Traballo Fin de Máster

Sorter Assignment Optimization

Ángel Mourelle Abelenda

Máster en Técnicas Estadísticas

Curso 2024-2025

Proposta de Traballo Fin de Máster

Título en galego: Sorter Assignment Optimization
Título en español: Sorter Assignment Optimization
English title: Sorter Assignment Optimization
Modalidad: Modalidade B
Autor/a: Ángel Mourelle Abelenda, Universidade de Santiago de Compostela
Director/a: Julio González Díaz, Universidade de Santiago de Compostela
Titor/a: Rubén Vázquez del Valle, Inditex
Breve resumo do traballo: Para levar a cabo os repartos de paqueteria nos clasificadores existen tres tipos de traballos, inducións, evacuacións e mixtos. Garantizando que sempre se cumprirán as datas de expedición e a produción sempre estará dentro duns umbrais determinados buscarase mellorar a productividade da máquina en inducións e a reducción das recirculacións asignando os traballadores de xeito óptimo.
Recomendacións:
Outras observacións:

Don/doña Julio González Díaz, profesor titular de la Universidade de Santiago de Compostelay don/-doña Rubén Vázquez del Valle, Lider de proyectos de optimización combinatoria del departamento de tecnología logística de Inditexinforman que el Trabajo Fin de Máster titulado

Sorter Assignment Optimization

fue realizado bajo su dirección por don/doña Ángel Mourelle Abelenda para el Máster en Técnicas Estadísticas. Estimando que el trabajo está terminado, dan su conformidad para su presentación y defensa ante un tribunal. Además, Don/doña Julio González Díazy don/doña Ángel Mourelle Abelenda

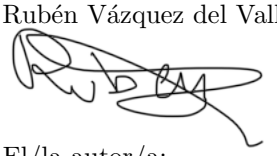
√ sí ☐ no

autorizan a la publicación de la memoria en el repositorio de acceso público asociado al Máster en Técnicas Estadísticas.

En Santiago de Compostela, a 3 de Junio de 2025.

El/la director/a:
Don/doña Julio González Díaz

El/la tutor/a:
Don/doña Rubén Vázquez del Valle



El/la autor/a:
Don/doña Ángel Mourelle Abelenda

MOURELLE ABELENDA ANGEL - 20629550E

Firmado digitalmente por MOURELLE ABELENDA ANGEL - 20629550E
Fecha: 2025.06.03 18:39:24 +02'00'

Declaración responsable. Para dar cumplimiento a la Ley 3/2022, de 24 de febrero, de convivencia universitaria, referente al plagio en el Trabajo Fin de Máster (Artículo 11, Disposición 2978 del BOE núm. 48 de 2022), **el/la autor/a declara** que el Trabajo Fin de Máster presentado es un documento original en el que se han tenido en cuenta las siguientes consideraciones relativas al uso de material de apoyo desarrollado por otros/as autores/as:

- Todas las fuentes usadas para la elaboración de este trabajo han sido citadas convenientemente (libros, artículos, apuntes de profesorado, páginas web, programas,...)
- Cualquier contenido copiado o traducido textualmente se ha puesto entre comillas, citando su procedencia.
- Se ha hecho constar explícitamente cuando un capítulo, sección, demostración,... sea una adaptación casi literal de alguna fuente existente.

Y, acepta que, si se demostrara lo contrario, se le apliquen las medidas disciplinarias que correspondan.

Agradecementos

Primeiramente, ao meu titor na empresa, Rubén Vázquez del Valle, con quen pasei tanto tempo durante a realización das prácticas. Lonxe de limitarse a supervisar os meus avances no proxecto, foi a fonte de moi valiosas ensinanzas matemáticas, laborais e persoais; que agardo non esquecer nunca.

Ao director deste TFM, Julio González Díaz, debo agradecerlle primeiramente a orientación que me proporcionou nun inicio, para poder encarar este traballo. Posteriormente, o seguimento da súa evolución e os seus consellos; e por último, a súa paciencia e esforzo na revisión dos borradores.

Dentro da empresa, cómpre dar as grazas en particular a José David García Boan, que me brindou unha axuda moi importante a nivel informático, que permitiu introducir a paralelización no simulador do capítulo 5. Pero sobre todo pola súa paciencia cun estudante sen experiencia que dificilmente se manexaba con ferramentas como GitHub, nin lograba instalar os solvers correctamente no seu portátil.

Finalmente, debo estender o agradecemento a todos os compañeiros de loxística cos que tiven a sorte de compartir espazo de traballo, pero sobre todo que me integraron no grupo dende o primeiro minuto. Probablemente foron os maiores causantes de que a experiencia das prácticas resultara para min tan positiva.

Índice xeral

Resumo

Resumo en galego

Este traballo busca mellorar a eficiencia un clasificador de almacén do sector téxtil mediante a óptima asignación de tarefas aos seus traballadores. Diversas prendas son inducidas no *sorter* dende o almacén, proceso onde se distinguen dous roles de traballo. Ao chegar a un destino concreto, a roupa cae na caixa que se atopa debaixo del. Tras encherse, as caixas son evacuadas por humanos, con outros dous roles de traballo posibles. O problema consistirá en decidir cantas persoas ocuparán cada un dos catro roles de traballo, buscando que a demanda, xerada polas necesidades dun conxunto de tendas, se satisfaga no menor tempo posible.

Este texto describe en profundidade o problema e formula un modelo de programación linear enteira para resolvelo. Tras comprobar que este non será capaz de resolver instancias de tamaño realista, e apreciar que o número total de asignacións traballador-rol é moito menor do esperado, contrúese un simulador que recree máis fielmente o *sorter* e avalíe as posibles asignacións, devolvendo a mellor delas como solución óptima.

English abstract

This work aims to improve the efficiency of a warehouse sorter in the textile industry through the optimal assignment of tasks to its workers. Various garments are fed into the sorter from the warehouse, a process where two work roles are distinguished. Upon reaching a specific destination, the product falls into the box below it. Once they are full, the boxes are evacuated by humans, with two other possible work roles. The decision-making problem will consist of how many people will occupy each of the four work roles, in order to satisfy the demand, generated by the needs of a set of stores, in the shortest possible time.

This text deeply describes the problem and formulates an integer linear programming model to solve it. After verifying that this model will not be able to solve realistic-sized instances, and observing that the total number of worker-role assignments is much lower than expected, a simulator is built that more faithfully recreates the sorter and evaluates the possible assignments, returning the best one as the optimal solution.

Prefacio

O problema que motiva este traballo enmárcase no campo da loxística de almacén, tendo como protagonista un clasificador, ao que tamén nos referiremos como *sorter*. Este é un sistema industrial que recibe prendas de roupa dende o almacén e diríxeas en bandexas cara os destinos ou xutes. Trátase de zonas habilitadas para que a bandexa se abra e a roupa caia nunha caixa, de modo que cada destino suministra os produtos que caen nel a unha tenda concreta (que actúan pois como os xeradores da demanda). Podemos considerar que o proceso global, que denominaremos reparto, remata cando todas as prendas caeron nalgunha caixa, de modo que a demanda de todas as tendas dos produtos de cada SKU (*Stock Keeping Unit*, a etiqueta que diferencia ese tipo de prenda das demais) queda satisfeita.

Con todo, este clasificador non está completamente automatizado ou robotizado, como moitos dos estudados na literatura, senón que precisan do factor humano, dividido en catro roles de traballo posibles. Os indutores sitúanse en partes concretas do *sorter* e adícanse e tomar as caixas chegadas dende o almacén e abrilas, lelas, e ir colocando as súas prendas en circulación. Como a apertura destas caixas ralentiza a indución de prendas, os chamados *waterspiders* darán apoio aos indutores, transitando distintos postos de indución para aforrarlles este proceso de rotura e lectura de caixas. Cando un destino se satura, é dicir, a caixa que ten debaixo está chea, pasa a bloquearse e deterse a actividade nesa zona. Para reiniciala, un operario debe apartar a caixa chea e colocar no seu lugar unha baleira, un proceso que denominamos evacuación. Chamaremos *packers* a aqueles traballadores que, tendo asignados unha zona de traballo específica, realizan estas evacuacións. De modo similar aos *waterspiders*, existirá un tipo de traballadores, chamados varios, que se moverán por todo o conxunto de destinos, evacuando aqueles que atopen cheos. Este é un modo de axudar a aqueles *packers* que enfronten circunstancialmente a unha carga de traballo maior.

Dito isto, o problema consistirá en determinar, dadas as características físicas do clasificador, a demanda e o total de traballadores, cantas persoas deben asignarse a cada unha das catro tarefas descritas, de modo que se minimice o tempo de cumprimento da demanda (tamén chamado *makespan* ao longo deste texto). Despois dunha busca bibliográfica infrutuosa, onde se constata que este problema non se explora moito na literatura, procédese a modelizar o problema dende cero.

O modelo creado demostrará non ser viable para a resolución de problemas de tamaño realista. No proceso de buscar algunha alternativa, descúbrese que o número posible de asignacións dos traballadores aos roles, considerando as restricións que impón o problema, é en realidade reducido. Neste punto nace a idea de crear un simulador que recree do xeito máis detallado posible o funcionamento do *sorter*, de modo que o óptimo se obteña sen máis que avaliar o comportamento de cada candidato, calculando o número de voltas do circuito que toma cumprir a demanda. Para achegarnos a uns tempos de execución do simulador compatibles cos estándares comerciais, a paralelización será unha ferramenta moi útil. Dado que a avaliación de cada candidato se scribirá como un proceso independente, será posible ensaiar moitas asignacións á vez, o cal permitirá reducir considerablemente o tempo necesaio para obter o óptimo do problema.

Este traballo estrutúrase como segue. O primeiro capítulo consta duns brevísimos preliminares. Neles limitámonos a describir superficialmente un problema de programación matemática, do tipo linear enteiro ou ILP (*Integer Linnear Programming*) en particular, e explicar algúns termos ou propiedades aos que se fai referencia nos seguintes capítulos. Tamén se comentan algúns aspectos de *Python*, a linguaxe na que se escriben o modelo e o simulador, e describimos brevemente como se crea un problema

de programación matemática coa libraría *Pyomo*.

Nun segundo capítulo explicarase en profundidade o problema, describindo as etapas do clasificador, os roles de traballo, e demais información que debe terse en conta no que resta de traballo. O seu último apartado consiste nunha revisión bibliográfica onde se describen algúns dos principais problemas tratados na literatura deste ámbito. O terceiro capítulo adícase á modelización, explicando con detemento os parámetros, variables e restricións que se inclúen no problema de programación enteira. Tamén se realizan ao final desta parte unha serie de comentarios sobre variantes deste modelo que se escribiron durante o proceso de prácticas.

O cuarto capítulo trata de criticar o modelo e revelalo como inoperante, aportanto diferentes argumentos. Na súa última sección, desenvólvese unha primeira versión do simulador baixo as simplificacións que asume o modelo, contrastando ambos os dous para instancias de tamaño reducido. Á vista dos bos resultados que aporta, o seguinte capítulo define unha versión evolucionada do simulador, probándoo finalmente cunha instancia dun tamaño máis próximo ao dun problema real. O último capítulo consta dunha breve conclusión.

Capítulo 1

Preliminares

Este capítulo introdutorio limítase a presentar os conceptos que se mencionarán repetidamente ao longo deste traballo. Estes enmárcanse en dúas áreas fundamentais: os modelos de optimización matemática e a programación en *Python*.

1.1. Programación matemática enteira

Este traballo ceméntase na modelización dun clasificador de almacén do sector textil, pois o enfoque inicial era a creación dun problema de programación matemática resoluble por un *solver* comercial, é dicir, un programa informático adicado a resolver modelos de programación matemática a partir dun conxunto de datos dado.

Como este texto non se centra exclusivamente nesta disciplina, e non se requiren para a súa comprensión coñecementos teóricos sobre programación matemática, limitarémonos a describir os principais elementos dun problema de programación linear enteira, e algunhas das súas características máis importantes para este traballo. As escasas definicións que aquí se mostrarán poden atoparse en [11], e o referido á complexidade computacional, en [1].

Un problema de programación linear enteira ou ILP[1] será aquel do tipo

$$\text{mín}\left\{\mathbf{c}^T\mathbf{x} : \mathbf{A}\mathbf{x} = \mathbf{b}, \mathbf{x} \geq \mathbf{0} \text{ e } x_j \in \mathbb{Z} \text{ para } j \in J\right\}. \tag{1.1}$$

Diremos que $\mathbf{c}, \mathbf{A}$ e $\mathbf{b}$ son os parámetros do problema, o que significa que os seus valores son coñecidos. Os elementos de $\mathbf{x}$ coñécense como variables, e a resolución do problema consiste en atopar os seus valores óptimos. Chamaremos solución óptima de 1.1 a aquel $\mathbf{x}$ que minimice a expresión $\mathbf{c}^T\mathbf{x}$, que denominamos función obxectivo.

O problema 1.1 está escrito de xeito matricial, onde por exemplo $\mathbf{A}\mathbf{x} = \mathbf{b}$ consiste nunha lista de ecuacións. Cada unha delas denomínase restrición, e aqueles $\mathbf{x}$ que verifican todas as restricións dun problema dinse solucións factibles. A formulación empregada neste caso coñécese como forma estándar, aquela en que todas as restricións son igualdades. Un problema defínese comunmente empregando restricións de desigualdade non estrita, que se poden traducir a esta forma estándar sen máis ca engadir variables auxiliares.

Os ILP caracterízanse porque algunhas das súas variables son números enteiros, e as súas restricións e función obxctivo son lineais. Supoñendo que 1.1 comprende n variables, se $J \subsetneq \{1, ..., n\}$ falamos dun problema de programación enteira mixto ou MIP[2], mais se $J = \{1, ..., n\}$, estamos ante un problema de programación enteira pura. Este será o caso do modelo que se presentará no capítulo 3, pois todas as variables son enteiras, e en moitos casos directamente binarias (é dicir, pertencentes a $\{0, 1\}$).

[1] Integer Linnear Programming
[2] *Mixed Integer Programming.*

Para poder cuantificar o tamaño dun problema de programación matemática, entendido como a cantidade de cálculos que un ordenador debería realizar para resolvelo (no peor caso posible), creouse o concepto de complexidade computacional. En [1] faise unha explicación pormenorizada onde se chega a calcular esta complexidade mediante o número de *bits* empregados pola máquina, pero para este texto non se precisa este nivel de profundidade. Fixarémonos no número de iteracións que precisa realizar, no peor dos casos, un algoritmo que tente resolver un problema de programación matemática. Os problemas de programación linear ou LPPs[3] teñen como algoritmo de resolución máis coñecido o chamado método símplex. Este presenta unha complexidade expoñencial, o que significa que, se un LPP ten n variables, require da orde de 2^n repeticións (dun certo bloque de cálculos) para chegar ao óptimo. Dunha banda, pese a ser expoñencial, este método presenta na práctica un rendemento case polinómico, e pola outra, existen métodos de complexidade propiamente polinómica como o algoritmo de punto interior de Karmarkar. Isto non acontece cando as variables son enteiras. Nun ILP, todo método de resolución exacta, i.e., que logre acadar a solución óptima, presentará unha complexidade computacional de tipo expoñencial. Este concepto cobrará sentido no capítulo 4, adicado a criticar o modelo proposto dende varios puntos de vista, sendo o principal deles a súa complexidade computacional.

Finalmente, pola natureza do problema que describiremos no capítulo 2 e modelizaremos no capítulo 3, é conveniente que o lector se familiarice co significado das simetrías ou solucións simétricas. No contexto dun problema de programación matemática como o descrito en 1.1, supoñamos que as compoñentes de $\mathbf{x}$, ou cando menos unha parte delas, fan referencia a unha mesma característica, pero para individuos diferentes. Se estas persoas non presentan no modelo outras características que os diferencien entre sí, pode entenderse que son intercambiables. Neste contexto,para unha solución óptima $\mathbf{x}$, existiría algunha $\bar{\mathbf{x}}$ que sexa simplemente unha permutación das compoñentes de $\mathbf{x}$, pois as propiedades das variables (en conxunto) son as mesmas. Nesta situación diremos que $\bar{\mathbf{x}}$ é unha simetría de $\mathbf{x}$. O problema que motiva este texto require modelizar un grupo de traballadores que son intercambiables entre si, e polo tanto estas simetrías estarán presentes. Cando isto acontece, deben impoñerse no modelo restricións adicionais, dirixidas en exclusiva a evitar que varias permutacións dunha solución dada se poidan considerar solucións factibles.

1.2. *Python* e *Pyomo*. Algúns conceptos básicos

Este traballo require que se programen tanto a definición e resolución do modelo de optimización a propoñer no capítulo 3, como os simuladores descritos nos capítulos 4 e 5. Este labor realizarase enteiramente na linguaxe de programación *Python*. Polo tanto, e sen querer entrar a explicar dende cero a lóxica desta linguaxe, comentaranse aquí algunhas das súas facetas, especialmente aquelas que se vaian mostrar en vindeiros capítulos. Os contidos que aquí se explican, así como todos aqueles que poida precisar un iniciado nesta linguaxe, están dispoñibles en [13].

En vindeiros capítulos, así como no apéndice A, mostraranse algunhas liñas de código que indican os datos de entrada (i.e., os valores dos parámetros) a partir dos cales se executan os programas. Esta información vén recollida en dicionarios de *Python* coma o da Figura 1.1. Como se pode ver neste exemplo, un dicionario consta dun conxunto de pares chave-valor. As chaves, que poden ser números ou cadeas de texto, son as etiquetas a través das cales se poden acceder aos valores. Estes últimos non só poden ser numéricos, senón que poden representar calquera estrutura de *Python* como listas, obxectos, ou outros cicionarios. Seguindo co exemplo, a expresión `dicionario['Chave_1']` representa a `Valor_1`. Para acceder a `Elem_3` abondaría con escribir `dicionario['Chave_2'][2]`, e finalmente, pode chamarse a `Valor_31` mediante a expresión `dicionario['Chave_3']['Chave_31']`. Esta é a lóxica que seguen os dicionarios en *Python*.

Outra funcionalidade de *Python* que convén mencionar, dado o seu protagonismo na construción do simulador dos capítulos 4 e 5, son as clases e os obxectos. As clases non son outra cousa que un medio para crear datos (chamados obxectos) que presenten unhas catacterísticas comúns entre si. A Figura 1.2 mostra a definición dunha clase moi sinxela, e permite visualizar as súas compoñentes. A

[3] *Linnear Programming Problems*

```
dicionario = {
        'Chave_1': Valor_1,
        'Chave_2': [Elem_1, Elem_2, Elem_3]
        'Chave_3': { 'Chave_31': Valor_31, 'Chave_32': Valor_32 }
}
```

Figura 1.1: Exemplo de dicionario en *Python*.

información sobre as características dos elementos dunha clase (os obxectos) denomínanse atributos, e defínense mediante unha función `__init__()` como a que se aprecia no cadro. Neste caso, todos os obxectos da clase `Produto` que se vaian definir terán exactamente tres atributos: `nome`, `prezo` e `unidades`. A figura tamén mostra a creación do obxecto `produto_1`, onde xa se especifican os valores que terá cada atributo, así como o modo de invocar estas variables. Pero ademais disto, unha clase permite definir métodos, que en esencia son funcións que modifican o valor dun ou de varios atributos. Neste exemplo creáronse dous métodos para modificar de xeito intuitivo o `prezo` ou `unidades` de calquera obxecto da clase `Produto`.

No sección 5.1.2 volta a falarse deste tipo de datos, pois son a base do simulador que nese capítulo se expón. Se ben a pretensión neste escrito é mostrar a menor cantidade de código posible, por estar desenvolvido no marco dun convenio de prácticas cunha empresa privada, é importante coñecer a nivel superficial en que consisten as clases e os obxectos, para ser así conscientes da utilidade que teñen neste traballo.

```
class Produto:
        def __init__(self, nome, prezo, unidades):
                # Nome do produto
                self.nome = nome
                self.prezo = prezo
                self.unidades = unidades

        def revalorizar(self, novo_prezo):
                self.prezo = novo_prezo

        def reposicion(self, n_prods):
                self.unidades += n_prods

produto_1 = Produto(nome = 'Bolsa_Lambetadas', prezo = 2.20, unidades = 12)
# Para visualizar o prezo do produto_1
print(produto_1.prezo)
# Para engadir 20 unidades do produto_1
produto_1.reposicion(20)
```

Figura 1.2: Exemplo de clase e obxectos en *Python*.

Para poder resolver o modelo que se describirá no capítulo 3, será necesario traducilo a unha libraría de *Python* que teña a capacidade de ler o modelo e enviarllo a un *solver*, é dicir, a un *software* deseñado para resolver problemas de programación matemática. Neste traballo empregarase *Pyomo*, que permite definir todos os elementos dun modelo de xeito moi intuitivo. *Pyomo* permite definir dous tipos principais de modelos: concretos, que levan incorporados a definición explícita de conxuntos e parámetros; e abstractos, que se definen sen estes valores, de modo que se poden executar instancias diferentes de dito problema sen máis que facilitarlle novos conxuntos de datos. En [5] explícanse os diferentes elementos que conforman o `AbstractModel`. Na Figura 1.3 escribimos un modelo moi sinxelo, pero que mostra os principais ingredientes dun problema de optimización.

Despois do código mostrado na Figura 1.3 deberíase escribir tanto as sentenzas que insertasen nel un conxunto de datos de entrada (por exemplo un dicionario, onde as chaves serían os nomes

```
import pyomo.environ as pyo

model = pyo.AbstractModel()

model.N = pyo.Set()
model.M = pyo.Set()
model.d = pyo.Param(model.N,model.M)
model.P = pyo.Param()
model.x = pyo.Var(model.N, model.M, domain=pyo.NonNegativeIntegers)


def funcion_obxectivo(model):
        return sum(model.d[n,m]*model.x[n,m] for n in model.N for m in model.M)
model.obx = pyo.Objective(rule=funcion_obxectivo)

def restricion(model, n):
        return sum(model.x[n,m] m in model.M) <= model.P
model.res = pyo.Constraint(model.N, rule=restricion)
```

Figura 1.3: Escritura dun modelo abstracto sinxelo en *Pyomo*.

dos parámetros), e aquelas que chaman a un *solver* e devolven a solución. Novamente, este traballo mostrará moi pouco código do escrito durante as prácticas, e ningún sobre a escritura do modelo en *Pyomo*, pero é importante introducir ao lector á lóxica baixo a que se traduce o problema ao ordenador. Tamén é interesante a poñer en valor a redacción dos ficheiros. Pode verse no exemplo que escribir un modelo tan reducido toma un número importante de sentenzas. Se se leva isto a un problema complexo, compréndese como a implementación do modelo do capítulo 3 en *Pyomo* tomou case 300 liñas de código[4].

[4]Incluíndo neste cómputo os comentarios, mais non os datos de entrada.

Capítulo 2

Descrición do problema. Revisión bibliográfica

O presente capítulo está adicado a explicar en detalle o problema que se pretende resolver, así como todos os factores que inflúen nel. Tras afondar nas características do problema poderanse comprender as particularidades que presenta e en que difire dos problemas de loxística de almacén que usualmente se tratan na literatura matemática.

O proceso en que se basea este TFM enmárcase na área de loxística de Inditex e forma parte da distribución de prendas de roupa a diferentes tendas. Dende este punto, será habitual referirnos ás tendas como TSA[1]. Trátase de optimizar o funcionamento do clasificador, ao que tamén nos referiremos como *sorter* de xeito habitual. O clasificador recibe caixas coas prendas, traídas dende o almacén; fai circular os produtos por un circuito, e deposítaos en caixas asignadas a cada tenda en función dunha demanda prestablecida.

A singularidade deste problema é, por unha banda, a presenza destacada do factor humano. O *sorter* non opera de xeito autónomo mediante elementos robóticos, como acontece en gran parte das publicacións do campo da loxística de almacén, senón que precisa de persoas físicas realizando labores concretos. Ademais, a incidencia deste factor humano é moi relevante no tempo necesario para satisfacer completamente unha demanda dada, tempo que na literatura anglosaxoa adoita denominarse *makespan*. Pola outra banda, o aspecto que buscamos resolver do problema de optimización asociado ao clasificador non é tanto este *makespan*, senón a asignación concreta de traballadores aos diferentes roles de traballo que desemboca no reparto[2] máis curto posible. Isto é, que se dispoñemos de n traballadores e hai un total de 4 roles ou tipos de traballo, o problema consiste en buscar a asignación óptima deses n traballadores aos diferentes roles. Este feito é o que lle outorga o título a este traballo: *Sorter Assignment Optimization*.

Despois de por en coñecemento as entrañas do problema, recóllese neste capítulo unha revisión bibliográfica a modo de estado da arte. A finalidade é comprobar como tras unha busca de artigos de investigación sobre clasificadores, optimización en loxística de almacén e outros temas relacionados, non se obtén apenas material que axude a modelizar este problema en particular. Para cada artigo haberá un ou varios motivos, pero seguramente o máis recorrente é que se tratan procesos de almacén con compoñentes robóticos no canto de traballadores humanos. Ademais, as publicacións dos últimos anos están moi enfocadas ao sector da venta en liña ou *e-commerce*, que xera tipoloxías de problema moi diferentes ao do que aquí nos ocupa.

[1] Siglas en castelán de *Tienda Sección Aparte*

[2] Nome co que nos referiremos ao proceso completo.

2.1. Funcionamento do clasificador

Para explicar como opera o clasificador, o máis axeitado será dividilo en dous procesos: a indución dun lado, e a caída en destino máis a evacuación polo outro. Correspóndense, respectivamente, coa entrada das prendas no circuito, a súa circulación e caída nunha caixa, e finalmente extraer as caixas de demanda segundo se van enchendo. Esta división tamén axudará a entender mellor os catro roles de traballo. Dous deles relaciónanse co proceso de indución e os dous restantes coa evacuación e a posibilidade da caída dunha prenda en destino.

O *sorter* é, *grosso modo*, un circuito en bucle polo que avanza un conxunto de bandexas dun modo similar ao dunha cinta transportadora.

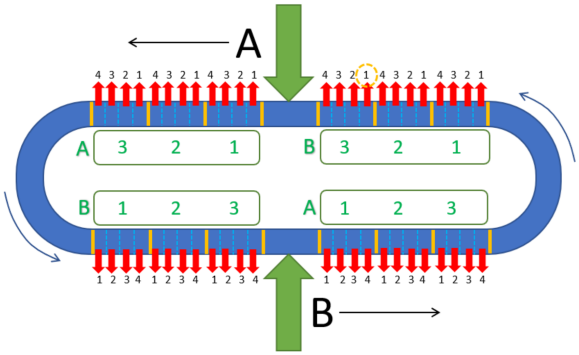


Figura 2.1: Esquema dun clasificador circular. Deseño creado dende a área de loxística de Inditex.

Un clasificador circular, coma o que se mostra na Figura 2.1, presenta dous lados, simbolizados no esquema como os A e B de maior tamaño. En cada lado podemos diferenciar claramente dúas compoñentes. A primeira é a zona onde se realiza a indución, que se presenta mediante as frechas verdes apuntando cara o sorter. Supoñen a entrada de prendas que avanzarán no sentido da marcha cara os xutes ou destinos. Estes son o segundo elemento visible no esquema. Cada unha das frechiñas vermellas representa un xute ou destino, que consiste nun conducto que finaliza nunha caixa onde caerán as prendas que sexa conveniente.

Pódese advertir no debuxo unha certa estruturación dos destinos. En efecto, o conxunto de destinos dun mesmo lado divídese en dous grupos (representados polas letras verdes da Figura 2.1). Cada grupo divídese en sectores (números en cor verda) e cada sector comprende un certo número de xutes (frechiñas vermellas). Esta ordenación dos destinos adquirirá máis sentido cando se explique a caída en destino e a evacuación.

Ao ter o *sorter* dous lados, permítese que unha prenda entre ao clasificador no lado A e caia nun destino do lado B, ou incluso que dea unha volta enteira e retorne ao lado A. Isto último non é recomendable, porque a bandexa que ocupa circularía máis tempo chea. Pero isto pode acontecer se, por exemplo, non lle é posible caer en ningún dos destinos dese lado e debe continuar entón a marcha. Diremos que se produciu unha recirculación, ou que determinada prenda recircula ou dá unha volta, se a indución se produciu nun lado e a caída en destino se dá nese lado, pero despois de pasar polo lado oposto. Se o produto se induce nun lado e cae nun destino do outro (na mesma volta), falaremos de media recirculación ou diremos que a prenda dou media volta.

Non todos os clasificadores teñen esta estrutura circular, pero será esta a que tomaremos como caso de estudo por indicación do titor da empresa. Por exemplo, o *sorter* que tiven ocasión de visitar non segue esta estrutura. Neste caso, o circuito contemplaba dous carriles de bandexas que circulaban en paralelo. Se ben hai algunhas diferenzas respecto do *sorter* circular, a grandes trazos podemos imaxinar que se toma un clasificador circular e se enrosca en forma de oito, que se acaba dobrando á metade, polo que a forma final é similar á dunha curva de Viviani se se puidese aplanar.

2.1.1. Indución

O primeiro paso do clasificador é a entrada de mercadorías no mesmo. No sorter están habilitados uns espazos de traballo, que chamaremos postos de indución, onde un traballador (indutor) coloca os obxectos nas bandexas. Definimos entón o primeiro dos catro roles de traballo, o encargado de introducir as prendas nas bandexas que pasan fronte ao posto de indución nese momento.

A roupa chega ao posto de indución en caixas, procedentes do almacén, a través dun circuíto á parte sobre o que non é preciso afondar para este problema. Unha caixa que chega a un posto de indución debe ser aberta e escaneada cun lector antes de ocuparse do seu contido. Polo tanto, se non hai ningunha caixa aberta, o indutor non pode continuar colocando prendas e debe deterse a abrir unha caixa nova.

A rotura das caixas procedentes de almacén pode parecer unha anécdota, mais na práctica é o procedemento que máis contribúe a ralentizar a indución no *sorter*. Para tratar de mitigar esa situación, creouse un rol de apoio: os *waterspiders*. Un *waterspider* é unha persoa que circula polos postos de indución deténdose a axudar aos indutores que están a se quedar sen prendas que colocar. Así, o *waterspider* trata de recortar, na medida das súas capacidades, o tempo que se adica a romper caixas no canto de inducir máis prendas.

Cantos máis indutores se asignen ao sorter, a maior velocidade se introducirán as prendas e antes se rematará de inducir a totalidade da demanda, ou iso nos indica a intuición. Sobre isto, é preciso facer algúns comentarios.

Primeiramente, os indutores non traballan ao mesmo ritmo. Debe comprenderse que o primeiro indutor no sentido da marcha do circuito terá á súa disposición unha maior cantidade de bandexas libres en que pode inducir, pois só estarán ocupadas aquelas correspondentes aos obxectos que recirculan (é dicir, que non puideron caer antes en destino e continúan en circulación). Polo tanto o número de prendas que induce está limitado pola capacidade física do traballador e se ten que abrir ou non algunha caixa. O segundo traballador verá ocupadas as bandexas citadas pero tamén todas as que induciu o operario anterior, polo que colocará menos produtos no clasificador, e así sucesivamente. Poderíamos dicir que a aportación de cada novo traballador respecto dos anteriores é cada vez menor (Figura 2.2).

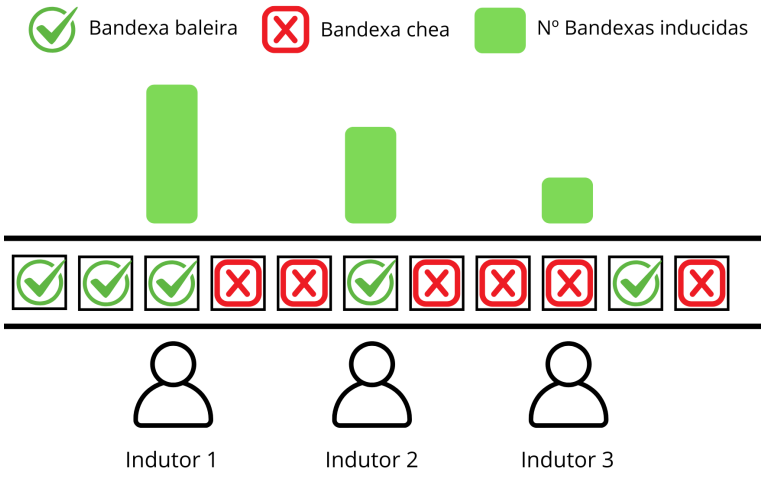


Figura 2.2: Esquema representativo da desigualdade na carga de traballo dos indutores. Elaboración propia.

Ademais, a cantidade de postos de indución dispoñibles limitará, como non pode ser doutra maneira, o número de traballadores que poden exercer ese rol. Os *waterspiders* estarán limitados polo número

de indutores, pois non ten sentido que un rol de apoio teña máis peso que aquel ao que se apoia. En particular, non tería sentido ter *waterspiders* fixos nos postos de indución por seren moitos, e menos aínda que chegase a haber máis dun por posto. Tamén a aportación (neste caso en forma de redución de tempo) de cada novo waterspider respecto dos anteriores é menor ao aumentar o seu número, como resulta lóxico.

Finalmente, é claro que asignar traballadores aos roles de indutor ou *waterspider* leva a que menos persoas se encarguen da evacuación. En casos extremos a abundancia de operarios na indución podería levar a que as evacuacións sexan moi escasas, co cal os destinos deixarían de estar dispoñibles por estar bloqueados. Neste caso as prendas recircularían en exceso, provocando que cada vez houbese menos bandexas baleiras sobre as que inducir, e polo tanto habería persoas case sen carga de traballo en indución. A virtude dunha boa asignación, en suma, está no equilibrio entre os factores que se acaban de expoñer, polo menos no que involucra á indución.

2.1.2. Caídas en destino e evacuación

Ao longo do circuito hai localizados un conxunto de fotocélulas, de modo que se unha bandexa que foi inducida non ten un destino asignado, ao pasar por diante dunha fotocélula, impónselle un. En particular, cando un obxecto é inducido, este sensor asigna á bandexa corresponente un xute concreto do *sorter*. A bandexa circulará a través do clasificador cara ese destino sen abrirse en ningún dos anteriores. Cando a bandexa se sitúa sobre el, ábrese e deixa caer o contido sempre que sexa posible. Se non o é, como queda sen xute asignado, ao pasar por diante da seguinte fotocélula asignaríaselle un novo, repetíndose o proceso.

Os criterios para que unha bandexa se abra e deixe caer o seu contido son os seguintes. Primeiramente, ese debe ser o destino que lle foi asignado previamente. Pero tamén se debe verificar que o destino non estea bloqueado. Un xute estará bloqueado sempre que nese destino ou nun do seu mesmo sector[3], a caixa que acolle as prendas caídas se encha. Cando esta se substitúa por unha nova caixa baleira, todos os xutes dese sector voltarán a se desbloquear.

O procedemento consistente en retirar unha caixa chea baixo un destino, e substituíla por unha nova caixa baleira, ademais de agregarlle o etiquetado correspondente, denomínase evacuación. Os traballadores que se adican a evacuar destinos denomínanse *packers*, se ben tamén o seu rol de apoio, os varios, realizan este labor. Un *packer* é un operario que ten asignado un certo conxunto de destinos, normalmente unha cantidade de sectores dun mesmo grupo. Un *packer* pode estar asignado a sectores enteiros ou pode compartir algún sector con outro compañeiro para que o reparto total sexa máis equitativo. Neste último caso falaríamos dun *packer multisector*, pero neste traballo limitarémonos a asignar sectores enteiros a estes traballadores. Aqueles designados como varios percorrerán os destinos do sorter, habitualmente restrinxíndose a un dos lados do clasificador (pois cambiar de lado supón percorrer unha distancia moi longa). Cando estes detectan que un packer se atopa saturado, isto é, que ten que atender en certo momento varias caixas cheas[4], proceden a axudalo e evacuar algunhas das que lle quedan pendentes. É dicir, actúan coma un packer itinerante que socorre aos traballadores que nun certo momento presentan unha carga de traballo superior.

2.1.3. Sobre a demanda

Os xeradores de demanda son as tendas ou TSAs. Concédese a cada unha delas un conxunto de destinos acorde co tamaño da demanda, de modo que cada destino activo[5] do sorter ten unha TSA asignada, polo que as caixas de dito xute satisfará a demanda desa tenda. Considérase que o reparto finaliza no momento en que a última prenda cae en destino, de modo que se completa a demanda de

[3] Lémbrese que os destinos se organizan e identifican mediante o seu lado, grupo, sector e xute concreto do sector.

[4] Isto pode provocar, se se atopan en sectores diferentes, que unha gran cantidade de destinos poidan permanecer bloqueados mentres non se evacúan as caixas

[5] Nun *sorter* non é necesario que todos os destinos sexan funcionais, en especial cando o reparto a realizar é reducido. Nestes casos considérase empregar menos xutes e así dispersar menos aos *packers*. En todo caso, a cantidade de destinos a considerar en cada problema concreto vén sempre determinado de antemán.

todas as TSAs. Realmente remataría ao evacuárense as caixas unha vez caída toda a roupa, pero a efectos da construción dun modelo, será máis doado pensalo deste modo.

Cada unha das tendas presentará unha demanda concreta de prendas de diferentes referencias. Entenderemos neste texto que unha referencia é un sinónimo de SKU (*Stock Keeping Unit*), isto é, o identificativo que un distribuidor dá a un produto a fin de poder contabilizar as existencias dese tipo de artigo. Por exemplo, unha referencia que pode demandar unha tenda é 'Camisola Negra Básica XL'. Por suposto, non se esixe que todas as TSAs demanden artigos de todas as referencias, pois isto depende da ubicación e tamaño da tenda en cuestión.

Este traballo non vai afondar no método que se segue para asignar cada destino a unha tenda. No modelo do capítulo 3 asúmese que esta asignación xa vén dada, así como no simulador do capítulo 5.

2.1.4. Sobre a asignación de destinos a prendas

Como comentario final, en parágrafos anteriores mencionouse que ás bandexas cargadas, ao pasar diante da fotocélula, impoñíaselles un destino. Este destino escóllese empregando un tipo de algoritmo que popularmente se coñece como *Round-Robin*. É dicir, un algoritmo que trata de repartir as prendas entrantes de forma relativamente equitativa, a fin de que a caída de obxectos en destino non se concentre en zonas concretas. Esa saturación daríase, por exemplo, se as prendas caesen en cascada, isto é, no primeiro destino que atopen e demande esas prendas.

Non se tratará, nin se me facilitou información sobre o algoritmo empregado, pero buscaremos definir un *Round-Robin* básico cando tratemos de simular un clasificador. Tamén é de importancia este concepto á hora de apreciar a complexidade do *sorter* e plantexar se un modelo matemático é máis ou menos útil para representalo.

2.2. Estado da arte. Revisión bibliográfica

Unha vez descrito en extensión o problema a resolver, o seguinte paso lóxico é realizar unha busca bibliográfica para coñecer o 'estado da arte', así como tentar achar algún caso similar e cuxo modelo poida servir como base do traballo que aquí se desenvolverá. Consultáronse artigos centrados na modelización e resolución dos problemas de optimización matemática derivados, mais tamén se acudiu a estudos destinados a construír simuladores ou desenvolver algoritmos (xenéticos en gran parte) para resolver os casos concretos que plantexan. Con todo, este labor resultou certamente infrutuoso.

O problema de asignación que se definiu neste capítulo nada ten que ver con aqueles máis explorados na literatura. Esta, ademais, está cada vez máis centrada no sector do *e-commerce*, é dicir, no reparto de produtos solicitados en liña por particulares. Os resumos da meirande parte dos artigos que comentaremos a continuación (como [12] ou [4]) inician con referencias ao crecente interese nos textos científicos sobre os almacéns e clasificadores adicados a este tipo de demanda. En [3] afóndase algo máis no grao de diferenza que supón o sector do *e-commerce* ou B2C[6] respecto das cadeas de suministro a tendas ou B2B[7]. Como se describe neste texto, na venda en liña, a demanda consta dunha gran cantidade de demandantes dun pequeno número de produtos. Estes paquetes están suxeitos a tempos de entrega moito máis axustados (dado que a celeridade na recepción adoita ser un reclamo comercial das empresas) e presentan unha carga de traballo moi variable no tempo, con picos nos luns[8], por exemplo. Estas son só algunhas das características que se mencionan, pero son as que máis interesan no contraste entre os problemas da literatura e o que se aborda neste traballo.

En efecto, os modelos construídos en [6], [4] e [12] comparten que, ademais de consistir a demanda en grandes pedidos de reducido tamaño, o lugar de destino de cada produto considérase coñecido de antemán. Esta suposición ten sentido no comercio en liña, pois os pedidos que entran no clasificador están en certo sentido personalizados e, como o final destes procesos é o transporte das mercadorías, xa

[6] Abreviatura da frase anglosaxoa *Business-to-consumer*.
[7] *Business-to-business*.
[8] Debidos a que é o día en que se procesan os pedidos da fin de semana.

se coñece por que zona abandonarán o *sorter*. Ademais, o número de 'portas' polas que saen, por chamar dalgún modo ao análogo nestes textos do que aquí coñecemos como 'destinos' é significativamente máis reducido. Isto débese a que, nos modelos dos artigos citados, os paquetes de diferentes demandantes abandonan o clasificador pola mesma zona.

Todo o anterior é moi diferente ao que acontece co problema descrito neste capítulo. A demanda do noso *sorter* constará dun número contido de demandantes (dado que son as tendas ás que ese almacén concreto suministra), onde cada un solicita unha gran cantidade de produtos de diferentes SKUs. Ademais, neste problema cada xerador de demanda ten asignados un ou máis destinos, é dicir, que non se dará o caso en que a demanda de dúas TSAs diferentes caia no mesmo xute. Por aportar un exemplo concreto, as instancias de maior tamaño que se mostran para o modelo descrito en [4] consisten nun clasificador con 250 bandexas e 10 *packing lanes*, o análogo aos destinos[9]. Na sección 5.2 deste traballo chégase a traballar con 1500 bandexas e 288 destinos, o cal reflexa a gran diferenza en tamaño. A suposición de coñecer de antemán o lugar de caída de cada paquete é moi forte para o noso caso, e nalgúns textos como [12] vaise máis alá, supoñendo coñecido, para cada destino, a orde dos obxectos que caerán nel. Para o clasificador do noso problema, as TSAs presentan demandas de prendas de roupa moi similares, polo que non ten sentido que se dea esta asignación. Cando se induce unha prenda si que se lle asigna un destino, pero se ao aproximarse a el este está bloqueado, indicaráselle un novo lugar de caída para non ter que dar outra volta completa ocupando a bandexa. Isto que se acaba de sinalar, por si mesmo, xa supón unha fenda considerable respecto dos modelos de optimización propostos na literatura.

Outro punto a destacar é que o tipo de problema (a asignación de persoas a roles de traballo), non é dos principais que se estudan derredor dos clasificadores de almacén. Os máis habituais son de tres tipos: *layout dessign*, o deseño do clasificador ou dunha parte do mesmo; creación de rutas, para robots ou humanos; e a creación dunha lista ordenada dos pedidos a atender. Por exemplo, o *sorter* descrito en [6] é, na súa estrutura base, moi parecido ao que pretendemos modelizar, pero o problema que resolve é de *layout dessign*, dirixido a colocar atallos que conecten da forma máis eficiente posible pares de localizacións polos que pasa a cadea de transporte[10]. O clasificador a modelizar en [4] tamén ten características semellantes[11], pero ademais de se tratar dun problema de *scheduling* (i.e., determinar a orde de procesamento dos pedidos), as dimensións son moito máis reducidas do que aquí interesa, motivadas polos comentarios xa realizados sobre o uso dos destinos. Máis alá da revisión bibliográfica que realizan os artigos da bibliografía, [7] e [12] inclúen táboas[12] onde se pode apreciar gráficamente como o estado da arte expresado polos autores consiste maioritariamente en problemas dun dos tres tipos que se acaban de comentar, ou no seu defecto, da combinación dos mesmos.

Outro inconveniente á hora de atopar bibliografía semellante ao problema que aquí se trata é a coñecida como *batching policy*[13], novamente moi ligada ao sector do *e-commerce*, empregada nos textos xunto coa *zoning policy*. Esta última si que se verificaría no noso caso, pois supón simplemente que o clasificador conta con zonas diferenciadas nas que unha tarefa se pode paralelizar. Isto correspóndese no *sorter* deste traballo cos postos de indución, que permiten que unha tarefa se realiza á vez en diferentes lugares. Non obstante, a *batching policy* asume que os produtos se poden xuntar en lotes, de modo que se transportan todos os seus elementos á vez. En [8] estúdase (se ben non dende a modelización) un clasificador que se adica só a diferenciar entre paquetes grandes e pequenos, pero tamén inclúe esta lóxica na súa formulación matemática do problema.

Unha parte significativa dos artigos que se atoparon nas primeiras buscas estaban adicados a problemas protagonizados pola figura dos AGVs [14], robots autónomos programables para o transporte de paquetes entre zonas designadas do almacén. Estes problemas mesturan o deseño de rutas coa ordenación e a asignación de produtos a cada vehículo, coma en [12]. Os elementos máis característicos

[9] As instancias poden consultarse nas táboas 4, 5 e 6 de [4]
[10] Representación gráfica do problema na Fig. 2 de [6]
[11] Pode consultarse a Fig. 1 de [4]
[12] *Table 1* en ambos os dous textos.
[13] Nome utilizado en [2], [3] e [4]
[14] *Automated Guided Vehicles*

destes problemas son as zonas onde os AGVs agardan e/ou cargan batería, as posibles localizacións onde toman un paquete (neste caso uns brazos ao longo da cinta de transporte), e aquelas onde o depositan (un camión para o caso). Outro exemplo é [10], que suma ao anterior tanto o enfoque multiobxectivo, que busca minimizar o *makespan*[15] pero tamén o número total de AGVs e o consumo eléctrico; así como a resolución mediante algoritmos xenéticos. Este tipo de métodos de resolución tamén se estudan en [6] e [9], ademais de seren mencionados nas revisións bibliográficas de [3] e [2].

Na pretensión de achar clasificadores con maiores similitudes, [2] contén unha dilatada explicación de moitos tipos diferentes de *sorters*, pero sempre automatizados por completo. O máis parecido ao que nos ocupa son quizais os ASS[16] orientados á industria postal, mais seguen a presentar fendas insalvables coma a asignación da demanda aos destinos[17]. Ademais, ningún dos tres tipos de problemas que o artigo trata para estes clasificadores se pode conectar sustancialmente co noso caso.

Con todo, atopouse en [9] un problema de asignación de traballadores a roles de traballo. A Figura 6 deste artigo representa tanto o clasificador como as zonas onde é posible asignar unha persoa. Cóntase cun número de traballadores fixos con cadanseu custo por xornada, así como traballadores parciais que poden ser chamados ou non, e cun custo maior. Este problema céntrase na perspectiva económica, cuantificando por exemplo a perda que supón que certo traballador non chegue a tempo a unha determinada localización. Ademais de que as tarefas non son do todo extrapolables ao noso caso por requerir unha maior carga de traballo manual (como transportar obxectos, o cal non acontece no noso clasificador), este artigo non tenta crear un modelo de optimización, senón que emprega a simulación e os algoritmos xenéticos para buscar a mellor asignación. Neste caso si que é posible establecer un nexo entre o problema tratado en [9] e o noso, pois o capítulo 4 deste traballo adícase á inviabilidade na práctica do modelo de optimización clásico como vía de resolución, inclinándonos polo uso dun simulador que comprobe diferentes candidatos a solución óptima.

Atopouse outro artigo onde o factor humano ten un rol protagonista, mais que non se parece ao do noso problema. En [7] preséntase un problema de asignación da demanda aos destinos, pero que pon especial atención no esforzo físico que, como consecuencia da asignación, asumirán os traballadores. Trátase entón de minimizar tanto o tempo de cumprimento da demanda coma este esforzo físico. Novamente, o factor humano do *sorter* non se parece case ao deste traballo, ademais do modo de asignar os paquetes ás ubicacións de destino.

Rematando esta pequena revisión bibliográfica, pode concluírse que o problema que se trata neste traballo é moi diferente daqueles vistos na literatura. Ademais de comprender á vez as fases de indución e caída en destinos máis evacuación, que noutros textos aparecen por separado, o propio factor humano tamén xoga un rol moi particular. Finalmente, lonxe de buscar ordenar a demanda ou a súa correspondencia cos destinos, procúrase a mellor asignación das persoas a un de catro roles de traballo posibles. Se algún dos *sorters* vistos nesta revisión contén algún elemento visible no noso traballo é o de [4], pero só no sentido en que este trata individualmente ás bandexas do clasificador e que toma como medida de tempo aquel que transcorre dende que unha bandexa abandona a súa posición ata que ocupa a da súa contigua. Aínda así, isto verase no capítulo 5 co simulador, e non no modelo, pois un paso de tempo tan curto nun clasificador tan grande provocaría un número desmedido de variables.

[15] Tempo de cumprimento da demanda.

[16] *Automated Sorting Systems*

[17] Lémbrese que no noso caso non se coñece de antemán o destino ao que van as prendas, ademais de que a demanda de dúas tendas diferentes non pode caer no mesmo destino.

Capítulo 3

Modelo de Optimización proposto

Neste capítulo exponse en detalle o modelo de programación matemática proposto para resolver o problema de asignación dos traballadores no clasificador. No capítulo anterior tratouse con detemento este problema, dado que sen esa explicación é certamente complicado poder entender correctamente o modelo proposto, polo menos nunha primeira lectura. Nesta liña, a presentación dos elementos do modelo realizarase de modo pausado e ben motivado, a fin de que calquera persoa allea ao contexto do problema poida acadar un bo nivel de comprensión das implicacións dos parámetros, variables e restricións que o conforman.

Cómpre destacar que o modelo foi deseñado partindo de cero, sen ser unha modificación de ningún outro co que xa contara a empresa. Se ben é certo que a compañía dispoñía dalgún modelo, optouse ao inicio das prácticas por non mostrarmo, agardando que afrontase o problema dende outra perspectiva e propuxese un modelo substancialmente diferente.

Como é de supoñer, non é posible reflectir a totalidade do visto no capítulo anterior nun modelo matemático: entre outros aspectos, débese tentar conter o número de variables na medida do posible. En consecuencia, a primeira sección deste capítulo adicarase a expoñer as suposicións e imposicións que se realizan sobre o modelo, a fin de simplificalo. Posteriormente presentarase a formulación na súa totalidade, definindo parámetros, variables e contextualizando cada bloque de restricións do mesmo. O primeiro documento do Apéndice B recolle de xeito esquemático este modelo matemático baixo o nome *Modelo Multirreferencia V1.0*. Este apéndice está formado por documentos realizados durante as prácticas, referidos exclusivamente á modelización e en texto, dado que neste traballo será moi pouco o código informático que mostremos.

No seguinte capítulo analizarase o comportamento do modelo á hora de resolver problemas de tamaño crecente. Porén, será sinxelo deducir ao longo da exposición que, para casos reais, o número de variables e restricións será excesivo, e por ese motivo o proceso de modelización detense neste punto. Non obstante, durante o desenvolvemento das prácticas creáronse outras versións do modelo con novas e/ou mellores funcionalidades. A última sección deste capítulo comenta brevemente en que consisten estas melloras, que se recollen nos dous documentos restantes do Apéndice B para a súa consulta, se for pertinente.

3.1. Simplificando a realidade. Suposicións do modelo

Como se expresaba nos parágrafos anteriores, os procesos que se dan no clasificador presentan un nivel de complexidade que obriga a ter que escoller varias vías de simplificación da realidade ao momento de traducir o problema a un modelo de optimización matemática clásico. Neste apartado exporase *grosso modo* a lóxica que seguirá o modelo, e polo tanto presentaranse as imposicións que sobre el se realizarán.

Estes supostos parten dunha reflexión sobre o obxectivo que se quere cumplir, e a partires desta

premisa decídese que aspectos do *sorter* real é imprescindible manter no modelo matemático. Aqueles que non son indispensables, polo tanto, son susceptibles de se omitir, a fin de conter o número de variables e restricións necesarias. Tamén se debe establecer, por exemplo, como tratará o modelo o paso do tempo ou como se contabilizan as prendas xa repartidas, as que se atopan no sorter ou que aínda non entraron nel.

O primeiro aspecto a tratar, e un dos que máis afectación terá sobre o tamaño do modelo, senón o que máis, é como se pretende modelizar o paso do tempo. Dado que se trata de conseguir un funcionamento do clasificador o máis eficiente posible en termos de prendas procesadas, debemos poder contemplar o tempo necesario para que unha cantidade dada de prendas (entre outras condicións iniciais) fagan o seu percorrido completo polo sistema. Neste caso, e como se explicou no capítulo anterior, o clasificador opera de modo circular (Figura 2.1), onde aquelas prendas que non entran, ou saen permanecen circulando.

O máis exacto, a efectos de reflectir o funcionamento do *sorter* nun modelo, sería tomar como unidade de tempo aquel que transcorre entre que unha bandexa deixa de ocupar unha posición dada no sorter e pasa a ocupar a seguinte. Tamén sería posible traducir todas as duracións a unidades de tempo reais coma os segundos, pero complicaría aínda máis o proceso, sendo máis viable tomar como unidade de tempo ese 'paso de bandexa' e logo traducir o resultado final a unidades reais. Non obstante, implementar esta lóxica é inviable para a programación matemática clásica, pois o conxunto de saltos temporais sería dun tamaño desproporcionado para casos reais, e ningún ordenador soportaría indexar variables nun conxunto así.

Dito isto, a unidade básica de tempo que se empregará no modelo matemático é o ciclo ou revolución. Para mellor comprensión deste concepto, convén imaxinar o clasificador parado, ou o que é o mesmo, calquera foto fixa que se poida tomar do mesmo. Se se asignase unha etiqueta a cada bandexa do sorter, comezando por unha bandexa de referencia á que se lle daría o número 1, pode afirmarse que o clasificador realizou unha revolución completa cando a bandexa 1 volta á posición que ocupaba no tempo inicial fixado (a Figura 3.1 mostra graficamente esta idea). No trascurso dunha revolución, unha bandexa pode recibir unha prenda ou non e, en caso de estar chea, pode deixar caer o seu contido ou non. Polo tanto, é unha medida de tempo que permite implemetar as demais mecánicas do sorter sen problema. Tamén supón aforrar variables de forma significativa, dado que cada revolución toma nun caso real un par de minutos, fronte ao 'paso de bandexa' mencionado, que equivale a menos dun segundo. Así, o conxunto de revolucións ten un tamaño menor, e como se deben indexar variables nel, tamén se conterá a súa cantidade total.

O seguinte aspecto que debe clarificarse é o tratamento das prendas no modelo, especialmente na caída en destino e a evacuación. O modelo debe contemplar o traballo con obxectos de diferentes SKUs [1]. Tómense como exemplos de SKU unha camisola branca da talla S e un suadoiro negro da talla XL[2]. Resulta evidente que estas dúas prendas non ocupan o mesmo espazo, pero o modelo ten que prever que as diferentes prendas poidan caer nunha mesma caixa, e recoñecer cando esta chega ao seu límite de capacidade. Debe polo tanto implementarse no modelo o volume que ocupa cada prenda dun determinado SKU, así como a capacidade das caixas ubicadas en destino. Para controlar cando un xute se satura (é dicir, énchese a súa caixa), debemos ademais contemplar un conxunto de variables que actúen a modo de contador do volume ou dos obxectos que un destino acolle nun determinado momento. Neste caso, e en contraste co exposto a propósito do paso do tempo, para o volume non é necesario establecer unhas unidades concretas, dado que o único relevante é que estas sexan as mesmas para prendas e para caixas, de modo que as operacións teñan sentido. Polo tanto, para realizar probas ou para casos ficticios, é viable tomar unidades de volume imaxinarias con valores enteiros, sempre que teñan sentido no problema.

En relación co anterior, deben de se aplicar tamén simplificacións na tradución das prendas ao sorter que se pretende modelizar. Non terá sentido algún tratar cada prenda de xeito individual, dado que na realidade un reparto implica decenas ou centos de miles delas. O enfoque que se emprega

[1] *Stock Keeping Units*

[2] Estes dous exemplos están moi simplificados, pois calquera mínima diferenza entre dúas prendas implica que teñan un SKU diferente, como o debuxo, o material, se é para home/muller/*unisex*, etc.

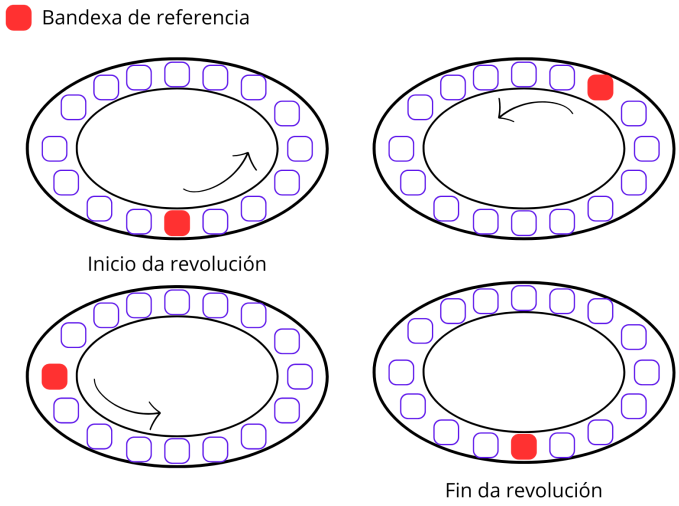


Figura 3.1: Esquema ilustrativo do modo de contabilizar as revolucións partindo dunha bandexa de referencia. Elaboración propia.

neste traballo é o de considerar só o total de prendas de cada referencia ou SKU que participan dun determinado proceso nunha certa revolución. En consecuencia, todos os obxectos dun mesmo SKU son iguais, no sentido en que se procesarán do mesmo xeito. Isto significa, dunha banda, que o modelo non admitirá que as prendas poidan caer fóra do *sorter*. Resumidamente, algúns paquetes, por inducírense na beira dunha bandexa, por exemplo, poden ser susceptibles de caer fóra do clasificador cando o movemento da cadea traza unha curva. Esta casuística está excluída do modelo. Da outra banda, cada prenda ocupa o mesmo volume unha vez cae no destino que se lle foi asignado. Isto tamén é unha simplificación necesaria, pois sería moi custoso computacionalmente modelizar a posibilidade de que unha prenda caia de xeito horizontal ou vertical, e como iso afecta á cantidade de obxectos que admite a caixa. Na Figura 3.2 pode apreciarse como unha mesma prenda dunhas dimensións concretas pode ocupar de xeito efectivo un volume maior ou menor da caixa, entendendo o 'volume efectivo' como a suma do propio da prenda máis aquel que imposibilita que ocupe un novo obxecto que caia proximamente. A solución aplicada, como se acaba de comentar, é supoñer un mesmo valor de volume efectivo para cada peza de roupa.

Respecto da demanda, explicouse que nun reparto (é dicir, nunha xornada de traballo planificada do clasificador) cada tenda ou TSA require unha determinada cantidade de prendas de cada SKU. Polo tanto, para cada instancia deste modelo matemático, a demanda será finita e virá dada por este *stock* de roupa de cada referencia por parte de cada establecemento. Como xa se comentou no capítulo anterior, as prendas do clasificador teñen asignado un destino ou xute onde caerán, e dito xute está ao servizo dunha tenda concreta. Na práctica, esta asignación TSA-destinos realízase despois de decidir que traballadores ocupan cada rol, pero como a finalidade deste modelo é decidir a mellor asignación destes roles de traballo, suporase que esta correspondencia foi xa establecida de antemán. O modo de calcular cantos destinos corresponden a unha tenda é en base á proporción da demanda que realiza, iso si, baseada no volume da mesma, pois é o que afecta ao ritmo ao que se enchen as caixas en destino. Existen conceptos técnicos a nivel da empresa como o de 'volume ideal', un coeficiente que axuda a realizar este cálculo, pero a efectos deste problema abondará con establecer unha repartición dos xutes ponderada respecto ao volume total que demanda cada tenda.

Finalmente, o aspecto do *sorter* que máis simplificado estará na súa modelización é o proceso de indución. Na realidade, esta fase do clasificador implica a apertura das caixas que traen a roupa dende o almacén, e incluílas no modelo non é viable. O principal motivo para afirmalo é que obrigaría

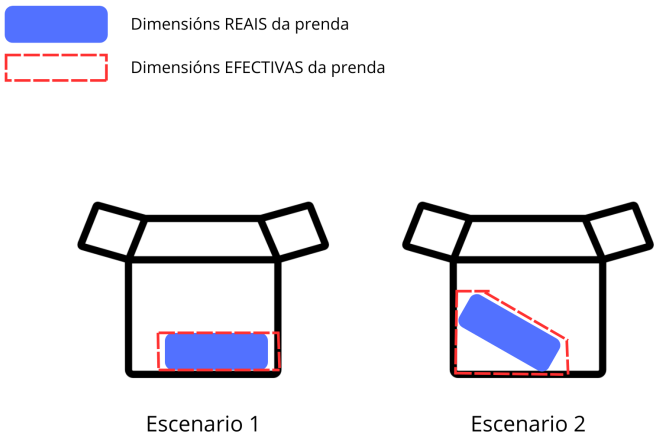


Figura 3.2: Esquema con dúas posibilidades de caída dunha mesma prenda na caixa en destino, e os volumes efectivos que ocuparían nela. Elaboración propia.

a tomar pasos de tempo máis 'finos' (é dicir, con intervalos de menor duración) para ter en conta o tempo que toma abrir cada caixa e inducir as prendas que contén. Non debe esquecerse que o obxectivo do modelo será obter a mellor asignación dos traballadores aos roles do traballo, o cal se corresponderá con obter o valor de catro variables, e polo tanto outros aspectos do clasificador tomarán unha posición de subordinación, sen permitirse que a súa inclusión no modelo lastre a capacidade de resolver computacionalmente as súas posibles instancias.

Como conclusión, o modo de implementar a indución no modelo será establecer un coeficiente de prendas inducidas por cada traballador ao longo dunha revolución. Esta constante pode interpretarse como un promedio de obxectos inducidos polo conxunto dos traballadores que ocupen o rol de indutor (dado que, como se explicou no capítulo anterior, coñécese que a cantidade de inducións non é homoxénea nos operarios), contando tamén co tempo que toma abrir as caixas procedentes do almacén. Adicionalmente, supoñemos que as distintas referencias son inducidas por orde, isto é, que se decide previamente unha orde para os SKU, de modo que as prendas dunha referencia non comezan a inducirse ata que non rematan de colocarse todas as do tipo anterior. Isto non supón unha simplificación tan notable, dado que na realidade, por proceder a roupa de caixas de almacén, adoita inducirse seguindo unha certa orde de chegada das caixas segundo o seu SKU.

Esta constante 'de rendemento' dos indutores non é o único coeficiente deste tipo que incluirá o modelo para simplificalo, dado que acontecerá o análogo para os roles de traballo restantes. Como os *waterspiders* axudan a mellorar a eficacia dos indutores, a súa aportación no aumento do número de prendas inducidas por revolución tamén se introducirá en forma de promedio. Esta lóxica trasladarase tamén aos *packers* e varios, mediante un valor representativo de cantas caixas é capaz de evacuar un traballador ao longo dunha revolución. É novamente unha simplificación importante, porque este novo 'promedio' non fai diferenza entre se os destinos que se enchen están próximos ou afastados entre si, pero claramente permite aforrar unha cantidade considerable de variables. Como pequena distinción, pode mencionarse que, mentres que os *waterspiders* non inducen prendas senón que aumentan a cantidade das mesmas que pode colocar cada indutor, o varios si que se encargan de evacuar por si mesmos os destinos, á marxe dos *packers*. Agora ben, sempre se deberá asumir que un varios ten menor capacidade de evacuación ca un *packer*, pois estes últimos traballan nunha zona específica do *sorter* e polo tanto a distancia entre os supostos destinos saturados é de agardar que sexa considerablemente menor.

A conclusión extraída desta sección é que a lóxica do modelo consistirá en traballar co número

total de prendas que sofren cada un dos procesos do clasificador en cada unha das súas revolucións. A cantidade de roupa que nun determinado ciclo pode ser inducida, pode caer en destino ou mesmo ser evacuada; virá dada principalmente polos parámetros que se acaban de expoñer, así como polas variables que indicarán o número de traballadores que ocupan o rol involucrado nese proceso, e por suposto polo acontecido nas revolucións previas.

3.2. Parámetros

Nesta sección presentamos os parámetros que contempla o modelo, así como os conxuntos nos que tanto estes como as variables se van indexar. Para maior comprensión, exporanse os parámetros en texto, e finalmente mostraranse nunha táboa a modo de resumo, doado de consultar se for necesario nos vindeiros apartados deste traballo.

Nun primeiro lugar preséntanse os parámetros que indican o tamaño dos conxuntos empregados. O número total de traballadores que o modelo debe asignar denotarase por W. Dun mesmo modo, e como a demanda está fixada como condición inicial, debe contarse cun parámetro TSA indicativo do número de tendas que demandan produtos, así como dun valor REF para o número total de referencias ou SKUs do reparto. No referido ás características do clasificador, coñecidas de antemán, o modelo precisa saber o número D de destinos ou xutes totais do *sorter*, así como o de postos de indución, $IPOST$, ou bandexas, S.

Finalmente, para poder definir variables ao longo dos ciclos do clasificador, o parámetro REV indica o número máximo de revolucións que o modelo contempla. A inclusión deste parámetro ten varias implicacións nos usos que se lle pode dar ao modelo. A primeira trata sobre a interpretación de resultados, dado que se un problema non ten solución, a conclusión a extraer é que para ningunha asignación posible de traballadores se logra completar o reparto dentro do límite temporal establecido. Como o feito de tomar un valor moi elevado para REV aumentaría enormemente o número de variables do modelo, este debe ser garantista mais non esaxerado. Neste sentido é posible facer un pequeno tanteo previo, á baixa, tomando algún valor axustado de REV e comprobando se non hai solución e debe elevarse esta cifra. Como non, a experiencia acumulada no uso reiterado do modelo debería volver máis sinxela a tarefa de seleccionar un valor acaído ao tamaño do problema.

Os parámetros que se presentaron nestes dous parágrafos anteriores inducen cadanseu conxunto. Se asumimos que todos os conxuntos están formados por enteiros positivos (agás o conxunto de revolucións, que inclúe ao 0), entón o conxunto dos traballadores do modelo sería $\{1, ..., W\}$, mentres que o conxunto de TSAs sería $\{1, ..., TSA\}$. Para non introducir unha cantidade excesiva de símbolos, cometerase frecuentemente o abuso de notación de nos referir ao conxunto de traballadores como W ou ao de TSAs como TSA, compartindo notación o conxunto co seu cardinal. Isto aplica a todos os parámetros que se acaban de mostrar nos parágrafos anteriores. Como aclaración, tomar os conxuntos deste modo non supón perda de xeralidade, pois á hora de traducir o noso modelo ao ordenador, será posible, por exemplo, introducir as diferentes TSA mediante nomes propios ou calquera outra nomenclatura, e o mesmo acontece cos demais parámetros. Porén, cando menos neste capítulo, conceptualizar así os conxuntos permite ter unha mellor idea sobre os potenciais tamaños de cada un deles.

O seguinte aspecto do problema que debemos traducir en parámetros é a demanda. Como xa se estableceu a notación para os conxuntos TSA de tendas e REF de SKUs, podemos tomar a demanda como un conxunto $DEMAND$, onde cada elemento $DEMAND_{ti}$ denota o número de prendas da referencia $i \in REF$ que require a tenda $t \in TSA$. Unha das suposicións realizadas sobre o modelo é o previo coñecemento de que destinos son asignados a cada TSA. Se D é o conxunto de destinos[3], entón podemos considerar a familia $TSADEST$ de conxuntos, onde $TSADEST_t \subset D$ representa os xutes que son asignados á tenda $t \in TSA$. Coa definición anterior pode dicirse que $TSADEST$ é unha partición de D [4]. Se ben na práctica pode darse o caso de que non se empreguen todos os destinos dun

[3]Lembramos que se abusa de notación e D denota ao conxunto $\{1, ..., D\}$.

[4]Neste traballo, diremos que unha familia A de conxuntos é unha partición dun conxunto X se $Y \in A \implies Y \subset X$, $\cup_{Y \in A} Y = X$ e os conxuntos que forman A son disxuntos dous a dous.

clasificador (por exemplo, cando a demanda for menor), abondaría con considerar como conxunto D só aos xutes que se empreguen, e polo tanto se asignen a algunha TSA.

Agora ben, para modelizar a caída de prendas nos xutes deben de se establecer as capacidades das caixas en destino, así como os volumes que ocupan as pezas de roupa de cada SKU. O primeiro punto só require un parámetro, pois suponse que todas as caixas son de idéntica capacidade, que será de B unidades de volume[5]. Para as pezas de roupa impleméntase o conxunto VOL, indexado en REF, de modo que para cada SKU $i \in REF$ diremos que as prendas de dita referencia ocupan VOL_i unidades de volume. Este valor será o utilizado para computar como se vai enchendo cada caixa baixo o destino $d \in D$, tendo en conta as consideracións vistas no apartado anterior e ilustradas na Figura 3.2.

Co explicado no parágrafo anterior pode deducirse que o modelo contabilizará, mediante o uso de certas variables, a capacidade dispoñible dunha caixa para acoller máis roupa, de modo que eventualmente estará chea e deberá evacuarse. Agora ben, só cos parámetros anteriores, podería darse o caso en que unha caixa tivese unha capacidade dispoñible positiva, pero tan reducida que ningunha prenda collese dentro dela, provocando que o dito destino quedase inutilizado (dado que nin podería acoller máis prendas nin se evacuaría por non estar a caixa chea). Para solventar isto, introduciremos dous parámetros que servirán como marxes de erro inferior ($LEMARGIN$) e superior ($UEMARGIN$). O uso da marxe $LEMARGIN$ de erro inferior resúmese como segue: asumirase que unha caixa está chea cando as pezas de roupa acaden un volume de $B - LEMARGIN$, para o que se asume que ou non collen máis prendas dentro, ou ben só algunhas moi específicas, de modo que convén evacuala. A inclusión da marxe de erro superior $UEMARGIN$ débese a que na realidade é frecuente que as caixas evacuadas desborden un pouco en canto á roupa, de modo que non se require ser estritos na limitación de volume que supón B. Polo tanto, imos supoñer que a última prenda que caia nunha caixa pode facelo mentres o volume total non supere as $B + UEMARGIN$ unidades de volume. Obviamente, ambas as dúas marxes poderían suprimirse sen máis que igualar cadanseu parámetro a cero. Para maior claridade, a Figura 3.3 contén un debuxo explicativo do uso destas marxes.

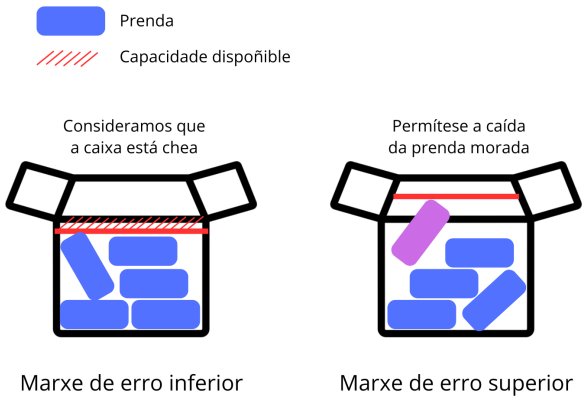


Figura 3.3: Esquema explicativo do significado dos parámetro $LEMARGIN$ e $UEMARGIN$. Elaboración propia.

Finalmente, os parámetros que resta por introducir son os indicativos das capacidades de traballo dos indutores, *waterspiders*, *packers* e varios. Como se mencionou, o modelo contempla, para cada revolución, só o número total de prendas que se inducen de cada SKU. A cantidade máxima de paquetes a colocar por parte de cada indutor vén determinada polo parámetro IAB. Por exemplo, se unha

[5] As unidades non son relevantes sempre que sexan as mesmas cas empregadas para computar os volumes dos paquetes.

solución supón que haxa 2 indutores, entón en cada revolución (se non temos en conta aos *waterspiders*) poden inducirse, como máximo, $2 \cdot IAB$ prendas, con independencia do seu SKU. Respecto dos *waterspiders*, como o seu labor se traduce en que o ritmo de indución dos traballadores aumente, a simplificación empregada consiste en fixar un valor por cada *waterspider*, que se suma ao máximo de prendas que se pode inducir. Se denotamos este valor por WAB, e por exemplo nunha solución figuran catro indutores e dous *waterspiders*, entón como máximo para cada revolución poderían inducirse $4 \cdot IAB + 2 \cdot WAB$ pezas de roupa.

Respecto dos *packers* e os varios, a implementación será análoga, con matices. O seu traballo consiste en evacuar as caixas cheas dos seus destinos e reemplazalas por outras novas. Polo tanto, no caso dos *packers*, asumimos que un parámetro PAB marca cantos destinos lle dá tempo a evacuar a cada un deles durante unha revolución dada. Xa se comentou que na realidade isto depende do próximos ou afastados que se atopen ditos destinos, polo que este valor debe entenderse como un promedio. Dito isto, e se ben os *waterspiders* se adicaban a aumentar a capacidade dos indutores, neste caso os varios evacúan eles mesmos os destinos, especialmente onde os *packers* teñan unha carga de traballo maior nese momento. Denotarase por VAB ao máximo de destinos que un varios logra evacuar ao longo dunha revolución calquera. Como detalle, debe lembrarse que cada *packer* ten unha zona do *sorter* asignada, mentres que os varios se moven polo clasificador para poderen axudar alá onde son máis necesarios. En consecuencia, é previsible que $PAB > VAB$, dado que os destinos a atender polos *packers* están polo xeral máis próximos entre si.

Unha vez presentados todos os parámetros, agrupámolos todos nunha táboa resumo (Figura 3.4), para maior claridade e facilidade de consulta.

W	Total de traballadores en $\{1, ..., W\}$.
S	Total de bandexas do clasificador.
TSA	Total de tendas $t \in \{1, ..., TSA\}$.
$IPOST$	Nº de postos de indución.
D	Nº de destinos, agrupados en $\{1, ..., D\}$.
REF	Nº de referencias ou SKUs, agrupados en $\{1, ..., REF\}$.
REV	Límite de revolucións para o problema, agrupadas en $\{0, 1, ..., REV\}$.
$DEMAND$	Conxunto de demandas por TSA e referencia.
$TSADEST$	Familia dos conxuntos de destinos asociados a cada TSA.
VOL	Conxunto de volumes das prendas de cada referencia.
B	Unidades de volume de calquera caixa en destino.
$LEMARGIN$	Marxe de erro inferior para o volume da caixa en destino.
$UEMARGIN$	Marxe de erro superior para o volume da caixa en destino.
IAB	Máximo de prendas que pode colocar cada indutor por revolución.
WAB	Máximo aporte de cada *waterspider* ao total que se poden inducir por ciclo.
PAB	Máximo nº de caixas que un *packer* pode evacuar por revolución
VAB	Máximo nº de caixas que un varios pode evacuar por revolución.

Figura 3.4: Táboa resumo dos parámetros do modelo.

3.3. Variables

Nesta sección presentamos os diferentes tipos de variables que conforman o modelo. Algunhas serán variables illadas, como as indicativas do número de traballadores que ocupa cada rol (as máis relevantes por seren os valores que se desexaba obter ao concebir o modelo), mentres que outras serán conxuntos de variables, indexados nalgúns dos presentados no apartado anterior. Dun modo similar á exposición dos parámetros, primeiro introduciranse as diferentes variables mediante unha explicación en prosa para ofrecer un contexto adecuado, e ao remate reuniranse nunha táboa simplificada onde apreciar os seus nomes, os conxuntos onde se indexan e unha brevísima explicación da súa función no modelo.

As primeiras variables, cuxo valor é o de maior importancia en calquera solución factible do modelo, son as seguintes: o número de traballadores asignados ao rol de indutor, que chamaremos n^I; e de igual modo o total n^{WS} de *waterspiders*, o número n^P de *packers*, e finalmente a cantidade de varios, denotada por n^{VR}. Obviamente, a suma destas variables debe ser o total de traballadores da instancia do modelo, que se estableceu como W. Adicionalmente, unha das restricións que se imporá no modelo é que o número de *packers* deba ser par, isto é, que $2|n^P$. Para lograr isto incluirase a variable $hpack$, cuxo significado é unicamente impoñer que $2 \cdot hpack = n^P$. Todas as variables anteriores son números enteiros non negativos.

O modelo, para cada posible combinación de valores nas variables anteriores, debe identificar individualmente aos *packers* e aos varios, a fin de asignarlle un conxunto de destinos a cada *packer* e regular o proceso de evacuación. Para isto créanse tres tipos de variables binarias. Primeiramente, as $rpack_e$ indican se o traballador $e \in W$ é ou non un *packer*. De xeito análogo actúan as variables $rvar_e, e \in W$ para os varios. Finalmente, as variables $apack_{de}, (d, e) \in D \times W$ indicarán se o destino $d \in D$ está ou non asignado ao operario $e \in W$. Por suposto, as restricións que se establecen no vindeiro apartado reguralán que os destinos só se asignan a *packers*, ademais de roturas de simetría para as variables $rpack$ e $rvar$.

Respecto da indución, o primeiro valor que se debe poder calcular é o número de prendas que se inducen de cada SKU en en certa revolución, o cal denotaremos por $nind_{ir} \in \mathbb{N}$, onde $i \in REF, r \in REV$. Créanse ademais variables binarias $already_{ir} \in \{0, 1\}$, con $i \in REF, r \in REV$, que indican se na revolución $r \in REV$ xa se induciron todas as pezas de roupa da referencia $i \in REF$. Os propósitos deste tipo de variables son dous. Dunha banda, permiten controlar cando se inducen todas as prendas dun SKU, impedindo dende ese punto que se introduzan máis. Da outra banda, e en liña coa suposición de que as referencias se inducen en orde, poden empregarse estas variables para regular cando comezar a inducir roupa dun novo SKU, por se rematar a indución do anterior.

O proceso que implica unha maior diversidade de variables son as caídas en destino e as evacuacións, dado que involucran o cómputo das revolucións e os diferentes SKU, pero tamén a cada destino e traballador. O primeiro paso, unha vez as prendas circulan polo clasificador, é a súa caída nun destino. Coa fin de reflectilo no modelo, as variables enteiras $nfall_{dir}$ representan o número de prendas da referencia $i \in REF$ que caen no destino $d \in D$ durante a revolución $r \in REV$. Para coñecer o contido das caixas, as variables $itemsinbox_{dir}$ contabilizarán cantos paquetes da referencia $i \in REF$ se atopan dentro da caixa baixo o xute $d \in D$ na revolución $r \in REV$. Estas permitirán coñecer en que momentos pode ou non caer unha nova peza de roupa, así como o momento en que se enchen, e deberán voltar a valer cero tras a evacuación do destino. Para dar remate ás caídas nos xutes, inclúense variables binarias $isfull_{dr}$, indicadoras de se o destino $d \in D$ se atopa cheo na revolución $r \in REV$, a fin de saber se procede ou non evacualo.

Para o proceso de evacuación, cómpre determinar cales dos traballadores, sexan *packer* ou varios, están operando e en que destinos o fan, ademais de en que revolución se produce dita acción. As respectivas variables binarias $ifpack_{der}$ e $ifvar_{der}$ servirán como indicadoras de se o *packer* ou varios $e \in W$ evacúa o destino $d \in D$ durante a revolución $r \in REV$.

Unha vez extraída a caixa chea do clasificador, e a fin de contabilizar o cumprimento da demanda de cada SKU, as variables enteiras $npack_{dir}$ contabilizan o número de prendas da referencia i que saíron do clasificador na revolución $r \in REV$ a través do destino $d \in D$. Como pequena aclaración, é importante que se indexen estas variables tamén no conxunto de destinos, pois estes satisfán a

demanda de cadansúa tenda asociada, a través da cal se avalía o seu cumprimento. Ademais, debe coñecerse cando se cumpre a demanda dunha TSA, pois iso implica que os destinos que lle dan servizo remataron as súas tarefas, e polo tanto as súas variables asociadas deixarán de se actualizar. Neste sentido, as variables binarias $cpletetsa_{tir}$ indicarán se en tempo $r \in REV$ a demanda de produtos da referencia $i \in REF$ da tenda $t \in TSA$ está completamente satisfeita.

Unha vez incluídos estes indicadores do cumprimento da demanda, poden engadirse as variables binarias $final_r$, que indicarán se á fin da revolución $r \in REV$ toda a demanda quedou xa satisfeita. Claramente, o paso de 0 a 1 desta variable darase cando en certa revolución, todas as variables do tipo $cpletetsa_{tir}$ sexan positivas. Ademais, deste xeito pode construírse unha función obxectivo conformada só polas variables $final_r$, de modo que o seu valor nos indique en que revolución se cumpre a demanda do *sorter*.

Non obstante, coas variables que se leva introducido polo momento, segue quedando un oco que encher no modelo: non todas as prendas inducidas nunha revolución caen en destino ao longo da mesma, senón que poden recircular, permanecendo na cadea ao inicio da seguinte etapa. Para contabilizar estas pezas de roupa empregaranse as variables enteiras $leftov_{ir}$, que computan cantas prendas do SKU $i \in REF$, que estaban en circulación durante a revolución $r \in REV$, non puideron caer en destino e voltarán a aparecer na etapa $r + 1$.

Con isto queda descrita a totalidade das variables do modelo, e a fin de poder consultalas ou aprecialas de xeito máis esquemático, agrúpanse nunha táboa (Figura 3.5). Nela figuran cada tipo de variable, o seu dominio, en que conxuntos se indexan os seus subíndices, e finalmente unha brevísima descrición do seu significado no modelo.

3.4. Restricións e función obxectivo

Preséntase deseguido a parte máis importante do modelo: a función obxectivo e as restricións. O capítulo anterior permite deducir as dificultades deste problema, e que para a súa modelización son necesarias unha gran cantidade e variedade de restricións. Por este motivo, esta sección dividirase en varios subapartados onde se traducen á restricións as diferentes partes do problema.

3.4.1. Función obxectivo

Neste modelo considerarase que a mellor asignación de traballadores aos roles de traballo será aquela que permita satisfacer a demanda no menor tempo posible. Como as variables $final_r$ serán nulas ata que se satisfaga a demanda, unha posible función obxectivo consistiría en maximizar a suma destas variables, é dicir, máx $\sum_{r \in REV} final_r$. Agora ben, para que o valor óptimo da función obxectivo se corresponda coa revolución en que se cumpre a demanda, e tendo en conta que o máximo valor posible para a suma destas variables é o parámetro REV, empregarase a función obxectivo seguinte:

$$\text{mín } REV - \sum_{r \in REV} final_r. \tag{3.1}$$

3.4.2. Sobre os traballadores

A restrición máis inmediata para o número de traballadores asignados a cada rol é que a suma suma represente o total W de traballadores. Debe terse en conta tamén que, como un clasificador ten un número $IPOST$ de postos de indución, este valor limita superiormente a cantidade n^I de indutores. Polo tanto, as dúas primeiras restricións do modelo serán as seguintes.

$$\begin{aligned} n^I + n^{WS} + n^P + n^{VR} &= W \\ n^I &\leq IPOST \end{aligned}$$

Variable	Dominio [6]	Subíndice	Significado
n^I	$\mathbb{N}$	-	Nº de indutores.
n^{WS}	$\mathbb{N}$	-	Nº de *waterspiders*.
n^P	$\mathbb{N}$	-	Nº de *packers*.
n^{VR}	$\mathbb{N}$	-	Nº de varios.
$hpack$	$\mathbb{N}$	-	Equivale á metade de n^P.
$rpack_e$	$\{0,1\}$	$e \in W$	O traballador e exerce de *packer*.
$rvar_e$	$\{0,1\}$	$e \in W$	O traballador e exerce de varios.
$apack_{de}$	$\{0,1\}$	$d \in D, e \in W$	O destino d está asignado ao traballador e.
$nind_{ir}$	$\mathbb{N}$	$i \in REF, r \in REV$	Nº prendas do SKU i inducidas no ciclo r.
$already_{ir}$	$\{0,1\}$	$i \in REF, r \in REV$	Se en tempo r foi inducida a demanda do SKU i.
$nfall_{dir}$	$\mathbb{N}$	$d \in D, i \in REF, r \in REV$	Nº prendas do SKU i que caen en d en tempo r.
$itemsinbox_{dir}$	$\mathbb{N}$	$d \in D, i \in REF, r \in REV$	Prendas de i baixo o destino d en tempo r.
$isfull_{dr}$	$\{0,1\}$	$d \in D, r \in REV$	Se a caixa baixo d se atopa chea en tempo r.
$ifpack_{der}$	$\{0,1\}$	$d \in D, e \in W, r \in REV$	Se o *packer* e está evacuando d en tempo r.
$ifvar_{der}$	$\{0,1\}$	$d \in D, e \in W, r \in REV$	Se o varios e está evacuando d en tempo r.
$npack_{dir}$	$\mathbb{N}$	$d \in D, i \in REF, r \in REV$	Prendas de i evacuadas dende d en tempo r.
$cpletetsa_{tir}$	$\{0,1\}$	$t \in TSA, i \in REF, r \in REV$	Se en r a demanda de t de produtos i se cumpre.
$final_r$	$\{0,1\}$	$r \in REV$	Se o proceso finalizou en tempo r.
$leftov_{ir}$	$\mathbb{N}$	$i \in REF, r \in REV$	Prendas de i que non caeron en tempo r.

Figura 3.5: Táboa resumo das variables do modelo.

En relación co anterior, debe establecerse formalmente que a cantidade de *waterspiders* está limitada polo número de indutores, e igualmente ocorre con varios e *packers*, respectivamente. En síntese, as persoas que ocupan un rol de apoio dependen de cantas se asocian ao rol ao que apoian.

$$\begin{aligned} n^I - n^{WS} &> 0 \\ n^P - n^{VR} &> 0 \end{aligned}$$

As variables binarias $rpack_e$ e $rvar_e$ que identifican, respectivamente, a *packers* e varios dentro do conxunto W de traballadores, precisan cadansúas restricións que limiten os seus posibles valores (sen ter en conta polo momento as simetrías, que se tratarán máis adiante). Claramente, a suma de cada grupo de variables debe ser, por definición, o número de persoas que ocupa o respectivo rol de traballo.

$$\begin{aligned} \sum_{e \in W} rpack_e &= n^P \\ \sum_{e \in W} rvar_e &= n^{VR} \end{aligned}$$

Respecto das restricións anteriores, se se fixa un traballador $e \in W$, este non poderá ser á vez un *packer* e un varios, e polo tanto deben engadirse as restricións do tipo

$$rpack_e + rvar_e \leq \quad 1, \quad \forall e \in W$$

Ao describir as variables que emprega o modelo para a fase de evacuación, viuse que as variables $ifpack_{der}$ e $ifvar_{der}$ fan referencia a cantos destinos logra evacuar nun ciclo, respectivamente, un *packer* ou un varios. Dado un $e \in W$, estas variables non poderán ser positivas agás que $rpack_e$ ou $rvar_e$ sexan positivas, en cadanseu caso. Poderíamos polo tanto establecer, por exemplo, que $ifpack_{der} \leq rpack_e, \forall d \in D, e \in W, r \in REV$. Agora ben, como tamén se teñen os parámetros PAB e $VRAB$ que limitan a cantidade de destinos onde $ifpack_{der}$ ou $ifvar_{der}$ poderían valer 1, unimos ambas as dúas ideas nos seguintes pares de restricións.

$$\begin{aligned} \sum_{dinD} ifpack_{der} \leq &\quad PAB \cdot rpack_e, \\ \sum_{d \in D} ifvar_{der} \leq &\quad VRAB \cdot rvar_e, \quad \forall e \in W, r \in REV \end{aligned}$$

Finalmente, para rematar coas restricións que afectan máis directamente aos traballadores, debe traducirse en restricións a asignación dos destinos a cada *packer* e como iso inflúe nas variables $ifpack_{der}$. Como as $apack_{de}$ indican que destinos se asignan a cada traballador, debe establecerse primeiro que cada xute só pode ser asignado a unha persoa. Ademais, deberase forzar que cada *packer* só teña permitido evacuar aqueles destinos que lle foron asignados (unhas limitacións que, como se viu anteriormente, non teñen os varios). Así, engádense as seguintes restricións:

$$\begin{aligned} \sum_{e \in W} apack_{de} \leq &\quad 1, \quad \forall d \in D \\ ifpack_{der} \leq &\quad apack_{de}, \quad \forall d \in D, e \in W, r \in REV. \end{aligned} \tag{3.2}$$

3.4.3. Sobre a indución

Como xa se comentou a inicios deste capítulo, ao describir as imposicións que se aplicarían ao modelo, a indución é un proceso que se ve moi simplificado no modelo. Isto trasládase ás restricións, de modo que se basean, en esencia, en poñer límites ao número de prendas inducidas en cada revolución, e controlan en que momento se induce a demanda total de produtos dun SKU, co cal debe pasarse a inducir prendas doutra referencia.

Os motivos para poñer cota superior ás variables $nind_{ir}$ son, en síntese, tres. Primeiramente, dada unha referencia i e unha revolución r, non se poderán inducir máis prendas dese SKU se xa entraron todos os paquetes necesarios para satisfacer a demanda. Se se acumulan as variables $nind_{is}$ con $s \leq r$, entón a súa suma non pode superar á demanda acumulada dese SKU ao longo das diferentes tendas:

$$nind_{ir} + \sum_{s=0}^{r-1} nind_{is} \leq \sum_{t \in TSA} DEMAND_{ti}, \quad \forall i \in REF, r \in REV$$

O segundo elemento que limita a cantidade de prendas inducidas será o número de indutores e de *waterspiders*, a través dos respectivos parámetros IAB e $WSAB$.

$$\sum_{i \in REF} nind_{ir} \leq IAB \cdot n^I + WSAB \cdot n^{WS} \quad \forall r \in REV$$

Finalmente, existe unha posibilidade máis que se debe contemplar no modelo, aínda que sexa altamente improbable agás en instancias moi concretas e pouco realistas: existe a posibilidade de que non queden bandexas baleiras onde colocar prendas. Polo tanto establecerase que a suma dos paquetes inducidos

nunha revolución non poida superar o número de bandexas menos aquelas que volven cheas de roupa que recircula.

$$\sum_{i \in REF} nind_{ir} \leq \quad S - \sum_{i \in REF} leftov_{i,r-1}, \qquad \forall r \in REV, r \geq 1$$

As demais restricións sobre a indución tratan de controlar cando se deixan de colocar prendas dun SKU (por cumprirse a demanda na entrada de roupa) para pasar a inducir elementos do seguinte. O primerio de todo será establecer a actualización (paso de 0 a 1) das variables $already_{ir}$ cando entre a demanda no clasificador.

$$\left(\sum_{t \in TSA} DEMAND_{ti} \right) \cdot already_{ir} \leq \quad \sum_{s=0}^{r} nind_{is}, \qquad \forall i \in REF, r \in REV$$

Adicionalmente, e aínda que intuitivamente poida resultar redundante, estableceremos que, se $alreay_{ir} = 1$, entón $already_{iv} = 1, \forall v > r$:

$$already_{ir} \geq \quad already_{i,r-1}, \qquad \forall i \in REF, r \in REV, r \geq 1$$

Ao inicio do problema, as variables $nind_{1,r}$ non teñen ningunha limitación relacionada con $already_{1,r}$, mentres que si forzará ás demais $nind_{ir}$ a valer 0. No punto en que $already_{1r} = 1$, entón acontecerá o mesmo para $i = 2$, e así sucesivamente. Obtéñense así as seguintes restricións, que impiden que se induzan prendas da referencia $i \geq 2$ se antes non se introdiron todas as do SKU $i - 1$:

$$nind_{ir} \leq \quad \left(\sum_{t \in TSA} DEMAND_{ti} \right) \cdot already_{i-1,r}, \qquad \forall i \in REF, i \geq 2, r \in REV$$

3.4.4. Sobre as caídas en destino

As restricións que a continuación se presentan consisten nas diferentes formas en que se limita a cantidade de prendas de cada SKU que poden caer en segundo que destinos ao longo de cada revolución. Como se pode deducir das explicacións dadas polo de agora, algúns condicionantes son a capacidade das caixas, a demanda restante e, como non, o propio o número de prendas que hai en circulación en cada momento.

Primeiramente, e no mellor dos casos, o número total de prendas que poden caer en destino nun ciclo fixado será o total de prendas en circulación, que se calcula sumando as que foron inducidas na mesma revolución máis as que recirculan dende a anterior.

$$\sum_{d \in D} nfall_{dir} \leq \quad nind_{ir} + leftov_{i,r-1}, \qquad \forall i \in REF, r \in REV, r \geq 1$$

Se se toma un destino $d \in D$ específico, entón outro limitante que xorde, antes incluso de considerar a capacidade restante da caixa, é a demanda da tenda á que serve. Nunha revolución $r \in REV$, se se consideran as prendas caídas ata o momento nos destinos asociados a unha tenda $t \in TSA$, entón en ditos xutes só poden caer como máximo as prendas que restan para cumprir a súa demanda.

$$\sum_{d \in TSADEST} \left(nfall_{dir} + \sum_{s=0}^{r} nfall_{dis} \right) \leq DEMAND_{ti}, \qquad \forall t \in TSA, i \in REF, r \in REV$$

Agora ben, ata que esta demanda non se satisfaga, o limitante principal da caída de roupa nun xute concreto será a capacidade restante da caixa que se atope nel. Deben terse en conta dous factores: se a caixa esta chea ou non, e o volume que resta por encher na caixa do destino dado. O primeiro punto é sinxelo: se un xute ten a caixa chea entón a cota superior será 0; se non a ten, a cota superior será o volume máximo da caixa.

$$\sum_{i \in REF} VOL_i \cdot nfall_{dir} \leq (B + UEMARGIN) \cdot [1 - isfull_{d,r-1}], \qquad \forall d \in D, r \in REV, r \geq 1$$

A capacidade restante da caixa obtense como a diferenza entre o total ($B + UEMARGIN$) menos o volume xa ocupado polas prendas, que depende das variables $itemsinbox_{dir}$. Así, as prendas que caen en $d \in D$ en tempo $r \in REV$, máis as que alí se atopaban en $r-1$, non poderán exceder o total.

$$\sum_{i \in REF} (nfall_{dir} + itemsinbox_{di,r-1}) \leq (B + UEMARGIN), \qquad \forall d \in D, r \in REV, r \geq 1$$

Cando as prendas dunha determinada referencia caen en certo destino, debe actualizarse o contador de produtos dentro da caixa, o cal se resume na seguinte restrición:

$$itemsinbox_{dir} \leq itemsinbox_{di,r-1} + nfall_{dir}, \qquad \forall d \in D, i \in REF, r \in REV, r \geq 1.$$

O símbolo '$\leq$' na restrición anterior non é casual, dado que en caso de ter empregado '=' non sería posible actualizar o valor das variables a cero cando se produza a evacuación do destino, o cal si é posible con esta formulación.

Finalmente, debe computarse en cada restrición a cantidade de prendas de cada SKU que non caen en destino e polo tanto recircularán polo clasificador ao longo do seguinte ciclo. Claramente, esta cifra sae de tomar a suma de produtos inducidos máis aqueles que recirculaban, e restarlle as pezas de roupa que caeron nun xute ao longo da presente revolución.

$$leftov_{ir} = leftov_{i,r-1} + nind_{ir} - \sum_{d \in D} nfall_{dir}, \qquad \forall i \in REF, r \in REV, r \geq 1.$$

3.4.5. Sobre as evacuacións

O primeiro punto que debe terse en consideración para evacuar un destino $d \in D$ na revolución $r \in REV$ é que este se sature, é dicir, que o indicador $isfull_{dr}$ pase a ser positivo. Tal como se afirmou ao expoñer os parámetros, consideraremos que unha caixa en destino está chea se o volume total que contén supera o valor $B - LEMARGIN$.

$$(B - LEMARGIN) \cdot isfull_{dr} \leq \sum_{i \in REF} VOL_i \cdot itemsinbox_{dir}, \qquad \forall d \in D, r \in REV$$

Pode apreciarse como $isfull_{dr}$ está forzada a ser nula mentres que a suma dos volumes das prendas que acolle a caixa non supera o umbral estipulado.

A continuación incluirase un grupo de restricións que cumpre dúas funcións á vez. Dunha banda, imporase que un *packer* ou varios só pode evacuar un destino se esté está xa saturado. Da outra banda, establece que un destino cheo só pode ser evacuado por unha persoa á vez.

$$\sum_{e \in W} (ifpack_{der} + ifvar_{der}) \leq isfull_{dr}, \qquad \forall d \in D, r \in REV$$

Do mesmo modo en que se estableceu a actualización de $isfull_{dr}$ cando se enche unha caixa, debe formularse a actualización a cero das variables $itemsinbox_{dir}$ cando xa se produciu a evacuación. Polo tanto imporase que, se na revolución $r-1$ se produciu a evacuación de $d \in D$, i.e., que $ifpack_{de,r-1} + ifvar_{de,r-1} = 1$ para algún $e \in W$; entón $\sum_{i \in REF} itemsinbox_{dir} = 0$. Esta lóxica queda plasmada no seguinte grupo de restricións:

$$\sum_{i \in REF} VOL_i \cdot itemsinbox_{dir} \leq (B + UEMARGIN) \cdot \left(1 - \sum_{e \in W} [ifpack_{de,r-1} + ifvar_{de,r-1}]\right)$$

$$\forall d \in D, r \in REV, r \geq 1$$

A formulación da restrición pode non ser todo o sinxela que cabería intuír da explicación que se acaba de dar, mais requírense de todos os elementos presentes nela para asegurar que, se nun destino

$d \in D$ non se produciu evacuación algunha, ou mesmo se a caixa nin sequera se enchiu aínda, se siga verificando que $\sum_{i \in REF} VOL_i \cdot itemsinbox_{dir} \leq (B + UEMARGIN)$, a fin de non interferir nas demais restricións que afectan ás variables $itemsinbox_{dir}$.

Neste bloque de restricións, resta introducir aquelas que dan significado ás variables $npack_{dir}$, que tratan de contabilizar o grao de cumplimento da demanda en destino. Son necesarias porque, se ben existen as variables $itemsinbox_{dir}$ que contabilizan as prendas nos xutes, estas variables voltarán a cero unha vez que completada a evacuación. Polo tanto, imporase que as variables $npack_{dir}$ recollan o testigo das $itemsinbox_{dir}$ xusto antes de que estas volvan a valer cero. Empregaranse para isto dous grupos de restricións.

$$npack_{dir} \leq itemsinbox_{dir}, \qquad \forall d \in D, i \in REF, r \in REV$$

$$npack_{dir} \leq \left(\sum_{t \in TSA} DEMAND_{ti} \right) \cdot \sum_{e \in W} (ifpack_{der} + ifvar_{der}), \quad \forall d \in D, i \in REF, r \in REV$$

A primeira das restricións é a máis sinxela, dado que só limita o valor ao que actualizar $npack_dir$ como $itemsinbox_{dir}$, o que nos interesa que tome chegado o seu momento. O signo '$\leq$' non é problemático, pois ao ser un modelo de optimización, esta pasará por que na práctica, os valores de $npack_{dir}$ serán ou ben 0 ou ben $itemsinbox_{dir}$. A segunda restrición significa que, agás que se evacúe o destino seleccionado, $npack_{dir}$ terá valor nulo. En caso contrario, establécese unha cota superior máis alta cá desexada, a fin de que xuntando ambas restricións a cota efectiva sexa $itemsinbox_{dir}$. O motivo de empregar dúas restricións diferentes é, claramente, evitar que en ningunha restrición se multipliquen dúas variables, pois suporía a creación doustras novas que representasen o seu produto, así como as súas respectivas restricións.

3.4.6. Sobre o cumprimento da demanda

En cada revolución, unha vez caída a roupa nas caixas, debe de se comprobar para cada tenda se a demanda desta queda satisfeita. Un modo de cuantificalo é, dada unha demanda de $i \in REF$ dunha tenda $t \in TSA$ e un tempo $r \in REV, r \geq 1$; tomar o acumulado das variables $npack_{dir}$ para os $d \in TSADEST_t$ e sumarlle as prendas que se atopan nas caixas nese momento, dadas pola suma da variables $itemsinbox_{dir}$. O resultado é a seguinte restrición.

$$DEMAND_{ti} \cdot cpletetsa_{tir} \leq \sum_{d \in TSADEST_t} \left(itemsinbox_{dir} + \sum_{s=0}^{r-1} nfall_{dis} \right),$$

$$\forall t \in TSA, i \in REF, r \in REV, r \geq 1$$

Con esta formulación, as variables $cpletetsa_{tir}$ só se poden actualizar a 1 cando o termo dereito da expresión sexa igual á demanda corresposdente.

Xa se comentou a pretensión de que, no caso en que a demanda fose imposible de se satisfacer no límite de tempo proposto, o problema debería ser infactible. Nesta liña, pode tomarse como restrición que, nalgunha das revolucións $r \in REV$, se deba cumprir a demanda total, é dicir:

$$\sum_{r \in REV} cpletetsa_{tir}, \qquad \forall t \in TSA, i \in REF.$$

Resta aínda darlle significado ás variables $final_r$, indicadores de que se cumpriu toda a demanda solicitada ao remate do ciclo $r \imath REV$. Para continuar na liña dos abusos de notación empregados neste capítulo, empregarase a redundancia de chamar $|REV|$ ao número total de revolucións, aínda que dito parámetro en orixe era quen recibía a notación REV. As restricións que regulan a actualización das variables $final_r$ son as seguintes:

$$|TSA| \cdot |REV| \cdot final_r \leq \sum_{t \in TSA} \sum_{i \in REF} cpletetsa_{tir}, \qquad r \in REV.$$

Finalmente engadimos, mesmo ao ser redundante, unha restrición que impón que algunha das variables $final_r$ se actualice a 1, de modo que se isto non sucede signifique que a instancia do problema é infactible.

$$\sum_{r \in REV} final_r \geq 1$$

3.4.7. Valores iniciais e finais das variables

Adícase esta breve sección a establecer os valores iniciais das variables, en particular aquelas dependentes do tempo, para impoñer que sexan nulas en $r = 0$. En efecto, o conxunto de revolucións comeza en 0 e non en 1 para dar coherencia a aquelas restricións que esixen a comparativa das variables nun momento $r \in REV$ respecto do intre $r - 1$, que crearían conflitos cando $r = 1$[7]. Polo tanto, e como resumo, afírmase que:

$$\begin{aligned}
nind_{i0} &= 0, & \forall i \in REF \\
already_{i0} &= 0, & \forall i \in REF \\
nfall_{di0} &= 0, & \forall d \in D, i \in REF \\
itemsinbox_{di0} &= 0, & \forall d \in D, i \in REF \\
isfull_{d0} &= 0, & \forall d \in D \\
ifpack_{de0} &= 0, & \forall d \in D, e \in W \\
ifvar_{de0} &= 0, & \forall d \in D, e \in E \\
npack_{di0} &= 0, & \forall d \in D, i \in REF \\
cpletetsa_{ti0} &= 0, & \forall t \in TSA, i \in REF \\
final_0 &= 0, & \\
leftov_{i0} &= 0, & \forall i \in REF.
\end{aligned}$$

Ademais, tamén se establecerán os valores finais dalgunhas variables, é dicir, cando $r = REV$, daquelas que dependen do tempo. Que as restricións deste tipo poidan resultar redundantes non supón un obstáculo na formulación do modelo, dado que os *solvers* eliminan as redundancias ao asimilar os problemas que se lles envía. Como aspecto positivo, poden contribuír na comprensión do problema por parte das persoas interesadas, así como cubrir algunha carencia non prevista no modelo que puidese levar a combinacións de valores non desexados para as variables.

O que se imporá neste apartado é que as variables como $nind_{ir}$, $nfall_{dir}$, $itemsinbox_{dir}$ e $npack_{dir}$, debidamente agrupadas, sumen ao longo das revolucións a demanda total do problema. Se o modelo funciona correctamente, o único modo de incumprir as seguintes restricións será a infactibilidade.

$$\begin{aligned}
\sum_{r \in REV} nind_{ir} &= \sum_{t \in TSA} DEMAND_{ti}, & \forall i \in REF \\
\sum_{d \in TSADEST_t} \sum_{r \in REV} nfall_{dir} &= \sum_{t \in TSA} DEMAND_{ti}, & \forall i \in REF, t \in TSA
\end{aligned}$$

$$\sum_{d \in TSADEST_t} \left(itemsinbox_{di,REV} + \sum_{s=0}^{REV-1} nind_{is} \right) = \sum_{t \in TSA} DEMAND_{ti}, \qquad \forall i \in REF, t \in TSA$$

3.4.8. Roturas de simetría

O modelo, tal e como está plantexado, presenta multitude de simetrías. Este será un dos puntos a criticar do mesmo, o cal se apreciará mellor no capítulo seguinte.

[7] Claramente, tamén se solventaría esta contrariedade se se eliminasen ou modificasen todas as restricións deste tipo. Non obstante, o máis sinxelo é incluír o intre temporal $r = 0$ e impoñer que as variables sexan sempre nulas nel.

Porén, nesta sección presentaranse algunhas restricións adicionais que permitirán eliminar simetrías alí onde é máis sinxelo: nas variables $rpack_e$, r_var_e e tamén $apack_{de}$. Neste caso a simetría implica que, se por exemplo $n^P = 3$, entón de todas as variables $rpack_e$ debería haber 3 con valor positivo, pero pode ser calquera terna no conxunto $\{1, ..., W\}$, o cal crea multitude de solucións posibles que son, en todos os demais aspectos, iguais (é dicir, a definición de simetría). O mesmo acontecería coas r_var_e e $apack_{de}$.

As anteriores son as simetrías máis doadas de eliminar, dado que abondará con impoñer unha orde no conxunto $\{1, .., W\}$ para asignar os seus memnros como *packers* ou varios, e do mesmo modo asignar unha orde nos destinos respecto da súa asignación a un *packer*. Para os traballadores, farase que $e = 1$ sexa sempre un *packer*, e do mesmo modo todos os traballadores dende 1 ata n^P. Dende o elemento $n^P + 1$ ao $n^P + n^{VR}$ serán todos varios. Isto pode conseguirse mediante as seguintes restricións:

$$\begin{aligned} rpack_1 = &\quad 1 \\ rpack_e - rpack_{e-1} \leq &\quad 0 \quad \forall e \in W, e \geq 2 \\ rvar_e - rvar_{e-1} - rpack_{e-1} \leq &\quad 0 \quad \forall e \in W, e \geq 2 \end{aligned}$$

Dentro do anterior conxunto de restricións, a primeira representa claramente que $e = 1$ será un *packer*. A segunda significa que se $e \geq 2$ ocupa o rol de *packer*, entón $e - 1$ tamén debe facelo. Finalmente, o derradeiro grupo de restricións impón que se $e \geq 2$ é un varios, entón $e - 1$ só pode ser outro varios ou un *packer* (o cal seria claramente o último da lista).

Unha vez establecida esta orde, podemos empregala como guía para as variables $apack_{de}$. Na realidade, cada traballador ten asignados un conxunto de destinos consecutivos. Non é relevante para o modelo que esta propiedade se inclúa, pero permite eliminar todas as simetrías asociadas a esta asignación. O método empregado será que o primeiro bloque de destinos sexa asignado ao traballador $e = 1$, que como se estableceu, xa é un *packer*. Dende aquí, cada novo bloque de destinos consecutivos será asignado a un novo traballador seguindo esta orde, dende $e = 1$ ata n^P. Para logralo, abonda con introducir o seguinte grupo de restricións:

$$apack_{\tilde{d},e} + apack_{d,\tilde{e}} \leq 1, \qquad \forall d, \tilde{d} \in D; \quad d < \tilde{d}; \quad \forall e, \tilde{e} \in W; \quad e < \tilde{e}.$$

As demais roturas de simetrías non son abordables sen aumentar de xeito significativo o número de variables e restricións do modelo. En síntese, dado que se tratará esta cuestión no seguinte capítulo, as simetrías restantes do problema proceden das caídas de prendas en caixas, e non se podería abordar esta problemática sen establecer no modelo o análogo a un algoritmo de Round-Robin para escoller o destino en que cae cada produto. Tratar de implementar iso convertiría o modelo en algo similar a un simulador, pero coa diferenza de que as instancias do problema serían totalmente inviables de se resolver por un *solver*.

3.4.9. Outras restricións

Pechando a descrición do modelo, quedan relegadas a esta sección dous tipos de restricións concretas, que non son exactamente derivadas do funcionamento do clasificador, senón de dúas directrices que o titor da empresa recomendou aplicar. Os motivos da súa inclusión son dous: dunha banda, a base experimental que ten a empresa para considerar que o mellor funcionamento do *sorter* pasa por seguir estas recomendacións; da outra banda, facer máis equitativa a carga de traballo entre os operarios, e non permitir solucións descompensadas neste sentido.

A primeira das directrices é o motivo polo cal se inclúe no modelo a variable *hpack*: que o número de *packers* debe ser par. Como se aprecia no esquema da Figura 2.1, o sorter ten dous lados, e polo tanto un número par de traballadores neste rol permite equilibrar o esforzo físico. Non se pode esixir que en todos os roles haxa un número par de empregados, e polo tanto só se impón neste caso, que é no que esta cuestión ten unha maior afectación. A restrición, claramente, será a seguinte:

$$n^P - 2 \cdot hpack = 0.$$

A outra directriz que debe incluírse no modelo é que os diferentes *packers* teñan asignados un número similar de destinos. Na realidade, os xutes clasifícanse en lado, grupo e sección, onde non todos os grupos teñen o mesmo número de seccións nin todas as seccións contan co mesmo número de xutes (polo tanto, esta cuestión tórnase máis complexa). Agora ben, neste modelo non se reflicte esta estruturación e polo tanto podemos establecer que, como moito, a diferenza entre os destinos asignados ao *packer* e e os asignados a $\tilde{e}$ será como moito de 1. Para lograr isto e contemplar o escenario en que e sexa un *packer* pero que $\tilde{e}$ non o sexa, servirémonos da orde establecida para as roturas de simetría. Así, esta formulación das restricións reflictirá exactamente a idea que se acaba de expoñer:

$$\sum_{d \in D} (apack_{d,\tilde{e}} - apack_{d,e}) \leq 1, \qquad \forall e, \tilde{e} \in W; \quad e < \tilde{e}.$$

Deste xeito, se $\tilde{e}$ é un *packer*, entón tamén o debe ser e, e polo tanto respéctase a diferenza. Se non é o caso, entón e pode ou ben ser un *packer*, co cal a diferenza é negativa e non supón un inconveniente; ou ben pode non selo, co cal a diferenza é cero.

3.5. Versións evolucionadas do modelo. Cambios dinámicos e indución mellorada

O modelo de optimización sobre o que versa este traballo xa quedou exposto. Con todo, existen algunhas expansións do modelo que se chegaron a idear durante as prácticas, pero que non se remataron de implementar. O motivo, como se comprobará no capítulo seguinte, é que o modelo no seu estado base era inviable no referido á resolución de problemas realistas mediante un *solver* comercial. Non obstante, e dado que se redactaron documentos de traballo ao respecto, recollidos no Apéndice B, poden ser un bo complemento para a comprensión do problema real, á vez que unha forma de mostrar como modelizar outros aspectos do clasificador ademais dos xa vistos.

3.5.1. Cambios na asignación traballador-rol

A primeira vía de expansión do modelo que se estudou foi unha solicitude do titor na empresa. Das explicacións dadas sobre o funcionamento do clasificador dedúcese que, nas primeiras revolucións, os indutores e *waterspiders* son os que soportan a maior carga de traballo, mentres que ao haber poucas prendas en circulación, os *packers* e varios case non teñen tarefas a realizar. Segundo vai quedando menos roupa que inducir no *sorter*, a situación cambia ata ser a contraria. Isto ocasiona que, na realidade, o número de traballadores que ocupan cada rol vaia variando ao longo do reparto. O que se pretende é introducir este carácter dinámico da asignación dos roles de traballo dentro do modelo.

En consecuencia, redactouse o documento de traballo que se pode consultar no Apéndice B como *Modelo Multirreferencia V1.5*, onde se realiza unha actualización completa do modelo incluíndo esta idea, cos cambios que se explicarán resumidamente a continuación, se ben non se darán todos os detalles por non ser esta a versión do problema que se analizará no vindeiro capítulo.

Para comezar, non ten moito sentido permitir que a asignación roles-traballadores poida cambiar en cada revolución, tanto porque non é realista realizar estas modificacións cada poucos minutos, como polo aumento no número de variables, dado que varias das existentes pasarían a indexarse en REV. No canto de facer isto, decidiuse definir dous tipos de conxuntos. Nun primeiro lugar, un novo conxunto $REVCUT \subset \{1, 2, ..., REV - 1\}$ de puntos temporales en que se pode cambiar a asignación dos roles de traballo. Estes límites definirán etapas, nas que se enmarcarán as diferentes revolucións. Para cada $r \in REV$, o parámetro $STRETCH_r \in \{1, ..., |REVCUT| + 1\}$ indica a etapa á que pertence r. Finalmente, e para a correcta formulación de restricións, denótase por $STRETCH^*$ o conxunto de elementos diferentes de $STRETCH$, isto é, sen repeticións. Por exemplo, se o primeiro punto de corte é $REVCUT_1 = 3$, teríase que $STRETCH_0 = STRETCH_1 = STRETCH_2 = 1$ e que $STRETCH_3 = 2$.

Aplicando estes novos parámetros, as seguintes variables deben indexarse tamén no conxunto $STRETCH^*$:

Variable anterior	Variable actual	Significado
n^I	n_s^I	Nº indutores na etapa $s \in STRETCH^*$.
n^{WS}	n_s^{WS}	Nº *waterspiders* na etapa $s \in STRETCH^*$.
n^P	n_s^P	Nº *packers* na etapa $s \in STRETCH^*$.
n^{VR}	n_s^{VR}	Nº varios na etapa $s \in STRETCH^*$.
$hpack$	$hpack_s$	$\frac{1}{2} \cdot n_s^P$ para cada etapa.
$rpack_e$	$rpack_{es}$	Se $e \in W$ é un *packer* durante a etapa $s \in STRETCH^*$.
$rvar_e$	$rvar_{es}$	Se $e \in W$ é un varios durante a etapa $s \in STRETCH^*$.
$apack_{de}$	$apack_{des}$	Se $d \in D$ está asignado a $e \in W$ na etapa $s \in STRETCH^*$.

O modo en que se adaptan as restricións xa explicadas a esta nova formulación é moi simple. Se nunha restrición aparecen variables indexadas en $STRETCH^*$ preséntanse dúas opcións. Primeiramente, se non aparecen outras variables indexadas en REV, a restrición mantense igual, co único cambio que supón a nova formulación da variable. Por exemplo, o primeiro grupo de restricións do bloque 3.2, aquelas que obrigan a que cada destino só se pode asignar a un *packer*, quedarían como segue:

$$\sum_{e \in W} apack_{des} \leq \quad 1, \quad \forall d \in D, s \in STRETCH^*.$$

Se, polo contrario, existen variables nesa mesma restrición que se indexan en $r \in REV$, deben indexarse as novas variables non en $s \in STRETCH^*$, senón en $STRETCH_r$. Un exemplo é o segundo grupo de restricións do mesmo bloque 3.2, que fai que ningún *packer* poida evacuar un destino se non está asignado a el, escribiríase así:

$$ifpack_{der} \leq \quad apack_{de,STRETCH_r}, \quad \forall d \in D, e \in W, r \in REV.$$

3.5.2. Modelización máis realista da inducción

Sobre o proceso de indución, o documento *Propuestas para modelizar la Inducción: Modelo Multirreferencia V1.5* recollido no Apéndice B propón dúas posibles vías de modelización da indución, pero sen chegar a reformular o problema completamente. Respecto dos *waterspiders*, bosquéxanse algunhas ideas sobre como se poden modelizar, dependendo do procedemento que se siga para os indutores. O motivo de que estes documentos non culminaran nunha versión completa do modelo é a ineficacia do mesmo nas súas primeiras versións, computacionalmente falando.

En todo caso, ambas as dúas vías para unha inclusión máis realista da indución perseguen dous obxectivos. O primeiro é reflectir no modelo as desigualdades na carga de traballo que asume cada indutor dentro do clasificador. No modelo que se presenta neste traballo, asúmese como simplificación da realidade que cada indutor ten unha capacidade fixa de prendas que é capaz de colocar nas bandexas ao longo dunha revolución, e que esa capacidade é a mesma para todos os traballadores. Na realidade non acontece así. A Figura 2.2 do pasado capítulo mostra esquemáticamente o motivo de que a capacidade indutora sexa desigual no conxunto de operarios. No sentido da marcha do *sorter*, que na

imaxe é de esquerda a dereita, o primeiro indutor verá baleiras todas as bandexas (agás aquelas que recirculen), e polo tanto meterá tanta roupa no clasificador como sexa fisicamente capaz. Non obstante, todas as prendas que induza implicarán que as súas respectivas bandexas cheguen cheas fronte aos seus compañeiros, de modo que, a medida que aparecen novos indutores no sentido da marcha, cada vez inducen menos roupa por non ter á súa disposición suficientes bandexas baleiras.

A proposta realizada para mostrar este comportamento consistía na inclusión dun conxunto $RATIO$ de parámetros, tantos como $IPOST$, de modo que cada $RATIO_p$ se corresponde coa proporción de bandexas (sobre o total S) que pode colocar o indutor p no sentido da marcha. A proporción realízase sobre S e non respecto de cada bandexa baleira que se lle presenta para simplificar os cálculos, pois en caso contrario implicaría un maior número de variables. Como exemplo, se se supón que $IPOST = 4$ e que $RATIO = \{0{,}35, 0{,}29, 0{,}23, 0{,}10\}$, entón de $S = 200$ bandexas teríase que o indutor 1 introduce $S \cdot RATIO_1 = 200$ pezas de roupa. Cos mesmos cálculos os seguintes tres indutores terían capacidade para inducir 58, 46 e 20 prendas, respectivamente. Axustando os parámetros $RATIO_p$ en base a datos obtidos de xornadas reais de traballo, quedaría ben reflectida no modelo esta desigualdade no esforzo dos indutores.

O segundo aspecto que se desexaba mostrar no modelo é o progresivo decrecemento na capacidade indutora dos traballadores a causa do cansazo. Para introducila no modelo, presentáronse dúas vías, a primeira consistente en actualizar estas capacidades en cada revolución, e a segunda en actualizalas seguindo un sistema de etapas, como o visto no apartado anterior con $STRETCH$. En ambos os dous casos, contemplábase a posibilidade de que, ben por descansos ou ben por cambios de tarefas, esta fatiga se reiniciase en certos puntos da xornada, por exemplo coincidindo cos cambios de etapa asociados a $REVCUT$ e $STRETCH$.

No enfoque de actualización por cada revolución, substituiríase o parámetro único IAB pola colección de variables $indab_{pr}$, onde $p \in \{1, ..., IPOST\}$ e $r \in REV$. Introduciríanse os parámetros $FATIGUE_p$ que, para cada indutor p, supoñen un ratio en $(0, 1)$ da diminución da cantidade de prendas que poden inducir por revolución. Non se utiliza un único parámetro pois cabe esperar que, ao ser desigual a carga de traballo dos indutores, tamén o sexa o ritmo ao que se cansan. Con esta notación, deixando definidos $RATIO$ e $FATIGUE$, os valores das variables $indab_{pr}$ obteríanse como segue:

$$indab_{p1} = S \cdot RATIO_p \cdot rind_p, \qquad \forall p \in \{1, ..., IPOST\} \tag{3.3}$$

$$indab_{pr} = FATIGUE_p \cdot indab_{p,r-1}, \qquad \forall p \in \{1, ..., IPOST\}, r \in REV, r \geq 2 \tag{3.4}$$

Pode apreciarse que, na restrición dada en (3.3), aparece unha nova variable $rind_p$, a cal será binaria e indicará se no posto de indución p hai un indutor. Asumirase tamén que se hai s indutores, entón ocuparán os postos de indución dende 1 ata s, por orde. A inclusión deste tipo de variable, similar ás xa vistas $rpack_e$ ou $rvar_e$ con $e \in W$, é necesaria á hora de poder calcular as $nind_{ir}$, que neste caso quedarían como se ve a continuación:

$$\sum_{i \in REF} nind_{ir} \leq \sum_{p=1}^{IPOST} indab_{pr}, \qquad \forall r \in REV, r \geq 1. \tag{3.5}$$

Se non se tivesen incluído as variables $rind_p$ en (3.3), entón a expresión anterior estaría sumando capacidades de postos de indución baleiros.

Adicionalmente, se desexamos que no cambio de etapa dado por $REVCUT$ se reinicien as capacidades indutoras sen ter en conta a fatiga, abonda con substituír as restricións (3.4) que correspondan polas seguintes:

$$indab_{pr} = S \cdot RATIO_p, \qquad \forall p = 1, ..., IPOST; \quad r \in REVCUT. \tag{3.6}$$

A segunda vía proposta, a actualización por etapas, e moi similar aos cambios de asignación por etapas vistos con $STRETCH$ e $STRETCH^*$. Neste caso crearíanse etapas que denotaríamos de igual modo como $IJUMP$ e $IJUMP^*$, de modo que só ao saltar de etapa será cando se aplique o parámetro

$FATIGUE$. En síntese, as restricións (3.3), (3.4) e (3.5) que se acaban de ver pasarían a formularse, respectivamente, como segue:

$$\begin{aligned} indab_{p1} = & \quad S \cdot RATIO_p, & \forall p \in \{1, ..., IPOST\} \\ indab_{ps} = & \quad FATIGUE_p \cdot indab_{p,s-1}, & \forall p \in \{1, ..., IPOST\}, s \in IJUMP^* \\ \sum_{i \in REF} nind_{ir} \leq & \quad \sum_{p=1}^{IPOST} indab_{p,IJUMP_r}, & \forall r \in REV, r \geq 1. \end{aligned}$$

Como se pode ver, as variables $indab_{ps}$ só se definen nas diferentes etapas temporais, e non en todas as revolucións. Na última restrición apréciase máis claramente como a capacidade indutora se mantén constante durante as revolucións que ocupa unha mesma etapa, para logo reducirse no salto de etapa por acción do parámetro de fatiga. Esta formulación non é incompatible con reiniciar a capacidade indutora en certos momentos do reparto, do mesmo moodo que se fixo na restrición (3.6).

Finalmente, non se entrou a presentar notación nin formulación para modelizar a actuación dos *waterspiders*. Como se di no mencionado documento de traballo[8], trátase dun rol moi secundario no modelo e que ocupa na práctica a poucas persoas, polo que en caso de modelizalos de xeito máis fino, o aumento de variables pode ser contraproducente. Agora ben, en caso de se querer modelizar de modo máis acaído á realidade estas tarefas, deberían cando menos crearse variables $rws_e, e \in W$, análogas ás $rpack_e$; así como $aws_{per}, p \in \{1, ..., IPOST\}, e \in W, r \in REV$ que, como acontecía coas $apack_{de}$, indiquen de modo binario se un determinado *waterspider* se atopa nun certo posto de indución nun momento dado. Unha vez sentadas estas bases, algunhas ideas que se propoñían eran que os *waterspiders* tivesen un aporte á capacidade indutora decrecente mentres máis tempo pasaban nun mesmo posto de indución ou mesmo que a súa presenza aliviase en certo modo a fatiga sufrida polo indutor.

[8] *Propuestas para modelizar la Inducción: Modelo Multirreferencia V1.5*, recollido no Apéndice B.

Capítulo 4

Análise crítica do modelo e a súa aplicabilidade

No Capítulo 2 deste texto, ao describirse o problema que se debía modelizar e resolver, e explicar con certo detemento o funcionamento do clasificador, incidíase na complexidade dos seus procesos. En consecuencia, xurdía a dificultade de traducir o problema a un modelo de optimización clásico, neste caso un ILP[1], de modo que fose viable a execución das súas instancias por parte dun *solver* comercial. Tamén no Capítulo 3 se apuntou, en diversos apartados, a inconvenientes que a formulación ía permitindo deducir nese momento, principalmente no referido ao número de variables e á abundancia de simetrías que non se podían romper facilmente [2].

Se o capítulo anterior estivo adicado a formular completamente o modelo de optimización, explicando os parámetros, variables e restricións empregadas; no presente capítulo analizarase e mesmo criticarase o modelo, explicando en detalle as problemáticas que o acompañan. Ao longo das seccións deste capítulo, irase vislumbrando a ineficacia do modelo como vía para resolver casos realistas, e apreciarase o interese de explorar outras vías.

Primeiramente, estudarase a capacidade do modelo proposto para escalar, é dicir, para manter un crecemento controlado do número de variables a medida que os parámetros medran en tamaño, no sentido de que sigan resolvendo as súas instancias, ata chegar a casos realistas. Verase entón a medida en que depende o número de variables do modelo de segundo que parámetros. Non debe esquecerse que os problemas de programación enteira ou ILP teñen un custo computacional de tipo expoñencial.

Mesmo no caso en que o modelo escalase correctamente, e fose asumible a nivel informático a resolución de instancias de gran tamaño do mesmo, cómpre reflexionar sobre a calidade das solucións que se poderían obter. O clasificador é un sistema moi complexo, máxime pola necesidade de modelizar o factor humano, que intervén en case todos os procesos. Aspectos como a indución están excesivamente simplificados, e cabe agardar que ningunha solución do modelo nos proporcione información adecuada sobre como realizan o seu traballo as diversas persoas implicadas. Deberá concluírse canta información pretende extraerse do modelo, cal pode ter valor para aplicar no *sorter* real.

Unha conclusión desta análise será que a única información de utilidade a extraer do modelo serán os valores das catro variables n^I, n^{WS}, n^P e n^{VR}. Este capítulo adicará unha breve sección a expoñer unha peculiaridade que presenta este problema do clasificador: que o número de posibles combinacións destas catro variables é, na praxis, moi baixo. Este feito dará pé a valorar as vantaxes que presentará a construción dun simulador. Este consistiría nun programa informático que, dada unha asignación de valores destas catro variables, recrease o *sorter* e simulase o reparto completo, devolvendo o tempo que

[1] *Integer Linear Programming problem*, é dicir, aquel onde as restricións e función obxectivo son lineais, pero todas as variables son discretas.

[2] Adicouse un apartado a aquelas que si se podían romper, e mencionouse que a meirande parte delas non eran rompibles sen aumentar enormemente o número de variables do modelo.

este tarda en completarse. Poderían entón ensaiarse todas as posibilidades, devolvendo por pantalla aquela que minimice o *makespan*, é dicir, o tempo de cumprimento da demanda.

Vistas as virtudes que podería presentar un simulador, especialmente en contraste co modelo de optimización matemática, a última sección mostra unha comparativa entre o uso do modelo e o dunha versión primitiva do simulador para resolver instancias de pequeno tamaño. No transcurso das prácticas propúxose a idea de que, antes de crear un simulador completo do *sorter*, se implementase unha versión adecuada ás simplificacións que presenta o modelo. Así, sería posible aplicar ambas as dúas ferramentas para resolver os mesmos problemas concretos e que, en certo modo, unha validase á outra. Neste último apartado describirase de modo superficial e gráfico o funcionamento informático do simulador e do estudo comparativo, e mostraranse os resultados que se obtiveron no seu momento.

4.1. Escalabilidade. Tamaño do problema e custo computacional

Para poder discutir se o modelo pode ou non escalar e resolver instancias de tamaño crecente, o primeiro será tratar de computar do número de variables do modelo. Estas expresaranse en función dos parámetros W, D, REF, REV e TSA, aqueles que inducen os conxuntos en que se indexan as variables. Non se busca un cálculo exacto, e polo tanto, ademais de omitir do cálculo os conxuntos unitarios de variables, nos casos en que se deban usar sumandos como $REV-1$ ou similares, simplemente sumaríase REV.

As variables n^I, n^P, n^{WS}, n^{VR} e *hpack* son as mesmas para todas as instancias, e polo tanto non se inclúen neste cómputo. Comezando por aqueles grupos de variables que presentan un único subíndice, en cada configuración de parámetros dada teranse W variables tanto $rpack_e$ como $rvar_e$. Ademais, haberá REV variables $final_r$.

Respecto das que presentan dous subíndices, as variables de asignación $apack_{de}$ son en total $D \cdot W$. Tamén as variables $isfull_{dr}$ se indexan seguindo os parámetros D e REV, ao que se suman as $REF \cdot REV$ variables da forma $nind_{ir}$, $already_{ir}$ e $leftov_{ir}$.

Falta sumar dous tipos de variables. Dunha banda, $nfall_{dir}, itemsinbox_{dir}$ e $npack_{dir}$ supoñen cadansúas $D \cdot REF \cdot REV$ variables. Súmanse tamén $D \cdot W \cdot REV$ incógnitas dos tipos $ifpack_{der}$ e $ifvar_{der}$. Finalmente, considéranse as $TSA \cdot REF \cdot REV$ variables do tipo $cpletetsa_{tir}$.

Se se suman todas as variables mencionadas anteriormente, obtense a cifra seguinte:

$$REV \cdot [D \cdot (3REF + 2W + 1) + TSA \cdot REF + 3REF] + D \cdot W + 2W + REV.$$

Tomándo só os termos principais, e obviando o parámetro W, dado que é significativamente máis reducido cós demais (na realidade, W toma valores entre 12 e 30), tense que o número de variables para unha instancia concreta do problema será da orde de

$$REV \cdot D \cdot REF.$$

A fin de aportar algo de contexto sobre os valores que poderían tomar estes parámetros nun reparto real, realizaranse uns breves comentarios orientativos sobre cada un deles. Respecto do total de revolucións REV, o tempo que tarda unha bandexa en facer todo o recorrido polo sorter, pode falarse de 2 ou 3 minutos. Tomando $2min\ 30s$ por ciclo, nunha hora completaríanse 24 voltas, e en oito horas xa serían 192. Sobre os destinos, tamén a través da información facilitada polo titor da empresa, pode agardarse unha cifra que ronde os $D = 300$ xutes en total. Finalmente, o valor REF é o máis complicado de estimar, pero non se debe contar con que sexa moito menor cós outros dous parámetros. Se cuestións como a cor, a talla ou o xénero para o que se deseña unha prenda implica o uso de novos SKUs, entón para cada deseño de prenda apreciaremos multitude de referencias diferentes. Algúns deseños só se pensan para un xénero e teñen pouca variedade de cores, mentres que algúns elementos máis xenéricos contan con máis cores, ambos os dous xéneros, así como un maior abano de tallas. Se asumimos que unha prenda promedio poida ter 5 tallas dispoñoibles[3], estar orientada a un só xénero

[3]Dende S ata XL son 4, ás que sumar ou ben a XS ou ben a XXL segundo o caso.

e presentar 4 variantes de cor, estampado ou outro elemento estético, estaríamos supoñendo que cada deseño de prenda aporta 20 SKUs ao cómputo total.

En definitiva, estase a falar de que un caso real comprende millóns de variables. Ademais, obsérvase que o crecemento desta cantidade é moi pronunciado segundo medra, por exemplo, a cantidade de tempo considerada ou a variedade de prendas do modelo. Á vista destas circunstancias, non debería resultar extraño que un solver resolvese correctamente instancias moi pequenas do problema, pero que ao tratar de facer medrar o tamaño das mesmas, o incremento de tempo necesario para obter a solución óptima se dispare.

Tras o acontecido co cómputo de variables totais do modelo, a perspectiva do custo computacional que supón resolver cada instancia para un ordenador non resulta esperanzadora. O primeiro que debe indicarse é que non estamos ante un problema de programación linear, no sentido en que as variables non teñen dominio real. Se fose ese o caso, é coñecida a existencia de métodos de complexidade polinomial (coma o algoritmo de punto interior de Karmarkar), ademais de que o método símplex presenta na realidade, agás casos patolóxicos [4], un rendemento practicamente polinomial (se ben é no peor dos casos un método expoñencial).

Neste caso estamos ante un problema de programación enteira. Se ben as restricións e o obxectivo son todas funcións lineais das variables do modelo, estas son enteiras ou binarias, é dicir, discretas. Polo tanto, sexa cal sexa o método de resolución, se obtén un óptimo global, sempre será un algoritmo de tipo expoñencial. Fronte a isto, pódese argumentar que existen multitude de métodos de resolución de ILPs que poderían ofertar na práctica un rendemento en apariencia polinomial. Poderíase pensar, dunha banda, en métodos de busca local, ou ben en algoritmos baseados en resolver relaxacións do ILP. Non obstante, tamén para este caso, o modelo presenta unha gran problemática: as simetrías.

Debido a que non se pode impoñer unha forma, orde ou algoritmo de caída das prendas nos destinos, para o noso modelo todos eles son intercambiables. Polo tanto, só ao respecto das variables que afectan aos destinos, xa se atopan tantas simetrías como maneiras diferentes de ordenar os D xutes, é dicir, $D!$. Se tomásemos un caso realista con $D = 300$, falaríase dun valor da orde de 10^{614}, o cal o volve absolutamente inasumible.

O anterior supón, na praxis, que os métodos baseados no movemento entre solucións veciñas, se volvan inviables. Amén da posibilidade de caer en solucións subóptimas, como todas as variables son enteiras, a cantidade de solucións veciñas que non aporten mellora ao obxectivo será inxente, e o método escollido tería igualmente que avalialas. Tamén se poderían facer outras consideracións sobre as abundantes variables binarias do modelo e as complicacións que implicarían para aqueles métodos baseados en resolver relaxacións do modelo, mais todo isto redunda no feito que se apreciará na última sección deste capítulo: que un *solver* precisa cantidades de tempo desmedidas para poder resolver instancias que, en tamaño, distan moito do que correspondería a un caso real.

4.2. Simplificacións excesivas. Afastamento da realidade

Máis alá dos inconvenientes que se expuxeron para a resolución de instancias deste modelo, cómpre analizar o significado que ten unha solución deste á hora da súa aplicación no caso real e, sobre todo, a maior ou menor utilidade desa información. Se os datos que pode devolvernos son de gran interese e aplicabilidade real, entón non debería abandonarse a vía do modelo, senón que se tería que afondar en formas de reducir simetrías, no uso de diversos métodos heurísticos de resolución e demais. Non obstante, se na lista de valores óptimos das variables que se obtén hai moi pouca información relevante, sumado ás dificultades para afrontar informáticamente o problema, a conclusión lóxica sería abortar a vía da optimización clásica para estudar outras, como a simulación de escenarios.

O primeiro será definir aquelas variables de indubidable relevancia, por seren as que motivan a construción do modelo en si mesmo. Por suposto, trátase de n^I, n^{WS}, n^P e n^{VR}, que definen o número de traballadores que ocupan cada rol de traballo. A razón para crear un modelo de optimización foi, dende o primeiro momento, averiguar cal é a combinación de valores destas variables que minimiza o

[4] Aqueles deseñados para obter os peores escenarios posibles, coma o Cubo de Klee-Minty.

tempo en que se completa o reparto. Sobre estas variables, a única crítica que cabería realizar sería a relativa a se o modelo, coas simplificacións que contén, realmente devolvería unha asignación que se correspondese coa mellor posible no clasificador de verdade. Esta dúbida tratará de clarificarse cando se contraste o modelo cun simulador.

Dado que a medida da calidade dunha asignación traballadores-roles é o *makespan*, é importante plantexar a posible correspondencia ou non do mínimo valor de revolucións que devolve unha solución do modelo, respecto das que realmente tomaría procesar a mesma demanda nun *sorter* real empregando a asinación obtida. Este é un aspecto a criticar do modelo, pois a gran cantidade de simplificacións que se asumen provocan que, prvisiblemente, as revolucións que indica unha solución óptima e as que se obterían na realidade poidan ser certamente diferentes. Por exemplo, o labor manual dos traballadores modelízase superficialmente, empregando unha constante que serve de promedio da cantidade de prendas que pode procesar, ou de destinos que pode evacuar, sempre no marco dunha revolución. Na realidade isto non acontece así, e a cantidade de prendas procesadas non é constante no tempo. Ademais, o modelo dá por suposto que, durante unha revolución, toda prenda que se induce ten unha oportunidade de caer nalgún dos destinos; e se isto non acontece, recirculará para a seguinte volta. Na realidade, o clasificador ten dous lados (como se mostraba na Figura 2.1) con cadanseus postos de indución e destinos. Polo tanto, as prendas inducidas no primeiro lado terán a posibilidade de caer en destinos do seu propio lado, ou ben nos do seguinte, se é que non lles foi posible. Porén, as prendas inducidas no segundo lado, só terán a oportunidade de caer nos xutes dese lado, e non en todos, co que en caso de non logralo, xa pasarían a considerarse como recirculacións (dende a perspectiva do modelo).

Con todo, será posible comparar as revolucións que mostra o modelo con aquelas obtidas mediante o uso dun simulador, o cal se fará no seguinte apartado. En todo caso, o que si é certo é que, establecidas unhas regras de funcionamento do sorter, se as simplificacións o permitisen, poderían corresponderse finalmente as mellores asignacións obtidas polo modelo con aquelas que se darían na realidade, incluso aínda que o tempo necesario para a satisfacción da demanda non fose o mesmo en ambos os dous casos.

Unha vez vistas as variables que son sen dúbida máis importantes para o modelo, cuestionarase agora a utilidade das demais que integran o modelo, en xeral. A liña argumental principal versa sobre os fortísimos supostos que se deben asumir, polo que a información puntual do que acontece respecto dun actor (persoa ou elemento do clasificador) nunha revolución concreta escasamente se pode extrapolar á realidade. Isto é, se ben se podería empregar o modelo para obter a información máis superficial pero tamén a prioritaria, como o é a distribución dos roles de traballo, non se pode utilizar para obter información máis específica que permita predecir algún aspecto do reparto antes de que este se chegue a procesar. Este é un dos puntos que, xunto coa maior velocidade de execución, xustificará a decisión de abordar este problema mediante un simulador.

O primeiro a destacar é que todo o que rodea á indución da roupa está simplificado ata o extremo: tal é así que todo se afronta supoñendo que cada indutor coloca o mesmo número de prendas, sexa cal for o número de traballadores. Do mesmo modo cos *waterspiders*: asúmese unha aportación fixa por cada un que ocupe este rol, sen ningún matiz adicional. Na realidade isto é moi diferente, pois como xa se explicou no último apartado do capítulo anterior (mostrándose graficamente na Figura 2.2), cada novo indutor que se suma ao clasificador coloca un número menor de prendas que aquel que o precede. Cos *waterspiders* acontece algo semellante: cantas máis persoas haxa para axudar aos indutores, máis tempo pasarán ociosos os *waterspiders* por ter aberto xa todas as caixas que teñen á súa disposición. Se por exemplo se teñen 2 axudantes para 5 indutores, estes rotarán entre os postos para equilibrar a carga de traballo, á vez que deixan caixas abertas para que, durante un certo tempo, o indutor ao que se acaba de axudar non requira máis asistencia. Porén, se houbese o dobre de *waterspiders*, a rotación sería mínima. De se ter que cifrar a aportación que cada un realiza ao conxunto dos indutores, esta medra menos por cada *waterspider* que se sume. Pero ademais, o modelo ignora a existencia das caixas que chegan de almacén e que se deben abrir para inducir o seu contido. Evidentemente, isto faise para non incrementar demasiado o número de variables do modelo, pero desenboca en grandes diferenzas respecto da realidade. Un exemplo claro é a primeira revolución: mentres que na realidade se debería invertir unha certa cantidade de tempo en abrir e escanear as caixas, co cal o número de prendas

inducidas sería máis reducido, para o modelo non supón diferenza con calquera outro ciclo, e polo tanto asume que se poden inducir tantos produtos como resulte a combinación linear das constantes IAB e $WSAB$ polas respectivas variables n^I e n^{WS}.

Respecto das caídas en destino, na realidade a cada prenda inducida asígnaselle un xute no que debe caer, que vén determinado por un algoritmo e transmitido á bandexa mediante fotocélulas distribuídas ao longo do clasificador. O modelo non pode reproducir un destes métodos de Round-Robin, pois nese caso sería difícil mesmo falar dun problema de optimización matemática. O modelo debe contemplar todas as posibles combinacións de caídas de prendas de cada referencia para cada destino e revolución, o cal supón un esforzo enorme a nivel informático. Ademais, se o modelo decidise que a solución óptima implicase que as prendas caesen de xeito diferente, por exemplo en cascada[5], entón non sería posible reproducila na realidade, agás que existise outra solución óptima coa mesma asignación de roles e cunha orde de caída de prendas máis repartida (o cal a efectos prácticos se pode entender como unha simetría). É dicir, que habería casos en que ademais de atopar un óptimo, precisaríase buscar se algunha das súas simetrías ten unha orde de caídas máis realista.

Finalmente, e dado que se deron motivos sobrados polos que as demais variables do modelo case non aportan información de interese na práctica, mencionarase brevemente a parte relativa ás evacuacións. Dunha banda, xa se expresou que o modelo presupón unha cantidade fixa de destinos que un *packer* ou un varios pode evacuar por revolución. Isto non se corresponde coa realidade, máxime no caso dos varios, onde a distancia á que se atopen eses destinos é decisiva, pois a variabilidade é maior[6]. Pero da outra banda, para poder actualizar os contadores dos destinos a cero, o modelo impón que as evacuacións duren unha volta enteira, polo que desde que un destino se satura ata que poidan volver a caer prendas nel, aínda que un traballador o evacúe ao momento, pasará como mínimo o tempo que toma unha revolución completa do clasificador. Isto, claramente, aumenta aínda máis a fenda entre os tempos que marcará o modelo para un óptimo, e os que potencialmente tería a aplicación desa solución nun caso real.

4.3. Peculiaridades do problema. Sería posible empregar un simulador?

Ao longo do presente capítulo, veuse a poñer en cuestión non só a capacidade do modelo para ofrecer solucións óptimas a repartos concretos, senón temén que eses óptimos se correspondan cos que habería que aplicar no clasficador de verdade. O xeito ideal de resolver esta dúbida sería a experimentación, é dicir, realizar unha proba cunha parte reducida do *sorter* real e cunha demanda sinxela. Por obvios motivos, isto non é unha posibilidade a contemplar. A opción escollida será a de crear un simulador, un programa que ensaie cada posible combinación das variables n^I, n^{WS}, n^P e n^{VR} e devolva aquela ou aquelas que obteñen menores tempos de duración do reparto.

A construción de tal ferramenta implica varios desafíos a nivel informático, pois debe acharse a maneira de traducir os diferentes aspectos do clasificador a constructos como variables, funcións, obxectos... Con todo, o gran reto é, cando menos en apariencia, o número de combinacións posibles das variables. Cada posible asignación de valores supón un escenario diferente para o que debe simularse un mesmo reparto. Isto debería volver inviable esta idea, dado que catro variables enteiras poden producir un número desmedido de combinacións posibles. Agora ben, realmente son tantas?

Fixándonos só no modelo, hai unha inmensidade de posibilidades para as catro variables, dependendo de poucas restricións. A máis importante delas é que a suma dos traballadores asignados a cada rol sexa W. O habitual é que un clasificador teña 16 ou 18 traballadores, podendo haber máis nalgúns *sorters* de gran tamaño, pero non máis de 30. Imos supoñer neste momento que $W = 20$. Aínda así,

[5] Cando se fala de caída de prendas en cascada referímonos a que todas busquen o primeiro destino dispoñible. Cando este se sature, pasaran a caer todas ao seguinte e así sucesivamente.

[6] Debe lembrarse que os varios non se circunscriben a un espazo concreto, senón que se desprazan por todo o *sorter*, polo que a distancia potencial entre dous destinos é superior.

só coa condición de que as catro variables sumen W, estamos ante

$$\binom{20+4-1}{4} = \frac{23!}{4! \cdot 19!} = 8855 \quad \text{posibilidades.}$$

Este valor obtense do número de 4-combinacións con repetición dun conxunto con 20 elementos.

Agora ben, resta engadir as demais restricións que se lle impoñen aos posibles valores de cada variable. Comezando polo número de *packers*, este debe ser obrigatoriamente par, o cal reduce o número de posibilidades. Tamén a cantidade de indutores está limitada pola de postos de indución, dos que podemos supoñer que son $IPOST = 12$. Faltan dous condicionantes por ter en conta: que $n^{WS} < n^I$ e que $n^{VR} < n^P$.

Para a posta en práctica do simulador, é necesario coñecer, dados uns parámetros, o número de combinacións posibles aos que nos estabamos a referir anteriormente. A tal fin creouse unha función `get_all_assignments` que as calcula e as agrupa todas nunha lista mediante un bucle moi sinxelo. A Figura 4.1 mostra o código que a define, moi sinxelo, e empregarase ademais para coñecer todas as posibles combinacións de valores das catro variables asumindo que $W = 20$ e que $IPOST = 12$.

```
1  def get_all_assignments(W_TOTAL, IPOST):
2  assignments = []
3    for INDUCTORS in range(1, IPOST + 1):
4      for PACKERS in range(2, W_TOTAL - INDUCTORS + 1, 2):
5        for WATERSPIDERS in range(INDUCTORS):
6          for VARIOS in range(PACKERS):
7            if INDUCTORS + PACKERS + WATERSPIDERS + VARIOS == W_TOTAL:
8              assignments += [[INDUCTORS, WATERSPIDERS, PACKERS, VARIOS]]
9    return assignments
10
11 print(len(get_all_assignments(20, 12)))
12 #166
```

Figura 4.1: Función `get_all_assignments` para a obtención dos posibles escenarios.

O contraste entre as 8855 posibilidades que se crían inicialmente, e as 166 que finalmente cumpren as restricións para poder orixinar solucións factibles do modelo, é moi considerable. Se ben se tomaron como referencia valores que o titor das prácticas indicou como "máis comúns", pode aplicarse a función anterior para ver o que acontecería nos casos máis xenerosos que referiu o dito titor. Se $W = 30$, habería 360 posibilidades se $IPOST = 12$, 432 se $IPOST = 14$ e 492 en caso de que $IPOST = 16$.

As cifras anteriores, a nivel computacional, son a priori un número reducido. Non obstante, o factor determinante para concluír isto é o tempo que ao simulador lle tome ensaiar cada un dos posibles escenarios. Para un caso de tamaño realista, é de agardar que haxa un custo computacional elevado para cada simulación, e polo tanto podería ser que este número de escenarios posibles, aínda que moi baixo a nivel combinatorio, fose demasiado elevado para estándares comerciais ou de produto.

Agora ben, dadas as características do problema, é doado predicir de antemán que certas asignacións de valores presentarán tempos de cumprimento da demanda moi reducidos. Un primeiro exemplo é aquel onde o número de indutores sexa moi reducido, un caso no que a entrada de prendas ao *sorter* sería moi lenta, o cal arrastraría aos demais procesos. Algo parecido pode acontecer se o número de indutores e *waterspiders* é demasiado alto, ou o que é o mesmo, que os encargados das evacuacións sexan insuficientes. Desta vez, os destinos pasarían demasiado tempo saturados debido á alta carga de traballo que experimentarían os seus encargados, non podendo caer novas prendas neles e aumentando o tempo de finalización da demanda.

No caso de se estableceren criterios para a exclusión de escenarios a priori pésimos en termos de función obxectivo, rebaixaríase o número de simulacións a realizar polo programa, aforrando significativamente no tempo de execución. Por poñer un exemplo, se no caso en que $W = 20$ e $IPOST = 12$

se impón que $n^I \geq 5$ e que $n^P + n^{VR} \geq 8$, o número total de combinacións a simular baixa dende as 166 iniciais a só 87.

Neste punto, a opción do simulador semella cada vez máis viable, en especial ao contrastalo co modelo. Onde este último non é capaz de escalar e polo tanto non permite resolver o problema de ningunha das maneiras, o simulador abre unha porta a obter solucións nunha cantidade de tempo que, maior ou menor, virá dada polo número de asignacións a comprobar (e, como non, da eficiencia do propio código).

Finalmente, cómpre destacar que a simulación de cada escenario por separado é un proceso independente, e polo tanto, paralelizable. Paralelizar un conxunto de procesos significa que, no canto de que o ordenador os realice secuencialmente (é dicir, un a continuación do outro), o equipo divida os seus recursos en varias partes onde cada unha realizará un proceso diferente á vez. É dicir, que se poderían avaliar as diferentes combinacións de valores das variables de dez en dez ou de vinte en vinte, e non de unha en unha. Por exemplo, resolver dez asignacións á vez toma moito máis tempo que facelo cunha soa, pero á vez moito menos tempo que resolver esas dez de xeito secuencial. Isto, especialmente en servidores empresariais, supón un acicate para dar utilidade e viabilidade á proposta de implementar un simulador.

4.4. Comparativa de modelo contra un simulador básico

A primeira fase na creación dun algoritmo, como alternativa ao modelo, foi precisamente aquel que tomase como referencia as súas simplificacións. Unha das motivacións para proceder deste modo era a posibilidade de contrastar os resultados de ambas as dúas ferramentas á hora de resolver instancias reducidas do problema. Neste caso, o condicionante do tamaño das instancias é o modelo, pola súa peor capacidade para escalar. Polo tanto, ensáianse só aquelas cauísticas que o modelo poida resolver, e mesmo se inclúe algunha na que o modelo tarda varias horas en facilitar a solución, unicamente de xeito ilustrativo e para explicitar a diferenza que supón respecto do simulador.

Aínda que se sabe de antemán que o rendemento do simulador será moito mellor co do modelo, realizar este pequeno estudo comparativo ten interese polo feito de comprobar se as solucións que aporta un instrumento e o outro son similares ou non. Búscase en particular comprobar se o simulador devolve como mellor asignación aquela que é óptima para o modelo, ou se no seu defecto o óptimo se atopa entre as mellores opcións que manexa o simulador (en termos, por suposto, de cumprimento da demanda). Isto pode interpretarse como unha certa validación do modelo, polo menos nesta fase inicial, ao ser capaz de dar as solucións do modelo, pero dun xeito moito máis eficiente. Tamén indicaría, en certo modo, que a lóxica do *sorter* está ben traducida ao código que rexe o comportamento do simulador.

Unha cuestión importante desta comparativa é que ignorará o rol dos *waterspiders*, isto é, que o simulador non os considerará e no modelo imporase que $n^{WS} = 0$. Como se afirmou reiteradamente, a indución no modelo afróntase de modo moi reduccionista. No simulador, se ben pode ser algo máis realista, nesta versión aínda non incorpora o auténtico labor dos *waterspiders*: a apertura das caixas para que non a teña que realizar o indutor ao que está a apoiar. Dedúcese pois que ignorar deliberadamente este rol neste pequeno estudo libéranos de ter que afrontar esta posible fonte de discrepancias de resultados entre ambas as dúas ferramentas de resolución.

4.4.1. Descrición do funcionamento do simulador e o comparador

Este primeiro simulador que se confronta co modelo de optimización clásica é unha versión primitiva daquel que se presentará no seguinte capítulo. Ademais, esta versión e a comparación de resultados foi, en boa parte, a motivación para afrontar o problema do clasificador mediante a construción dun simulador.

A definición pormenorizada do programa informático creado non é o máis acaído nesta parte do traballo: os detalles de programación non están dentro do abarcado por este traballo, xa se definirá con

calma a versión final do simulador no seguinte capítulo, e finalmente algunhas das explicacións que se poderían dar para esta versión non aplicarían para seguintes versións.

Con todo, é interesante explicar *grosso modo* a lóxica básica que emprega o simulador construído en *Python*. Esta exposición será principalmente gráfica mediante o uso de diagramas de fluxo que, de modo moi superficial, tratarán de aportar as nocións máis primarias sobre como e en que orde teñen lugar os procesos do clasificador. Para os diagramas de fluxo que se empregarán neste e no vindeiro capítulo, os elementos gráficos presentes nel corresponderanse en significado coa seguinte lenda:

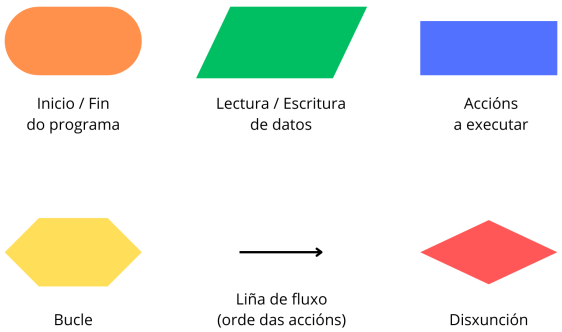


Figura 4.2: Lenda de símbolos a empregar nos diagramas de fluxo.

Primeiramente, o modelo de optimización tamén se resolve en *Python* mediante o uso da librería *Pyomo*, especializada neste tipo de problemas. A definición das instancias, isto é, dos conxuntos de parámetros que permiten construír o problema, realízase mediante un diccionario que o programa `model_function` lerá dende un arquivo. Unha vez lidos os datos, créanse as variables, restricións e función obxectivo, para ser resolto polo *solver* que se indique no código.

O simulador, pola súa parte, estará estruturado en base a obxectos de diferentes clases. Por exemplo, cada indutor está representado por un obxecto da clase `Inductor`, e do mesmo modo existen clases como `Packer`, `Box` (clase dos destinos) ou `TSA` (para contabilizar o cumprimento da demanda). Neste caso, ademais das revolucións, o simulador traballará con cada bandexa de xeito individual, algo que no modelo non era posible polo enorme custo en variables. Tamén estas son obxectos dunha clase `Tray`.

Faise esta aclaración para mostrar que o simulador non pode traballar directamente co mesmo diccionario. Precísase coñecer a asignación de traballadores que vai poñer en práctica para así crear os obxectos correspondentes. Non obstante, isto pode resolverse aplicando, para cada asignación a ensaiar, unha función `data_converter`, que toma o dicionario e a asignación como argumentos e devolve un obxecto con todos os elementos que conforman o clasificador a recrear.

Con todos estes ingredientes conformamos o programa comparador `main_compare`. Este recibe varias instancias a resolver e, para cada unha, aplica primeiro o modelo e logo o simulador, logo de obter todas as asignacións posibles e facendo a conversión dos datos que se acaba de explicar. Todo o anterior resúmese no diagrama da Figura 4.3.

Sobre a propia función simuladora, lonxe de explicala con gran profundidade, comentarase como funciona de xeito moi superficial. Como xa se dixo, o simulador actuará a nivel de cada bandexa, polo que existirán tres niveis de acción no simulador. O primeiro é a revolución, entendida como o paso das bandexas por todas as posicións do clasificador ata volver ao lugar de referencia. En segundo lugar, o paso de tempo, que é a foto fixa onde cada bandexa ocupa unha posición concreta so *sorter*. E finalmente, cada bandexa, que nun intre de tempo dado dunha revolución concreta, ocupará unha certa posición.

Entendido isto, nun momento concreto, fixada unha bandexa, esta pode estar nunha de tres situa-

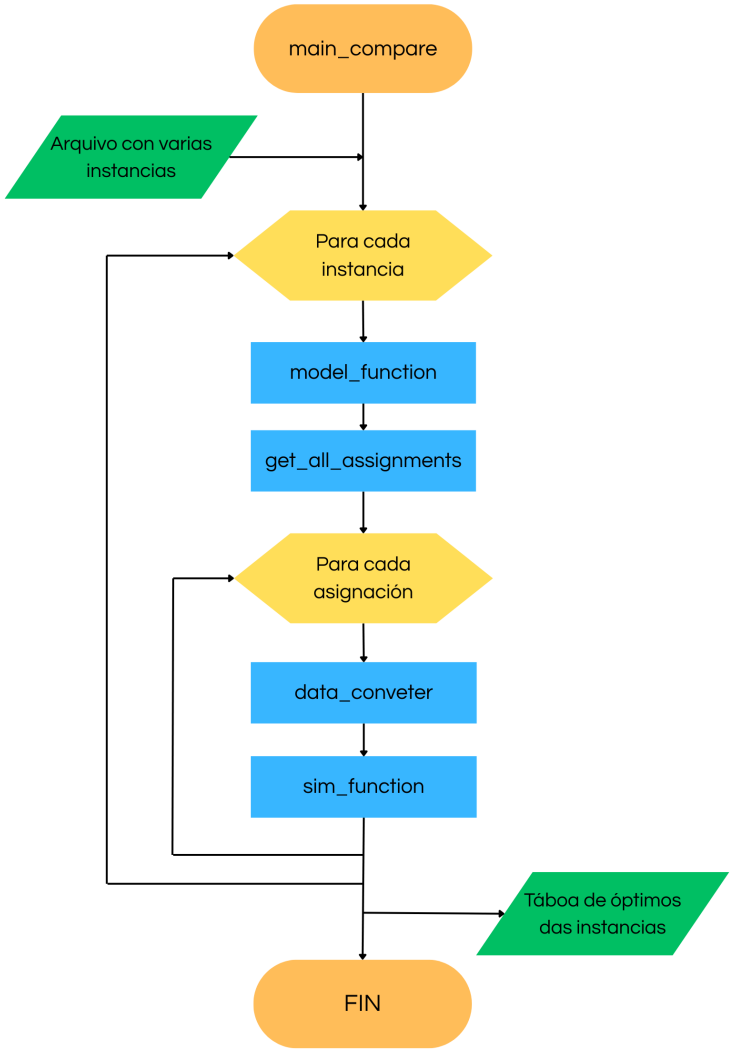


Figura 4.3: Diagrama de Fluxo do programa `main_compare.py`.

cións. Se está sobre un posto de indución, entón deben realizarse as accións correspondentes (comprobar se está baleira, e se o indutor pode inducir nela). Tamén pode acontecer que estea sobre un destino, e neste caso cómpre realizar dúas preguntas: se a caixa está xa chea, o cal activaría o código responsable da evacuación; e se é posible a caída da prenda da bandexa (se a houber). Finalmente, cada vez que unha prenda cae, compróbase se esa é a que restaba para cumprir a demanda, caso en que remataría a execución e se anotaría o número de revolucións trascorridas. Esta explicación compréndese de xeito máis gráfico observando a Figura 4.4.

4.4.2. Resultados

A continuación mostraranse os resultados de resolver varias instancias mediante o programa `main_compare` que se introduciu no apartado anterior. A idea consiste en seleccionar varias partes das instancias e resolver variacións de cada unha para ver se hai máis ou menos discrepancias de resultados. Para maior comprensión desta cuestión, a instancia de referencia para este estudo vén dada polo dicionario de *Python* recollido na Figura 4.5. Por exemplo, pode ser de interese comprobar como se comportan ambas as

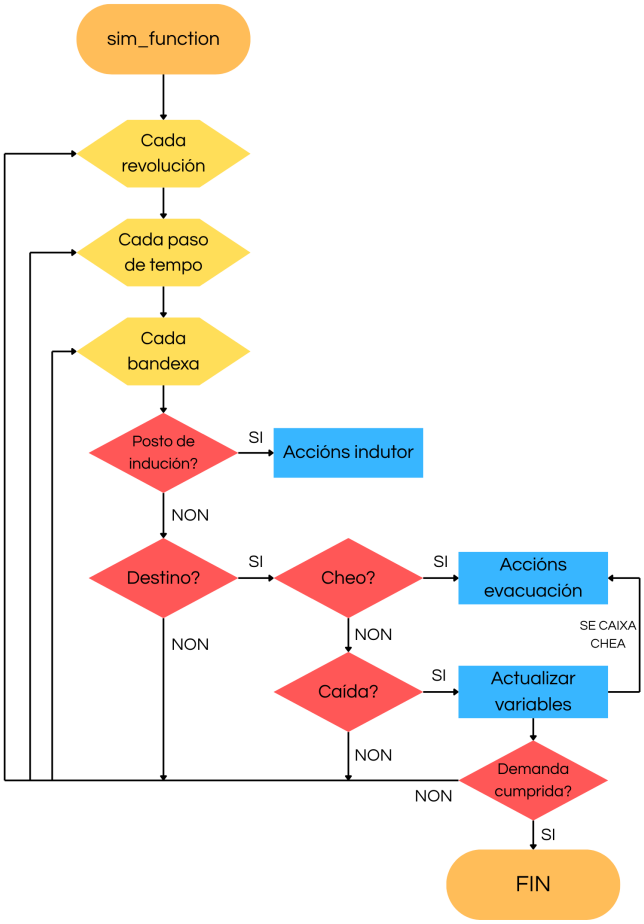


Figura 4.4: Diagrama de Fluxo da función `sim_function.py`.

dúas ferramentas se, tomando este punto de partida, se aumenta a demanda de xeito simétrico, é dicir, na mesma medida para todas as TSAs e SKUs. En tal caso, podemos tomar esta instancia, unha na que se demanden 10 produtos de cada tipo, 20... O mesmo se pode facer para outros aspectos do problema, e así, e caso de xurdir nalgún apartado unha gran discrepancia entre os óptimos de modelo e simulador, identificar máis facilemnte as causas que os producen.

A primeira proba que se realiza consiste en aumentar de xeito simétrico a demanda de prendas de todas as referencias por parte de todas as tendas. Na instancia base da Figura 4.5 cada TSA pide 5 pezas de roupa de cada SKU. Resolverase este caso, así como aquel en que se piden $10, 15, 20$ e 30 prendas de todas as referencias por parte de todas as tendas.

Antes de comentar específicamente os resultados mostrados na Figura 4.6, explicarase a súa estrutura para comprender mellor as seguintes comparativas que se comentarán. En cada unha das tres táboas, a primeira columna indica o parámetro que se está a modificar no estudo. Por exemplo, neste caso modifícanse por igual todos os $DEMAND_{ti}$, é dicir, a demanda de cada tenda de produtos de cada SKU. Agora ben, a primeira táboa (superior esquerda) estuda as asignacións óptimas (no formato $'n^I, n^P, n^{VR}$,). A segunda e cuarta columna indican, respectivamente, a asignación óptima proporcionadas por modelo e simulador. A terceira columna dinos que posición ocupa o óptimo do modelo dentro das métricas do simulador. É dicir, en caso de que os óptimos non coincidan, consideramos aceptable que a proposta do modelo sexa por exemplo a terceira mellor asignación ao entendemento

```
data_1 = {None: {
        'W_TOTAL': {None: 9},
        'W': {None: range(1, 10)},
        'S': {None: 100},
        'IPOST': {None: 7},
        'REV_TOTAL': {None: 25},
        'REV': {None: range(1, 26)},
        'REF': {None: range(1,4)},
        'REF_TOTAL': {None: 3},
        'VOL': {1:1, 2:2, 3:1} ,
        'TSA': {None: range(1,4)},
        'TSA_TOTAL': {None: 3},
        'D': {None: range(1,22)},
        'DEMAND': {(1,1):5, (1,2):5, (1,3):5,
                   (2,1):5, (2,2):5, (2,3):5,
                   (3,1):5, (3,2):5, (3,3):5 },
        'TSADEST': {1: range(1,8), 2: range(8, 15), 3: range(15, 22)},
        'B': {None: 10},
        'LEMARGIN': {None: 1},
        'UEMARGIN': {None: 1},
        'IAB': {None: 6},
        'PAB': {None: 3},
        'WSAB': {None: 1},
        'VRAB': {None: 4},
}}
```

Figura 4.5: Instancia de referencia para a comparativa entre modelo e simulador.

do simulador, especialmente se o número de revolucións que implica cada proposta é moi similar.

A segunda táboa, a superior dereita, está adicada ao número de revolucións. A segunda e cuarta columna informan das voltas que, para a súa asignación óptima, lle toma satisfacer a demanda a modelo e simulador, respectivamente. A terceira columna, de xeito similar á táboa anterior, mostra cantas revolucións conclúe o simulador que toma o reparto para a mellor asignación dada polo modelo. Como se pode comprobar, as terceiras columnas destas dúas táboas son un modo de validar, en certo sentido, os resultados. O motivo é que permiten ver como de semellantes son o funcionamento de ambas as dúas ferramentas. Aínda que os resultados non foren iguais, seguirán a ser prometedores se o simulador e o modelo obteñen resultados parecidos ou 'próximos' alá onde non coinciden. Finalmente, a terceira táboa é a máis simple, pois mostra os tempos de execución (en segundos) que lle toman ao modelo e o simulador, espectivamente, obter a mellor asignación de traballadores. En xeral, a diferenza nestas táboas será abismal, beneficiando ao simulador e prexudicando ao modelo de optimización clásica.

Dito todo isto, a táboa da Figura 4.6 amosa un comportamento moi bo. Se ben no escenario de referencia os óptimos son distintos, isto débese á concentración de solucións factibles preto do óptimo. Isto extráese de que, pese a que a proposta do modelo é a cuarta mellor a ollos do simulador, para este último só hai unha revolución de diferenza entre ambas, polo que como mínimo hai tres solucións que só distan nunha revolución do óptimo (segundo a lóxica do simulador). Comprobamos ademais, ollando a terceira columna da segunda táboa, que polo xeral o simulador tende a asignar tempos un pouco menores (en particular unha revolución menos), pero que esta diferenza é consistente ao longo das instancias. Finalmente, debe dirixirse a atención cara os tempos de execución. Segundo medra o tamaño do problema, os tempos de execución do simulador suben de maneira moi contida[7], pero non así o modelo. Estes casos son sumamente reducidos e, malia isto, para resolver unha demanda de só 30 prendas por cada TSA e SKUs [8] o modelo tarda preto de dúas horas. Este tipo de diferenzas nos tempos de execución repítense ao longo de todas as comparativas, mais dende aquí trataranse de evitar os casos nos que o modelo tarde tanto tempo en rematar as súas execucións.

[7]Con todo, esta é unha versión primitiva do programa, polo que o código non é todo o eficiente que debería.
[8]Só hai nestas instancias 3 TSAs e 3 SKUs.

Nº items (REF & TSA)	Model optimal assignment	Position in simulator ranking	Simulator optimal assignment
5	5,4,0	4	7,2,0
10	7,2,0	1	7,2,0
15	7,2,0	1	7,2,0
20	7,2,0	1	7,2,0
30	7,2,0	1	7,2,0

Nº items (REF & TSA)	Min. revolutions with model	Rev. of model sol. with simulator	Min. revolutions with simulator
5	3	3	2
10	4	3	3
15	5	4	4
20	6	5	5
30	8	8	8

Nº items (REF & TSA)	Model execution time	Simulator execution time
5	13.8855	0.2510
10	95.0035	0.4669
15	569.4525	0.4408
20	805.6334	0.5240
30	7117.3183	0.7185

Figura 4.6: Resultados da comparativa ao incrementar simetricamente a demanda.

Tamén é posible aumentar ir aumentando a demanda de todas as tendas só dalgunha referencia concreta, ou á inversa, aumentar a demanda de prendas de todas as referencias só para algunha das tendas. Estes casos tamén se ensaiaron dun modo similar ao anterior. Como os resultados son iguais [9] non se van incluír as súas respectivas ternas de táboas neste apartado. Non obstante, poderanse consultar no Apéndice A.2 deste traballo, onde aparecen acompañadas de cadansúa instancia de referencia. Co mesmo resultado, tamén se probou a modificar únicamente o valor de W mantendo os demais parámetros fixos, un experimento cuxas táboas tamén aparecen no apéndice.

Non obstante, honrando a verdade, vaise a comentar o único caso dos estudados no que o comportamento de modelo e simulador é diferente, se ben non ten afectación algunha na práctica. A instancia de partida que se considerou foi a que se mostra na Figura 4.7.

Para comentala rápidamente, esta instancia contempla prendas de tres SKUs con diferente volume cada unha, e tres tendas que realizan a mesma demanda de cada referencia. Cada TSA pide 10 prendas da referencia de volume $1ud$, 7 da de volume $2ud$ e 3 da de $3ud$ (tómase ud como unidade ficticia de volume). Pode observarse como cada TSA ten asignados tres destinos. O parámetro que se modifica é xusto ese, o reparto dos destinos para as TSAs. O que se quere ver é o comportamento de modelo e simulador cando esta asignación se volve máis desigual.

En efecto, pode verse na Figura 4.8 como cando a asignación de destinos ás tendas se torna desigual, os resultados de modelo e simulador vólvense moi diferentes. Tamén se pode apreciar como, aínda que a demanda non medra dunha instancia a outra, o modelo pasa de tardar 43 segundos en resolver o caso onde a asignación é equilibrada, a tardar 47 minutos en resolver o caso con menor equilibrio na asignación (isto afírmase porque, como se comentou antes, todas as tendas realizan a mesma demanda).

[9] No sentido en que o modelo e o simulador dan os mesmos óptimos cun número de revolucións similar, ademais de manterse as enormes diferenzas nos tempos de execución

```
data_1 = {None: {
        'W_TOTAL': {None: 12},
        'W': {None: range(1, 13)},
        'S': {None: 100},
        'IPOST': {None: 8},
        'REV_TOTAL': {None: 25},
        'REV': {None: range(1, 26)},
        'REF': {None: range(1,4)},
        'REF_TOTAL': {None: 3},
        'VOL': {1:1, 2:2, 3:3} ,
        'TSA': {None: range(1,5)},
        'TSA_TOTAL': {None: 4},
        'D': {None: range(1,13)},
        'DEMAND': {(1,1):10, (1,2):7, (1,3):3,
                (2,1):10, (2,2):7, (2,3):3,
                (3,1):10, (3,2):7, (3,3):3,
                (4,1):10, (4,2):7, (4,3):3},
        'TSADEST': {1: range(1,4), 2: range(4, 7), 3:range(7,10), 4:range(10,13)},
        'B': {None: 4},
        'LEMARGIN': {None: 1},
        'UEMARGIN': {None: 1},
        'IAB': {None: 5},
        'PAB': {None: 2},
        'WSAB': {None: 1},
        'VRAB': {None: 4},
}}
```

Figura 4.7: Instancia de referencia para a segunda comparativa.

Non obstante, este 'caso patolóxico', como se podería denominar, non sucede na realidade. Na práctica trátase de buscar un equilibrio entre a demanda das tendas e o número de destinos que lle son asignados a cada unha.

En resumo, o modelo e o simulador presentan un comportamento moi similar a medida que medra a demanda ou varía o número de traballadores totais. Tamén se experimentaron outros aspectos, coma o crecemento da demanda pola aparición de novas tendas, con idénticos resultados, polo que non se inclúen todos neste pequeno estudo. Ademais, a outra gran conclusión é a abismal diferenza nos tempos de execución, que mostran o potencial que ten a construción dun simulador máis avanzado como ferramenta para chegar a resolver o problema do *sorter* en casos dunha dimensión máis realista.

Nº destinations per TSA	Model optimal assignment	Position in simulator ranking	Simulator optimal assignment
3, 3, 3, 3	8,2,2	15	5,6,1
6, 2, 2, 2	4,8,0	1	4,8,0
4, 4, 2, 2	6,4,2	10	4,8,0
5, 4, 2, 1	6,4,2	4	4,8,0

Nº destinations per TSA	Min. revolutions with model	Rev. of model sol. with simulator	Min. revolutions with simulator
3, 3, 3, 3	6	10	4
6, 2, 2, 2	8	6	6
4, 4, 2, 2	8	8	6
5, 4, 2, 1	14	12	7

Nº destinations per TSA	Model execution time	Simulator execution time
3, 3, 3, 3	43.3668	0.2676
6,2,2,2	204.7974	0.3491
4,4,2,2	1313.9814	0.3926
5,4,2,1	2826.7202	0.4866

Figura 4.8: Resultados da comparativa ao modificar a asignación de destinos ás tendas.

Capítulo 5

Implementación do simulador

A conclusión do capítulo 4 era a aparición do simulador como única proposta viable para poder acadar resultados do problema, vista a ineficacia do modelo de optimización proposto no capítulo 3, que tomaba grandes cantidades de tempo para resolver instancias de tamaño moi reducido.

Polo tanto, esta parte do traballo adícase enteiramente a desenvolver mencionado simulador, que na última sección do capítulo anterior aparecía como unha versión moi primitiva. Non entra dentro dos obxectivos deste traballo detallar cada punto do código, polo que realizará unha explicación máis ben superficial e apoiada en diagramas de fluxo. Tampouco será proporcionado o código íntegro do simulador, senón que se mostrarán só algúns fragmentos do mesmo que permitan aclarar algunhas cuestións importantes, ben na parte descritiva ou ben na faceta matemática desta ferramenta. A decisión de non publicar o código íntegro débese a que foi desenvolvido no marco dun convenio de prácticas cunha empresa privada. Se ben non se traballa con datos reais neste texto, óptase por non mostrar máis código do necesario para a mellor comprensión do problema por parte dun lector vinculado ao ámbito matemático.

O primeiro apartado deste capítulo trata sobre o simulador en si mesmo, explicando o mellor posible o seu funcionamento. En particular, ademais da exposición *grosso modo* da súa lóxica interna, abordaranse con maior detemento os aspectos referidos á natureza dos datos cos que se traballa, e a maior ou menor facilidade de obtención dos mesmos; así como os bucles que presenta, por ser un factor determinante no seu custo computacional. Tamén se explicará en que puntos se deben executar varios procesos independentes entre si, dado que serán susceptibles de se poderen paralelizar, o cal permite un horizonte con tempos de execución moito máis reducidos[1].

Tras por en coñecemento o funcionamento do simulador, a seguinte sección adicarase a presentar e comentar os resultados dalgunhas execucións dun tamaño máis realista, a total diferenza das instancias vistas no capítulo anterior. Se ben o tamaño da demanda non se chega a corresponder co dun reparto completo de gran tamaño (que podería chegar a comprender preto das 200000 prendas en total), si que acada as decenas de miles de produtos, un tamaño suficiente para extraer conclusións máis acaídas á realidade sobre a utilidade do simulador. Non é preciso forzar a un equipo local, coma un ordenador portátil, a asumir a simulación dun reparto tan grande, polo menos á hora de comprobar o potencial da ferramenta. Se esta se chegase a implementar para casos reais, a potencia requerida é claramente a dun servidor e non a dun equipo illado.

Finalmente, adícase unha última sección a comentar de que maneiras se poden mellorar os tempos de execución obtidos no apartado anterior. Estas cuestions van dende o linguaxe de programación empregado, a refactorización do código para ser máis eficiente ou o afondamento nun maior número de procesos susceptibles de se paralelizaren.

[1] É dicir, se varios procesos se poden paralelizar, á hora de executar o algoritmo en servidores de tamaño empresarial, agardarase unha caída dos tempos de execución respecto á de calquera equipo local ou persoal (coma un ordenador portátil).

5.1. Descrición da lóxica do simulador

O simulador está programado, como a versión primitiva empregada no capítulo 4, na linguaxe de programación *Python*. Esta, se ben é externa ao ecosistema no que se implementaría a ferramenta en caso de se implementar (úsanse outras linguaxes), é un marco moi cómodo e flexible para as persoas procedentes da programación orientada a matemáticas. A súa curva de aprendizaxe é certamente suave, ten un amplo acceso a librarías para suplir calquera necesidade de programación e, finalmente, permite traballar de xeito moi intuitivo con obxectos, unha peza fundamental deste simulador tal e como está plantexado.

O prototipo de simulador do capítulo anterior e o que aquí se presentará mostran dous tipos de diferenzas. Dun lado, algúns aspectos 'macro' modifícanse: o modo de crear un obxecto representativo do clasificador ou o modo e orde de chamada das funcións. Do outro, este simulador abarca unha cantidade moito maior de procesos do *sorter* real. Inclúense as caixas chegadas de almacén, o que permite recrear de xeito realista toda a parte da indución: a apertura de caixas, a colocación do seu contido no sistema, o labor dos *waterspiders*... Os destinos aparecen xa estruturados como na realidade, pertencendo cada un deles a un lado, grupo e sección concretos. Tamén se implementa un algortimo de Round-Robin que se asemella á asignación de xutes ás prendas mediante as fotocélulas, e en xeral, un aumento considerable da complexidade do problema que se traslada a liñas de código.

A descrición do simulador estruturarase, para a súa mellor comprensión, en catro puntos. Primeiramente, comentaranse a trazo groso os datos de entrada que comporta. Logo, falarase de como o simulador se fundamenta no uso de obxectos, que clases se definen e que utilidade teñen. Finalmente, os apartados restantes tratan de bosquexar a estrutura da función que, dada unha instancia unha asignación, recrea o comportamento do clasificador; e p programa principal que coordina todo o proceso de simulación.

5.1.1. Entrada de datos

É importante deterse brevemente neste punto, pois a información que se envía ao simulador para tratar de achar a asignación óptima de traballadores ten dous tipos de natureza diferentes, que convén explicar.

Dunha banda, atópanse aqueles datos que indubidablemente comprende cada instancia que se deba resolver: a demanda, o número total de traballadores a asignar, as características propias do simulador[2]... Estes datos teñen correspondencia exacta coa realidade e veñen facilitados en ficheiros doutros soportes, que deben ser traducidos ao formato de entrada do simulador.

Mais da outra banda, o algoritmo precisa doutros datos que, se ben a empresa pode calcular, non son tan exactos, pois son aqueles que atenden ao factor humano. Neste grupo atoparíamos información como por exemplo canto tempo lle toma a unha persoa abrir e escanear unha caixa, ou evacuar un destino, ou calquera outra capacidade humana que afecte ao modelo. Estes datos poden ser estimados de varias formas, pero a súa correspondencia coa realidade non é exacta. Para facilitar esta argumentación, a Figura 5.1 inclúe os nomes dos parámetros ou conxuntos de parámetros que constitúen unha instancia que o simulador pode executar.

Na Figura 5.1 pode identificarse facilmente o primeiro tipo de datos como aqueles que comparten nome con parámetros xa vistos no modelo do capítulo 3. Son tales como o número máximo de revolucións `N_REVOLUTIONS` a considerar, ou a cantidade `N_IPOST` de postos de indución, a demanda `DEMAND_TSA` ou mesmo a capacidade `BOX_LIMIT` das caixas en destino.

Tamén se precisou engadir nova información desta natureza, a fin de asemellar mellor o *sorter* do simulador con aquel da realidade. Por exemplo, a variable[3] `DESTINATION_DISTANCE` representa a distancia, medida en número de bandexas (ou o que é o mesmo, posicións [4]), que separa a dous destinos

[2] Aquelas como o número de bandexas, a disposición e cantidade de postos de indución e destinos, etc.

[3] Neste capítulo, agás que se especifique o contrario, a palabra 'variable' emprégase no seu contexto informático e non no dos modelos de optimización.

[4] Como xa se comentou anteriormente no traballo, dada unha foto fixa do *sorter* antes de comezar a actividade, pódese

```
data_dict = {
        'N_REVOLUTIONS': N_REVOLUTIONS,
        'N_WORKERS': N_WORKERS,
        'N_POSITIONS': N_POSITIONS,
        'N_IPOST': N_IPOST,
        'IPOST_POSITIONS': IPOST_POSITIONS,
        'N_GROUPS_PER_SIDE': N_GROUPS_PER_SIDE,
        'N_SECTORS_PER_GROUP': N_SECTORS_PER_GROUP,
        'N_DESTINATIONS_PER_SECTOR': N_DESTINATIONS_PER_SECTOR,
        'N_DESTINATIONS': N_DESTINATIONS,
        'DESTINATION_DISTANCE': DESTINATION_DISTANCE,
        'SECTOR_DISTANCE': SECTOR_DISTANCE,
        'GROUP_DISTANCE': GROUP_DISTANCE,
        'DESTINATION_LIST': DESTINATION_LIST,
        'DESTINATION_POSITIONS': DESTINATION_POSITIONS,
        'BOX_LIMIT': BOX_LIMIT,
        'LOWER_MARGIN': LOWER_MARGIN,
        'UPPER_MARGIN': UPPER_MARGIN,
        'ITEMS_PER_INCOMING_BOX': ITEMS_PER_INCOMING_BOX,
        'REF': REF,
        'TSA': TSA,
        'VOL': VOL,
        'DEMAND_TSA': DEMAND_TSA,
        'DEMAND_REF': DEMAND_REF,
        'OPENING_TIME': OPENING_TIME,
        'INDUCTION_TIME': INDUCTION_TIME,
        'EVAC_TIME': EVAC_TIME,
        'MOVE_TIME_DEST': MOVE_TIME_DEST,
        'MOVE_TIME_GROUP': MOVE_TIME_GROUP,
        'MOVE_TIME_IPOST': MOVE_TIME_IPOST,
        'MAX_INCOMING_BOX_PER_IPOST': MAX_INCOMING_BOX_PER_IPOST,
        'TSA_DEST': TSA_DEST,
        'MULTISECTOR': MULTISECTOR
}
```

Figura 5.1: Dicionario de *Python* que recolle o tipo de datos que require o simulador.

consecutivos do mesmo sector. De modo similar, as variables `SECTOR_DISTANCE` e `GROUP_DISTANCE` indicarían a distancia entre o último destino dun sector ou grupo, respectivamente, e o primeiro destino do seguinte.

Agora ben, tal e como se dixo, hai parámetros dunha natureza menos exacta, e que se deberían obter a partir da experiencia en casos reais pasados. Ao ser un simulador que inclúe ao factor humano, é inevitable o uso de parámetros que regulen determinadas actividades humanas. Con todo, á hora de establecer estes datos de entrada tentouse que fosen, na medida do posible, aqueles máis sinxelos de obter mediante métricas sinxelas dos traballadores. Por exemplo, `OPENING_TIME` representa o tempo que lle toma a un indutor ou *waterspider* abrir e escanear unha caixa, antes de que se poidan comezar a inducir as súas prendas. Este valor non é exacto, pero pode extraerse promediando un conxunto de observacións sobre o tempo que lle leva este labor a un grupo de traballadores ao longo dalgúns repartos. Dun xeito parecido, `EVAC_TIME` significa o promedio de tempo que se tarda en evacuar un destino e volver a colocar unha caixa baleira debaixo del. Variables como `MOVE_TIME_DEST`, `MOVE_TIME_GROUP` ou `MOVE_TIME_IPOST` indican o tempo que toma un traballador para desprazarse entre dous destinos, grupos (de destinos agrupados en seccións) ou postos de indución, respectivamente. Novamente estes datos non son tan inmediatos, se ben se pode calcular un promedio mediante a experiencia empírica, e son necesarios para simular o factor humano nun ordenador.

Empregando os tipos de datos que se aprecian na Figura 5.1, así como unha asignación de traba-

sinalar unha bandexa de referencia, dicindo que esa 'bandexa 1' ocupa a 'posición 1', de modo que no seguinte paso de tempo a 'bandexa 1' ocupará a 'posición 2'.

lladores a roles de traballo, unha función crea un obxecto de *Python* representativo do clasificador que se vai ensaiar, para logo efectuarse a avaliación de resultados mediante unha función simuladora.

5.1.2. Uso de obxectos

Como se sinalou no primeiro capítulo de preliminares, un obxecto en *Python* é un constructo que presenta atributos e métodos. Estes obxectos agrúpanse en clases, de modo que definir unha clase en *Python* significa indicar que atributos e que métodos serán os que presente cada obxecto desa clase. Os atributos son variables que se crean á vez que o obxecto e se enlazan a el. Por exemplo, se `Laura` for un obxecto da clase `Empregado`, algúns posibles atributos que podería ter serían a data de incorporación á empresa ou o seu cargo na mesma. Os métodos, doutra banda, son funcións que modifican os valores dos atributos. Seguindo co exemplo anterior, a clase `Empregado` podería ter definido o método `Ascenso`, que cos debidos argumentos, modificaría o cargo e/ou o salario de dita persoa.

Este anecdótico exemplo permite comprender a gran utilidade que presentan os obxectos para este simulador. Cada traballador pode ser un obxecto da súa respectiva clase, ou cada destino, ou cada tenda, de modo que se poidan crear e modificar máis cómoda e eficientemente.

En efecto, a práctica totalidade das variables coas que se traballa no simulador, non sendo os datos de entrada vistos na sección anterior, serán obxectos. Polo tanto, adicarase este espazo a describir superficialmente as distintas clases cuxos obxectos rexen o comportamento do clasificador na simulación.

Comezando polos traballadores, e como resulta obvio, serán obxectos dunha de catro posibles clases, cuxas características no simulador son listadas a continuación.

- `Inductor`
  - **Atributos**: posto de indución, tempo que tarda en abrir unha caixa ou inducir unha prenda, indicadores de que tarefa está a realizar nese momento e contadores (de qué caixas están ou non abertas, por exemplo).
  - **Métodos**: abrir unha caixa (`open`), inducir unha prenda de roupa (`induce`), e eliminar unha caixa tras colocar todo o seu contido (`remove`).
- `Waterspider`
  - **Atributos**: en que posto de indución se atopa, os tempos que lle toma abrir unha caixa ou cambiar de posto e contadores e indicadores das tarefas que está a realizar.
  - **Métodos**: abrir unha caixa (`open`), escoller a que novo posto de indución se desprazará (`select_ipost`) e o desprazamento en si (`move`).
- `Packer`
  - **Atributos**: lista de destinos que ten asignados, tempo que lle toma a evacuación, en que destino se atopa actualmente e outros contadores ou indicadores sobre as actividades que realiza.
  - **Métodos**: evacuar un destino (`evacuate`), escoller a que novo destino dirixirse (`set_target_destination`) e desprazarse cara el (`move`).
- `Varios`
  - **Atributos**: os mesmos ca clase `Packer`, agás a lista de destinos asignados, que aquí non existe.
  - **Métodos**: os mesmos cos da clase `Packer`.

Ademais dos traballadores, os demais elementos do clasificador constitúense en clases de obxectos. As bandexas son obxectos da clase `Tray`, dado que se precisa coñecer, en cada momento, a posición na que están, a prenda que conteñen (en caso de habela) e en que xute debería caer. A actualización constante destas variables tamén é máis doada definindo métodos como `fill`, `next_position` ou `empty`. Cada destino é tamén un obxecto da clase `Destination`, que contén, ademais dos seus atributos identificativos[5] contadores de volume ocupado ou indicadores de se hai algunha persoa diante del nese momento (ademais dos métodos que regulan estas tarefas). Mesmo para contabilizar a demanda, tamén son moi útiles os obxectos: definindo unha clase `TSA` é posible coñecer a demanda que nun momento determinado resta por satisfacer, e ila actualizando comodamente segundo van caendo obxectos nos destinos que serven a dita tenda. Tamén as caixas que chegan do almacén son obxectos, pois así é máis doado asignalas aos postos de inducir e contabilizar cando se induciron todas as súas prendas.

Con todo, hai obxectos que manexa o simulador, que vertebran o enteiro funcionamento do mesmo e que son posiblemente os de maior importancia: os da clase `Sorter`. Na Figura 5.2 pode apreciarse como esta clase só presenta atributos. Ademais da asignación de traballadores que se pretende ensaiar (dada polos catro primeiros atributos), as demais variables son diccionarios, o contido dos cales serán todos os obxectos definidos para recrear o clasificador segundo esa asignación de traballadores. Por exemplo, o atributo `destination_dict` ten a función de almacenar todos os obxectos representativos dos destnos que comprende o *sorter* a representar.

```
1  class Sorter:
2    def __init__(self, n_inductors, n_waterspiders, n_packers, n_varios):
3      # Workers per role
4      self.n_inductors    = n_inductors
5      self.n_waterspiders = n_waterspiders
6      self.n_packers      = n_packers
7      self.n_varios       = n_varios
8      # Dictionaries
9      self.incoming_box_dict  = {}
10     self.inductor_dict      = {}
11     self.waterspider_dict   = {}
12     self.packer_dict        = {}
13     self.varios_dict        = {}
14     self.destination_dict   = {}
15     self.tsa_dict           = {}
16     self.tray_dict          = {}
17     # List of not-delivered incoming_box
18     self.incoming_box_list  = []
```

Figura 5.2: Deifinición da clase `Sorter` no simulador.

A importancia desta clase radica en que, tal e como está definida, pode construírse unha función simuladora tomando como argumento só un obxecto da clase `Sorter`, pois contén todos os elementos que se precisan para imitar ao clasificador real e ofrecer un tempo de cumprimento da demanda. Como se pode extraer desta explicación, unha vez dados os datos de entrada xa vistos na Figura 5.1, e obtidas todos os posibles candidatos a solución [6], unha función `make_sorter` será a encargada de crear un obxecto desta clase, que servirá como dato de entrada para a función simuladora.

Como se pode extraer deste apartado, basear a programación do simulador en obxectos permitirá unha cómoda escritura do código que represente os diferentes procesos do simulador. Á hora da verdade, o programa debe comprender unha serie de procesos que se deben repetir constantemente, pero afectando a diferentes elementos que comparten características (por exemplo, a indución sempre se produce por parte de indutores e afecta a bandexas), polo que a definición de clases de obxectos é fundamental. Tamén permite unha gran modularidade no código, pois facilita levar a cabo unha gran

[5] Posición incluíndo lado, grupo e sección; a que TSA serve ou a súa capacidade e marxe de erro superior e inferior.
[6] Mediante a función `get_all_assignments` mostrada na Figura 4.1.

variedade de actualizacións ou melloras do programa sen ter que rescribir grandes partes do código máis alá das estritamente afectadas.

5.1.3. Función simuladora

A parte sen dúbida máis importante do simulador é aquela que, dado un *sorter* coas características obtidas dos datos máis a asignación de traballadores que se pretende estudar, realice todos os procesos coma un clasificador real e ofreza un tempo de cumprimento da demanda. Non en van, a función que a tal fin se define comprende máis de 400 liñas de código[7]. A maiores tamén existe un ficheiro con funcións auxiliares ás que o simulador recorre frecuentemente. Por exemplo, atópanse aquí as que permiten bloquear todos os destinos da mesma sección que aquel cuxa caixa se acaba de encher (e tamén desbloquealos ao baleirarse), ou aquela que contén o algoritmo de Round-Robin que imita o comportamento das fotocélulas (é dicir, que asigna a cada prenda un xute no que debe caer).

A explicación, como se dixo ao inicio do capítulo, será superficial, limitándose a dar unha descrición xeral dos procesos que se realizan, ordenalos e estruturalos nos bucles ou disxuncións en que se poidan atopar. Tomarase apoio nas Figuras 5.3, 5.4 e 5.5, onde se organiza graficamente toda esta información.

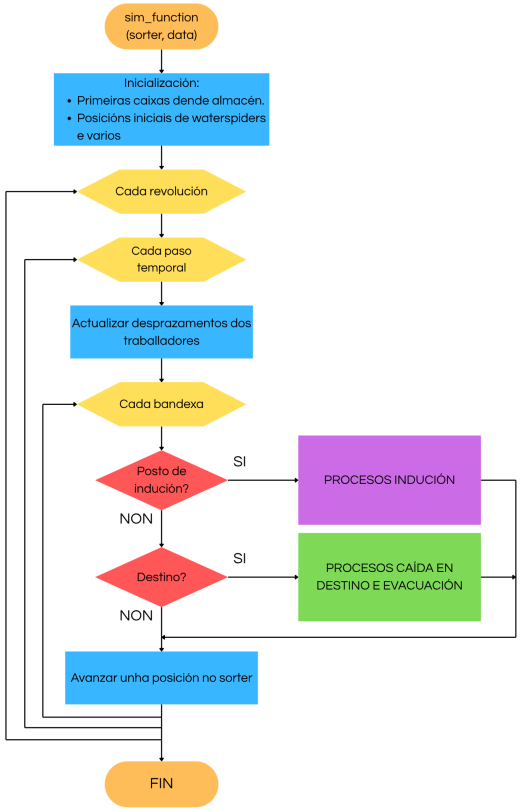


Figura 5.3: Diagrama de fluxo simplificado da función simuladora. Elaboración propia.

A simple vista, non son tantas as diferenzas entre a forma do diagrama da Figura 5.3 e a Figura 4.4 vista no capítulo 4. En efecto, a estrutura a trazo groso é similar, creando bucles para o conxunto das revolucións, o dos pasos de tempo que se suceden en cada volta, e finalmente o listado das bandexas, cuxo estado se debe avaliar para cada intre temporal. Non obstante, xa aquí se aprecian dúas grandes

[7] Incluíndo iso si, comentarios e separacións entre bloques de código.

diferenzas. Primeiramente, unha das evolucións do simulador respecto da súa versión anterior é a inclusión do proceso de apertura e lectura das caixas procedentes de almacén. Por iso, o primeiro paso de toda avaliación dunha asignación é enviar as primeiras caixas a cada posto de indución, de modo que ao iniciar o reparto as primeiras tarefas sexan precisamente as de abrir estas caixas. A outra gran adición é que nesta versión tamén se intenta imitar o movemento físico das persoas cando deben cambiar de ubicación, como é o caso dos *packers* e varios, que deben transitar entre diferentes xutes; ou o dos *waterspiders*, que se moven entre postos de indución. Neste caso, a primeira acción dealizada en cada paso temporal é a actualización dos atributos referidos ao movemento, así como a comprobación de se ditos traballadores rematan o desprazamento ou o continuarán na seguinte iteración.

Outro punto de continuísmo respecto do visto no capítulo anterior é que, para cada bandexa, a comprobación que se realiza é se esta se localiza ou non fronte a un posto de indución, e se se atopa ou non sobre un destino. Ambas as dúas opcións son mutuamente excluíntes. De atoparse nun posto de indución, realizaranse unha serie de pasos que, na Figura 5.3 se engloban nun cadro morado, mentres que de estar a bandexa sobre un destino, procederase con aquelas accións que se simplifican enmarcándoas no cadro verde. Finalmente, sexa cal for a súa situación, a última acción sempre consiste en que a bandexa avance unha posición.

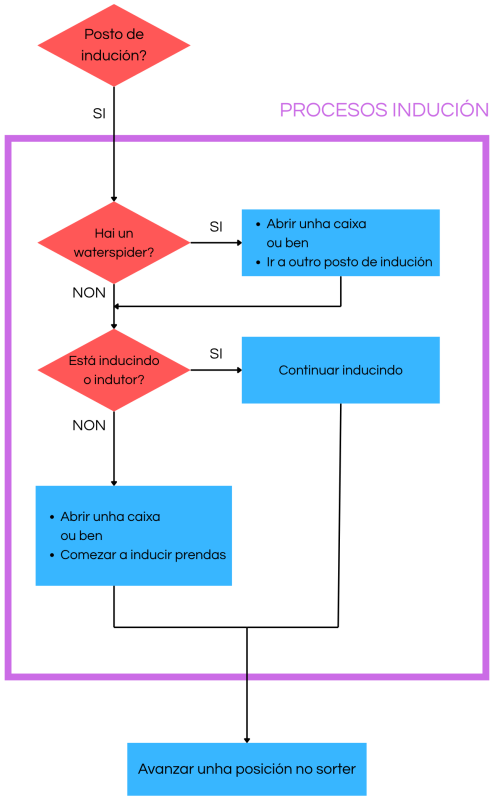


Figura 5.4: Diagrama de fluxo simplificado sobre os procesos de indución da Figura 5.3. Elaboración propia.

Na Figura 5.4 móstranse as decisións que se desatan se ao comprobar o estado dunha bandexa, esta se atopa nun posto de indución. O primiero paso é comprobar se hai un *waterspider* en dita posición, pois en caso afirmativo poderá ou ben abrir[8] unha caixa, ou ben dirixirse a un posto de indución

[8]Enténdese que 'abrir' comprende os procesos de comezar a apertura dunha caixa, continuala ou rematala.

diferente (se no actual non ten máis traballo que realizar). Sexa cal for o caso, debe comprobarse tamén se o indutor está inducindo, vala a redundancia, o cal significa que se comezaran a colocar prendas dunha caixa e aínda non se rematou. En tal caso, só cabe continuar con dito proceso. Se non, pode ser que se está a abrir unha caixa, ou que no intre anterior rematou de inducir ou abrir outra caixa. Neste caso avalíase se se debe comezar a inducir as prendas dunha nova caixa, ou a abrila se todas están pechadas. Finalmente, a consecuencia de calquera das dúas opcións será que a bandexa avance unha posición.

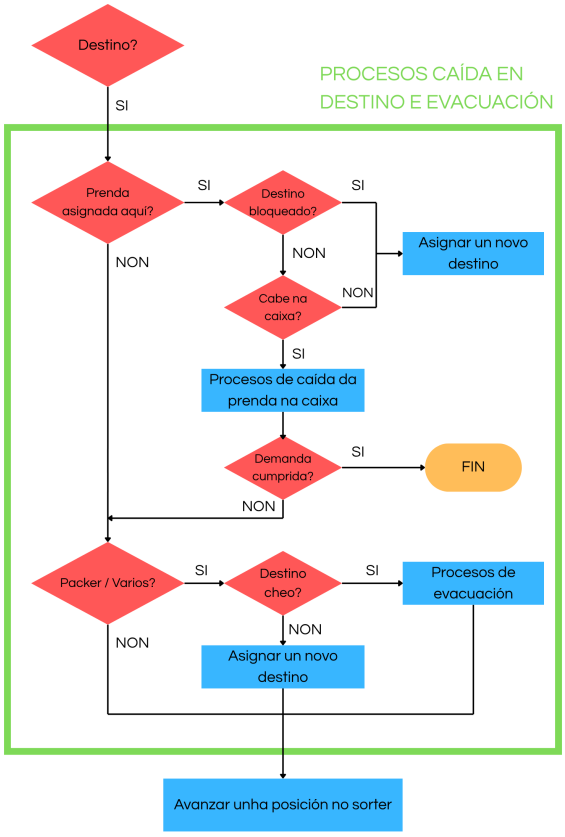


Figura 5.5: Diagrama de fluxo simplificado sobre os procesos de caída en destino e evacuación da Figura 5.3. Elaboración propia.

No caso da indución, os únicos axentes involucrados eran os traballadores[9], as caixas entrantes e as bandexas, co cal o procedemento se volvía relativamente sinxelo, como se pode apreciar na Figura 5.4. Non obstante, a Figura 5.5 mostra que os procesos de caída das prendas en destino, e a evacuación dos mesmos, é moito máis complicado. Neste caso nos axentes involucrados deben diferenciarse os traballadores, os destinos, as bandexas e as tendas (estas últimas para a contabilización da demanda). Ademais, a existencia de mecánicas como a asignación de novos xutes ás prendas, ou o bloqueo dos destinos da mesma sección ca aquel cuxa caixa se enche, volven máis complexo estes procesos. Tal é así que, para maior comprensión, o diagrama está aínda máis simplificado cos outros esquemas.

Como ben se aprecia no diagrama da Figura 5.5, e de modo similar ao acontecido coa indución, deben levarse a cabo dous bloques de accións diferentes, unha a continuación da outra. Se a bandexa se atopa sobre un destino, o primeiro é sempre combrobar se esa bandexa contén un produto e se tiña asignada esta posición para caer. En tal caso, tamén se precisa saber se é posible a caída, é dicir, se

[9]No código, de feito, os indutores e os seus postos de indución tratábanse en conxunto.

o destino non está bloqueado e se a súa caixa pode acoller a prenda sen sobrepasar o seu límite. Se a resposta a algunha destas cuestións é negativa, debe asignarse a esa bandexa un novo destino ao que ir, mentres que se se dan ambas as dúas condicións, procederase á caída e correspondente actualización das variables. Para rematar este bloque, compróbase se queda satisfeita toda a demanda, en cuxo caso remataría a simulación do escenario que se estaba a avaliar.

O seguinte paso é comprobar se nese momento hai algún traballador, sexa *packer* ou varios, que se atope fronte á caixa do dito destino. En tal caso comprobarase se a caixa se enchiu ou segue chea, o cal indicaría que debe iniciarse ou continuarse a acción de evacuala e substituíla por unha nova caixa valeira. En caso de non estar chea, debe comprobarse a que nova ubicación se moverá esta persoa para continuar co seu traballo. Como non, a última acción a realizar sempre será indicarlle á bandexa que debe avanzar unha posición, para continuar coa simulación correctamente.

5.1.4. Programa principal

Finalmente, todo o visto neste capítulo unirase nun programa principal `main.py`, representado esquematicamente na Figura 5.6, e que constitúe o esquelete do simulador.

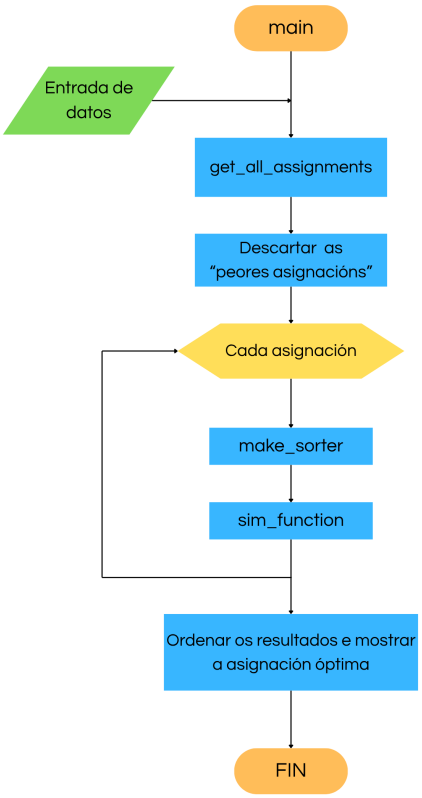


Figura 5.6: Diagrama de fluxo do programa principal que coordina e executa o simulador. Elaboración propia.

Como se pode ver na dita Figura 5.6, e tras ler os datos (cuxa estrutura puido observarse na Figura 5.1), o primeiro paso é obter a lista das posibles asignacións que avaliará o modelo. Coa función `get_all_assignments` (Figura 4.1) calcúlanse todas as posibilidades, das cales se descartará aquelas que de antemán se considere que terán un rendemento pésimo. Para cada un dos candidatos, debe

crearse o correspondente obxecto da clase `Sorter`, como se comentou nun apartado anterio, e sobre el aplícase a función simuladora `sim_function` (Figura 5.3). Unha vez comprobadas todas as opcións só queda ordenar os seus tempos de cumprimento de demanda e devolver por pantalla a asignación óptima de traballadores aos catro roles de traballo.

Antes de pasar ao seguinte apartado, cómpre facer un pequeno comentario indicando que tipos de procesos dos vistos no programa son independentes entre si, de modo que se poderían paralelizar para aforrar espacio. Deles, o máis evidente é a avaliación de cada posible candidato a asignación óptima. Cada posibilidade implica a creación dun obxecto `Sorter` diferente, a partir do cal se convoca á función simuladora, e polo tanto non hai ningún impedimento a realizar varias avaliacións ao mesmo tempo.

Agora ben, hai un nivel máis onde as accións son independentes e tamén sería posible este procesamento en paralelo. Dentro dunha mesma simulación, aqueles pasos que corresponden a intres temporais diferentes dependen forzosamente dos tempos anteriores, polo que se deben executar secuencialmente. Porén, as comprobacións que en cada momento concreto se realizan sobre as diferentes bandexas son independentes, polo que tamén se poderían paralelizar. Neste traballo só se introducirá a paralelización para o primeiro dos casos mencionados, pero aínda así non se esquece que existiría esta posibilidade sen máis que realizar pequenos axustes no código.

5.2. Execución nunha instancia de tamaño realista

Unha vez descrito o simulador, precísase comprobar se serve ao propósito para o cal foi creado: ofrecer solucións en tempos razoables. Isto tradúcese en que debe avaliarse o tempo que toma, para unha instancia de maior tamaño, avaliar todas as asignacións candidatas, a fin de mostrar a mellor.

Con isto en mente, o primeiro é dar debido contexto sobre os datos de entrada cos que se traballará. O ficheiro de datos concreto, que resulta nun dicionario coma o visto na Figura 5.1, non se mostrará aquí, mais pode atoparse no Apéndice A.1. Por resumir o seu contido, o número total de traballadores será 30, e o de postos de indución, 12. O clasificador presentará 1500 bandexas e 288 destinos, uns valores realistas segundo se indicou por parte do titor da empresa. Finalmente, demanda sumará 20000 prendas en total, de dous tamaños diferentes, repartidas en 15 TSAs e 10 SKUs. Un reparto realista ben pode chegar a presentar unha demanda dez veces superior a esta, mais debe terse en conta que a execución se realizará en local no ordenador portátil facilitado pola empresa. Polo tanto, unha proba en local cunha décima parte da demanda real non supón un problema para ensaiar o prototipo dunha ferramenta que logo se podería executar en servidores que exceden con creces esta diferenza de dez a un.

Durante a execución, o ordenador ensaiará as asignacións de dez en dez, empregando librarías de *Python* como `concurrent.futures`, para realizar os pasos en paralelo; `os`, pois estableceuse o número de procesos simultáneos en función da capacidade do procesador; e `uuid`, para etiquetar correctamente cada unha destas pólas. O motivo de empregar a paralelización xa se explicou, pero neste apartado poderase comprobar na práctica. Ao executar o algoritmo as primeiras veces, apreciábanse tempos de entre 3 e 7 minutos para resolver cada asignación, e como o obxectivo era avaliar case un centenar delas, non era unha posibilidade continuar con esa vía. Non obstante, ao implementar este cálculo paralelo, podíanse comprobar 10 asignacións en menos de 20 minutos, o cal reducía considerablemente o tempo total.

Tamén é importante mencionar que, se ben para 20 traballadores e e 12 postos de indución habería 166 posibilidades a estudar [10], só se estudan 87 deses escenarios. Estableceuse como filtro que só se estudarían aquelas asignacións que tivesen un número de indutores maior ou igual ca 5, e onde a suma dos *packers* e os varios non fose inferior a 8. É dicir, estase a descartar aquelas asignacións que, por ter moi poucas persoas en indución ou en destino, provocarían escenarios a priori pésimos. Estes consumirían, non obstante, moito tempo de cálculo, o cal se desexa recortar, co que de aquí procede esta decisión.

[10] Pode comprobarse na Figura 4.1.

Para levar un rexisto dos tempos de execución, e tendo en conta que se resolven varios escenarios á vez, optouse por rexistrar os tempos que toma procesar cada revolución de cada asignación. Esta información pode resultar de utilidade para apreciar en que partes do reparto hai un maior esforzo do ordenador, ao ter que procesar máis prendas e/ou máis accións dos axentes implicados. Por exemplo, dado un escenario, as primeiras revolucións son as máis rápidas de calcular dado que case non haberá evacuacións. De igual modo, dende o punto en que se inducen todas as prendas, o código deixa de ter que executar o bloque de accións correspondente. Empregando R para ler a saída dos datos, obtense unha táboa coa forma que se mostra na Figura 5.7. A columna `Assignment` consta dunha lista de catro elementos que se corresponden, respectivamente, co número de indutores, *waterspiders*, *packers* e varios.

```
> datos = read.csv(file = "execution_times_20_data_dict_3_assignments_
    all.csv", header = TRUE, sep = ",", dec = ".")

> head(datos)
        Assignment         Revolution     Time
1       [[7, 0, 10, 3]]    1              23.31055
2       [[7, 0, 10, 3]]    2              15.66593
3       [[7, 0, 10, 3]]    3              15.85636
4       [[7, 0, 10, 3]]    4              15.74654
5       [[7, 0, 10, 3]]    5              17.56301
6       [[7, 0, 10, 3]]    6              17.17558
```

Figura 5.7: Saída de datos da execución do algoritmo nun `.csv`, lida en R.

Para interpretar correctamente estes datos, e todos aqueles que destes se deriven, debe poñer un adecuado contexto sobre como se mide o tempo. Ao paralelizárense os procesos, neste caso comprobando 10 asignacións á vez, debe entenderse que cada unha procésase cunha décima parte da potencia. Para cada asignación mídense os tempos que toman as súas revolucións, polo que se cada revolución tomase 30s, o correcto sería concluír que 10 revolucións se procesan en 30s, e non en 300s. Isto xustifica por que as medicións de tempo son aparentemente tan elevadas, dado que na práctica o axeitado sería considerar que son dez veces menores. Por este motivo, as representacións gráficas sobre estes datos puntualizan o feito de que se realizan con paralelización.

```
> attach(datos)
# Funcion summary()
> summary(Time)
Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
9.096   25.881   26.785   43.174   27.755 1974.628
# Outros cuantis de interese
> quantile(Time, probs = c(0.95, 0.96, 0.97, 0.98, 0.99))
95%       96%       97%       98%       99%
30.02068  30.53698  31.46489  34.16217 880.56411
```

Figura 5.8: Media, cuartís e outros cuantís de interese para os tempos de execución de cada revolución (con paralelización).

A información máis inmediata que se pode obter desta saída de datos son a media e os cuartís do tempo que toma calcular cada revolución. Na Figura 5.8 móstranse estes valores, obtidos mediante a función `summary` de R. O tempo que toma considerar cada asignación, como se pode ver no primeiro

e terceiro cuartil, está moi contido agás algúns casos extremos. O mesmo cadro de código inclúe tamén os cuantís por centésimas dende 0,95 ata 0,99, onde se pode apreciar que o 98 % das revolucións logran procesarse en menos de 35 segundos. Esta presenza de tempos inusualmente altos, por ser unha proporción tan reducida, pode deberse a algunha asignación na que se produza un certo 'pescozo de botella' na parte das caídas, que é a que entraña unha maior cantidade de comprobacións e código a executar. Con todo, por infrecuente, este escenario non resulta preocupante. Omítese a valoración do tempo medio precisamente por estar tan influenciado por estes valores extremos que, sendo poucos, son moi altos.

```
> library(dplyr)
> # Definicions
> Time_per_assignment <- datos %>%
+   group_by(Assignment) %>%
+   summarise(Total_Time_Each_Assignment = sum(Time, na.rm = TRUE))
> Max_revolution_per_assignment <- datos %>%
+   group_by(Assignment) %>%
+   summarise(Ending_Revolution = max(Revolution, na.rm = TRUE))
> # Funcion summary()
> summary(Time_per_assignment)
Assignment         Total_Time_Each_Assignment
Length:87           Min.   :1206
Class :character    1st Qu.:1728
Mode  :character    Median :1982
                    Mean   :2209
                    3rd Qu.:2492
                    Max.   :3918
> summary(Max_revolution_per_assignment)
Assignment         Ending_Revolution
Length:87           Min.   :26.00
Class :character    1st Qu.:34.00
Mode  :character    Median :38.00
                    Mean   :51.16
                    3rd Qu.:67.00
                    Max.   :90.00
# Mellores candidatos
> Max_revolution_per_assignment = Max_revolution_per_assignment[order(Max_revolution_
    per_assignment$Ending_Revolution),]
> head(Max_revolution_per_assignment)
# A tibble: 6    2
Assignment     Ending_Revolution
<chr>                      <int>
1 [[7, 2, 6, 5]]              26
2 [[7, 3, 6, 4]]              27
3 [[8, 0, 8, 4]]              27
4 [[6, 3, 6, 5]]              28
5 [[7, 0, 8, 5]]              28
6 [[8, 1, 6, 5]]              28
```

Figura 5.9: Media e cuartís dos tempos totais que toma resolver cada asignación (con paralelización), e da revolución en que se satisfai a súa demanda.

Do mesmo modo en que se estudaron os tempos de excución para cada revolución, pódense estudar para resolver toda a asignación, como se mostra na Figura 5.9. Ollando a saída de `Time_per_assignment` atopamos uns tempos de execución mellor distribuídos ca aqueles vistos na Figura 5.8. Isto débese a que, como tamén se observa na saída de `Max_revolution_per_assignment`, as diferentes asignacións estudadas non se concentran tanto en termos de tempo de reparto (moi relacionado co tempo de execución). As medias voltan a ser arrastradas polos valores máis altos, se ben isto é máis suave no caso dos tempos de procesamento. Como cabería esperar, as asignacións presentan un amplo abano

nas revolucións que toman, estando a metade delas por debaixo das 38 voltas, pero chegando a algún caso pésimo que acada o límite, establecido nas 90 revolucións. Os tempos de remate de cada candidato poden visualizarse graficamente no histograma da Figura 5.10. Como último comentario sobre estes cadros, ao final da Figura 5.9 móstranse as seis mellores asignacións, para a instancia concreta que se analiza, a ollos do simulador.

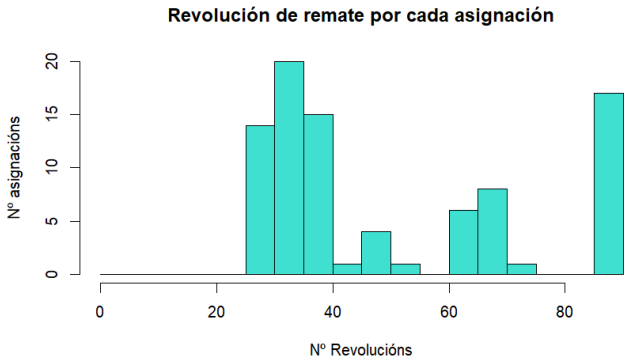


Figura 5.10: Histograma do número de revolucións que toma satisfacer a demanda para cada asignación.

Para pechar esta execución e os comentarios ao respecto dela, pode representarse nunha gráfica o tempo de cómputo, medio e mediano, de cada revolución numerada ao longo das simulacións. Isto e, para a revolución número 5, por exemplo, estaría a figurar a media ou mediana de todos os tempos de cálculo da revolución número 5 de cada asignación. Dita gráfica atópase na Figura 5.11 e permite observar como, por exemplo, as primeiras voltas do clasificador sempre presentan tempos de execución moi reducidos. A partires deste punto, debe de ollarse só para o gráfico de medianas, pois as medias están demasiado influenciadas por tempos de execución atípicos[11]. Interesados pois na segunda imaxe aprécianse decensos preto daquelas revolucións onde máis asignacións logran cumprir a demanda (o cal se fai máis evidente ollando o histograma da Figura 5.10), do cal podemos intuír que os cálculos das últimas revolucións de cada asignación tamén presentan tempos menores. Finalmente, obsérvase que nas últimas revolucións os valores son moi elevados, o cal ten doada explicación. Debe lembrarse que se fixou un máximo de voltas, 90, a partir das cales o simulador deixa de ensaiar unha asignación por considerala inviable. É dicir, os últimos datos das gráficas non se corresponden con revolucións finais de ningún candidato, senón intermedias.

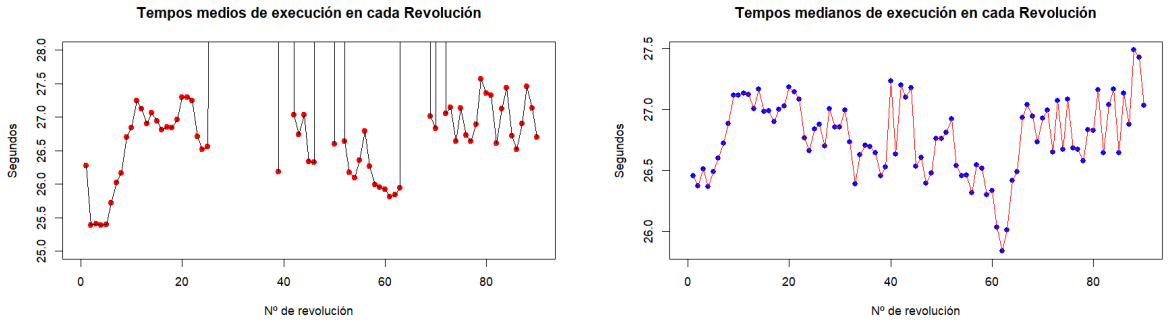


Figura 5.11: Gráfica de tempos medios e medianos de execución de cada revolución ordenada das asignacións.

[11] Xa comentado e ilustrado na Figura 5.8.

En resumo, o simulador presenta tempos adecuados para a execución desta instancia, se se teñen en conta as circunstancias que concorren: realizada en local nun ordenador portátil, as técnicas de programación e eficiencia do código e demais. A conclusión principal a extraer desta proba é que este simulador supón un punto de partida prometedor, dende o cal chegar a construír unha ferramente que permita, en tempos de execución aceptables a nivel comercial, ofrecer unha solución que se poida por en práctica no clasificador real.

5.3. Vías de mellora do simulador e dos tempos de execución

O apartado anterior mostra que, de xeito evidente, o simulador é unha ferramenta de resolución do problema de asignación moi superior ao modelo de optimización clásica. Este último non é sequera capaz de ofrecer resposta algunha a instancias do problema cun tamaño nin dez veces menor ca aquel que se acaba de analizar.

Con todo, o código executado non é outra cousa ca un prototipo, que deberá mellorarse de varias maneiras, a fin de obter un código que se poida incluír nun produto usable para aquelas persoas ás que está dirixido.

O primeiro que se debería acometer son as melloras na programación. Dunha banda, cómpre refactorizar o código, é dicir, rescribilo para que realice as mesmas funcións, pero dun xeito máis eficiente a nivel computacional. Isto traduciríase en bucles con moitas menos iteracións ou un gasto de memoria menor para almacenar variables, por exemplo. Ademais, deberíase valorar directamente o cambio de linguaxe do simulador, máxime se se quere incluír nun produto de maior tamaño, o cal empregue outra linguaxe de programación distinta.

Outra das melloras consiste en completar o código con utilidades que neste estadío aínda non presenta, pero que serán importantes para poder servir ao maior número de clasificadores posible. A máis inmediata supón crear unha vía de entrada para o formato dos datos da demanda orixinais, aqueles que manexa a empresa, de modo que se traduzan automáticamente ás variables empregadas no simulador. Máis alá disto, debería aumentarse a capacidade de configuración do tipo de clasificador sobre o que traballa o simulador. Actualmente, o obxecto `Sorter` que constrúe o programa é sempre simétrico, no sentido en que ambos os dous lados teñen o mesmo número de destinos, organizados no mesmo número de seccións e grupos. Isto non acontece así sempre, polo que se debería habilitar unha maior personalización neste aspecto.

Ademais disto, cando se mostrou a entrada de datos na Figura 5.1, non se mencionou a súa última variable, chamada `MULTISECTOR`. Non se emprega na versión actual do simulador, pero incluírase de cara a futuras melloras do mesmo. Á hora de asignar os destinos aos diferentes *packers* que participan do reparto, existen dúas formas de facelo. A primeira consiste en asignar un número de sectores a cada un, de modo que todos os destinos do mesmo sector están en mans da mesma persoa. A outra opción consiste en que dúas persoas poidan compartir un mesmo sector. Esta variable viña a se incluír para nun futuro dar pé a que, segundo o valor da mesma, se realizase un ou outro reparto de destinos.

Outro aspecto en que se pode mellorar a rapidez do simulador é a selección inicial de candidatos a asignación óptima, como se pode observar na Figura 5.10, a proba da sección anterior avaliaba moitos candidatos que excedían o número máximo de revolucións que contemplaba o algoritmo. A través de repetidas execucións cun amplo abano de instancias diferentes, así como da evidencia experimental en casos reais, sería recomendable afinar máis os criterios de exclusión de asignacións, de modo que se avalíen só aquelas que teñas maiores opcións (ou no seu defecto, que non entre ningunha das que producen tempos de cumprimento da demanda excesivos).

Cando se presentou o modelo no Capítulo 3, adicouse un apartado á posibilidade de cambiar de xeito dinámico a asignación de traballadores. É dicir, establecer uns puntos temporais en que puidese cambiar esta configuración para adaptarse á situación do reparto. O algoritmo, tal e como se plantexa, pode adaptarse facilmente a esta situación. Precísase, en primeiro lugar, poder rexistrar unha foto fixa do clasificador[12] no momento en que se pretende realizar este cambio, dado que esas serán as condicións

[12] É dicir, coñecer as posicións das bandexas e o seu contido, a dos traballadores, e finalmente o estado dos destinos.

iniciais coas que se estudarán novamente as asignacións posibles. Logo cabería implementar o modo en que o usuario pode seleccionar os puntos de corte, entre outros bloques de código. Non obstante, este cambio dinámico de asignación é, a través da mellora do simulador actual, unha posibilidade a considerar.

Finalmente, tamén se debe destacar a gran diferenza que supón unha execución en local nun ordenador portátil da que significaría o uso das capacidades dun servidor. Neste caso si sería viable chegar a procesar repartos completos co simulador, incluíndo toda a demanda. A nivel de código, debería implementarse unha certa personalización sobre o grao de paralelización que se desexa. Así, poderíase traballar cun número maior ou menor de procesos á vez en función do tamaño da demanda, ou do tipo de servidor, ou mesmo da fracción da capacidade de cálculo do mesmo que se desexa acaparar. Unha vez despregado nunha máquina máis potente, ademais das melloras anteriormente mencionadas, é de prever que os tempos de execución se adecúen moito máis aos estándares requeridos para o seu uso na realidade.

Capítulo 6

Conclusión

Ao longo deste escrito púxose en coñecemento do lector a información e contexto necesario para entender o problema que se pretendía resolver. Xa nesta descrición, á que se adicou case un capítulo enteiro, se deixavan entrever as dificultades que xurdirían á hora de traducir este tipo de clasificador a un modelo de optimización clásico. Este *sorter*, ou no seu defecto, o tipo de problema que sobre el se quería abordar, é moi particular: a cantidade de procesos que comprende, a mestura da automatización co factor humano... Isto acentuouse cando se realizou a revisión bibliográfica, pois neste caso non era viable impoñer as condicións que comparten a meirande parte dos clasificadores da literatura.

Pouco a pouco, íase perfilando un horizonte no que a resolución dun modelo de programación enteira non sería posible para este problema, pois o custo computacional do mesmo excedería o razoable para un ordenador. Con todo, creouse un modelo que, se ben presentaba simplificacións moi fortes nalgunhas áreas, permitía recrear o funcionamento do clasificador e buscar a asignación óptima das persoas a cadanseu rol de traballo. Dunha banda, o número de variables crecía demasiado segundo o facían o tempo e a demanda considerados, e xa para casos moi sinxelos se producían tempos de execución de varias horas. Da outra banda, e tamén responsable do mal rendemento, unha característica particular deste problema[1] levaba a que aparecesen multitude de simetrías que non era viable tentar romper.

Cando se constatou, ademais do anterior, que o número total de asignacións traballadores-roles factibles era, a nivel combinatorio e informático, moi reducido, apareceu a idea de construír un simulador. Tras constatar na sección 4.4, baixo condicións similares, o prometedor que resultaba o simulador en tempos de execución, prodeciuse a crear unha versión máis completa. Esta tentaba incluír todos aqueles aspectos que o modelo debía ignorar, a fin de ser o máis fiel posible ao comportamento do clasificador na realidade.

Acadouse entón un prototipo, que se puxo a proba con instancias dun tamaño moito máis próximo ao real. Se ben os tempos de execución, por obvios motivos, non eran acordes aos estándares comerciais, si que se podían considerar unha boa base sobre a que rematar construíndo un produto funcional. As vías máis inmediatas de avance, como a refactorización do código e o cambio de linguaxe, para integrarse mellor na infraestrutura da empresa e ser máis eficiente, foron descritas ao final do pasado capítulo.

En resumo, neste proxecto comprobouse que a construción dun modelo de optimización, se ben é posible, non resulta eficaz cando o problema é especialmente complexo e non respecta certas condicións, necesarias para conter o número de variables. En contadas ocasións, sendo esta unha delas, simplemente é posible comprobar todas as posibilidades (ou cando menos, unha parte delas a priori mellor).

Tomando este traballo como base dun proxecto maior, podería acadarse un produto sinxelo e viable para resolver o problema de asignación de traballadores a roles. Pero máis alá diso, a construción dun simulador permitiría realizar novos tipos de estudos ou resolver outros problemas, pois permitiría recrear o comportameno do clasificador, incluíndo ao factor humano. Poderían, por exemplo, ensaiarse regras de reparto, ensaiar patróns de rutas para *waterspiders* ou varios, probar diferentes asignacións

[1] Non poder coñecer de antemán a que destino concreto se dirixe cada prenda

de destinos ás tendas, e un longo etcétera. Se ben é certo que o traballo non puido seguir a liña que pretendía nunha orixe, a do modelo, logrouse acadar o cometido do mesmo, ofrecendo un prototipo prometedor e unha perspectiva de construción dunha ferramenta que aporte valor á empresa.

Apéndice A

Fragmentos de código e táboas complementarias

A.1. Datos de entrada para a sección 5.2

```
        N_REVOLUTIONS = 90

        N_WORKERS   = 20

        N_POSITIONS                 = 1500

        N_IPOST = 12
        IPOST_POSITIONS = {1: [12*j + 1 for j in range(6)], 2: [12*j + 750 for j in
            range(6)]}

        # Destination positions
        N_GROUPS_PER_SIDE           = 2
        N_SECTORS_PER_GROUP        = 8
        N_DESTINATIONS_PER_SECTOR  = 9
        N_DESTINATIONS = 2 * N_GROUPS_PER_SIDE * N_SECTORS_PER_GROUP *
            N_DESTINATIONS_PER_SECTOR
        DESTINATION_DISTANCE = 3
        SECTOR_DISTANCE = 3
        GROUP_DISTANCE = 80
        DESTINATION_LIST = list()
        DESTINATION_POSITIONS = list()

        c = 0
        for i in range(1, N_GROUPS_PER_SIDE + 1):
        for j in range(1, N_SECTORS_PER_GROUP + 1):
        for k in range(1, N_DESTINATIONS_PER_SECTOR + 1):
        DESTINATION_LIST.append([1,i,j,k,75+c])
        DESTINATION_POSITIONS.append(75+c)
        c += DESTINATION_DISTANCE
        c += SECTOR_DISTANCE - DESTINATION_DISTANCE
        c += GROUP_DISTANCE - DESTINATION_DISTANCE

        c = 0
        for i in range(1, N_GROUPS_PER_SIDE + 1):
        for j in range(1, N_SECTORS_PER_GROUP + 1):
        for k in range(1, N_DESTINATIONS_PER_SECTOR + 1):
        DESTINATION_LIST.append([2,i,j,k,825+c])
        DESTINATION_POSITIONS.append(825+c)
        c += DESTINATION_DISTANCE
        c += SECTOR_DISTANCE - DESTINATION_DISTANCE
```

```
39          c += GROUP_DISTANCE - DESTINATION_DISTANCE
40          ##
41
42          BOX_LIMIT = 50
43          LOWER_MARGIN = 3
44          UPPER_MARGIN = 1.5
45
46
47          REF = [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 ]
48          TSA = [ 1, 2 , 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
49          VOL = {1: 1, 2:2.5, 3: 1, 4:2.5, 5: 1, 6:2.5, 7: 1, 8:2.5, 9: 1, 10:2.5, 11:
                1, 12:2.5, 13: 1, 14:2.5, 15: 1}
50          ITEMS_PER_INCOMING_BOX = {1:50, 2: 20, 3:50, 4:20, 5:50, 6:20, 7:50, 8:20,
                9:50, 10:20}
51
52          DEMAND_TSA = {
53                  1:{1:130, 2:120, 3: 140, 4: 110, 5: 130, 6: 120, 7: 135, 8: 130, 9:
                        150, 10: 100},
54                  2:{1:130, 2:120, 3: 140, 4: 110, 5: 130, 6: 120, 7: 135, 8: 130, 9:
                        150, 10: 100},
55                  3:{1:130, 2:120, 3: 140, 4: 110, 5: 130, 6: 120, 7: 135, 8: 130, 9:
                        150, 10: 100},
56                  4:{1:130, 2:120, 3: 140, 4: 110, 5: 130, 6: 120, 7: 135, 8: 130, 9:
                        150, 10: 100},
57                  5:{1:130, 2:120, 3: 140, 4: 110, 5: 130, 6: 120, 7: 135, 8: 130, 9:
                        150, 10: 100},
58                  6:{1:130, 2:120, 3: 140, 4: 110, 5: 130, 6: 120, 7: 135, 8: 130, 9:
                        150, 10: 100},
59                  7:{1:130, 2:120, 3: 140, 4: 110, 5: 130, 6: 120, 7: 135, 8: 130, 9:
                        150, 10: 100},
60                  8:{1:130, 2:120, 3: 140, 4: 110, 5: 130, 6: 120, 7: 135, 8: 130, 9:
                        150, 10: 100},
61                  9:{1:130, 2:120, 3: 140, 4: 110, 5: 130, 6: 120, 7: 135, 8: 130, 9:
                        150, 10: 100},
62                  10:{1:130, 2:120, 3: 140, 4: 110, 5: 130, 6: 120, 7: 135, 8: 130, 9:
                        150, 10: 100},
63                  11:{1:130, 2:120, 3: 140, 4: 110, 5: 130, 6: 120, 7: 135, 8: 130, 9:
                        150, 10: 100},
64                  12:{1:130, 2:120, 3: 140, 4: 110, 5: 130, 6: 120, 7: 135, 8: 130, 9:
                        150, 10: 100},
65                  13:{1:130, 2:120, 3: 140, 4: 110, 5: 130, 6: 120, 7: 135, 8: 130, 9:
                        150, 10: 100},
66                  14:{1:130, 2:120, 3: 140, 4: 110, 5: 130, 6: 120, 7: 135, 8: 130, 9:
                        150, 10: 100},
67                  15:{1:130, 2:120, 3: 140, 4: 110, 5: 130, 6: 120, 7: 135, 8: 130, 9:
                        150, 10: 100}
68          }
69
70          TSA_DEST = {1:20, 2: 20, 3: 20, 4: 19, 5:19, 6:19, 7:19, 8:19, 9:19, 10:19,
                11:19, 12:19, 13:19, 14:19, 15:19}
71
72          DEMAND_REF = {}
73          for i in REF:
74          DEMAND_REF[i] = sum(DEMAND_TSA[j][i] for j in DEMAND_TSA)
75
76          OPENING_TIME = 60
77          INDUCTION_TIME = 8
78          EVAC_TIME = 120
79          MOVE_TIME_DEST = 5
80          MOVE_TIME_GROUP = 100
81          MOVE_TIME_IPOST = 20
82
83          MAX_INCOMING_BOX_PER_IPOST = 7
84
85          MULTISECTOR = True
```

```
86
87          data_dict = {
88                  'N_REVOLUTIONS': N_REVOLUTIONS,
89                  'N_WORKERS': N_WORKERS,
90                  'N_POSITIONS': N_POSITIONS,
91                  'N_IPOST': N_IPOST,
92                  'IPOST_POSITIONS': IPOST_POSITIONS,
93                  'N_GROUPS_PER_SIDE': N_GROUPS_PER_SIDE,
94                  'N_SECTORS_PER_GROUP': N_SECTORS_PER_GROUP,
95                  'N_DESTINATIONS_PER_SECTOR': N_DESTINATIONS_PER_SECTOR,
96                  'N_DESTINATIONS': N_DESTINATIONS,
97                  'DESTINATION_DISTANCE': DESTINATION_DISTANCE,
98                  'SECTOR_DISTANCE': SECTOR_DISTANCE,
99                  'GROUP_DISTANCE': GROUP_DISTANCE,
100                 'DESTINATION_LIST': DESTINATION_LIST,
101                 'DESTINATION_POSITIONS': DESTINATION_POSITIONS,
102                 'BOX_LIMIT': BOX_LIMIT,
103                 'LOWER_MARGIN': LOWER_MARGIN,
104                 'UPPER_MARGIN': UPPER_MARGIN,
105                 'ITEMS_PER_INCOMING_BOX': ITEMS_PER_INCOMING_BOX,
106                 'REF': REF,
107                 'TSA': TSA,
108                 'VOL': VOL,
109                 'DEMAND_TSA': DEMAND_TSA,
110                 'DEMAND_REF': DEMAND_REF,
111                 'OPENING_TIME': OPENING_TIME,
112                 'INDUCTION_TIME': INDUCTION_TIME,
113                 'EVAC_TIME': EVAC_TIME,
114                 'MOVE_TIME_DEST': MOVE_TIME_DEST,
115                 'MOVE_TIME_GROUP': MOVE_TIME_GROUP,
116                 'MOVE_TIME_IPOST': MOVE_TIME_IPOST,
117                 'MAX_INCOMING_BOX_PER_IPOST': MAX_INCOMING_BOX_PER_IPOST,
118                 'TSA_DEST': TSA_DEST,
119                 'MULTISECTOR': MULTISECTOR
120         }
```

A.2. Resto de Figuras do traballo. Instancias e táboas comparativas complementarias da sección 4.4

Demand by TSA (each SKU)	Model optimal assignment	Position in simulator ranking	Simulator optimal assignment
10, 10, 10	7,2,0	1	7,2,0
20, 10, 10	7,2,0	1	7,2,0
30, 10, 10	7,2,0	1	7,2,0
30, 20, 10	7,2,0	1	7,2,0
40, 20, 10	7,2,0	1	7,2,0

Demand by TSA (each SKU)	Min. revolutions with model	Rev. of model sol. with simulator	Min. revolutions with simulator
10, 10, 10	4	3	3
20, 10, 10	4	4	4
30, 10, 10	5	5	5
30, 20, 10	6	5	5
40, 20, 10	6	6	6

Demand by TSA (each SKU)	Model execution time	Simulator execution time
10, 10, 10	51.0332	0.2508
20, 10, 10	39.2329	0.3755
30, 10, 10	464.3865	0.7476
30, 20, 10	2904.3784	0.7033
40, 20, 10	1144.8626	0.4705

Figura A.1: Resultados da comparativa ao incrementar asimetricamente a demanda de cada TSA, para todos os SKU por igual.

Demand by TSA (each SKU)	Model optimal assignment	Position in simulator ranking	Simulator optimal assignment
10, 10, 10	7,2,0	1	7,2,0
20, 10, 10	7,2,0	1	7,2,0
10, 20, 10	7,2,0	1	7,2,0
30, 10, 10	7,2,0	1	7,2,0
10, 30, 10	7,2,0	1	7,2,0

Demand by TSA (each SKU)	Min. revolutions with model	Rev. of model sol. with simulator	Min. revolutions with simulator
10, 10, 10	4	3	3
20, 10, 10	4	4	4
10, 20, 10	4	4	4
30, 10, 10	5	5	5
10, 30, 10	5	5	5

Demand by TSA (each SKU)	Model execution time	Simulator execution time
10, 10, 10	55.9893	0.2933
20, 10, 10	40.3427	0.3578
10, 20, 10	376.2209	0.4655
30, 10, 10	1628.1903	0.5279
10, 30, 10	1038.2077	0.4697

Figura A.2: Resultados da comparativa ao incrementar asimétricamente a demanada de cada referencia, en todas as TSAs por igual.

```
data_1 = {None: {
                 'W_TOTAL': {None: 9},
                 'W': {None: range(1, 10)},
                 'S': {None: 100},
                 'IPOST': {None: 7},
                 'REV_TOTAL': {None: 25},
                 'REV': {None: range(1, 26)},
                 'REF': {None: range(1,4)},
                 'REF_TOTAL': {None: 3},
                 'VOL': {1:1, 2:2, 3:1} ,
                 'TSA': {None: range(1,4)},
                 'TSA_TOTAL': {None: 3},
                 'D': {None: range(1,22)},
                 'DEMAND': {(1,1):10, (1,2):10, (1,3):10,
                         (2,1):10, (2,2):10, (2,3):10,
                         (3,1):10, (3,2):10, (3,3):10 },
                 'TSADEST': {1: range(1,8), 2: range(8, 15),
                         3: range(15, 22)},
                 'B': {None: 10},
                 'LEMARGIN': {None: 1},
                 'UEMARGIN': {None: 1},
                 'IAB': {None: 6},
                 'PAB': {None: 3},
                 'WSAB': {None: 1},
                 'VRAB': {None: 4},
}}
```

Figura A.3: Instancia de referencia para as comparativas da Figura A.2 e da Figura A.1.

Nº workers	Model optimal assignment	Position in simulator ranking	Simulator optimal assignment
5	3,2,0	1	3,2,0
8	6,2,0	1	6,2,0
10	8,2,0	1	8,2,0

Nº workers	Min. revolutions with model	Rev. of model sol. with simulator	Min. revolutions with simulator
5	5	5	5
8	4	3	3
10	4	3	3

Nº workers	Model execution time	Simulator execution time
5	30.6112	0.0630
8	25.3609	0.1602
10	157.9836	0.1661

Figura A.4: Resultados da comparativa ao incrementar o número total de traballadores, mantendo os demais parámetros.

```
data_1 = {None: {
                 'W_TOTAL': {None: 5},
                 'W': {None: range(1, 6)},
                 'S': {None: 100},
                 'IPOST': {None: 3},
                 'REV_TOTAL': {None: 25},
                 'REV': {None: range(1, 26)},
                 'REF': {None: range(1,4)},
                 'REF_TOTAL': {None: 3},
                 'VOL': {1:1, 2:2, 3:3} ,
                 'TSA': {None: range(1,4)},
                 'TSA_TOTAL': {None: 3},
                 'D': {None: range(1,13)},
                 'DEMAND': {(1,1):10, (1,2):7, (1,3):3,
                         (2,1):10, (2,2):7, (2,3):3,
                         (3,1):10, (3,2):7, (3,3):3},
                 'TSADEST': {1: range(1,5), 2: range(5, 9), 3:range(9,13)},
                 'B': {None: 7},
                 'LEMARGIN': {None: 1},
                 'UEMARGIN': {None: 1},
                 'IAB': {None: 5},
                 'PAB': {None: 3},
                 'WSAB': {None: 1},
                 'VRAB': {None: 4},
}}
```

Figura A.5: Instancia de referencia para a comparativa da Figura A.4.

Apéndice B

Documentos de traballo

O presente apéndice consta de tres documentos de traballo, nos cales se documentaba o modelo e as súas posibles melloras. Estes documentos eran regularmente actualizados e expostos aos titores académico e de empresa durante a realización das prácticas.

Os mentados documentos reciben os seguintes títulos:

1. ***Modelo Multirreferencia V1.0***: este é o modelo que se expón en detalle neste TFM, que se somete a crítica e no que se basea o posterior deseño dun simulador. Inclúe de xeito esquemático todo o que se pode ver no Capítulo 3.

2. ***Modelo Multirreferencia V1.5***: nova versión na que se engade, respecto do documento anterior, a posibilidade de que en certos intres de tempo se poida escoller unha nova asignación de traballadores a roles. O modelo está completo e sería aplicable no estado en que aparece no documento.

3. ***Propuestas para modelizar la Inducción. Modelo Multirreferencia V1.5***: tomando como base a versión V1.5 do modelo, realízanse dúas propostas para modelizar de xeito máis realista o proceso de indución, isto é, o labor de indutores e *waterspiders*. Non está implementado no modelo, senón que se limita a propoñer en cada caso os parámetros que se engadirían e como se adaptarían algunhas restricións a modo de exemplo.

Modelo Multirreferencia
V 1.0

Ángel Mourelle Abelenda

Supuestos del modelo

- **Tiempo en ciclos o revoluciones**: la 'unidad de tiempo' en este modelo son las revoluciones, es decir, el tiempo que tarda el sorter en volver a una posición determinada tras abandonarla (por ejemplo, con respecto a una bandeja). A lo largo de una revolución cada bandeja pasa por todos los procesos (inducción, caída en destino, etc.). Normalmente es conocido el tiempo que dura una revolución, por lo que bastaría con dividir el tiempo total entre este para obtener el número de revoluciones a introducir como parámetro del problema.
- **Multirreferencia**: los items llevan una referencia asociada que indica el tipo de producto (camiseta blanca, camiseta negra, sudadera roja...).
- **Volumen como magnitud**: el contenido de las cajas en destino, así como la determinación de si están o no llenas, se hará en base a unidades de volumen. Cada referencia lleva asociado un volumen, por lo que cada item contará con su volumen.
- **Los items con igual referencia son iguales**: no se trabaja con los items de forma individual. El modelo solo contemplará en cada paso la cantidad total de items de cada referencia que se inducen, circulan, caen, recirculan, etc.
- **Demanda**: la demanda es finita y consiste en la cantidad de items de cada referencia que solicita cada TSA.
- **Destinos asociados a TSAs**: cada destino tendrá asociada una y solo una TSA. De este modo, la demanda de una TSA se satisfará entre todos los destinos que tienen asociada esa TSA.
- **Inductores en bloque**: de momento se contabilizan los inductores y su tarea como un bloque, y no tanto la suma de individuos con cargas de trabajo diferente. Los *water-spiders* tendrán una influencia reducida en el modelo, y su número dependerá de los puestos de inducción disponibles [1]. La idea es que más adelante se pueda evolucionar el modelo 'de forma modular', añadiendo variables y modificando cómo se calcula la suma de items inducidos en cada revolución.
- **Las referencias se inducen por orden**: los items de la segunda referencia registrada solo se empezarán a inducir cuando se hayan inducido los de la primera referencia, y así sucesivamente.

[1] Si se necesita mayor capacidad inductora que los inductores posibles, será cuando el modelo designe algún *water-spider*.

Parámetros

- W: nº total de trabajadores, que serán asignados a cada rol. Trabajadores $e \in \{1, ..., W\}$
- REV: nº total de revoluciones. Revoluciones $r \in \{1, ..., REV\}$.
- S: nº total de bandejas (*slots*) donde se pueden depositar items. No tendrá mucha utilidad en el problema, más allá de acotar la capacidad de inducción, junto con las recirculaciones, en casos específicos.
- $IPOST$: nº de puestos de inducción, es decir, cota superior al número de inductores para el modelo.
- D: nº de destinos, aquellos en los que pueden caer items en cajas. Destinos $d \in \{1, .., D\}$.
- B: capacidad (en volumen) de cada caja en destino. Lo ideal es que una caja se llene al sumar volumen B (salvando márgenes de error) y entonces se reponga.
- $LEMARGIN$ y $UEMARGIN$: respectivamente, los márgenes de error inferiores y superiores a los vlúmenes de la caja. Podemos añadir el último item en destino a una caja si sobrepasa B en un volumen menor a $UEMARGIN$. Del mismo modo, podremos considerar llena una caja con volumen superior a $B - LEMARGIN$. Como cada referencia puede tener un volumen diferente, estos márgenes permiten evitar los casos en que las cajas no se 'llenen de forma exacta'.
- REF: nº total de referencias. Referencias $i \in \{1, ..., REF\}$.
- VOL_i: para cada $i \in \{1, ..., REF\}$, unidades de volumen que ocupa cada item de la referencia i.
- TSA: nº de TSAs [2], quienes generan la demanda. TSAs $t \in \{1, ..., TSA\}$.
- $DEMAND_{ti}$: demanda total de items de cada referencia i que hace la TSA t.
- $TSADEST_t$: conjunto de destinos que se asignan a cada TSA.
- IAB: nº promedio de items que un inductor es capaz de inducir por revolución.
- PAB: nº de cajas llenas que un *packer* es capaz de reponer durante una revolución.
- $WSAB$: plus de items que puede inducir un inductor si es ayudado por un *water-spider*. Con las asunciones de este modelo (inductores com obloque), podemos también considerar los *water-spiders* como bloque (de menor número que los inductores) y su aportación como una suma a la capacidad de los inductores.
- $VRAB$: análogo a PAB para varios. Se usa un parámetro diferente pues es presumible que si el varios se puede desplazar por todos los destinos, esta tardanza se pueda reflejar en un valor de $VRAB$ menor que el PAB de los *packers*.

De ahora en adelante será habitual el abuso de notación de denotar, por ejemplo, a REV como el conjunto $\{1, ..., REV\}$. Lo mismo aplicará para W, REF, TSA, D.

[2] Tienda Sección Aparte

Variables

- $n^I \in \{1, ..., W\}$: cantidad de inductores.
- $n^{WS} \in \{1, ..., W\}$: cantidad de *water-spiders*.
- $n^P \in \{1, ..., W\}$: cantidad de *packers*.
- $n^{VR} \in \{1, ..., W\}$: cantidad de varios (apoyo a *packers*).
- $hpack \in \mathbb{Z}^+$: mitad de la cantidad de *packers*. Imponemos que esta variable sea entera para que el nº de *packers* sea par.
- $rpack_e$: indicador binario de si el trabajador $e \in \{1, ..., W\}$ es un *packer*.
- $rvar_e$: análogo para varios (rol de apoyo de los *packers*).
- $apack_{de}$: indicador binario de si el destino d está asignado al trabajador (*packer*) e.
- $nind_{ir}$: nº de items que se inducen de la referencia $i \in REF$ en la revolución $r \in REV$.
- $already_{ir}$: indicador binario de si la demanda de la referencia $i \in REF$ ha sido completamente inducida en la revolución $r \in REV$.
- $nfall_{dir}$: nº de items de la referencia i que, en la revlución r, caen en el destino d.
- $itemsinbox_{dir}$: items de la referncia $i \in REF$ que contendrá la caja (tras la caída de items) en destino $d \in D$ en tiempo $r \in REV$.
- $isfull_{dr}$: indicador binario de si la caja en el destino d se llena a lo largo de la revolución r.
- $ifpack_{der}$: indica si en tiempo r el trabajador (*Packer*) $e \in W$ está empaquetando (reponiendo) la caja en el destino d. Se puede hacer el packing de una caja que se llena en tiempo r.
- $ifvar_{der}$: lo análogo para los *Varios* (rol de apoyo de los *Packers*). La diferencia de modelizado se basa en el hecho de que los *Packers* tienen sus destinos asignados, mientras que los *Varios* no tienen esa limitación.
- $npack_{dir}$: nº de items de la referencia i que, cuando se reponga la caja d, se sumarán a la demanda satisfecha.
- $leftov_{ir}$: nº de items de la referencia i en movimiento que no han caído en destino en tiempo r, es decir, que recircularán en el siguiente ciclo $r + 1$.
- $cpletetsa_{tir}$: indicador binario de si la demanda de la TSA t del item de referencia i ha sido satisfecha en la revolución r.
- $final_r$: : variable binaria que nos indicará si toda la demanda de todas las TSA ya se ha cubierto en el ciclo r.

Función objetivo

La idea más básica es cumplir con la demanda en el menor tiempo posible. Una forma de hacer esto es maximizando la suma de indicadores binarios de que la demanda se ha completado, pues para cada revolución valdrán 1 si ya se ha cumplido toda la demanda. Aún mejor sería minimizar la resta del total de revoluciones menos la suma anterior, dado que el resultado sería el número de revoluciones en que se completa la demanda menos uno. Más adelante se verá que el modelo, salvo casos muy concretos, finalizará una revolución más tarde, por lo que no tenemos que sumar esa unidad de tiempo:

$$\text{mín} \quad REV - \sum_{r \in REV} final_r$$

Restricciones

Trabajadores

- La suma total de los trabajadores que ocupan los diferentes roles es W.

$$n^I + n^{WS} + n^P + n^{VR} = W \tag{1}$$

- El número de inductores está acotado por el número de puestos de inducción.

$$n^I \leq IPOST \tag{2}$$

- El número de *water-spiders* está limitado por el número de inductores.

$$n^{WS} \leq n^I \tag{3}$$

- El número de varios está limitado por el número de packers.

$$n^{VR} \leq n^P$$

- Dos tipos de restricciones adicionales asegurarán que se respeta el número de *packers* y varios:

$$\sum_{e \in W} rpack_e = n^P \tag{4}$$
$$\sum_{e \in W} rvar_e = n^{VR}$$

- Un mismo trabajador no puede estar asignado a más de un rol a la vez. En este caso, no puede ejercer a la vez de *paker* y de varios.

$$rpack_e + rvar_e \leq 1 \qquad e \in W \tag{5}$$

- Si un trabajador ejerce acciones de *packer* o de varios, obligadamente debe tener asignado ese rol. En este caso acotamos la acción de los trabajadores por su capacidad y el indicador binario de su rol[3].

$$\sum_{d \in D} ifpack_{der} \leq PAB \cdot rpack_e \qquad e \in W, r \in REV \tag{6}$$

$$\sum_{d \in D} ifvar_{der} \leq VRAB \cdot rvar_e \qquad e \in W, r \in REV$$

- Cada destino está asignado únicamente a un *packer*:

$$\sum_{e \in W} apack_{de} = 1 \qquad d \in D \tag{7}$$

- Un packer solo puede hacer su labor en un destino que tenga asignado (limitación que no tienen los varios):

$$ifpack_{der} \leq apack_{de} \qquad d \in D, e \in W, r \in REV \tag{8}$$

[3]Esta restricción puede ser escrita en el apartado de *Packing*, pero se escribió inicialmente en este apartado y se mantiene por el momento.

Inducción

- **Cotas superiores a la inducción**: el número de items inducidos por referencia y revolución estará acotado por la demanda restante y la capacidad inductora.

  - Una cota superior a los items inducidos de una referencia es la diferencia entre la demanda total y los items que ya fueron inducidos. Cuando se induzcan todos los items demandados de una referencia, esta resta resulta cero y no se inducen más items de dicha referencia.

$$nind_{ir} \leq \sum_{t \in TSA} DEMAND_{ti} - \sum_{s=0}^{r-1} nind_{is} \qquad i \in REF, r \in REV \setminus \{0\} \tag{9}$$

  - La suma de todos los items inducidos en una revolución está limitado por la capacidad inductora total más la aportación de los *water-spiders*.

$$\sum_{i \in REF} nind_{ir} \leq IAB \cdot n^I + WSAB \cdot n^{WS} \qquad r \in REV \tag{10}$$

  - La suma de items inducidos también estaría acotada teoricamente[4] por el total de bandejas menos las recirculaciones:

$$\sum_{i \in REF} nind_{ir} \leq S - \sum_{i \in REF} leftov_{i(r-1)} \qquad r \in REV \setminus \{0\}$$

- **Inducción de referencias por orden**: se debe dar sentido a las variables $already_{ir}$, así como acotar $nind_{ir}$ en función de estas variables.

  - Las variables $already_{ir}$ podrán tomar valor 1 cuando, hasta la revolución r incluída, se hayan inducido todos los items demandados de la referencia i.

$$\left(\sum_{t \in TSA} DEMAND_{ti} \right) \cdot already_{ir} \leq \sum_{s=0}^{r} nind_{is} \qquad i \in REF, \quad r \in REV \tag{11}$$

  - Aunque pueda resultar redundante, explicitamos que la variable $already_{ir}$ no puede decrecer, dado que si toma el valor 1 en la revolución r lo tomará en las siguientes.

$$already_{ir} \geq already_{i-r-1} \qquad i \in REF, \quad r \in REV \setminus \{0\} \tag{12}$$

  - La primera referencia se induce si impedimentos. Desde $i = 2$, se tendrá que $nind_{ir} = 0$ mientras $already_{i-1,r} = 0$. Cuando $already_{i-1,r} = 1$, se eliminará este tipo de impedimento.

$$nind_{ir} \leq \left(\sum_{t \in TSA} DEMAND_{ti} \right) \cdot already_{i-1,r} \qquad i \in REF, \quad i \geq 2, \quad r \in REV \tag{13}$$

[4]Esta restricción no tiene mucha utilidad práctica, dado que el valor de S suele ser lo suficientemente grande como para no romper esta restricción.

Movimiento del sorter y Caída en destino

- El total de items que caerán en todos los destinos está acotado por los items en circulación, para cada una de las referencias. Los items en circulación se entienden como los items que se han inducido en este ciclo más las recirculaciones:

$$\sum_{d \in D} nfall_{dir} \leq nind_{ir} + leftov_{i(r-1)} \qquad i \in REF, r \in REV \setminus \{0\} \tag{14}$$

- En cada destino y revolución, la caída de items (cualquier referencia) está limitada por el volumen total de la caja, añadiendo el margen superior. También debemos considerar que si una caja se llenó en el ciclo anterior, entonces no podrán caer items[5].

$$\sum_{i \in REF} VOL_i \cdot nfall_{dir} \leq (B + UEMARGIN) \cdot \left[1 - isfull_{d(r-1)}\right] \qquad d \in D, r \in REV \setminus \{0\} \tag{15}$$

- La caída de items en destino estará limitada por la demanda restante en la TSA correspondiente (aquella que aún no ha caído en destino), es decir:

$$\sum_{d \in TSADEST_t} nfall_{dir} \leq DEMAND_{ti} - \sum_{d \in TSADEST} \sum_{s=0}^{r-1} nfall_{dis} \qquad t \in TSA, i \in REF, r \in REV \tag{16}$$

- Otra cota a la caída de items en un destino es el volumen disponible en la caja de dicho destino (que se computa observando el valor de *itemsinbox* en el instante de tiempo anterior).

$$\sum_{i \in REF} VOL_i \cdot nfall_{dir} \leq (B + UEMARGIN) - \sum_{i \in REF} VOL_i \cdot itemsinbox_{di(r-1)} \tag{17}$$
$$d \in D, \quad r \in REV \setminus \{0\}$$

- Actualizamos los items en caja, pudiendo sumar a los existentes en la revolución anterior aquellos items que acaban de caer en caja. Nótese que, según está escrita la restricción, no limita la actualización a cero tras el *packing*.

$$itemsinbox_{dir} \leq itemsinbox_{di(r-1)} + nfall_{dir} \qquad d \in D, i \in REF, r \in REV \setminus \{0\} \tag{18}$$

[5] Aún en el caso en que la caja se llenase en el ciclo anterior y fuese repuesta, hay una revolución 'de enfriamiento' donde la caja no admite items

Packing

- Una caja puede pasar de no estar llena a estarlo si su volumen supera el total de la caja menos la cota inferior.

$$(B - LEMARGIN) \cdot isfull_{dr} \leq \sum_{i \in REF} VOL_i \cdot itemsinbox_{dir} \quad d \in D, r \in REV \tag{19}$$

- Solo se empaqueta si la caja se encuentra llena. Como $isfull_{dr}$ es binaria, también se incluye en esta restricción que en cada destino y tiempo solo puede haber una persona haciendo el *packing* a la vez:

$$\sum_{e \in W} (ifpack_{der} + ifvar_{der}) \leq isfull_{dr} \qquad d \in D, r \in REV \tag{20}$$

- Una caja se vacía si se ha realizado el packing en el instante anterior. Esta restricción hace que, de facto, se considere que pasa una revolución completa desde que una caja se llena hasta que se pueden volver a introducir items en ella. Podemos decir que en el lapso de una revolución (aquella en la que $itemsinbox$ valdrá cero) es donde se hace el *packing* de las cajas.

$$\sum_{i \in REF} VOL_i \cdot itemsinbox_{dir} \leq (B + UEMARGIN) \cdot \left(1 - \sum_{e \in W} \left[ifpack_{de(r-1)} + ifvar_{de(r-1)}\right]\right) \tag{21}$$

$$d \in D, r \in REV \setminus \{0\}$$

- **Definición de** $npack$. Los siguientes bloques de restricciones definen a $npack_{dir}$ para que se actualice solo cuando se vacía la caja correspondiente:

  - Si no hay otra restricción que lo impida, podrá tomar el valor de los items de su referencia en la caja:

$$npack_{dir} \leq itemsinbox_{dir} \qquad d \in D, i \in REF, r \in REV \tag{22}$$

  - Tomará el valor 0 a menos que en su revolución se realice el packing. En este último caso se le asigna una cota superior mayor que $itemsinbox_{dir}$:

$$npack_{dir} \leq \left(\sum_{t \in TSA} DEMAND_{ti}\right) \cdot \sum_{e \in W} (ifpack_{der} + ifvar_{der}) \qquad d \in D, i \in REF, r \in REV \tag{23}$$

- **Actualización de la demanda satisfecha en destino**. En la última revolución en que caen items en destino, es esperable que los items no lleguen a llenar las cajas para ser computada la demanda como completa. Por eso incluímos las siguientes restricciones que actualizan $cpletetsa_{tir}$ cuando toda la demanda restante se encuentra en cajas. Por como se define el modelo, debemos

hacer esta comprobación en la siguiente revolución (en caso contrario, *itemsinbox* se contabiliza dos veces), de aí que se tuviese en cuenta en la función objetivo este desfase de una revolución.

$$DEMAND_{ti} \cdot cpletetsa_{tir} \leq \sum_{d \in TSADEST_t} \sum_{s=0}^{r-1} npack_{dir} + \sum_{d \in TSADEST_t} itemsinbox_{dir} \qquad (24)$$
$$t \in TSA, \quad i \in REF, \quad r \in REV \setminus \{0\}$$

- **Actualización de las recirculaciones**: al final de una revolución, las recirculaciones son las anteriores, más los items inducidos, menos los items caídos en destino:

$$leftov_{ir} = leftov_{i(r-1)} + nind_{ir} - \sum_{d \in D} nfall_{dir} \qquad i \in REF, r \in REV \setminus \{0\}$$

Cumplimiento de la demanda

- Obligatoriedad de cumplir en algún momento la demanda de cada TSA. Una forma de implementarlo es esta:

$$\sum_{r \in REV} cpletetsa_{tir} \geq 1 \qquad t \in TSA, i \in REF \qquad (25)$$

- Debemos dar significado a la variable $final_r$ como indicador de que la demanda total se haya completado.

$$final_r \leq \frac{\sum_{t \in TSA} \sum_{i \in REF} cpletetsa_{tir}}{|TSA| \cdot |REF|} \qquad r \in REV \qquad (26)$$

- Finalmente, imponemos que a partir de algún momento esta variable deba valer 1:

$$\sum_{r \in REV} final_r \geq 1 \qquad (27)$$

Valores iniciales y finales de las variables

Respecto de los valores iniciales, se puede resumir este apartado en que toda las variables dependientes del tiempo tomarán el valor 0 cuando $r = 0$ ($r \in REV$). Por ejemplo:

$$nind_{i0} = nfall_{di0} = ... = ifpack_{de0} = ... = itemsinbox_{di0} = cpletetsa_{ti0} = final_0 = 0$$

$$i \in REF, \quad d \in D, \quad e \in W, \quad t \in TSA$$

No debería ser necesario incluír restricciones sobre el valor de las variables en el último instante de tiempo, o sobre la suma de sus valores a lo largo del tiempo, puesto que el modelo se construye con la expectativa de que se alcancen estos valores. No obstante, podemos escribir por precaución las siguientes restricciones:

- Al final del proceso, deben haber sido inducidos los mismos items de cada referencia que constituían la demanda:

$$\sum_{r \in REV} nind_{ir} = \sum_{t \in TSA} DEMAND_{ti} \qquad i \in REF$$

- De igual modo, para la demanda de cada TSA, deben haber caído los mismos items de cada referencia en los destinos asignados a dicha tienda.

$$\sum_{d \in TSADEST_t} \sum_{r \in REV} nfall_{dir} = DEMAND_{ti} \qquad i \in REF$$

- Al final del proceso, al sumar los items que quedan en caja más aquellos a los que ha realizado el *packing* en revoluciones anteriores, debemos obtener la demanda total:

$$\sum_{d \in TSADEST_t} \sum_{s=0}^{r-1} npack_{dis} + \sum_{d \in TSADEST_t} itemsinbox_{diREV} = DEMAND_{ti} \qquad i \in REF$$

- Cuando el proceso termina no puede haber items circulando, tampoco recirculando.

$$leftov_{iREV} = 0$$

Roturas de simetría

Una vez validado el modelo, resulta inmediato apreciar la gran cantidad de simetrías que presenta. Las más obvias son aquellas propias de la asignación de los trabajadores a sus roles, así como la asignación de destinos a los *packers* (por ejemplo, si hay u inductor y un *packer*, puede ser que el trabajador 1 sea inductor y el trabajador 2 sea el *packer*, pero también puede ser al revés).

Las restricciones que se detallan a continuación tratan de imponer un orden en como se asignan los índices de trabajadores a roles, también de destinos a *packers*, para evitar el mayor número de simetrías posible.

- Este bloque de restricciones implica que el orden de los trabajadores (en tanto que son el conjunto $\{1, ..., W\}$) se tenga en cuenta al asignar los destinos a los packers. Los packers con valores numéricos más bajos serán los primeros en asignarse a los destinos. Además, aunque no tenga trascendencia en este modelo concreto, evitamos el caso en que los destinos asignados a un packer no son consecutivos.

$$apack_{\tilde{d}e} + apack_{d\tilde{e}} \leq 1 \qquad d < \tilde{d} \quad d, \tilde{d} \in D \quad e < \tilde{e} \quad e, \tilde{e} \in W \tag{28}$$

- Inpondremos un orden de asignación de trabajadores a sus roles. Como de momento solo se asignan explícitamente los roles de *packer* y varios a los trabajadores, impondremos que los trabajadores con primeros índices sean siempre asignados al rol de *packer*, y los inmediatamente siguentes al rol de varios.

  - El trabajador $1 \in W$ será siempre un packer:

$$rpack_1 = 1$$

  - A partir del trabajador $2 \in W$ obligaremos a que si $e \in W$ es un packer, entonces $e-1$ también lo sea.

$$rpack_e - epack_{e-1} \leq 0 \qquad e \in W \quad e \geq 2 \tag{29}$$

  - Para afinar aún más en la rotura de simetrías, si el trabajador e es un varios, nos interesará que siempre el trabajador $e-1$ deba ser o bien un varios o bien un packer.

$$rvar_e - rvar_{e-1} - rpack_{e-1} \leq 0 \qquad e \in W \quad e \geq 2 \tag{30}$$

  - NOTA: si se amplía el modelo se pueden completar estas restricciones, por ejemplo imponiendo que el orden de asignación de los trabajadores en base a su índice sea packers - varios - inductores - waterspiders.

Otras restricciones

- **Reparto equitativo de destinos entre packers**: vamos a obligar a que la cantidad de desinos asignada a dos packers no difiera en más de 1 destino. Las roturas de simetría incluídas en el apartado anterior obligan a que si $e \in W$ no es un *packer*, entonces $\tilde{e} > e$ tampoco lo es. Entonces podemos imponer este reparto usando solamente las variables *apack* tal y como se muestra a continuación (nótese que se contempla tanto que ambos trabajadores sean *packers*, como que solo uno o incluso ninguno lo sea):

$$\sum_{d \in D} (apack_{d\tilde{e}} - apack_{de}) \leq 1 \qquad e, \tilde{e} \in W \quad e < \tilde{e} \tag{31}$$

  - Si en algún momento eliminamos dicha restricción, tendríamos que recurrir a la siguiente formulación:

$$\sum_{d \in D} (apack_{d\tilde{e}} - apack_{de}) \leq 1 + D \cdot (rpack_e - rpack_{\tilde{e}}) \qquad e, \tilde{e} \in W \quad e < \tilde{e}$$

- El número de *packers* debe ser un número par.

$$n^P - 2 \cdot hpack = 0$$

Tamaño del problema

En esta última sección nos limitamos a contabilizar el número de variables y restricciones de que se compone el problema, en función de los parámetros W, REF, REV, TSA y D.

Dado que no nos interesa el número concreto de variables o restricciones sino su escala, o como aumentan en función de los parámetros mencionados, no haremos un cálculo exacto sino un redondeo. Por ejemplo escribiremos REV en lugar de $REV - 1$ y D en lugar de $D + 2$.

Número de Variables

$$[(2W + 2REF) \cdot D + TSA \cdot REF + REF] \cdot REV + D \cdot W \tag{32}$$

Número de Restricciones

En este caso, para mayor comprensión, se hace un desglose por apartados y luego se muestra el total:

- **Trabajadores**:
$$REV \cdot W \cdot D + D + W$$
- **Inducción**:
$$3 \cdot REV \cdot REF$$
- **Movimiento del sorter y Caída en destino**:
$$REV \cdot (D \cdot REF + TSA \cdot REF + D + REF)$$
- **Packing**:
$$REV \cdot (D \cdot REF + TSA \cdot REF + D + REF)$$
- **Cumplimiento de la demanda**:
$$REV + TSA \cdot REF$$
- **Roturas de Simetría**:
$$\frac{1}{4} \cdot D^2 \cdot W^2$$
- **Otras restricciones**:
$$W^2$$

Omitimos la contabilización de las restricciones consistentes en dar un valor inicial a las variables en tiempo cero, aunque si se quieren añadir basta con recordar que coinciden con el número de variables.

Sumando las anteriores, nos queda un total de restricciones de:

$$REV \cdot (D \cdot W + D \cdot REF + TSA \cdot REF + 2D + 5REF) + \frac{1}{4}D^2 \cdot W^2 + D + W + TSA + REF \tag{33}$$

Modelo Multirreferencia
V 1.5

Ángel Mourelle Abelenda

Supuestos del modelo

- **Tiempo en ciclos o revoluciones**: la 'unidad de tiempo' en este modelo son las revoluciones, es decir, el tiempo que tarda el sorter en volver a una posición determinada tras abandonarla (por ejemplo, con respecto a una bandeja). A lo largo de una revolución cada bandeja pasa por todos los procesos (inducción, caída en destino, etc.). Normalmente es conocido el tiempo que dura una revolución, por lo que bastaría con dividir el tiempo total entre este para obtener el número de revoluciones a introducir como parámetro del problema.
- **Multirreferencia**: los items llevan una referencia asociada que indica el tipo de producto (camiseta blanca, camiseta negra, sudadera roja...).
- **Volumen como magnitud**: el contenido de las cajas en destino, así como la determinación de si están o no llenas, se hará en base a unidades de volumen. Cada referencia lleva asociado un volumen, por lo que cada item contará con su volumen.
- **Los items con igual referencia son iguales**: no se trabaja con los items de forma individual. El modelo solo contemplará en cada paso la cantidad total de items de cada referencia que se inducen, circulan, caen, recirculan, etc.
- **Demanda**: la demanda es finita y consiste en la cantidad de items de cada referencia que solicita cada TSA.
- **Destinos asociados a TSAs**: cada destino tendrá asociada una y solo una TSA. De este modo, la demanda de una TSA se satisfará entre todos los destinos que tienen asociada esa TSA.
- **Inductores en bloque**: de momento se contabilizan los inductores y su tarea como un bloque, y no tanto la suma de individuos con cargas de trabajo diferente. Los *water-spiders* tendrán una influencia reducida en el modelo, y su número dependerá de los puestos de inducción disponibles [1]. La idea es que más adelante se pueda evolucionar el modelo 'de forma modular', añadiendo variables y modificando cómo se calcula la suma de items inducidos en cada revolución.
- **Las referencias se inducen por orden**: los items de la segunda referencia registrada solo se empezarán a inducir cuando se hayan inducido los de la primera referencia, y así sucesivamente.

[1] Si se necesita mayor capacidad inductora que los inductores posibles, será cuando el modelo designe algún *water-spider*.

Principal añadido a la versión V 1.5

Cambios dinámicos del número de trabajadores por rol durante el proceso. El principal añadido a esta versión del modelo. No permitiremos que en cada unidad de tiempo se pueda reestructurar el esquema de trabajadores por rol, dado que esto daría lugar a demasiadas variables sin utilidad (pues no se puede cambiar este esquema constantemente, y al poner límites temporales lo que ocurre es que la mayoría de las variables no aportan nada al modelo).

En su lugar, consideraremos un conjunto de puntos temporales, a saber $REVCUT$, que serán aquellos donde se permite la evaluación y eventual cambio en el número de trabajadores por rol. Otro parámetro, $STRETCH$, devuelve para cada unidad temporal la 'etapa' en la que se ubica, es decir, después de que momento en $REVCUT$ se encuentra. Así, las variables dependientes del rol que ocupa un trabajador no se iteran en todas las unidades de tiempo, sino solo a lo largo de las etapas[2]

[2]Un ejemplo: Si en 200 revoluciones consideramos proponer 3 instantes en $REVCUT$ (que no sean el inicial ni el final), entonces habrá 5 etapas (5 valores distintos aunque repetidos en $STRETCH$), y muchas variables pasan de iterar en un conjunto de 200 índices a uno de 5.

Parámetros

- W: nº total de trabajadores, que serán asignados a cada rol. Trabajadores $e \in \{1, ..., W\}$
- S: nº total de bandejas (*slots*) donde se pueden depositar items. No tendrá mucha utilidad en el problema, más allá de acotar la capacidad de inducción, junto con las recirculaciones, en casos específicos.
- $IPOST$: nº de puestos de inducción, es decir, cota superior al número de inductores para el modelo.
- REV: nº total de revoluciones. Revoluciones $r \in \{1, ..., REV\}$.
- $REVCUT$: instantes $r_1, r_2, ..., r_{|REVCUTS|} \in \{1, 2, ..., REV - 1\}$ en los que se permitirá cambiar el número de trabajadores que ocupan cada rol, sin contar la configuración inicial, que también se calcula.
- $STRETCH$: parámetro indexado en REV que a cada r asigna un valor en el conjunto $\{1, 2, ..., |REVCUT| + 2\}$ de etapas en que se divide REV en base a los elementos de $REVCUT$[3].
- $STRETCH^*$: conjunto de elementos diferentes[4] del conjunto $STRETCH$. Supone por tanto el conjunto de índices de las etapas en que se divide REV.
- REF: nº total de referencias. Referencias $i \in \{1, ..., REF\}$.
- VOL_i: para cada $i \in \{1, ..., REF\}$, unidades de volumen que ocupa cada item de la referencia i.
- TSA: nº de TSAs [5], quienes generan la demanda. TSAs $t \in \{1, ..., TSA\}$.
- D: nº de destinos, aquellos en los que pueden caer items en cajas. Destinos $d \in \{1, .., D\}$.
- $DEMAND_{ti}$: demanda total de items de cada referencia i que hace la TSA t.
- $TSADEST_t$: conjunto de destinos que se asignan a cada TSA.
- B: capacidad (en volumen) de cada caja en destino. Lo ideal es que una caja se llene al sumar volumen B (salvando márgenes de error) y entonces se reponga.
- $LEMARGIN$ y $UEMARGIN$: respectivamente, los márgenes de error inferiores y superiores a los vlúmenes de la caja. Podemos añadir el último item en destino a una caja si sobrepasa B en un volumen menor a $UEMARGIN$. Del mismo modo, podremos considerar llena una caja con volumen superior a $B - LEMARGIN$. Como cada referencia puede tener un volumen diferente, estos márgenes permiten evitar los casos en que las cajas no se 'llenen de forma exacta'.
- IAB: nº promedio de items que un inductor es capaz de inducir por revolución.
- PAB: nº de cajas llenas que un *packer* es capaz de reponer durante una revolución.
- $WSAB$: plus de items que puede inducir un inductor si es ayudado por un *water-spider*. Con las asunciones de este modelo (inductores com obloque), podemos también considerar los *water-spiders* como bloque (de menor número que los inductores) y su aportación como una suma a la capacidad de los inductores.
- $VRAB$: análogo a PAB para varios. Se usa un parámetro diferente pues es presumible que si el varios se puede desplazar por todos los destinos, esta tardanza se pueda reflejar en un valor de $VRAB$ menor que el PAB de los *packers*.

De ahora en adelante será habitual el abuso de notación de denotar, por ejemplo, a REV como el conjunto $\{1, ..., REV\}$. Lo mismo aplicará para W, REF, TSA, D.

[3] Por ejemplo, si $REVCUT_1 = 3$, entonces $STRETCH_0 = STRETCH_1 = STRETCH_2 = 1$ y $STRETCH_3 = 2$, y así sucesivamente.
[4] Es decir, sin contar las repeticiones.
[5] Tienda Sección Aparte

Variables

- $n_s^I \in \{1, ..., W\}$: cantidad de inductores en cada etapa $s \in STRETCH^*$.
- $n_s^{WS} \in \{1, ..., W\}$: cantidad de *water-spiders* en cada etapa $s \in STRETCH^*$.
- $n_s^P \in \{1, ..., W\}$: cantidad de *packers* por etapa.
- $hpack_s \in \mathbb{Z}^+$: mitad de la cantidad de *packers* en cada etapa. Imponemos que esta variable sea entera para que el nº de *packers* sea par.
- $n_s^{VR} \in \{1, ..., W\}$: cantidad de varios (apoyo a *packers*) en cada etapa.
- $rpack_{es}$: indicador binario de si el trabajador $e \in \{1, ..., W\}$ ejerce de *packer* durante la etapa $s \in STRETCH^*$.
- $rvar_{es}$: análogo para varios (rol de apoyo de los *packers*).
- $apack_{des}$: indicador binario de si el destino d está asignado al trabajador (*packer*) e durante la etapa s.
- $nind_{ir}$: nº de items que se inducen de la referencia $i \in REF$ en la revolución $r \in REV$.
- $already_{ir}$: indicador binario de si la demanda de la referencia $i \in REF$ ha sido completamente inducida en la revolución $r \in REV$.
- $nfall_{dir}$: nº de items de la referencia i que, en la revolución r, caen en el destino d.
- $itemsinbox_{dir}$: items de la referencia $i \in REF$ que contendrá la caja (tras la caída de items) en destino $d \in D$ en tiempo $r \in REV$.
- $isfull_{dr}$: indicador binario de si la caja en el destino d se llena a lo largo de la revolución r.
- $ifpack_{der}$: indica si en tiempo r el trabajador (*Packer*) $e \in W$ está empaquetando (reponiendo) la caja en el destino d. Se puede hacer el packing de una caja que se llena en tiempo r.
- $ifvar_{der}$: lo análogo para los *Varios* (rol de apoyo de los *Packers*). La diferencia de modelizado se basa en el hecho de que los *Packers* tienen sus destinos asignados, mientras que los *Varios* no tienen esa limitación.
- $npack_{dir}$: nº de items de la referencia i que, cuando se reponga la caja d, se sumarán a la demanda satisfecha.
- $leftov_{ir}$: nº de items de la referencia i en movimiento que no han caído en destino en tiempo r, es decir, que recircularán en el siguiente ciclo $r + 1$.
- $cpletetsa_{tir}$: indicador binario de si la demanda de la TSA t del item de referencia i ha sido satisfecha en la revolución r.
- $final_r$: : variable binaria que nos indicará si toda la demanda de todas las TSA ya se ha cubierto en la revolución r.

Función objetivo

La idea más básica es cumplir con la demanda en el menor tiempo posible. Una forma de hacer esto es maximizando la suma de indicadores binarios de que la demanda se ha completado, pues para cada revolución valdrán 1 si ya se ha cumplido toda la demanda. Aún mejor sería minimizar la resta del total de revoluciones menos la suma anterior, dado que el resultado sería el número de revoluciones en que se completa la demanda menos uno. Más adelante se verá que el modelo, salvo casos muy concretos, finalizará una revolución más tarde, por lo que no tenemos que sumar esa unidad de tiempo:

$$\text{mín} \quad REV - \sum_{r \in REV} final_r$$

Restricciones

Trabajadores

- En cada una de las etapas, la suma total de los trabajadores que ocupan los diferentes roles es W.

$$n_s^I + n_s^{WS} + n_s^P + n_s^{VR} = W \qquad s \in STRETCH^* \tag{1}$$

- El número de inductores está acotado por el número de puestos de inducción.

$$n_s^I \leq IPOST \qquad s \in STRETCH^* \tag{2}$$

- El número de *water-spiders* está limitado por el número de inductores, y el número de varios está limitado por el número de packers.

$$n_s^{WS} \leq n^I \qquad s \in STRETCH^* \tag{3}$$

$$n_s^{VR} \leq n^P \qquad s \in STRETCH^*$$

- Dos tipos de restricciones adicionales asegurarán que se respeta el número de *packers* y varios:

$$\sum_{e \in W} rpack_{es} = n_s^P \qquad s \in STRETCH^* \tag{4}$$

$$\sum_{e \in W} rvar_{es} = n_s^{VR} \qquad s \in STRETCH^*$$

- Un mismo trabajador no puede estar asignado a más de un rol a la vez. En este caso, no puede ejercer a la vez de *paker* y de varios. Aplica a todas las etapas.

$$rpack_{es} + rvar_{es} \leq 1 \qquad e \in W, \quad s \in STRETCH^* \tag{5}$$

- Si un trabajador ejerce acciones de *packer* o de varios, obligadamente debe tener asignado ese rol. En este caso acotamos la acción de los trabajadores por su capacidad y el indicador binario de su rol, en la etapa correspondiente a esa revolución[6].

[6]Esta restricción puede ser escrita en el apartado de *Packing*, pero se escribió inicialmente en este apartado y se mantiene por el momento.

$$\sum_{d \in D} ifpack_{der} \leq PAB \cdot rpack_{e,STRETCH_r} \qquad e \in W, \quad r \in REV \tag{6}$$

$$\sum_{d \in D} ifvar_{der} \leq VRAB \cdot rvar_{e,STRETCH_r} \qquad e \in W, \quad r \in REV$$

- En cada etaapa, cada destino está asignado únicamente a un *packer*:

$$\sum_{e \in W} apack_{des} = 1 \qquad d \in D, \quad s \in STRETCH^* \tag{7}$$

- Un packer solo puede hacer su labor en un destino que tenga asignado en la etapa correspondiente (limitación que no tienen los varios):

$$ifpack_{der} \leq apack_{d,e,STRETCH_r} \qquad d \in D, e \in W, r \in REV \tag{8}$$

Inducción

- **Cotas superiores a la inducción**: el número de items inducidos por referencia y revolución estará acotado por la demanda restante y la capacidad inductora.

  - Una cota superior a los items inducidos de una referencia es la diferencia entre la demanda total y los items que ya fueron inducidos. Cuando se induzcan todos los items demandados de una referencia, esta resta resulta cero y no se inducen más items de dicha referencia.

$$nind_{ir} \leq \sum_{t \in TSA} DEMAND_{ti} - \sum_{k=0}^{r-1} nind_{ik} \qquad i \in REF, r \in REV \setminus \{0\} \tag{9}$$

  - La suma de todos los items inducidos en una revolución está limitado por la capacidad inductora total más la aportación de los *water-spiders* en ese momento.

$$\sum_{i \in REF} nind_{ir} \leq IAB \cdot n^{I}_{STRETCH_r} + WSAB \cdot n^{WS}_{STRETCH_r} \qquad r \in REV \tag{10}$$

  - La suma de items inducidos también estaría acotada teoricamente[7] por el total de bandejas menos las recirculaciones:

$$\sum_{i \in REF} nind_{ir} \leq S - \sum_{i \in REF} leftov_{i(r-1)} \qquad r \in REV \setminus \{0\}$$

- **Inducción de referencias por orden**: se debe dar sentido a las variables $already_{ir}$, así como acotar $nind_{ir}$ en función de estas variables.

  - Las variables $already_{ir}$ podrán tomar valor 1 cuando, hasta la revolución r incluída, se hayan inducido todos los items demandados de la referencia i.

$$\left(\sum_{t \in TSA} DEMAND_{ti} \right) \cdot already_{ir} \leq \sum_{k=0}^{r} nind_{ik} \qquad i \in REF, \quad r \in REV \tag{11}$$

[7] Esta restricción no tiene mucha utilidad práctica, dado que el valor de S suele ser lo suficientemente grande como para no romper esta restricción.

- La primera referencia se induce si impedimentos. Desde $i = 2$, se tendrá que $nind_{ir} = 0$ mientras $already_{i-1,r} = 0$. Cuando $already_{i-1,r} = 1$, se eliminará este tipo de impedimento.

$$nind_{ir} \leq \left(\sum_{t \in TSA} DEMAND_{ti} \right) \cdot already_{i-1,r} \qquad i \in REF, \quad i \geq 2, \quad r \in REV \tag{12}$$

Movimiento del sorter y Caída en destino

- El total de items que caerán en todos los destinos está acotado por los items en circulación, para cada una de las referencias. Los items en circulación se entienden como los items que se han inducido en este ciclo más las recirculaciones:

$$\sum_{d \in D} nfall_{dir} \leq nind_{ir} + leftov_{i(r-1)} \qquad i \in REF, \quad r \in REV \setminus \{0\} \tag{13}$$

- En cada destino y revolución, la caída de items (cualquier referencia) está limitada por el volumen total de la caja, añadiendo el margen superior. También debemos considerar que si una caja se llenó en el ciclo anterior, entonces no podrán caer items[8].

$$\sum_{i \in REF} VOL_i \cdot nfall_{dir} \leq (B + UEMARGIN) \cdot \left[1 - isfull_{d(r-1)}\right] \qquad d \in D, r \in REV \setminus \{0\} \tag{14}$$

- La caída de items en destino estará limitada por la demanda restante en la TSA correspondiente (aquella que aún no ha caído en destino), es decir:

$$\sum_{d \in TSADEST_t} nfall_{dir} \leq DEMAND_{ti} - \sum_{d \in TSADEST} \sum_{k=0}^{r-1} nfall_{dik} \qquad t \in TSA, i \in REF, r \in REV \tag{15}$$

- Otra cota a la caída de items en un destino es el volumen disponible en la caja de dicho destino (que se computa observando el valor de *itemsinbox* en el instante de tiempo anterior).

$$\sum_{i \in REF} VOL_i \cdot nfall_{dir} \leq (B + UEMARGIN) - \sum_{i \in REF} VOL_i \cdot itemsinbox_{di(r-1)} \tag{16}$$
$$d \in D, \quad r \in REV \setminus \{0\}$$

- Actualizamos los items en caja, pudiendo sumar a los existentes en la revolución anterior aquellos items que acaban de caer en caja. Nótese que, según está escrita la restricción, no limita la actualización a cero tras el *packing*.

$$itemsinbox_{dir} \leq itemsinbox_{di(r-1)} + nfall_{dir} \qquad d \in D, i \in REF, r \in REV \setminus \{0\} \tag{17}$$

[8] Aún en el caso en que la caja se llenase en el ciclo anterior y fuese repuesta, hay una revolución 'de enfriamiento' donde la caja no admite items

Packing

- Una caja puede pasar de no estar llena a estarlo si su volumen supera el total de la caja menos la cota inferior.

$$(B - LEMARGIN) \cdot isfull_{dr} \leq \sum_{i \in REF} VOL_i \cdot itemsinbox_{dir} \quad d \in D, r \in REV \tag{18}$$

- Solo se empaqueta si la caja se encuentra llena. Como $isfull_{dr}$ es binaria, también se incluye en esta restricción que en cada destino y tiempo solo puede haber una persona haciendo el *packing* a la vez:

$$\sum_{e \in W} (ifpack_{der} + ifvar_{der}) \leq isfull_{dr} \qquad d \in D, r \in REV \tag{19}$$

- Una caja se vacía si se ha realizado el packing en el instante anterior. Esta restricción hace que, de facto, se considere que pasa una revolución completa desde que una caja se llena hasta que se pueden volver a introducir items en ella. Podemos decir que en el lapso de una revolución (aquella en la que *itemsinbox* valdrá cero) es donde se hace el *packing* de las cajas.

$$\sum_{i \in REF} VOL_i \cdot itemsinbox_{dir} \leq (B + UEMARGIN) \cdot \left(1 - \sum_{e \in W} \left[ifpack_{de(r-1)} + ifvar_{de(r-1)}\right]\right) \tag{20}$$
$$d \in D, r \in REV \setminus \{0\}$$

- **Definición de** *npack*. Los siguientes bloques de restricciones definen a $npack_{dir}$ para que se actualice solo cuando se vacía la caja correspondiente:

  - Si no hay otra restricción que lo impida, podrá tomar el valor de los items de su referencia en la caja:

$$npack_{dir} \leq itemsinbox_{di(r-1)} + nfall_{dir} \qquad d \in D, i \in REF, r \in REV \tag{21}$$

  - Tomará el valor 0 a menos que en su revolución se realice el packing. En este último caso se le asigna una cota superior mayor que $itemsinbox_{dir}$:

$$npack_{dir} \leq \left(\sum_{t \in TSA} DEMAND_{ti}\right) \cdot \sum_{e \in W} (ifpack_{der} + ifvar_{der}) \qquad d \in D, i \in REF, r \in REV \tag{22}$$

  - Del mimso modo, podemos añadir también una cota inferior a *npack* como la siguiente:

$$npack_{dir} \geq (B - LEMARGIN) \cdot \sum_{e \in W} (ifpack_{der} + ifvar_{der}) \quad d \in D, i \in REF, r \in REV \setminus \{0\}$$

- **Actualización de la demanda satisfecha en destino**. En la última revolución en que caen items en destino, es esperable que los items no lleguen a llenar las cajas para ser computada la demanda como completa. Por eso incluímos las siguientes restricciones que actualizan $cpletetsa_{tir}$ cuando toda la demanda restante se encuentra en cajas. Por como se define el modelo, debemos hacer esta comprobación en la siguiente revolución (en caso contrario, *itemsinbox* se contabiliza dos veces), de aí que se tuviese en cuenta en la función objetivo este desfase de una revolución.

$$DEMAND_{ti} \cdot cpletetsa_{tir} \leq \sum_{d \in TSADEST_t} \sum_{k=0}^{r-1} npack_{dik} + \sum_{d \in TSADEST_t} itemsinbox_{dir} \tag{23}$$
$$t \in TSA, \quad i \in REF, \quad r \in REV \setminus \{0\}$$

- **Actualización de las recirculaciones**: al final de una revolución, las recirculaciones son las anteriores, más los items inducidos, menos los items caídos en destino:

$$leftov_{ir} = leftov_{i(r-1)} + nind_{ir} - \sum_{d \in D} nfall_{dir} \qquad i \in REF, r \in REV \setminus \{0\}$$

Cumplimiento de la demanda

- Obligatoriedad de cumplir en algún momento la demanda de cada TSA. Una forma de implementarlo es esta:

$$\sum_{r \in REV} cpletetsa_{tir} \geq 1 \qquad t \in TSA, i \in REF \tag{24}$$

- Debemos dar significado a la variable $final_r$ como indicador de que la demanda total se haya completado.

$$final_r \leq \frac{\sum_{t \in TSA} \sum_{i \in REF} cpletetsa_{tir}}{|TSA| \cdot |REF|} \qquad r \in REV \tag{25}$$

- Finalmente, imponemos que a partir de algún momento esta variable deba valer 1:

$$\sum_{r \in REV} final_r \geq 1 \tag{26}$$

Valores iniciales y finales de las variables

Respecto de los valores iniciales, se puede resumir este apartado en que toda las variables dependientes del tiempo tomarán el valor 0 cuando $r = 0$ ($r \in REV$). Por ejemplo:

$$nind_{i0} = nfall_{di0} = ... = ifpack_{de0} = ... = itemsinbox_{di0} = cpletetsa_{ti0} = final_0 = 0$$

$$i \in REF, \quad d \in D, \quad e \in W, \quad t \in TSA$$

No debería ser necesario incluír restricciones sobre el valor de las variables en el último instante de tiempo, o sobre la suma de sus valores a lo largo del tiempo, puesto que el modelo se construye con la expectativa de que se alcancen estos valores. No obstante, podemos escribir por precaución las siguientes restricciones:

- Al final del proceso, deben haber sido inducidos los mismos items de cada referencia que constituían la demanda:

$$\sum_{r \in REV} nind_{ir} = \sum_{t \in TSA} DEMAND_{ti} \qquad i \in REF$$

- De igual modo, para la demanda de cada TSA, deben haber caído los mismos items de cada referencia en los destinos asignados a dicha tienda.

$$\sum_{d \in TSADEST_t} \sum_{r \in REV} nfall_{dir} = DEMAND_{ti} \qquad i \in REF$$

- Al final del proceso, al sumar los items que quedan en caja más aquellos a los que ha realizado el *packing* en revoluciones anteriores, debemos obtener la demanda total:

$$\sum_{d \in TSADEST_t} \sum_{k=0}^{r-1} npack_{dik} + \sum_{d \in TSADEST_t} itemsinbox_{diREV} = DEMAND_{ti} \qquad i \in REF$$

- Cuando el proceso termina no puede haber items circulando, tampoco recirculando.

$$leftov_{iREV} = 0$$

Roturas de simetría

Una vez validado el modelo, resulta inmediato apreciar la gran cantidad de simetrías que presenta. Las más obvias son aquellas propias de la asignación de los trabajadores a sus roles, así como la asignación de destinos a los *packers* (por ejemplo, si hay u inductor y un *packer*, puede ser que el trabajador 1 sea inductor y el trabajador 2 sea el *packer*, pero también puede ser al revés).

Las restricciones que se detallan a continuación tratan de imponer un orden en como se asignan los índices de trabajadores a roles, también de destinos a *packers*, para evitar el mayor número de simetrías posible.

- Este bloque de restricciones implica que el orden de los trabajadores (en tanto que son el conjunto $\{1, ..., W\}$) se tenga en cuenta al asignar los destinos a los packers. Los packers con valores numéricos más bajos serán los primeros en asignarse a los destinos en cada etapa. Además, aunque no tenga trascendencia en este modelo concreto, evitamos el caso en que los destinos asignados a un packer no son consecutivos.

$$apack_{\tilde{d}es} + apack_{d\tilde{e}s} \leq 1 \qquad d < \tilde{d} \quad d, \tilde{d} \in D \quad e < \tilde{e} \quad e, \tilde{e} \in W, \quad s \in STRETCH^* \tag{27}$$

- Inpondremos un orden de asignación de trabajadores a sus roles. Como de momento solo se asignan explícitamente los roles de *packer* y varios a los trabajadores, impondremos que los trabajadores con primeros índices sean siempre asignados al rol de *packer*, y los inmediatamente siguentes al rol de varios.

  - El trabajador $1 \in W$ será siempre un packer:

$$rpack_{1s} = 1 \qquad s \in STRETCH^*$$

  - A partir del trabajador $2 \in W$ obligaremos a que si $e \in W$ es un packer, entonces $e - 1$ también lo sea.

$$rpack_{es} - epack_{(e-1)s} \leq 0 \qquad e \in W, \quad e \geq 2, \quad s \in STRETCH^* \tag{28}$$

  - Para afinar aún más en la rotura de simetrías, si el trabajador e es un varios, nos interesará que siempre el trabajador $e - 1$ deba ser o bien un varios o bien un packer.

$$rvar_{es} - rvar_{(e-1)s} - rpack_{(e-1)s} \leq 0 \qquad e \in W, \quad e \geq 2, \quad s \in STRETCH^* \tag{29}$$

  - NOTA: si se amplía el modelo se pueden completar estas restricciones, por ejemplo imponiendo que el orden de asignación de los trabajadores en base a su índice sea packers - varios - inductores - waterspiders.

Otras restricciones

- El número de *packers* siempre debe ser un número par.

$$n_s^P - 2 \cdot hpack_s = 0 \qquad s \in STRETCH^* \tag{30}$$

- **Reparto equitativo de destinos entre packers**: vamos a obligar a que la cantidad de desinos asignada a dos packers no difiera en más de 1 destino. Las roturas de simetría incluídas en el apartado anterior obligan (en cada etapa) a que si $e \in W$ no es un *packer*, entonces $\tilde{e} > e$ tampoco lo es. Entonces podemos imponer este reparto usando solamente las variables *apack* tal y como se muestra a continuación (nótese que se contempla tanto que ambos trabajadores sean *packers*, como que solo uno o incluso ninguno lo sea):

$$\sum_{d \in D} (apack_{d\tilde{e}s} - apack_{des}) \leq 1 \qquad e, \tilde{e} \in W, \quad e < \tilde{e}, \quad s \in STRETCH^* \tag{31}$$

- Si en algún momento eliminamos dicha restricción, tendríamos que recurrir a la siguiente formulación:

$$\sum_{d \in D} (apack_{d\tilde{e}s} - apack_{des}) \leq 1 + D \cdot (rpack_{es} - rpack_{\tilde{e}s}) \qquad e, \tilde{e} \in W, \quad e < \tilde{e}, \quad s \in STRETCH^*$$

Propuestas para modelizar la Inducción
Modelo Multirreferencia V1.5

Ángel Mourelle Abelenda

Motivación

Este documento presentará dos posibles enfoques de cara a la adición del proceso de inducción y la aportanción de los water-spiders al modelo. Se toma como referencia la versión V1.5 del modelo, que incluye la posibilidad de realizar cambios en el nº de trabajadores por rol en ciertos momentos del proceso.

El motivo de preparar este documento es que ambas posibilidades presentan pros y contras, por lo que se tratará de hacer un boceto ilustrativo de lo que significa cada forma de implementar la inducción y sus implicaciones, a fin de poder decidir, con conocimiento de causa, cual es la mejor a implementar.

Puntos comunes a ambos enfoques

La diferencia fundamental entre los dos enfoques es como trabajar con el paso del tiempo y el desgaste que produce en la productividad de los trabajadores, así como en el movimiento y acción de los water-spiders.

Por lo demás, para ambos casos se plantea medir la capacidad inductora de cada trabajador en base a un ratio o proporción de inducciones respecto del totalde bandejas. Es decir, si un trabajador tiene una ratio de 0,4, significa que de cada 10 bandejas totales, el empleado es capaz de inducir 4.

Observación 1. *Lo ideal sería considerar un ratio de inducciones por bandeja vacía que pasa por el puesto de inducción, es decir, que se calculase respecto del tota de bandejas menos las que ya han inducidos los trabajadores ubicados en los puestos de inducción previos. En este caso, la ratio iría aumentando en el sentido de la marcha del sorter, pues cada vez llegarían más espaciadas las bandejas vacías*[1]*. No obstante, este enfoque entraña una modelización más engorrosa, y es más sencillo simplemente escribir unas ratios equivalentes a estas pero calculadas sobre el total de bandejas.*

La virtud de este parámetro es que se pueden considerar ratios diferentes para cada trabajador. Supongamos que hai $IPOST = 4$ puestos de inducción, y todos están ocupados por un inductor. Digamos que $RATIO = \{0{,}35, 0{,}29, 0{,}23, 0{,}10\}$, donde cada coordenada es el ratio del trabajador que se ubica en ese puesto de inducción. La ratio disminuye en el sentido de la marcha del sorter, porque el primer inductor dispone de todas las bandejas vacías e induce al máximo ritmo a que es capaz, mientras que a los siguientes trabajadores les llegan bandejas llenas y vacías (cuanos más puestos se avanzan, más proporción de bandejas llenas y menos bandejas vacías que se pueden inducir). Si hay $S = 200$ bandejas, entonces el inductor

[1] Llegarían intercaladas con bandejas llenas.

en el primer puesto (en el sentido de la marcha) induce $S \cdot RATIO_1 = 70$ items. El segundo induciría $S \cdot RATIO_2 = 58$ items. Aplicando igual razonamiento, el tercer inductor coloca 46 items, y el cuarto, 20. Apreciamos entonces un comportamiento 'hipergeométrico' en la carga de trabajo de los inductores, que coincide con lo que se nos hizo saber que se observa en la realidad. Este es el motivo principal por el que se opta por este tipo de modelización. para ambas opciones.

Observación 2. *Por el momento se asumirá que los puestos de inducción se asignan a los inductores por orden. Esto significa que si, por ejemplo, se dispone de* 4 *puestos de inducción y* 3 *inductores, entonces estos ocuparán siempre los puestos* $1, 2$ *y* 3*, nunca el* 4*.*

Esta es una hipótesis considerada asumible (por Rubén) y que nos evita añadir más variables solo para facilitar que se puedan ocupar puestos salteados. Además, los valores de RATIO se espera que se calculen asumiendo que los puestos anteriores a uno dado (excepto en el primer puesto) están ocupados.

El principal motivo de usar las 'ratios' comentadas es que permite modelizar el carácter hipergeométrico de la carga de trabajo. Pero el otro gran motivo es que nos permitirá modelizar el 'desgaste' o la pérdida de productividad a lo largo del tiempo en función de la carga de trabajo. Esto también permite modelizar el papel de lo water-spiders de una forma un poco más detallada y realista.

Opción 1: Actualización por revolución

La primeira propuesta es que se reevalúen las capacidades inductoras en cada revolución. Es decir, en cada ciclo estaríamos calculando de nuevo cuantos items puede inducir cada trabajador. También reevaluaríamos la conveniencia o no de que un water-spider se cambie de puesto.

En este caso, se cambiaría el parámetro IAB (*induction ability*) del modelo por las variables $indab_{pr}$ donde p es el puesto de inducción y r la revolución. Se incluirían restricciones del tipo

$$indab_{pr} = S \cdot RATIO_p \qquad p = 1, ..., IPOST, \quad r \in REV \setminus \{0\} \tag{1}$$

Por supuesto, no cabe esperar que los valores anteriores sean enteros, por lo que habría que hacer una especie de redondeo. Una forma simple de resolverlo es mediante las siguientes restricciones.

$$\sum_{i \in REF} nind_{ir} \leq \sum_{p=1}^{IPOST} indab_{pr} \qquad r \in REV \setminus \{0\} \tag{2}$$

Lo anterior supone que a grandes rasgos se practica un rendondeo a la baja, pero al sumar todos los valores se cuenta con que pueden añadir uno o unos pocos items de margen, lo cual puede ser un buen acercamiento a este 'redondeo'.

Sobre esta base se puede plantear la incusión de un desgaste en la capacidad inductora a lo largo del tiempo. Al igual que con $RATIO$, se puede considerar un parámetro $FATIGUE$, también una proporción, que modelice este cansancio. En este sentido, las restricciones mostradas en (1) se reescribirían así:

$$\begin{gathered} indab_{pr} = FATIGUE_p \cdot indab_{p,r-1} \qquad p = 1, ..., IPOST, \quad r \in REV \setminus \{0, 1\} \\ indab_{p1} = S \cdot RATIO_p \qquad p = 1, ..., IPOST \end{gathered} \tag{3}$$

Para terminar con la modelización de la inducción (al margen de los water-spiders), podemos considerar que la capacidad inductora 'se resetea' cada cierto tiempo, y para ellos e pueden considerar dos posibilidades. En primer lugar, se puede hacer usando el parámetro $STRETCH^*$ que representa en que revoluciones se puede cambiar la configuración de trabajadores por rol. Otra opción sería considerar una serie de puntos temporales diferente, que represente un descanso, cambio de turno o cambio de labor. Para este caso las restricciones son muy simples. Si por ejemplo incluímos este 'reseteo' en las revoluciones de $STRETCH^*$, las restricciones de (3) dejarían de aplicar en $r \in STRETCH^*$ y en su lugar tendríamos las siguentes.

$$indab_{ps} = S \cdot RATIO_p \qquad p = 1, ..., IPOST, \quad s \in STRETCH^* \tag{4}$$

Como aclaración final, se están obviando en este boceto otras restricciones que regulan el comportamiento de las nuevas variables que estamos introduciendo. En este documento, como resumen que es, solo escribiremos aquellas que expliquen el comportamiento que queremos reflejar.

Opción 2: Actualización por etapas

Este enfoque es más simple de explicar, dado que utiliza la misma lógica que los cambios de configuración de trabajadores por rol, que se incluyó en la V1.5 del modelo a través del parámetro $STRETCH^*$.

En este caso la idea sería proponer un parámetro similar, por ejemplo $IJUMP^*$ (*induction jump*) que contiene las revoluciones en que se realizarán las reevaluaciones sobre la capacidad inductora. Al igual que con $STRETCH^*$, existirá otro parámetro $IJUMP$ indexada en REV y que en cada coordenada contiene la etapa a la que pertenece dicha revolución (donde las etapas significan el intervalo semicerrado entre dos revoluciones de $IJUMP^*$).

Por lo demás, se seguirían utilizando los parámetros $RATIO$ ya vistos, pero cuyo cálculo se adaptaría al tiempo que pasa entre dos elementos de $IJUMP$, que por cierto sería deseable que se distribuyesen de forma homogénea a lo largo de REV.

Así, utilizando las mismas variables $indab_{pr}$ ya vistas, así como $RATIO_p$ y $FATIGUE_p$, tendríamos las siguientes restricciones.

$$indab_{ps} = FATIGUE_p \cdot indab_{p,r-1} \qquad p = 1, ..., IPOST, \quad s \in IJUMP^* \tag{5}$$

$$indab_{p1} = S \cdot RATIO_p \qquad p = 1, ..., IPOST$$

Así, solo definimos estas variables en los elementos de $IJUMP^*$. Para contabilizar las inducciones totales en una revolución, la nueva restricción similar a (2) sería la siguiente.

$$\sum_{i=1}^{REF} nind_{ir} \leq \sum_{p=1}^{IPOST} indab_{p,IJUMP_r} \qquad r \in REV \setminus \{0\} \tag{6}$$

En este caso tendremos que durante ciertas revoluciones la capacidad de un inductor se mantiene constante, para luego dar un salto hacia abajo, que puede ser más o menos pronunciado en función de los valores de $FATIGUE$, pero la lógica es la misma en cada opción más allá del tratamiento del tiempo en el modelo.

Más allá de esta cuestión y del mismo modo que en la primera opción expuesta, se pueden añadir revoluciones concretas en que el cansancio se 'resetee'.

Modelización de los Water-Spiders

En este apartado se tratará de explicar como se prevee implementar la labor de los water-spiders en el modelo. De varias conversaciones con el tutor en empresa (Rubén) se deduce que es rol menos relevante de los cuatro en la modelización, así como el que menos trabajadores suele ocupar. Por tanto, es interesante ver que nivel de detalle se desea en el modelo y el coste computacional que conlleva (número de variables y restricciones adicionales).

Si tomamos un enfoque de máximos[2], la inclusión de los water-spiders en el modelo puede permitir la adición de las siguientes dinámicas:

- Que el aporte que los water-spiders realizan al inductor del puesto en que están se reduzca con el paso del tiempo. Es decir, como la labor del water-spider es ayudar en la rotura y lectura de la caja para ahorrar algo de tiempo al inductor, cabe esperar que tras un cierto tiempo ya haya roto varias cajas y 'adelantado' ese trabajo al inductor (por lo que su aporte se reduce mucho).
- Que la ayuda de un water-spider a un inductor pueda significar un cierto 'refresco' de la fatiga del mismo. si un water-spider está ayudando a un trabajador, podemos considerar que este se cansa menos o incluso que recupere algo de energía (traducida en el modelo con un parámetro opuesto al de $FATIGUE$).
- Que se tenga en cuenta la distancia que recorre el water-spider al cambiar de puesto, en forma de penalización. Considerar por tanto que no sea lo mismo si se está en el puesto 1 se cambia al puesto 3 o al 5. De las tres dinámicas esta sería quizás la de menor impacto en el modelo.

No se detalla ahora mismo la implementación concreta en el modelo, entre otras cuestiones porque depende de la opción que tomemos para trabajar con el tiempo en el problema. Aún así, la modelización es ciertamente similar a la explicada para la inducción, con la diferencia de tener en cuenta el eventual cambio de puesto de inducción.

[2]De modelizar todo lo posible dentro de la lógica establecida en el modelo.

Bibliografía

[1] Bazaraa MS, Jarvis JJ, Sherali HD (2004) Linear Programming and Network Flows. WileyInterscience.

[2] Boysen N, Briskorn D, Fedtke S, Schmickerath M (2019) Automated sortation conveyors: A survey from an operational research perspective. European Journal of Operational Research , Vol. 276, No. 3, p. 796-815. https://doi.org/10.1016/j.ejor.2018.08.014

[3] Boysen N, de Koster R, Füßler D (2021) The forgotten sons: Warehousing systems for brick-and-mortar retail chains. European Journal of Operational Research, Vol. 288, No. 2, p. 361-381, https://doi.org/10.1016/j.ejor.2020.04.058

[4] Boysen N, Stephan K, Schwerdfeger S (2024) Order consolidation in warehouses: The loop sorter scheduling problem. European Journal of Operational Research , Vol. 316, No. 2, p. 459-472. https://doi.org/10.1016/j.ejor.2024.02.042

[5] Bynum ML, Hackebeil GA, Hart WE, Laird CD, Nicholson BL, Siirola JD, Watson JP, Woodruff DL (2021) Optimization Modeling in Python, 3rd Edition, Springer, p.158-168. https://doi.org/10.1007/978-3-030-68928-5

[6] Chen T, Chen JC, Huang C, Chang P (2021) Solving the layout design problem by simulation-optimization approach–A case study on a sortation conveyor system. Simulation Modelling Practice and Theory , Vol. 106, p. 102192. https://doi.org/10.1016/j.simpat.2020.102192

[7] Diefenbach H, Grosse EH, Glock CH (2024) Human-and-cost-centric storage assignment optimization in picker-to-parts warehouses.European Journal of Operational Research , Vol. 315, No. 3, p. 1049-1068. https://doi.org/10.1016/j.ejor.2024.01.033

[8] Grambo, P, Mullick T, Furukawa T, Matoba M, Nasu Y (2019) Automatic Sorting-and-Holding for Stacking Heterogeneous Packages in Logistic Hubs. IFAC-PapersOnLine , Vol. 52, No. 10, p. 109-114. https://doi.org/10.1016/j.ifacol.2019.10.008

[9] Grznár P, Krajčovič M, Gola A, Dulina L, Furmannová B, Mozol Š, Plinta D, Burganová N, Danilczuk W, Svitek R (2021) The Use of a Genetic Algorithm for Sorting Warehouse Optimisation. Processes , Vol. 9, No. 7. https://doi.org/10.3390/pr9071197

[10] Liu Y, Ji S, Su Z, Guo D (2019) Multi-objective AGV scheduling in an automatic sorting system of an unmanned (intelligent) warehouse by using two adaptive genetic algorithms and a multi-adaptive genetic algorithm. PLOS ONE , Vol. 14, No. 12, p. 1-21. https://doi.org/10.1371/journal.pone.0226161

[11] Salazar JJ (2001) Programación Matemática.

[12] Tan Z, Li H, He X (2021) Optimizing parcel sorting process of vertical sorting system in e-commerce warehouse. Advanced Engineering Informatics , Vol. 48, p. 101279. https://doi.org/10.1016/j.aei.2021.101279

[13] Python Software Foundation (2001-2025) The Python Tutorial, https://docs.python.org/3/tutorial/index.html . Consultado o 21 de maio de 2025.