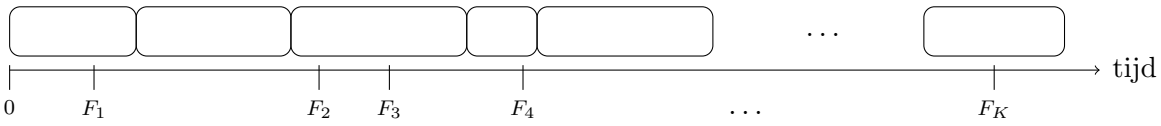
Universiteit Utrecht

Single Machine Scheduling met verboden tijden



Bachelorscriptie
Suzanne Vincken

Begeleider:
Dr. E.J. van Leeuwen
Departement Informatica

Faculteit Bètawetenschappen
Wiskunde (en Informatica)

13-01-2019

Samenvatting

In deze scriptie wordt het Single Machine Scheduling (SMS) probleem met verboden tijden onderzocht. We geven een motivatie voor dit probleem en een kort overzicht van onderzoeken die er zijn uitgevoerd omtrent dit onderwerp. Daarna gaan we dieper in op drie gevallen van het probleem.

Eerst wordt het SMS probleem met K verboden starttijden besproken, waarbij taken niet mogen starten op deze verboden tijden. Dit is gebaseerd op de paper van J.-C. Billaut en F. Sourd, 2009 [1]. We laten zien dat de beslissingsvariant van dit probleem sterk $\mathcal{NP}$-volledig is. Daarna laten we zien dat er altijd een optimaal schema bestaat als er minstens $L = 2K(K+1)$ taken van verschillende lengte zijn. We geven een dynamisch programmeeralgoritme voor het geval dat er minder dan L verschillende taken zijn. Dit algoritme heeft een looptijd van $O\left(K^3 n^{2K^2+2K-1}\right)$, dat polynomiaal is als K een constante is.

Vervolgens wordt het SMS probleem met één verboden startinterval behandeld, waarbij taken niet mogen starten in dat interval. We bewijzen dat de beslissingsvariant $\mathcal{NP}$-volledig is. Daarna geven we een pseudo-polynomiaal algoritme voor dit probleem met een looptijd van $O\left(n \cdot F_1\right)$, waarbij n staat voor het aantal taken en F_1 voor de eerste verboden starttijd. Dit algoritme is van eigen werk.

Tot slot wordt het SMS probleem met verboden start- en eindtijden behandeld waarbij de taken niet mogen starten of eindigen op een verboden tijd. Dit is gebaseerd op de paper van M. Gabay, C. Rapine en N. Brauner, 2016 [6] en er wordt gebruik gemaakt van een aantal resultaten uit de paper van C. Rapine en N. Brauner, 2013 [18]. We tonen aan dat het SMS probleem met verboden start- en eindtijden sterk $\mathcal{NP}$-volledig is. Daarna bewijzen we dat dit probleem *fixed parameter tractable (FPT)* is, ten aanzien van de parameter K, het aantal verboden tijden. We maken onderscheid tussen instanties waarbij er meer verschillende taken zijn dan verboden tijden en instanties waarbij dat niet zo is. Voor het eerste geval geven we een exact algoritme. Voor het tweede geval geven we een geheeltallig lineair programma (ILP) dat een oplossing vindt, gevolgd door een algoritme van eigen werk dat deze oplossing vervolgens construeert.

Inhoudsopgave

1 Inleiding

Het *Single Machine Scheduling (SMS)* probleem gaat over het inplannen van taken op een enkele machine. Wij zullen gaan kijken naar het SMS probleem met verboden tijden. Dit houdt in dat een taak niet mag beginnen en/of eindigen op een verboden tijd. Een taak mag tijdens een verboden tijd wel uitgevoerd worden door de machine. Het doel is nu om een zo efficiënt mogelijk schema te maken. We willen natuurlijk dat de machine zo snel mogelijk alle taken uitvoert en dat deze zo min mogelijk tijd geen taak aan het uitvoeren is.

Een belangrijke toepassing van dit probleem vinden we in het uitvoeren van chemische experimenten [18]. C. Rapine en N. Brauner hebben samengewerkt met het *Institut Français du Pétrole*, een groot onderzoekscentrum dat onderzoek doet omtrent energie en transport. Zij doen hun onderzoeken onder andere met behulp van chemische experimenten. Deze experimenten duren gemiddeld tussen de drie dagen en drie weken. Het uitvoeren van zo'n experiment kan door de machine worden gedaan zonder de hulp van een scheikundige. Echter, bij het opstarten en afsluiten van een experiment is het belangrijk dat de scheikundige aanwezig is. Het opstarten en het afsluiten kunnen in korte tijd, maar het is wel noodzakelijk dat dit door een medewerker wordt geassisteerd. Oftewel, als er geen medewerkers aanwezig zijn, omdat het bijvoorbeeld midden in de nacht of weekend is, dan kan een experiment niet worden opgestart of afgerond. Bij het maken van een schema is het dus van belang dat de taken zo worden ingepland dat ze niet starten of eindigen op een verboden tijd.

Het kan ook zo zijn dat er externe apparatuur gehuurd moet worden voor de start of de afronding van een experiment. Stel bijvoorbeeld dat er een groot product geproduceerd is dat met een hijskraan verplaatst moet worden, dan moet deze hijskraan wel ingehuurd worden. Als een product klaar is, maar niet meteen kan worden verplaatst door de hijskraan, dan kan de machine niet verder met zijn volgende taak en dan gaat er dus kostbare tijd verloren. Het is dus van belang dat de taken zo worden ingepland dat de benodigde apparatuur present is bij het opstarten of afronden van een experiment.

In deze paper zullen we eerst in Sectie 2 een overzicht geven van onderzoek dat is gedaan omtrent het SMS probleem. Vervolgens zullen we twee gevallen van het SMS probleem behandelen. We kijken eerst naar het SMS probleem met verboden starttijden, aan de hand van een paper van J.-C. Billaut en F. Sourd, 2009 [1]. Bij dit probleem mogen taken niet starten op een verboden starttijd, zie Sectie 3. We zullen ook een speciaal geval bekijken van dit probleem waarbij er precies één interval van verboden starttijden is, zie Sectie 4. In deze sectie behandelen we ook een algoritme van eigen werk. Tot slot zullen we in Sectie 5 het SMS probleem behandelen waarbij we verboden start- en eindtijden hebben. Deze sectie is gebaseerd op twee papers: de paper van M. Gabay, C. Rapine en N. Brauner, 2016 [6] en de paper van C. Rapine en N. Brauner, 2013 [18]. Verder bevat deze sectie ook een algoritme van eigen werk.

2 Literatuur

Er is heel veel onderzoek gedaan naar Single Machine Scheduling. We zullen ons hier beperken tot SMS met verboden tijden. Over het algemeen worden daar twee gevallen van onderzocht.

1. De machine heeft onderhoudsperioden waarin er geen taken uitgevoerd kunnen worden.
2. De machine is altijd beschikbaar, maar er is een aantal tijden en/of intervallen waarop de taken niet mogen starten of eindigen. Taken mogen wel uitgevoerd worden tijdens zo'n verboden interval.

In het algemeen is het bij beide problemen het doel om de eindtijd van het schema te minimaliseren.

We zullen in de Secties 2.1 en 2.2 een overzicht geven van een aantal onderzoeken die gedaan zijn rondom de twee bovenstaande problemen. In Sectie 2.3 zullen we kort nog een aantal andere onderzoeken noemen.

2.1 Een machine met onderhoudsperioden

Het SMS probleem met machineonderhoud bevat veel deelproblemen. Zo wordt er onderscheid gemaakt tussen twee soorten machineonderhoud: gepland onderhoud (deterministisch) en ongepland onderhoud (stochastisch). Er wordt ook vaak onderscheid gemaakt tussen verschillende soorten taken. Bij machineonderhoud kunnen hervatbare taken en niet-hervatbare taken voorkomen. Een niet-hervatbare taak moet in één keer worden uitgevoerd en mag niet worden onderbroken. Als zo'n taak wel wordt onderbroken, dan moet de hele taak opnieuw worden uitgevoerd. Hervatbare taken kunnen na een onderhoudsperiode gewoon worden hervat. Soms wordt er ook een variant van het probleem besproken waarbij de taken extra eigenschappen hebben, zoals een werktijd die afhankelijk is van de periode waarin hij wordt uitgevoerd.

We zullen hieronder alleen de geplande onderhoudsperioden met niet-hervatbare taken behandelen. Dit zullen we opdelen in twee groepen: periodieke onderhoudsperioden en 1 geplande onderhoudsperiode. We bespreken een deel van de papers die over dit onderwerp zijn geschreven.

In 2010 hebben Y. Ma, C. Chu en C. Zuo [16] een samenvatting gegeven van het onderzoek dat is gedaan naar het Machine Scheduling probleem met machineonderhoud. Ze behandelen hierbij alleen de variant waarbij de onderhoudsperioden gepland zijn. Verder beperken ze zich niet tot single-machines, maar ze behandelen ook multi-machines. Bij deze problemen kunnen taken meerdere eigenschappen hebben, naast hervatbaar of niet-hervatbaar. Zo kunnen taken bijvoorbeeld ook deadlines hebben of een werktijd die afhangt van hun starttijd. In de paper worden voor deze problemen complexiteitsanalyses, exacte algoritmes en benaderingsalgoritmes besproken die in eerdere onderzoeken zijn gevonden.

Periodiek gepland machineonderhoud met niet-hervatbare taken

In 2003 hebben C.J. Liao en W.J. Chen [15] in hun paper een branch-and-bound algoritme gegeven dat een optimale oplossing geeft. Verder geven ze een heuristiek voor het oplossen van grote instanties van dit probleem en ze hebben computationele resultaten gegeven.

In 2007 hebben M. Ji, Y. He en T.C.E. Cheng [9] laten zien dat de worst-case ratio van het klassieke LPT algoritme (Longest Processing Time first [14]) gelijk is aan 2. Daarna laten ze zien dat er geen betere polynomiale benaderingsalgoritmes bestaan, als er geldt dat $\mathcal{P} \neq \mathcal{NP}$.

Een periode gepland machineonderhoud met niet-hervatbare taken

In 2005 hebben C. Sadfi *et al.* [20] een benaderingsalgoritme gegeven om dit probleem op te lossen. Dit algoritme heeft worst-case een bovengrens van $\frac{3}{17}$ op de fout.

In 2006 hebben Y. He, W. Zhong en H. Gu [8] in hun paper een *polynomial time aprroximation scheme (PTAS)* gegeven voor dit probleem.

In 2009 hebben I. Kacem en R. Mahjoub [11] een paper uitgebracht die gebaseerd is op de bovengenoemde paper van Y. He, W. Zhong en H. Gu uit 2006 [8]. I. Kacem en R. Mahjoub

geven een *fully polynomial time approximation scheme (FPTAS)* met een complexiteit van $O\left(n^2/\epsilon^2\right)$, waarin n het aantal taken voorstelt en ϵ de toegestane error bound. Dit algoritme is sneller dan de eerder ontworpen algoritmes.

2.2 SMS met verboden start- en/of eindtijden

Hieronder staat een aantal resultaten genoemd omtrent het SMS probleem met verboden start- en/of eindtijden. Deze lijst is niet volledig. Een aantal van de onderstaande papers wordt in de rest van deze scriptie nader besproken. Dit staat er dan specifiek bij.

In 2009 hebben J.-C. Billaut en F. Sourd [1] een paper gepubliceerd waarin ze het Single Machine Scheduling probleem met verboden starttijden (SMS-FST) behandelen. Ze laten zien dat het SMS-FST probleem voor de minimalisatie van de lengte van het schema een sterk $\mathcal{NP}$-volledig probleem is. Verder geven zij polynomiale algoritmes om dit probleem op te lossen voor een begrensd aantal verboden starttijden. We zullen deze paper in meer detail behandelen in Sectie 3 en 4.

Ook in 2009 hebben N. Brauner *et al.* [2] gekeken naar het Single Machine Scheduling probleem met verboden start- en eindtijden (SMS-FSE). Ze hebben verschillende eigenschappen en performance ratios gegeven voor *list scheduling* algoritmes. Een list scheduling algoritme maakt gebruik van een lijst met een topologische ordening. De elementen uit deze lijst worden zo vroeg mogelijk in het schema geplaatst, zodat ze nog wel voldoen aan hun eigen beperkingen of de beperkingen van het schema [12]. Verder geven N. Brauner *et al.* in hun paper grenzen voor de beste polynomiale benaderingen.

In 2012 hebben C. Rapine *et al.* [19] onderzoek gedaan naar het SMS-FSE probleem met kleine verboden perioden erin, zoals een nacht of een weekend. Daarna bekijken ze benaderingsalgoritmen waarbij het aantal verboden perioden klein is. In het bijzonder bespreken ze het probleem waarbij er periodieke verboden perioden zijn. Verder bekijken ze list scheduling algoritmes.

In 2013 hebben C. Rapine en N. Brauner [18] een paper gepubliceerd waarin ze het SMS-FSE probleem behandelen. Ze laten zien dat een schema zonder gaten bestaat, als het aantal verboden tijden minder is dan het aantal verschillende werktijden van de taken. Verder vinden ze een algoritme dat in polynomiale tijd een oplossing geeft, als het aantal verboden tijden constant is. We zullen een aantal resultaten uit deze paper gebruiken in Sectie 5.

Ook in 2013 hebben Y. Chen, A. Zhang en Z. Tan [4] ook het SMS-FSE probleem onderzocht. Ze laten zien dat het minimaliseren van de eindtijd van een schema een $\mathcal{NP}$-moeilijk probleem is, zelfs als er maar één verboden interval is dat korter is dan de werktijd van elke taak. Chen, Zhang en Tan geven ook een benaderingsalgoritme voor SMS-FSE. Dit algoritme heeft een worst-case ratio van $\frac{20}{17}$.

In 2016 hebben M. Gabay, C. Rapine en N. Brauner [6] het SMS-FSE probleem onderzocht met een compacte codering. Ze geven een polyomiaal algoritme voor een instantie waarin er meer verschillende taken zijn dan verboden tijden. Verder geven ze een geheeltallig lineair programma voor wanneer er minder verschillende taken zijn dan verboden tijden. Ze laten zien dat het probleem *fixed parameter tractable (FPT)* is ten aanzien van het aantal verboden tijden. We zullen deze paper behandelen in Sectie 5

2.3 Ander onderzoek

Zoals in de vorige twee secties al is genoemd, is er nog veel meer onderzoek gedaan. Er zijn ook nog veel openstaande problemen. M. Mnich en R. van Bevern, 2018 [17] hebben een paper geschreven met een overzicht van 15 openstaande problemen rondom machine scheduling. Voor andere varianten van dit probleem verwijzen we de lezer door naar [13, 3].

3 SMS met verboden starttijden

In deze sectie zullen we het *Single Machine Scheduling with Forbidden Start Times* (SMS-FST) probleem behandelen. Dit doen we aan de hand van de paper "Single machine scheduling with forbidden start times" van J.-C. Billaut en F. Sourd, 2007 [1]. Wanneer een andere bron is gebruikt, staat dit er expliciet bij.

Als eerste zullen we het probleem formeel definiëren.

Definitie 3.1 (Het SMS-FST probleem)**.** Gegeven is een verzameling $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$ met n taken. Deze taken willen we inplannen in het schema van een machine. Elke taak heeft een vaste werktijd p_i en een nog te bepalen starttijd S_i met $i \in \{1, \ldots, n\}$. De totale werktijd van de taken noemen we $P = \sum_{i=1}^{n} p_i$.

We definiëren de verboden starttijden als de verzameling $\mathcal{F} = \{F_1, F_2, \ldots, F_K\}$ met K verboden starttijden. We nemen hierbij zonder verlies van algemeenheid aan dat er geldt $0 < F_1 < F_2 < \ldots < F_K$.

Het probleem is nu als volgt gedefinieerd: Maak een schema van álle taken in $\mathcal{J}$ met een zo kort mogelijke lengte, waarvoor geldt dat geen enkele taak op een verboden starttijd start, oftewel $S_i \notin \mathcal{F}$ voor alle $i \in \{1, \ldots, n\}$. ⊠

Als een schema op een tijdslot $[t, t+1]$ een gat bevat, dan noemen we dit tijdslot een nutteloos tijdslot. In een optimaal schema zitten zo min mogelijk nutteloze tijdsloten. We zullen in een optimaal schema dus nooit een nutteloos tijdslot hebben, zonder dat deze strikt noodzakelijk is, omdat het anders de verboden starttijden schendt. Het eerste wat we nu op kunnen merken, is dat de lengte van elk schema een bovengrens heeft. We bewijzen de volgende stelling. Dit bewijs is van eigen werk.

Stelling 3.2. *Een bovengrens voor de lengte van elk uitvoerbaar schema is $P + |\mathcal{F}|$. Oftewel, een schema heeft maximaal $|\mathcal{F}|$ nutteloze tijdsloten.*

Bewijs. We laten zien dat de bovengrens van een worst-case scenario gelijk is aan $P + |\mathcal{F}|$. Neem een willekeurige instantie van het SMS-FST probleem. Laat $\mathcal{J}$ de verzameling met n taken zijn. Laat $P = \sum_{i=1}^{n} p_i$ de werktijd zijn van alle taken bij elkaar. Laat $\mathcal{F}$ de verzameling met verboden starttijden zijn en laat $K = |\mathcal{F}|$.

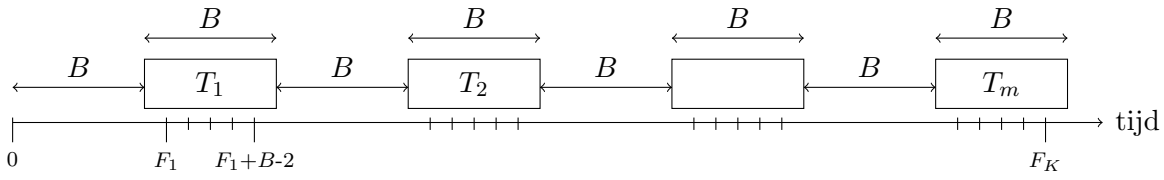
Merk nu op dat er precies K tijden zijn waarop we geen nieuwe taak mogen starten. In het ergste geval moeten we in een schema precies K keer wachten met het starten van een taak. Er geldt dan dan de lengte van het schema $P + K$ is. Hieruit volgt dat een worst-case scenario een maximale lengte heeft van $P + K = P + |\mathcal{F}|$.

We concluderen dat een bovengrens voor de lengte van elk uitvoerbaar schema gelijk is aan $P + |\mathcal{F}|$. ■

Nu we weten dat de lengte van elk schema een bovengrens heeft, is het ook interessant om onszelf af te vragen wat de minimale lengte van een schema is. Een ondergrens voor de lengte van elk schema is duidelijk P, maar dat betekent niet dat elk optimaal schema een lengte P kan hebben. Het zou namelijk zo kunnen zijn dat het onmogelijk is om één of meerdere verboden starttijden te omzijlen. Vanuit deze gedachten willen we bij het SMS-FST probleem graag antwoord geven op de volgende vraag:

Kunnen we een schema maken met precies lengte P?

Dit beslissingsprobleem is een sterk $\mathcal{NP}$-volledig probleem. Dit zullen we bewijzen door een reductie te doen vanuit het sterke $\mathcal{NP}$-volledige probleem 3-PARTITIE. Wat sterke $\mathcal{NP}$-volledigheid precies inhoud, wordt behandeld in Sectie 4. We geven eerst de definitie van het 3-PARTITIE probleem.



Figuur 1: Het schema bij de reductie van 3-PARTITIE naar SMS-FST [1].

Definitie 3.3 (3-PARTITIE)**.** [7, p.96] Gegeven is een eindige verzameling A met $3m$ elementen en een grens $B > 0$. Alle elementen $a \in A$ hebben grootte $s(a)$ zodat $B/4 < s(a) < B/2$ en $\sum_{a \in A} s(a) = mB$.

De beslissingsvraag luidt als volgt: Kan de verzameling A worden verdeeld in m disjuncte verzamelingen $V_1, V_2, \ldots, V_m$ (met elk drie elementen) zodat voor alle $1 \leq i \leq m$ geldt dat $\sum_{a \in V_i} s(a) = B$? ⊠

We zullen nu een idee geven van het bewijs van de volgende stelling. Voor het volledige bewijs, zie [1].

Stelling 3.4. *Het beslissingsprobleem van* SMS-FST *is een sterk $\mathcal{NP}$-volledig probleem.*

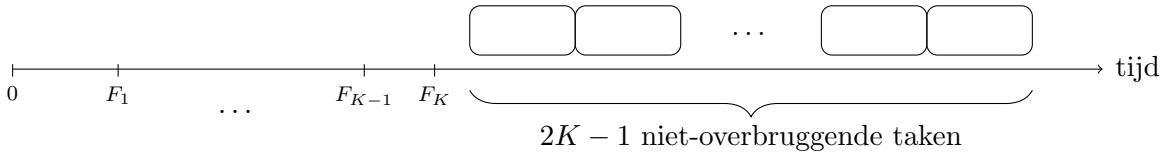
Bewijs idee. Neem een willekeurige instantie van het 3-PARTITIE probleem. We maken nu een instantie van SMS-FST zó dat we steeds een blok met drie taken proberen in te plannen met samen werktijd B. Laat $n = 4m$, waarbij de eerste $3m$ taken overeenkomen met de elementen uit A, met een werktijd gelijk aan $s(a)$. De overige taken $T_1, T_2, \ldots, T_m$ hebben een werktijd gelijk aan B. Laat nu $|\mathcal{F}| = m(B-1)$ en zorg dat de verboden starttijden precies zo zijn als in Figuur 1. De vraag is of we nu een schema kunnen maken met precies lengte $2mB$.

Nu zullen we laten zien dat SMS-FST een oplossing heeft dan en slechts dan als 3-PARTITIE een oplossing heeft. We hebben een instantie van SMS-FST gecreëerd waarvoor het altijd optimaal is om de taken T_1 tot met T_m over de verboden starttijden heen te plannen. De taken die corresponderen met de elementen in A hebben namelijk niet een werktijd die de hele verboden startintervallen kunnen overbruggen. Als we die taken dus wel precies voor een verboden startinterval inplannen, dan houden we een aantal nutteloze tijdsloten over. We plannen dus de eerste $3m$ taken in de gaten van B groot, zie ter referentie Figuur 1. Als dit lukt zonder nutteloze tijdsloten, dan hebben we een oplossing gevonden voor SMS-FST die duidelijk ook een oplossing vormt voor 3-PARTITIE. Omgekeerd, is het duidelijk dat een oplossing van 3-PARTITIE ook een oplossing is voor de gegeven instantie van SMS-FST.

We kunnen nu concluderen dat er dus wel moet gelden dat SMS-FST minstens net zo moeilijk is als 3-PARTITIE. Dus volgt er dat SMS-FST ook sterk $\mathcal{NP}$-volledig is. ■

We weten nu dus dat SMS-FST een sterk $\mathcal{NP}$-volledig probleem is. Toch zouden we graag willen vaststellen in welke gevallen het mogelijk is om een schema van lengte P te maken. J.-C. Billaut en F. Sourd [1] hebben eerst de gevallen $K = 1$ en $K = 2$ behandeld. Deze gevallen zijn op te lossen met gevalsonderscheiding. De algoritmes die worden beschreven, hebben een looptijd van $O(n)$. We weten dus dat we een SMS-FST probleem met $K \leq 2$ op kunnen lossen in lineaire tijd.

Als K groter wordt, is gevalsonderscheiding niet meer te doen. We kunnen dan een schema construeren door gebruik te maken van een dynamisch programmeeralgoritme. J.-C. Billaut en F. Sourd [1] hebben aangetoont dat er altijd een optimaal schema met lengte P kan worden gemaakt als er tenminste $2K(K+1)$ paarsgewijs verschillende taken zijn. Dat wil zeggen: alle taken hebben een verschillende werktijd. Voordat we dit bewijzen, voeren we twee nieuwe definities in.



Figuur 2: Schema bij Lemma 3.7.

Definitie 3.5 (Actief pseudo-schema)**.** Een *actief pseudo-schema* is een schema zonder nutteloze tijdsloten die mogelijk een aantal taken heeft die op een verboden starttijd starten. ⊠

Definitie 3.6 (Overbruggende taak)**.** Een taak J_j wordt een *overbruggende taak* genoemd als er een k bestaat zodat $S_j < F_k \leq S_j + p_j$. In woorden betekent dit dat een taak een verboden starttijd kruist, oftewel als deze taak vóór een verboden starttijd begint en op of na een verboden starttijd eindigt. Een niet-overbruggende taak kruist dus geen verboden starttijd. ⊠

In het bewijs maken ze daarnaast gebruik van onderstaand lemma en gevolg.

Lemma 3.7. *Laat $\mathcal{J}$ een instantie met taken zijn waarin de taken paarsgewijs verschillend zijn, dat wil zeggen dat alle taken een verschillende werktijd hebben. Als er een actief pseudo-schema bestaat waarin de laatste $2K-1$ taken niet-overbruggende taken zijn, dan bestaat er een schema met lengte P. (Zie ter ondersteuning Figuur 2.)*

Bewijs idee. Laat $\mathcal{J}$ een instantie zijn met paarsgewijs verschillende taken. We noemen voor het gemak de verzameling van de laatste $2K-1$ niet-overbruggende taken $\mathcal{S}$. We bekijken de eerste taak, zeg J_j, die eindigt op een verboden starttijd. Stel $t = S_j$. Aangezien de laatste $2K-1$ taken allemaal een andere werktijd hebben, kunnen we de eerste taak J_{i_1} uit $\mathcal{S}$ pakken, zodat $t + p_{i_1}$ geen verboden starttijd is. We plakken taak J_{i_1} in het schema vóór taak J_j. Als er nu geldt dat $t + p_{i_1} + p_j \in \mathcal{F}$, dan plakken we nog een taak J_{i_2} voor J_j. Omdat er precies K verboden starttijden zijn, hoeven we dit maximaal K keer te doen. We hebben in het schema in elk geval tot en met taak J_j geen enkele taak laten starten op een verboden starttijd. We herhalen bovenstaande stappen voor de overgebleven taken die starten op een verboden starttijd. We hoeven dit maximaal K keer te herhalen. Omdat alle taken achter elkaar worden geplakt, zonder nutteloze tijdsloten, levert dit een schema op met lengte P. ■

Gevolg 3.8. *Laat $\mathcal{J}$ een instantie met taken zijn waarin de taken paarsgewijs verschillend zijn, dat wil zeggen dat alle taken een verschillende werktijd hebben. Als er een actief pseudo-schema bestaat waarin een deelrij van $2K-1$ niet-overbruggende taken voorkomt, dan bestaat er een schema met lengte P.*

Bewijs idee. Het is duidelijk dat we vóór de deelrij van $2K-1$ niet-overbruggende taken alle verboden starttijden kunnen mijden, door het gebruik van Lemma 3.7. Als we het eerste deel van het schema kloppend hebben gemaakt, dan bekijken we het schema vanaf het einde en dan passen we opnieuw Lemma 3.7 toe. ■

Nu kunnen we Stelling 3.9 bewijzen. We gebruiken hierbij de volgende conventie: Verschillende taken, zijn taken die een verschillende werktijd hebben.

Stelling 3.9. *Als er ten minste $L = 2K(K+1)$ verschillende taken zijn, dan kunnen we een schema maken met lengte P.*

Bewijs idee. Laat $\mathcal{J}$ de verzameling met alle taken zijn. Definieer $\mathcal{J}^\star$ als maximale deelverzameling van $\mathcal{J}$ zodat $\mathcal{J}^\star$ van elke verschillende taak er precies één bevat. Er geldt dan dat $|\mathcal{J}^\star| \geq L$. We gaan nu de taken uit $\mathcal{J} \setminus \mathcal{J}^\star$ inplannen tot vóór F_1 en we bekijken twee gevallen. We maken hierbij gebruik van deelrijen. Een deelrij is een verzameling van opeenvolgende niet-overbruggende taken tussen twee overbruggende taken in.

1. Alle taken uit $\mathcal{J} \setminus \mathcal{J}^\star$ zijn ingepland vóór F_1.

   We hebben nu dus nog $|\mathcal{J}^\star| \geq L$ taken over om in te plannen en we hebben nog steeds K verboden starttijden. We maken een actief pseudo-schema. Merk op dat er maximaal $K+1$ deelrijen zijn en minstens $2K(K+1)$ taken. Er volgt dat de langste deelrij minstens $2K-1$ taken bevat. Vanwege Gevolg 3.8 kunnen we het schema kloppend maken.

2. Niet alle taken uit $\mathcal{J} \setminus \mathcal{J}^\star$ zijn ingepland vóór F_1.

   De tijd waarop de laatste taak uit het schema eindigt, noemen we t. We pakken nu een taak, zeg $J_1 \in \mathcal{J}^\star$ die over F_1 heen kan worden gepland, zodat $t + p_1 \notin \mathcal{F}$ en $t + p_1 > F_1$. Als er niet zo'n taak bestaat, dan plakken we eerst een kortere taak, zeg $J_2 \in \mathcal{J}^\star$ ervoor. Aangezien $|\mathcal{J}^\star| \geq L$ zijn deze taken altijd te vinden. We hebben nu een kloppend schema die klaar is na F_1. Er zijn nu $L - 2 > 2(K-1)K$ verschillende niet-ingeplande taken over en maximaal $K-1$ verboden starttijden. Met inductie kunnen we het schema afmaken.

■

We kunnen met $L \geq 2K(K+1)$ verschillende taken dus inderdaad altijd een schema maken van lengte P. Nu rest de vraag hoe we dan een schema opstellen voor $M \leq L-1$ verschillende taken. J.-C. Billaut en F. Sourd [1] hebben hiervoor een dynamisch programmeer (DP) algoritme gemaakt. Dit algoritme gaat als volgt.

We gebruiken voor een instantie de volgende notatie: er zijn α_1 taken met werktijd p_1, α_2 taken met werktijd p_2, ... en α_M taken met werktijd p_M. Laat g het aantal toegestane nutteloze tijdsloten zijn in het te maken schema. Uit Stelling 3.2 volgt er dat $0 \leq g \leq K$. We definiëren de boolean waardes $B(i_1, i_2, \ldots, i_M, g)$ die `true` zijn dan en slechts dan als er geldt dat we een schema kunnen maken met i_1 taken met werktijd p_1, i_2 taken met werktijd p_2, ... en i_M taken met werktijd p_M zodat de lengte van het schema kleiner dan of gelijk is aan $T = g + \sum_{j=1}^{M} i_j p_j$.

In het optimale geval voldoet het schema aan één van de onderstaande gevallen.

1. Het laatste tijdslot, $[T-1, T]$, is nutteloos. In dit geval geldt er dat $B(i_1, i_2, \ldots, i_M, g) =$ `true` dan en slechts dan als $B(i_1, i_2, \ldots, i_M, g-1) =$ `true`.

2. Het laatste tijdslot, $[T-1, T]$, is niet nutteloos. Laat J_j de taak zijn die eindigt op T. Hieruit volgt dat $T - p_j \notin \mathcal{F}$. Oftewel, er geldt dat $B(i_1, i_2, \ldots, i_M, g) =$ `true` dan en slechts dan als $B(i_1, \ldots, i_j - 1, \ldots, i_M, g) =$ `true`.

We kunnen hieruit de volgende recurrente betrekking opstellen:

$$B(i_1, i_2, \ldots, i_M, g) = B(i_1, i_2, \ldots, i_M, g-1) \vee \left(\bigvee_{j \mid T - p_j \notin \mathcal{F}} B(i_1, \ldots, i_j - 1, \ldots, i_M, g) \right) \quad (1)$$

met $T = g + \sum_{j=1}^{M} i_j p_j$, $0 \leq g \leq K$ en $0 \leq i_j \leq \alpha_j$ voor alle $0 \leq j \leq M$.

De lengte van een kortste schema wordt gegeven door $P + g^\star$, waarbij $g^\star$ staat voor het kleinste (gehele) getal zodat $B(\alpha_1, \alpha_2, \ldots, \alpha_M, g^\star) = \texttt{true}$.

We zullen nu de looptijd van het bovenstaande DP-algoritme bepalen. Het berekenen van $B(i_1, i_2, \ldots, i_M, g)$ kan in $O(K + M)$. Omdat er geldt dat

$$M \leq L - 1 = 2K(K+1) - 1 = 2K^2 + 2K - 1 \tag{2}$$

volgt hieruit dat $O(K+M) = O\left(K^2\right)$. De herhaalde disjunctie in vergelijking (1) controleert voor alle $0 \leq g \leq K$ en voor alle combinaties van taken 1 tot met M de waarde van B. Dit geeft ons een looptijd van $O\left(K \cdot \prod_{j=1}^{M} \alpha_j\right)$. Omdat er geldt dat $\alpha_j \leq n$ voor alle $j \in \{1, \ldots, M\}$ volgt er dat

$$O\left(K \cdot \prod_{j=1}^{M} \alpha_j\right) = O\left(Kn^M\right).$$

Samen met het berekenen van de waarden van B en vergelijking (2) volgt er nu dat de totale looptijd van het DP-algoritme gelijk is aan

$$O\left(K^2 \cdot Kn^M\right) = O\left(K^3 n^{2K^2+2K-1}\right).$$

We kunnen hieruit concluderen dat het algoritme polynomiaal is als K een constante is die niet in de invoer wordt meegegeven.

4 SMS met een verboden startinterval

Een speciaal geval van het SMS-FST probleem, is het *Single Machine Scheduling with a Forbidden Start Interval* probleem (SMS-FSI). We geven hieronder de definitie.

Definitie 4.1 (Het SMS-FSI probleem)**.** Gegeven is een verzameling $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$ met n taken. Deze taken willen we inplannen in het schema van een machine. Elke taak heeft een vaste werktijd p_i en een nog te bepalen starttijd S_i met $i \in \{1, \ldots, n\}$. De totale werktijd van de taken noemen we $P = \sum_{i=1}^{n} p_i$.

We definiëren de verboden starttijden als het discrete interval $\mathcal{F} = [F_1, F_K]$ met K elementen. Definieer $|\mathcal{F}| = K$. We nemen hierbij zonder verlies van algemeenheid aan dat $0 < F_1$.

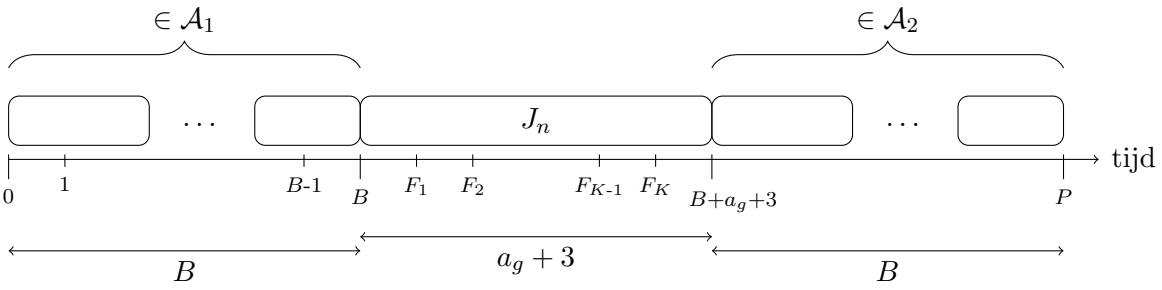
Het probleem is nu als volgt gedefinieerd: Maak een schema van álle taken in $\mathcal{J}$ met een zo kort mogelijke lengte, waarvoor geldt dat geen enkele taak op een verboden starttijd start, oftewel $S_i \notin \mathcal{F}$ voor alle $1 \leq i \leq n$. ⊠

De beslissingsvariant van dit probleem luidt hetzelfde als die van SMS-FST: Kunnen we een schema maken met precies lengte P? We zullen laten zien dat ook de beslissingsvariant van SMS-FSI $\mathcal{NP}$-volledig is. Het bewijs van deze stelling is gebaseerd op het bewijs van Theorem 2.1 uit de paper van J.-C. Billaut en F. Sourd [1].

Stelling 4.2. *De beslissingsvariant van het* SMS *probleem met precies één interval van verboden starttijden is $\mathcal{NP}$-volledig.*

Bewijs. Merk als eerste op dat het probleem in de klasse $\mathcal{NP}$ zit. We kunnen in elk geval een eventuele oplossing in polynomiale tijd controleren.

Om de stelling te bewijzen, maken we gebruik van het probleem PARTITIE, dat $\mathcal{NP}$-volledig is. De beslissingsvariant van PARTITIE is als volgt gedefinieerd.



Figuur 3: De instantie van het SMS-FSI probleem bij Stelling 4.2.

Gegeven is een multiset $\mathcal{A} = \{a_1, \ldots, a_m\}$ met $a_l \in \mathbb{N}$ voor $l \in \{1, \ldots, m\}$. We maken een partitie van de elementen van $\mathcal{A}$ in twee multisets $\mathcal{A}_1$ en $\mathcal{A}_2$. Zeg dat alle elementen in $\mathcal{A}$ optellen tot $2B$. We willen graag antwoord op de vraag:

$$\text{Bestaat er een partitie zó dat er geldt } \sum_{a_i \in \mathcal{A}_1} a_i = \sum_{a_j \in \mathcal{A}_2} a_j = B \text{ ?}$$

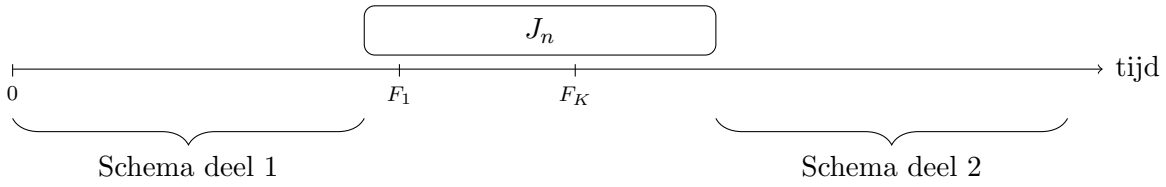
Zij verder gegeven dat a_g een grootste element is in $\mathcal{A}$. Deze hoeft niet uniek gedefinieerd te zijn, in dat geval kiezen we één van deze grootste elementen. Merk op dat $a_g < B$.

We maken nu een instantie van het SMS-FSI probleem. Laat $\mathcal{J}$ de verzameling van taken zijn. Laat $|\mathcal{J}| = n = m + 1$. De eerste m elementen komen overeen met de elementen uit $\mathcal{A}$: deze elementen hebben een werktijd gelijk aan de waarde van a_l, oftewel voor taak J_l geldt $p_l = a_l$ met $l \in \{1, \ldots, m\}$. Zij voor de overgebleven taak J_n dat $p_n = a_g + 3$. Laat $\mathcal{F}$ het discrete interval van $[F_1, F_K]$ zijn met $F_1 = B + 1$ en $F_K = B + a_g + 2$. Oftewel, $|\mathcal{F}| = a_g + 1$. De vraag is nu of we een schema kunnen maken met precies lengte $P = 2B + p_n$.

Zie Figuur 3 ter ondersteuning van onderstaand bewijs. We laten zien dat SMS-FSI een oplossing heeft dan en slechts dan als PARTITIE een oplossing heeft. We willen de taken zó inplannen dat er geen nutteloze tijdsloten zijn. Dit betekent dat we het tijdschema zó willen maken, dat deze klaar is op $P = 2B + p_n$. Er volgt meteen dat $S_n = F_1 - 1$, oftewel we plannen de langste taak over het verboden startinterval heen. We weten namelijk dat de langste taak in $\mathcal{J} \setminus \{J_n\}$ een werktijd heeft van $a_g < a_g + 1 = |\mathcal{F}|$. Het is dus altijd optimaal om taak J_n met $p_n = a_g + 3$ over het interval $\mathcal{F}$ van verboden starttijden te plannen. Laat J_n dus beginnen op tijdstip B. Deze taak zal dan eindigen op $B + a_g + 3 \notin [B + 1, B + a_g + 2] = [F_1, F_K]$.

We houden nu twee intervallen over van lengte B waar we de taken uit $\mathcal{J} \setminus \{J_n\}$ in gaan plannen. Als het schema eindigt op tijdstip P en dus geen nutteloze tijdsloten bevat, dan kunnen we een oplossing voor PARTITIE geven. Het deel van het schema dat voor het verboden startinterval is gepland, heeft precies een lengte gelijk aan B, net zoals het deel van het schema dat na het verboden startinterval is gepland. Laat $\mathcal{A}_1$ de multiset zijn met de elementen uit $\mathcal{A}$ die corresponderen met de taken uit het eerste deel van het schema. Laat $\mathcal{A}_2$ de multiset zijn met de elementen uit $\mathcal{A}$ die corresponderen met de taken uit het tweede deel van het schema. We hebben een oplossing voor PARTITIE gevonden.

Andersom, als we een oplossing voor PARTITIE hebben, dan kunnen we de taken met werktijden die corresponderen met de elementen uit multiset $\mathcal{A}_1$ inplannen vóór het verboden startinterval, en de taken met werktijden die corresponderen met de elementen uit multiset $\mathcal{A}_2$ inplannen na het verboden startinterval. We plannen taak J_n zoals hierboven over het verboden startinterval. We hebben nu een oplossing voor onze instantie van SMS-FSI, zonder nutteloze tijdsloten.



Figuur 4: De opbouw van een schema voor het SMS-FSI algortime.

We concluderen dat het single machine scheduling probleem met een interval van verboden starttijden minstens net zo moeilijk is als PARTITIE. Hieruit volgt dat het beslissingsprobleem $\mathcal{NP}$-volledig is. ■

Het SMS-FSI probleem is niet sterk $\mathcal{NP}$-volledig, net als PARTITIE, in tegenstelling tot SMS-FST en 3-PARTITIE. We kunnen voor SMS-FSI een pseudo-polynomiaal algoritme maken. We zullen in Sectie 4.1 eerst het algoritme geven, daarna bespreken we waarom deze pseudo-polynomiaal is en wat dat inhoudt.

4.1 Een algoritme voor SMS-FSI

Appendix A bevat een dynamisch programmeer (DP) algoritme, geschreven in C#, dat een optimale oplossing geeft voor het SMS-FSI probleem. Dit algoritme is van eigen werk, waarvan een deel gebaseerd is op het DP-algoritme van PARTITIE [7, p.90-91]. Hieronder zullen we de werking en de correctheid van het algoritme uitleggen en aantonen.

We gaan een schema maken voor een gegeven instantie van het SMS-FSI probleem. Merk als eerste op dat we altijd een optimaal schema kunnen maken als er geldt dat $P - p_n < F_1$. We gaan er in het vervolg dus vanuit dat dit niet het geval is. Het totale schema gaan we opbouwen in drie stukken, zoals aangegeven is in Figuur 4. Het idee van het algoritme is als volgt: we gebruiken het DP-algoritme van PARTITIE om te kijken hoeveel taken we vóór het verboden startinterval kunnen inplannen. Het schema van taken die we vóór het startinterval inplannen, noemen we schema deel 1. We plannen een langste taak vlak voor het verboden startinterval. Daarachteraan plakken we alle taken die nog niet zijn ingepland. Dit deel noemen we schema deel 2. We zullen aantonen dat het altijd optimaal is om de langste taak vlak voor het verboden startinterval te laten beginnen. De langste taak hoeft niet per se uniek gedefinieerd te zijn. In dat geval pakken we een van de taken met de langste werktijd. Deze taak noemen we de langste taak.

Stelling 4.3. *De langste taak direct vóór het verboden startinterval laten beginnen geeft altijd een optimaal schema.*

Bewijs. Neem een instantie van het SMS-FSI probleem, waarvoor we een optimaal schema hebben. Laat $\mathcal{J} = \{J_1, \ldots, J_n\}$ de taken zijn, waarin J_n de langste taak is. Voor elke taak is p_i zijn werktijd en S_i zijn starttijd. Laat $P + g$ de lengte van het optimale schema zijn, met $g \in \{0, 1, \ldots, K\}$.

Als in het optimale schema J_n over het verboden startinterval heen zit, dan zijn we klaar. We onderscheiden daarom de volgende twee gevallen: J_n zit in schema deel 1 en J_n zit in schema deel 2. Laat J_v de taak zijn die over het verboden startinterval heen zit. Neem hierbij aan dat $p_v < p_n$, anders is J_v ook een langste taak en dan zijn we meteen klaar. We zullen laten zien dat het omruilen van taak J_v met taak J_n ook een optimaal schema oplevert. We maken onderscheid tussen de starttijden S_v en S_n uit het optimale schema en de starttijden S'_v en S'_n in ons geconstrueerde schema.

1. J_n zit in schema deel 1.

   Laat
   $$\Delta = p_n - p_v > 0 \tag{3}$$
   het verschil in werktijd zijn van taak J_v en J_n. Merk op dat Δ altijd groter is dan 0, omdat taak J_n de langste taak is. Laat nu taak J_v beginnen op $S'_v = S_n$. Alle taken na J_v in schema deel 1 sluiten we aan op J_v, zodat er geen nutteloze tijdsloten ontstaan. De starttijd van deze taken wordt dus vervroegd met Δ. We kunnen taak J_n achter de laatste taak van schema deel 1 plakken. Taak J_n begint nu op $S'_n = S_v - \Delta$ en is nu (met gebruik van vergelijking (3)) klaar op:
   $$S'_n + p_n = S_v - \Delta + p_n = S_v - \Delta - \Delta + p_v = S_v + p_v.$$
   Aan schema deel 2 verandert er niks. Hieruit volgt dat de lengte van het geconstrueerde shema hetzelfde is als het gegeven optimale schema. De lengte van het geconstrueerde schema is dus gelijk aan $P + g$. We hebben een optimaal schema gecreëerd waarin J_n direct voor het verboden startinterval begint.

2. J_n zit in schema deel 2.

   Zij $\Delta = p_n - p_v > 0$. Laat nu taak J_n beginnen op $S'_n = S_v$, vóór het verboden startinterval. Schuif de taken uit schema deel 2 die in het gegeven optimale schema voor taak J_n begonnen met Δ naar achteren. Plak taak J_v in het overgebleven gat. Dit gat heeft precies grootte $p_n - \Delta = p_n - (p_n - p_v) = p_v$. Met eventuele taken achter J_v gebeurt niks, net als de taken in schema deel 1. De lengte van het geconstrueerde schema is $P + g$. We hebben een optimaal schema gecreëerd waarin J_n direct voor het verboden startinterval begint.

We concluderen dat de langste taak direct vóór het verboden startinterval laten beginnen inderdaad altijd een optimaal schema oplevert. ■

Nu we zeker weten dat we de langste taak altijd over het verboden startinterval kunnen plannen, kunnen we uitleggen hoe het algoritme in elkaar zit. We kunnen het algoritme opdelen in verschillende stukjes: de invoer, het maken en vullen van de DP-tabel, het berekenen van de minimale lengte van het schema en het construeren van de oplossing.

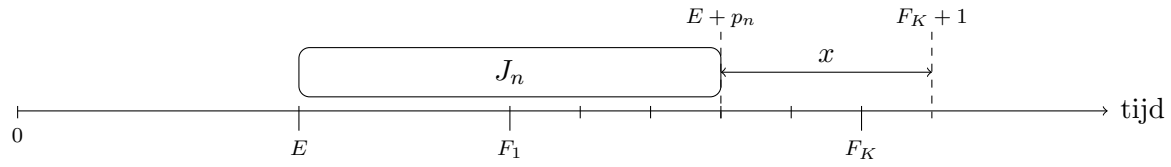
De invoer

De invoer bevat de volgende gegevens:
- het aantal taken,
- de werktijden van de taken,
- de eerste verboden starttijd en
- de laatste verboden starttijd.

De taken worden opgeslagen in een geïndexeerde verzameling $\mathcal{J}$.

Het maken en vullen van de DP-tabel

We zoeken in $\mathcal{J}$ de grootste taak J_n (of een van de grootste taken als deze niet uniek is) en we zetten deze achteraan in $\mathcal{J}$. We willen deze taak over het verboden startinterval plannen, daarom nemen we deze taak niet mee in onderstaand DP-algoritme. Verder voegen we als eerste taak in $\mathcal{J}$ een taak J_0 toe met $p_0 = 0$. Voor het DP-algoritme dat we vervolgens uit willen voeren, gebruiken we een boolean functie $b(t, e)$. Hierin geeft t de taken weer en e staat voor de eindtijden. De taken lopen van taak 0 tot met taak $n - 1$. De eindtijden lopen



Figuur 5: De hoeveelheid nutteloze tijdsloten in een schema van het SMS-FSI algoritme.

van 0 tot en met $F_1 - 1$. Als voor een t en e geldt dat $b(t, e) = \texttt{true}$, dan betekent dit dat we eindtijd e kunnen bereiken door een deelverzameling van de taken 1 tot en met t in te plannen.

We vullen de DP-tabel door gebruik te maken van onderstaande recurrente betrekking. Voor taak t $(0 \leq t \leq n-1)$ met werktijd p_t en eindtijd e $(0 \leq e \leq F_1 - 1)$ geldt er dat

$$b(t,e) = \begin{cases} \texttt{true} & \text{als } e = 0, \quad (4a) \\ \texttt{false} & \text{als } t = 0 \text{ en } e > 0, \quad (4b) \\ b(t-1,e) & \text{als } t, e > 0 \text{ en } p_t > e, \quad (4c) \\ b(t-1,e) \vee b(t-1,e-p_t) & \text{als } t, e > 0 \text{ en } p_t \leq e. \quad (4d) \end{cases}$$

Het is duidelijk dat deze recurrente betrekking volledig is. Dat de recurrente betrekking ook correct is, volgt uit de volgende analyse. Voor geval (4a) geldt er dat we een schema met eindtijd nul kunnen maken door geen taken in te plannen. Voor geval (4b) geldt er dat we een eindtijd groter dan 0 nooit kunnen bereiken met J_0 die werktijd 0 heeft. Voor $t, e > 0$ komen we in geval (4c) of (4d). In zowel geval (4c) als geval (4d) wordt gecontroleerd of met de voorgaande taken (taak 1 tot en met taak $t-1$) eindtijd e bereikt kan worden. Lukt dat niet en geldt er dat $p_t \leq e$: kunnen we dan met voorgaande taken plus deze taak eindtijd e bereiken? Dit is ook wel equivalent met de vraag: kunnen we met de voorgaande taken eindtijd $e - p_t$ bereiken?

Door de tabel voor alle e voor $t = 1$ tot en met $t = n - 1$ te doorlopen, wordt de hele tabel gevuld.

De oplossing die dit DP-algoritme als uitkomst geeft, is optimaal. Uit het algoritme volgt hoeveel taken we kunnen inplannen vóór het verboden startinterval. We plakken de langste taak over het verboden startinterval, wat een optimale oplossing geeft vanwege Stelling 4.3. Alle overgebleven taken komen aan het einde. We kunnen nu de lengte van een optimaal schema aflezen.

Het berekenen van de lengte van het schema

Als het DP-algoritme de hele tabel gevuld heeft, dan kunnen we aflezen welke eindtijd we kunnen halen voor schema deel 1. Deze eindtijd noemen we E. Om de waarde van E te bepalen, kijken we eerst of $b(n-1, F_1-1) = \texttt{true}$. Als dit geldt, dan geldt er dus $E = F_1 - 1$.

Als $b(n-1, F_1-1) = \texttt{false}$, dan gaan we kijken voor welke $0 \leq i \leq F_1 - 1$ er geldt dat $b(n-1, i) = \texttt{true}$. We laten hierbij i lopen van $F_1 - 1$ tot 0. De eerste i die een $\texttt{true}$ oplevert, stellen we gelijk aan E.

Nu we de lengte van schema deel 1 hebben, kunnen we de lengte van het totale schema berekenen. We plakken taak J_n achter schema deel 1. Als er nu geldt dat $E + p_n > F_K$, dan kunnen we een schema maken van lengte P. Anders krijgt het schema een lengte van $P + x$, met $x = F_K + 1 - (E + p_n)$. Dit staat voor P plus de hoeveelheid nutteloze tijdsloten die ontstaan. Voor een visualisatie van x zie Figuur 5.

Het schema construeren

We kunnen schema deel 1 construeren door de DP-tabel achterstevoren door te lopen. We beginnen met $b(t, E)$ en we zoeken een y zodat er geldt dat $b(t-y, E) =$ `true` en $b(t-y-1, E)$ = `false`. Deze taak $j = t - y$ voegen we toe aan schema deel 1 en vervolgens zoeken we verder vanaf $b(j-1, E-p_j)$. Dit blijven we herhalen net zolang totdat we taak 0 hebben toegevoegd. We hebben nu schema deel 1 compleet. Het totale schema bestaat uit schema deel 1, gevolgd door taak J_n, gevolgd door alle taken die nog niet zijn ingepland.

4.2 De looptijd van het algoritme voor SMS-FSI

We hebben nu een algoritme om een optimaal schema te vinden voor een instantie van SMS-FSI, maar wat heeft dit algoritme voor looptijd? De looptijd van het beschreven algoritme zullen we nu analyseren.

De eventuele invoer lezen en de langste taak zoeken kan in $O(n)$. De DP-tabel vullen kost $O(n(F_1 - 1)) = O(n \cdot F_1)$ tijd. Het construeren van een schema kan in $O(n)$. Het eventueel printen van een schema kost ook $O(n)$. Hieruit volgt dat de looptijd van het algoritme gelijk is aan $O(n \cdot F_1)$.

Dit lijkt polynomiaal te zijn, maar F_1 is geen constante. De invoergrootte van dit getal is $\log(F_1)$ bits. Hieruit volgt dat het getal $F_1 = 2^{\log_2(F_1)}$ exponentieel is in de grootte van de invoer. De looptijd van het algoritme is dus exponentieel. Echter, als we de invoer unair coderen, dan is de grootte van het getal lineair in de grootte van de invoer. Dus dan is het algoritme polynomiaal. Een algoritme dat deze eigenschappen heeft, wordt een *pseudo-polynomiaal algoritme* genoemd.

We kunnen nu ook uitleggen wat het verschil is tussen $\mathcal{NP}$-volledig en sterk $\mathcal{NP}$-volledig. We bekijken dan de looptijd en de invoer. Als een $\mathcal{NP}$-volledig probleem geen numerieke invoer heeft, dan kunnen we bovenstaand trucje, kijken naar de invoergrootte van getallen, niet toepassen. We kunnen dan dus ook geen pseudo-polynomiaal algoritme vinden. Zo'n probleem is gewoon $\mathcal{NP}$-volledig.

Van de $\mathcal{NP}$-volledige problemen die wél numerieke invoer bevatten, kunnen we twee gevallen onderscheiden. Als een $\mathcal{NP}$-volledig probleem nog steeds $\mathcal{NP}$-volledig is als het numerieke invoer heeft die begrensd is door een polynoom, dan noemen we het probleem *sterk $\mathcal{NP}$-volledig*. Er is in dit geval geen pseudo-polynomiaal algoritme te maken, behalve als er geldt dat $\mathcal{P} = \mathcal{NP}$. We hebben hier twee voorbeelden van gezien, namelijk SMS-FST en 3-PARTITIE. De grootte van de getallen van de invoer is bij deze twee problemen niet relevant voor de looptijd van het algoritme. Om hier een beeld van te krijgen, kunnen we kijken naar een sorteeralgoritme. Het maakt niet uit of we een array met n getallen onder de duizend willen sorteren, of dat we een array met n getallen boven de miljard willen sorteren. Het algoritme hoeft nog steeds maar n getallen te sorteren. Dit is heel anders als we kijken naar bijvoorbeeld de $\mathcal{NP}$-volledige problemen PARTITIE en SMS-FSI. Bij PARTITIE kijken we of we een deelverzameling kunnen vinden die optelt tot lengtes $1, 2, \ldots, B$. Als B klein is, moet het algoritme veel minder stappen doen dan als B heel groot is. Als de grootte van B begrensd is door een polynoom, dan kunnen we voor PARTITIE een pseudo-polynomiaal algoritme maken. Zulke problemen zijn $\mathcal{NP}$-volledig en niet sterk $\mathcal{NP}$-volledig.

5 SMS met verboden start- en eindtijden

We hebben nu een probleem gezien met alleen verboden starttijden. In de praktijk is het soms ook zo dat een taak alleen op tijden mag eindigen waarop er iemand aanwezig is om dit te assisteren. We bekijken daarom nu het *Single Machine Scheduling with Forbidden*

Start and End times (SMS-FSE) probleem. Dit doen we aan de hand van de paper "High-multiplicity scheduling on one machine with forbidden start and completion times" van M. Gabay, C. Rapine en N. Brauner, 2016 [6]. Hiervoor is de paper "A polynomial time algorithm for makespan minimization on one machine with forbidden start and completion times" van C. Rapine en N. Brauner, 2013 [18] als basis gebruikt. Wij zullen de eerstgenoemde paper nader bespreken, waarbij we een aantal resultaten zullen gebruiken uit de tweede genoemde paper.

Het SMS-FSE probleem ziet er hetzelfde uit als het SMS-FST probleem, behalve dat hierbij de verboden starttijden nu verboden start- en eindtijden zijn. C. Rapine en N. Brauner [18] hebben bewezen dat dit optimaliseringsprobleem sterk $\mathcal{NP}$-moeilijk is. Wij kunnen nu de volgende stelling bewijzen. Dit bewijs is van eigen werk.

Stelling 5.1. *De beslissingsvariant van* SMS-FSE *is een sterk $\mathcal{NP}$-volledig probleem.*

Bewijs. We weten al dat de optimaliseringsvariant van SMS-FSE sterk $\mathcal{NP}$-moeilijk is. Dus we hoeven alleen nog te laten zien dat de beslissingsvariant in de klasse $\mathcal{NP}$ zit. De beslissingsvariant luidt: Kunnen we een schema maken met precies lengte P? We moeten laten zien dat we een oplossing kunnen opslaan in polynomiale ruimte en dat we de correctheid van deze oplossing kunnen controleren in polynomiale tijd.

Het is duidelijk dat een oplossing van het SMS-FSE probleem hetzelfde kan worden opgeslagen als een oplossing van het SMS-FST probleem. We weten dat het beslissingsprobleem van SMS-FST $\mathcal{NP}$-volledig is en dus dat het probleem in de klasse $\mathcal{NP}$ zit. We kunnen dus concluderen dat een oplossing van het SMS-FSE probleem kan worden opgeslagen in polynomiale ruimte.

Als we een oplossing hebben gevonden, kunnen we voor elke taak controleren of deze niet start en/of eindigt op een verboden tijd en of de taak aansluit op de vorige taak. Dit is duidelijk in polynomiale tijd te controleren. Er volgt dat de beslissingsvariant van het SMS-FSE probleem in de klasse $\mathcal{NP}$ zit. ■

M. Gabay, C. Rapine en N. Brauner [6] hebben het SMS-FSE probleem geherformuleerd met gebruik van een *high-multiplicity encoding*. Hierbij slaan we niet meer alle taken $1, \ldots, n$ op, maar we slaan van elke verschillende taak zijn werktijd en multipliciteit op. De definitie van het probleem is nu als volgt.

Definitie 5.2 (Het SMS-FSE probleem met een high-multiplicity encoding)**.** Gegeven is een verzameling $\mathcal{J}$ met taken. Zij s het aantal verschillende taken; twee taken zijn verschillend als hun werktijd verschillend is. We noteren de taken uit $\mathcal{J}$ met behulp van twee vectoren:

$p = (p_1, \ldots, p_s)$ waarbij p_i staat voor de werktijd, en
$m = (m_1, \ldots, m_s)$ waarbij m_i staat voor de hoeveelheid taken met werktijd p_i.

We nemen hierbij zonder verlies van algemeenheid aan dat $p_1 > p_2 > \cdots > p_s$. We zeggen dat een taak van type i is als deze taak werktijd p_i heeft. Zij $n = \sum_{i=1}^{s} m_i$ het totaal aantal taken in $\mathcal{J}$. Laat $P = \sum_{i=1}^{s} m_i p_i$ de totale werktijd zijn van alle taken bij elkaar.

We definiëren de verboden start- en eindtijden als de verzameling $\mathcal{F} = \{F_1, F_2, \ldots, F_K\}$ met K verboden tijden. We nemen hierbij zonder verlies van algemeenheid aan dat er geldt $0 < F_1 < F_2 < \ldots < F_K$. *Opmerking: Als $t = 0 \in \mathcal{F}$, dan volgt er meteen dat er een nutteloos tijdslot ontstaat op $t = 0$. We passen dan de instantie aan, zodat het schema begint op de eerstvolgende tijd t zodat $t \notin \mathcal{F}$. Deze transformatie kunnen we altijd toepassen [18].*

De taken uit $\mathcal{J}$ willen we inplannen in het schema van een machine. We houden voor elke verboden tijd bij hoeveel taken er van elke soort zijn ingepland. Dit doen we met behulp van $K+1$ vectoren en K paren:

$$\begin{aligned} &\left(m_1^j, \ldots, m_s^j\right) && \text{met } j \in \{1, \ldots, K+1\} \text{ en} \\ &(J_\ell, S_\ell) && \text{met } \ell \in \{1, \ldots, K\}. \end{aligned}$$

Hierbij staat m_i^j voor de hoeveelheid taken met werktijd p_i die zijn ingepland tussen F_{j-1} en F_j. We zien hierbij F_0 als de niet-verboden tijd 0 en F_{K+1} als tijd na de eindtijd van het schema: m_i^{K+1} staat voor hoeveel taken van type i er na de laatste verboden tijd zijn ingepland. Een paar (J_ℓ, S_ℓ) staat voor een taak J_ℓ van type i met werktijd p_i (voor een nader te bepalen i) en met starttijd S_ℓ, zodat taak J_ℓ de verboden tijd F_ℓ overbrugt.

We zeggen dat een taak j klaar is op tijdstip t als $S_j + p_j \leq t$.

We noteren een instantie als $I = (\mathcal{J}, \mathcal{F})$. Het probleem is nu als volgt gedefinieerd: Maak een schema van álle taken in $\mathcal{J}$ met een zo kort mogelijke lengte, waarvoor geldt dat geen enkele taak op een verboden tijd start of eindigt. ⊠

In een high-multiplicity encoding hangt de grootte van de invoer lineair af van het aantal verschillende taken en gemiddeld nog maar logaritmisch van het totaal aantal taken (worst-case natuurlijk nog steeds lineair). Hierdoor kan het zijn dat een polynomiaal algoritme onder normale codering exponentieel wordt onder een high-multiplicity encoding van de invoer. We hebben in Sectie 3 laten zien dat we voor SMS-FST een polynomiaal algoritme bestaat als we K, het aantal verboden starttijden, als constante beschouwen. We willen aantonen dat het SMS-FSE probleem *fixed parameter tractable* (FPT) is. We noemen een probleem FPT als dit probleem een looptijd heeft van $O\left(f(k) \cdot s^{O(1)}\right)$ voor een parameter k, waarbij $f(k)$ een berekenbare functie is die alleen afhankelijk is van k.

We zullen laten zien dat SMS-FSE onder een high-multiplicity encoding FPT is ten aanzien van de parameter K, het aantal verboden tijden. Dit betekent dat we een oplossing kunnen vinden in polynomiale tijd als we K als constante beschouwen.

We onderscheiden twee gevallen van het probleem. We zeggen dat een instantie I *grote diversiteit* heeft als $s > K$. Een instantie I heeft een *kleine diversiteit* als er geldt dat $s \leq K$. We zullen in de volgende twee secties bekijken hoe we uit instanties van grote of kleine diversiteit een optimaal schema kunnen bepalen. We zullen algoritmes geven voor het oplossen van de problemen en we laten zien dat de looptijd een vorm heeft van $O\left(f(K) \cdot s^{O(1)}\right)$. Daarmee is dan bewezen dat SMS-FSE fixed parameter tractable is.

5.1 Instanties met grote diversiteit

In deze sectie zullen we een algoritme geven dat een oplossing construeert voor een grote diversiteitsinstantie. C. Rapine en N. Brauner [18] hebben al laten zien dat er voor een grote diversiteitsinstantie een oplossing zonder nutteloze tijdsloten bestaat.

Stelling 5.3 (C. Rapine en N. Brauner [18])**.** *Als I een grote diversiteitsinstantie is, oftewel $s > K$, en $0, P \notin \mathcal{F}$, dan is er een schema met lengte P te maken, oftewel een schema zonder nutteloze tijdsloten.*

C. Rapine en N. Brauner [18] hebben ook een algoritme gegeven dat voor een instantie $I = (\mathcal{J}, \mathcal{F})$ onder normale codering een optimale oplossing oplevert. Dit algoritme wordt L-partitie genoemd. Voor een instantie onder normale codering geeft het algoritme L-partitie een optimale oplossing.

Stelling 5.4 (C. Rapine en N. Brauner [18])**.** *Een grote diversiteitsinstantie van het* SMS-FSE *probleem met normale codering kan door het algoritme L-partitie in* $O\left(K^3 n\right)$ *worden opgelost.*

Wij kunnen het algoritme L-partitie niet meteen toepassen op onze instantie, omdat wij gebruik maken van een high-multiplicity encoding. Om een polynomiaal algoritme te kunnen maken, zullen we meerdere taken tegelijk moeten inplannen. We weten wel dat er een optimaal schema bestaat, vanwege Stelling 5.3. We gaan een algoritme bespreken van M. Gabay, C. Rapine en N. Brauner [6] dat een optimaal schema maakt en meerdere taken tegelijk inplant. We voeren hiervoor eerst een aantal nieuwe definities in.

Definitie 5.5 (Deelschema)**.** We noemen een schema een *deelschema* als maar een deel van alle taken zijn ingepland. Als we nu twee deelschema's π en π' hebben, dan bedoelen we met $\pi\pi'$ het (deel)schema dat we krijgen door de twee schema's achter elkaar te plakken. ⊠

Merk hierbij op dat zowel π als π' niet mogen beginnen en eindigen op een verboden tijd. Dan zouden π en π' geen correct deelschema zijn.

Definitie 5.6 (Optimaal deelschema)**.** We noemen een deelschema π van een instantie I *optimaal* als er een deelschema σ bestaat zodat $\pi\sigma$ een optimaal schema voor I geeft. ⊠

Als de schema's π en σ geen nutteloze tijdsloten bevatten, dan is het logisch dat het schema $\pi\sigma$ ook geen nutteloze tijdsloten bevat. Als I een grote diversiteitsinstantie is, dan weten we door Stelling 5.3 dat er een schema zonder nutteloze tijdsloten bestaat. Mocht het voor instantie I zo zijn dat $P \in \mathcal{F}$ (per Definitie 5.2 geldt $0 \notin \mathcal{F}$), dan weten we dat er in het schema een nutteloos tijdslot op $t = 0$ ontstaat [18]. We passen dan de instantie aan, zodat het schema begint op de eerstvolgende tijd t zodat $t \notin \mathcal{F}$ en zodat $t + P \notin \mathcal{F}$. Deze transformatie kunnen we altijd toepassen.

Als we al een deelschema π hebben zonder nutteloze tijdsloten, en de resterende instantie I' is een grote diversiteitsinstantie, dan weten we dus ook vanwege Stelling 5.3 dat we een schema σ zonder nutteloze tijdsloten kunnen vinden. We noemen π (en σ) dan dus een optimaal deelschema. Merk op dat π niet kan eindigen op een verboden tijd, want dan zou het geen correct schema zijn. Er geldt dus ook dat σ niet begint op een verboden tijd. Als we deze schema's achter elkaar plakken, dan ontstaat er dus geen nutteloos tijdslot.

Het totale algoritme dat we gaan beschrijven maakt gebruik van een algoritme dat een optimaal deelschema maakt. Het idee is om herhaaldelijk dit algoritme voor een optimaal deelschema aan te roepen. Door deze deelschema's achter elkaar te plakken, krijgen we het totale optimale schema. Echter, om ervoor te zorgen dat het totale algoritme polynomiaal blijft, willen we wel garanderen dat we een begrensd aantal keer het algoritme voor een optimaal deelschema hoeven aan te roepen. We weten dat we maar K verboden tijden hebben. Dus als we zorgen dat elk optimaal deelschema minstens één verboden tijd overbrugt, dan hoeven we het algoritme voor een optimaal deelschema maximaal K keer aan te roepen. We noemen zo'n schema een efficiënt optimaal deelschema.

Definitie 5.7 (Efficiënt optimaal deelschema)**.** We zeggen dat een optimaal deelschema π van een instantie I *efficiënt* is als dit optimale deelschema ten minste één verboden tijd overbrugt. ⊠

Algoritme 1 is het algoritme voor een optimaal deelschema dat M. Gabay, C. Rapine en N. Brauner [6] gegeven hebben. Dit wordt gebruikt voor het oplossen van een grote diversiteitsinstantie van het SMS-FSE probleem. Dit algoritme zal een optimaal deelschema teruggeven en de resterende instantie zal een grote diversiteitsinstantie zijn. We gaan dit algoritme maximaal K keer aanroepen.

Hieronder zullen we de werking van het algoritme uitleggen en we zullen aantonen dat dit algoritme inderdaad een optimaal deelschema oplevert.

Laat $I = (\mathcal{J}, \mathcal{F})$ de grote diversiteitsinstantie zijn waarvoor we een optimaal schema willen bepalen. We weten vanwege Stelling 5.3 dat er een optimaal schema bestaat. Om in de gaten te houden dat we steeds een grote diversiteitsinstantie over houden, zullen we gebruik maken van twee verzamelingen A en B met taken. De verzameling B bevat van de eerste $K+1$ type taken precies één exemplaar. De verzameling $A = \mathcal{J} \setminus B$ bestaat uit de rest van de taken. Hieronder leggen we uit wat er in Algoritme 1 gebeurt, zie links de regelnummers. We gebruiken π voor de oplossing van onze instantie. Merk op dat π dus bestaat uit de oplossingsvectoren zoals aangegeven in Definitie 5.2. We gebruiken t voor de tijd, beginnend op $t = 0$, en i voor het type taak.

1-3: We zorgen dat we van de eerste $K+1$ types één taak uit A halen en deze in B stoppen. (Merk hierbij op dat we voor verzameling B niks hoeven op te slaan. De verzameling B bevat precies één taak van elk van de types $i = 1$ tot en met $i = K+1$. We kunnen de werktijden van deze taken gewoon opvragen in de vector p.) Daarna initialiseren we $i = 1$, voor de type taak waar we mee beginnen, t op 0, voor de huidige eindtijd van het schema en $\pi = \emptyset$ voor de lege oplossing, een schema zonder taken waarin alle oplossingsvectoren de waarde nul hebben.

4-7: We gaan zoveel mogelijk taken uit A proberen in te plannen. We beginnen bij type $i = 1$, dit zijn de taken met de langste werktijd. We proberen álle taken van type i toe te voegen vóór verboden tijd F_1. Als dit past, dan voegen we alle taken toe en gaan we door met de taken van type $i+1$. Als niet álle taken van één type meer voor F_1 passen, dan stoppen we de loop.

We hebben nu twee mogelijke resultaten. Alle taken uit A zijn ingepland (8-11), of niet alle taken uit A zijn ingepland (12-14).

8-11: Als $i > s$ dan zijn alle taken uit A ingepland vóór F_1. We hoeven nu alleen nog de $K+1$ taken uit B in te plannen. We kunnen L-partitie gebruiken om het resterende probleem op te lossen, zie Stelling 5.4. Dit kan in $O\left(K^3 n\right) = O\left(K^3(K+1)\right) = O\left(K^4\right)$ tijd.

12-14: Als niet alle taken uit A zijn ingepland ($1 \leq i \leq s$), dan proberen we van type i zoveel mogelijk taken (zeg α) voor F_1 te plannen.

De langste taak die we nu over hebben is van type i en we hebben een gat van $F_1 - t$ over tot de eerste verboden tijd. Dit gat zullen we nu proberen te overbruggen.

16-22: We zoeken een taak uit B die het gat $F_1 - t$ kan opvullen en die F_1 daarbij kan overbruggen. Als deze taak bestaat, dan plannen we deze in en dan zijn we klaar. We hebben een efficiënt optimaal deelschema π geconstrueerd. Als deze niet bestaat, dan volgt de volgende loop.

23-30: Alle taken uit B die over F_1 passen, eindigen op een verboden tijd (anders was er eentje in de vorige stap 16-22 ingepland). We nemen nu een taak van type x uit B met een werktijd die korter is dan $F_1 - t$. We plannen deze taak in, gevolgd door een taak van type 1 die F_1 zal overbruggen. Deze taak van type 1 zit nog in verzameling A: Waarom dit zo is, wordt afgeleid in het bewijs van Lemma 5.8.

Algoritme 1 Een algoritme voor het maken van een optimaal deelschema [6].

Require: Een grote diversiteitsinstantie $(\mathcal{J}, \mathcal{F})$ waarbij de vector $(p_1, \ldots, p_s)$ zo is geordend dat $p_1 > \cdots > p_s$ en waarvoor geldt dat $0 \notin \mathcal{F}$ en $P \notin \mathcal{F}$.
Ensure: Een optimaal deelschema π.

{Update de vector m zodat van de eerste $K+1$ type taken één taak overblijft voor de verzameling B.}
Zet $m_i = m_i - 1$ voor $i = 1$ tot en met $K+1$
$i = 1$; $t = 0$; $\pi = \emptyset$;
while $i \leq s$ en $t + m_i p_i < F_1$ **do**
  {Voeg de m_i taken van type i toe aan π.}
  Update π zodat $m_i^1 = m_i$; $t = t + m_i p_i$; $i = i + 1$;
end while
if $i > s$ **then**
  {Er zijn nog maar $K+1$ taken om in te plannen.}
  return π
end if
{Voeg zoveel mogelijk taken van type i toe vóór F_1 als mogelijk.}
$\alpha = \lceil (F_1 - t)/p_i \rceil - 1$;
Update π zodat $m_i^1 = \alpha$; $t = t + \alpha p_i$;
{Breidt π uit zodat π eindigt na F_1.}
{Probeer om één taak uit B te gebruiken zodat F_1 wordt overbrugd.}
for $l = 1$ tot en met $K+1$ zodat $t + p_l \geq F_1$ **do**
  if $t + p_l \notin \mathcal{F}$ **then**
    Update π zodat $J_1 = J_l$ en $S_1 = t$
    return π
  end if
end for
{Gebruik één taak uit B en één taak van type 1 (uit A) om de verboden tijd(en) te overbruggen.}
for $l = 2$ tot en met $K+1$ zodat $t + p_l < F_1$ **do**
  if $t + p_l + p_1 \notin \mathcal{F}$ **then**
    Update π zodat $m_l^1 = m_l^1 + 1$; $t = t + p_l$;
    Update π zodat $J_1 = J_l$ en $S_1 = t$
    return π
  end if
end for

We zullen in Lemma 5.8 laten zien dat het algoritme correct is. Er moet dan natuurlijk gelden dat het algoritme een optimaal deelschema oplevert. Definitie 5.6 stelt dan dat er ook voor de resterende instantie een optimaal deelschema moet bestaan. Uit Stelling 5.3 volgt nu dat we dus een grote diversiteitsinstantie over moeten houden. Verder zullen we aantonen dat een oplossing van het algoritme óf nog $K+1$ ongeplande taken heeft, óf dat dit een efficiënt optimaal deelschema is. In het eerste geval kunnen we L-partitie gebruiken om het schema af te maken. In het tweede geval hoeven we het algoritme inderdaad maximaal K keer aan te roepen.

Lemma 5.8 (M. Gabay, C. Rapine en N. Brauner [6])**.** *Gegeven een grote diversiteitsinstantie $I = (\mathcal{J}, \mathcal{F})$, levert Algoritme 1 een optimaal deelschema π op. Bovendien, als $I' = (\mathcal{J}', \mathcal{F}')$ de overgebleven instantie is, dan is I' een grote diversiteitsinstantie en er geldt*

1. *ofwel dat $|\mathcal{J}'| = |\mathcal{F}| + 1 = K + 1$ en alle overgebleven taken hebben verschillende werktijden (alle taken uit A zijn ingepland),*
2. *ofwel dat $|\mathcal{F}'| < |\mathcal{F}|$ en dan is π een efficiënt optimaal deelschema.*

Bewijs. Zij $I = (\mathcal{J}, \mathcal{F})$ de grote diversiteitsinstantie waarvoor we Algoritme 1 gaan aanroepen. Laat $I' = (\mathcal{J}', \mathcal{F}')$ de overgebleven instantie zijn na het gebruik van Algoritme 1.

Als na het gebruik van Algoritme 1 alle taken uit A zijn ingepland, dan hebben we de hele loop van de regels 4-7 gehad en dan geldt er $i > s$. We eindigen bij de regels 8-11 en we hoeven alleen nog de laatste $K + 1$ taken uit B in te plannen. We zitten dan in geval 1 van dit lemma.

Als na het gebruik van Algoritme 1 niet alle taken uit A zijn ingepland, dan zijn we in de loop van de regels 4-7 gestrand bij een bepaalde i. Per constructie gaat het algoritme nu proberen om het deelschema zó uit te breiden, dat deze optimaal wordt. Oftewel, het zal proberen om het schema zo uit te breiden dat deze na F_1 eindigt. Om te laten zien dat het algoritme correct is, zullen we moeten laten zien dat dit altijd lukt en dat we vervolgens een grote diversiteitsinstantie I' overhouden. We zitten dan in geval 2 van dit lemma.

Zeg dat we de loop van de regels 4-7 hebben gehad. Niet alle taken zijn ingepland, dus we zijn gestrand bij een bepaalde $i < s$. Het algoritme probeert nog α taken van type i toe te voegen (regels 12-14).

We bekijken $t < F_1$, de eindtijd van de laatst geplande taak. Merk op dat de grootste taak die we nog in moeten plannen van type i is. Voor het vervolg van het bewijs gaan we verzameling B opsplitsen in twee nieuwe verzamelingen. Laat $B = X \cup Y$ zodat $Y = \{j \in B \mid t + p_j \geq F_1\}$ de verzameling is met taken die F_1 kunnen overbruggen en $X = B \setminus Y$ de verzameling van taken die te kort zijn om F_1 te overbruggen. We weten dat de verzameling Y niet leeg is, omdat sowieso de taak van type 1 erin zit. Verder zullen we de volgende notaties aanhouden: s' staat voor het aantal verschillende taken in $\mathcal{J}'$ en $K' = |\mathcal{F}'|$ staat voor het aantal verboden tijden na t.

We zullen laten zien dat we van π een efficiënt optimaal deelschema kunnen maken en dat de resterende instantie I' een grote diversiteit heeft. Dat wil zeggen, als π eindigt na de l^{de} verboden tijd, dan zijn er maximaal l taken uit B ingepland. Als dat zo is, dan geldt er dat $s' \geq |B| - l > K - l \geq K'$ en dan heeft I' dus grote diversiteit.

Het algoritme gaat door met de loop van de regels 16-22. Als deze loop slaagt, dan hebben we dus een taak uit Y kunnen inplannen die niet eindigt op een verboden tijd. Het is duidelijk dat het schema π nu klaar is na F_1. Aangezien we één taak uit B hebben ingepland over een verboden tijd geldt er dat I' is een grote diversiteitsinstantie is.

Als de loop niet slaagt, dan gaan we door naar de tweede loop op regels 23-30. In de vorige loop heeft het algoritme geprobeerd om een taak uit Y in te plannen. Dit is niet gelukt. Dit betekent dat $t + p_j \in \mathcal{F}$ voor alle taken van type j in Y. Om toch een taak over F_1 te kunnen plannen, gaan we een taak uit X gebruiken om vóór F_1 te plannen. Vervolgens plannen we taak daar achteraan die F_1 wél kan overbruggen zonder op een verboden tijd te eindigen. Omdat we een grote diversiteitsinstantie hebben, weten we met behulp van Stelling 5.3 dat we zo'n combinatie van een grote en een kleine taak altijd kunnen vinden. Dat het algoritme deze taken ook vindt, zullen we nu laten zien.

Laat de kleine taak een taak zijn van type x uit X zijn. Neem als lange taak, de taak van type 1 uit Y. Als er nu geldt dat $t + p_x + p_1 > F_2$, dan zijn we klaar. We hebben twee taken uit B gebruikt, maar we hebben ook twee verboden tijden overbrugd. We hebben een efficiënt optimaal deelschema gevonden en I' is een grote diversiteitsinstantie.

Als er geldt dat $t + p_x + p_1 \leq F_2$, dan geldt er ook dat $t + p_1 < F_2$. We hebben echter eerder al geprobeerd om een taak uit Y in te plannen en het resultaat was dat alle taken op een verboden tijd eindigden. Hieruit volgt dat $t + p_1 < F_2$ ook op een verboden tijd eindigt. Oftewel, $t + p_1 = F_1$. We kunnen nu concluderen dat het algoritme in de eerste loop, regels 4-7, al is gestopt bij $i = 1$. Namelijk, als $i > 1$ dan zou het algoritme in elk geval nog één of meer taken van type $i' \geq 2$ hebben kunnen inplannen (regels 12-14). Er volgt dan dat $t + p_1 > F_1$, wat in tegenspraak is met de conclusie dat $t + p_1 = F_1$. Er geldt dus dat $i = 1$. Hieruit volgt dat er nog minstens één taak van type 1 in A zit. We gaan nu een taak van type x uit B en een taak van type 1 uit A inplannen zodat $t + p_x + p_1 > F_1$. We krijgen $s' \geq s - 1$ dus is I' een grote diversiteitsinstantie. Met Stelling 5.3 volgt er nu dat we een optimaal deelschema σ kunnen maken zodat het schema $\pi\sigma$ een schema is voor I met precies lengte P.

We hebben laten zien dat een resultaat van Algoritme 1 altijd voldoet aan geval 1 of geval 2 van Lemma 5.8. Hiermee is bewezen dat Algoritme 1 een optimaal deelschema oplevert. ■

De looptijd van Algoritme 1 is duidelijk $O(s + K)$. Aangezien we een grote diversiteitsinstantie hebben en $s > K$, volgt hieruit dat de looptijd $O(s)$ is. We roepen dit algoritme maximaal K keer aan, mogelijk gevolgd door een aanroep van L-partitie. Hieruit volgt dat de looptijd $O\left(sK + K^4\right)$ is. Deze looptijd is polynomiaal als K een constante is. Er volgt dat het SMS-FSE probleem voor grote diversiteitsinstanties FPT is ten aanzien van de parameter K.

5.2 Instanties met kleine diversiteit

Het algoritme beschreven in Sectie 5.1 kunnen we gebruiken voor grote diversiteitsinstanties. Voor kleine diversiteitsinstanties hebben M. Gabay, C. Rapine en N. Brauner [6] een integer lineair program (ILP) opgesteld. Hieronder zullen we uitleggen hoe het ILP eruit ziet, hoe we deze oplossen en hoe we vervolgens een schema construeren. We zullen dan zien dat het beschreven algoritme ook polynomiaal is als K een constante is. We hebben dan bewezen dat het SMS-FSE probleem FPT is ten aanzien van de parameter K.

Het verschil tussen een grote en een kleine diversiteitsinstantie is het aantal verschillende taken. Voor een kleine diversiteitsinstantie I geldt dat $s \leq K$. We kunnen Stelling 5.3 dus niet toepassen. Het kan zijn dat we voor onze instantie I geen schema kunnen maken met lengte P, oftewel dat we geen schema kunnen maken zonder nutteloze tijdsloten. Om dit probleem te verhelpen, gaan we onze kleine diversiteitsinstantie I omzetten in een grote diversiteitsinstantie I'. Dit doen we door $K + 1$ verschillende optionele taken toe te voegen.

Deze taken kunnen we inplannen als we een gat willen opvullen, maar ze hoeven niet in het uiteindelijke schema te zitten. Door $K+1$ verschillende taken toe te voegen, weten we zeker dat we een grote diversiteitsinstantie krijgen. We kunnen nu een schema π' maken voor I' zonder nutteloze tijdsloten. Dit schema kunnen we vervolgens omzetten in een schema π voor I door de optionele taken weg te laten. De taken uit onze instantie I noemen we verplichte taken. We hanteren de volgende definities:

- *Verplichte taken.* Dit zijn alle taken uit I. Deze taken móeten worden ingepland in het schema. We hebben taken van type i met werktijd p_i en multipliciteit m_i, met $i \in \{1, \ldots, s\}$.
- *Optionele taken.* Dit zijn taken die we toevoegen om een grote diversiteitsinstantie te creëren. We voegen $K+1$ verschillende taken toe van type i met $i = s+1$ tot en met $i = s+K+1$. Deze taken hebben werktijd $p_i = i - s$ en m_i is ongelimiteerd. Merk op dat de taken dus een werktijd hebben van $1, 2, \ldots, K+1$. Het zou kunnen zijn dat één of meer van deze types overeenkomen met één of meer types van de verplichte taken. Dit is geen probleem, omdat we uiteindelijk toch eerst de verplichte taken willen inplannen. Verder kunnen we zoveel taken van type $i \in \{s+1, \ldots, s+K+1\}$ inplannen als we nodig hebben.

We gebruiken $\widetilde{C}_{\max}(\pi')$ voor de eindtijd van de laatste verplichte taak in π'.

In het ILP gaan we gebruik maken van een bovengrens Q voor elk schema.

Stelling 5.9 (C. Rapine en N. Brauner [18])**.** *Een bovengrens voor de lengte van elk schema is* $Q = P + 2K$.

Verder voegen we nog een hele lange taak toe, die alle resterende verboden tijden zou kunnen overbruggen, als we de laatste verplichte taak al hebben ingepland. Laat deze taak van type $i = s+K+2$ zijn, zodat $p_{s+K+2} = F_K + 1$. Verder maken we gebruik van de notatie $F_{K+1} = Q + p_{s+K+2} + 1$.

Onze instantie I' heeft nu $s' = s+K+2$ taken. Voor taak van type i geldt:
- $i = 1, \ldots, s$: dit is een verplichte taak;
- $i = s+1, \ldots, s+K+1$: dit is een optionele taak en m_i is onbegrensd;
- $i = s+K+2$: dit is de resterende taak die alle verboden tijden kan overbruggen.

In het ILP zullen we gebruik maken van de volgende beslissingsvariabelen:

m_{ij}		het aantal taken van type i die klaar zijn voor verboden tijd F_j met $i = 1, \ldots, s'$ en $j = 1, \ldots, K+1$.
$R_{j\ell}$	$= 1$	als er een taak bestaat die precies de verboden tijden F_j tot en met $F_{\ell-1}$ overbrugt, met $j = 1, \ldots, K$ en $\ell = j+1, \ldots, K+1$,
	$= 0$	anders.
x_{ij}	$= 1$	als een taak van type i de verboden tijd F_j overbrugt en als dit de eerste verboden tijd is die deze taak overbrugt, met $i = 1, \ldots, s'$ en $j = 1, \ldots, K$,
	$= 0$	anders.
y_j	$= 1$	als alle verplichte taken klaar zijn voor F_j, met $j = 1, \ldots, K$,
	$= 0$	anders.

Waarbij er moet gelden dat
- $m_{ij} \geq 0$ en geheeltallig is en
- $R_{j\ell}$, x_{ij} en y_j boolean variabelen zijn.

Daarnaast gebruiken we de volgende variabelen, die afhankelijk zijn van de waarden van de beslissingsvariabelen.

W_j	staat voor de totale werktijd van de taken die klaar zijn voor F_j. Deze bevat zowel de werktijd van de verplichte taken als die van de optionele taken, met $j = 1, \ldots, K+1$.
$\widetilde{C}_{\max}$	de eindtijd van de laatste verplichte taak. $\widetilde{C}_{\max}$ is een niet-negatief reëel getal.

Merk hierbij op dat W_j alleen taken meeneemt die klaar zijn voor de verboden tijd F_j. Als de taken wel begonnen zijn voor F_j, maar niet klaar zijn voor F_j, dan zit de werktijd van deze taak niet in W_j.

Hieronder staat het ILP dat M. Gabay, C. Rapine en N. Brauner [6] hebben opgesteld. Het ILP levert een optimaal schema op zonder nutteloze tijdsloten, waarbij de eindtijd van de laatste verplichte taak geminimaliseerd wordt.

$$\min \ \widetilde{C}_{\max}$$

onder de voorwaarden:

Alle verboden tijden moeten worden overbrugd door precies één taak. Per definitie geldt er dat $R_{j\ell}$ voor alle combinaties van j en ℓ precies één keer 1 is en verder altijd 0 is. Beperking 5 en 6 zijn de basisgevallen voor beperking 7: er moet precies één taak over F_1 gaan en precies één taak over F_K.

$$\sum_{\ell=2}^{K+1} R_{1\ell} = 1 \tag{5}$$

$$\sum_{j=1}^{K} R_{j,K+1} = 1 \tag{6}$$

Voor elke ℓ geldt er: Als er een taak over $F_{\ell-1}$ én F_ℓ gaat, dan is er geen taak die $F_{\ell-1}$ overbrugt en klaar is voor F_ℓ en er is geen taak die start voor F_ℓ en deze verboden tijd overbrugt. Oftewel, de onderstaande sommaties zijn dan beide gelijk aan nul. Als er geen taak is die over $F_{\ell-1}$ én F_ℓ gaat, dan geldt er voor een j dat $F_{j\ell} = 1$ en voor een a dat $R_{\ell a} = 1$.

$$\sum_{j=1}^{\ell-1} R_{j\ell} = \sum_{a=\ell+1}^{K+1} R_{\ell a} \qquad \forall \ell = 2, \ldots, K \tag{7}$$

Stel x_{ij} in op zijn goede waarde.

$$\sum_{i=1}^{s'} x_{ij} = \sum_{\ell=j+1}^{K+1} R_{j\ell} \qquad \forall j = 1, \ldots, K \tag{8}$$

Het aantal taken dat nog ingepland moet worden daalt, en het aantal ingeplande taken stijgt.

$$m_{i,j+1} \geq m_{ij} \qquad \begin{array}{l} \forall i = 1, \ldots, s' \\ \forall j = 1, \ldots, K \end{array} \tag{9}$$

Als een taak van type i de verboden tijden F_j tot en met $F_{\ell-1}$ overbrugd, dan wordt $m_{i\ell}$ met één opgehoogd.

$$m_{i\ell} \geq m_{ij} + x_{ij} + R_{j\ell} - 1 \qquad \begin{array}{l} \forall i = 1, \ldots, s' \\ 1 \leq j < \ell \leq K+1 \end{array} \tag{10}$$

Alle verplichte taken moeten ingepland zijn.

$$m_{i,K+1} = m_i \qquad \forall i = 1, \ldots, s \tag{11}$$

De variabele y_j mag pas gelijk aan 1 zijn als álle verplichte taken klaar zijn, daarvoor niet.

$$\sum_{i=1}^{s} m_{ij} \geq y_j \sum_{i=1}^{s} m_i \qquad \forall j = 1, \ldots, K \tag{12}$$

Geef W_j de goede waarde.

$$W_j = \sum_{i=1}^{s'} m_{ij} p_i \qquad \forall j = 1, \ldots, K+1 \tag{13}$$

De taken die klaar zijn voor F_j, mogen niet op tijdstip F_j eindigen.

$$W_j \leq F_j - 1 \qquad \forall j = 1, \ldots, K+1 \tag{14}$$

Als er een taak is die F_j, $F_{\ell-1}$ en tussenliggende verboden tijden overbrugt ($R_{j\ell} = 1$), dan kan W_j niet worden verhoogd tussen deze verboden tijden. Oftewel, W_j kan pas worden opgehoogd als een taak echt klaar is. Als $R_{j\ell} = 0$, dan mag $W_{\ell-1}$ veel verschillen van W_j.

$$W_{\ell-1} \leq W_j + Q\,(1 - R_{j\ell}) \qquad \forall 1 \leq j < \ell \leq K+1 \tag{15}$$

Als $R_{j\ell} = 1$ en een taak van type i overbrugt F_j, dan moet deze taak lang genoeg zijn om klaar te zijn tussen $F_{\ell-1}$ en F_ℓ.

$$W_j + \sum_{i=1}^{s'} p_i x_{ij} \geq \sum_{\ell=j+1}^{K+1} (F_{\ell-1} + 1)\, R_{j\ell} \qquad \forall j = 1, \ldots, K \tag{16}$$

Als $R_{j\ell} = 1$ en een taak van type i overbrugt F_j, dan moet deze taak klaar zijn voor F_ℓ.

$$W_j + \sum_{i=1}^{s'} p_i x_{ij} \leq F_j - 1 + \sum_{\ell=j+1}^{K+1} (F_\ell - F_j)\, R_{j\ell} \qquad \forall j = 1, \ldots, K \tag{17}$$

Zorg dat $\widetilde{C}_{\max}$ gelijk is aan de eerste W_j zodat $y_j = 1$, waarbij $y_j = (0, \ldots, 0, 1, \ldots, 1)$. Beperking 18 is het basisgeval van beperking 19.

$$\widetilde{C}_{\max} \geq W_1 \tag{18}$$

$$\widetilde{C}_{\max} \geq W_j - y_{j-1} Q \qquad \forall j = 2, \ldots, K+1 \tag{19}$$

Nu we bovenstaand ILP hebben, is de vraag hoe dit probleem kan worden opgelost. We gebruiken hiervoor de volgende stelling.

Stelling 5.10 (F. Eisenbrand [5])**.** *Een geheeltallig programma met een binaire codering van lengte l in gefixeerde dimensie, dat gedefinieerd is door een gefixeerd aantal constraints, kan opgelost worden met $O(l)$ rekenkundige operaties op rationale getallen met een binaire codering met lengte $O(l)$.*

Het aantal beslissingsvariabelen in het ILP is $O\left(s' \cdot K + K^2\right)$. Omdat er geldt dat $s' = s + K + 2$ en $s \leq K$, volgt er dat $O(s' \cdot K + K^2) = O\left(2K^2\right) = O\left(K^2\right)$. De dimensie van het aantal beperkingen wordt gedomineerd door de beperking 10 die $O\left(K^3\right)$ is. Oftewel, we hebben een ILP dat een gefixeerde dimensie heeft als we K als constante beschouwen, dus we kunnen Stelling 5.10 toepassen. De (beslissings)variabelen die in het ILP worden gebruikt hebben een grootte van maximaal $O(P + 2K) = O(P + K)$. Oftewel, de binaire codering van het ILP is $O(\log((P+K)(K^2+K^3)))$. Dit is lineair in de grootte van de invoer.

We hoeven nu alleen nog uit de oplossing van het ILP een schema te construeren. Dit kunnen we doen met Algoritme 2. Dit algoritme is van eigen werk.

We zullen de $K+1$ vectoren en de K paren die een oplossing weergeven (zie Definitie 5.2) een waarde geven. We doen dit in een loop die van $j = 1$ tot en met $j = K + 1$ loopt, zie regel 3. Voor elke waarde van j gaan we de oplossingsvector waarden toekennen. We slaan op hoeveel taken er zijn ingepland vóór F_j en we slaan op welke taak er over F_j heen is gepland, zie regels 4-12. Vervolgens zullen we de tijd t updaten. We slaan alle tijd van de ingeplande optionele taken over, zie regels 13-16. Daarna weten we op welk tijdstip de taak begint die F_j overbrugt. We slaan deze tijd op in de oplossing, zie regels 17-20. Tot slot, als een taak meerdere verboden tijden overbrugt, dan hoeven we de tussenliggende oplossingsvector niet te updaten. Er zijn dan namelijk nul taken ingepland. We slaan deze dus over en we updaten de paren van de andere verboden tijden die worden overbrugt, zie regels 21-24. Als de buitenste loop, van regel 3, klaar is, dan hebben we alle $K + 1$ vectoren en alle K paren de waarde van de oplossing van het ILP toegekend.

De looptijd van Algoritme 2 is $O\left(K\left(s + K + 1\right)\right) = O\left(sK + K^2\right)$. Aangezien we een kleine diversiteitsinstantie hebben en $s \leq K$, wordt dit $O\left(K^2\right)$. Dit is polynomiaal als K een constante is. Er volgt dat het SMS-FSE probleem ook voor een kleine diversiteitsinstantie FPT is ten aanzien van K.

Algoritme 2 Een algoritme voor het maken van een schema uit een oplossing van het ILP.

Require: Een oplossing van bovenstaand ILP, waarbij alle beslissingsvariabelen waardes hebben gekregen.

Ensure: Een optimaal schema voor onze kleine diversiteitsinstantie I.

Zet $t = 0$; $\ell = 0$;
{Geef de vectoren die de oplossing bijhouden een waarde.}
for $j = 1$ tot en met $K + 1$ **do**
    {Sla de waarde van de verplichte taken op en update t.}
    for $i = 1$ tot en met s **do**
        Zet $m_i^j = m_{ij}$; $t = t + m_{ij}p_i$;
        {Check welke taak over F_j heen is gepland.}
        if $x_{ij} = 1$ **then**
            {Sla deze taak op in de oplossingsvector.}
            J_j is een taak van type i met werktijd p_i; $\ell = j$;
        end if
    end for
    {Sla de lengte van eventuele toegevoegde optionele taken over.}
    for $i = s + 1$ tot en met $s + K + 2$ **do**
        Zet $t = t + m_{ij}p_i$
    end for
    {Laat nu de taak beginnen die F_j overbrugt en update t.}
    if $j \leq K$ **then**
        $S_j = t$; $t = t + p_j$;
    end if
    {Als deze taak meerdere verboden tijden overbrugt, sla deze verboden tijden dan over.}
    while $t > F_{j+1}$ **do**
        $j = j + 1$; $J_j = J_\ell$; $S_j = S_\ell$;
    end while
end for

Referenties

[1] J.-C. Billaut and F. Sourd. Single machine scheduling with forbidden start times. *4OR-Q J Oper Res*, 7(1):37–50, 2009.

[2] N. Brauner, G. Finke, V. Lehoux-Lebacque, C. Rapine, H. Kellerer, C. Potts, and V. Strusevich. Operator non-availability periods. *4OR-Q J Oper Res*, 7(3):239–253, 2009.

[3] P. Brucker. *Scheduling Algorithms*. Springer-Verlag Berlin Heidelberg, 2004.

[4] Y. Chen, A. Zhang, and Z. Tan. Complexity and approximation of single machine scheduling with an operator non-availability period to minimize total completion time. *Information Sciences*, 251:150–163, 2013.

[5] F. Eisenbrand. Fast Integer Programming in Fixed Dimension. *European Symposium on Algorithms*, pages 196–207, 2003.

[6] M. Gabay, C. Rapine, and N. Brauner. High-multiplicity scheduling on one machine with forbidden start and completion times. *Journal of Scheduling*, 19(5):609–616, 2016.

[7] M.R. Garey and D.S. Johnson. *Computers and Intractability - A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.

[8] Y. He, W. Zhong, and H. Gu. Improved algorithms for two single machine scheduling problems. *Theoretical Computer Science*, 363(3):257–265, 2006.

[9] M. Ji, Y. He, and T.C.E. Cheng. Single-machine scheduling with periodic maintenance to minimize makespan. *Computers & Operations Research*, 34(6):1764–1770, 2007.

[10] I. Kacem, C. Chu, and A. Souissi. Single-machine scheduling with an availability constraint to minimize the weighted sum of the completion times. *Computers & Operations Research*, 35(3):827–844, 2008.

[11] I. Kacem and R. Mahjoub. Fully polynomial time approximation scheme for the weighted flow-time minimization on a single machine with a fixed non-availability interval. *Computers & Industrial Engineering*, 56(4):1708–1712, 2009.

[12] M. Lam. Software pipelining: an effective scheduling technique for VLIW machines. *ACM SIGPLAN Notices*, 23(7):318–328, 1988.

[13] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys. Chapter 9 Sequencing and scheduling: Algorithms and complexity. *Handbooks in Operations Research and Management Science*, 4:445–522, 1993.

[14] C.-Y. Lee. Machine scheduling with an availability constraint. *Journal of Global Optimization*, 9(3-4):395–416, 1996.

[15] C.J. Liao and W.J. Chen. Single-machine scheduling with periodic maintenance and nonresumable jobs. *Computers & Operations Research*, 30(9):1335–1347, 2003.

[16] Y. Ma, C. Chu, and C. Zuo. A survey of scheduling with deterministic machine availability constraints. *Computers & Industrial Engineering*, 58(2):199–211, 2010.

[17] M. Mnich and R. van Bevern. Parameterized complexity of machine scheduling: 15 open problems. *Computers & Operations Research*, 100:254–261, 2018.

[18] C. Rapine and N. Brauner. A polynomial time algorithm for makespan minimization on one machine with forbidden start and completion times. *Discrete Optimization*, 10(4):241–250, 2013.

[19] C. Rapine, N. Brauner, G. Finke, and V. Lebacque. Single machine scheduling with small operator-non-availability periods. *Journal of Scheduling*, 15(2):127–139, 2012.

[20] C. Sadfi, B. Penz, C. Rapine, J. Błażewicz, and P. Formanowicz. An improved approximation algorithm for the single machine total completion time scheduling problem with availability constraints. *European Journal of Operational Research*, 161(1):3–10, 2005.

[21] J.B. Sidney. Optimal Single-Machine Scheduling with Earliness and Tardiness Penalties. *European Journal of Operational Research*, 161(1):3–10, 2005.

A Een algoritme voor SMS-FSI

Hieronder het zelfgeprogrammeerde algoritme in C#.

```
1  class Program
2  {
3    int n;                 // Totaal aantal taken.
4    int[] taken;           // Werktijd van de taken.
5    int pn;                // Werktijd van taak n.
6    int P;                 // Totale werktijd van de taken.
7
8    int F1;                // Eerste verboden starttijd.
9    int FK;                // Laatste verboden starttijd.
10
11   bool[,] DPtabel;       // DP tabel: taken, eindtijden.
12   int aantalTaken;       // Lengte taken array.
13   int aantalEindtijden; // Lengte eindtijden array.
14
15   static void Main()
16   { new Program(); }
17
18   private Program()
19   {
20     Invoer();
21
22     // Check of alle taken voor het verboden start
23     // interval kunnen worden ingepland.
24     if (P - pn < F1)
25     {
26       Console.WriteLine("De lengte van een optimaal schema is P = " + P);
27       Console.WriteLine("Alle taken kunnen voor het verboden start
             interval worden ingepland.");
28       return;
29     }
30
31     // Zet de taak met de langste werktijd achteraan.
32     LangsteTaakAchteraan();
33
34     // Maak de DP tabel aan.
35     pn = taken[n];
36     aantalTaken = n;
37     aantalEindtijden = F1;
38     DPtabel = new bool[aantalTaken, aantalEindtijden];
39
40     // Voer het DP algoritme uit.
41     DPalgoritme();
42
43     // Het resultaat van het DP algoritme.
44     (int indexTaak, int tijd) = OplossingDP();
45     Console.WriteLine((indexTaak, tijd));
46
47     // Print de lengte van het schema.
48     PrintLengteSchema(tijd);
49
50     // Print het schema.
51     List<int> opl1_indices = OplossingSchemaDeel1(indexTaak, tijd);
52     PrintSchema(opl1_indices);
53     Console.ReadKey();
54   }
55
56   void Invoer()
```

```
57  {
58    // Aantal taken.
59    n = int.Parse(Console.ReadLine());
60
61    // De taken.
62    taken = new int[n + 1];
63    taken[0] = 0; // Voeg taak 0 toe met werktijd 0.
64    string[] invoerTaken = Console.ReadLine().Split();
65    int pi;
66    for (int t = 0; t < n; t++)
67    {
68      // Werktijd taak t.
69      pi = int.Parse(invoerTaken[t]);
70      taken[t + 1] = pi;
71      P += pi;
72    }
73
74    // Eerste verboden starttijd.
75    F1 = int.Parse(Console.ReadLine());
76
77    // Laatste verboden starttijd.
78    FK = int.Parse(Console.ReadLine());
79  }
80
81  void LangsteTaakAchteraan()
82  {
83    int werktijdpn = 0;
84    int indexpn = 0;
85    for (int i = 1; i < taken.Length; i++)
86      if (taken[i] > werktijdpn)
87      {
88        werktijdpn = taken[i];
89        indexpn = i;
90      }
91
92    int temp = taken[taken.Length - 1];
93    taken[taken.Length - 1] = werktijdpn;
94    taken[indexpn] = temp;
95  }
96
97  void DPalgoritme()
98  {
99    // Zet de eindtijd 0 op true voor alle taken.
100   for (int t = 0; t < aantalTaken; t++)
101     DPtabel[t, 0] = true;
102
103   for (int t = 1; t < aantalTaken; t++)
104     for (int e = 1; e < aantalEindtijden; e++)
105
106       // Kan ik met de voorgaande taken e bereiken?
107       if (DPtabel[t - 1, e])
108         DPtabel[t, e] = true;
109
110       // Kan ik met de voorgaande taken + deze taak e bereiken?
111       else if (taken[t] <= e && DPtabel[t - 1, e - taken[t]])
112         DPtabel[t, e] = true;
113 }
114
115 // Geef index en eindtijd van best haalbare schema deel 1.
116 (int, int) OplossingDP()
117 {
118   int t = aantalTaken - 1;
```

```
119     int e;
120     for (e = aantalEindtijden - 1; e >= 0; e--)
121       if (DPtabel[t, e])
122       {
123         while (t > 0 && DPtabel[t - 1, e])
124           t--;
125         break;
126       }
127     return (t, e);
128   }
129
130   List<int> OplossingSchemaDeel1(int indexTaak, int tijd)
131   {
132     List<int> oplossingIndices = new List<int>();
133     int t = indexTaak;
134     int e = tijd;
135     while (t > 0 && e >= 0)
136     if (DPtabel[t, e])
137     {
138       while (t > 0 && DPtabel[t - 1, e])
139         t--;
140
141       oplossingIndices.Add(t);
142       e -= taken[t];
143       t--;
144     }
145     else
146     e--;
147
148     return oplossingIndices;
149   }
150
151   void PrintLengteSchema(int eindtijd)
152   {
153     if (eindtijd + taken[n] > FK || eindtijd + taken[n] == P)
154     Console.WriteLine("De lengte van een optimaal schema is P = " + P);
155     else
156     {
157       int verschil = FK - (eindtijd + pn) + 1;
158       int lengte = P + verschil;
159       Console.WriteLine("De lengte van een optimaal schema is P + " +
              verschil + " = " + lengte);
160     }
161   }
162
163   void PrintSchema(List<int> opl1_indices)
164   {
165     List<(int,int)> opl2 = new List<(int,int)>();
166
167     // Initialiseer t1 met opl.Count - 2, omdat de laatste taak 'taak 0'
          is met werktijd 0.
168     // Initialiseer t2 voor de taken van deel 2 van het schema.
169     int t1;
170     int t2 = 0;
171     for (t1 = opl1_indices.Count - 2; t1 >= 0; t1--)
172       while(t2 < n)
173         if (taken[t2] != taken[opl1_indices[t1]])
174         {
175           opl2.Add((t2,taken[t2]));
176           t2++;
177         }
178         else
```

```
179         {
180           Console.WriteLine("taak " + opl1_indices[t1] + " met werktijd "
                  + taken[opl1_indices[t1]]);
181           t2++;
182           break;
183         }
184
185     // Voeg alle overige taken aan de oplossing toe.
186     for (int t = t2; t < n; t++)
187       opl2.Add((t,taken[t]));
188
189     // Print taak n.
190     Console.WriteLine("taak " + n + " met werktijd " + pn);
191
192     // Print het schema deel 2.
193     for (int i = 1; i < opl2.Count; i++)
194     {
195       (int t, int e) = opl2[i];
196       Console.WriteLine("taak " + t + " met werktijd " + e);
197     }
198   }
199 }
```