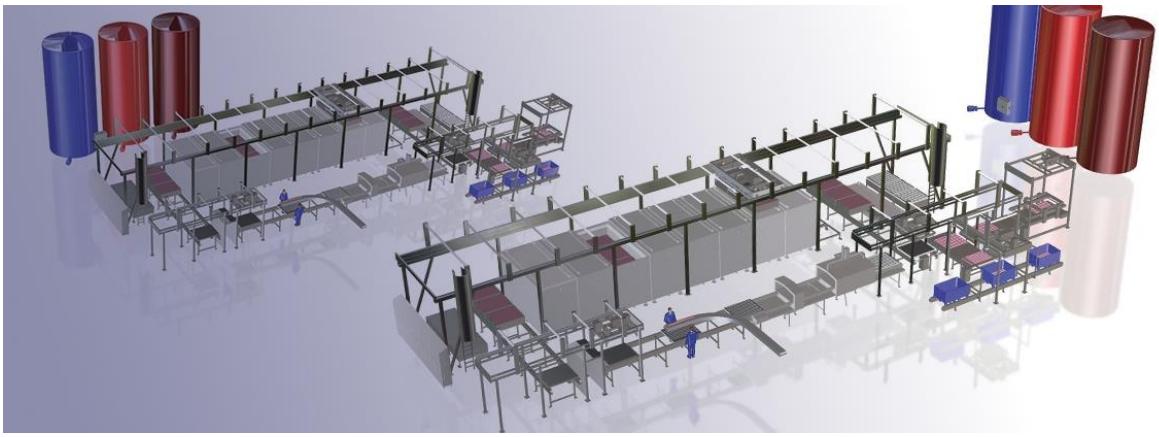
Sujet d'ordonnancement

-

Optimisation d'une chaîne de production



Carla JUVIN
Patty COUPEAU

EI4 - Systèmes Automatisés Génie Informatique

2018/2019

Sommaire

Présentation du contexte

Armor Inox est un constructeur d'équipements de processus industriels pour les produits alimentaires spécialisé dans la cuisson et la manutention du jambon, des viandes cuites (porc, volaille, bœuf, ...) et des légumes. Depuis sa création en 1972, Armor Inox a développé plus de 100 systèmes de cuisson en fonctionnement dans le monde entier.

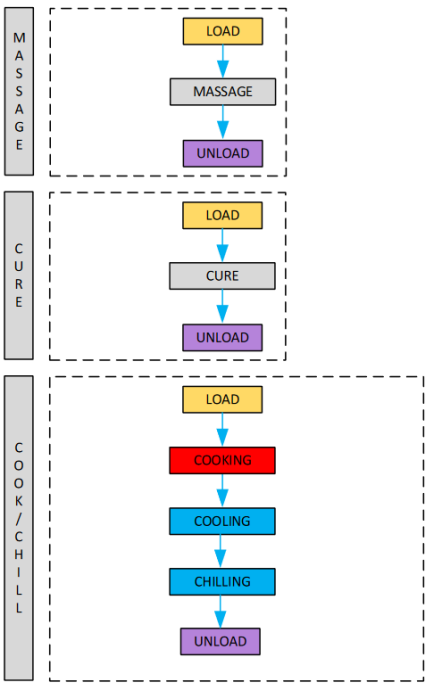
Installé à Mauron dans 10 000m² d'ateliers, la firme a rejoint en 2011 le groupe américain The Middleby Corporation spécialisé dans les équipements de processus industriels pour la fabrication, la cuisson et le conditionnement de produits alimentaires. Armor Inox constitue aujourd'hui le leader des solutions automatisées pour la cuisson et le refroidissement de jambon cuit.

Présentation du sujet

Dans le but d'augmenter leurs services aux clients, Armor Inox aimerait fournir à ses clients une chaîne de production et un plan de production appropriés à leurs besoins.

Considérons une usine souhaitant produire une certaine quantité de lots alimentaires appelés SKU. Armor Inox, en plus de lui fournir les équipements nécessaires à la production, sera en mesure de lui proposer une organisation de sa production permettant d'optimiser ses coûts.

Le but de notre projet a été de fournir aux clients d'Armor Inox un planning optimisé et détaillé correspondant à leur commande de lots alimentaires et aux équipements dont ils disposent.

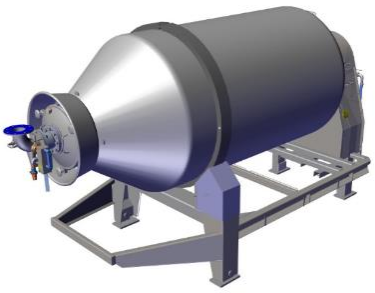


Pour obtenir le produit fini, chaque lot subit un ensemble de transformations nécessitant des équipements industriels. L'enchaînement de ces transformations constitue la chaîne de production ci-contre.

Chaque équipement permet de traiter des quantités de produits différentes. Dans le cadre de notre étude, le nombre d'équipements disponibles était fixé.

Nous allons détailler les divers processus constituant cette chaîne.

a) Massagers



Un *massager* est une baratte dans laquelle les aliments vont être mélangés au début de la chaîne de production. Armor Inox produit 3 types de massagers différant par leur capacité détaillée dans le tableau ci-dessous. Dans notre étude, nous avons considéré un massager de chaque type soit trois massagers.

EQUIPMENT MASSAGER & CURE TOTE				
	VOLUMES		LOADABLE BATCHES	
	MIN	MAX	MIN	MAX
Massager Type 1	680 kg	1300 kg	1	1
Massager Type 2	2300 kg	3500 kg	1	1
Massager Type 3	3600 kg	5200 kg	1	1

On ne peut mélanger dans une baratte que des produits provenant d'un même lot ou SKU. Les SKU sont regroupés par type de protéine ou de légume (par la suite, nous distinguerons ces types par des "couleurs"). Chaque couleur a un temps de traitement spécifique, détaillé ci-dessous :

ACTIVITY TIMES					
PROCESS STEP	MASSAGE				
ACTIVITY	LOAD	MASSAGE	WAIT		UNLOAD
PROCESS FLOW	FINITE	FINITE	MIN	MAX	FINITE
Protein A - Shape D	15min	2h25	0min	1h00	25min
Protein A - Shape R	15min	3h40	0min	1h00	25min
Protein B - Shape D	15min	2h40	0min	1h00	25min
Protein B - Shape R	15min	3h50	0min	1h00	25min
Vegetable A	15min	40min	0min	0min	60min
Vegetable B	15min	55min	0min	0min	60min

b) Cure Tote

Les *Cure Tote* sont des bacs d'entreposage entre le massage et la cuisson permettant de ne pas occuper les barattes inutilement. Nous ne les avons pas considérés dans notre étude.

c) Thermal

Lors de la phase de cuisson/refroidissement, les lots sont découpés en incréments qui seront placés dans des moules eux-mêmes introduits dans des cuves de cuisson appelées thermals. La cuisson dans ces cuves se fait par immersion avec circulation continue de l'eau. Un segment est une rangée de moules (que l'on appellera incréments) dans un thermal. Selon sa couleur, un segment n'occupe pas la même place dans un thermal car le poids ainsi que le nombre d'incréments qui le composent diffèrent. Ces caractéristiques sont accessibles dans le tableau ci-dessous :

EQUIPMENT THERMAL PROCESS TANKS					
	INCREMENT		BATCH SEGMENT		
	VALUE	NB MAX	VALUE	NB MAX	OCCUPANCY
PROCESS FLOW					
Protein A - Shape D	100 kg	10	**1 000 kg**	14	7.14%
Protein A - Shape R	200 kg	6	**1 200 kg**	12	8.33%
Protein B - Shape D	130 kg	9	**1 170 kg**	10	10.00%
Protein B - Shape R	220 kg	7	**1 540 kg**	8	12.50%
Vegetable A	500 kg	3	**1 500 kg**	15	6.67%
Vegetable B	510 kg	3	**1 530 kg**	15	6.67%

Tous les lots ou SKU ne peuvent pas être mélangés dans une cuve de cuisson. Ainsi, on ne fait cuire ensemble que les lots ayant les mêmes caractéristiques de cuisson. Par exemple, on ne peut pas placer dans une même cuve du jambon et des légumes. Les caractéristiques de cuisson sont répertoriées dans ce tableau :

ACTIVITY TIMES							
PROCESS STEP	THERMAL PROCESS						
ACTIVITY	LOAD	COOK	COOL	CHILL	WAIT		UNLOAD
PROCESS FLOW	LINEAR	FINITE	FINITE	FINITE	MIN	MAX	LINEAR
Protein A - Shape D	1min/100kg	5h20	1h00	8h30	0min	20h00	1min/100kg
Protein A - Shape R	1min/100kg	6h50	1h00	10h50	0min	20h00	1min/100kg
Protein B - Shape D	1min/100kg	4h40	1h00	6h20	0min	20h00	1min/100kg
Protein B - Shape R	1min/100kg	5h20	1h00	8h30	0min	20h00	1min/100kg
Vegetable A	1min/100kg	7h00	0min	3h30	0min	10min	1min/100kg
Vegetable B	1min/100kg	8h00	0min	4h00	0min	10min	1min/100kg

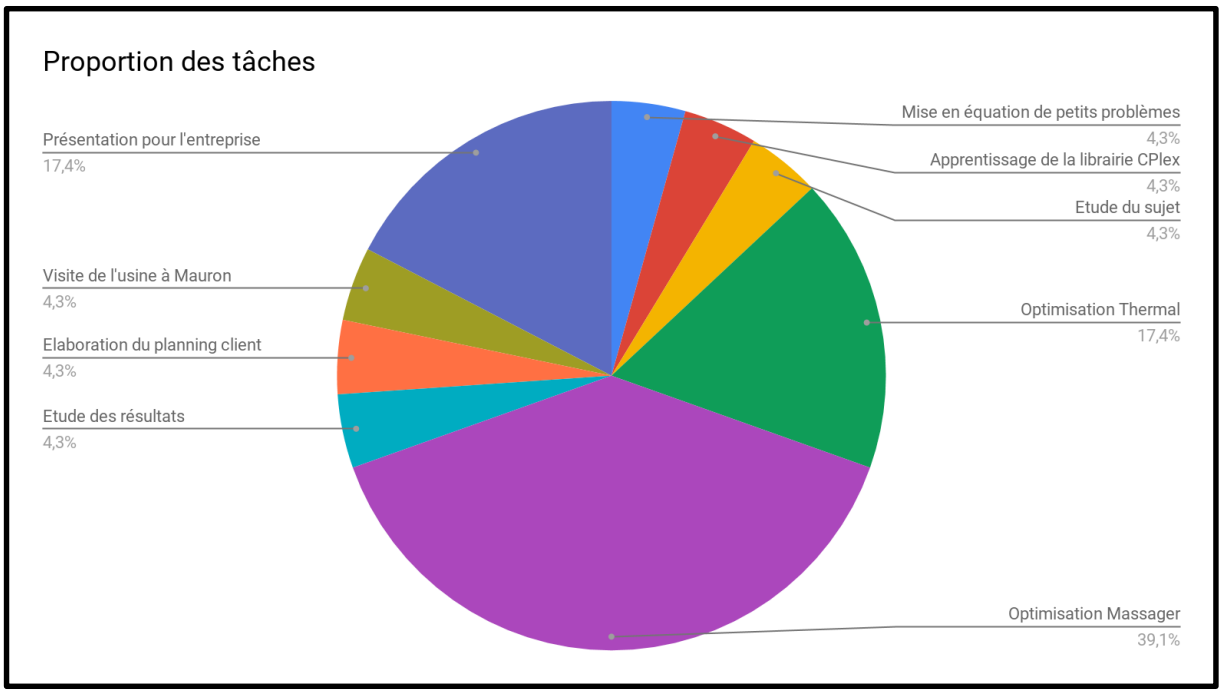
Dans notre étude, nous nous sommes limitées à deux Thermals.

Organisation du projet

Diagramme de Gantt - répartition des tâches

	Sem1	Sem2	Sem3	Sem4	Sem5	Sem6	Sem7	Sem8	Sem9	Sem10	Sem11	Sem12	Sem13	Sem14	Sem15	Sem16	Sem17	Sem18
Mise en équation de petits problèmes d'optimisation																		
Apprentissage de la librairie d'optimisation CPlex																		
Etude du sujet de projet et acquisition des données																		
Optimisation du processus Thermal																		
Optimisation du processus Massager																		
Etude des résultats																		
Elaboration d'un planning (diagramme de Gantt des résultats)																		
Visite de l'usine à Mauron																		
Présentation pour l'entreprise (préparation + oral)																		

Diagramme circulaire - Proportion des tâches



Les étapes du projet

Comme l'indique le diagramme des tâches, les étapes d'optimisation des processus Massager et Thermal ont occupé plus de la moitié de nos heures de projet. C'est donc le raisonnement utilisé pour ces optimisations que nous allons développer dans la suite du rapport.

Tout d'abord, lorsque nous sommes face à un problème d'optimisation conséquent, il convient de le traduire sous forme de systèmes d'équations puis de rentrer l'ensemble des équations dans un logiciel capable de résoudre le problème rapidement. C'est pourquoi, le choix d'une librairie d'optimisation a été la première décision à prendre durant notre projet.

I. Librairie d'optimisation CPlex

Nous avons choisi de travailler avec la librairie d'optimisation Cplex.

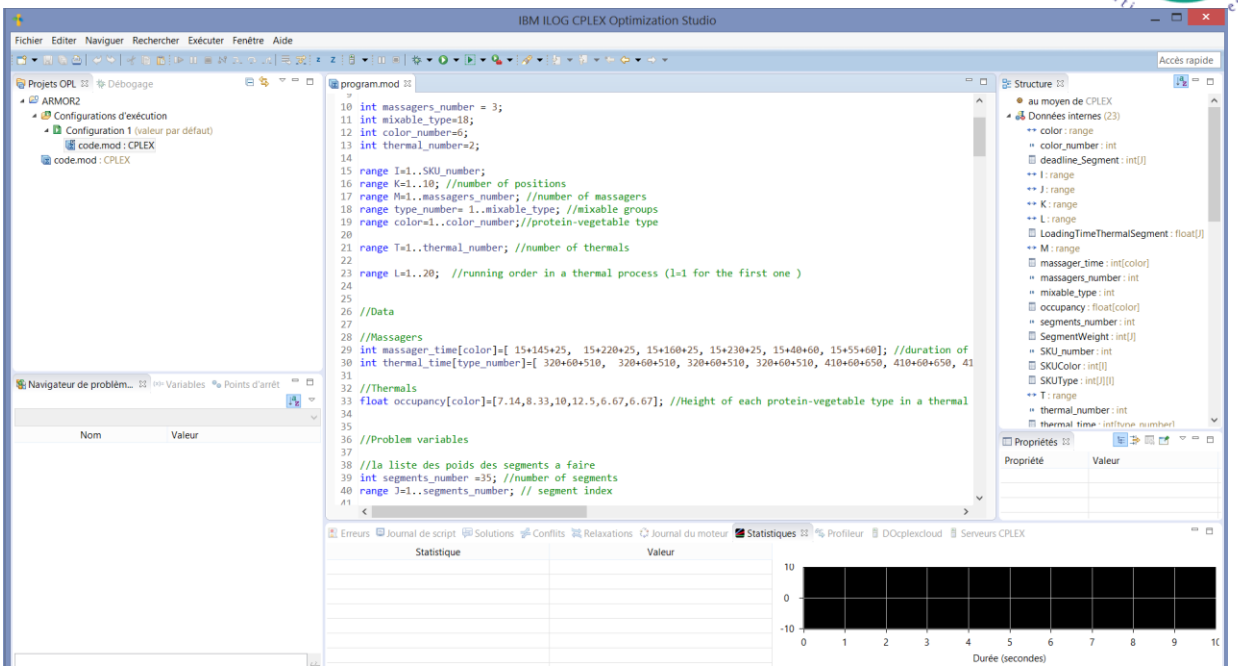
IBM ILOG CPLEX Optimization Studio est un solveur de programmes linéaires commercialisé par la société IBM. Il regroupe un ensemble d'outils pour la programmation mathématique et la programmation par contraintes en associant :

- Un environnement de développement intégré (Integrated Development Environment - IDE) nommé Cplex Studio IDE (sous Windows) ou oplide (sous Linux).
- Un langage de modélisation : le langage OPL (Optimization Programming Language), qui présente une syntaxe proche de la formulation mathématique.
- Deux solveurs : IBM ILOG CPLEX pour la programmation mathématique (résolution de programmes linéaires en nombres fractionnaires, mixtes ou entiers et de programmes quadratiques) et IBM ILOG CP Optimizer pour la programmation par contraintes.

Pour résoudre les problèmes d'optimisation, les solveurs d'IBM ont recours à l'algorithme du simplexe, un algorithme de résolution de problèmes d'optimisation linéaire. Petit rappel sur ce dernier :

Algorithme du simplexe : Introduit en 1947 par le mathématicien américain George Bernard Dantzig, il permet de minimiser une fonction sur un ensemble défini par des inégalités. Il consiste à trouver une solution de base au problème puis de passer d'une solution admissible à l'autre, en réduisant le coût.

Nous avons choisi de travailler avec cette librairie d'optimisation car elle présente l'avantage d'avoir une interface simple à comprendre et d'utiliser un langage proche de la formulation mathématique de nos équations. Ainsi, il n'y avait pas besoin de chercher la manière de traduire nos inéquations dans un autre langage.



Interface de programmation sur IBM CPlex

Pour ce choix, nous avons été confrontée à un problème. En effet, l'entreprise Armor Inox souhaitait que nous travaillions avec une autre librairie d'optimisation : Gurobi. Cependant, nous avions déjà modélisé l'ensemble du problème à l'aide de Cplex et ne trouvions pas d'intérêt à changer de librairie. De plus, l'interface Gurobi est difficile à prendre en main. Nous avons démontré à l'entreprise que l'utilisation de CPlex serait plus facile et efficace dans la réalisation de notre projet. Ils ont finalement été convaincus.

Une fois le logiciel d'optimisation choisi, il a fallu s'organiser sur la manière de résoudre le problème. En effet, celui-ci nous a paru complexe aux premiers abords car la chaîne de production à optimiser comprenait deux processus avec un nombre conséquent de contraintes.

II. Choix relatifs aux objectifs d'optimisation

Dans un premier temps, nous nous sommes fixées deux objectifs d'optimisation qui nous ont influencées dans notre façon d'aborder le problème.

a) Limiter les excès de production

L'objectif principal pour les clients d'Armor Inox est de ne pas produire trop en excès afin de limiter les pertes. C'est pourquoi nous avons choisi de déterminer les quantités à produire de chaque lot avant même d'optimiser le temps de production. Ainsi, nous avons défini le poids de chaque produit (SKU) en fonction de la quantité désirée par le client et en tenant compte des contraintes posées par le matériel disponible :

- Les SKU ne peuvent pas être mélangés dans une baratte. Donc, il faut que la quantité produite d'un lot soit supérieure à la capacité du plus petit Massager.

  Soit m1 le plus petit type de baratte: $\boldsymbol{QuantiteSKU \geq CapaciteMin(m1)}$

- Les moules placés dans les cuves de cuisson sont soit vides soit totalement remplis. Or, un moule ne contient qu'un type de SKU. Les moules associés à différents SKU peuvent avoir des capacités différentes. Donc, il faut que la quantité produite d'un SKU soit un multiple de la capacité du moule qui lui est associé.

  Soit moule1 le type de moule associé au SKU1: $QuantiteSKU1 = n * Capacité(moule1)\ avec\ n \in N$

Une fois fixés les poids à produire pour chaque SKU, nous avons choisi de diviser chaque SKU en ligne de thermal afin de faciliter la modélisation du problème. Ces découpages des SKU seront appelés **segments** dans notre étude. Ainsi, le poids maximal d'un segment est le poids pouvant être placé sur un étage d'une cuve de cuisson. Nous développerons plus tard dans le rapport la notion de segment et le choix de travailler avec ce type d'objet.

Cette décision de fixer les poids à produire dès le début aura des répercussions sur notre deuxième objectif d'optimisation. En effet, on s'interdit de produire en excès afin de produire plus vite. Pour illustrer cela prenons un exemple :

Un client commande de faibles quantités de beaucoup de SKU différents. Nous ne disposons que d'une baratte de petite capacité. Le poids des lots étant inférieur à la capacité du massager de type 2, tous nos SKU vont devoir passer dans le petit massager. Mais, les SKU ne peuvent pas être mélangés dans une baratte. Donc, il va falloir attendre que chaque lot passe dans la petite baratte. Si nous n'avions pas fixé le poids à réaliser, nous aurions pu choisir de produire davantage de certains SKU afin de les faire passer dans un plus gros massager. Ainsi, plusieurs SKU auraient pu être traités simultanément et nous aurions gagné du temps dans la production.

Cependant, nous justifions notre choix par le fait que l'objectif de limiter les excès de production est prioritaire sur le terrain à celui de produire le plus rapidement possible.

Voyons maintenant d'un peu plus près notre deuxième objectif d'optimisation.

b) Optimiser le temps de production de la commande

Notre objectif était d'optimiser l'ensemble de la chaîne de production constituée des Massagers et des Thermals. Pour cela, nous avons réalisé un travail de recherche afin d'en apprendre davantage sur les problèmes de flowshop qui s'apparentaient à notre problème. Nous avons étudié des problèmes de ce type dans le livre "***Programmation linéaire***" de Christelle Guéret, Christian Prins et Marc Sevaux.

Un problème de flow shop repose sur la théorie de l'ordonnancement. Il définit un ensemble de n tâches et m machines avec les contraintes suivantes :

- Toutes les tâches doivent passer sur toutes les machines
- Une machine ne peut traiter qu'une tâche à la fois.

Notre problème diffère car, contrairement aux problèmes de flow shop, l'ordre des opérations est fixé. Un produit doit obligatoirement passer dans le Massager puis dans le Thermal.

Malgré ce travail de recherche, nous ne sommes pas parvenues à modéliser cet objectif sous forme d'un problème linéaire. Par conséquent, nous avons décidé de séparer les deux processus et de traiter deux problèmes d'optimisation différents. Nous nous sommes concentrées dans un premier temps sur le processus des Thermals qui nous apparaissait comme le processus critique. En effet, l'opération de cuisson dure beaucoup plus longtemps que celui de mélange. Par conséquent, si le processus le plus long est fluide c'est l'ensemble de la chaîne de production qui sera fluide. C'est pourquoi, nous avons pensé judicieux de rendre cette étape totalement optimale et d'ordonnancer le processus de "Massager" en fonction des résultats trouvés.

Nous sommes conscientes que les résultats obtenus ne sont pas optimaux du fait de cette séparation des deux problèmes. Cependant, nous estimons que notre solution rend la production fluide et rapide. En effet, le processus le plus long est optimisé afin d'avoir le meilleur planning de cuisson possible. Ensuite, nous optimisons le planning des Massagers afin de respecter au mieux le planning des Thermals.

Fixer nos objectifs a conclu la première partie de notre projet. Cela nous a permis de mieux comprendre et appréhender le sujet. Savoir ce que l'on souhaite optimiser est indispensable pour se lancer dans l'étape suivante du projet, à savoir, la modélisation du problème.

III. Modélisation du problème

Le principal travail réalisé fut la modélisation du problème afin de représenter les différentes contraintes du monde réel. L'optimisation du problème s'est déroulée en 3 étapes que nous allons développer dans ce point. Après avoir établi les variables nécessaires à la modélisation du problème, nous détaillerons les 3 étapes de modélisation suivantes :

- Optimisation des Thermals: cette étape a pour objectif de trouver le planning d'affectation optimal des segments dans le processus Thermal. Nous présenterons la modélisation et le résultat obtenu.
- Ordonnancement des Thermals: cette étape est basée sur une heuristique que nous justifierons. L'objectif est de déterminer un ordre de passage des segments dans les Thermals afin d'optimiser le temps de production total.
- Optimisation des Massagers: cette étape a pour objectif de trouver le planning optimal d'affectation des segments dans les massagers pour respecter au mieux le planning des thermals obtenu lors de la 1e étape d'optimisation. Nous présenterons la modélisation et le résultat obtenu.

Avant de pouvoir réaliser les étapes d'optimisation, il nous faut d'abord poser les différentes variables dont nous aurons besoin.

On considère une usine qui produit SKU_number lots alimentaires différents. Nous avons numéroté chaque SKU afin de pouvoir les identifier. C'est pourquoi, nous avons créé un intervalle : $range\ SKUs = 1..SKU\_number$.

Une couleur permet d'obtenir diverses caractéristiques (temps de processus et capacité des moules correspondants dans le Thermal process). Nous avons créé un intervalle pour identifier ces couleurs : $range\ color = 1..color\_number$.

A chaque SKU est associé une couleur. Ceci est modélisé grâce à une matrice ligne : $int\ SKUColor[SKUs]$

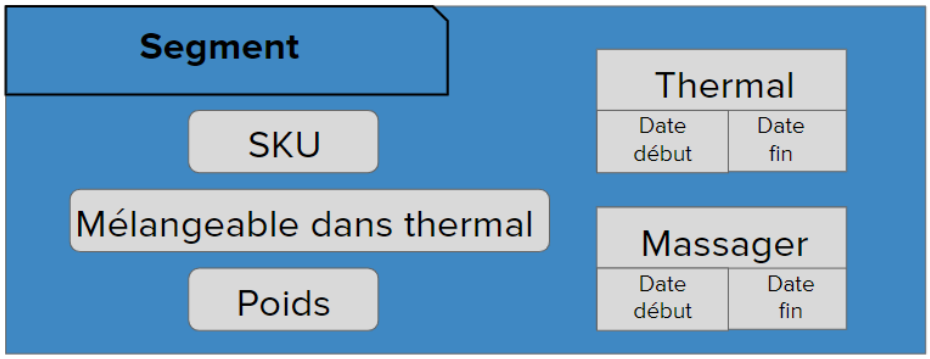
$$Pour\ tout\ s\ dans\ SKUs\ \ SKUColor[s]\ =\ s.color.$$

Ces couleurs nous permettent de connaître la hauteur d'une rangée dans une cuve de cuisson. Nous avons représenté cette hauteur avec une matrice ligne:
$float\ occupancy[color]$

$$Pour\ tout\ c\ dans\ color\ float\ occupancy[c]\ =\ c.height$$

Afin de travailler avec les mêmes objets tout au long de notre optimisation, nous avons décidé de considérer dans l'ensemble de notre réflexion des segments, c'est à dire des rangées de Thermal. Nous considérons à présent que l'entrée de notre problème contient une liste de segments de taille $segments\_number$.

Pour pouvoir identifier les segments, nous avons créé un intervalle: $range\ segments = 1..segments\_number$.
Chacun des segments possède un ensemble de caractéristiques dont voici une représentation schématique :



Chaque segment à réaliser possède un poids déterminé. Nous l'avons modélisé en créant une matrice ligne contenant l'ensemble des poids: $int\ SegmentWeight[segments]$.

$$Pour\ tout\ s\ dans\ segments\ SegmentWeight[s]\ =\ s.Weight.$$

Chaque segment provient d'un SKU. Nous avons créé une matrice de booléens à deux dimensions nous permettant de savoir si un segment appartient à un sku ou non :

$$int\ SKUType[Segments][SKUs]$$
$$Pour\ tout\ seg\ dans\ segments$$
$$Pour\ tout\ sku\ dans\ SKUs$$
$$Si\ seg.SKU\ ==\ sku$$
$$SKUType[seg][sku]\ =\ 1$$
$$sinon$$
$$SKUType[seg][sku]\ =\ 0$$

Certains segments peuvent être mélangés lors du processus de cuisson dans les Thermals. Nous avons donc créé un intervalle représentant l'ensemble des groupes qui pourront réunir ces SKUs : $\boldsymbol{range\ type\_number\ =\ 1..mixable\_type}$.
Deux segments appartenant au même groupe de mélange peuvent passer ensemble dans une cuve de cuisson.

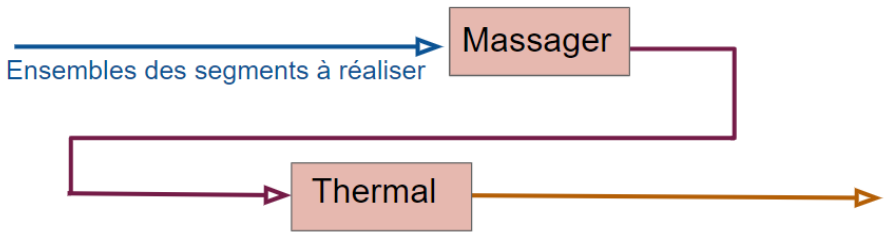
Le temps nécessaire pour le traitement dans un thermal dépend du groupe de mélange. C'est pourquoi nous avons défini une matrice ligne $int\ thermal_time[mixable_\boldsymbol{type}]$tel que:

$$Pour\ tout\ group\ dans\ mixable\_\boldsymbol{type}$$
$$thermal\_time[group] == group.ThermalTime$$

Pour définir à quel groupe de mélange appartient un segment, nous avons généré une matrice : $int\ SegmentType\_mixable[Segment][type\_number]$:

$$Pour\ tout\ seg\ dans\ segments$$
$$Pour\ tout\ group\ dans\ mixable\_\boldsymbol{type}$$
$$Si\ seg.\boldsymbol{mixableType} == group$$
$$SegmentType\_mixable[seg][group] = 1$$
$$sinon$$
$$SegmentType\_mixable[seg][group] = 0$$

L'entrée de notre problème est donc une liste des segments possédant l'ensemble des caractéristiques présentées. Ces segments vont devoir passer dans la chaîne de production :



Comme expliqué précédemment, nous avons modélisé en premier lieu le processus critique, c'est-à-dire les Thermals, afin d'optimiser le temps de production.

1) Optimisation des Thermals

L'objectif de cette première étape d'optimisation est de déterminer un planning optimisé pour le processus thermal, c'est-à-dire de déterminer pour chaque segment dans quel thermal il doit passer et avec qui il doit être mélangé afin que le processus thermal se termine le plus tôt possible.

Variables :

L'usine dispose d'un certain nombre $\boldsymbol{thermal_number}$de cuves de cuisson, appelé Thermals. Nous avons créé un intervalle afin de les identifier: $\boldsymbol{range\ T} = \boldsymbol{1..thermal\_number}$.

Ces cuves peuvent être utilisées plusieurs fois. Nous avons donc indexé les positions (dans le temps) afin de modéliser l'ordre dans lesquels sont traités les segments: $\boldsymbol{range\ P} = \boldsymbol{1..position\_number}$.

Le but est de déterminer dans quel thermal vont être traités les segments et avec qui ils seront mélangés. Nous avons imaginé une matrice à trois dimensions indiquant si un segment se trouve dans un thermal à un moment donné :$\boldsymbol{dvar\ boolean\ V}[segments][\boldsymbol{T}][\boldsymbol{P}]$. Cette matrice est une variable de décision qui sera calculée grâce à la résolution du problème d'optimisation.

$$Pour\ tout\ seg\ dans\ segments$$
$$Pour\ tout\ thermal\ dans\ T$$
$$Pour\ tout\ p\ dans\ P$$
$$Si\ seg\ est\ traité\ dans\ thermal\ à\ la\ position\ p$$
$$\boldsymbol{V[seg][thermal][p]} = 1$$
$$sinon$$
$$\boldsymbol{V[seg][thermal][p]} = 0$$

Pour s'assurer de ne mélanger que des segments d'un même groupe de mélange, nous avons créé une variable de décision :

$$\boldsymbol{dvar\ int\ Tank\_Type}[\ mixable\_\boldsymbol{type}][\boldsymbol{T}][\boldsymbol{P}].$$
$$Pour\ tout\ group\ dans\ \ mixable\_\boldsymbol{type}$$
$$Pour\ tout\ thermal\ dans\ T$$
$$Pour\ tout\ p\ dans\ P$$
$$Si\ group\ est\ traité\ dans\ thermal\ à\ la\ position\ p$$
$$\boldsymbol{Tank\_Type[group][thermal][p]} = 1$$
$$sinon$$
$$\boldsymbol{Tank\_Type[group][thermal][p]} = 0$$

Notre objectif est de minimiser le temps du processus Thermal. Nous devons donc mettre en place des variables de décision relatives aux dates de début et de fin de traitement dans les cuves :
$dvar\ int\ starting\_time\_Thermal[T][P]$ et $dvar\ int\ ending\_time\_Thermal[T][P]$.

Ce premier problème à optimiser est soumis à un ensemble de contraintes que nous détaillons ci-après.

Contraintes :

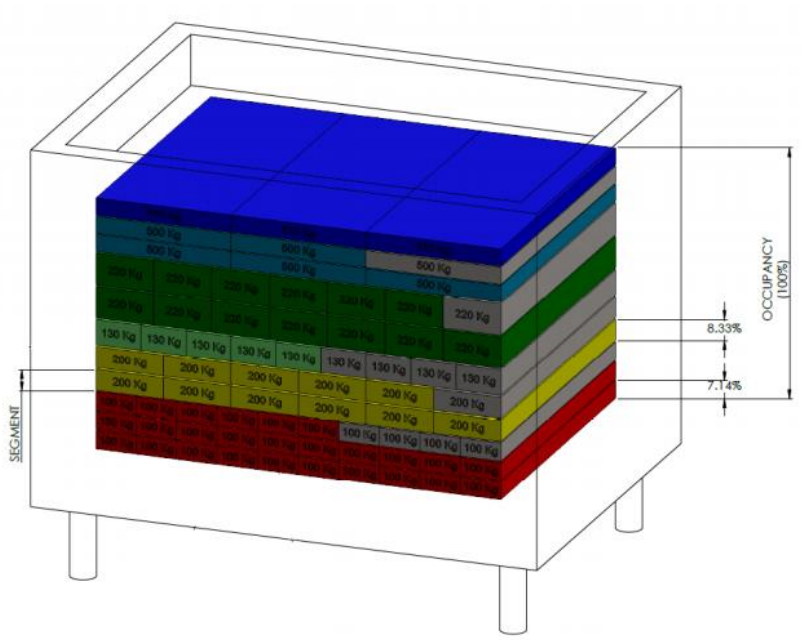
1) Chaque segment est traité qu'une fois dans un seul thermal. Une fois que le produit est cuit, on ne va pas le cuire une seconde fois. Nous avons traduit cette contrainte comme tel : $forall(seg\ in\ segments)\ sum(t\ in\ T, p\ in\ P)\ \ V[seg][thermal][p] == 1$.

Démonstration :
• $Pour\ un\ segment\ donné\ seg\ dans\ Segments,$ Si $sum(t\ in\ T, p\ in\ P)\ \ V[seg][thermal][p] < 1$ alors $sum(t\ in\ T, p\ in\ P)\ \ V[seg][thermal][p] == 0$ et donc
$Pour\ tout\ thermal\ dans\ T\ et\ Pour\ tout\ p\ dans\ P\ V[seg][thermal][p] == 0.$
Donc le segment n'est traité à aucun moment dans aucun des thermals. Ce segment n'est donc pas transformé, ce qui ne respecte pas la contrainte.
• $Pour\ un\ segment\ donné\ seg\ dans\ Segments,$ Si $sum(t\ in\ T, p\ in\ P)\ V[seg][thermal][p] > 1$ alors $il\ existe\ au\ moins\ deux\ couples\ (t,p)\ dans\ (T,P)\ \ tel\ que\ \ V[seg][thermal][p] == 1.$
Donc le segment est traité deux fois, ce qui ne respecte pas la contrainte

2) A un moment donné, un thermal ne peut contenir qu'un seul type de mélange. En effet, on ne peut pas mélanger dans une même cuve des produits ayant des caractéristiques de cuisson différentes. Nous avons modélisé cela grâce à la contrainte :
$forall(t\ in\ T, p\ in\ P)\ sum(group\ in\ mixable\_\boldsymbol{type})\ Tank\_Type[group][t][p] <= 1$

Démonstration :
$Pour\ un\ thermal\ donné\ t\ dans\ T$ à $une\ position\ p\ donnée\ dans\ P,$
Si $sum(group\ in\ \ mixable\_\boldsymbol{type})\ Tank\_Type[group][t][p] > 1$
alors $il\ existe\ au\ moins\ deux\ \ mixable\_\boldsymbol{type}\ tel\ que\ \ Tank\_Type[group][t][p] == 1.$
Donc le thermal t à la position p contient plusieurs types de mélange $group$, ce qui ne respecte pas la contrainte.

3) Chaque segment ou rangée de thermal possède une hauteur. La superposition des rangées ne doit pas dépasser de la cuve. Dit autrement, la hauteur des segments dans un thermal à un moment donné ne doit pas dépasser la capacité (100%) du thermal comme représenté sur le schéma ci-dessous:
$forall(t\ in\ T, p\ in\ P)\ sum(seg\ in\ Segments)\ \ V[seg][t][p] * occupancy[SegmentType\_color[seg]]\ <= 100$



4) Si on affecte un segment à un thermal à une position donnée, le type de mélange du segment (group) sera traité dans ce thermal à cet instant. Cela signifie que le thermal affecté à ce segment ne contiendra à cet instant que des segments du type group : le type group sera le type de la cuve.
$forall(t\ in\ T, p\ in\ P, seg\ in\ Segments)\ \ V[seg][t][p]\ < = \ sum(group\ in\ mixable\_\boldsymbol{type})\ SegmentType\_mixable[seg][group] * Tank\_Type[group][t][p]$

Démonstration :

Si le segment seg est dans un thermal donné t dans T à une position p donnée dans P $V[seg][t][p] = 1$, donc
$sum(group\ in\ mixable\_\boldsymbol{type})\ SegmentType\_mixable[seg][group] * Tank\_Type[group][t][p] >= 1$
Or, pour un segment donné seg, il n'existe qu'un seul $group\ dans\ mixable\_\boldsymbol{type}$ tel que $SegmentType\_mixable[seg][group] == 1$ (un segment n'appartient qu'à un groupe mélange). Donc tous les autres produits $SegmentType\_mixable[seg][group] * Tank\_Type[group][t][p]$ sont nuls. Ce qui oblige $Tank\_Type[group][t][p] == 1$ pour $group\ tel\ que\ seg.\boldsymbol{mixableType} == group$.

$Si\ le\ segment\ seg\ n'est\ pas\ dans\ un\ thermal\ donné\ t\ dans\ T$ à $une\ position\ p\ donnée\ dans\ P$ $V[seg][t][p] = 0$ donc $sum(group\ in\ type\_number)\ SegmentType\_mixable[seg][group] * Tank\_Type[group][t][p]$>=0. Ce qui n'est pas contraignant.

5) La date de début de traitement dans un thermal t à la position p doit être supérieure à la date de fin du traitement précédent, c'est-à-dire, à la date de début de la position précédente dans le même thermal, à laquelle on ajoute le temps de traitement. Cela est logique, on ne peut pas commencer une cuisson si la précédente n'est pas finie :
$forall(t\ in\ T, p\ in\ 1..19)\ starting\_time\_Thermal[t][p+1] >= starting\_time\_Thermal[t][p] + sum\ (\ group\ in\ mixableType) Tank\_Type[group][t][p] *\ thermal\_time[group]$

Démonstration :
$Pour\ un\ thermal\ donné\ t\ dans\ T$ à $une\ position\ p\ donnée\ dans\ P$,d'après la contrainte n°2 il existe au plus un $group$dans $mixableType$tel que $Tank\_Type[group][t][p]$==1.

Si ce thermal t est utilisé à cette position p, alors $Tank_Type[group][t][p] = 1$seulement pour le $mixableType\ group$ des segments présents à cet instant. Ainsi, le seul produit non nul est $Tank_Type[group][t][p] * thermal_time[group]$et représente le temps de traitement nécessaire à ces segments.
Si ce thermal t n'est pas utilisé à cette position p, alors $Tank\_Type[group][t][p] == 0$ pour tout $group$ dans $mixableType$ et tous les produits $Tank_Type[group][t][p] * thermal_time[group]$sont nuls
$sum\ (\ group\ in\ mixableType) Tank\_Type[group][t][p] *\ thermal\_time[group]$ est donc bien le temps de traitement nécessaire au Thermal tà la position p.

6) La date de fin de traitement dans un thermal t à la position p est égale à la date de début, à laquelle on ajoute le temps de traitement:
$forall(t\ in\ T, p\ in\ 1..20)\ ending\_time\_Thermal[t][p] == starting\_time\_Thermal[t][p] + sum\ (\ group\ in\ mixableType) Tank\_Type[group][t][p] *\ thermal\_time[group]$

Objectif :

Le but est de rendre le processus le plus rapide possible. Pour cela, nous devons faire en sorte que la plus grande date de sortie des Thermals soit la plus petite possible. Nous avons modélisé cela avec l'objectif : $minimize \ \ A$
avec : $forall(t \ in \ T) \ ending\_time\_Thermal[t][position\_number] <= A;$

Démonstration :

A est supérieur à la date de fin de processus pour chacun des thermals. A est donc la plus grande date de sortie des Thermals, c'est-à-dire, la date de sortie du dernier passage du dernier thermal. Minimiser Arevient donc à rendre le processus le plus court possible.

Résultat :

A l'issue de cette première étape d'optimisation, nous connaissons le planning optimisé du processus thermal. En effet, pour chaque segment, nous savons dans quel thermal il doit passer et avec qui il doit être mélangé.

L'affectation obtenue est optimale. C'est-à-dire qu'il n'est pas possible d'aller plus vite en réalisant une autre affectation de la liste de segments proposée. Pour chaque thermal, nous connaissons les différentes cuissons qui vont devoir s'y dérouler. Maintenant, il faut déterminer dans quel ordre elles vont se faire. Ceci est le but de la 2e étape d'optimisation.

2) Ordonnancement des Thermals

Nous connaissons désormais les différentes cuissons qui vont se dérouler au sein de chaque thermal. Déterminer et modifier l'ordre dans lequel ces opérations vont se dérouler ne changera pas la durée totale du processus. En effet, la somme des durées des traitements s'effectuant dans un Thermal restera la même puisque nous respectons les affectations de l'étape 1. Cependant, les produits traités dans les cuves de cuisson doivent auparavant passés par le processus Massager. Il convient donc de déterminer un ordonnancement qui prendra en compte les durées de massage afin de respecter au mieux le planning Thermal désiré.

Plusieurs SKU différents peuvent être mélangés dans une cuve de cuisson mais pas dans une baratte. Donc, si les mélanges sont réalisés en premier dans les Thermals, il faut que plusieurs Massagers se chargent de produire dès le début tous les SKU participant au mélange. Or, puisque nous sommes limités en équipements cela est compliqué. Nous obtenons notre première règle d'ordonnancement : **Réaliser les passages composés de mélanges en dernier.**

Pour départager les ordres de passages de cuisson, nous prenons en compte que les produits doivent passer dans les Massagers avant d'être cuits dans les Thermals. Ainsi, si on requiert en première position dans les Thermals une grosse quantité de matière, il sera compliqué pour les Massagers de produire une telle quantité à temps. Ceci nous fixe la deuxième règle d'ordonnancement : **Réaliser les plus petits passages en premier.**

Bien évidemment, ces règles sont basées sur une heuristique. Nous les avons conçues en fonction de notre approche personnelle du problème. Il est possible qu'un ordonnancement basé sur d'autres règles puisse donner un meilleur résultat.

Pour organiser un ordonnancement dépendant de ces règles, nous avons programmé en C# un algorithme de tri rapide.

L'entrée de l'algorithme de tri est la liste des segments avec le Thermal qui leur a été affecté ainsi que les segments avec lesquels ils doivent être mélangés.

Chaque passage de cuve va se faire attribuer des pénalités en fonction du poids et du nombre de SKU différents qu'il contient.

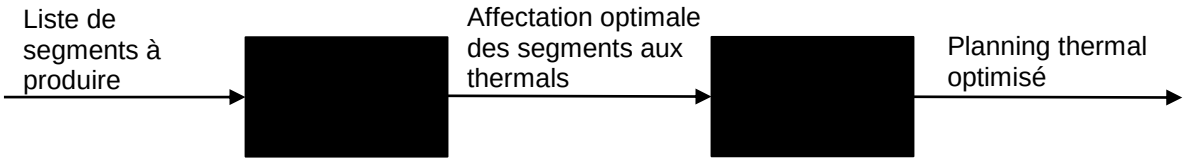
La pénalité est initialisée à 0.

- Pénalité = Pénalité + $(\sum \text{seg.poids})/100$
- Pénalité = Pénalité + 1000* nbSKU

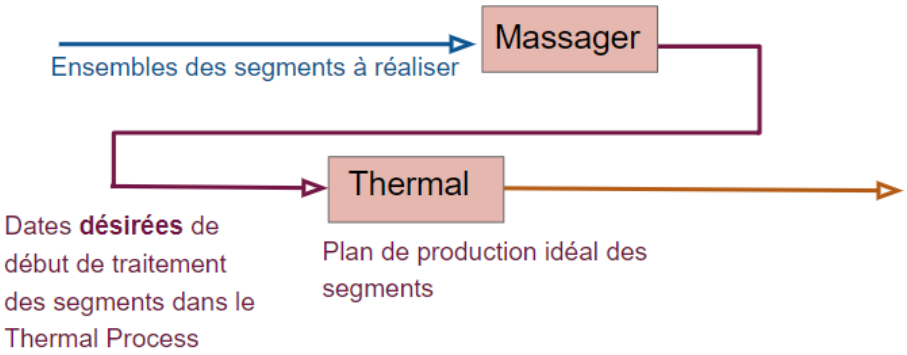
Le tri rapide se fera par ordre croissant du nombre de pénalités. Ainsi, les segments avec le moins de pénalités passeront en premier dans les Thermals.

En sortie de l'algorithme, nous obtenons l'ordre de passage dans chaque Thermal des segments. Nous disposons maintenant d'un planning ordonnancé de cuisson respectant les règles de priorité que nous nous sommes fixées.

A l'issue de cette 2e étape d'ordonnancement, nous connaissons le planning idéal du processus Thermal. En effet, pour chaque segment, nous savons dans quel Thermal il doit passer, avec qui il doit être mélangé et à quelle date il est attendu pour sa cuisson.



Notre chaîne de production ressemble maintenant à ceci :



Il va maintenant falloir optimiser le processus Massager afin de respecter au mieux le planning de production idéal des Thermals. Ceci est le but de la 3e étape d'optimisation.

3) Optimisation Massager

A l'entame de cette dernière étape d'optimisation, nous connaissons le planning idéal de production pour le processus Thermal. Il va maintenant falloir optimiser le processus Massager afin de respecter ce planning du mieux possible.

Variables :
L'usine dispose d'un certain nombre $massager_number$de barattes, appelées Massagers. Nous avons créé un intervalle afin de les identifier: $range\ M = 1..massager\_number$.
Ces barattes peuvent être utilisées plusieurs fois. Nous avons donc indexé les positions (dans le temps) afin de modéliser l'ordre dans lesquels sont traités les segments:
$range\ P = 1..position\_number$.

Le but est de déterminer dans quel Massager et dans quel ordre vont être traités les segments. Nous avons imaginé une matrice à trois dimensions indiquant si un segment se trouve dans un Massager à un instant donné :$dvar\ boolean\ U[segments][M][P]$. Cette matrice est une variable de décision qui sera calculée grâce à la résolution du problème d'optimisation.

$$Pour\ tout\ seg\ dans\ segments$$
$$Pour\ tout\ massager\ dans\ M$$
$$Pour\ tout\ p\ dans\ P$$
$$Si\ seg\ est\ traité\ dans\ massager\ \text{à}\ la\ position\ p$$
$$U[seg][massager][p] = 1$$
$$sinon$$
$$U[seg][massager][p] = 0$$

Pour s'assurer de ne mélanger dans une baratte que des segments provenant d'un même SKU nous avons créé une variable de décision:$dvar\ int\ Massager\_Type[SKUs][P][M]$;.

$$Pour\ tout\ sku\ dans\ SKUs$$
$$Pour\ tout\ massager\ dans\ M$$
$$Pour\ tout\ p\ dans\ P$$
$$Si\ sku\ est\ traité\ dans\ massager\ \text{à}\ la\ position\ p$$
$$Massager\_Type[sku][massager][p] = 1$$
$$sinon$$
$$Massager\_Type[sku][massager][p] = 0$$

Notre objectif est de respecter au mieux le planning du processus Thermal obtenu précédemment. Nous disposons à l'issue des deux premières étapes d'un vecteur indiquant pour chaque segment la date à laquelle il est espéré entrer dans une cuve de cuisson: $int\ deadline\_Segment[segments]$.

Nous devons mettre en place des variables de décision relatives aux dates de début et de fin de traitement dans les barattes afin d'évaluer ce respect du planning : $dvar\ int\ starting\_time\_Massager[M][P]$ et $dvar\ int\ ending\_time\_Massager[M][P]$.

Ce problème à optimiser est soumis à un ensemble de contraintes que nous détaillons ci-après.

Contraintes :

1) Chaque segment est traité une et une seule fois dans l'un des massagers. Nous avons traduit cette contrainte comme tel: $forall(seg\ in\ Segments)\ sum(p\ in\ P, m\ in\ M)\ U[seg][m][p]\ ==\ 1$

Démonstration :

• $Pour\ un\ segment\ donné\ seg\ dans\ Segments$, Si $sum(p\ in\ P, m\ in\ M)\ U[seg][m][p]\ <\ 1$ alors $sum(p\ in\ P, m\ in\ M)\ U[seg][m][p] == 0$ et $Pour\ tout\ massager\ dans\ M\ et\ Pour\ tout\ p\ dans\ P\ U[seg][massager][p] == 0$.
Donc le segment n'est traité à aucun moment dans aucun des Massagers. Ce segment n'est donc pas transformé, ce qui ne respecte pas la contrainte.

• $Pour\ un\ segment\ donné\ seg\ dans\ Segments,$ Si $sum(p\ in\ P, m\ in\ M)\ U[seg][m][p]\ >\ 1$ alors $il\ existe\ au\ moins\ deux\ couples\ (m,p)\ dans\ (M,P)\ \ tel\ que\ \ U[seg][m][p] == 1$. Donc, le segment est traité deux fois, ce qui ne respecte pas la contrainte.

2) Si on affecte un segment à un Massager à une position donnée, le type de SKU du segment (sku) sera traité dans ce massager à cet instant: le type sku sera le type de la cuve : $forall(m\ in\ M, p\ in\ P, seg\ in\ Segments)$,
$U[seg][m][p] <= sum(sku\ in\ SKUs)\ SKUType[seg][sku] * Massager\_Type[sku][m][p]$

Démonstration :

$Si\ le\ segment\ seg\ est\ dans\ un\ massager\ donné\ m\ dans\ Massagers$ à $une\ position\ p\ donnée\ dans\ P$ $U[seg][m][p] = 1$, donc
$sum(sku\ in\ SKUs)\ SKUType[seg][sku] * Massager\_Type[sku][m][p]$ >= 1
Or, pour un segment donné seg, il n'existe qu'un seul $sku\ dans\ SKUs$ tel que $SKUType[seg][sku] == 1$ (un segment n'appartient qu'à un groupe SKU). Donc, tous les autres produits $SKUType[seg][sku] * Massager\_Type[sku][m][p]$ sont nuls. Ce qui oblige $Massager\_Type[sku][m][p] == 1$ pour $sku\ tel\ que\ seg.sku\ == sku$.
$Si\ le\ segment\ seg\ n'est\ pas\ dans\ un\ massager\ donné\ m\ dans\ M$ à $une\ position\ p\ donnée\ dans\ P$ $U[seg][m][p] = 0$, donc
$sum(sku\ in\ SKUs)\ SKUType[seg][sku] * Massager\_Type[sku][m][p]$>= 0. Ce qui n'est pas contraignant.

2) Chaque segment possède un poids. La quantité de SKU placée dans un Massager à un instant donné doit respecter les capacités de la baratte. Pour le Massager de type 1 dont les capacités sont (680kg-1300kg) cela se traduit ainsi :

$$forall(p\ in\ p)\ sum(seg\ in\ Segments)\ U[seg][m][p] * SegmentWeight[seg]\ <= 1300;$$

$$forall(p\ in\ P)\ sum(seg\ in\ Segments)\ U[seg][m][p] * SegmentWeight[seg] >= sum(sku\ in\ SKUs)\ 680 * Massager\_Type[sku][m][p]$$

Démonstration :

D'après la contrainte 1, un segment est traité une seule fois et dans un seul massager. Par conséquent, dans un massager :

$forall(p\ in\ p)\ sum(seg\ in\ Segments)\ U[seg][m][p]$= nombre de segments affectés à ce massager à la position p

De plus, $U[seg][m][p] * SegmentWeight[seg]$est égal au poids du segment si le segment est affecté au massager m à la position p car $U[seg][m][p] = 1$sinon ce produit est égal à 0 car $U[seg][m][p] = 0$. Le poids du segment est comptabilisé uniquement s'il est affecté au massager à la position p. La première inéquation décrit donc simplement le fait que la somme des poids des segments affectés à un instant dans le massager de type 1 doit être inférieure à 1300 kg (capacité maximale).

La deuxième contrainte est légèrement différente.

$forall(p\ in\ P)\ sum(seg\ in\ Segments)\ U[seg][m][p] * SegmentWeight[seg]$ décrit toujours le poids contenu dans le massager à la position p. Si le massager ne contient aucun segment à la position p :

$sum(sku\ in\ SKUs)\ Massager\_Type[sku][m][p] = 0$ et

$sum(seg\ in\ Segments)\ U[seg][m][p] * SegmentWeight[seg] = 0$

La condition est bien respectée.

Sinon, $sum(sku\ in\ SKUs)\ Massager\_Type[sku][m][p] = 1$ car un massager ne contient qu'un type de SKU à un instant donné d'après la contrainte 2. Par conséquent, on demande que le poids des segments présents dans le massager à la position p soit supérieur à 680 kg (capacité minimale).

Pour les autres types de Massagers, les contraintes sont les mêmes sauf que les valeurs des capacités sont modifiées :

-type 2 : 2300kg-3500kg

-type 3 : 3600kg-5200kg

4) La date de début de traitement dans un massager m à la position p doit être supérieure à la date de fin du traitement précédent, c'est-à-dire, à la date de début de la position précédente dans le même massager, à laquelle on ajoute le temps de traitement. Cela est logique, on ne peut pas commencer une transformation si la précédente n'est pas finie :

$$forall(m\ in\ M, p\ in\ 1..9)\ starting\_time[m][p+1]\ > = \ starting\_time[m][p] + \ sum(sku\ in\ SKUs)\ Massager\_Type[sku][m][p] * massager\_time[sku]$$

Démonstration :
$Pour\ un\ massager\ donné\ m\ dans\ M$ à $une\ position\ p\ donnée\ dans\ P,$ il existe au plus un skudans $SKUs$ tel que $Massager\_Type[sku][m][p]$==1.

Si ce massager m est utilisé à cette position p, alors $Massager\_Type[sku][m][p] =$ 1seulement pour le $SKUs\ sku$ des segments présents à cette instant. Ainsi, le seul produit non nul est $Massager_Type[sku][m][p] * massager_time[sku]$et représente le temps de traitement nécessaire à ces segments.
Si ce massager m n'est pas utilisé à cette position p, alors $Massager\_Type[sku][m][p] == 0$ pour tout sku dans $SKUs$ et tous les produits $Massager\_Type[sku][m][p] * massager\_time[sku]$ sont nuls.
$sum\ (\ sku\ in\ SKUS) Massager\_Type[sku][m][p] * massager\_time[sku]$ est donc bien le temps de traitement nécessaire au Massager m à la position p.

5) La date de fin de traitement dans un massager m à la position p est égale à la date de début, à laquelle on ajoute le temps de traitement:
$$forall(m\ in\ M, p\ in\ 1..10)\ ending\_time[m][p]\ == starting\_time[m][p]\ +\ sum(sku\ in\ SKUs)\ Massager\_Type[sku][m][p] * massager\_time[sku]$$

6) La date de fin de traitement espérée d'un massager m à la position p doit être égale à la plus petite deadline des segments contenus dans ce massager m à la position p. En effet, un Massager contient à un instant donné des segments qui vont passer à des moments différents dans les cuves de cuisson. Le but est que chaque segment ait fini cette première transformation avant la date à laquelle il est censé commencer sa cuisson. Par ailleurs, l'opération de Massage ne peut pas se terminer avant la date 0 correspondant au lancement de la chaîne de production. Pour cette contrainte, on introduit la matrice $deadline[m][p]$indiquant la date de fin espérée du passage p dans le massager m:

$$forall(k\ in\ K, m\ in\ M)\ deadline[k][m]\ >= \ 0$$
$$forall(p\ in\ P, m\ in\ M, seg\ in\ Segments)\ deadline[m][p]\ < = \ deadline\_Segment[seg] * U[seg][m][p]\ + (1 - U[seg][m][p]) * 100000$$

Démonstration :
Si le segment seg est présent dans le massager m à la position p :
$deadline\_Segment[seg] * U[seg][m][p]$ = date de fin espérée du segment seg et
$$(1 - U[seg][m][p]) * 100000 = 0$$
Dans ce cas, on demande que le Massager ait terminé sa transformation avant la date à laquelle le segment qu'il contient est attendu dans les cuves de cuisson.

Si le segment seg n'est pas présent dans le massager m à la position p:
$$deadline\_Segment[seg] * U[seg][m][p] = 0$$
$$(1 - U[seg][m][p]) * 100000 = 100000$$
Par conséquent, on demande que le massager termine sa transformation avant une très grande date. Cela ne contraint pas le processus.

Objectif :

Le but est que le retard pris sur le planning initial soit le plus petit possible. Pour cela, nous devons faire en sorte que le plus grand retard soit le plus petit possible. Nous avons modélisé cela avec l'objectif : $minimize \ \ B$
avec : $forall(p \ in \ p, m \ in \ M) \ Real\_delay[m][p] \ <= \ B$ tel que:

$Real\_delay[m][p]$ contient le retard de la $p \ eme$ transformation du massager m c'est-à-dire la différence entre son réel temps de sortie des massagers et la date à laquelle il était espéré se terminer pour ne pas retarder le processus thermal :
$$forall \ (m \ in \ M, p \ in \ P) \ \ Real\_delay[m][p] \ == \ -deadline[m][p] \ + \ ending\_time[m][p]$$

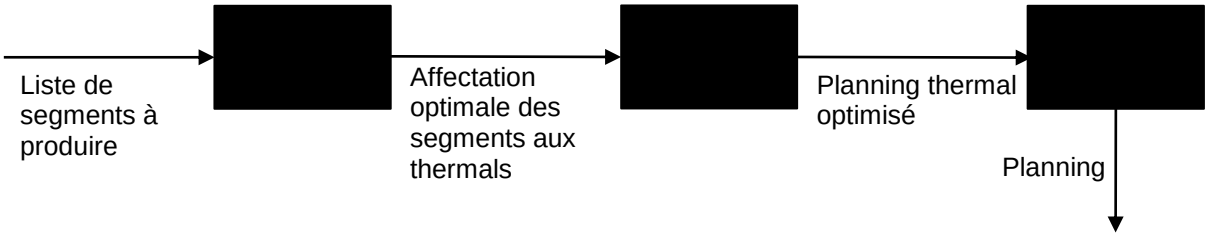
Démonstration :

B est supérieur au retard de chaque position pour chacun des Massagers. B est donc le plus grand retard pris par un segment pour son entrée dans le processus Thermal. Minimiser Brevient donc à rendre le plus grand retard le plus petit possible.

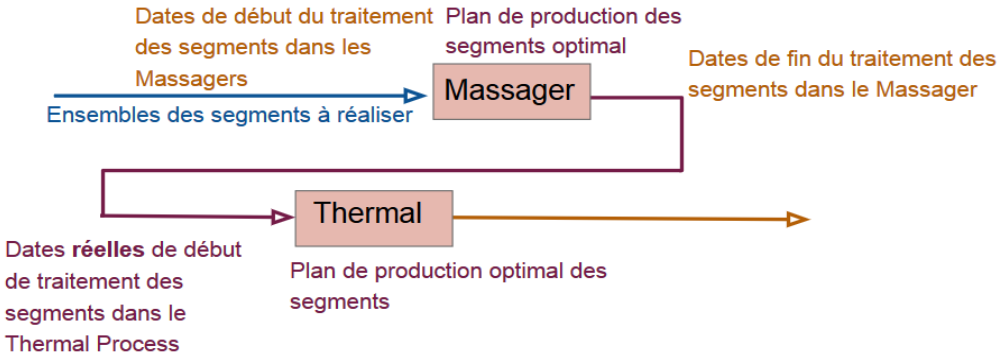
Nous avons minimisé le plus grand retard et non pas la somme des retards en se basant sur un raisonnement très simple. Si nous organisons une réunion et que tous les participants ont 10 minutes de retard, la réunion pourra commencer avec 10 minutes de retard (on ne somme pas leurs retards). Mais, si tous les participants sont à l'heure sauf un qui a 30 minutes de retard, on ne pourra pas commencer la réunion avant que tout le monde soit présent soit avec 30 minutes de retard.
Dans notre cas, c'est la même situation. Il faut minimiser le plus grand retard pour retarder le moins possible le planning thermal.

Résultat :

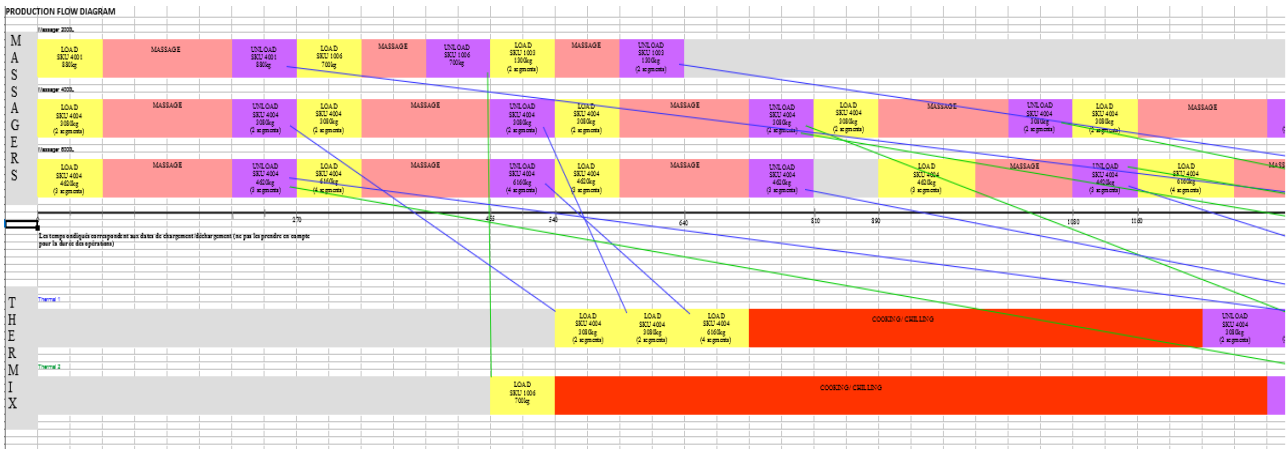
A l'issue de cette dernière étape d'optimisation, nous connaissons le planning optimisé de l'ensemble de la chaîne de production. A partir du planning des thermals, nous avons calculé l'affectation et l'ordonnancement dans les massagers minimisant le retard pris pour la suite de la production.



Le planning massager obtenu nous permet grâce à un décalage d'obtenir le réel planning thermal prenant en compte les retards de production. Nous connaissons maintenant en détail le cycle de vie de chaque produit.



Chez Armor Inox, les employés travaillent en s'appuyant sur un diagramme de Gantt détaillant le planning de l'ensemble des processus. Grâce à un programme C#, nous avons traduit les résultats obtenus sous la forme d'un diagramme de Gantt représenté sous Excel. Le rendu final de notre projet se traduira dans l'entreprise sous la forme suivante:



Perspectives du projet

I. Bilan sur le projet

Ce projet d'optimisation a été particulièrement formateur pour nous car c'était la première fois que nous travaillions pour une entreprise avec un cahier des charges précis. Nous avons acquis des compétences dans divers domaines tels que :

- L'analyse : analyse du besoin de l'entreprise et des résultats obtenus par notre programme.
- La modélisation : modéliser l'ensemble du cahier des charges fourni sous forme de problème linéaire. Transcription dans le langage opl pour résoudre le problème.
- La communication : Nous avons appris à communiquer de façon simple le travail réalisé afin que cela soit compréhensible par les employés d'Armor Inox qui ne connaissent pas le domaine de l'optimisation. Le travail de présentation pour l'entreprise a pris une place conséquente dans le projet car un bon travail mal communiqué ne perdure pas. En effet, l'objectif était de convaincre le responsable du service de l'efficacité de notre méthode. Nous y sommes parvenues.

Cependant, nous étions un peu perdues à l'entame du projet du fait d'un manque de connaissances sur le sujet. En effet, étant donné que nous avons commencé notre projet en septembre, nous n'avions pas encore eu les cours d'optimisation dispensés lors de la 2e année de cycle ingénieur. Ainsi, nous ne disposions pas des compétences nécessaires pour mettre en équation les contraintes du problème qui était tout de même relativement conséquent. C'est pourquoi, nous avons demandé de l'aide à M. Lagrange afin d'en apprendre davantage sur le domaine de l'optimisation et pour découvrir l'ensemble des astuces utilisées dans la modélisation.

Bien que nous soyons parvenues à atteindre nos objectifs, nous continuons à avoir un regard critique sur le travail réalisé. En effet, notre façon d'aborder le problème n'était pas la seule possible et nous aurions pu réaliser certaines parties différemment.

Tout d'abord, il est évident que si nous avions réussi à traiter l'ensemble de la chaîne de production sous la forme d'un unique problème, la solution obtenue aurait été vraiment optimale et donc probablement meilleure que celle obtenue. Cependant, nous n'y sommes pas parvenues malgré le travail de recherche réalisé. Mais, nous n'avons pas abandonné et nous avons réfléchi différemment pour obtenir un résultat satisfaisant en terme d'optimisation. De plus, l'ordonnancement dans les Thermals basé sur une heuristique et sur notre approche personnelle du problème nous empêche également d'affirmer que notre solution est la meilleure. Il est possible qu'un algorithme basé sur une toute autre approche pourrait donner de meilleurs résultats. Cependant, nous avons discuté avec le personnel d'Armor Inox et il semble que les règles d'ordonnancement mises en place soient cohérentes avec la réalité du terrain. C'est pourquoi, nous considérons que nos résultats sont satisfaisants et proches d'une solution optimale.

Ensuite, les résultats finaux auraient été totalement différents si nous n'avions pas fixé les quantités à produire dès le début. Mais, cela aurait supposé que l'optimisation du temps de production était prioritaire à la limitation des excès de production. Ce n'était pas ce qui était demandé dans le cahier des charges. Le choix fait est donc totalement expliqué par les consignes de départ.

Par ailleurs, nous avons choisi de ne pas intégrer les Cure Tote au planning final. Leur intégration aurait permis d'obtenir un résultat au plus près de la réalité et décrivant précisément le cycle de production de chaque lot. Cependant, comme ces machines ne servent qu'à l'attente et que les temps admissibles d'attente sont très importants, nous avons préféré ne pas en tenir compte pour simplifier le problème. Aujourd'hui, il serait simple de les intégrer. En effet, sur le planning final, le temps entre la sortie des barattes et le placement dans les Thermals correspond au temps passé dans les Cure Tote. On comprend donc qu'aborder le sujet avec la considération des Cure Tote n'aurait pas changé le problème.

Enfin, après discussion avec Armor Inox, nous avons découvert que nos résultats ne sont pas réalisables dans la réalité. En effet, ils ne disposent dans leur atelier que d'une machine pour charger les Thermals et d'une machine pour les décharger. Or, dans notre planning final, il arrive que les deux Thermals doivent être chargés (ou déchargés) parallèlement. Cependant, cette contrainte n'était pas présente dans le cahier des charges. C'est pourquoi nous ne l'avons pas considéré. Aujourd'hui, le problème peut être résolu en décalant à certains moments le planning final mais nous ne sommes plus certaines que cela donne un résultat optimisé. Il serait intéressant de reprendre le problème du départ avec cette nouvelle contrainte et de voir si les résultats obtenus sont très différents ou non.

A la fin des 90h de projet, nous avons fourni à l'entreprise nos résultats. Ils en étaient très satisfaits et nous avons discuté de cette dernière nouvelle contrainte concernant les chargements/déchargements ainsi que d'une possible automatisation de l'optimisation au travers d'un logiciel. Nous avons décidé de continuer à travailler avec eux pour enrichir le projet. Dans la dernière partie du rapport, nous aborderons l'évolution du projet entamée depuis le mois de janvier.

II. Poursuite du projet avec l'entreprise

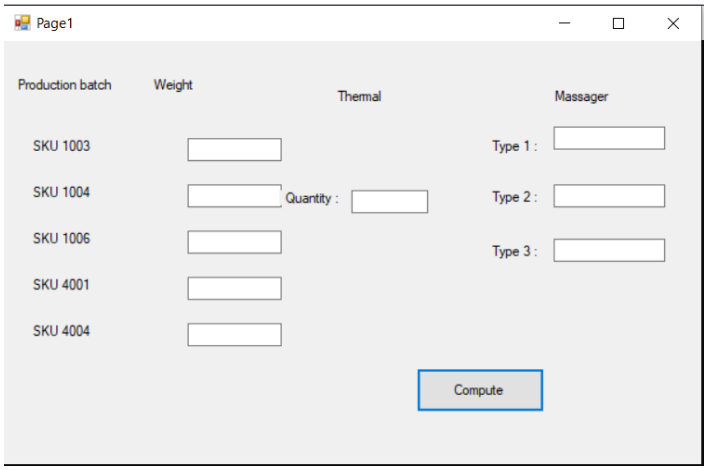
Le principal objectif de la suite du projet consiste à automatiser l'ensemble du travail d'optimisation réalisé au travers d'un logiciel. Armor Inox souhaiterait avoir une interface à partir de laquelle ils peuvent entrer :

- La quantité demandée pour chaque SKU par le client
- Le nombre de Massagers de chaque type à disposition
- Le nombre de Thermals à disposition

A partir de ces données, notre programme doit calculer et mettre à jour les variables du problème avant de faire appel au fichier Cplex d'optimisation. Ainsi, à partir de l'entrée de ces données, notre programme doit pouvoir ressortir le planning de Gantt final de la production optimisée.

La poursuite du projet nous a permis de développer d'autres compétences constituant notre formation. En effet, nous avons programmé notre logiciel en C# ce qui nous a permis d'utiliser nos connaissances en programmation orientée objet. Par ailleurs, nous avons appris à faire le lien entre un programme codé en C# et un programme codé en langage opl afin de récupérer les résultats du problème d'optimisation.

Ci-dessous, vous pouvez voir un premier aperçu de l'interface actuel du logiciel programmé. Notre but est uniquement de réaliser une preuve de concept. L'entreprise se chargera d'améliorer le design de l'interface proposée.



Enfin, si nous disposons de suffisamment de temps, nous tenterons de gérer les problèmes de chargement et déchargement simultanés. Notre objectif sera simplement de fournir après entrée des données le planning optimisé avec le décalage nécessaire pour que les chargements et déchargements soient réalisables tels quels dans la réalité.

A l'issue du projet, nous avons rendu à l'entreprise un code documenté et expliqué afin de permettre un développement possible de ce dernier dans l'avenir au sein de l'entreprise. Par ailleurs, la présentation faite à une partie des employés d'Armor Inox contribue à ce partage de savoir. Le projet entamé tout au long de cette année pourra donc connaître une suite facilement si l'entreprise en ressent le besoin.

Optimisation d'une chaîne de production

Ce travail d'optimisation a été réalisé dans le cadre du projet d'étude de 2e année de cycle ingénieur au sein de l'école d'ingénieurs Polytech Angers en partenariat avec l'entreprise Armor Inox. L'association avec une entreprise a permis l'élaboration d'un projet réaliste et le développement de compétences nécessaires à un ingénieur telles que la rigueur, le respect de délais ou le relationnel. La chaîne de production étudiée ainsi que le cahier des charges proposé par Armor Inox sont présentés à l'entame de ce rapport. La modélisation sous forme de problème linéaire, l'utilisation d'une librairie d'optimisation et la résolution du problème d'optimisation confié sont détaillés par la suite en précisant les choix et astuces utilisés. Une vision critique des résultats obtenus et des décisions adoptées achèvera ce compte-rendu de projet.

Mots clés : optimisation, modélisation, programmation linéaire, Armor Inox

This optimisation work was realised in the context of the research project of the 2nd year of Engineering within the Engineering School Polytech Angers in connection with the company Armor Inox. The relation with a firm resulted in the creation of a realistic project and in the development of skills required for an engineer such as rigour, respect of deadlines or human contact. The production line studied and the specifications proposed by Armor Inox are introduced at the beginning of the report. The modelling as a linear problem, the using of an optimisation library and the solving of the assigned optimization problem are detailed thereafter with the clarification of the choices and tips used. A critical view of the results achieved and of the decisions taken will end this project report.