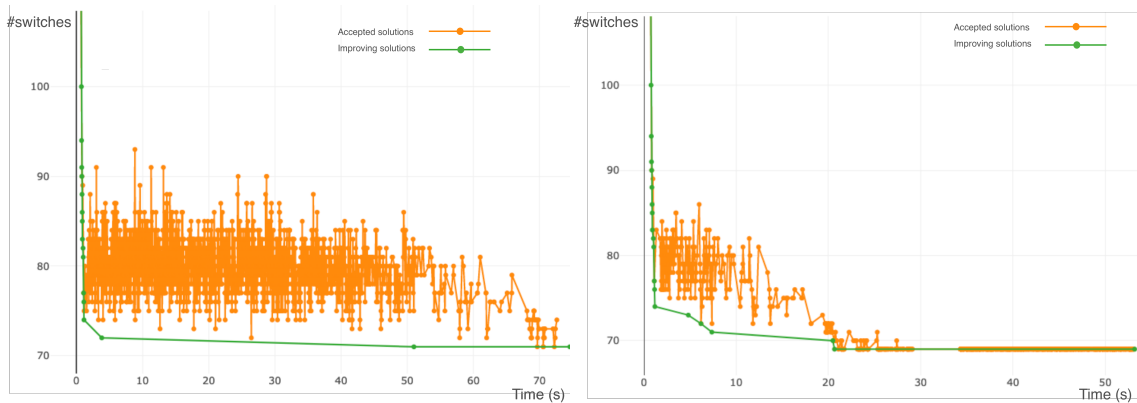
$$\mathcal{T}_1 = 1 \quad (7.2)$$

Daarna zijn er 16 batch experimenten opgezet met variërende temperaturen. Vergelijking 7.3 geeft de temperaturen weer met de beste resultaten.

$$\mathcal{T}_0 = 10 \quad (7.3)$$

$$\mathcal{T}_1 = 0.001 \quad (7.4)$$

Figuur 7.1 vergelijkt het verloop van de kost functie voor het tunen en na het tunen. De x-as is de tijd, de y-as is het aantal switches. De oranje curve zijn de geaccepteerde oplossingen door Simulated Annealing. De groene curve zijn de verbeterende oplossingen. Met de nieuwe parameters lijkt het erop dat er nog plateaus doorbroken worden.



(a) Voor tuning: $\mathcal{T}_0 = 100, \mathcal{T}_1 = 1$

(b) Na tuning: $\mathcal{T}_0 = 10, \mathcal{T}_1 = 0.001$

Figuur 7.1: Kost functie voor: Catanzaro instantie: $N = 30, M = 40, C = 17, i = 1$

De batch experimenten zijn niet de beste manier om parameters te bepalen. Paramater tuning zou hierbij een betere aanpak zijn. Door bijvoorbeeld gebruik te maken van *IRACE* [6].

7.2.2.2 Steepest Descent vs Simulated Annealing

De volgende resultaten vergelijken Steepest Descent en Simulated Annealing. Job swaps zijn gebruikt als local search operator. Alle andere parameters blijven hetzelfde.

author	N	M	C	i	SD	SA
yanasse	8	15	10	30	7.8	6.83
crama	30	40	25	10	52.6	40.5
catanzaro	15	20	12	10	9.2	7.2
catanzaro	30	40	17	10	98.5	77.9
catanzaro	40	60	30	10	143.6	119.3

Simulated annealing vertoont betere resultaten.

7.3 Local search operator

7.3.1 Ruin en recreate configuratie

Vele configuraties zijn mogelijk bij *SMRI*. Batch experimenten werden gebruikt om de meest passende strategieën te vinden. Het is overdreven om elke mogelijke configuratie weer te geven.

Daarom worden de drie meest interessante configuraties weergegeven, zie tabel [7.2](#)

	SMRI-A	SMRI-B	SMRI-C
SELECT	random tool	multi random tool	random tool
MATCH	vereist tool	vereist tool	0-block
REMOVE	random	maximal matches	random
INSERT	best insert + blinks	best insert + blinks	best insert + blinks

Tabel 7.2: Beste SMRI configuraties

Het meest verassende was dat matches op een vereist tool en op zero-blocks significant beter scoren dan andere. Het vermoeden is dat dit komt door volgende redenen:

- Het matchen van jobs met zero-blocks neemt de oorzaak van een tool switch direct aan, namelijk de jobs die dit veroorzaken. Zoals toegelicht in hoofdstuk 2, sectie 2.2.3
- Het matchen van jobs op een vereist tool, associeert soortgelijke jobs met elkaar, deze jobs bevinden zich als het ware dicht bij elkaar, het creëert ruimte om ze goed te plaatsen in relatie tot mekaar.

7.3.1.1 Combinatie SMRI-B en SMRI-C

De overweging is gemaakt of het interessant zou zijn om meerdere configuraties samen te gebruiken. Hierbij was een combinatie van SMRI-B en SMRI-C, de beste kandidaat uit een reeks van batch experimenten, deze noemen we SMRI-BC. De combinatie gebeurt met "roulette wheel

selection”. Bij elke iteratie wordt een willekeurige configuratie gekozen, waarbij bepaalde configuraties een grotere kans hebben om gekozen te worden dan andere. De volgende kansen vertonen de beste resultaten(tabel 7.3):

SMRI-B	SMRI-C
70%	30%

Tabel 7.3: Roulette wheel selection: kansen

7.3.2 Vergelijking job swaps vs SMRI

De volgende resultaten vergelijken:

- SMRI-A
- Job swaps

author	N	M	C	i	SWAP	SMRI
yanasse	8	15	10	30	6.83	6.83
crama	30	40	25	10	43.7	40.5
catanzaro	15	20	12	10	7.2	7.2
catanzaro	30	40	17	10	90.2	77.9
catanzaro	40	60	30	10	131.1	119.3

Het is duidelijk dat de ontwikkelde SMRI methode beter werkt dan job swaps.

7.4 State of the art

Het volgende deel vergelijkt onze implementatie met state of the art resultaten van het SSP. Tabel 7.5 geeft de werken weer waarmee dit werk vergeleken wordt.

Titel	Auteur	Jaar	Methode
Improved heuristic algorithms for the Job Sequencing and Tool Switching Problem	Paiva et al. [5]	2017	Iterated Local Search (ILS)
A Simple and Effective Hybrid Genetic Search for the Job Sequencing and Tool Switching Problem	Mecler et al. [4]	2019	Hybrid Genetic Search

Tabel 7.4: State of the art werken

De resultaten werden overgenomen uit de tabellen van Mecler Et. al [4].

7.4.1 Configuratie

Op basis van bovenstaande experimenten en vele batch experimenten werd een configuratie gekozen om te vergelijken met state of the art resultaten. Een backup local search werd als laatste stap in de methode toegevoegd. Deze methode voert steepest descent uit met job swaps. Deze laatste stap zal er voor zorgen dat een oplossing niet in een situatie eindigt waar de oplossing zich in een directe omgeving bevindt van betere oplossingen. Zo een oplossing kan snel gevonden worden door steepest descent en job swaps. Daardoor eindigt de oplossing in minstens een lokaal minimum.

METHODE		PARAMETERS	
Evaluatie	switches	$it(vmin)$	30000
Constructieve heuristiek	tool sequencing	$it(vmax)$	670000
Metaheuristiek	simulated annealing	$\mathcal{T}_0$	10
Local search operator	SMRI-BC	$\mathcal{T}_1$	0.001
Backup local search	steepest descent + job swaps	$ruinSize$	10
		$blinkRate$	0.01

7.5 Resultaten

DIF staat voor het percentage verschil tussen de beste oplossing en onze oplossing.³ Negatief is slechter, positief beter. De onderste rij van de tabel is het gemiddelde van elke kolom. De tijd is t , in seconden.

N	M	C	i	SMRI			ILS			HGS			DIF
				S_{best}	S_{avg}	t	S_{best}	S_{avg}	t	S_{best}	S_{avg}	t	
10	10	7	10	3.0	3.0	3.48	3.0	3.0	0.04	3.0	3.0	0.05	0.0
10	10	6	10	4.1	4.1	4.75	4.1	4.1	0.05	4.1	4.1	0.06	0.0
10	10	5	10	5.8	5.8	5.34	5.8	5.8	0.09	5.8	5.8	0.06	0.0
15	20	12	10	7.2	7.2	15.9	7.2	7.2	0.69	7.2	7.2	0.6	0.0
10	10	4	10	8.5	8.5	6.59	8.5	8.5	0.09	8.5	8.5	0.07	0.0
15	20	10	10	9.7	9.7	29.65	9.7	9.7	1.09	9.7	9.7	0.69	0.0
15	20	8	10	13.7	13.7	35.43	13.7	13.7	1.34	13.7	13.7	0.79	0.0
15	20	6	10	20.5	20.5	33.6	20.5	20.5	1.06	20.5	20.5	0.88	0.0
30	40	25	10	27.8	27.83	300.47	26.3	26.47	133.27	26.2	26.32	48.01	-6.11
30	40	20	10	48.5	48.6	285.79	46.6	46.78	167.41	46.6	46.66	60.58	-4.08
30	40	17	10	66.8	67.03	319.76	65.5	65.82	132.97	65.4	65.43	71.77	-2.09
30	40	15	10	85.1	85.1	291.06	83.9	84.09	96.49	83.5	83.5	74.71	-1.92
40	60	30	10	91.7	91.93	250.37	85.2	85.42	1473.24	84.3	84.66	484.14	-8.32
40	60	25	10	127.5	127.77	300.5	121.2	121.68	1037.2	120.3	120.61	466.12	-5.71
40	60	22	10	156.6	156.7	311.87	151.6	152.05	684.95	150.3	150.45	437.85	-4.09
40	60	20	10	181.6	181.9	333.92	178.0	178.36	474.0	176.5	176.71	501.21	-2.77
			10	53.6	53.7	158.03	51.9	52.1	262.75	51.6	51.7	134.22	-4.39

Tabel 7.5: Vergelijking Catanzaro dataset

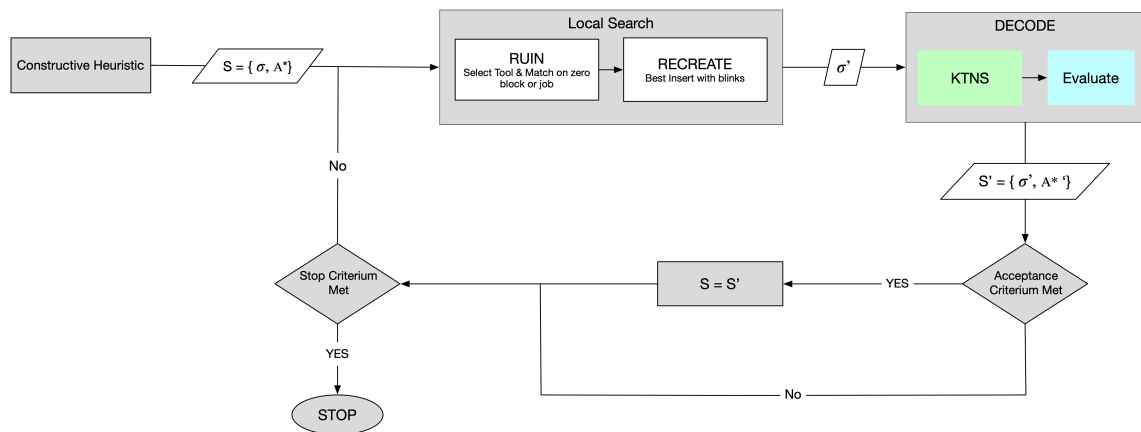
Het is duidelijk dat we de state of the art nog niet hebben overschreden. We zijn tussen de twee en acht procent verwijderd van de beste resultaten telkens. Alle kleine instanties hebben we opgelost tot de optimale oplossing, aangeduid door nul procent.

3

Deze methode is geen goede statische methode. Een bijlage wordt beschikbaar gesteld tijdens de presentatie die een wilcoxon signed-rank test uitvoert, deze zal dit rechtzetten.

Hoofdstuk 8

Conclusie



Figuur 8.1: Finale local search architectuur voor het SSP

8.1 Samenvatting

In dit werk hebben we het SSP behandeld. De eigenschappen werden duidelijk gemaakt. Er wordt een indirecte representatie van het probleem gebruikt om het op te lossen. Hierbij wordt de zoektocht uitgevoerd op de job sequentie. Telkens wordt een permutatie van deze sequentie bepaald. Vervolgens bepaalt de decoder de echte oplossing. Deze stap zorgt voor extra reken complexiteit maar beperkt de oplossingen ruimte voor de local search. De decoder is een implementatie van KTNS. Hierbij minimaliseert men het aantal tool switches door de snelst nodige tools te bewaren in het magazijn. Een nieuwe ruin en recreate methode werd geïntroduceerd. Als ruin stap wordt een tool stochastisch geselecteerd. Vervolgens worden alle jobs die deze bevatten verwijderd. De logica hier: deze jobs zijn gerelateerd. Als recreate stap worden ze volgens een greedy insert geplaatst op de beste plaats. Het werk werd vergeleek en geanalyseerd door middel van benchmark instanties die reeds vele malen in de literatuur aan bod kwamen. Deze nieuwe ruin en recreate methode benadert state of the art resultaten.

Bibliografie

- [1] G. V. Berghe, *Cursus Optimalisatietechnieken: powerpoint: Metaheuristieken*.
- [2] J. Christiaens en G. Vanden Berghe, „Slack Induction by String Removals for Vehicle Routing Problems”, *Transportation Science*, nummer April, 2020, ISSN: 0041-1655. DOI: [10.1287/trsc.2019.0914](https://doi.org/10.1287/trsc.2019.0914).
- [3] D. Calmels, „The job sequencing and tool switching problem: state-of-the-art literature review, classification, and trends”, *International Journal of Production Research*, jaargang 57, nummer 15-16, pagina's 5005–5025, aug 2019, ISSN: 0020-7543. DOI: [10.1080/00207543.2018.1505057](https://doi.org/10.1080/00207543.2018.1505057). adres: <https://www.tandfonline.com/doi/full/10.1080/00207543.2018.1505057>.
- [4] J. Mecler, A. Subramanian en T. Vidal, „A simple and effective hybrid genetic search for the job sequencing and tool switching problem”, 2019. arXiv: [1910.10021](https://arxiv.org/abs/1910.10021). adres: <http://arxiv.org/abs/1910.10021>.
- [5] G. S. Paiva en M. A. M. Carvalho, „Improved heuristic algorithms for the Job Sequencing and Tool Switching Problem”, *Computers and Operations Research*, jaargang 88, pagina's 208–219, 2017, ISSN: 03050548. DOI: [10.1016/j.cor.2017.07.013](https://doi.org/10.1016/j.cor.2017.07.013).
- [6] M. López-Ibáñez, J. Dubois-Lacoste, L. Pérez Cáceres, M. Birattari en T. Stützle, „The irace package: Iterated racing for automatic algorithm configuration”, *Operations Research Perspectives*, jaargang 3, pagina's 43–58, 2016, ISSN: 22147160. DOI: [10.1016/j.orp.2016.09.002](https://doi.org/10.1016/j.orp.2016.09.002).
- [7] D. Catanzaro, L. Gouveia en M. Labbé, „Improved integer linear programming formulations for the job Sequencing and tool Switching Problem”, *European Journal of Operational Research*, jaargang 244, nummer 3, pagina's 766–777, aug 2015. DOI: [10.1016/j.ejor.2015.02.018](https://doi.org/10.1016/j.ejor.2015.02.018).
- [8] H. H. Yanasse, R. d. C. M. Rodrigues en E. L. F. Senne, „Um algoritmo enumerativo baseado em ordenamento parcial para resolução do problema de minimização de trocas de ferramentas”, *Gestão & Produção*, jaargang 16, nummer 3, pagina's 370–381, 2009. DOI: [10.1590/s0104-530x2009000300005](https://doi.org/10.1590/s0104-530x2009000300005).
- [9] G. Ghiani, A. Grieco en E. Guerriero, „An exact solution to the TLP problem in an NC machine”, *Robotics and Computer-Integrated Manufacturing*, jaargang 23, nummer 6, pagina's 645–649, 2007, ISSN: 07365845. DOI: [10.1016/j.rcim.2007.02.011](https://doi.org/10.1016/j.rcim.2007.02.011).

- [10] Y. Crama, A. W. Kolen, A. G. Oerlemans en F. C. Spieksma, „Minimizing the number of tool switches on a flexible machine”, *International Journal of Flexible Manufacturing Systems*, jaargang 6, nummer 1, pagina's 33–54, 1994, ISSN: 09206299. DOI: [10.1007/BF01324874](https://doi.org/10.1007/BF01324874).
- [11] J. F. Bard, „A heuristic for minimizing the number of tool switches on a flexible machine”, *IIE Transactions (Institute of Industrial Engineers)*, jaargang 20, nummer 4, pagina's 382–391, 1988, ISSN: 15458830. DOI: [10.1080/07408178808966195](https://doi.org/10.1080/07408178808966195).
- [12] C. S. Tang, „Models Arising from a Flexible Manufacturing Machine , Part I : Minimization of the Number of Tool Switches Author (s): Christopher S . Tang and Eric V . Denardo Published by : INFORMS Stable URL : <http://www.jstor.org/stable/171322> REFERENCES Linked re”, jaargang 36, nummer 5, pagina's 767–777, 1988.
- [13] S. Kirkpatrick, C. D. Gelatt en M. P. Vecchi, „Optimization by simulated annealing”, *Science*, jaargang 220, nummer 4598, pagina's 671–680, 1983, ISSN: 00368075. DOI: [10.1126/science.220.4598.671](https://doi.org/10.1126/science.220.4598.671).

Bijlage A

Beschrijving van deze masterproef in de vorm van een wetenschappelijk artikel

Job Sequencing and Tool Switching Problem

Simon Vermeir, Greet Vanden Berghe

KU Leuven, Department of Computer Science, CODES & imec, 9000 Gent, Belgium

hello@simonvermeir.com, greet.vandenbergh@cs.kuleuven.be

August 2020

Abstract

Using tools for an extended amount of time without repeated switching is an efficient way to execute tasks or jobs. The Job Sequencing and Tool Switching Problem wants to take advantage of this fact. Frequently machines are capable of handling various jobs that each require their own set of tools. It is not conceivable to store these all at once. Hence switching is required; a time-consuming process. However, it is possible to significantly lower these switching times. By precisely arranging jobs and accommodating tools that are required later. Utilizing a heuristic approach it is possible to solve problems that would otherwise be impracticable to solve. This work introduces an elegant Ruin and Recreate method that allows for the exploration of a broad neighborhood while keeping tabs on the relationships between jobs. A significant speedup in the decoder and local Search procedure is achieved by using unique problem properties. Allowing a wide range of solutions to be explored in an acceptable time. Finally, choosing an appropriate objective function avoids plateaus in the search landscape.

1 Introduction

Frequently machines are capable of handling various jobs that each require their own set of tools. An example would be machining a part to a precise shape using different drills. Tools can be kept inside of a magazine, this way the machine can complete jobs without stopping.

Unfortunately, there is often not enough capacity for all tools of all jobs. Thus, switching is required when transitioning. A considerable amount of time could be wasted, tools may have to be calibrated or could be very heavy.

Choosing a good job sequence may therefore prove beneficial. Moreover, if sufficient capacity is left in the magazine tools could simply be kept in the magazine for later jobs. The goal of this paper is to develop a solution in order to minimize these switches as much as possible.

2 Problem Overview

Known as the "Job Sequencing and Tool Switching Problem" it can be stated formally in the following way: *Given a set of N jobs each requiring a specific set of M tools and a magazine with a certain tool capacity C . Determine the order each job is carried out in and which tools are needed during that job, in order to minimize the number of tool switches needed.*

Many variations and applications exist. Parallel jobs [1], Tool dependent switching cost [2] Although these variations are interesting, the focus here is on the uniform variant: *Tools are uniform in size and can be put anywhere in the magazine, switching time is constant, one machine is used and all tools of one job can fit in the magazine at once..*

2.1 Formulation

Let $J = \{1, 2, \dots, N\}$ be the set of jobs, $T = \{1, 2, \dots, M\}$ the set of all tools and T_j the set of tools required for a job where $j \in J, T_j \subseteq T$. The magazine size is C and $C \leq \max(|T_j|), j \in J$ [3].

Table 1: SSP instance example 1

Jobs	0	1	2	3	4
Tools	0	0	2	1	0
	1	2	3	2	3
	3		4	4	5
Magazine size = 3					

Example from **Paiva Et. al** [4]

An instance can be modeled by way of a $M \times N$ binary matrix A , with $a_{ij} = 1$ if job j requires tool i , 0 otherwise. [3]. Thus, each column j can be seen as T_j . As can be seen in table 2

Table 2: Example 1: binary matrix A

	Jobs	0	1	2	3	4
Tools	0	1	1	0	0	1
	1	1	0	0	1	0
	2	0	1	1	1	0
	3	1	0	1	0	1
	4	0	0	1	1	0
	5	0	0	0	0	1

The solution to the problem is some sequence σ which can be seen as a permutation of the set of jobs J . And a matrix A^* , with $a_{ij}^* = 1$ if job j requires or keeps tool i , 0 otherwise.

Alternatively, by permutating the columns of A^* according to the sequence σ , the solution can be modeled as a $M \times N$ matrix P . This provides an intuitive way to interpret the state of the magazine at any time, as can be seen in figure 1[3].

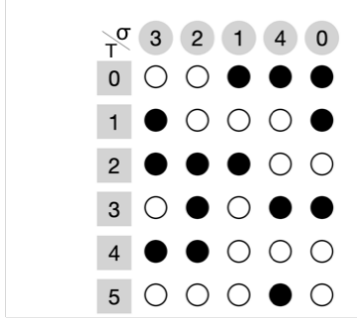


Figure 1: Example 1: Permutation matrix P

Finally a tool setup can be considered as the insertion of a tool in the magazine. A tool switch is the combination of a tool setup and tool removal [3]. The objective is to minimize the number of tool switches.

2.2 Properties

Tang and Denardo formally introduced the problem for the first time. They separated the problem in two subproblems:

- The sequencing problem (SP): Determine the order in which the jobs have to be executed. The sequencing problem is $\mathcal{NP}$ hard. As shown by [3]
- The tooling problem (TP): Given the job sequence, determine which tools are needed at every position. Solvable in *Polynomial* time. As shown by [5]

Although the Sequencing Problem is hard to solve, they demonstrated that the tooling problem is trivial. They introduced the "Keep Tool Needest Soonest" policy or "KTNS", which states the following:

- Tools are only inserted when they are needed.
- When tools need to be removed to accommodate new tools, remove the ones needed the least soon. i.e. "keep the tools needed soonest"

Applying this policy ensures that every job is able to use its required tools, additionally it uses the left over space in the magazine to accommodate tools that are required later. Beyond that, they proved that given a sequence σ (SP), the *optimal* tool configuration (TP) can be found by using KTNS. *This important fact enables us to model the problem as finding the optimal job sequence. Once a sequence is found the tool switches can be reduced by applying KTNS.*

Moreover, by applying this policy the number of tool switches can easily be recognized each time a one turns into a zero in every row within P . Which is equivalent to replacing a tool by another one. [6].

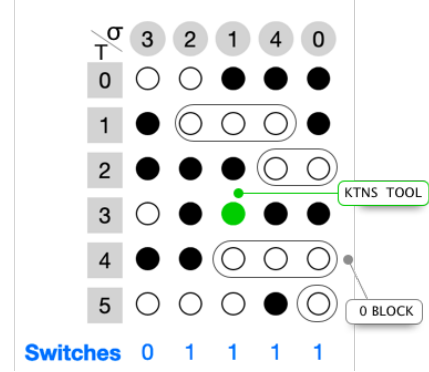


Figure 2: Example 1: Permutation matrix with KTNS applied

Additionally **Tang and Denaro** proposed modeling the SSP as an instance of the Traveling Salesman Problem. They defined a weighted undirected graph $G = (V, E, lb)$, where the vertices V represent the set of jobs J , and the edges $\{i, j\} \in E$ represent job j following job i . Finally the weight of an edge $\{i, j\}$ is $lb(i, j) = \max(|T_i \cup T_j| - C, 0)$, a lower bound for the amount of switches between job i and j [3].

Crama Et al. [3] put forward the idea of recognizing 0/1 blocks within the problem, as can be seen in figure 2. Interestingly the number of tool switches can only be reduced when we are able to eliminate a complete 0 block. This means we can reformulate the objective as reducing the amount of 0 blocks. Additionally, they reformulated KTNS as filling in the maximum amount of complete 0 blocks while respecting the constraints earlier mentioned.

Moreover they noticed that solving the problem as a TSP using the Lower bound proposed by **Tang and Denaro** does not work well when considering a sparse matrix. Due to the fact that a lot of tools can be added by KTNS.

Ghiani Et al. [7] recognized a symmetry property within the problem. They stated and proved that thanks to KTNS the inverse of every job sequence has the same amount of tool switches. Which can be used to reduce the search space.

Ultimately, a lot of attempts at the SSP have been made. Yet the difficulty of the SSP limits the ability of exact methods to be used on instances with more than 25 jobs [8]. This illustrates the fact that a large portion of the research has put a focus on heuristic methods recently.

3 Methodology

3.1 Local Search

Local search is used to solve the SSP. Figure ?? illustrates the mechanism used. A solution to an instance of the SSP is represented by $S = \{\sigma, A^*\}$. The constructive by **Paiva Et al.** [4] is used.

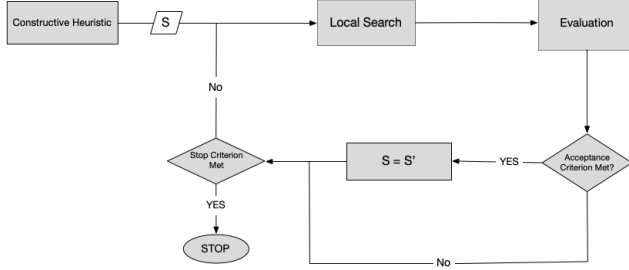


Figure 3: Local search method

3.2 Decoder

An indirect representation of the problem is used [6]. When a solution is changed to a neighbor solution it is changed by permutating its job sequence. The full solution can be computed afterwards by performing $KTNS(\sigma)$ on the job sequence. Afterwards the solution is evaluated by counting the number of switches. We call the combination of these two steps the decoding of our solution. Algorithm 1 gives the algorithm used to perform KTNS. The developed algorithm does not use recursion as opposed to the algorithm proposed by **Tang and Denardo** [5].

4 Ruin and Recreate

A new method is proposed for computing a candidate solution S' .

This method is based on **Slack Induction by String Removals for Vehicle Routing Problems** by [9]. Ruin and recreate works by using one or more operators to destroy parts of a solution, and recreating it into a new candidate solution. The advantage of this method is that a large neighborhood can be explored. In our case the sequence σ gets destroyed by removing certain jobs and recreated by inserting them again in a different place.

The method proposed here is called SMRI and uses 4 distinct phases: select, match, remove and insert. The method developed is visually illustrated on the next page step by step.

Algorithm 1: KTNS

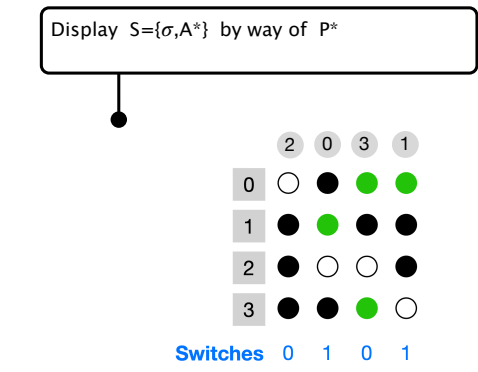
Input: New sequence σ'

Output: New Solution $S' = \{\sigma', A'^*\}$

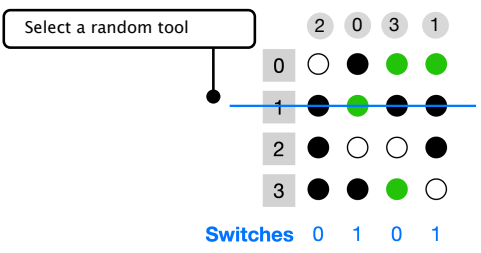
```

1 function KTNS( $\sigma'$ )
2    $A'^* = A^*$ 
3    $ktnsId = ktnsId + 1$ 
4    $prevJob \leftarrow job - 1$ 
5    $nextJob \leftarrow job + 1$ 
6   if job not first in  $\sigma$  then
7      $nToolsAdd \leftarrow \max(0, |T_{job}| - C)$ 
8      $diff : LinkedList \leftarrow T_{prevJob}^* \setminus T_{job}$ 
9     while  $nToolsAdd > 0$  &  $diff.length > 0$  &
        $nextJob < N$  do
10      for tool  $\in diff$  do
11        if tool  $\in T_{nextJob}$  then
12           $A_{tool, job} \leftarrow ktnsId$ 
13           $diff.removeTool(tool)$ 
14           $nToolsAdd \leftarrow nToolsAdd - 1$ 
15           $nextJob \leftarrow nextJob + 1$ 
16      for tool  $\in diff$  do
17         $A_{tool, job} \leftarrow ktnsId$ 
18         $diff.removeTool(tool)$ 
19         $nToolsAdd \leftarrow nToolsAdd - 1$ 
20        if  $nToolsAdd == 0$  &  $diff.length == 0$ 
           then
21          break
22    $S' = \{\sigma', A'^*\}$ 
23   return  $S'$ 
  
```

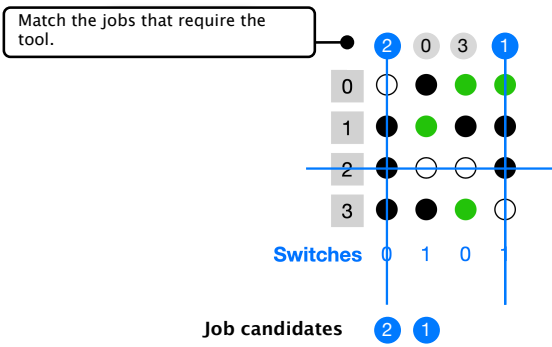
START



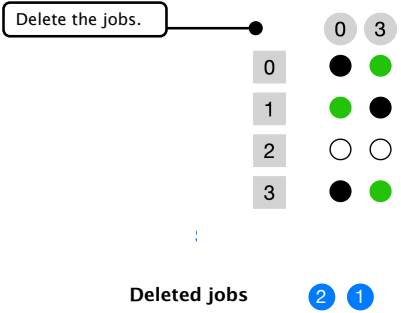
PHASE 1
SELECT



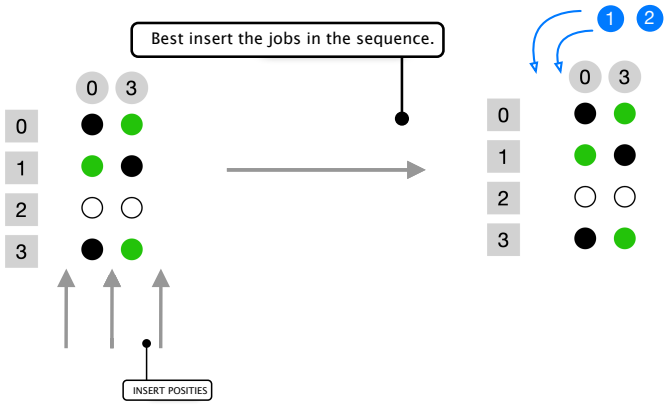
PHASE 2
MATCH



PHASE 3
REMOVE

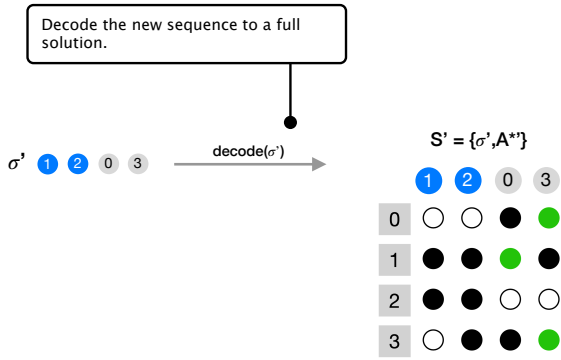


PHASE 4
INSERT



END

decode(σ')



4.1 Meta Heuristic

Kirkpatrick Et. Al. Kirkpatrick1983 introduced Simulated Annealing, a meta heuristic based on the static model of Annealing.

Annealing or annealing is a process used in the metal industry. It tries to get irregularities out of steel.

The systematic approach by **Christiaens and Vanden Berghe** [9] from **Slack Induction by String Removals for Vehicle Routing Problems** has been applied for the implementation of Simulated Annealing. Adjustments were made as needed for the application to this problem. The algortihm used is provided in 2.

Algorithm 2: Simulated Annealing

```

Input: Intital solution  $S = \{\sigma, A^*\}$ 
Input: Best solution  $S_{best} = \{\sigma, A^*\}$ 
1 function SimulatedAnnealing( $S$ )
2   for  $f$  iterations do
3      $\sigma' = \text{ruinAndRecreate}(S)$ 
4      $S' = \text{decode}(\sigma')$ 
5     if
6        $\text{cost}(S) < \text{cost}(S') - \mathcal{T} \ln(\text{random}(0,1))$ 
7       then
8          $S \leftarrow S'$ 
9       if  $n\text{Switches}(S') < n\text{Switches}(S_{best})$ 
10        then
11           $S_{best} \leftarrow S'$ 
12        if  $S' == S_{best}$ , for  $n$  iterations then
13          return  $S_{best}$ 
14         $\mathcal{T} \leftarrow \mathcal{T} \cdot c$ 
15 return  $S_{best}$ 

```

5 Results

Our new method is compared against the state of the art, using the dataset provided by **Catanzaro Et al.**. The results can be found in table 3 , last page. Our results (SMRI) are compared against the results of **Paiva Et al.**[4] and **Mecler Et al.**[6]. The results where obtained from **Mecler Et al.**[6]. DIF stands for the percentage difference between the best solution and our solution, negative means ours is worse performing.

6 Conclusion

A new ruin and recreate method was introduced. As a ruin step, a tool is selected stochastically. Then all jobs requiring that tool are deleted. The logic here: these jobs are related. As a recreate step they are placed in the best place according to a greedy insert. Our methode did not exceed state of the art results. However, it was clear that the method proposed here has potential. It should therefore be further investigated.

References

- [1] A. C. Beezão, J. F. Cordeau, G. Laporte, and H. H. Yanasse, "Scheduling identical parallel machines with tooling constraints," *European Journal of Operational Research*, volume 257, number 3, pages 834–844, 2017, ISSN: 03772217. DOI: 10.1016/j.ejor.2016.08.008.
- [2] D. Calmels, "The job sequencing and tool switching problem: state-of-the-art literature review, classification, and trends," *International Journal of Production Research*, volume 57, number 15-16, pages 5005–5025, Aug. 2019, ISSN: 0020-7543. DOI: 10.1080/00207543.2018.1505057. [Online]. Available: <https://www.tandfonline.com/doi/full/10.1080/00207543.2018.1505057>.
- [3] Y. Crama, A. W. Kolen, A. G. Oerlemans, and F. C. Spieksma, "Minimizing the number of tool switches on a flexible machine," *International Journal of Flexible Manufacturing Systems*, volume 6, number 1, pages 33–54, 1994, ISSN: 09206299. DOI: 10.1007/BF01324874.
- [4] G. S. Paiva and M. A. M. Carvalho, "Improved heuristic algorithms for the Job Sequencing and Tool Switching Problem," *Computers and Operations Research*, volume 88, pages 208–219, 2017, ISSN: 03050548. DOI: 10.1016/j.cor.2017.07.013.
- [5] C. S. Tang, "Models Arising from a Flexible Manufacturing Machine , Part I : Minimization of the Number of Tool Switches.," volume 36, number 5, pages 767–777, 1988.
- [6] J. Mecler, A. Subramanian, and T. Vidal, "A simple and effective hybrid genetic search for the job sequencing and tool switching problem," 2019. arXiv: 1910.10021. [Online]. Available: <http://arxiv.org/abs/1910.10021>.
- [7] G. Ghiani, A. Grieco, and E. Guerriero, "An exact solution to the TLP problem in an NC machine," *Robotics and Computer-Integrated Manufacturing*, volume 23, number 6, pages 645–649, 2007, ISSN: 07365845. DOI: 10.1016/j.rcim.2007.02.011.
- [8] D. Catanzaro, L. Gouveia, and M. Labbé, "Improved integer linear programming formulations for the job Sequencing and tool Switching Problem," *European Journal of Operational Research*, volume 244, number 3, pages 766–777, Aug. 2015. DOI: 10.1016/j.ejor.2015.02.018.
- [9] J. Christiaens and G. Vanden Berghe, "Slack Induction by String Removals for Vehicle Routing Problems," *Transportation Science*, number April, 2020, ISSN: 0041-1655. DOI: 10.1287/trsc.2019.0914.

N	M	C	i	SMRI $\mathcal{S}_{best}$	SMRI $\mathcal{S}_{avg}$	SMRI t	ILS $\mathcal{S}_{best}$	ILS $\mathcal{S}_{avg}$	ILS t	HGS $\mathcal{S}_{best}$	HGS $\mathcal{S}_{avg}$	HGS t	DIF
10	10	7	10	3.0	3.0	3.48	3.0	3.0	0.04	3.0	3.0	0.05	0.0
10	10	6	10	4.1	4.1	4.75	4.1	4.1	0.05	4.1	4.1	0.06	0.0
10	10	5	10	5.8	5.8	5.34	5.8	5.8	0.09	5.8	5.8	0.06	0.0
15	20	12	10	7.2	7.2	15.9	7.2	7.2	0.69	7.2	7.2	0.6	0.0
10	10	4	10	8.5	8.5	6.59	8.5	8.5	0.09	8.5	8.5	0.07	0.0
15	20	10	10	9.7	9.7	29.65	9.7	9.7	1.09	9.7	9.7	0.69	0.0
15	20	8	10	13.7	13.7	35.43	13.7	13.7	1.34	13.7	13.7	0.79	0.0
15	20	6	10	20.5	20.5	33.6	20.5	20.5	1.06	20.5	20.5	0.88	0.0
30	40	25	10	27.8	27.83	300.47	26.3	26.47	133.27	26.2	26.32	48.01	-6.11
30	40	20	10	48.5	48.6	285.79	46.6	46.78	167.41	46.6	46.66	60.58	-4.08
30	40	17	10	66.8	67.03	319.76	65.5	65.82	132.97	65.4	65.43	71.77	-2.09
30	40	15	10	85.1	85.1	291.06	83.9	84.09	96.49	83.5	83.5	74.71	-1.92
40	60	30	10	91.7	91.93	250.37	85.2	85.42	1473.24	84.3	84.66	484.14	-8.32
40	60	25	10	127.5	127.77	300.5	121.2	121.68	1037.2	120.3	120.61	466.12	-5.71
40	60	22	10	156.6	156.7	311.87	151.6	152.05	684.95	150.3	150.45	437.85	-4.09
40	60	20	10	181.6	181.9	333.92	178.0	178.36	474.0	176.5	176.71	501.21	-2.77
10				53.6	53.7	158.03	51.9	52.1	262.75	51.6	51.7	134.22	-4.39

Table 3: Results for the Catanzaro dataset

Bijlage B

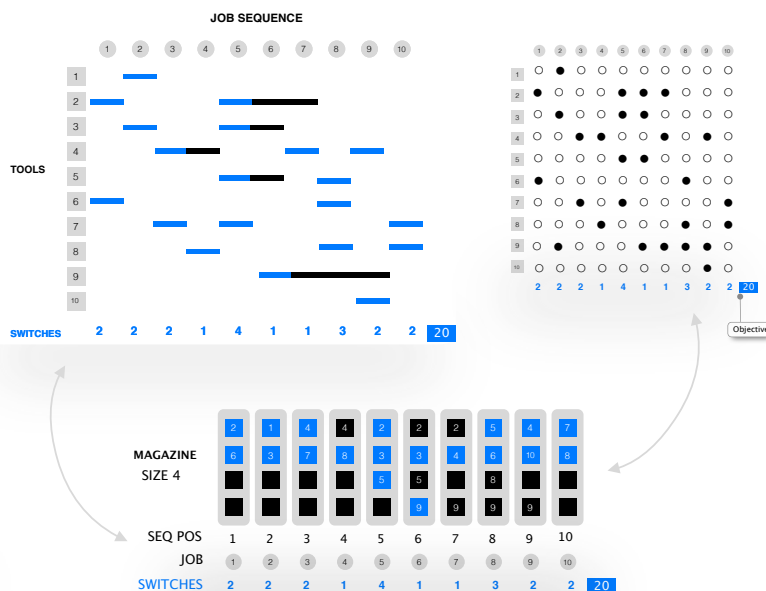
Poster

The Job Sequencing and Tool Switching Problem

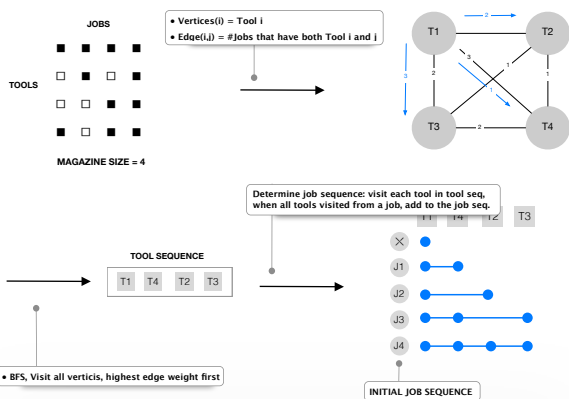
Abstract

Using tools for an extended amount of time without repeated switching is an efficient way to execute tasks or jobs. Frequently machines are capable of handling various jobs that each require their own set of tools. It is not conceivable to store these all at once. Hence switching is required, a time-consuming process. However, it is possible to significantly lower these switching times. By precisely arranging these jobs and accommodating tools that are required later. Utilizing a heuristic approach it is possible to solve these problems that would otherwise be impracticable to solve.

Method



INITIAL SOLUTION



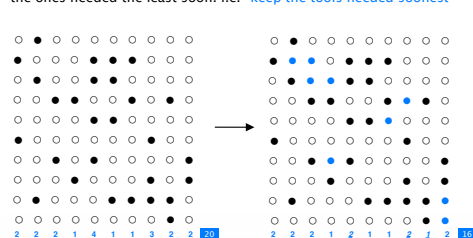
Introduced by Paiva et al.

DECODER

Converts an indirect solution to a full one. This solution consists of the job sequence and the job-tool matrix. The job tool-matrix contains the tools needed for each job. These include tools required by the job and by a subsequent job. They are determined using the Keep Tool Needest Soonest policy or KTNS.

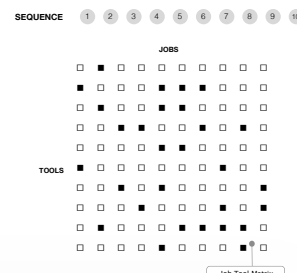
KTNS

- Tools are only inserted when they are needed.
- When tools need to be removed to accommodate new tools, remove the ones needed the least soon. i.e. "keep the tools needed soonest"

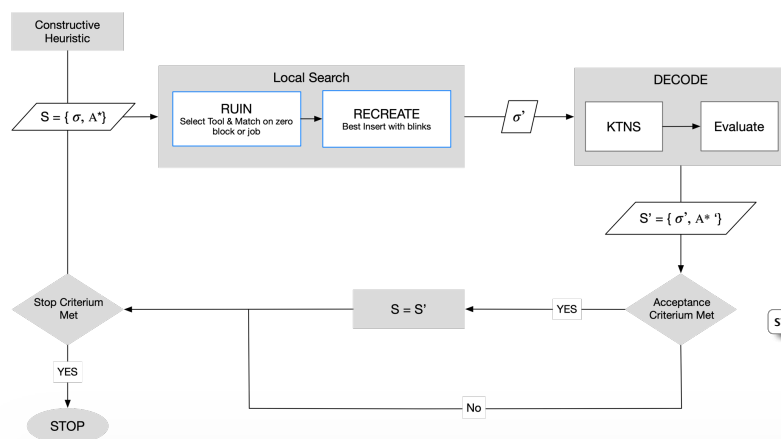


Introduced by Tang & Denaro

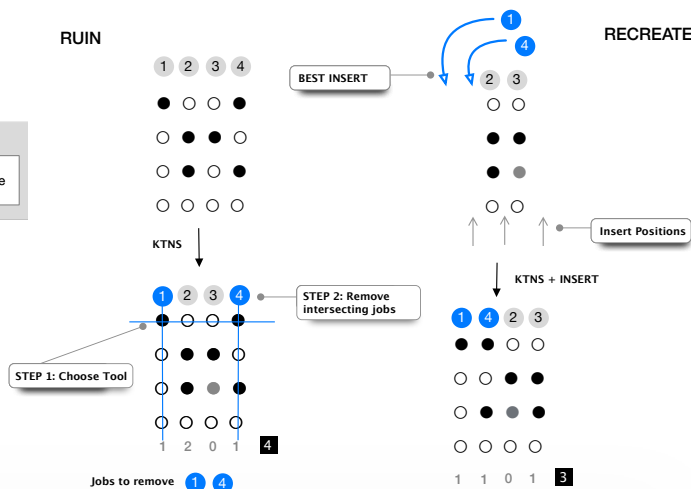
FULL SOLUTION



LOCAL SEARCH



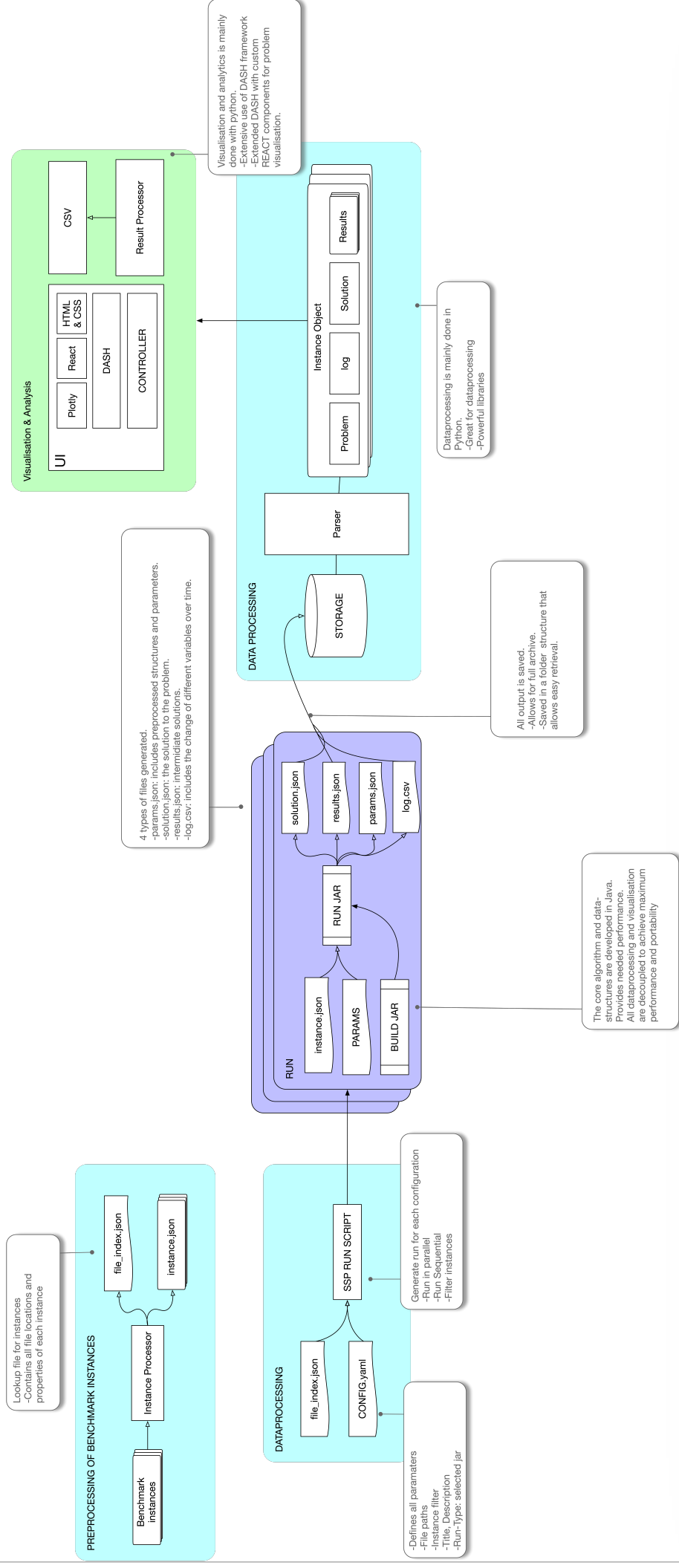
RUIN



Bijlage C

Architectuur dataprocessing

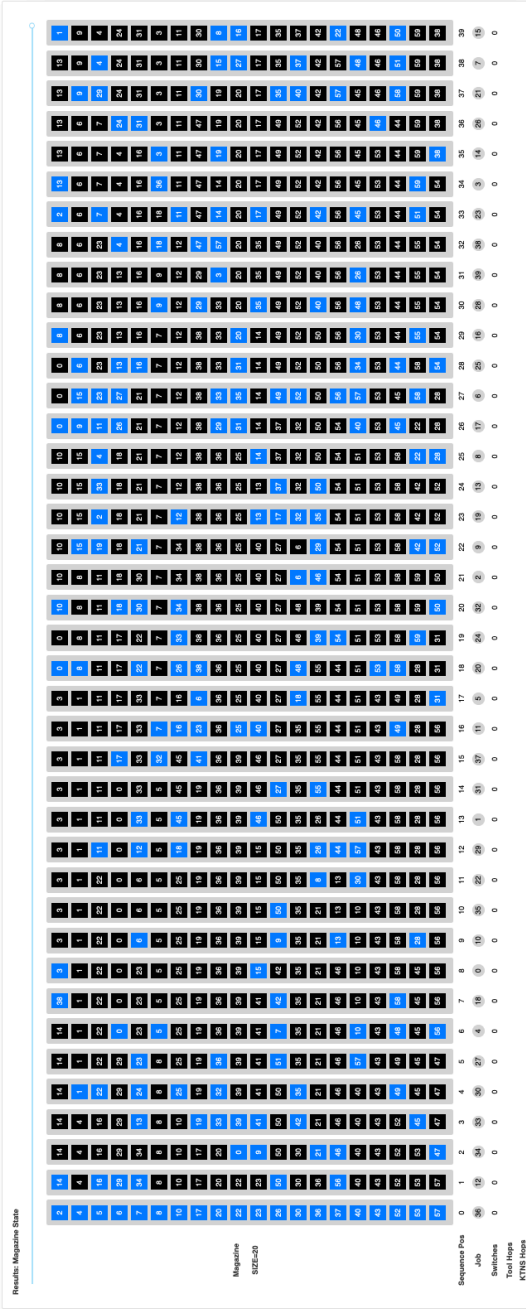
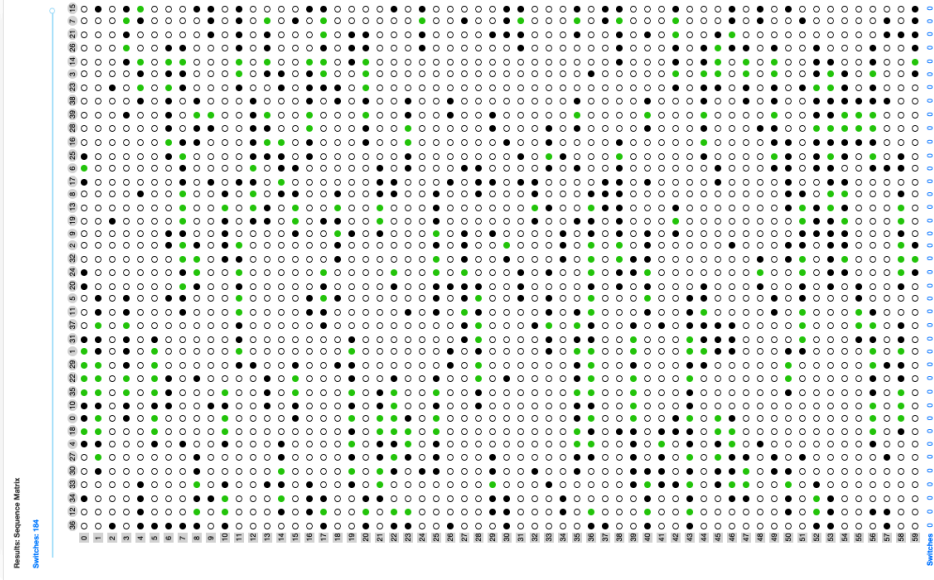
SSP processing architecture



SSP dashboard

[illegible]

SSP dashboard



```
Parameters
w"root": i { i from
  "PROJECT_ROOT", { dd filename
    "ROOT_FOLDER", string "/home/monowormal/189/data/instances/instance0"
    "REFERENCE_FOLDER", string "/home/monowormal/189/data/instances/instance0/cat_40_40_20_1"
    "REFERENCE", string "cat_40_40_20_1"
    "RNN_SIZE", string "60x21"
    "num_epochs", i { i from 1 to 100
  }
}
```

FACULTEIT INDUSTRIËLE INGENIEURSWETENSCHAPPEN
TECHNOLOGIECAMPUS GENT
Gebroeders De Smetstraat 1
8200 GENT, België
tel. + 32 50 66 48 00
iiw.gent@kuleuven.be
www.iw.kuleuven.be

