UNIVERSIDAD DE SALAMANCA

DEPARTAMENTO DE ESTADÍSTICA

DOCTORADO EN ESTADÍSTICA MULTIVARIANTE APLICADA

TESIS DOCTORAL

"ANÁLISIS DE TABLAS DE 3 VÍAS MEDIANTE EL DISEÑO E IMPLEMENTACIÓN DE ALGORITMOS QUE CALCULAN COMPONENTES ORTOGONALES DISJUNTOS EN EL MODELO PARAFAC Y EN LOS MODELOS TUCKER"

AUTOR: CARLOS MANUEL MARTÍN BARREIRO

DIRECTORAS: DRA. MARÍA PURIFICACIÓN GALINDO VILLARDÓN

DRA. ANA MARÍA MARTÍN CASADO

2021

ANÁLISIS DE TABLAS DE 3 VÍAS MEDIANTE EL DISEÑO E IMPLEMENTACIÓN DE ALGORITMOS QUE CALCULAN COMPONENTES ORTOGONALES DISJUNTOS EN EL MODELO PARAFAC Y EN LOS MODELOS TUCKER

Memoria que, para optar al Grado de Doctor por el Departamento de Estadística de la Universidad de Salamanca, presenta:

Carlos Manuel Martín Barreiro

Salamanca, 2021

VNiVERSiDAD D SALAMANCA
CAMPUS DE EXCELENCIA INTERNACIONAL

DEPARTAMENTO DE ESTADÍSTICA

DRA. MARÍA PURIFICACIÓN GALINDO VILLARDÓN

CATEDRÁTICA DEL DEPARTAMENTO DE ESTADÍSTICA DE LA

UNIVERSIDAD DE SALAMANCA

Y

DRA. ANA MARÍA MARTÍN CASADO

PROFESORA TITULAR DEL DEPARTAMENTO DE ESTADÍSTICA DE LA

UNIVERSIDAD DE SALAMANCA

CERTIFICAN:

Que Carlos Manuel Martín Barreiro ha realizado en el Departamento de Estadística de la Universidad de Salamanca, bajo su dirección, el trabajo que para optar al Grado de Doctor, presenta con el título: **ANÁLISIS DE TABLAS DE 3 VÍAS MEDIANTE EL DISEÑO E IMPLEMENTACIÓN DE ALGORITMOS QUE CALCULAN COMPONENTES ORTOGONALES DISJUNTOS EN EL MODELO PARAFAC Y EN LOS MODELOS TUCKER** y para que conste, firman el presente certificado en Salamanca, a 9 de septiembre del 2021.

Dra. María Purificación Galindo Villardón Dra. Ana María Martín Casado

AGRADECIMIENTOS

Primero a DIOS por la vida, por la salud, por llenarme de tantas bendiciones.

A mi supervisora, la Doctora María Purificación Galindo Villardón, por la amistad, por las enseñanzas, por la paciencia, por los llamados de atención, por la motivación y por darme la oportunidad de ser parte de la gran familia de la Universidad de Salamanca.

A mi supervisora, la Doctora Ana María Martín Casado, por toda su importante ayuda y colaboración en el desarrollo de mi tesis.

Al profesor Doctor Víctor Leiva Sánchez de la Pontificia Universidad Católica de Valparaíso, Chile, por su valioso aporte en mi trabajo doctoral, pero principalmente por su amistad y por sus acertados consejos.

A mis padres Eva Rosalía y Carlos Julio, por su apoyo, por sus consejos, porque hoy puedo estar aquí por ustedes.

A mi abuela Alicia y a mi tía Maruja, madres que Dios puso en mi camino.

A mi bella esposa Zoraya y a mi princesita Ali, las amo, son mi motor, me llenan de fortaleza, son mi felicidad.

A mis tíos Edmundo, Lino, Joel, Ricardo, Gustavo y Jorge quienes me han ayudado a lo largo de mi vida, por sus consejos y su valioso apoyo. Parte de mi formación se las debo a ustedes.

A mis tías Rita Edith y Mercedes, las quiero mucho, siempre han estado conmigo, en todo momento.

A mi hermana Jael, y a mis hermanos José y Julio por sus consejos, por su apoyo, porque siempre he podido contar con ustedes.

A mis primos Nico y Adrián. A mis amigos y compañeros John Ramírez, Greibin Villegas, Xavier Cabezas y Joseph Páez. Hermanos que Dios puso en mi camino.

A mi linda Mayeve.

ÍNDICE GENERAL

ÍNDICE DE ALGORITMOS

ÍNDICE DE FIGURAS

NOTACIÓN

$\mathbb{R}^I$	Conjunto de todas las I-tuplas con entradas reales
$M_{I\times J}$	Conjunto de todas las matrices con entradas reales de tamaño $I \times J$
$T_{I\times J\times K}$	Conjunto de todos los tensores o tablas de 3 vías con entradas reales de tamaño $I \times J \times K$
$\boldsymbol{a}$	Vector
$\boldsymbol{a} \in \mathbb{R}^I$	Vector de tamaño I
$\|\boldsymbol{a}\|$	Norma euclídea del vector $\boldsymbol{a}$
$\boldsymbol{X}$	Matriz o tabla de 2 vías
$\boldsymbol{X}^T$	Transpuesta de la matriz $\boldsymbol{X}$
$\boldsymbol{X}^+$	Inversa generalizada de la matriz $\boldsymbol{X}$
$\boldsymbol{X}^{-1}$	Inversa de la matriz $\boldsymbol{X}$
$\|\boldsymbol{X}\|$	Norma de Frobenius aplicada a la matriz $\boldsymbol{X}$
$\boldsymbol{X} \in M_{I\times J}$	$\boldsymbol{X}$ es una tabla de 2 vías o matriz de tamaño $I \times J$
$\boldsymbol{X} = (x_{ij})$	x_{ij} es el elemento genérico de la matriz $\boldsymbol{X}$ de tamaño $I \times J$
$\hat{\boldsymbol{X}}$	Una matriz que es aproximación de la tabla de 2 vías $\boldsymbol{X}$ mediante algún método de descomposición matricial
$\underline{\mathbf{X}}$	Tensor o tabla de 3 vías
$\underline{\mathbf{X}} \in T_{I\times J\times K}$	$\underline{\mathbf{X}}$ es un tensor o una tabla de 3 vías de tamaño $I \times J \times K$
$\underline{\mathbf{X}} = (x_{ijk})$	x_{ijk} es el elemento genérico del tensor $\underline{\mathbf{X}}$ de tamaño $I \times J \times K$
$\|\underline{\mathbf{X}}\|$	Norma de Frobenius de la tabla de 3 vías $\underline{\mathbf{X}}$
$\underline{\mathbf{G}}$	Core en los modelos Tucker
$\hat{\mathbf{X}}$	Un tensor que es aproximación de la tabla de 3 vías $\underline{\mathbf{X}}$ mediante algún método de descomposición tensorial

$\cdot_{ex}$	Producto externo
$Vec(\boldsymbol{A})$	Operación Vec aplicada a la matriz $\boldsymbol{A}$
$\otimes$	Producto de Kronecker
$\odot$	Producto de Khatri-Rao
$\circ$	Producto de Hadamard
Δ	Producto tensorial
$\stackrel{\text{def}}{=}$	Definición matemática formal
$\forall i$	Para toda i
$\exists i$	Existe al menos una i
$\exists!\, i$	Existe una única i

ACRÓNIMOS

ALS	**A**LTERNATING **L**EAST **S**QUARES
CBPSO-DC	**C**ONSTRAINT **B**INARY **P**ARTICLE **S**WARM **O**PTIMIZATION - **D**ISJOINT **C**OMPONENTS
C-HULL	**C**ONVEX – **HULL**
DIFFIT	**DIF**FERENCE IN **FIT**
DPCA	**D**ISJOINT **P**RINCIPAL **C**OMPONENT **A**NALYSIS
DTLD	**D**IRECT **T**RI**L**INEAR **D**ECOMPOSITION
NCH	**N**UMERICAL **C**ONVEX **H**ULL
PARAFAC	**PARA**LLEL **FAC**TOR ANALYSIS
PCA	**P**RINCIPAL **C**OMPONENT **A**NALYSIS
PSO	**P**ARTICLE **S**WARM **O**PTIMIZATION
SVD	**S**INGULAR **V**ALUE **D**ECOMPOSITION

RESUMEN

Al realizar un análisis de componentes en tablas de 2 vías (tensores de orden 2 o matrices, por simplicidad), los componentes ortogonales disjuntos permiten obtener matrices de cargas de estructura simple, es decir, matrices de cargas de fácil interpretación. El principal objetivo de esta tesis es extender el uso y el éxito de los componentes ortogonales disjuntos al análisis de componentes para tablas de 3 vías (tensores de orden 3 o simplemente tensores).

Para alcanzar tal objetivo, se han diseñado e implementado 4 algoritmos heurísticos que permiten el cálculo de componentes ortogonales disjuntos en las matrices de cargas de los modelos multivariantes utilizados para realizar un análisis de componentes en tablas de 3 vías. Además se proponen procedimientos para la utilización de estos algoritmos.

El primer algoritmo propuesto, de nombre CBPSO-ParafacALS, permite el cálculo de componentes ortogonales disjuntos en el modelo PARAFAC. Los restantes 3 algoritmos que se proponen, denominados CBPSO-TuckALS3, CBPSO-TuckALS2 y CBPSO-TuckALS1, permiten el cálculo de componentes ortogonales disjuntos en los modelos Tucker3, Tucker2 y Tucker1, respectivamente. Para el caso particular del modelo PARAFAC, se presenta y se discute el tan conocido “problema de degeneración”. Se muestra en estudios computacionales que el algoritmo CBPSO-ParafacALS permite también analizar una tabla de 3 vías, acorde al modelo PARAFAC, cuando el problema de degeneración está presente.

Los 4 algoritmos emplean los cortes frontales, horizontales y verticales de la tabla de 3 vías que se analiza, para poder realizar el cálculo de los componentes ortogonales disjuntos. Además, estos algoritmos heurísticos están basados en un algoritmo de mínimos cuadrados alternantes y en un algoritmo de optimización por enjambre de partículas que es binario y con restricciones.

Se han llevado a cabo experimentos computacionales que ilustran el principal beneficio de los algoritmos propuestos: la obtención de matrices de cargas de estructura simple que facilitan el análisis y la interpretación de resultados en los modelos PARAFAC, Tucker3, Tucker2 y Tucker1 utilizados para el estudio multivariante de tablas de 3 vías. El empleo de componentes ortogonales disjuntos, sin embargo, sufre de pérdida de fit en el modelo. Este es un punto muy importante que el analista de los datos o investigador debe considerar al momento de emplear componentes ortogonales disjuntos.

ABSTRACT

When performing component analysis on two-way tables (tensors of order 2 or matrices, for simplicity), the disjoint orthogonal components allow to obtain simple structure loading matrices, which means interpretable loading matrices. The main purpose of this thesis is to expand the use and success of disjoint orthogonal components to the analysis of components for three-way tables (tensors of order 3 or simply tensors).

In order to reach that goal, four heuristic algorithms have been designed and implemented that allow the calculation of disjoint orthogonal components on the loading matrices of the multivariate models used to make a component analysis of three-way tables. On top of that, for the use of these four heuristic algorithms procedures are proposed.

The first proposed algorithm, named CBPSO-ParafacALS, allows the calculation of disjoint orthogonal components in the PARAFAC model. The other three proposed algorithms, named CBPSO-TuckALS3, CBPSO-TuckALS2 and CBPSO-TuckALS1, allow the calculation of disjoint orthogonal components on the Tucker3, Tucker2 and Tucker1 models, respectively. For the particular case of the PARAFAC model, the well-known "degeneracy problem" is presented and discussed. It is shown, in computational studies, that the CBPSO-ParafacALS algorithm also allows the analysis of a three-way table, according to the PARAFAC model, when the "degeneracy" problem is present.

The four algorithms use frontal, horizontal and vertical slices of the three-way table under analysis, in order to calculate the disjoint orthogonal components. Furthermore, these heuristic algorithms are based on an alternating least-squares algorithm and an optimized by a swarm of particles algorithm that is binary and with constraints.

Computational experiments have been carried out that highlight the main benefit of the proposed algorithms: obtaining simple structure loading matrices that facilitate the analysis and interpretation of the results on the PARAFAC, Tucker3, Tucker2 and Tucker1 models used for the multivariate study of three-way tables. However, the use of disjoint orthogonal components undergoes a loss of fit in the model. This is a very important point that the data analyst or researcher must consider when using disjoint orthogonal components.

INTRODUCCIÓN

El objetivo de los métodos multivariantes (entre ellos el análisis de componentes principales, PCA por sus siglas en inglés) es el de ofrecer una descripción simplificada de un conjunto de datos multidimensionales. Al considerar dicho conjunto, los individuos y las variables pueden ser numerosos. Para realizar este tipo de análisis el empleo de matrices es fundamental, y existe en la literatura de la Estadística Multivariante toda una teoría detrás, que incluye modelos matemáticos y algoritmos.

En la teoría matemática de los tensores, un vector es un tensor de orden 1, una matriz es un tensor de orden 2, mientras que un arreglo o tabla de 3 vías es un tensor de orden 3. Los tensores son generalizaciones de vectores y matrices (ver Kolda 2001 y Kolda and Bader 2009).

Para realizar un PCA se trabaja con una matriz de datos de dos vías y dos modos. El primer modo es el de los individuos, mientras que el segundo modo es el de las variables. El término "vía" hace referencia al orden del tensor, y el término "modo" se refiere a un conjunto de entidades diferentes, tal como se afirma en Giordani et al. 2014. Si se considera entonces una vía adicional, por ejemplo, la ubicación, la situación o el tiempo en el cual las variables están siendo medidas, se tendría una "tabla de 3 vías" con un modo adicional en la vía agregada (ver Kiers and Mechelen 2001). Típicamente, en la tercera vía se encuentra el modo de los tiempos, las situaciones o las ubicaciones. Existen otros métodos multivariantes, como el escalamiento multidimensional (Shepard 1962, Kruskal 1964a, Kruskal 1964b), donde un mismo modo se encuentra por ejemplo en las dos vías de una matriz. En Torgerson 1952 se propuso el primer procedimiento formal para el escalamiento multidimensional en tablas de 2 vías. Para el escalamiento multidimensional en tablas de 3 vías ver Borg and Groenen 2005 y Husson and Pages 2006.

La descomposición tensorial tiene sus orígenes en el siglo XX (ver Hitchcock 1927). A inicios de los años 60, en Tucker 1963 y luego en Tucker 1966, se incorpora el uso de tensores, con los "modelos Tucker", en el contexto multivariante. Posteriormente, a principio de la década de los 70, Harshman 1970 y Carroll and Chang 1970, proponen el "modelo PARAFAC" (ver Kolda and Bader 2009).

En el análisis de componentes para tablas de 3 vías, se intenta identificar patrones o tendencias en el espacio de los individuos, en el espacio de las variables y en el espacio de los tiempos (situaciones o ubicaciones) como se afirma en Kroonenberg and de Leeuw 1980. Además, se debe encontrar un modelo que represente bien a los datos y que sea fácil de interpretar en términos de las interacciones entre las entidades de los 3 modos (ver Kiers and Van Mechelen 2001). En otras palabras, los modelos persiguen representar los datos originales en espacios de dimensión reducida para la detección de patrones y brindan facilidades para cuantificar las interacciones entre todos sus modos.

Cada espacio de baja dimensión es generado por ejes principales o componentes, los cuales maximizan su varianza, y son además combinaciones lineales de las entidades originales (ver Goossens et al. 2002). Dentro del espacio de cada modo, la interpretación de los componentes se hace de acuerdo a los escalares (cargas) de las correspondientes combinaciones lineales. En algunas ocasiones la interpretación puede ser difícil, complicada, y el analista de los datos puede caer en errores o en una caracterización inadecuada de los componentes. Es deseable que cada componente, o eje principal, tenga unas pocas entidades que contribuyan de forma importante a la variabilidad del componente.

Con la intención de mejorar la interpretación, existen técnicas de escalado y técnicas de rotación (se pueden ver aplicaciones en Giordani et al. 2014) que, manteniendo el fit del modelo, proporcionan matrices de cargas con estructura simple. Es importante indicar que estas técnicas no garantizan la obtención de una estructura que haga sencilla la interpretación. Por tal razón, se han desarrollado otros métodos que permiten una descomposición tensorial en la cual, los componentes tienen algunas cargas nulas (unos pocos escalares en cada una de las combinaciones lineales son iguales a cero), lo que facilita la interpretación y el análisis en una tabla de 3 vías, pero en el modelo se pierde algo de fit. Algunos de estos métodos, conocidos como técnicas “sparse” aplicadas a los tensores, los podemos ver en Papalexakis et al. 2012 para el modelo PARAFAC y Perros et al. 2015 para los modelos Tucker. En Sun et al. 2017 se usa una descomposición tensorial tipo sparse mediante un procedimiento de selección de entidades. Se propone un algoritmo en Yokota and Cichocki 2014 para el modelo Tucker imponiendo restricciones de ortogonalidad para las matrices de cargas y restricciones sparse para el “core”. Para el uso de la técnica “sparse” en matrices ver Zou, Hastie and Tibshirani 2006.

A diferencia de los métodos sparse, en los métodos “disjoint” aplicados a matrices (ver Vichi and Saporta 2009, Macedo and Freitas 2015) se busca una descomposición matricial en la cual la matriz de cargas tenga, por cada fila, una única entrada distinta de cero y, por cada columna, tenga al menos una entrada de la matriz distinta de cero. De esta manera es posible obtener matrices de cargas de estructura simple donde la interpretación se facilita. En Ramirez-Figueroa et al. 2021 podemos ver un método disjoint para matrices de datos que realiza el cálculo de componentes ortogonales disjuntos en la matriz de cargas. El algoritmo propuesto en dicha investigación está basado en una optimización por enjambre de partículas (PSO, por sus siglas en inglés) binaria y con restricciones.

En conclusión, después de realizar una búsqueda exhaustiva en la literatura científica, se puede afirmar que para realizar un análisis de componentes en tablas de 3 vías, con los modelos Tucker y con el modelo PARAFAC, existen las siguientes técnicas para intentar construir matrices de cargas interpretables:

- Escalado (modelos Tucker y modelo PARAFAC)
- Rotaciones (modelos Tucker)
- Restricciones de ortogonalidad (modelos Tucker y modelo PARAFAC)
- Sparse (modelos Tucker y modelo PARAFAC)

No se ha propuesto ningún algoritmo para el cálculo de componentes disjuntos ni en los modelos Tucker ni en el modelo PARAFAC. El cálculo de componentes disjuntos se realiza actualmente únicamente en tablas de 2 vías. Los componentes disjuntos en tablas de 2 vías presentan beneficios importantes ya que facilitan la interpretación de una matriz de cargas. Esta tesis se ha escrito con la intención de llevar los componentes disjuntos a las tablas de 3 vías, para facilitar la interpretación de las matrices de cargas al realizar un análisis de componentes. El objetivo principal de esta tesis es por tanto introducir algoritmos nuevos para el cálculo de componentes disjuntos en los modelos Tucker y en el modelo PARAFAC.

Para el cálculo de componentes disjuntos en tablas de 3 vías se deben resolver modelos matemáticos de optimización de una alta complejidad computacional. Por tal motivo, en esta tesis se proponen algoritmos heurísticos para realizar el mencionado cálculo. Cabe indicar que para calcular componentes disjuntos en tablas de 2 vías se utilizan algoritmos heurísticos que podemos ver en Vichi and Saporta 2009, Macedo and Freitas 2015, Ferrara et al. 2016 y Ramirez-Figueroa et al. 2021.

El trabajo descrito en esta tesis debe cumplir con los siguientes objetivos:

1.- Implementar los siguientes algoritmos existentes usando R, R.NET y el lenguaje de programación C#.NET:

a) Algoritmo de Vichi and Saporta 2009 y algoritmo CBPSO-DC de Ramirez-Figueroa et al. 2021 para el cálculo de componentes ortogonales disjuntos en tablas de 2 vías.

b) Algoritmo ParafacALS y algoritmo TuckALS3 para el cálculo de componentes en el modelo PARAFAC y en el modelo Tucker3, respectivamente.

c) Algoritmos para el preprocesado de tablas de 3 vías.

2.- Diseñar e implementar algoritmos heurísticos, usando R, R.NET y el lenguaje de programación C#.NET, que permitan calcular componentes ortogonales disjuntos en el modelo PARAFAC y en los modelos Tucker: Tucker3, Tucker2 y Tucker1.

3.- Realizar pruebas computacionales (experimentos) con tablas de 3 vías, tanto con datos simulados como con datos reales, para evaluar el rendimiento de los algoritmos propuestos.

4.- Diseñar un procedimiento para el uso de los algoritmos que calculan componentes ortogonales disjuntos en tablas de 3 vías.

El primer objetivo nace de la necesidad de conocer cómo se calculan componentes ortogonales disjuntos en tablas de 2 vías para estudiar la posible extensión de dicho cálculo a tablas de 3 vías. Por otro lado, es también importante realizar el cálculo clásico de componentes (junto con el preprocesamiento correspondiente) en tablas de 3 vías para posteriormente comparar esos resultados con los que se obtienen al emplear componentes ortogonales disjuntos.

El segundo objetivo es la principal motivación de esta tesis, que es el de proponer algoritmos que calculen componentes ortogonales disjuntos en el modelo PARAFAC y en los modelos Tucker para facilitar la interpretación durante el estudio de una tabla de 3 vías. Debido a la complejidad computacional de los modelos de optimización correspondientes se han diseñado e implementado algoritmos heurísticos. Se escogió además diseñar los algoritmos heurísticos propuestos haciendo uso de la optimización por enjambre de partículas debido a que el cálculo de componentes en cualquiera de los modelos es un problema de optimización de tipo continuo y los algoritmos PSO presentan buenos resultados en ese tipo de problemas (ver Ramirez-Figueroa et al. 2021).

El tercer objetivo tiene como finalidad mostrar, mediante experimentos con un ordenador, que cuando se utilizan los algoritmos heurísticos propuestos se obtienen soluciones interpretables, lo que facilita el estudio de una tabla de 3 vías. Es decir, este objetivo permite poner en evidencia los beneficios que se obtienen al calcular componentes ortogonales disjuntos en el modelo PARAFAC y en los modelos Tucker.

El cuarto y último objetivo permite además proponer procedimientos que sirvan como una guía para el investigador respecto de la utilización de los algoritmos propuestos, en otras palabras, diseñar procedimientos, paso a paso, que recomienden el momento oportuno para el cálculo de componentes ortogonales disjuntos durante el análisis de una tabla de 3 vías.

En esta tesis se proponen 4 algoritmos heurísticos para el cálculo de componentes ortogonales disjuntos en el modelo PARAFAC y en los modelos Tucker: Tucker3, Tucker2, Tucker1. Respectivamente, estos algoritmos han sido denominados CBPSO-ParafacALS, CBPSO-TuckALS3, CBPSO-TuckALS2 y CBPSO-TuckALS1. Estos algoritmos son el resultado de un esfuerzo por incorporar los componentes ortogonales disjuntos en tablas de 3 vías, debido a los beneficios en interpretación observados en tablas de 2 vías. La razón de ser de esta tesis por ende es extender el uso y el éxito de los componentes ortogonales disjuntos a las tablas de 3 vías. De la mano con estos algoritmos se han desarrollado procedimientos que recomiendan las condiciones bajo las cuales se deben calcular componentes ortogonales disjuntos al realizar un análisis de componentes en tablas de 3 vías.

Esta tesis está organizada de la siguiente manera:

En el **CAPÍTULO UNO** "INTRODUCCIÓN A LOS TENSORES Y OPERACIONES ALGEBRAICAS DE INTERÉS" se muestran las definiciones y propiedades de algunas operaciones matriciales no convencionales que son utilizadas en los modelos matemáticos para el análisis de componentes en tablas de 3 vías.

En el **CAPÍTULO DOS** "MODELOS MULTIVARIANTES PARA EL ANÁLISIS DE COMPONENTES EN TABLAS DE 3 VÍAS" se presenta el álgebra y también los algoritmos necesarios para el estudio del modelo PARAFAC y los modelos Tucker. Estos modelos son el punto de inicio para el cálculo de los componentes ortogonales disjuntos en tablas de 3 vías.

En el **CAPÍTULO TRES** "COMPONENTES ORTOGONALES DISJUNTOS EN TABLAS DE 3 VÍAS" se definen los componentes ortogonales disjuntos en tablas de 3 vías. Se presentan todos los modelos matemáticos de optimización que se deben resolver para el cálculo de componentes ortogonales disjuntos. Se incluye además una propuesta procedimental respecto del uso de los componentes ortogonales disjuntos en tablas de 3 vías, es decir, bajo qué condiciones se recomienda el cálculo de este tipo especial de componentes en tablas de 3 vías.

En el **CAPÍTULO CUATRO** "ALGORITMOS HEURÍSTICOS PARA EL CÁLCULO DE COMPONENTES ORTOGONALES DISJUNTOS EN TABLAS DE 3 VÍAS" se define qué es un algoritmo heurístico y se discuten algunas características de este tipo de algoritmos. Posteriormente se muestra todo el detalle de los algoritmos heurísticos propuestos.

Finalmente, en el **CAPÍTULO CINCO** "APLICACIÓN DE LOS ALGORITMOS HEURÍSTICOS PROPUESTOS A DATOS REALES Y SIMULADOS" se muestran las pruebas computacionales que se han llevado a cabo y que ilustran los beneficios de interpretación de los algoritmos propuestos, tanto con datos simulados como con datos reales tomados de artículos científicos.

CAPÍTULO UNO: "INTRODUCCIÓN A LOS TENSORES Y OPERACIONES ALGEBRAICAS DE INTERÉS"

En este capítulo se hace una breve introducción a los tensores. Además se presentan las definiciones de algunas operaciones matriciales no convencionales que se utilizan en los modelos multivariantes para el análisis de componentes en tablas de 3 vías (ver Smilde et al. 2004, Schott 1997).

El producto de Kronecker se usa en los 3 modelos Tucker: Tucker3, Tucker2 y Tucker1, mientras que el producto de Khatri-Rao y el producto de Hadamard son utilizados en el modelo PARAFAC. El producto externo y la operación Vec también juegan un papel importante en los modelos antes mencionados. Además, el producto tensorial se utiliza también en los modelos Tucker y en el modelo PARAFAC. Por cada operación algebraica se incluye:

- Definición
- Estructura funcional
- Algunas propiedades relevantes
- Ejemplo ilustrativo
- Algoritmo

En el caso puntual de los algoritmos, estos están propuestos con la finalidad de que el lector los pueda implementar en cualquier lenguaje de programación. En esta tesis todas las implementaciones se han efectuado en el lenguaje de programación C#.NET y en el software estadístico R. Los algoritmos heurísticos que se proponen en este trabajo para el cálculo de componentes ortogonales disjuntos en tablas de 3 vías utilizan las operaciones detalladas en este capítulo.

1.1 INTRODUCCIÓN A LOS TENSORES

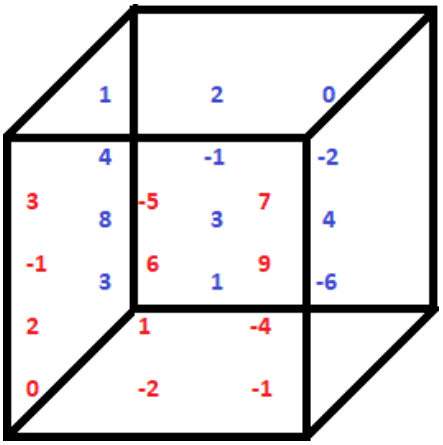
Un tensor en matemáticas es una entidad algebraica que pertenece a un espacio vectorial en cuya estructura existe al menos un elemento. El orden de un tensor está dado por el número de índices que se necesitan para especificar un elemento del tensor. Los tensores son una generalización o una extensión de los escalares, vectores y matrices (ver Smilde et al. 2004). Un número real es un tensor de orden 0, por otro lado un vector $\boldsymbol{x} \in \mathbb{R}^I$ o I-tupla es un tensor de orden 1 y finalmente una matriz $\boldsymbol{X}$ de tamaño $I \times J$ ($\boldsymbol{X} \in M_{I\times J}$) es un tensor de orden 2.

Una tabla de 3 vías (tensor de orden 3) es un arreglo denotado $\underline{\mathbf{X}}$ de tamaño $I \times J \times K$ que típicamente, en el análisis multidimensional, contiene observaciones sobre I individuos de J variables en K situaciones o tiempos. Usaremos $T_{I\times J\times K}$ para representar el conjunto que contiene todas las tablas de 3 vías con entradas reales. En los tensores, un modo está definido como un conjunto de entidades. Por tanto, $\underline{\mathbf{X}} \in T_{I\times J\times K}$ tiene el modo de individuos o modo-A (primer modo), el modo de variables o modo-B (segundo modo) y el modo de situaciones (tiempos) o modo-C (tercer modo).

El elemento genérico $x_{ijk} \in \underline{\mathbf{X}}$ almacena la medida del individuo $i \in \{1, \dots, I\}$ en la variable $j \in \{1, \dots, J\}$ y en la situación (tiempo) $k \in \{1, \dots, K\}$.

Cualquier tabla de 3 vías $\underline{\mathbf{X}}$ puede ser convertida en una matriz (tensor de orden 2) mediante un proceso de transformación conocido como “matricization” (en inglés). En esta tesis se usan 3 tipos de transformaciones: la conversión modo-A que genera una matriz $\boldsymbol{X_A}$ de tamaño $I \times JK$, la conversión modo-B que genera una matriz $\boldsymbol{X_B}$ de tamaño $J \times IK$ y la conversión modo-C que genera una matriz $\boldsymbol{X_C}$ de tamaño $K \times IJ$. Estas supermatrices están definidas en Kolda and Bader 2009, donde $\boldsymbol{X_A}$, $\boldsymbol{X_B}$ y $\boldsymbol{X_C}$ son conocidas como las matrices de cortes frontales, horizontales y verticales de $\underline{\mathbf{X}}$, respectivamente.

Como una ilustración, considere la tabla de 3 vías $\underline{\mathbf{X}}$ de tamaño $4 \times 3 \times 2$ que se muestra a continuación:



Al aplicar el proceso de generación de las supermatrices (matricization) obtenemos:

$$\boldsymbol{X_A} = \begin{pmatrix} 3 & -5 & 7 & 1 & 2 & 0 \\ -1 & 6 & 9 & 4 & -1 & -2 \\ 2 & 1 & -4 & 8 & 3 & 4 \\ 0 & -2 & -1 & 3 & 1 & -6 \end{pmatrix} \in M_{4\times 6}$$

$$\boldsymbol{X_B} = \begin{pmatrix} 3 & -1 & 2 & 0 & 1 & 4 & 8 & 3 \\ -5 & 6 & 1 & -2 & 2 & -1 & 3 & 1 \\ 7 & 9 & -4 & -1 & 0 & -2 & 4 & -6 \end{pmatrix} \in M_{3\times 8}$$

$$\boldsymbol{X_C} = \begin{pmatrix} 3 & -1 & 2 & 0 & -5 & 6 & 1 & -2 & 7 & 9 & -4 & -1 \\ 1 & 4 & 8 & 3 & 2 & -1 & 3 & 1 & 0 & -2 & 4 & -6 \end{pmatrix} \in M_{2\times 12}$$

Para generar la supermatriz $\boldsymbol{X_A}$ de "cortes frontales", se propone el **Algoritmo 1.1.1**:

Algoritmo 1.1.1: Generación de supermatriz de cortes frontales

---

Datos de entrada: $\underline{\mathbf{X}} \in T_{I\times J\times K}$
Inicio
    A partir de $\underline{\mathbf{X}}$ obtener la terna: $(I \quad J \quad K)$
    Construir la matriz $\boldsymbol{X_A}$ de tamaño $I \times JK$
    $Col \leftarrow 1$
    Para cada k desde 1 hasta K
        Para cada j desde 1 hasta J
            Para cada i desde 1 hasta I
                $\boldsymbol{X_A}[i, Col] \leftarrow \underline{\mathbf{X}}[i, j, k]$
            Fin del "Para cada i"
            $Col \leftarrow Col + 1$
        Fin del "Para cada j"
    Fin del "Para cada k"
Fin
Datos de salida: $\boldsymbol{X_A}$

Para generar la supermatriz $\boldsymbol{X_B}$ de "cortes horizontales", se propone el **Algoritmo 1.1.2**:

Algoritmo 1.1.2: Generación de supermatriz de cortes horizontales

---

Datos de entrada: $\underline{\mathbf{X}} \in T_{I\times J\times K}$
Inicio
    A partir de $\underline{\mathbf{X}}$ obtener la terna: $(I \quad J \quad K)$
    Construir la matriz $\boldsymbol{X_B}$ de tamaño $J \times IK$
    $Col \leftarrow 1$
    Para cada k desde 1 hasta K
        Para cada i desde 1 hasta I
            Para cada j desde 1 hasta J
                $\boldsymbol{X_B}[j, Col] \leftarrow \underline{\mathbf{X}}[i, j, k]$
            Fin del "Para cada j"
            $Col \leftarrow Col + 1$
        Fin del "Para cada i"
    Fin del "Para cada k"
Fin
Datos de salida: $\boldsymbol{X_B}$

Para generar la supermatriz $\boldsymbol{X_C}$ de "cortes verticales", se propone el **Algoritmo 1.1.3**:

Algoritmo 1.1.3: Generación de supermatriz de cortes verticales

---

Datos de entrada: $\underline{\mathbf{X}} \in T_{I\times J\times K}$
Inicio
    A partir de $\underline{\mathbf{X}}$ obtener la terna: $(I \quad J \quad K)$
    Construir la matriz $\boldsymbol{X_C}$ de tamaño $K \times IJ$
    $Col \leftarrow 1$
    Para cada j desde 1 hasta J
        Para cada i desde 1 hasta I
            Para cada k desde 1 hasta K
                $\boldsymbol{X_C}[k, Col] \leftarrow \underline{\mathbf{X}}[i, j, k]$
            Fin del "Para cada k"
            $Col \leftarrow Col + 1$
        Fin del "Para cada i"
    Fin del "Para cada j"
Fin
Datos de salida: $\boldsymbol{X_C}$

Las 3 supermatrices juegan un papel muy importante en las ecuaciones que gobiernan los modelos matemáticos para el análisis de componentes de tablas de 3 vías.

1.2 PRODUCTO EXTERNO

Considere los vectores $\boldsymbol{a} \in \mathbb{R}^I$ y $\boldsymbol{b} \in \mathbb{R}^J$. Se define el producto externo entre $\boldsymbol{a}$ y $\boldsymbol{b}$, denotado $\boldsymbol{a} \cdot_{ex} \boldsymbol{b}$, como:

$$\boldsymbol{a} \cdot_{ex} \boldsymbol{b} \stackrel{\text{def}}{=} \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_I \end{pmatrix} \cdot_{ex} \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_J \end{pmatrix} = \boldsymbol{a}\boldsymbol{b}^T = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_I \end{pmatrix} \begin{pmatrix} b_1 & b_2 & \cdots & b_J \end{pmatrix}$$

$$\Rightarrow \boldsymbol{a} \cdot_{ex} \boldsymbol{b} = \begin{pmatrix} a_1b_1 & a_1b_2 & \cdots & a_1b_J \\ a_2b_1 & a_2b_2 & \cdots & a_2b_J \\ \vdots & \vdots & \vdots & \vdots \\ a_Ib_1 & a_Ib_2 & \cdots & a_Ib_J \end{pmatrix}$$

No necesariamente los vectores deben tener el mismo tamaño. El resultado del producto externo $\boldsymbol{a} \cdot_{ex} \boldsymbol{b}$ es una matriz de tamaño $I \times J$ de rango uno. Si los vectores tienen el mismo tamaño, el resultado del producto externo es una matriz cuadrada. El producto externo tiene la siguiente estructura funcional:

$$\cdot_{ex}: \mathbb{R}^I \times \mathbb{R}^J \to M_{IJ}$$
$$(\boldsymbol{a}, \boldsymbol{b}) \mapsto \boldsymbol{a} \cdot_{ex} \boldsymbol{b}$$

Los elementos del conjunto $M_{I \times J}$ son todas las matrices con entradas reales de tamaño $I \times J$. Es de observar entonces que $\forall \boldsymbol{a} \in \mathbb{R}^I \; \forall \boldsymbol{b} \in \mathbb{R}^J$: $\boldsymbol{a} \cdot_{ex} \boldsymbol{b} \in M_{I \times J}$.

Se procede a hacer una ilustración del producto externo. Suponga que $\boldsymbol{a} = \begin{pmatrix} 4 \\ -2 \\ 3 \end{pmatrix} \in \mathbb{R}^3$ y $\boldsymbol{b} = \begin{pmatrix} 7 \\ 1 \end{pmatrix} \in \mathbb{R}^2$. Luego:

$$\boldsymbol{a} \cdot_{ex} \boldsymbol{b} = \begin{pmatrix} 4 \\ -2 \\ 3 \end{pmatrix} \cdot_{ex} \begin{pmatrix} 7 \\ 1 \end{pmatrix} = \boldsymbol{a}\boldsymbol{b}^T = \begin{pmatrix} 4 \\ -2 \\ 3 \end{pmatrix} \begin{pmatrix} 7 & -1 \end{pmatrix} = \begin{pmatrix} 28 & -4 \\ -14 & 2 \\ 21 & -3 \end{pmatrix} \in M_{3 \times 2}$$

A continuación se propone el **Algoritmo 1.2** para calcular el producto externo entre 2 vectores:

Algoritmo 1.2: Cálculo del producto externo entre 2 vectores

Datos de entrada: $\boldsymbol{a} \in \mathbb{R}^I, \boldsymbol{b} \in \mathbb{R}^J$
Inicio
    A partir de $\boldsymbol{a}$ obtener I
    A partir de $\boldsymbol{b}$ obtener J
    Construir la matriz $\boldsymbol{C}$ de tamaño $I \times J$
    Para cada i desde 1 hasta I
        Para cada j desde 1 hasta J
            $\boldsymbol{C}[i,j] \leftarrow \boldsymbol{a}[i] * \boldsymbol{b}[j]$
        Fin del "Para cada j"
    Fin del "Para cada i"
Fin
Datos de salida: $\boldsymbol{C}$

1.3 OPERACIÓN VEC

Sea la matriz $\boldsymbol{A} \in M_{I\times J}$ con columnas $\boldsymbol{a_1}, \boldsymbol{a_2}, \dots, \boldsymbol{a_J} \in \mathbb{R}^I$. La operación Vec sobre la matriz $\boldsymbol{A}$, denotada $Vec(\boldsymbol{A})$, se define como:

$$Vec(\boldsymbol{A}) \stackrel{\text{def}}{=} \begin{pmatrix} \boldsymbol{a_1} \\ \boldsymbol{a_2} \\ \vdots \\ \boldsymbol{a_J} \end{pmatrix}$$

El resultado de la operación $Vec(\boldsymbol{A})$ es un vector del espacio vectorial $\mathbb{R}^{IJ}$ que se construye apilando una columna de la matriz $\boldsymbol{A}$ sobre otra. La operación Vec tiene la siguiente estructura funcional:

$$Vec(\cdot): M_{I\times J} \to \mathbb{R}^{IJ}$$

$$\boldsymbol{A} \longmapsto Vec(\boldsymbol{A})$$

Es de observar entonces que $\forall \boldsymbol{A} \in M_{I\times J}$: $Vec(\boldsymbol{A}) \in \mathbb{R}^{IJ}$. Se procede a hacer una ilustración de la operación Vec. Suponga que:

$$\boldsymbol{A} = \begin{pmatrix} 2 & 1 & 7 \\ -5 & 3 & -1 \end{pmatrix} \in M_{2\times 3}$$

Luego: $Vec(\boldsymbol{A}) = \begin{matrix} 2 \\ -5 \\ 1 \\ 3 \\ 7 \\ (-1) \end{matrix} \in \mathbb{R}^6$

Algunas propiedades importantes de la operación Vec son las siguientes:

1.- $\forall \boldsymbol{a} \in \mathbb{R}^I$: $Vec(\boldsymbol{a}) = Vec(\boldsymbol{a}^T) = \boldsymbol{a}$

2.- $\forall \boldsymbol{A}, \boldsymbol{B} \in M_{I\times J}$: $Vec(\boldsymbol{A})^T Vec(\boldsymbol{B}) = tr(\boldsymbol{A}^T\boldsymbol{B})$

3.- $\forall \boldsymbol{A}, \boldsymbol{B} \in M_{I\times J}$: $Vec(\boldsymbol{A} + \boldsymbol{B}) = Vec(\boldsymbol{A}) + Vec(\boldsymbol{B})$

4.- $\forall \alpha \in \mathbb{R}\ \forall \boldsymbol{A} \in M_{I\times J}$: $Vec(\alpha\ \boldsymbol{A}) = \alpha\ Vec(\boldsymbol{A})$

La función $tr(\cdot)$ representa la traza de una matriz cuadrada, que está definida como la suma de los elementos ubicados en la diagonal principal de la matriz. Las propiedades 3 y 4 nos dicen que el operador Vec es lineal, lo que es muy importante en el estudio de los modelos de tablas de 3 vías.

A continuación se propone el **Algoritmo 1.3** para calcular la operación Vec de una matriz:

Algoritmo 1.3: Cálculo de la operación Vec de una matriz

Datos de entrada: $\boldsymbol{A} \in M_{I\times J}$
Inicio
    A partir de $\boldsymbol{A}$ obtener la dupla: $(I \quad J)$
    Construir el vector $\boldsymbol{b}$ de tamaño IJ
    $Pos \leftarrow 0$
    Para cada j desde 1 hasta J
        Para cada i desde 1 hasta I
            $Pos \leftarrow Pos + 1$
            $\boldsymbol{b}[Pos] \leftarrow \boldsymbol{A}[i, j]$
        Fin del "Para cada i"
    Fin del "Para cada j"
Fin
Datos de salida: $\boldsymbol{b}$

1.4 PRODUCTO DE KRONECKER

Sean las matrices $\boldsymbol{A} = (a_{ij}) \in M_{I\times J}$ y $\boldsymbol{B} \in M_{K\times L}$. El producto de Kronecker entre $\boldsymbol{A}$ y $\boldsymbol{B}$, denotado $\boldsymbol{A} \otimes \boldsymbol{B}$, se define como:

$$\boldsymbol{A} \otimes \boldsymbol{B} \stackrel{\text{def}}{=} \begin{pmatrix} a_{11}\boldsymbol{B} & a_{12}\boldsymbol{B} & \cdots & a_{1J}\boldsymbol{B} \\ a_{21}\boldsymbol{B} & a_{22}\boldsymbol{B} & \cdots & a_{2J}\boldsymbol{B} \\ \vdots & \vdots & \vdots & \vdots \\ a_{I1}\boldsymbol{B} & a_{I2}\boldsymbol{B} & \cdots & a_{IJ}\boldsymbol{B} \end{pmatrix}$$

El resultado del producto de Kronecker $\boldsymbol{A} \otimes \boldsymbol{B}$ es una matriz de tamaño $IK \times JL$. No necesariamente las matrices deben tener el mismo tamaño. El producto de Kronecker tiene la siguiente estructura funcional:

$$\otimes : M_{I\times J} \times M_{K\times L} \to M_{IK\times JL}$$
$$(\boldsymbol{A}, \boldsymbol{B}) \mapsto \boldsymbol{A} \otimes \boldsymbol{B}$$

Es de observar entonces que $\forall \boldsymbol{A} \in M_{I\times J} \forall \boldsymbol{B} \in M_{K\times L}$: $\boldsymbol{A} \otimes \boldsymbol{B} \in M_{IK\times JL}$. Se procede a hacer una ilustración del producto de Kronecker.

Suponga que $\boldsymbol{A} = \begin{pmatrix} 2 & -1 \\ 5 & 3 \end{pmatrix} \in M_{2\times 2}$ y que $\boldsymbol{B} = \begin{pmatrix} 3 & 1 \\ 1 & 4 \\ -1 & 6 \end{pmatrix} \in M_{3\times 2}$. Luego:

$$\boldsymbol{A} \otimes \boldsymbol{B} = \begin{pmatrix} 2\boldsymbol{B} & -\boldsymbol{B} \\ 5\boldsymbol{B} & 3\boldsymbol{B} \end{pmatrix}$$

$$\Rightarrow \boldsymbol{A} \otimes \boldsymbol{B} = \begin{pmatrix} 6 & 2 & -3 & -1 \\ 2 & 8 & -1 & -4 \\ -2 & 12 & 1 & -6 \\ 15 & 5 & 9 & 3 \\ 5 & 20 & 3 & 12 \\ -5 & 30 & -3 & 18 \end{pmatrix} \in M_{6\times 4}$$

Se tienen 2 propiedades importantes que relacionan la operación Vec con el producto de Kronecker:

1.- $\forall \boldsymbol{a} \in \mathbb{R}^I \; \forall \boldsymbol{b} \in \mathbb{R}^J: \boldsymbol{a} \otimes \boldsymbol{b} = Vec(\boldsymbol{b} \cdot_{ex} \boldsymbol{a}) = Vec(\boldsymbol{b}\boldsymbol{a}^T)$

2.- $\forall \boldsymbol{A} \in M_{I\times J} \forall \boldsymbol{B} \in M_{J\times K} \forall \boldsymbol{C} \in M_{K\times L}: Vec(\boldsymbol{ABC}) = (\boldsymbol{C}^T \otimes \boldsymbol{A}) Vec(\boldsymbol{B})$

Otras propiedades del producto de Kronecker son las siguientes:

1.- $\forall \alpha \in \mathbb{R} \; \forall \boldsymbol{A} \in M_{I\times J}: \alpha \otimes \boldsymbol{A} = \alpha \boldsymbol{A}$

2.- $\forall \boldsymbol{A} \in M_{I\times J} \forall \boldsymbol{B} \in M_{K\times L} \forall \boldsymbol{C} \in M_{P\times Q}: \boldsymbol{A} \otimes (\boldsymbol{B} \otimes \boldsymbol{C}) = (\boldsymbol{A} \otimes \boldsymbol{B}) \otimes \boldsymbol{C}$

3.- $\forall \boldsymbol{A} \in M_{I\times J} \forall \boldsymbol{C} \in M_{J\times K} \forall \boldsymbol{B} \in M_{L\times P} \forall \boldsymbol{D} \in M_{P\times Q}: (\boldsymbol{A} \otimes \boldsymbol{B})(\boldsymbol{C} \otimes \boldsymbol{D}) = \boldsymbol{AC} \otimes \boldsymbol{BD}$

4.- $\forall \boldsymbol{A} \in M_{I\times J} \forall \boldsymbol{B}, \boldsymbol{C} \in M_{K\times L}: \boldsymbol{A} \otimes (\boldsymbol{B} + \boldsymbol{C}) = (\boldsymbol{A} \otimes \boldsymbol{B}) + (\boldsymbol{A} \otimes \boldsymbol{C})$

5.- $\forall \boldsymbol{A} \in M_{I\times J} \forall \boldsymbol{B} \in M_{K\times L}: (\boldsymbol{A} \otimes \boldsymbol{B})^{\boldsymbol{T}} = \boldsymbol{A}^T \otimes \boldsymbol{B}^T$

6.- $\forall \boldsymbol{A} \in M_{I\times J} \forall \boldsymbol{B} \in M_{K\times L}: (\boldsymbol{A} \otimes \boldsymbol{B})^{+} = \boldsymbol{A}^{+} \otimes \boldsymbol{B}^{+}$

7.- $\forall \boldsymbol{A} \in M_{I\times I} \forall \boldsymbol{B} \in M_{J\times J}: (\boldsymbol{A} \otimes \boldsymbol{B})^{\boldsymbol{-1}} = \boldsymbol{A}^{-1} \otimes \boldsymbol{B}^{-1}$

8.- $\forall \boldsymbol{A} \in M_{I\times I} \forall \boldsymbol{B} \in M_{J\times J}: tr(\boldsymbol{A} \otimes \boldsymbol{B}) = tr(\boldsymbol{A})\, tr(\boldsymbol{B})$

9.- $\forall \boldsymbol{a} \in \mathbb{R}^I \; \forall \boldsymbol{b} \in \mathbb{R}^J: \boldsymbol{a} \cdot_{ex} \boldsymbol{b} = \boldsymbol{a}\boldsymbol{b}^T = \boldsymbol{a} \otimes \boldsymbol{b}^T = \boldsymbol{b}^T \otimes \boldsymbol{a}$

La función $(\cdot)^T$ representa la transpuesta de una matriz, la función $(\cdot)^+$ representa la inversa generalizada de una matriz y la función $(\cdot)^{-1}$ representa la inversa tradicional de una matriz cuadrada. Se debe señalar que el producto de Kronecker no es conmutativo, pero sí es asociativo como podemos apreciar en la segunda propiedad y también es distributivo respecto de la suma como se observa en la cuarta propiedad.

A continuación se propone el **Algoritmo 1.4** para calcular el producto de Kronecker entre 2 matrices:

Algoritmo 1.4: Cálculo del producto de Kronecker entre 2 matrices

---

Datos de entrada: $\boldsymbol{A} \in M_{I\times J}, \boldsymbol{B} \in M_{K\times L}$
Inicio
    A partir de $\boldsymbol{A}$ obtener la dupla: $(I \quad J)$
    A partir de $\boldsymbol{B}$ obtener la dupla: $(K \quad L)$
    Construir la matriz $\boldsymbol{C}$ de tamaño $IK \times JL$
    $Row \leftarrow 1$
    Para cada i desde 1 hasta I
        Para cada k desde 1 hasta K
            $Col \leftarrow 1$
            Para cada j desde 1 hasta J
                Para cada l desde 1 hasta L
                    $\boldsymbol{C}[Row, Col] \leftarrow \boldsymbol{A}[i, j] * \boldsymbol{B}[k, l]$
                    $Col \leftarrow Col + 1$
                Fin del "Para cada l"
            Fin del "Para cada j"
            $Row \leftarrow Row + 1$
        Fin del "Para cada k"
    Fin del "Para cada i"
Fin
Datos de salida: $\boldsymbol{C}$

1.5 PRODUCTO DE KHATRI-RAO

Sean las matrices $\boldsymbol{A} \in M_{I\times J}$ y $\boldsymbol{B} \in M_{K\times J}$. Sean $\boldsymbol{a_1}, \boldsymbol{a_2}, \cdots, \boldsymbol{a_J} \in \mathbb{R}^I$ las J columnas de $\boldsymbol{A}$ y $\boldsymbol{b_1}, \boldsymbol{b_2}, \cdots, \boldsymbol{b_J} \in \mathbb{R}^k$ las J columnas de $\boldsymbol{B}$. El producto de Khatri-Rao de $\boldsymbol{A}$ y $\boldsymbol{B}$, denotado $\boldsymbol{A} \odot \boldsymbol{B}$, se define como:

$$\boldsymbol{A} \odot \boldsymbol{B} \overset{\text{def}}{=} (\boldsymbol{a_1} \otimes \boldsymbol{b_1} \quad \boldsymbol{a_2} \otimes \boldsymbol{b_2} \quad \cdots \quad \boldsymbol{a_J} \otimes \boldsymbol{b_J})$$

El resultado del producto de Khatri-Rao $\boldsymbol{A} \odot \boldsymbol{B}$ es una matriz de tamaño $IK \times J$. Para realizar el producto de Khatri-Rao ambas matrices deben tener igual número de columnas. El producto de Khatri-Rao tiene la siguiente estructura funcional:

$$\odot : M_{I\times J} \times M_{K\times J} \longrightarrow M_{IK\times J}$$
$$(\boldsymbol{A}, \boldsymbol{B}) \longmapsto \boldsymbol{A} \odot \boldsymbol{B}$$

Es de observar entonces que $\forall \boldsymbol{A} \in M_{I\times J} \forall \boldsymbol{B} \in M_{K\times J}$: $\boldsymbol{A} \odot \boldsymbol{B} \in M_{IK\times J}$. Se procede a hacer una ilustración del producto de Khatri-Rao.

Suponga que $\boldsymbol{A} = \begin{pmatrix} 1 & -1 \\ 2 & 4 \end{pmatrix} \in M_{2\times 2}$ y que $\boldsymbol{B} = \begin{pmatrix} 9 & 1 \\ -2 & 2 \\ -3 & -5 \end{pmatrix} \in M_{3\times 2}$. Luego:

$$\boldsymbol{A} \odot \boldsymbol{B} = \left(\begin{pmatrix} 1 \\ 2 \end{pmatrix} \otimes \begin{pmatrix} 9 \\ -2 \\ -3 \end{pmatrix} \quad \begin{pmatrix} -1 \\ 4 \end{pmatrix} \otimes \begin{pmatrix} 1 \\ 2 \\ -5 \end{pmatrix}\right)$$

$$\Rightarrow \boldsymbol{A} \odot \boldsymbol{B} = \begin{matrix} 9 & -1 \\ -2 & -2 \\ -3 & 5 \\ 18 & 4 \\ -4 & 8 \\ (-6 & -20) \end{matrix} \in M_{6\times 2}$$

Propiedades importantes del producto de Khatri-Rao son las siguientes:

1.- $\forall \boldsymbol{A} \in M_{I\times J} \forall \boldsymbol{B} \in M_{K\times J} \forall \boldsymbol{C} \in M_{L\times J}$: $\boldsymbol{A} \odot (\boldsymbol{B} \odot \boldsymbol{C}) = (\boldsymbol{A} \odot \boldsymbol{B}) \odot \boldsymbol{C}$

2.- $\forall \boldsymbol{A} \in M_{I\times J} \forall \boldsymbol{C} \in M_{J\times K} \forall \boldsymbol{B} \in M_{L\times P} \forall \boldsymbol{D} \in M_{P\times K}$: $(\boldsymbol{A} \otimes \boldsymbol{B})(\boldsymbol{C} \odot \boldsymbol{D}) = \boldsymbol{AC} \odot \boldsymbol{BD}$

Vale la pena señalar que el producto de Khatri-Rao no es conmutativo, pero sí es asociativo como podemos observar en la primera propiedad.

A continuación se propone el **Algoritmo 1.5** para calcular el producto de Khatri-Rao entre 2 matrices:

Algoritmo 1.5: Cálculo del producto de Khatri-Rao entre 2 matrices

---

Datos de entrada: $\boldsymbol{A} \in M_{I\times J}, \boldsymbol{B} \in M_{K\times J}$
Inicio
A partir de $\boldsymbol{A}$ obtener la dupla: $(I \quad J)$
A partir de $\boldsymbol{B}$ obtener K
Construir la matriz $\boldsymbol{C}$ de tamaño $IK \times J$
Para cada j desde 1 hasta J
$Row \leftarrow 1$
Para cada i desde 1 hasta I
Para cada k desde 1 hasta K
$\boldsymbol{C}[Row, j] \leftarrow \boldsymbol{A}[i, j] * \boldsymbol{B}[k, j]$
$Row \leftarrow Row + 1$
Fin del "Para cada k"
Fin del "Para cada i"
Fin del "Para cada j"
Fin
Datos de salida: $\boldsymbol{C}$

1.6 PRODUCTO DE HADAMARD

Sean las matrices $\boldsymbol{A} = (a_{ij}), \boldsymbol{B} = (b_{ij}) \in M_{I\times J}$. Se define el producto de Hadamard de $\boldsymbol{A}$ y $\boldsymbol{B}$, denotado $\boldsymbol{A} \circ \boldsymbol{B}$, como:

$$\boldsymbol{A} \circ \boldsymbol{B} \stackrel{\text{def}}{=} (a_{ij}b_{ij})$$

El resultado del producto de Hadamard $\boldsymbol{A} \circ \boldsymbol{B}$ es una matriz de tamaño $I \times J$. Para realizar el producto de Hadamard ambas matrices deben ser del mismo tamaño. El producto de Hadamard tiene la siguiente estructura funcional:

$$\circ: M_{I\times J} \times M_{I\times J} \longrightarrow M_{I\times J}$$
$$(\boldsymbol{A}, \boldsymbol{B}) \longmapsto \boldsymbol{A} \circ \boldsymbol{B}$$

Es de observar entonces que $\forall \boldsymbol{A}, \boldsymbol{B} \in M_{I\times J}$: $\boldsymbol{A} \circ \boldsymbol{B} \in M_{I\times J}$. Se procede a hacer una ilustración del producto de Hadamard.

Suponga que se tienen $\boldsymbol{A} = \begin{pmatrix} 3 & 5 & 1 \\ 0 & -2 & 6 \\ -1 & -3 & 9 \end{pmatrix} \in M_{3\times 3}$ y $\boldsymbol{B} = \begin{pmatrix} -2 & 2 & 4 \\ 7 & -3 & -1 \\ 1 & 2 & 0 \end{pmatrix} \in M_{3\times 3}$.

Luego:

$$\boldsymbol{A} \circ \boldsymbol{B} = \begin{pmatrix} -6 & 10 & 4 \\ 0 & 6 & -6 \\ -1 & -6 & 0 \end{pmatrix} \in M_{3\times 3}$$

Propiedades importantes del producto de Hadamard son las siguientes:

1.- $\forall \boldsymbol{A}, \boldsymbol{B} \in M_{I\times J}$: $\boldsymbol{A} \circ \boldsymbol{B} = \boldsymbol{B} \circ \boldsymbol{A}$

2.- $\forall \boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C} \in M_{I\times J}$: $\boldsymbol{A} \circ (\boldsymbol{B} \circ \boldsymbol{C}) = (\boldsymbol{A} \circ \boldsymbol{B}) \circ \boldsymbol{C}$

3.- $\forall \boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C} \in M_{I\times J}$: $\boldsymbol{A} \circ (\boldsymbol{B} + \boldsymbol{C}) = (\boldsymbol{A} \circ \boldsymbol{B}) + (\boldsymbol{A} \circ \boldsymbol{C})$

4.- $\forall \boldsymbol{A}, \boldsymbol{B} \in M_{I\times J}$: $(\boldsymbol{A} \circ \boldsymbol{B})^{\boldsymbol{T}} = \boldsymbol{A}^T \circ \boldsymbol{B}^T$

5.- $\forall \boldsymbol{A} \in M_{I\times J} \forall \boldsymbol{B} \in M_{K\times J}$: $(\boldsymbol{A} \odot \boldsymbol{B})^T(\boldsymbol{A} \odot \boldsymbol{B}) = (\boldsymbol{A}^T\boldsymbol{A}) \circ (\boldsymbol{B}^T\boldsymbol{B})$

La primera propiedad nos dice que el producto de Hadamard es conmutativo, la segunda propiedad nos dice que también es asociativo y la tercera que el producto de Hadamard es distributivo respecto de la suma. La última propiedad relaciona el producto de Khatri-Rao con el producto de Hadamard, y es muy importante en el modelo PARAFAC.

A continuación se propone el **Algoritmo 1.6** para calcular el producto de Hadamard entre 2 matrices:

Algoritmo 1.6: Cálculo del producto de Hadamard entre 2 matrices

---

Datos de entrada: $\boldsymbol{A} \in M_{I\times J}, \boldsymbol{B} \in M_{I\times J}$
Inicio
    A partir de $\boldsymbol{A}$ o $\boldsymbol{B}$ obtener la dupla: $(I \quad J)$
    Construir la matriz $\boldsymbol{C}$ de tamaño $I \times J$
    Para cada i desde 1 hasta I
        Para cada j desde 1 hasta J
            $\boldsymbol{C}[i,j] \leftarrow \boldsymbol{A}[i,j] * \boldsymbol{B}[i,j]$
        Fin del "Para cada j"
    Fin del "Para cada i"
Fin
Datos de salida: $\boldsymbol{C}$

1.7 PRODUCTO TENSORIAL

Considere los vectores $\boldsymbol{a} \in \mathbb{R}^I$, $\boldsymbol{b} \in \mathbb{R}^J$ y $\boldsymbol{c} = (c_k) \in \mathbb{R}^K$. El producto tensorial de $\boldsymbol{a}$, $\boldsymbol{b}$ y $\boldsymbol{c}$, denotado $\boldsymbol{a}\Delta\boldsymbol{b}\Delta\boldsymbol{c}$, se define como una tabla de 3 vías $\underline{\mathbf{X}}$ cuyo $k - ésimo$ corte frontal ($k = 1, \dots, K$) está dado por:

$$\boldsymbol{X_k} \stackrel{\text{def}}{=} c_k(\boldsymbol{a} \cdot_{ex} \boldsymbol{b}) = c_k\boldsymbol{a}\boldsymbol{b}^T$$

El producto tensorial tiene la siguiente estructura funcional:

$$\Delta: \mathbb{R}^I \times \mathbb{R}^J \times \mathbb{R}^K \rightarrow$$
$$T_{I\times J\times K}(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}) \longmapsto \boldsymbol{a}\Delta\boldsymbol{b}\Delta\boldsymbol{c}$$

Es de observar que $\forall \boldsymbol{a} \in \mathbb{R}^I \ \forall \boldsymbol{b} \in \mathbb{R}^J \ \forall \boldsymbol{c} \in \mathbb{R}^K$: $\boldsymbol{a}\Delta\boldsymbol{b}\Delta\boldsymbol{c} \in T_{I\times J\times K}$. Se procede a hacer una ilustración del producto tensorial.

Suponga que $\boldsymbol{a} = \begin{pmatrix} 3 \\ -1 \\ 2 \end{pmatrix} \in \mathbb{R}^3$, $\boldsymbol{b} = \begin{pmatrix} -4 \\ 1 \end{pmatrix} \in \mathbb{R}^2$ y $c = \begin{pmatrix} -1 \\ 2 \\ 3 \\ 5 \end{pmatrix} \in \mathbb{R}^4$.

El producto tensorial de $\boldsymbol{a}$, $\boldsymbol{b}$ y $\boldsymbol{c}$ da como resultado una tabla de 3 vías $\underline{\mathbf{X}}$ de tamaño $3 \times 2 \times 4$. Los 4 cortes frontales del producto tensorial $\boldsymbol{a}\Delta\boldsymbol{b}\Delta\boldsymbol{c}$ se presentan a continuación:

$$\boldsymbol{X_1} = (-1)\begin{pmatrix} -12 & 3 \\ 4 & -1 \\ 8 & 2 \end{pmatrix} = \begin{pmatrix} 12 & -3 \\ -4 & 1 \\ -8 & -2 \end{pmatrix}$$

$$\boldsymbol{X_2} = (2)\begin{pmatrix} -12 & 3 \\ 4 & -1 \\ 8 & 2 \end{pmatrix} = \begin{pmatrix} -24 & 6 \\ 8 & -2 \\ 16 & 4 \end{pmatrix}$$

$$\boldsymbol{X_3} = (3)\begin{pmatrix} -12 & 3 \\ 4 & -1 \\ 8 & 2 \end{pmatrix} = \begin{pmatrix} -36 & 9 \\ 12 & -3 \\ 24 & 6 \end{pmatrix}$$

$$\boldsymbol{X_4} = (5)\begin{pmatrix} -12 & 3 \\ 4 & -1 \\ 8 & 2 \end{pmatrix} = \begin{pmatrix} -60 & 15 \\ 20 & -5 \\ 40 & 10 \end{pmatrix}$$

En la **Sección 2.2.4** se presenta la definición de rango para una tabla de 3 vías. Es importante mencionar que el resultado del producto tensorial $\boldsymbol{a}\Delta\boldsymbol{b}\Delta\boldsymbol{c}$ es una tabla de 3 vías de rango uno. En el ejemplo anterior se puede apreciar que todos los cortes frontales del tensor $\underline{\mathbf{X}}$ son múltiplos de la siguiente matriz de tamaño 3×2:

$$\boldsymbol{a}\boldsymbol{b}^T = \begin{pmatrix} -12 & 3 \\ 4 & -1 \\ 8 & 2 \end{pmatrix}$$

A continuación se presenta el **Algoritmo 1.7** que permite calcular el producto tensorial entre 3 vectores:

Algoritmo 1.7: Cálculo del producto tensorial entre 3 vectores

---

Datos de entrada: $\boldsymbol{a} \in \mathbb{R}^I, \boldsymbol{b} \in \mathbb{R}^J, \boldsymbol{c} \in \mathbb{R}^K$
Inicio
A partir de $\boldsymbol{a}$ obtener I
A partir de $\boldsymbol{b}$ obtener J
A partir de $\boldsymbol{c}$ obtener K
Construir el tensor $\underline{\mathbf{X}}$ de tamaño $I \times J \times K$
Para cada k desde 1 hasta K
Para cada i desde 1 hasta I
Para cada j desde 1 hasta J
$\underline{\mathbf{X}}[i, j, k] \leftarrow \boldsymbol{a}[i] * \boldsymbol{b}[j] * \boldsymbol{c}[k]$
Fin del "Para cada j"
Fin del "Para cada i"
Fin del "Para cada k"
Fin
Datos de salida: $\underline{\mathbf{X}}$

CAPÍTULO DOS: "MODELOS MULTIVARIANTES PARA EL ANÁLISIS DE COMPONENTES EN TABLAS DE 3 VÍAS"

En este capítulo se presenta el modelo PARAFAC y se presentan los modelos Tucker. Estos modelos matemáticos se utilizan en la Estadística Multivariante para realizar un análisis de componentes en tablas de 3 vías. Como parte de la presentación de los modelos se incluye el álgebra tensorial correspondiente, las principales ecuaciones, los problemas de optimización relacionados y los algoritmos más populares para el cálculo de la descomposición tensorial. Además, se discuten algunas propiedades de los modelos. Finalmente se hace una comparación entre los modelos. Sin embargo, antes de presentarlos, el capítulo define los distintos tipos de preprocesado para tablas de 3 vías y presenta algunos algoritmos para facilitar la implementación del preprocesado.

Los algoritmos que se proponen en esta tesis toman como punto de partida para el cálculo de componentes ortogonales disjuntos los modelos Tucker y el modelo PARAFAC.

2.1 PREPROCESADO DE TABLAS DE 3 VÍAS

En el análisis de componentes para matrices de datos, al realizar el cálculo de componentes, se pueden tomar los datos en bruto o aplicar un preprocesado de los datos. Este preprocesado típicamente consiste en un centrado, o en un centrado y normalizado, por variable y a lo largo de todos los individuos (ver Smilde et al. 2004). En un análisis de componentes en tablas de 3 vías el tema del preprocesamiento es más complicado ya que se tienen 3 modos diferentes, uno por cada vía. Por tanto se tienen más alternativas al momento de efectuar un preprocesamiento de los datos (ver Kiers and Van Mechelen 2001, Smilde et al. 2004). Entre las opciones de centrado tenemos: centrado de primer modo, centrado de segundo modo y centrado de tercer modo. Entre las opciones de normalizado tenemos: normalizado de primer modo, normalizado de segundo modo y normalizado de tercer modo. El tipo de preprocesado a realizar queda siempre a criterio del analista de los datos. Todas estas alternativas se explican a continuación.

Considere el elemento genérico $x_{ijk} \in \underline{\mathbf{X}}$ de la tabla de 3 vías inicial (antes del preprocesado) y $z_{ijk} \in \underline{\mathbf{Z}}$ el elemento genérico de la tabla resultante después del preprocesado.

Para un centrado de primer modo usamos la ecuación:

$$z_{ijk} = x_{ijk} - \frac{\sum_{i=1}^{I} x_{ijk}}{I}, i = 1, \dots, I; j = 1, \dots, J; k = 1, \dots, K \qquad (2.1.1)$$

Se observa que se fija la variable y se fija también la situación (o tiempo), y luego se realiza un centrado haciendo un barrido a lo largo de todos los individuos.

En el **Algoritmo 2.1.1** se muestran los pasos para realizar un centrado de primer modo:

Algoritmo 2.1.1: Centrado de primer modo para tabla de 3 vías

---

Datos de entrada: $\underline{\mathbf{X}} \in T_{I\times J\times K}$
Inicio
    A partir de $\underline{\mathbf{X}}$ obtener la terna: $(I \quad J \quad K)$
    Para cada k desde 1 hasta K
        Para cada j desde 1 hasta J
            $Media \leftarrow 0$
            Para cada i desde 1 hasta I
                $Media \leftarrow Media + \underline{\mathbf{X}}[i,j,k]$
            Fin del "Para cada i"
            $Media \leftarrow Media/I$
            Para cada i desde 1 hasta I
                $\underline{\mathbf{X}}[i,j,k] \leftarrow \underline{\mathbf{X}}[i,j,k] - Media$
            Fin del "Para cada i"
        Fin del "Para cada j"
    Fin del "Para cada k"
Fin
Datos de salida: $\underline{\mathbf{X}}$

Centrar en el primer modo significa entonces restar a cada dato de la tabla de 3 vías la "media" de los individuos considerados en el barrido, una vez que se han fijado las entidades de los otros modos.

Para realizar un centrado de segundo modo se usa la siguiente ecuación:

$$z_{ijk} = x_{ijk} - \frac{\sum_{j=1}^{J} x_{ijk}}{J}, i = 1, \dots, I; j = 1, \dots, J; k = 1, \dots, K \qquad (2.1.2)$$

En este caso, se fija el individuo y se fija la situación (o tiempo), y luego se realiza un centrado haciendo un barrido a lo largo de todas las variables.

En el **Algoritmo 2.1.2** se muestran los pasos para realizar un centrado de segundo modo:

Algoritmo 2.1.2: Centrado de segundo modo para tabla de 3 vías

---

Datos de entrada: $\underline{\mathbf{X}} \in T_{I\times J\times K}$
Inicio
    A partir de $\underline{\mathbf{X}}$ obtener la terna: $(I \quad J \quad K)$
    Para cada i desde 1 hasta I
        Para cada k desde 1 hasta K
            $Media \leftarrow 0$
            Para cada j desde 1 hasta J
                $Media \leftarrow Media + \underline{\mathbf{X}}[i,j,k]$
            Fin del "Para cada j"
            $Media \leftarrow Media/J$
            Para cada j desde 1 hasta J
                $\underline{\mathbf{X}}[i,j,k] \leftarrow \underline{\mathbf{X}}[i,j,k] - Media$
            Fin del "Para cada j"
        Fin del "Para cada k"
    Fin del "Para cada i"
Fin
Datos de salida: $\underline{\mathbf{X}}$

Centrar en el segundo modo significa entonces restar a cada dato de la tabla de 3 vías la "media" de las variables consideradas en el barrido, una vez que se han fijado las entidades de los otros modos.

Para realizar un centrado de tercer modo se usa la siguiente ecuación:

$$z_{ijk} = x_{ijk} - \frac{\sum_{k=1}^{K} x_{ijk}}{K}, i = 1, \dots, I; j = 1, \dots, J; k = 1, \dots, K \qquad (2.1.3)$$

En este caso, se fija el individuo y se fija la variable, y luego se realiza un centrado haciendo un barrido a lo largo de todas las situaciones (o tiempos).

En el **Algoritmo 2.1.3** se muestran los pasos para realizar un centrado de tercer modo:

Algoritmo 2.1.3: Centrado de tercer modo para tabla de 3 vías

---

Datos de entrada: $\underline{\mathbf{X}} \in T_{I\times J\times K}$
Inicio
    A partir de $\underline{\mathbf{X}}$ obtener la terna: $(I \quad J \quad K)$
    Para cada i desde 1 hasta I
        Para cada j desde 1 hasta J
            $Media \leftarrow 0$
            Para cada k desde 1 hasta K
                $Media \leftarrow Media + \underline{\mathbf{X}}[i,j,k]$
            Fin del "Para cada k"
            $Media \leftarrow Media/K$
            Para cada k desde 1 hasta K
                $\underline{\mathbf{X}}[i,j,k] \leftarrow \underline{\mathbf{X}}[i,j,k] - Media$
            Fin del "Para cada k"
        Fin del "Para cada j"
    Fin del "Para cada i"
Fin
Datos de salida: $\underline{\mathbf{X}}$

Centrar en el tercer modo significa entonces restar a cada dato de la tabla de 3 vías la "media" de las situaciones (o tiempos) consideradas en el barrido, una vez que se han fijado las entidades de los otros modos. Se procede ahora a la explicación del normalizado.

Para un normalizado de primer modo o modo-A se usa la ecuación:

$$z_{ijk} = \frac{x_{ijk}}{\sqrt{\frac{\sum_{j=1}^{J}\sum_{k=1}^{K} x_{ijk}^{2}}{JK}}}, i = 1,\dots,I; j = 1,\dots,J; k = 1,\dots,K \qquad (2.1.4)$$

Se observa que se fija el individuo (es decir, una entidad del primer modo) y se hace un barrido en los otros dos modos (modo-B y modo-C) para calcular una suma de cuadrados de los datos. Esta suma se la divide para el número de datos del barrido, en este caso JK y luego, al aplicarse una raíz cuadrada, se obtiene un factor de desviación.

Cada dato correspondiente al barrido realizado, es dividido finalmente por la desviación antes calculada.

A continuación se propone el **Algoritmo 2.1.4** que permite un normalizado en el primer modo o modo-A:

Algoritmo 2.1.4: Normalizado de primer modo para tabla de 3 vías

---

Datos de entrada: $\underline{\mathbf{X}} \in T_{I\times J\times K}$
Inicio
    A partir de $\underline{\mathbf{X}}$ obtener la terna: $(I \quad J \quad K)$
    Para cada i desde 1 hasta I
        $Suma \leftarrow 0$
        Para cada j desde 1 hasta J
            Para cada k desde 1 hasta K
                $Suma \leftarrow Suma + (\underline{\mathbf{X}}[i,j,k])^{2}$
            Fin del "Para cada k"
        Fin del "Para cada j"
        $Factor \leftarrow \sqrt{\frac{Suma}{JK}}$
        Para cada j desde 1 hasta J
            Para cada k desde 1 hasta K
                $\underline{\mathbf{X}}[i,j,k] \leftarrow \underline{\mathbf{X}}[i,j,k]/Factor$
            Fin del "Para cada k"
        Fin del "Para cada j"
    Fin del "Para cada i"
Fin
Datos de salida: $\underline{\mathbf{X}}$

Para un normalizado de segundo modo o modo-B se usa la ecuación:

$$z_{ijk} = \frac{x_{ijk}}{\sqrt{\frac{\sum_{i=1}^{I}\sum_{k=1}^{K} x^{2}_{ijk}}{IK}}}, i = 1,\dots,I; j = 1,\dots,J; k = 1,\dots,K \qquad (2.1.5)$$

Se observa que se fija la variable (es decir, una entidad del segundo modo) y se hace un barrido en los otros dos modos (modo-A y modo-C) para calcular una suma de cuadrados de los datos. Esta suma se la divide para el número de datos del barrido, en este caso IK y luego, al aplicarse una raíz cuadrada, se obtiene un factor de desviación.

Cada dato correspondiente al barrido realizado, es dividido finalmente por la desviación antes calculada.

A continuación se propone el **Algoritmo 2.1.5** que permite un normalizado en el segundo modo o modo-B:

Algoritmo 2.1.5: Normalizado de segundo modo para tabla de 3 vías

---

Datos de entrada: $\underline{\mathbf{X}} \in T_{I\times J\times K}$
Inicio
    A partir de $\underline{\mathbf{X}}$ obtener la terna: $(I \quad J \quad K)$
    Para cada j desde 1 hasta J
        $Suma \leftarrow 0$
        Para cada i desde 1 hasta I
            Para cada k desde 1 hasta K
                $Suma \leftarrow Suma + (\underline{\mathbf{X}}[i,j,k])^{2}$
            Fin del "Para cada k"
        Fin del "Para cada i"
        $Factor \leftarrow \sqrt{\frac{Suma}{IK}}$
        Para cada i desde 1 hasta I
            Para cada k desde 1 hasta K
                $\underline{\mathbf{X}}[i,j,k] \leftarrow \underline{\mathbf{X}}[i,j,k]/Factor$
            Fin del "Para cada k"
        Fin del "Para cada i"
    Fin del "Para cada j"
Fin
Datos de salida: $\underline{\mathbf{X}}$

Para un normalizado de tercer modo o modo-C se usa la ecuación:

$$z_{ijk} = \frac{x_{ijk}}{\sqrt{\frac{\sum_{i=1}^{I}\sum_{j=1}^{J} x_{ijk}^{2}}{IJ}}}, i = 1,\dots,I; j = 1,\dots,J; k = 1,\dots,K \qquad (2.1.6)$$

Se observa que se fija la situación o tiempo (es decir, una entidad del tercer modo) y se hace un barrido en los otros dos modos (modo-A y modo-B) para calcular una suma de cuadrados de los datos. Esta suma se la divide para el número de datos del barrido, en este caso IJ y luego, al aplicarse una raíz cuadrada, se obtiene un factor de desviación.

Cada dato correspondiente al barrido realizado, es dividido finalmente por la desviación antes calculada.

A continuación se propone el **Algoritmo 2.1.6** que permite un normalizado en el tercer modo o modo-C:

Algoritmo 2.1.6: Normalizado de tercer modo para tabla de 3 vías

---

Datos de entrada: $\underline{\mathbf{X}} \in T_{I\times J\times K}$
Inicio
A partir de $\underline{\mathbf{X}}$ obtener la terna: $(I \quad J \quad K)$
Para cada k desde 1 hasta k
$Suma \leftarrow 0$
Para cada i desde 1 hasta I
Para cada j desde 1 hasta J
$Suma \leftarrow Suma + (\underline{\mathbf{X}}[i,j,k])^{2}$
Fin del "Para cada j"
Fin del "Para cada i"
$Factor \leftarrow \sqrt{\frac{Suma}{IJ}}$
Para cada i desde 1 hasta I
Para cada j desde 1 hasta J
$\underline{\mathbf{X}}[i,j,k] \leftarrow \underline{\mathbf{X}}[i,j,k]/Factor$
Fin del "Para cada j"
Fin del "Para cada i"
Fin del "Para cada k"
Fin
Datos de salida: $\underline{\mathbf{X}}$

Como se afirma en Kiers and Van Mechelen 2001, antes de efectuar cualquier tipo de normalizado se recomienda primero realizar un centrado. En algunos casos se efectúa únicamente un centrado. Se pueden combinar además centrados y normalizados en más de un modo antes de realizar el cálculo de componentes, de acuerdo al modelo PARAFAC o a cualquiera de los modelos Tucker. Como se dijo antes, son alternativas que quedan al criterio del analista de la tabla de 3 vías.

En las siguientes secciones se presentan los modelos Tucker y el modelo PARAFAC, los cuales son métodos de descomposición tensorial (es decir, aplican a tablas de 3 vías). Para ver métodos de descomposición matricial como la descomposición en valores singulares (SVD, por sus siglas en inglés) ver Eckart and Young 1936, Young and Householder 1938, Eckart and Young 1939.

2.2 EL MODELO PARAFAC

Este modelo multilineal propuesto por Carroll and Chang 1970, Harshman 1970 para una tabla de 3 vías $\underline{\mathbf{X}} \in T_{I\times J\times K}$ con elemento genérico x_{ijk} está definido por la ecuación escalar:

$$\forall i = 1, \dots, I \; \forall j = 1, \dots, J \; \forall k = 1, \dots, K: x_{ijk} = \hat{x}_{jk} + e_{ijk} \qquad (2.2.1)$$

Donde $\hat{x}_{jk} = \sum_{r=}^{R} a_{ir} b_{jr} c_{kr}$ y el término e_{ijk} es el residuo que contiene toda la variación no explicada por el modelo. En este modelo se persigue una reducción simultánea en la dimensión de los 3 modos. La constante R representa el número de componentes en la reducción dimensional de cada modo, y debe satisfacer las condiciones $R < I, R < J, R < K$.

Para un tratamiento matricial del modelo tenemos 3 "matrices de cargas" o llamadas también "matrices de componentes": la matriz $\boldsymbol{A} = (a_{ir}) \in M_{I\times R}$ de componentes o cargas del primer modo (individuos), la matriz $\boldsymbol{B} = (b_{jr}) \in M_{J\times R}$ de componentes o cargas del segundo modo (variables) y, finalmente, la matriz $\boldsymbol{C} = (c_{kr}) \in M_{K\times R}$ de componentes o cargas del tercer modo (situaciones o tiempos).

Usando tensores, el modelo PARAFAC se puede escribir mediante la ecuación de descomposición tensorial:

$$\underline{\mathbf{X}} = \hat{\underline{\mathbf{X}}} + \underline{\mathbf{E}} \qquad (2.2.2)$$

Donde $\hat{\mathbf{X}} \in T_{I\times J\times K}$ es el tensor que atrapa toda la variación explicada por el modelo PARAFAC y $\underline{\mathbf{E}} \in T_{I\times J\times K}$ es el tensor de residuos.

El modelo PARAFAC también puede ser definido mediante ecuaciones matriciales, una ecuación por cada modo (ver Smilde et al. 2004). Las 3 ecuaciones son equivalentes, es decir, sirven para representar cada elemento de la tabla de 3 vías, pero desde perspectivas distintas. La ecuación del primer modo es la siguiente:

$$\boldsymbol{X_A} = \boldsymbol{A}(\boldsymbol{C} \odot \boldsymbol{B})^T + \boldsymbol{E_A} \qquad (2.2.3)$$

La ecuación del segundo modo es:

$$\boldsymbol{X_B} = \boldsymbol{B}(\boldsymbol{C} \odot \boldsymbol{A})^T + \boldsymbol{E_B} \qquad (2.2.4)$$

Finalmente, la ecuación del tercer modo es:

$$\boldsymbol{X_C} = \boldsymbol{C}(\boldsymbol{B} \odot \boldsymbol{A})^T + \boldsymbol{E_C} \qquad (2.2.5)$$

Donde $\odot$ representa el producto matricial de Khatri-Rao. Además las matrices $\boldsymbol{E_A} \in M_{I\times JK}$, $\boldsymbol{E_B} \in M_{J\times IK}$ y $\boldsymbol{E_C} \in M_{K\times IJ}$ son las correspondientes matrices de errores.

Con $k = 1, \dots, K$ sea $\boldsymbol{X_k}$ el $k - ésimo$ corte frontal de $\underline{\mathbf{X}}$, el cual es una matriz de tamaño $I \times J$. Se puede probar que:

$$\boldsymbol{X_k} = c_{k1}\boldsymbol{a_1}\boldsymbol{b_1}^T + c_{k2}\boldsymbol{a_2}\boldsymbol{b_2}^T + \cdots + c_{kR}\boldsymbol{a_R}\boldsymbol{b_R}^T + \boldsymbol{E_k} \qquad (2.2.6)$$

Donde los vectores $\boldsymbol{a_1}, \boldsymbol{a_2}, \dots, \boldsymbol{a_R} \in \mathbb{R}^I$ son las R columnas de la matriz de cargas $\boldsymbol{A}$ y los vectores $\boldsymbol{b_1}, \boldsymbol{b_2}, \dots, \boldsymbol{b_R} \in \mathbb{R}^J$ son las R columnas de la matriz de cargas $\boldsymbol{B}$.

Además $c_{k1}, c_{k2}, \dots, c_{kR}$ son las entradas de la matriz de cargas $\boldsymbol{C}$ que corresponden al $k - ésimo$ renglón o fila de $\boldsymbol{C}$. Cada producto externo $\boldsymbol{a_r}\boldsymbol{b_r}^T$ $(r = 1, \dots, R)$ da como resultado una matriz de rango uno de tamaño $I \times J$. La matriz $\boldsymbol{E_k} \in M_{I\times J}$ es la correspondiente matriz de los residuos. Es decir que cada corte frontal k se puede aproximar con una combinación lineal del producto externo de las columnas correspondientes de las matrices de cargas $\boldsymbol{A}$ y $\boldsymbol{B}$, donde los escalares de dicha combinación lineal corresponden al renglón o fila k de la matriz de cargas $\boldsymbol{C}$. La **Ecuación 2.3.6**, en forma matricial, se puede escribir como la **Ecuación 2.3.7**:

$$\boldsymbol{X_k} = \boldsymbol{A}\boldsymbol{D_k}\boldsymbol{B}^T + \boldsymbol{E_k} \quad (2.2.7)$$

Donde la matriz diagonal $\boldsymbol{D_k} \in M_{R\times R}$ contiene en su diagonal principal el $k - ésimo$ renglón o fila de la matriz $\boldsymbol{C}$.

Es de observar que todos los cortes frontales se aproximan usando los mismos componentes (columnas) de $\boldsymbol{A}$ y $\boldsymbol{B}$, pero con diferentes pesos que se encuentran en los renglones de $\boldsymbol{C}$. Existen ecuaciones similares a las **Ecuaciones 2.2.6 y 2.2.7** para los cortes horizontales y verticales (ver Smilde et al. 2004).

Las 3 matrices de cargas pueden ser estimadas con diversos algoritmos, sin embargo, el más popular por su uso, es el algoritmo de mínimos cuadrados alternantes (ALS, por sus siglas en inglés) llamado ParafacALS (ver Smilde et al. 2004, Giordani et al. 2014), que se muestra a continuación en el **Algoritmo 2.2**:

Algoritmo 2.2: Algoritmo ParafacALS

Datos de entrada: $R, \underline{\mathbf{X}} \in T_{I\times J\times K}$
Inicio
A partir de $\underline{\mathbf{X}}$ obtener la terna: $(I \quad J \quad K)$
A partir de $\underline{\mathbf{X}}$ obtener las súper-matrices: $\boldsymbol{X_A}, \boldsymbol{X_B}, \boldsymbol{X_C}$
Construir la matriz $\boldsymbol{A}$ de tamaño $I \times R$
Construir la matriz $\boldsymbol{B}$ de tamaño $J \times R$
Construir la matriz $\boldsymbol{C}$ de tamaño $K \times R$
Inicializar las matrices $\boldsymbol{B}$ y $\boldsymbol{C}$
Repetir
$\boldsymbol{A} \leftarrow \boldsymbol{X_A}[(\boldsymbol{C} \odot \boldsymbol{B})^T]^+$
$\boldsymbol{B} \leftarrow \boldsymbol{X_B}[(\boldsymbol{C} \odot \boldsymbol{A})^T]^+$
$\boldsymbol{C} \leftarrow \boldsymbol{X_C}[(\boldsymbol{B} \odot \boldsymbol{A})^T]^+$
Hasta que se cumpla $CriterioParadaParafacALS$
$\boldsymbol{\Psi} \leftarrow (\boldsymbol{A}^T\boldsymbol{A}) \circ (\boldsymbol{B}^T\boldsymbol{B}) \circ (\boldsymbol{C}^T\boldsymbol{C})$
Fin
Datos de salida: $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{\Psi}$

El algoritmo ParafacALS calcula las 3 matrices de cargas fijando 2 de ellas y encontrando la tercera. Cuando comienza el algoritmo se inicializan 2 matrices, por ejemplo $\boldsymbol{B}$ y $\boldsymbol{C}$. Esta inicialización puede ser aleatoria, y es como se ha implementado en este trabajo de investigación. Luego se calcula $\boldsymbol{A}$. Con $\boldsymbol{A}$ y la $\boldsymbol{C}$ anterior, se calcula una nueva $\boldsymbol{B}$. Con $\boldsymbol{A}$ y la nueva $\boldsymbol{B}$, se calcula una nueva $\boldsymbol{C}$. De esta manera tenemos nuevas matrices $\boldsymbol{B}$ y $\boldsymbol{C}$. Esto es realizado iterativamente hasta que se cumpla un criterio de parada que, en el **Algoritmo 2.2**, lo hemos etiquetado como $CriterioParadaParafacALS$.

Este criterio de parada consiste en alcanzar un máximo número de iteraciones, lo que se puede controlar mediante una variable, por ejemplo $ALSMaxIter$. O que las 3 matrices de cargas no difieran de forma importante entre 2 iteraciones consecutivas, lo que se cumpla primero. Sea Tol una variable de tolerancia que se fija antes de la ejecución del algoritmo y que permite determinar la distancia o diferencia máxima permitida de una misma matriz de cargas en 2 iteraciones consecutivas $iter$ y $iter + 1$. Entonces, se debe verificar que se cumpla la proposición lógica:

$$\|\boldsymbol{A}_{iter} - \boldsymbol{A}_{iter+1}\| \leq Tol \wedge \|\boldsymbol{B}_{iter} - \boldsymbol{B}_{iter+1}\| \leq Tol \wedge \|\boldsymbol{C}_{iter} - \boldsymbol{C}_{iter+1}\| \leq Tol$$

La norma $\|\cdot\|$ utilizada en esta tesis es la popular norma de Frobenius. En otras palabras, el criterio de parada del algoritmo ParafacALS consiste en verificar la condición que primero se cumpla, la de alcanzar un máximo número de iteraciones o la condición de que las 3 matrices no difieran significativamente.

La bondad de ajuste del modelo PARAFAC, o "fit" del modelo PARAFAC, se calcula mediante la expresión:

$$\frac{\sum_{i=1}^{I}\sum_{j=1}^{J}\sum_{k=1}^{K} x_{ijk}^2}{\sum_{i=1}^{I}\sum_{j=1}^{J}\sum_{k=1}^{K} x_{ijk}^2} \times 100\%$$

La salida del algoritmo ParafacALS son las matrices de cargas $\boldsymbol{A}$, $\boldsymbol{B}$ y $\boldsymbol{C}$. Además, una matriz $\boldsymbol{\Psi}$ de tamaño $R \times R$ que se conoce como "matriz Tucker" o también, matriz de correlación de componentes. Esta matriz es de mucha utilidad ya que permite detectar la posible presencia del conocido problema de degeneración (ver Harshman and Lundy 1984) que se discute con más detalle en la **Sección 2.2.2** que se encuentra más adelante.

En general, los componentes en el modelo PARAFAC no son ortogonales, pueden ser oblicuos. Debido a esto el cálculo de los componentes se puede hacer de forma simultánea, ya que no depende el uno del otro. Contrario a lo que ocurre en un análisis de componentes para matrices de datos donde se deben estimar uno a uno los componentes, ya que cada nuevo componente a calcular depende de los componentes que se obtuvieron previamente (por ejemplo, el tercer componente debe ser ortogonal a los 2 primeros). Una discusión sobre este punto lo podemos ver en Smilde et al. 2004. Un análisis del modelo PARAFAC se hace en Sánchez and Kowalski 1990.

2.2.1 LA TÉCNICA DE ESCALADO EN EL MODELO PARAFAC

Bajo condiciones normales, el **Algoritmo 2.2** da unas estimaciones únicas para las matrices de cargas $\boldsymbol{A}$, $\boldsymbol{B}$ y $\boldsymbol{C}$, salvo diferencias por permutación de columnas o escalado.

Esta propiedad de unicidad es muy importante en el modelo PARAFAC, en otras palabras, no existe libertad de rotación como se afirma en Kiers and Van Mechelen 2001. La interpretación geométrica es que en el modelo PARAFAC no sólo los espacios de dimensión reducida son únicos, también lo son sus bases.

La permutación de columnas es simplemente un intercambio arbitrario entre 2 columnas. Las 3 matrices de cargas $\boldsymbol{A}$, $\boldsymbol{B}$ y $\boldsymbol{C}$ tienen R columnas (una por cada componente). Si intercambiamos dos columnas cualesquiera en una matriz de cargas, en esas mismas posiciones se deben intercambiar las columnas en las otras 2 matrices de cargas. Al tratarse únicamente del intercambio de columnas el fit se mantiene.

El escalado consiste en trabajar con columnas de norma uno en 2 matrices de cargas y, compensar dicho cambio o alteración, en la tercera matriz de cargas. En Giordani et al. 2014 podemos ver una implementación de escalado. La técnica de escalado es una de las más simples que se aplica con la intención de mejorar la interpretación del modelo sin que haya pérdida de fit. Para una ilustración, suponga que $\|\boldsymbol{a_r}\|$ es la norma euclídea de la $r-ésima$ columna de $\boldsymbol{A}$ y $\|\boldsymbol{b_r}\|$ es la norma euclídea de la $r-ésima$ columna de $\boldsymbol{B}$. A partir de la **Ecuación 2.2.6** se tiene la **Ecuación 2.2.1.1**:

$$\boldsymbol{X_k} = \sum_{r=1}^{R} [(\|\boldsymbol{a_r}\|\|\boldsymbol{b_r}\| c_{kr}) \frac{\boldsymbol{a_r}}{\|\boldsymbol{a_r}\|} (\frac{\boldsymbol{b_r}}{\|\boldsymbol{b_r}\|})^T] + \boldsymbol{E_k} = \sum_{r=1}^{R} [c'_{kr}\ \boldsymbol{a'_r}\ \boldsymbol{b'^T_r}] + \boldsymbol{E_k}$$

(2.2.1.1)

Donde:

$$c'_k = \|\boldsymbol{a_r}\|\|\boldsymbol{b_r}\| c_{kr}$$

$$\boldsymbol{a'_r} = \frac{\boldsymbol{a_r}}{\|\boldsymbol{a_r}\|}$$

$$\boldsymbol{b'_r} = \frac{\boldsymbol{b_r}}{\|\boldsymbol{b_r}\|}$$

De esta manera, se dice que un escalado a columnas unitarias en las matrices de cargas $\boldsymbol{A}$ y $\boldsymbol{B}$ se compensa en la matriz $\boldsymbol{C}$. Se puede también escalar a columnas unitarias en una única matriz de cargas y compensar en cualquiera de las otras 2 matrices de cargas. Sin embargo, es más común el primer caso, es decir, escalar a columnas unitarias en 2 matrices de cargas para compensar esos cambios en la tercera matriz.

A continuación se proponen 3 algoritmos de escalado. El **Algoritmo 2.2.1.1** para aplicar escalado a matrices $\boldsymbol{A}$ y $\boldsymbol{B}$ con compensación en la matriz $\boldsymbol{C}$. Luego el **Algoritmo 2.2.1.2** para aplicar escalado a matrices $\boldsymbol{A}$ y $\boldsymbol{C}$ con compensación en $\boldsymbol{B}$. Finalmente el **Algoritmo 2.2.1.3** para aplicar escalado a matrices $\boldsymbol{B}$ y $\boldsymbol{C}$ con compensación en la matriz $\boldsymbol{A}$.

Algoritmo 2.2.1.1: Escalamiento de matrices de cargas $\boldsymbol{A}$ y $\boldsymbol{B}$ en el modelo PARAFAC

Datos de entrada: $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$
Inicio
    A partir de $\boldsymbol{A}$ obtener la duada: $(I \quad R)$
    A partir de $\boldsymbol{B}$ obtener J
    A partir de $\boldsymbol{C}$ obtener K
    Para cada r desde 1 hasta R
        $\alpha \leftarrow \|\boldsymbol{a}_r\|$
        $\beta \leftarrow \|\boldsymbol{b}_r\|$
        Para cada i desde 1 hasta I
            $\boldsymbol{A}[i,r] \leftarrow \boldsymbol{A}[i,r]/\alpha$
        Fin del "Para cada i"
        Para cada j desde 1 hasta J
            $\boldsymbol{B}[j,r] \leftarrow \boldsymbol{B}[j,r]/\beta$
        Fin del "Para cada j"
        Para cada k desde 1 hasta K
            $\boldsymbol{C}[k,r] \leftarrow \alpha * \beta * \boldsymbol{C}[k,r]$
        Fin del "Para cada k"
    Fin del "Para cada r"
Fin
Datos de salida: $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$

Algoritmo 2.2.1.2: Escalamiento de matrices de cargas $\boldsymbol{A}$ y $\boldsymbol{C}$ en el modelo PARAFAC

Datos de entrada: $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$
Inicio
A partir de $\boldsymbol{A}$ obtener la duada: $(I \quad R)$
A partir de $\boldsymbol{B}$ obtener J
A partir de $\boldsymbol{C}$ obtener K
Para cada r desde 1 hasta R
$\alpha \leftarrow \|\boldsymbol{a_r}\|$
$\gamma \leftarrow \|\boldsymbol{c_r}\|$
Para cada i desde 1 hasta I
$\boldsymbol{A}[i,r] \leftarrow \boldsymbol{A}[i,r]/\alpha$
Fin del "Para cada i"
Para cada k desde 1 hasta K
$\boldsymbol{C}[k,r] \leftarrow \boldsymbol{C}[k,r]/\gamma$
Fin del "Para cada k"
Para cada j desde 1 hasta J
$\boldsymbol{B}[j,r] \leftarrow \alpha * \gamma * \boldsymbol{B}[j,r]$
Fin del "Para cada j"
Fin del "Para cada r"
Fin
Datos de salida: $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$

Algoritmo 2.2.1.3: Escalamiento de matrices de cargas $\boldsymbol{B}$ y $\boldsymbol{C}$ en el modelo PARAFAC

Datos de entrada: $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$
Inicio
    A partir de $\boldsymbol{A}$ obtener la duada: $(I \quad R)$
    A partir de $\boldsymbol{B}$ obtener J
    A partir de $\boldsymbol{C}$ obtener K
    Para cada r desde 1 hasta R
        $\beta \leftarrow \|\boldsymbol{b_r}\|$
        $\gamma \leftarrow \|\boldsymbol{c_r}\|$
        Para cada j desde 1 hasta J
            $\boldsymbol{B}[j,r] \leftarrow \boldsymbol{B}[j,r]/\beta$
        Fin del "Para cada j"
        Para cada k desde 1 hasta K
            $\boldsymbol{C}[k,r] \leftarrow \boldsymbol{C}[k,r]/\gamma$
        Fin del "Para cada k"
        Para cada i desde 1 hasta I
            $\boldsymbol{A}[i,r] \leftarrow \beta * \gamma * \boldsymbol{A}[i,r]$
        Fin del "Para cada i"
    Fin del "Para cada r"
Fin
Datos de salida: $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$

El tipo de escalado a escoger dependerá de las necesidades del analista, en términos de prioridades respecto de los modos. Por ejemplo, el analista o investigador puede estar más interesado en la interpretación de los componentes en las matrices de cargas $\boldsymbol{B}$ y $\boldsymbol{C}$. Si tal es el caso, se puede hacer uso del **Algoritmo 2.2.1.3** para aplicar un "Escalamiento de $\boldsymbol{B}$ y $\boldsymbol{C}$" y compensar ese escalado en la matriz de cargas $\boldsymbol{A}$.

2.2.2 EL PROBLEMA DE DEGENERACIÓN EN EL MODELO PARAFAC

En el modelo PARAFAC existe un problema conocido como "problema de degeneración" que consiste en una aparente convergencia en el fit del modelo (a medida que se itera el fit tiende a estabilizarse en un valor) pero en una divergencia en las entradas de las matrices de cargas (los valores de las matrices se comportan de manera inestable). Lógicamente, si este problema se presenta al analizar una tabla de 3 vías con el modelo PARAFAC, no podemos hacer afirmaciones o llegar a conclusiones de ningún tipo ya que los valores de las matrices de cargas no son confiables (ver Harshman and Lundy 1984, Stegeman 2006, Stegeman 2007, Silva and Lim 2008).

Para detectar la presencia de la degeneración se usa la matriz $\boldsymbol{\Psi}$ del **Algoritmo 2.2** la cual es una matriz de correlaciones entre los componentes. Si el número de iteraciones es extremadamente alto (superior a las 1000 iteraciones) y una de las entradas de la matriz $\boldsymbol{\Psi}$ es un valor cercano a -1, se dice que estamos ante el problema de degeneración del modelo PARAFAC, que ocurre debido a que existe un par de componentes que son casi idénticos y de signo opuesto, en otras palabras, dichos componentes tienen una alta correlación negativa, por lo que sus contribuciones aproximadamente se ven canceladas. Si está presente el problema de degeneración, a medida que aumenta el número de iteraciones del **Algoritmo 2.2**, más tiende a estabilizarse el fit del modelo y más se acerca a -1 una de las entradas de la matriz $\boldsymbol{\Psi}$, pero las entradas de las matrices de cargas son inestables y para nada confiables (ver Smilde et al. 2004).

Por tal razón el **Algoritmo 2.2** presenta el fenómeno de "quemado" por el altísimo número de iteraciones que realiza y porque además consume los recursos del computador de forma importante (ver Rocci and Giordani 2010, Giordani and Rocci 2013).

Por otro lado, si la matriz $\boldsymbol{\Psi}$ satisface la condición $\boldsymbol{\Psi} \cong \boldsymbol{I_R}$ donde $\boldsymbol{I_R}$ es la matriz identidad de tamaño $R \times R$ en un modelo PARAFAC con R componentes, y además, el fit presenta convergencia en unas centenas de iteraciones o menos, se considera que el problema de degeneración no está presente y se puede continuar con el análisis de los componentes.

Para solucionar el problema de degeneración Harshman and Lundy 1984 proponen incorporar restricciones de ortogonalidad al menos en unas de las matrices de cargas. Para ver una ilustración de este tipo de solución se puede observar un experimento computacional en Giordani et al. 2014. Sin embargo, usar restricciones de ortogonalidad puede hacer que el modelo PARAFAC pierda la propiedad de unicidad en la solución. Otra forma de solucionar el problema de degeneración es cambiar a un modelo Tucker. En esta tesis proponemos utilizar el algoritmo heurístico CBPSO-ParafacALS también como una alternativa para solucionar el problema de degeneración del modelo PARAFAC. En el algoritmo propuesto usamos restricciones de ortogonalidad y restricciones "disjoint" al menos en una de las matrices de cargas.

En el **Capítulo 5** se muestran experimentos computacionales donde se discute el problema de degeneración del modelo PARAFAC.

2.2.3 CONSIDERACIONES ADICIONALES EN EL MODELO PARAFAC

El **Algoritmo 2.2** se ha hecho muy popular para el cálculo de componentes en el modelo PARAFAC, pero presenta algunas desventajas. El algoritmo ParafacALS no garantiza convergencia, es decir, no existe garantía teórica de que se alcanzará la solución óptima, sin embargo, en la práctica presenta buenos resultados. Dicho con otras palabras, no existe un tratamiento matemático riguroso de la convergencia y es por eso que se afirma que la convergencia al mínimo global no tiene garantía, pero en la práctica usualmente se alcanza. Si en una ejecución no es alcanzada la convergencia, se pueden cambiar las entradas iniciales de las matrices $\boldsymbol{B}$ y $\boldsymbol{C}$ para nuevamente ejecutar el algoritmo. El valor inicial de estas entradas es muy importante para alcanzar la convergencia con pocas iteraciones. Incluso, paquetes del software estadístico R, como el paquete "ThreeWay" utilizado en Giordani et al. 2014, ejecuta 3 veces (si el usuario no especifica otro valor) el **Algoritmo 2.2** con 3 inicializaciones aleatorias distintas, ante una única invocación del correspondiente comando por parte del usuario.

Otra desventaja del algoritmo ParafacALS es el consumo importante del procesador y de la memoria del computador. Es decir, se trata de un algoritmo de mínimos cuadrados alternantes con una alta complejidad computacional, principalmente por la gran cantidad de operaciones matemáticas elementales involucradas en el producto matricial de Khatri-Rao, en la inversa generalizada de matrices y en el producto de Hadamard de matrices. Además, el algoritmo debe almacenar una enorme cantidad de resultados parciales y, los cortes frontales, horizontales y verticales de la tabla de 3 vías. En Tomasi and Bro 2006 se hace un breve análisis de esto, y de algunas mejoras propuestas al algoritmo ParafacALS que minimizan el número de operaciones básicas utilizadas. En Smilde et al. 2004 se puede también ver una discusión de este tema.

En Tomasi and Bro 2006 se hace un análisis comparativo de algunos de los algoritmos más importantes para el cálculo de componentes en el modelo PARAFAC. Además del algoritmo ParafacALS, existen otros algoritmos que destacan como: el algoritmo de descomposición trilineal directa (DTLD, por sus siglas en inglés), el algoritmo de Gauss-Newton, el algoritmo "line search" y el algoritmo "compression". Los algoritmos mencionados son propuestas que en la práctica no ganaron favoritismo y donde usualmente se impone el algoritmo ParafacALS como el algoritmo más popular.

Si se tiene una tabla de 3 vías, pero sólo con 2 modos (las entidades en los modos $\boldsymbol{A}$ y $\boldsymbol{B}$ son las mismas) el modelo PARAFAC puede ser visto como un modelo INDSCAL. Este último modelo es una extensión a 3 vías, del modelo de escalamiento multidimensional conocido con matrices de datos (tabla de 2 vías). En Smilde et al. 2004 se puede ver una discusión comparativa más profunda de estos 2 modelos. Afirman Ten Berge, Kiers and Krijnen 1993 que cuando el modelo PARAFAC se utiliza como el modelo INDSCAL, en los resultados del cálculo de componentes se espera que las matrices de cargas $\boldsymbol{A}$ y $\boldsymbol{B}$ sean iguales, y además la matriz de cargas $\boldsymbol{C}$ debe contener valores no negativos. Con el algoritmo ParafacALS no existe garantía de alcanzar tales requerimientos, pero vale indicar que en la práctica se satisfacen.

2.2.4 RANGO y k-RANGO DE UNA TABLA DE 3 VÍAS

El modelo PARAFAC es utilizado para definir el rango de una tabla de 3 vías. Por definición (ver Kruskal 1977), el rango de una tabla de 3 vías $\underline{\mathbf{X}}$, denotado $\rho(\underline{\mathbf{X}})$, es el mínimo número R de componentes con fit del 100% en una descomposición tensorial para $\underline{\mathbf{X}}$, de acuerdo con el modelo PARAFAC. Se trata de una generalización de lo que ocurre con matrices o tablas de 2 vías.

En otras palabras, se dice que una tabla de 3 vías $\underline{\mathbf{X}}$ de tamaño $I \times J \times K$ tiene rango R ($\rho(\underline{\mathbf{X}}) = R$) si y sólo si el elemento genérico x_{ijk} con $i \in \{1, \dots, I\}$, $j \in \{1, \dots, J\}$ y $k \in \{1, \dots, K\}$ se puede escribir, con el mínimo número R de términos, de la forma:

$$x_{ijk} = \sum_{r=1}^{R} a_{ir} b_{jr} c_{kr}$$

El k-rango de una matriz o tabla de 2 vías $\boldsymbol{X}$ (ver Kruskal 1976, Harshman and Lundy 1984) es muy importante para determinar si una descomposición PARAFAC es única. Si $\boldsymbol{X}$ es de tamaño $I \times J$, su rango, denotado $\rho(\boldsymbol{X})$, está definido como el máximo número de columnas linealmente independientes. Se conoce que en general $\rho(\boldsymbol{X}) \leq J$. En el caso particular que $\rho(\boldsymbol{X}) = J$ se dice que la matriz $\boldsymbol{X}$ es de rango completo.

Suponga que el rango de una matriz $\boldsymbol{X}$ es R. Esto no quiere decir que cualquier conjunto con R columnas de $\boldsymbol{X}$ es linealmente independiente. Lo que significa $\rho(\boldsymbol{X}) = R$ es que la matriz $\boldsymbol{X}$ tiene un conjunto linealmente independiente con R columnas, pero no que todo conjunto con R columnas de $\boldsymbol{X}$ es linealmente independiente. El k-rango de una matriz $\boldsymbol{X}$, denotado $k_{\boldsymbol{X}}$, está definido como el entero más grande k para el cual todo conjunto con k columnas de la matriz $\boldsymbol{X}$ es linealmente independiente.

Como se dijo en la **Sección 2.2.1** el modelo PARAFAC tiene la propiedad de unicidad, en otras palabras, no es posible rotar las matrices de cargas sin cambiar el fit del modelo. Como además se indicó en la **Sección 2.2.2** no se tiene garantía de encontrar una solución debido al problema de degeneración. Pero, si los datos de la tabla de 3 vías $\underline{\mathbf{X}}$ tienen una estructura acorde a un modelo PARAFAC con R componentes, entonces se podrá encontrar una única solución que puede ser estimada con cualquiera de los algoritmos conocidos, entre ellos, el popular algoritmo ParafacALS. En Kruskal 1989 se propuso la siguiente condición, en términos de las matrices de cargas, que garantiza unicidad en la solución de un modelo PARAFAC con $R \geq 2$ componentes:

$$k_{\boldsymbol{A}} + k_{\boldsymbol{B}} + k_{\boldsymbol{C}} \geq 2(R + 1)$$

La condición dice que la suma del k-rango de las matrices de cargas $\boldsymbol{A}$, $\boldsymbol{B}$ y $\boldsymbol{C}$ debe ser al menos el duplo del número R de componentes más uno. Se trata de una condición suficiente, no una condición necesaria. Para $R = 2$ o $R = 3$ es además una condición necesaria. Para $R > 3$ no se conoce una condición necesaria para la unicidad del modelo PARAFAC.

Usando el producto tensorial de la **Sección 1.7** y la **Ecuación 2.2.6** podemos expresar el modelo PARAFAC haciendo uso de la ecuación:

$$\underline{\mathbf{X}} = \sum_{r=1}^{R} \boldsymbol{a_r} \Delta \boldsymbol{b_r} \Delta \boldsymbol{c_r} + \underline{\mathbf{E}} \qquad (2.2.4.1)$$

Donde los vectores $\boldsymbol{a_1}, \boldsymbol{a_2}, \ldots, \boldsymbol{a_R} \in \mathbb{R}^I$ son las R columnas de la matriz de cargas $\boldsymbol{A}$, los vectores $\boldsymbol{b_1}, \boldsymbol{b_2}, \ldots, \boldsymbol{b_R} \in \mathbb{R}^J$ son las R columnas de la matriz de cargas $\boldsymbol{B}$ y los vectores $\boldsymbol{c_1}, \boldsymbol{c_2}, \ldots, \boldsymbol{c_R} \in \mathbb{R}^K$ son las R columnas de la matriz de cargas $\boldsymbol{C}$.

Cada término $\boldsymbol{a_r} \Delta \boldsymbol{b_r} \Delta \boldsymbol{c_r}$ para $r = 1, \ldots, R$ genera como resultado una tabla de 3 vías de rango uno. Kruskal 1977 definió el rango de una tabla de 3 vías como el número más pequeño de componentes para el cual se obtiene un fit perfecto del 100% con un modelo PARAFAC. Debido a esto, una propiedad importante del modelo PARAFAC es que su solución con R componentes es la mejor aproximación de rango R para una tabla de datos de 3 vías.

Por tanto, la **Ecuación 2.2.4.1** es una ecuación en modo tensorial para representar al modelo PARAFAC. Se puede usar el producto de Kronecker y la operación Vec para tener otra ecuación que represente al modelo PARAFAC:

$$Vec(\underline{\mathbf{X}}) = \sum_{r=}^{R} \boldsymbol{a_r} \otimes \boldsymbol{b_r} \otimes \boldsymbol{c_r} + Vec(\underline{\mathbf{E}}) \qquad (2.2.4.2)$$

En esta última ecuación $Vec(\underline{\mathbf{X}}) = Vec(\boldsymbol{X})$ y $Vec(\underline{\mathbf{E}}) = Vec(\boldsymbol{E})$, donde $\boldsymbol{X}$ y $\boldsymbol{E}$ son las supermatrices correspondientes relacionadas con la tabla de 3 vías $\underline{\mathbf{X}}$ (ver Smilde et al. 2004).

2.3 EL MODELO TUCKER3

Este modelo multilineal propuesto por Tucker 1966 para una tabla de 3 vías $\underline{\mathbf{X}} \in T_{I\times J\times K}$ con elemento genérico x_{ijk} está definido por la ecuación escalar:

$$\forall i = 1, \dots, I \; \forall j = 1, \dots, J \; \forall k = 1, \dots, K: x_{ijk} = \hat{x}_{jk} + e_{ijk} \qquad (2.3.1)$$

Donde $\hat{x}_{jk} = \sum_{p=1}^{P} \sum_{q=1}^{Q} \sum_{r=1}^{R} g_{pqr} a_{ip} b_{jq} c_{kr}$ y el término e_{ijk} es el residuo que contiene toda la variación no explicada por el modelo. En este modelo se busca una reducción simultánea en la dimensión de los 3 modos, con P componentes para el primer modo, Q componentes para el segundo modo y R componentes para el tercer modo. Se deben satisfacer las condiciones $P < I, Q < J, R < K$.

Para un tratamiento matricial del modelo Tucker3 se tienen 3 "matrices de cargas" o también llamadas "matrices de componentes": la matriz $\boldsymbol{A} = (a_{ip}) \in M_{I\times P}$ del primer modo (individuos), la matriz $\boldsymbol{B} = (b_{jq}) \in M_{J\times Q}$ del segundo modo (variables) y, finalmente, la matriz $\boldsymbol{C} = (c_{kr}) \in M_{K\times R}$ del tercer modo (situaciones o tiempos).

El modelo Tucker3 hace uso de un core $\underline{\mathbf{G}}$, el cual es un tensor de orden 3, es decir, un arreglo de tamaño $P \times Q \times R$ con elemento genérico g_{pqr}. El propósito del núcleo o core $\underline{\mathbf{G}}$ es permitir las interacciones entre los componentes de todos los modos. Por el contrario, el modelo PARAFAC de la **Sección 2.2** no utiliza un core y por ende no tiene soporte para las interacciones entre todos los componentes de todos los modos. Únicamente el $r - ésimo$ componente del primer modo interactúa con el $r - ésimo$ componente del segundo modo y con el $r - ésimo$ componente del tercer modo, para todo $r = 1, \dots, R$ en un modelo PARAFAC con R componentes.

La matriz de cargas $\boldsymbol{A}$ tiene P columnas que representan el sistema referencial en el espacio reducido de los individuos, la matriz de cargas $\boldsymbol{B}$ tiene Q columnas que constituyen el sistema referencial en el espacio reducido de las variables y la matriz de cargas $\boldsymbol{C}$ tiene R columnas que representan el sistema referencial en el espacio reducido de las situaciones o tiempos. El core $\underline{\mathbf{G}}$ proporciona soporte para las PQR interacciones de los componentes en todos los modos.

Usando tensores, el modelo Tucker3 se puede escribir mediante la ecuación de descomposición tensorial:

$$\underline{\mathbf{X}} = \hat{\underline{\mathbf{X}}} + \underline{\mathbf{E}} \qquad (2.3.2)$$

Donde $\hat{\mathbf{X}} \in T_{I\times J\times K}$ es el tensor que atrapa toda la variación explicada por el modelo Tucker3 y $\underline{\mathbf{E}} \in T_{I\times J\times K}$ es el tensor de residuos.

El modelo Tucker3 también puede ser definido mediante ecuaciones matriciales, una ecuación por cada modo. Las 3 ecuaciones son equivalentes, es decir, sirven para representar cada elemento de la tabla de 3 vías, pero desde perspectivas distintas. La ecuación del primer modo es la siguiente:

$$\boldsymbol{X_A} = \boldsymbol{A G_A (C \otimes B)^T + E_A} \quad (2.3.3)$$

La ecuación del segundo modo es:

$$\boldsymbol{X_B} = \boldsymbol{B G_B (C \otimes A)^T + E_B} \quad (2.3.4)$$

Finalmente, la ecuación del tercer modo es:

$$\boldsymbol{X_C} = \boldsymbol{C G_C (B \otimes A)^T + E_C} \quad (2.3.5)$$

Donde $\otimes$ representa el producto matricial de Kronecker. Las matrices $\boldsymbol{G_A} \in M_{P\times QR}$, $\boldsymbol{G_B} \in M_{Q\times PR}$ y $\boldsymbol{G_C} \in M_{R\times PQ}$ son las matrices de cortes frontales, horizontales y verticales del core $\underline{\mathbf{G}}$ que se pueden obtener usando, respectivamente, los algoritmos propuestos: **Algoritmo 1.1.1**, **Algoritmo 1.1.2** y **Algoritmo 1.1.3** de la **Sección 1.1** de este documento de tesis. Además las matrices $\boldsymbol{E_A} \in M_{I\times JK}$, $\boldsymbol{E_B} \in M_{J\times IK}$ y $\boldsymbol{E_C} \in M_{K\times IJ}$ son las correspondientes matrices de errores.

Para calcular componentes ortogonales en el modelo Tucker3 el algoritmo más popular por su uso es el algoritmo de mínimos cuadrados alternantes conocido como TuckALS3, que usa una descomposición en valores singulares. Igual como lo hace el algoritmo ParafacALS, el algoritmo TuckALS3 fija 2 matrices de cargas para poder calcular la tercera. Se puede realizar el cálculo de componentes arbitrarios en el modelo Tucker3, es decir componentes oblicuos, no necesariamente ortogonales, como se afirma en Smilde et al. 2004. Sin embargo, el cálculo de componentes ortogonales presenta beneficios que en la **Sección 2.3.3** se discuten.

A continuación en el **Algoritmo 2.3** se muestra el algoritmo TuckALS3:

Algoritmo 2.3: Algoritmo TuckALS3

Datos de entrada: $P, Q, R, \underline{\mathbf{X}} \in T_{I \times J \times K}$
Inicio
A partir de $\underline{\mathbf{X}}$ obtener la terna: $(I \quad J \quad K)$
A partir de $\underline{\mathbf{X}}$ obtener las supermatrices: $\boldsymbol{X_A}, \boldsymbol{X_B}, \boldsymbol{X_C}$
Construir la matriz $\boldsymbol{A}$ de tamaño $I \times P$
Construir la matriz $\boldsymbol{B}$ de tamaño $J \times Q$
Construir la matriz $\boldsymbol{C}$ de tamaño $K \times R$
$\boldsymbol{B} \leftarrow svd(\boldsymbol{X_B}, Q)$
$\boldsymbol{C} \leftarrow svd(\boldsymbol{X_C}, R)$
Repetir
$\boldsymbol{A} \leftarrow svd(\boldsymbol{X_A}(\boldsymbol{C} \otimes \boldsymbol{B}), P)$
$\boldsymbol{B} \leftarrow svd(\boldsymbol{X_B}(\boldsymbol{C} \otimes \boldsymbol{A}), Q)$
$\boldsymbol{C} \leftarrow svd(\boldsymbol{X_C}(\boldsymbol{B} \otimes \boldsymbol{A}), R)$
Hasta que se cumpla $CriterioParadaTuckALS3$
$\boldsymbol{G_A} \leftarrow \boldsymbol{A}^T \boldsymbol{X_A}(\boldsymbol{C} \otimes \boldsymbol{B})$
Fin
Datos de salida: $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{G_A}$

La notación $\boldsymbol{W} \leftarrow svd(\boldsymbol{V}, S)$ significa que $\boldsymbol{W}$ es una matriz cuyas columnas son los primeros S vectores singulares izquierdos de la matriz $\boldsymbol{V}$. La salida del algoritmo TuckALS3 son las 3 matrices de cargas $\boldsymbol{A}$, $\boldsymbol{B}$ y $\boldsymbol{C}$. Además, el algoritmo TuckALS3 entrega el core del modelo Tucker3 mediante su matriz de cortes frontales $\boldsymbol{G_A}$. A través de esta supermatriz de tamaño $P \times QR$ se tiene acceso a todos los pesos de las interacciones. El criterio de parada del **Algoritmo 2.3** etiquetado como $CriterioParadaTuckALS3$, es el mismo criterio de parada del **Algoritmo 2.2**, es decir, lo que se cumpla primero, o un número máximo de iteraciones es alcanzado, o las matrices de cargas $\boldsymbol{A}$, $\boldsymbol{B}$ y $\boldsymbol{C}$ no difieren significativamente en 2 iteraciones consecutivas. Como se afirma en Smilde et al. 2004, el algoritmo TuckALS3 usualmente converge en unas pocas iteraciones ya que en el modelo Tucker3 no existe el problema de degeneración que encontramos en el modelo PARAFAC. Sin embargo, realiza una cantidad enorme de operaciones matemáticas que consumen muchos recursos computacionales. En ocasiones el algoritmo TuckALS3 puede quedar atrapado en un óptimo local, sin embargo, en Timmerman and Kiers 2000 se afirma que con una elección apropiada de los componentes esto ocurre muy rara vez y TuckALS3 converge al óptimo global.

La bondad de ajuste del modelo Tucker3, o "fit" del modelo Tucker3, se calcula igual que con el modelo PARAFAC, es decir, mediante la expresión:

$$\frac{\sum_{i=1}^{I}\sum_{j=1}^{J}\sum_{k=1}^{K}\hat{x}_{ijk}^{2}}{\sum_{i=1}^{I}\sum_{j=1}^{J}\sum_{k=1}^{K}x_{ijk}^{2}} \times 100\% \quad (2.3.6)$$

Es importante hacer una observación, el modelo PARAFAC de la **Sección 2.2** es un caso particular del modelo Tucker3 cuando $P = Q = R$ y el core $\underline{\mathbf{G}}$ de tamaño $R \times R \times R$ es la tabla identidad de 3 vías, es decir:

$$g_{pqr} = \begin{cases} 1, si\ p = q = r \\ 0, si\ no \end{cases}$$

Donde $p, q, r = 1, \dots, R$ y g_{pqr} es el elemento genérico de $\underline{\mathbf{G}}$. Si tal es el caso, decimos que el core o núcleo $\underline{\mathbf{G}}$ es el arreglo identidad de tamaño $R \times R \times R$, ya que los elementos ubicados en la superdiagonal son iguales a uno y, fuera de ella, los elementos son iguales a cero.

El modelo matemático Tucker3 es conocido como la generalización del análisis de componentes principales para tablas de 3 vías tal como se afirma en Henrion 1994. Para una discusión más detallada acerca de la extensión del modelo Tucker3 al análisis de componentes en tablas de n vías se puede consultar Kroonenberg and de Leeuw 1980, Kroonenberg 1983, Kroonenberg 1984. Es importante indicar que en general no es posible rotar el core $\underline{\mathbf{G}}$ del modelo Tucker3 para obtener el modelo PARAFAC, tal como se afirma en Smilde et al. 2004. También ver Tucker 1963.

Para una descomposición matricial con "core" aplicada a una matriz de datos ver Levin 1965, Magnus and Neudecker 1988. Para una descomposición sin "core" ver Hotelling 1933. Para una discusión de ambos esquemas de descomposición ver Ten Berge and Kiers 1996, Ten Berge and Kiers 1997.

2.3.1 LIBERTAD ROTACIONAL EN EL MODELO TUCKER3

En el modelo PARAFAC de la **Sección 2.2** se afirmó que existe unicidad en la solución. Es importante tener claro lo que significa la unicidad de la solución en un modelo. Se dice que un modelo presenta unicidad en su solución si todos los parámetros (en el análisis de componentes, por ejemplo, las entradas de las matrices de cargas) del modelo se pueden estimar de manera única, independientemente del algoritmo utilizado para la estimación de dichos parámetros. Como se mencionó en la **Sección 2.2.1** el intercambio de columnas y el escalado son formas triviales de no unicidad que no se toman en cuenta para determinar la unicidad de un modelo, ya que se consideran irrelevantes. El modelo Tucker3 no presenta unicidad en su solución tal como se afirma en Bro 1998. Esto es debido a que las matrices de cargas y el core pueden ser rotados de forma tal que se obtenga como resultado otra solución del modelo (por lo tanto se mantiene el fit del modelo). El objetivo de efectuar rotaciones, y por ende buscar otras soluciones, es intentar obtener matrices de cargas de estructura simple, es decir, matrices de cargas interpretables.

A continuación un análisis de la rotación en las matrices de cargas y en el core del modelo Tucker3. Sean $\boldsymbol{D}$, $\boldsymbol{E}$ y $\boldsymbol{F}$ matrices invertibles de tamaño $P \times P$, $Q \times Q$ y $R \times R$, respectivamente. Es decir, existen las matrices $\boldsymbol{D}^{-1}$, $\boldsymbol{E}^{-1}$ y $\boldsymbol{F}^{-1}$. Considere la ecuación matricial de cortes frontales para el modelo Tucker3:

$$\boldsymbol{X_A} = \boldsymbol{A}\boldsymbol{G_A}(\boldsymbol{C} \otimes \boldsymbol{B})^T + \boldsymbol{E_A}$$

Mediante la propiedad 5 del producto de Kronecker presentada en el **Capítulo 1**:

$$\boldsymbol{X_A} = \boldsymbol{A}\boldsymbol{G_A}(\boldsymbol{C}^T \otimes \boldsymbol{B}^T) + \boldsymbol{E_A}$$

$$\boldsymbol{X_A} = \boldsymbol{A}(\boldsymbol{D}\boldsymbol{D}^{-1})\boldsymbol{G_A}((\boldsymbol{F}\boldsymbol{F}^{-1})\boldsymbol{C}^T \otimes (\boldsymbol{E}\boldsymbol{E}^{-1})\boldsymbol{B}^T) + \boldsymbol{E_A}$$

Agrupando al interior del producto de Kronecker:

$$\boldsymbol{X_A} = \boldsymbol{A}(\boldsymbol{D}\boldsymbol{D}^{-1})\boldsymbol{G_A}(\boldsymbol{F}(\boldsymbol{F}^{-1}\boldsymbol{C}^T) \otimes \boldsymbol{E}(\boldsymbol{E}^{-1}\boldsymbol{B}^T)) + \boldsymbol{E_A}$$

Usando la propiedad 3 del producto de Kronecker:

$$\boldsymbol{X_A} = \boldsymbol{A}(\boldsymbol{D}\boldsymbol{D}^{-1})\boldsymbol{G_A}(\boldsymbol{F} \otimes \boldsymbol{E})(\boldsymbol{F}^{-1}\boldsymbol{C}^T \otimes \boldsymbol{E}^{-1}\boldsymbol{B}^T) + \boldsymbol{E_A}$$

Por propiedad de la transpuesta de una matriz:

$$\boldsymbol{X_A = A(DD^{-1})G_A(F \otimes E)((C(F^{-1})^T)^T \otimes (B(E^{-1})^T)^T) + E_A}$$

Nuevamente utilizando la propiedad 5:

$$\boldsymbol{X_A = A(DD^{-1})G_A(F \otimes E)(C(F^{-1})^T \otimes B(E^{-1})^T)^T + E_A}$$

Finalmente, agrupando la matriz de cargas del primer modo y la supermatriz de cortes frontales del core:

$$\boldsymbol{X_A = (AD)(D^{-1}G_A(F \otimes E))(C(F^{-1})^T \otimes B(E^{-1})^T)^T + E_A} \qquad (2.3.1.1)$$

En la **Ecuación 2.3.1.1** al tomar:

$$\boldsymbol{\bar{A} = AD}$$

$$\boldsymbol{\bar{B} = B(E^{-1})^T}$$

$$\boldsymbol{\bar{C} = C(F^{-1})^T}$$

$$\boldsymbol{\bar{G}_A = D^{-1}G_A(F \otimes E)}$$

Se tiene la **Ecuación 2.3.1.2**:

$$\boldsymbol{X_A = \bar{A}\bar{G}_A(\bar{C} \otimes \bar{B})^T + E_A} \qquad (2.3.1.2)$$

Por tanto, se aprecia que mediante rotaciones de las matrices de cargas y del core del modelo, podemos obtener otras soluciones, ya que el fit se mantiene (los residuos no cambiaron). En el análisis algebraico anterior se efectuaron rotaciones en todas las matrices de cargas simultáneamente y en el core $\underline{\mathbf{G}}$. Podemos también efectuar rotaciones únicamente en una matriz de cargas, cualquiera de ellas, o arbitrariamente en dos de las matrices de cargas.

En general, las rotaciones pueden ser oblicuas y así se realizó el análisis anterior. Si las matrices de rotación $\boldsymbol{D}$, $\boldsymbol{E}$ y $\boldsymbol{F}$ son ortogonales, es decir, si en particular se cumple que $\boldsymbol{D}^T = \boldsymbol{D}^{-1}$, $\boldsymbol{E}^T = \boldsymbol{E}^{-1}$ y $\boldsymbol{F}^T = \boldsymbol{F}^{-1}$ entonces:

$$\bar{\boldsymbol{A}} = \boldsymbol{AD}$$

$$\bar{\boldsymbol{B}} = \boldsymbol{BE}$$

$$\bar{\boldsymbol{C}} = \boldsymbol{CF}$$

$$\bar{\boldsymbol{G}}_{\boldsymbol{A}} = \boldsymbol{D}^T \boldsymbol{G}_{\boldsymbol{A}} (\boldsymbol{F} \otimes \boldsymbol{E})$$

Podemos concluir afirmando que en el modelo Tucker3 existe libertad rotacional ya que las matrices de cargas no son únicas. Esta es la propiedad del modelo Tucker3 conocida como propiedad de "libertad rotacional", ya que las rotaciones no afectan al fit del modelo. No es posible esto en PARAFAC, es suficiente con observar las 3 ecuaciones matriciales equivalentes de su modelo. En el modelo Tucker3 las rotaciones de las matrices de cargas se compensan en el core o núcleo $\underline{\mathbf{G}}$, mientras que el modelo PARAFAC no tiene core. La no unicidad del modelo Tucker3 se puede utilizar en favor de obtener una solución de estructura simple que mejore la interpretación de los resultados. Por ejemplo en Kiers 1998 se utiliza un procedimiento orthomax de rotaciones ortogonales.

Si por ejemplo el objetivo es simplificar la estructura del core, un tipo de rotación es la de máxima varianza. Rotar el core $\underline{\mathbf{G}}$ en el sentido de "máxima varianza" significa que se aplica una rotación tal que muy pocas entradas tendrán un valor absoluto alto (es decir que la variabilidad del core se concentra en unas pocas entradas). Para la implementación se rotan ortogonalmente las matrices de cargas, y se escogen las matrices de rotación $\boldsymbol{D}$, $\boldsymbol{E}$ y $\boldsymbol{F}$ de forma tal que se maximiza la suma de los cuadrados de las entradas del core $\underline{\mathbf{G}}$. En otras palabras, se escogen las matrices de rotación que maximizan la función $\|\boldsymbol{G}_{\boldsymbol{A}}\|^2$ donde $\|\cdot\|$ es la norma de Frobenius.

2.3.2 OBTENCIÓN DEL MODELO TUCKER3 A PARTIR DE UNA ESTRUCTURA LATENTE

Suponga que se tiene una tabla de 3 vías $\underline{\mathbf{X}}$ de tamaño $I \times J \times K$. El elemento genérico x_{ijk} representa la observación (o medida) del objeto i, en la variable j y en la situación k . Suponga además que se tiene una estructura latente con P objetos, Q variables y R situaciones, que puede representar a la estructura original. En la tabla de 3 vías $\underline{\mathbf{G}}$ de tamaño $P \times Q \times R$ se tiene el elemento genérico g_{pqr} que representa la medida del objeto latente p, en la variable latente q y en la situación latente r. El análisis que se presenta a continuación se puede ver en Kroonenberg 1984. La medida del objeto original i en la variable latente q y en la situación latente r, denotada z_{iqr}, está dada por:

$$z_{iqr} = \sum_{p=1}^{P} a_{ip} g_{pqr}$$

Se observa que la medida z_{iqr} del objeto original i se puede escribir como una combinación lineal de las medidas g_{pqr} de todos los P objetos latentes, fijando la variable latente q y fijando la situación r. Los escalares de la combinación lineal son los números a_{ip} para toda $p = 1, \dots, P$. Es decir, los escalares corresponden a la $i - ésima$ fila o renglón de la matriz de cargas $\boldsymbol{A}$.

Ahora, sea y_{ijr} la medida del objeto original i en la variable original j y en la situación latente r, la cual está dada por:

$$y_{ijr} = \sum_{q=1}^{Q} b_{jq} z_{iqr}$$

Se observa que la medida y_{ijr} del objeto original i en la variable original j se puede escribir como combinación lineal de las medidas z_{iqr} del objeto original i en todas las Q variables latentes, fijando la situación latente r. Los escalares de la combinación lineal son los números b_{jq} para toda $q = 1, \dots, Q$. Es decir, los escalares de la combinación lineal corresponden a la $j - ésima$ fila o renglón de la matriz de cargas $\boldsymbol{B}$.

Finalmente, se conoce que x_{ijk} representa la medida del objeto original i, en la variable original j y en la situación original k, la cual está dada por:

$$x_{ijk} = \sum_{r=1}^{R} c_{kr}\, y_{ijr}$$

Se observa que la medida x_{ijk} del objeto original i, en la variable original j y en la situación original k se puede escribir como combinación lineal de las medidas y_{ijr} del objeto original i, en la variable original j, en todas las R situaciones latentes. Los escalares de la combinación lineal son los números c_{kr} para toda $r = 1, \dots, R$. Es decir, los escalares de la combinación lineal corresponden a la $k - ésima$ fila o renglón de la matriz de cargas $\boldsymbol{C}$.

Reemplazando y_{ijr} en la última ecuación:

$$x_{ijk} = \sum_{r=1}^{R} c_{kr}\, y_{ijr}$$

$$x_{ijk} = \sum_{r=1}^{R} c_{kr} \left(\sum_{q=1}^{Q} b_{jq} z_{iqr}\right)$$

Debido a que c_{kr} no depende de la variable q:

$$x_{ijk} = \sum_{r=1}^{R} \sum_{q=1}^{Q} c_{kr} b_{jq} z_{iqr}$$

Reemplazando z_{iqr}:

$$x_{ijk} = \sum_{r=1}^{R} \sum_{q=1}^{Q} c_{kr} b_{jq} \left(\sum_{p=1}^{P} a_{ip} g_{pqr}\right)$$

Debido a que el producto $c_{kr}b_{jq}$ no depende de p:

$$x_{ijk} = \sum_{r=1}^{R} \sum_{q=1}^{Q} \sum_{p=1}^{P} c_{kr} b_{jq} a_{ip} g_{pqr}$$

Cambiando el orden de los factores y el barrido de las variables, finalmente tenemos:

$$x_{ijk} = \sum_{p=1}^{P} \sum_{q=1}^{Q} \sum_{r=1}^{R} g_{pqr} a_{ip} b_{jq} c_{kr}$$

Por tanto se ha probado cómo se puede obtener el modelo Tucker3 a partir de una estructura latente.

2.3.3 CONSIDERACIONES ADICIONALES EN EL MODELO TUCKER3

Usando el producto tensorial de la **Sección 1.7** podemos expresar el modelo Tucker3 haciendo uso de la ecuación:

$$\underline{\mathbf{X}} = \sum_{p=1}^{P} \sum_{q=1}^{Q} \sum_{r=1}^{R} g_{pqr}(\boldsymbol{a_p} \Delta \boldsymbol{b_q} \Delta \boldsymbol{c_r}) + \underline{\mathbf{E}} \qquad (2.3.3.1)$$

Donde los vectores $\boldsymbol{a_1}, \boldsymbol{a_2}, \dots, \boldsymbol{a_P} \in \mathbb{R}^I$ son las P columnas de la matriz de cargas $\boldsymbol{A}$, los vectores $\boldsymbol{b_1}, \boldsymbol{b_2}, \dots, \boldsymbol{b_Q} \in \mathbb{R}^J$ son las Q columnas de la matriz de cargas $\boldsymbol{B}$ y los vectores $\boldsymbol{c_1}, \boldsymbol{c_2}, \dots, \boldsymbol{c_R} \in \mathbb{R}^K$ son las R columnas de la matriz de cargas $\boldsymbol{C}$.

Por tanto, la **Ecuación 2.3.3.1** es una ecuación en modo tensorial para representar al modelo Tucker3. Una ecuación para cada corte frontal $\boldsymbol{X_k}$ del modelo Tucker3 con valores $k = 1, \dots, K$ equivalente a la **Ecuación 2.2.6** del modelo PARAFAC, se muestra:

$$\boldsymbol{X_k} = \boldsymbol{A}\left(\sum_{r=1}^{R} c_{kr}\boldsymbol{G_r}\right)\boldsymbol{B}^T + \boldsymbol{E_k} \qquad (2.3.3.2)$$

Donde $\boldsymbol{A}$ y $\boldsymbol{B}$ son las matrices de cargas, c_{kr} es elemento de la matriz de cargas $\boldsymbol{C}$ de la fila k y columna r, $\boldsymbol{G_r}$ es el $r - \acute{e}simo$ corte frontal de tamaño $P \times Q$ del core $\underline{\mathbf{G}}$ y $\boldsymbol{E_k}$ es la correspondiente matriz de residuos. De la misma manera que con el modelo PARAFAC, existen ecuaciones equivalentes a la **Ecuación 2.3.3.2** que corresponden a los otros modos del modelo Tucker3.

En el modelo Tucker3 se pueden quitar las restricciones de ortogonalidad sobre las matrices de cargas y realizar un cálculo de componentes oblicuos. Sin embargo, se recomiendan tales restricciones porque se facilita la interpretación de las matrices de cargas, sin la necesidad de aplicar algún tipo de escalado. Para un análisis complementario, se pueden calcular ambos tipos de componentes: oblicuos y ortogonales. Si las matrices de cargas son ortogonales se cumple la siguiente propiedad importante:

$$\|\hat{\mathbf{X}}\|^2 = \|\hat{\boldsymbol{X}}_{\boldsymbol{A}}\|^2 = \|\boldsymbol{A}\boldsymbol{G}_{\boldsymbol{A}}(\boldsymbol{C} \otimes \boldsymbol{B})^T\|^2 = \|\boldsymbol{G}\|_{\boldsymbol{A}}^2 = \|\underline{\mathbf{G}}\|^2 \qquad (2.3.3.3)$$

Donde $\hat{\mathbf{X}}$ corresponde al tensor construido con el modelo Tucker3 (Es decir, la aproximación de $\underline{\mathbf{X}}$, ver **Ecuación 2.3.2**) y $\hat{\boldsymbol{X}}_{\boldsymbol{A}}$ es la supermatriz de cortes frontales de $\hat{\mathbf{X}}$ cuyo tamaño es $I \times JK$. Además $\underline{\mathbf{G}}$ es el core de tamaño $P \times Q \times R$ y $\boldsymbol{G_A}$ es la supermatriz de cortes frontales de $\underline{\mathbf{G}}$. Finalmente $\boldsymbol{A}$, $\boldsymbol{B}$ y $\boldsymbol{C}$ son las matrices de cargas.

La **Ecuación 2.3.3.3** nos permite afirmar que:

$$\|\hat{\mathbf{X}}\|^2 = \sum_{p=1}^{P}\sum_{q=1}^{Q}\sum_{r=1}^{R} g_{pqr}^2$$

Por tanto, otra forma de calcular el fit en el modelo Tucker3 y que es equivalente a la **Ecuación 2.3.6** se presenta a continuación:

$$\frac{\sum_{p=1}^{P}\sum_{q=1}^{Q}\sum_{r=1}^{R} g_{pqr}^2}{\sum_{i=1}^{I}\sum_{j=1}^{J}\sum_{k=1}^{K} x_{ijk}^2} \times 100\%$$

El cuadrado del elemento g_{pqr} del core $\underline{\mathbf{G}}$ es la variación que aporta la interacción de los componentes $\boldsymbol{a_p}$, $\boldsymbol{b_q}$ y $\boldsymbol{c_r}$ del modelo, a la variación de $\hat{\mathbf{X}}$. En conclusión, la variación explicada por el modelo Tucker3 es igual a la suma de los cuadrados de las entradas del core $\underline{\mathbf{G}}$. Muy importante, esto nos dice que toda la variación de $\hat{\mathbf{X}}$ está contenida en $\underline{\mathbf{G}}$.

Hay que tener presente que las restricciones de ortogonalidad en el modelo Tucker3 no proporcionan unicidad en la solución. Existe un número infinito de transformaciones ortogonales que producen exactamente el mismo fit. En general, en Tucker3 se tienen subespacios únicos de dimensión reducida, pero no bases únicas.

2.4 LOS MODELOS TUCKER2

Los modelos Tucker2 son casos particulares del modelo Tucker3. En un modelo Tucker2, de los 3 modos, se reducen 2 de ellos. Existen por tanto 3 modelos Tucker2, los cuales podemos observar en la siguiente tabla:

Tabla 2.4.1: Modelos Tucker2

MODELO	MODOS A REDUCIR	MATRIZ DE CARGAS DEL MODO NO REDUCIDO
Tucker2-AB	modo-A, modo-B	$\boldsymbol{C} = \boldsymbol{I}_{K\times K}$
Tucker2-AC	modo-A, modo-C	$\boldsymbol{B} = \boldsymbol{I}_{J\times J}$
Tucker2-BC	modo-B, modo-C	$\boldsymbol{A} = \boldsymbol{I}_{I\times I}$

Sea $\underline{\mathbf{X}}$ una tabla de 3 vías de tamaño $I \times J \times K$. En el modelo Tucker2-AB se reduce el modo-A a P componentes ($P < I$) y se reduce también el modo-B pero a Q componentes ($Q < J$). El modo-C no se reduce, se queda con sus K entidades originales. Para obtener una ecuación matricial de este modelo, tomamos la **Ecuación 2.3.5** del tercer modo del modelo Tucker3:

$$\boldsymbol{X_C} = \boldsymbol{C}\boldsymbol{G_C}(\boldsymbol{B} \otimes \boldsymbol{A})^T + \boldsymbol{E_C}$$

Luego fijamos la matriz de cargas $\boldsymbol{C}$ con la matriz identidad de tamaño $K \times K$, es decir $\boldsymbol{C} = \boldsymbol{I}_{K\times K}$. Finalmente obtenemos:

$$\boldsymbol{X_C} = \boldsymbol{G_C}(\boldsymbol{B} \otimes \boldsymbol{A})^T + \boldsymbol{E_C} \qquad (2.4.1)$$

En este caso la supermatriz de cortes verticales $\boldsymbol{G_C}$ del core $\underline{\mathbf{G}}$ es de tamaño $K \times PQ$, ya que el core es una tabla de 3 vías de tamaño $P \times Q \times K$. La supermatriz $\boldsymbol{E_C}$ de tamaño $K \times IJ$ es la matriz de residuos. En términos escalares, la ecuación para este modelo de Tucker2 es:

$$x_{ijk} = \sum_{p=1}^{P}\sum_{q=1}^{Q} g_{pqk}a_{ip}b_{jq} + e_{ijk} \qquad (2.4.2)$$

Donde x_{ijk} es el elemento genérico de $\underline{\mathbf{X}}$, a_{ip} es el elemento genérico de la matriz de cargas $\boldsymbol{A}$ de tamaño $I \times P$, b_{jq} es el elemento genérico de la matriz de cargas $\boldsymbol{B}$ de tamaño $J \times Q$ y g_{pqk} es el elemento genérico del core $\underline{\mathbf{G}}$. El tensor de errores $\underline{\mathbf{E}}$ tiene elemento genérico e_{ijk}.

Para encontrar las matrices de cargas $\boldsymbol{A}$ y $\boldsymbol{B}$, y el core $\underline{\mathbf{G}}$ en el modelo Tucker2-AB existe un popular algoritmo de mínimos cuadrados alternantes, llamado TuckALS2-AB. Este algoritmo es un caso particular del algoritmo TuckALS3 presentado en el **Algoritmo 2.3** y que realiza el cálculo de componentes ortogonales en el modelo Tucker3.

A continuación se presenta en el **Algoritmo 2.4.1** el algoritmo TuckALS2-AB que realiza el cálculo de componentes ortogonales en el modelo Tucker2-AB:

Algoritmo 2.4.1: Algoritmo TuckALS2-AB

Datos de entrada: $P, Q, \underline{\mathbf{X}} \in T_{I\times J\times K}$
Inicio
A partir de $\underline{\mathbf{X}}$ obtener la terna: $(I \quad J \quad K)$
A partir de $\underline{\mathbf{X}}$ obtener las supermatrices: $\boldsymbol{X_A}, \boldsymbol{X_B}, \boldsymbol{X_C}$
Construir la matriz $\boldsymbol{A}$ de tamaño $I \times P$
Construir la matriz $\boldsymbol{B}$ de tamaño $J \times Q$
$\boldsymbol{B} \leftarrow svd(\boldsymbol{X_B}, Q)$
Repetir
$\boldsymbol{A} \leftarrow svd(\boldsymbol{X_A}(\boldsymbol{I}_{K\times K} \otimes \boldsymbol{B}), P)$
$\boldsymbol{B} \leftarrow svd(\boldsymbol{X_B}(\boldsymbol{I}_{K\times K} \otimes \boldsymbol{A}), Q)$
Hasta que se cumpla $CriterioParadaTuckALS2 - AB$
$\boldsymbol{G_C} \leftarrow \boldsymbol{X_C}(\boldsymbol{B} \otimes \boldsymbol{A})$
Fin
Datos de salida: $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{G_C}$

El criterio de parada de TuckALS2-AB, etiquetado $CriterioParadaTuckALS2 - AB$, chequea dos condiciones. La primera condición es si se ha alcanzado un máximo número de iteraciones, y la segunda es si en dos iteraciones consecutivas $iter$ y $iter + 1$ las matrices de cargas no difieren significativamente para una tolerancia dada Tol que es un dato de entrada del algoritmo, es decir:

$$\|\boldsymbol{A}_{iter} - \boldsymbol{A}_{iter+1}\| \leq Tol \wedge \|\boldsymbol{B}_{iter} - \boldsymbol{B}_{iter+1}\| \leq Tol$$

La condición que se cumpla primero hace que el **Algoritmo 2.4.1** detenga las iteraciones. Finalmente el algoritmo realiza el cálculo del core $\underline{\mathbf{G}}$ y entrega los correspondientes datos de salida.

En el modelo Tucker2-AC se reduce el modo-A a P componentes ($P < I$) y se reduce también el modo-C pero a R componentes ($R < K$). El modo-B no se reduce, se queda con sus J entidades originales. Para obtener una ecuación matricial de este modelo, tomamos la **Ecuación 2.3.4** del segundo modo del modelo Tucker3:

$$\boldsymbol{X_B = BG_B(C \otimes A)^T + E_B}$$

Luego fijamos la matriz de cargas $\boldsymbol{B}$ con la matriz identidad de tamaño $J \times J$, es decir $\boldsymbol{B = I_{J\times J}}$. Finalmente obtenemos:

$$\boldsymbol{X_B = G_B(C \otimes A)^T + E_B} \qquad (2.4.3)$$

En este caso la supermatriz de cortes horizontales $\boldsymbol{G_B}$ del core $\underline{\mathbf{G}}$ es de tamaño $J \times PR$, ya que el core es una tabla de 3 vías de tamaño $P \times J \times R$. La supermatriz $\boldsymbol{E_B}$ de tamaño $J \times IK$ es la matriz de residuos. En términos escalares, la ecuación para este modelo de Tucker2 es:

$$x_{ijk} = \sum_{p=1}^{P} \sum_{r=1}^{R} g_{pjr} a_{ip} c_{kr} + e_{ijk} \qquad (2.4.4)$$

Donde x_{ijk} es el elemento genérico de $\underline{\mathbf{X}}$, a_{ip} es el elemento genérico de la matriz de cargas $\boldsymbol{A}$ de tamaño $I \times P$, c_{kr} es el elemento genérico de la matriz de cargas $\boldsymbol{C}$ de tamaño $K \times R$ y g_{pjr} es el elemento genérico del core $\underline{\mathbf{G}}$. El tensor de errores $\underline{\mathbf{E}}$ tiene elemento genérico e_{ijk}.

Para encontrar las matrices de cargas $\boldsymbol{A}$ y $\boldsymbol{C}$, y el core $\underline{\mathbf{G}}$ en el modelo Tucker2-AC existe un popular algoritmo de mínimos cuadrados alternantes, llamado TuckALS2-AC. Este algoritmo es un caso particular del algoritmo TuckALS3 presentado en el **Algoritmo 2.3** y que realiza el cálculo de componentes ortogonales en el modelo Tucker3.

A continuación se presenta en el **Algoritmo 2.4.2** el algoritmo TuckALS2-AC que realiza el cálculo de componentes ortogonales en el modelo Tucker2-AC:

Algoritmo 2.4.2: Algoritmo TuckALS2-AC

Datos de entrada: $P, R, \underline{\mathbf{X}} \in T_{I\times J\times K}$
Inicio
A partir de $\underline{\mathbf{X}}$ obtener la terna: $(I \quad J \quad K)$
A partir de $\underline{\mathbf{X}}$ obtener las supermatrices: $\boldsymbol{X_A}, \boldsymbol{X_B}, \boldsymbol{X_C}$
Construir la matriz $\boldsymbol{A}$ de tamaño $I \times P$
Construir la matriz $\boldsymbol{C}$ de tamaño $K \times R$
$\boldsymbol{C} \leftarrow svd(\boldsymbol{X_C}, R)$
Repetir
$\boldsymbol{A} \leftarrow svd(\boldsymbol{X_A}(\boldsymbol{C} \otimes \boldsymbol{I}_{J\times J}), P)$
$\boldsymbol{C} \leftarrow svd(\boldsymbol{X_C}(\boldsymbol{I}_{J\times J} \otimes \boldsymbol{A}), R)$
Hasta que se cumpla $CriterioParadaTuckALS2 - AC$
$\boldsymbol{G_B} \leftarrow \boldsymbol{X_B}(\boldsymbol{C} \otimes \boldsymbol{A})$
Fin
Datos de salida: $\boldsymbol{A}, \boldsymbol{C}, \boldsymbol{G_B}$

El criterio de parada de TuckALS2-AC, etiquetado $CriterioParadaTuckALS2 - AC$, verifica dos condiciones. La primera condición es si se ha alcanzado un máximo número de iteraciones, y la segunda es si en dos iteraciones consecutivas $iter$ y $iter + 1$ las matrices de cargas no difieren significativamente para una tolerancia dada Tol que es un dato de entrada del algoritmo, es decir:

$$\|\boldsymbol{A}_{iter} - \boldsymbol{A}_{iter+1}\| \leq Tol \wedge \|\boldsymbol{C}_{iter} - \boldsymbol{C}_{iter+1}\| \leq Tol$$

La condición que se cumpla primero hace que el **Algoritmo 2.4.2** detenga las iteraciones. Finalmente el algoritmo realiza el cálculo del core $\underline{\mathbf{G}}$ y entrega los correspondientes datos de salida.

En el modelo Tucker2-BC se reduce el modo-B a Q componentes ($Q < J$) y se reduce también el modo-C pero a R componentes ($R < K$). El modo-A no se reduce, se queda con sus I entidades originales. Para obtener una ecuación matricial de este modelo, tomamos la **Ecuación 2.3.3** del primer modo del modelo Tucker3:

$$\boldsymbol{X_A} = \boldsymbol{A G_A (C \otimes B)^T + E_A}$$

Luego fijamos la matriz de cargas $\boldsymbol{A}$ con la matriz identidad de tamaño $I \times I$, es decir $\boldsymbol{A} = \boldsymbol{I}_{I \times I}$. Finalmente obtenemos:

$$\boldsymbol{X_A} = \boldsymbol{G_A (C \otimes B)^T + E_A} \qquad (2.4.5)$$

En este caso la supermatriz de cortes frontales $\boldsymbol{G_A}$ del core $\underline{\mathbf{G}}$ es de tamaño $I \times QR$, ya que el core es una tabla de 3 vías de tamaño $I \times Q \times R$. La supermatriz $\boldsymbol{E_A}$ de tamaño $I \times JK$ es la matriz de residuos. En términos escalares, la ecuación para este modelo de Tucker2 es:

$$x_{ijk} = \sum_{q=}^{Q} \sum_{r=1}^{R} g_{iqr} b_{jq} c_{kr} + e_{ijk} \qquad (2.4.6)$$

Donde x_{ijk} es el elemento genérico de $\underline{\mathbf{X}}$, b_{jq} es el elemento genérico de la matriz de cargas $\boldsymbol{B}$ de tamaño $J \times Q$, c_{kr} es el elemento genérico de la matriz de cargas $\boldsymbol{C}$ de tamaño $K \times R$ y g_{iqr} es el elemento genérico del core $\underline{\mathbf{G}}$. El tensor de errores $\underline{\mathbf{E}}$ tiene elemento genérico e_{ijk}.

Para encontrar las matrices de cargas $\boldsymbol{B}$ y $\boldsymbol{C}$, y el core $\underline{\mathbf{G}}$ en el modelo Tucker2-BC existe un popular algoritmo de mínimos cuadrados alternantes, llamado TuckALS2-BC. Este algoritmo es un caso particular del algoritmo TuckALS3 presentado en el **Algoritmo 2.3** y que realiza el cálculo de componentes ortogonales en el modelo Tucker3.

El criterio de parada de TuckALS2-BC, etiquetado $CriterioParadaTuckALS2 - BC$, chequea dos condiciones. La primera condición es si se ha alcanzado un máximo número de iteraciones, y la segunda es si en dos iteraciones consecutivas $iter$ y $iter + 1$ las matrices de cargas no difieren significativamente para una tolerancia dada Tol que es un dato de entrada del algoritmo, es decir:

$$\|\boldsymbol{B}_{iter} - \boldsymbol{B}_{iter+1}\| \leq Tol \wedge \|\boldsymbol{C}_{iter} - \boldsymbol{C}_{iter+1}\| \leq Tol$$

La condición que se cumpla primero hace que el algoritmo TuckALS2-BC detenga las iteraciones. Finalmente el algoritmo realiza el cálculo del core $\underline{\mathbf{G}}$ y entrega los correspondientes datos de salida.

A continuación se presenta en el **Algoritmo 2.4.3** el algoritmo TuckALS2-BC que realiza el cálculo de componentes ortogonales en el modelo Tucker2-BC:

Algoritmo 2.4.3: Algoritmo TuckALS2-BC

Datos de entrada: $Q, R, \underline{\mathbf{X}} \in T_{I\times J\times K}$
Inicio
A partir de $\underline{\mathbf{X}}$ obtener la terna: $(I \quad J \quad K)$
A partir de $\underline{\mathbf{X}}$ obtener las supermatrices: $\boldsymbol{X_A}, \boldsymbol{X_B}, \boldsymbol{X_C}$
Construir la matriz $\boldsymbol{B}$ de tamaño $J \times Q$
Construir la matriz $\boldsymbol{C}$ de tamaño $K \times R$
$\boldsymbol{C} \leftarrow svd(\boldsymbol{X_C}, R)$
Repetir
$\boldsymbol{B} \leftarrow svd(\boldsymbol{X_B}(\boldsymbol{C} \otimes \boldsymbol{I}_{I\times I}), Q)$
$\boldsymbol{C} \leftarrow svd(\boldsymbol{X_C}(\boldsymbol{B} \otimes \boldsymbol{I}_{I\times I}), R)$
Hasta que se cumpla $CriterioParadaTuckALS2-BC$
$\boldsymbol{G_A} \leftarrow \boldsymbol{X_A}(\boldsymbol{C} \otimes \boldsymbol{B})$
Fin
Datos de salida: $\boldsymbol{B}, \boldsymbol{C}, \boldsymbol{G_A}$

El fit, en cualquiera de los 3 modelos Tucker2, se calcula igual que en el modelo PARAFAC y que en el modelo Tucker3. En Smilde et al. 2004 se discuten los 3 modelos Tucker2 y los algoritmos ALS correspondientes para el cálculo de componentes ortogonales. Tal como ocurre con el algoritmo TuckALS3, los algoritmos TuckALS2-AB, TuckALS2-AC y TuckALS2-BC convergen en unas pocas iteraciones. A pesar de realizar una gran cantidad de operaciones básicas elementales, son muy populares en la práctica. El problema de degeneración no está presente en el modelo Tucker3 y, por lo tanto, tampoco en los 3 modelos Tucker2. En todos los algoritmos ALS mostrados, el cálculo del core se realiza una vez que hayan finalizado las iteraciones para no consumir tiempo de procesador innecesariamente, a menos que sea importante el uso del core como criterio de parada de los algoritmos.

En el software estadístico R se tiene el paquete “PTAk” (ver Leibovici 2010) que puede ser utilizado para realizar análisis de componentes en tablas de 3 vías. Se tiene también, más recientemente, el paquete “ThreeWay” (ver Giordani et al. 2014). Este último paquete, no sólo tiene soporte para el modelo PARAFAC y para el modelo Tucker3, además permite el cálculo de componentes en los 3 modelos Tucker2 y en los 3 modelos Tucker1. Se recomienda iniciar cualquier análisis con el correspondiente preprocesado de los datos, el cual queda a criterio del analista o investigador de los datos.

En el paquete “ThreeWay” existen funciones que proporcionan soporte para el preprocesado de los datos. Kiers 2000 afirma que los modelos Tucker y el modelo PARAFAC requieren de datos centrados, o de datos centrados y normalizados. Adicionalmente podemos ver en Harshman and Lundy 1984, Bro and Smilde 2003 un análisis del preprocesado en tablas de 3 vías.

2.5 LOS MODELOS TUCKER1

Los modelos Tucker1, al igual que los modelos Tucker2, son casos particulares del modelo Tucker3. En un modelo Tucker1, de los 3 modos, se reduce únicamente uno de ellos. Existen por tanto 3 modelos Tucker1, los cuales podemos observar en la **Tabla 2.5.1** que se muestra a continuación:

Tabla 2.5.1: Modelos Tucker1

MODELO	MODO A REDUCIR	MATRIZ DE CARGAS DE MODOS NO REDUCIDOS
Tucker1-A	modo-A	$\boldsymbol{B} = \boldsymbol{I}_{J\times J}, \boldsymbol{C} = \boldsymbol{I}_{K\times K}$
Tucker1-B	modo-B	$\boldsymbol{A} = \boldsymbol{I}_{I\times I}, \boldsymbol{C} = \boldsymbol{I}_{K\times K}$
Tucker1-C	modo-C	$\boldsymbol{A} = \boldsymbol{I}_{I\times I}, \boldsymbol{B} = \boldsymbol{I}_{J\times J}$

Sea $\underline{\mathbf{X}}$ una tabla de 3 vías de tamaño $I \times J \times K$. En el modelo Tucker1-A se reduce el modo-A a P componentes ($P < I$). El modo-B no se reduce, se queda con sus J entidades originales. El modo-C tampoco se reduce, se queda con sus K entidades originales. Para obtener una ecuación matricial de este modelo, tomamos la **Ecuación 2.3.3** del primer modo del modelo Tucker3:

$$\boldsymbol{X_A} = \boldsymbol{A G_A}(\boldsymbol{C} \otimes \boldsymbol{B})^T + \boldsymbol{E_A}$$

Luego fijamos la matriz de cargas $\boldsymbol{B}$ con la matriz identidad de tamaño $J \times J$ y fijamos la matriz de cargas $\boldsymbol{C}$ con la matriz identidad de tamaño $K \times K$. Por lo tanto, se tiene que $\boldsymbol{B} = \boldsymbol{I}_{J\times J}$ y $\boldsymbol{C} = \boldsymbol{I}_{K\times K}$. Finalmente obtenemos:

$$\boldsymbol{X_A} = \boldsymbol{A G_A} + \boldsymbol{E_A} \qquad (2.5.1)$$

En este caso la supermatriz de cortes frontales $\boldsymbol{G_A}$ del core $\underline{\mathbf{G}}$ es de tamaño $P \times JK$, ya que el core es una tabla de 3 vías de tamaño $P \times J \times K$. La supermatriz $\boldsymbol{E_A}$ de tamaño $I \times JK$ es la matriz de residuos. En términos escalares, la ecuación para este modelo de Tucker1 es:

$$x_{ijk} = \sum_{p=1}^{P} g_{pjk} a_{ip} + e_{ijk} \qquad (2.5.2)$$

Donde x_{ijk} es el elemento genérico de $\underline{\mathbf{X}}$, a_{ip} es el elemento genérico de la matriz de cargas $\boldsymbol{A}$ de tamaño $I \times P$ y g_{pjk} es el elemento genérico del core $\underline{\mathbf{G}}$. El tensor de errores $\underline{\mathbf{E}}$ tiene elemento genérico e_{ijk}.

La **Ecuación 2.5.1** muestra claramente que el modelo Tucker1-A es equivalente a un análisis de componentes principales para tablas de 2 vías, donde la matriz de cargas $\boldsymbol{A}$ del modelo Tucker1-A corresponde a la matriz de "scores" del modelo de 2 vías, y la supermatriz de cortes frontales $\boldsymbol{G_A}$ del core $\underline{\mathbf{G}}$ corresponde a la transpuesta de la matriz de cargas del modelo de 2 vías. La supermatriz de cortes frontales $\boldsymbol{X_A}$ del tensor $\underline{\mathbf{X}}$ es la correspondiente matriz de datos del modelo de 2 vías que se va a aproximar.

En el modelo Tucker1-B se reduce el modo-B a Q componentes ($Q < J$). El modo-A no se reduce, se queda con sus I entidades originales. El modo-C tampoco se reduce, se queda con sus K entidades originales. Para obtener una ecuación matricial de este modelo, tomamos la **Ecuación 2.3.4** del segundo modo del modelo Tucker3:

$$\boldsymbol{X_B} = \boldsymbol{B G_B} (\boldsymbol{C} \otimes \boldsymbol{A})^T + \boldsymbol{E_B}$$

Luego fijamos la matriz de cargas $\boldsymbol{A}$ con la matriz identidad de tamaño $I \times I$ y fijamos la matriz de cargas $\boldsymbol{C}$ con la matriz identidad de tamaño $K \times K$. Por lo tanto, se tiene que $\boldsymbol{A} = \boldsymbol{I}_{I \times I}$ y $\boldsymbol{C} = \boldsymbol{I}_{K \times K}$. Finalmente obtenemos:

$$\boldsymbol{X_B} = \boldsymbol{B G_B} + \boldsymbol{E_B} \qquad (2.5.3)$$

En este caso la supermatriz de cortes horizontales $\boldsymbol{G_B}$ del core $\underline{\mathbf{G}}$ es de tamaño $Q \times IK$, ya que el core es una tabla de 3 vías de tamaño $I \times Q \times K$. La supermatriz $\boldsymbol{E_B}$ de tamaño $J \times IK$ es la matriz de residuos. En términos escalares, la ecuación para este modelo de Tucker1 es:

$$x_{ijk} = \sum_{q=}^{Q} g_{iqk} b_{jq} + e_{ijk} \qquad (2.5.4)$$

Donde x_{ijk} es el elemento genérico de $\underline{\mathbf{X}}$, b_{jq} es el elemento genérico de la matriz de cargas $\boldsymbol{B}$ de tamaño $J \times Q$ y g_{iqk} es el elemento genérico del core $\underline{\mathbf{G}}$. El tensor de errores $\underline{\mathbf{E}}$ tiene elemento genérico e_{ijk}.

La **Ecuación 2.5.3** muestra claramente que el modelo Tucker1-B es equivalente a un análisis de componentes principales para tablas de 2 vías, donde la matriz de cargas $\boldsymbol{B}$ del modelo Tucker1-B corresponde a la matriz de "scores" del modelo de 2 vías, y la supermatriz de cortes horizontales $\boldsymbol{G_B}$ del core $\underline{\mathbf{G}}$ corresponde a la transpuesta de la matriz de cargas del modelo de 2 vías. La supermatriz de cortes horizontales $\boldsymbol{X_B}$ del tensor $\underline{\mathbf{X}}$ es la correspondiente matriz de datos del modelo de 2 vías que se va a aproximar.

En el modelo Tucker1-C se reduce el modo-C a R componentes ($R < K$). El modo-A no se reduce, se queda con sus I entidades originales. El modo-B tampoco se reduce, se queda con sus J entidades originales. Para obtener una ecuación matricial de este modelo, tomamos la **Ecuación 2.3.5** del tercer modo del modelo Tucker3:

$$\boldsymbol{X_C} = \boldsymbol{C}\boldsymbol{G_C}(\boldsymbol{B} \otimes \boldsymbol{A})^T + \boldsymbol{E_C}$$

Luego fijamos la matriz de cargas $\boldsymbol{A}$ con la matriz identidad de tamaño $I \times I$ y fijamos la matriz de cargas $\boldsymbol{B}$ con la matriz identidad de tamaño $J \times J$. Por lo tanto, se tiene que $\boldsymbol{A} = \boldsymbol{I}_{I\times I}$ y $\boldsymbol{B} = \boldsymbol{I}_{J\times J}$. Finalmente obtenemos:

$$\boldsymbol{X_C} = \boldsymbol{C}\boldsymbol{G_C} + \boldsymbol{E_C} \qquad (2.5.5)$$

En este caso la supermatriz de cortes verticales $\boldsymbol{G_C}$ del core $\underline{\mathbf{G}}$ es de tamaño $R \times IJ$, ya que el core es una tabla de 3 vías de tamaño $I \times J \times R$. La supermatriz $\boldsymbol{E_C}$ de tamaño $K \times IJ$ es la matriz de residuos. En términos escalares, la ecuación para este modelo de Tucker1 es:

$$x_{ijk} = \sum_{r=1}^{R} g_{ijr}c_{kr} + e_{ijk} \qquad (2.5.6)$$

Donde x_{ijk} es el elemento genérico de $\underline{\mathbf{X}}$, c_{kr} es el elemento genérico de la matriz de cargas $\boldsymbol{C}$ de tamaño $K \times R$ y g_{ijr} es el elemento genérico del core $\underline{\mathbf{G}}$. El tensor de errores $\underline{\mathbf{E}}$ tiene elemento genérico e_{ijk}.

La **Ecuación 2.5.5** muestra claramente que el modelo Tucker1-C es equivalente a un análisis de componentes principales para tablas de 2 vías, donde la matriz de cargas $\boldsymbol{C}$ del modelo Tucker1-C corresponde a la matriz de "scores" del modelo de 2 vías, y la supermatriz de cortes verticales $\boldsymbol{G_C}$ del core $\underline{\mathbf{G}}$ corresponde a la transpuesta de la matriz de cargas del modelo de 2 vías. La supermatriz de cortes verticales $\boldsymbol{X_C}$ del tensor $\underline{\mathbf{X}}$ es la correspondiente matriz de datos del modelo de 2 vías que se va a aproximar.

En resumen, el modelo Tucker1-A equivale a un análisis de componentes de tablas de 2 vías para la supermatriz de cortes frontales $\boldsymbol{X_A}$, el modelo Tucker1-B equivale a un análisis de componentes de tablas de 2 vías para la supermatriz de cortes horizontales $\boldsymbol{X_B}$ y el modelo Tucker1-C equivale a un análisis de componentes de tablas de 2 vías para la supermatriz de cortes verticales $\boldsymbol{X_C}$. Para el cálculo de componentes ortogonales en los 3 modelos Tucker1 podemos utilizar cualquiera de los algoritmos conocidos en el análisis de componentes principales de tablas de 2 vías. Por tal razón, en este documento, se decidió no incluir los algoritmos TuckALS1 para los 3 modelos Tucker1. El fit, en los 3 modelos Tucker1, se calcula igual que en el modelo PARAFAC y que en el modelo Tucker3.

Como se podrá ver en el **Capítulo 4**, se pueden calcular componentes ortogonales disjuntos en los 3 modelos Tucker1 sin la necesidad de emplear el algoritmo TuckALS1 como un algoritmo auxiliar dentro del algoritmo CBPSO-TuckALS1.

2.6 MODELOS DE OPTIMIZACIÓN UTILIZADOS EN EL ANÁLISIS DE COMPONENTES PARA TABLAS DE 3 VÍAS

Un problema de optimización, en general, se modela matemáticamente con los siguientes elementos:

- El tipo de optimización
- La función objetivo
- El espacio de búsqueda

El tipo de optimización es si se trata de una maximización o de una minimización. La función objetivo es la función que se desea optimizar y el espacio de búsqueda es el conjunto donde se encuentra la solución, si existe.

Sea Φ un conjunto de interés y f una función que permite "valorar" a cada elemento de Φ . Entonces f es una función con la siguiente estructura:

$$\Phi \rightarrow \mathbb{R}$$

$$\phi \longmapsto f(\phi)$$

Si $\phi \in \Phi$, entonces $f(\phi)$ representa su valoración. El valor de cada elemento de Φ es un número real. Sea Ω un conjunto tal que $\Omega \subseteq \Phi, \Omega \neq \emptyset$.

El problema de optimizar f en el conjunto Ω es representado mediante el modelo matemático que se presenta a continuación:

$$\min|\max \ \ f(\phi)$$

$$sujeto\ a\colon \phi \in \Omega$$

Donde $\min|\max$ indica el tipo de optimización, f es la función objetivo y Ω es el espacio de búsqueda (también se lo conoce como el espacio de soluciones factibles) que está directamente relacionado con las restricciones del problema. Si $\Omega = \Phi$, se dice que se trata de una optimización sin restricciones. Si estrictamente $\Omega \subset \Phi$, se dice que se trata de un problema restringido o con restricciones.

Se dice que $\phi^* \in \Omega$ es una solución del problema de optimización si y sólo si:

$$\forall \phi \in \Omega: f(\phi^*) \leq f(\phi) \qquad \text{(MINIMIZACIÓN)}$$

$$\forall \phi \in \Omega: f(\phi^*) \geq f(\phi) \qquad \text{(MAXIMIZACIÓN)}$$

En otras palabras, resolver un problema de optimización consiste en encontrar aquel elemento ϕ^* que se encuentra en el conjunto Ω que hace que la función f tome el menor valor (minimización) o que tome el mayor valor (maximización). De existir, al elemento ϕ^* se le llama el "óptimo del problema".

Preguntas típicas ante un problema de optimización son las siguientes:

- ¿El problema tiene solución?
- De tener solución, ¿es única?
- ¿Podemos encontrar analíticamente una solución?

Respecto de la primera pregunta un problema de optimización podría no tener solución, por lo tanto se trata de una pregunta relacionada con la existencia del problema. La segunda pregunta está relacionada con la unicidad del problema, es decir, un problema de optimización puede tener una única solución o puede presentar soluciones óptimas alternativas. Para la última pregunta la respuesta es que en general no podemos encontrar de forma analítica una solución, por lo que debemos recurrir a métodos numéricos. La función objetivo f (lineal o no lineal) y, el tamaño y las características del espacio de búsqueda Ω pueden complicar el encontrar el óptimo de un problema de optimización. Respecto de los métodos numéricos tenemos los métodos exactos que, bajo ciertas condiciones, garantizan encontrar una solución del problema mediante la convergencia de alguna sucesión y, los métodos inexactos (como las heurísticas) que hacen el "mejor esfuerzo" por encontrar la solución (durante su ejecución pueden cambiar la estrategia de búsqueda del óptimo), o por obtener, al menos, una "buena solución", es decir, una solución que se encuentre tan cerca del óptimo como sea posible. Los métodos inexactos, por tanto, no garantizan encontrar la solución del problema, pero eso no quiere decir que no la puedan encontrar.

Los métodos exactos y los métodos inexactos se expresan en términos de algoritmos. Luego, estos algoritmos se implementan mediante algún lenguaje de programación y se ejecutan en un computador. Los recursos computacionales, como memoria principal y procesador, son siempre una limitación para resolver un problema de optimización, al menos en un tiempo razonable.

Para el cálculo de componentes en el análisis de tablas de 3 vías, tanto en el modelo PARAFAC como en los modelos Tucker, se debe resolver un problema de minimización. Los algoritmos de mínimos cuadrados alternantes que se han revisado en este capítulo son métodos inexactos que no garantizan teóricamente que pueden encontrar el óptimo del problema, pero en la práctica presentan buenos resultados. Los modelos matemáticos de los problemas de optimización en el análisis de componentes de tablas de 3 vías se presentan a continuación:

Modelo de Optimización 2.6.1: PARAFAC sin restricciones

$$\min_{\boldsymbol{A},\boldsymbol{B},\boldsymbol{C}} f(\boldsymbol{A},\boldsymbol{B},\boldsymbol{C}) = \|\boldsymbol{X_A} - \boldsymbol{A}(\boldsymbol{C} \odot \boldsymbol{B})^T\|^2$$

Para el **Modelo de Optimización 2.6.1** se utilizó la **Ecuación 2.2.3** del primer modo del modelo PARAFAC. Sin embargo, se pueden obtener modelos de optimización equivalentes usando las ecuaciones de los otros modos. Se trata de un modelo de optimización sin restricciones, ya que en general los componentes en el modelo PARAFAC pueden ser oblicuos. El objetivo es minimizar la suma de los cuadrados de las entradas del tensor de residuos $\underline{\mathbf{E}}$, lo que equivale a minimizar $\|\boldsymbol{E_A}\|^2$. Cualquier algoritmo que se use para resolver este modelo de optimización debe encontrar las matrices $\boldsymbol{A}$, $\boldsymbol{B}$ y $\boldsymbol{C}$ que hacen que $\|\boldsymbol{E_A}\|^2$ tome el menor valor posible. El **Algoritmo 2.2** está diseñado para resolver este modelo de optimización.

Modelo de Optimización 2.6.2: Tucker3 con restricciones de ortogonalidad

$$\min_{\boldsymbol{A},\boldsymbol{B},\boldsymbol{C},\boldsymbol{G_A}} f(\boldsymbol{A},\boldsymbol{B},\boldsymbol{C},\boldsymbol{G_A}) = \|\boldsymbol{X_A} - \boldsymbol{A}\boldsymbol{G_A}(\boldsymbol{C} \otimes \boldsymbol{B})^T\|^2$$

$$sujeto\ a\text{:}$$

$$\boldsymbol{A}^T\boldsymbol{A} = \boldsymbol{I}_{P\times P}$$

$$\boldsymbol{B}^T\boldsymbol{B} = \boldsymbol{I}_{Q\times Q}$$

$$\boldsymbol{C}^T\boldsymbol{C} = \boldsymbol{I}_{R\times R}$$

Para el **Modelo de Optimización 2.6.2** se utilizó la **Ecuación 2.3.3** del primer modo del modelo Tucker3. Sin embargo, se pueden obtener modelos de optimización equivalentes usando las ecuaciones de los otros modos. Se trata de un modelo de optimización con restricciones, ya que el objetivo es calcular componentes ortogonales en el modelo Tucker3. Si se quitan las restricciones, el cálculo en general es de componentes oblicuos. El objetivo es minimizar la suma de los cuadrados de las entradas del tensor de residuos $\underline{\mathbf{E}}$, lo que equivale a minimizar $\|\boldsymbol{E_A}\|^2$. Cualquier algoritmo que se use para resolver este modelo de optimización debe encontrar las matrices $\boldsymbol{A}$, $\boldsymbol{B}$ y $\boldsymbol{C}$, y además el core, que hacen que $\|\boldsymbol{E_A}\|^2$ tome el valor más pequeño posible. El **Algoritmo 2.3** está diseñado para resolver este modelo de optimización.

La restricción $\boldsymbol{A}^T\boldsymbol{A} = \boldsymbol{I}_{P\times P}$, donde $\boldsymbol{I}_{P\times P}$ es la matriz identidad de tamaño $P \times P$, es para el cálculo de componentes ortogonales en la matriz de cargas $\boldsymbol{A}$. En otras palabras, con tal restricción, las columnas de $\boldsymbol{A}$, que es una matriz de tamaño $I \times P$, constituyen un conjunto ortonormal de P vectores en el espacio vectorial $\mathbb{R}^I$. De la misma manera, las restricciones $\boldsymbol{B}^T\boldsymbol{B} = \boldsymbol{I}_{Q\times Q}$ y $\boldsymbol{C}^T\boldsymbol{C} = \boldsymbol{I}_{R\times R}$ permiten el cálculo de componentes ortogonales en las matrices de cargas $\boldsymbol{B}$ y $\boldsymbol{C}$.

Modelo de Optimización 2.6.3: Tucker2-AB con restricciones de ortogonalidad

---

$$\min_{\boldsymbol{A},\boldsymbol{B},\boldsymbol{G_C}} f(\boldsymbol{A},\boldsymbol{B},\boldsymbol{G_C}) = \|\boldsymbol{X_C} - \boldsymbol{G_C}(\boldsymbol{B}\otimes\boldsymbol{A})^T\|^2$$

$$sujeto\ a\text{:}$$

$$\boldsymbol{A}^T\boldsymbol{A} = \boldsymbol{I}_{P\times P}$$

$$\boldsymbol{B}^T\boldsymbol{B} = \boldsymbol{I}_{Q\times Q}$$

Para el **Modelo de Optimización 2.6.3** se utilizó la **Ecuación 2.4.1** del modelo multivariante Tucker2-AB. Se trata de un modelo de optimización con restricciones, ya que el objetivo es calcular componentes ortogonales en el modelo Tucker2-AB. Si se quitan las restricciones, el cálculo en general es de componentes oblicuos. El objetivo es minimizar la suma de los cuadrados de las entradas del tensor de residuos $\underline{\mathbf{E}}$, lo que equivale a minimizar $\|\boldsymbol{E_C}\|^2$. Cualquier algoritmo que se use para resolver este modelo de optimización debe encontrar las matrices $\boldsymbol{A}$ y $\boldsymbol{B}$, y además el core, que hacen que $\|\boldsymbol{E_C}\|^2$ tome el valor más pequeño posible. El **Algoritmo 2.4.1** está diseñado para resolver este modelo de optimización. Las matrices de cargas $\boldsymbol{A}$ y $\boldsymbol{B}$ tienen restricciones de ortogonalidad.

Modelo de Optimización 2.6.4: Tucker2-AC con restricciones de ortogonalidad

$$\min_{\boldsymbol{A},\boldsymbol{C},\boldsymbol{G_B}} f(\boldsymbol{A},\boldsymbol{C},\boldsymbol{G_B}) = \|\boldsymbol{X_B} - \boldsymbol{G_B}(\boldsymbol{C} \otimes \boldsymbol{A})^T\|^2$$

$$sujeto\ a\text{:}$$

$$\boldsymbol{A}^T\boldsymbol{A} = \boldsymbol{I}_{P\times P}$$

$$\boldsymbol{C}^T\boldsymbol{C} = \boldsymbol{I}_{R\times R}$$

Para el **Modelo de Optimización 2.6.4** se utilizó la **Ecuación 2.4.3** del modelo multivariante Tucker2-AC. Se trata de un modelo de optimización con restricciones, ya que el objetivo es calcular componentes ortogonales en el modelo Tucker2-AC. Si se quitan las restricciones, el cálculo en general es de componentes oblicuos. El objetivo es minimizar la suma de los cuadrados de las entradas del tensor de residuos $\underline{\mathbf{E}}$, lo que equivale a minimizar $\|\boldsymbol{E_B}\|^2$. Cualquier algoritmo que se use para resolver este modelo de optimización debe encontrar las matrices $\boldsymbol{A}$ y $\boldsymbol{C}$, y además el core, que hacen que $\|\boldsymbol{E_B}\|^2$ tome el valor más pequeño posible. El **Algoritmo 2.4.2** está diseñado para resolver este modelo de optimización. Las matrices de cargas $\boldsymbol{A}$ y $\boldsymbol{C}$ tienen restricciones de ortogonalidad.

Modelo de Optimización 2.6.5: Tucker2-BC con restricciones de ortogonalidad

$$\min_{\boldsymbol{B},\boldsymbol{C},\boldsymbol{G_A}} f(\boldsymbol{B},\boldsymbol{C},\boldsymbol{G_A}) = \|\boldsymbol{X_A} - \boldsymbol{G_A}(\boldsymbol{C} \otimes \boldsymbol{B})^T\|^2$$

$$sujeto\ a\text{:}$$

$$\boldsymbol{B}^T\boldsymbol{B} = \boldsymbol{I}_{Q\times Q}$$

$$\boldsymbol{C}^T\boldsymbol{C} = \boldsymbol{I}_{R\times R}$$

Para el **Modelo de Optimización 2.6.5** se utilizó la **Ecuación 2.4.5** del modelo multivariante Tucker2-BC. Se trata de un modelo de optimización con restricciones, ya que el objetivo es calcular componentes ortogonales en el modelo Tucker2-BC. Si se quitan las restricciones, el cálculo en general es de componentes oblicuos. El objetivo es minimizar la suma de los cuadrados de las entradas del tensor de residuos $\underline{\mathbf{E}}$, lo que equivale a minimizar $\|\boldsymbol{E_A}\|^2$. Cualquier algoritmo que se use para resolver este modelo de optimización debe encontrar las matrices $\boldsymbol{B}$ y $\boldsymbol{C}$, y además el core, que hacen que $\|\boldsymbol{E_A}\|^2$ tome el valor más pequeño posible. El **Algoritmo 2.4.3** está diseñado para resolver este modelo de optimización. Las matrices de cargas $\boldsymbol{B}$ y $\boldsymbol{C}$ tienen restricciones de ortogonalidad.

Modelo de Optimización 2.6.6: Tucker1-A con restricciones de ortogonalidad

$$\min_{\boldsymbol{A},\boldsymbol{G_A}} f(\boldsymbol{A},\boldsymbol{G_A}) = \|\boldsymbol{X_A} - \boldsymbol{A}\boldsymbol{G_A}\|^2$$
$$sujeto\ a\text{:}$$
$$\boldsymbol{A}^T\boldsymbol{A} = \boldsymbol{I}_{P\times P}$$

Para el **Modelo de Optimización 2.6.6** se utilizó la **Ecuación 2.5.1** del modelo multivariante Tucker1-A. Se trata de un modelo de optimización con una restricción de ortogonalidad para la matriz de cargas $\boldsymbol{A}$, ya que el objetivo es calcular componentes ortogonales en el modelo Tucker1-A. Si se quita la restricción, el cálculo en general es de componentes oblicuos. El objetivo es minimizar la suma de los cuadrados de las entradas del tensor de residuos $\underline{\mathbf{E}}$, lo que equivale a minimizar $\|\boldsymbol{E_A}\|^2$. Cualquier algoritmo que se use para resolver este modelo de optimización debe encontrar la matriz $\boldsymbol{A}$ y el core que hacen que $\|\boldsymbol{E_A}\|^2$ tome el valor más pequeño posible.

Modelo de Optimización 2.6.7: Tucker1-B con restricciones de ortogonalidad

$$\min_{\boldsymbol{B},\boldsymbol{G_B}} f(\boldsymbol{B},\boldsymbol{G_B}) = \|\boldsymbol{X_B} - \boldsymbol{B}\boldsymbol{G_B}\|^2$$
$$sujeto\ a\text{:}$$
$$\boldsymbol{B}^T\boldsymbol{B} = \boldsymbol{I}_{Q\times Q}$$

Para el **Modelo de Optimización 2.6.7** se utilizó la **Ecuación 2.5.3** del modelo multivariante Tucker1-B. Se trata de un modelo de optimización con una restricción de ortogonalidad para la matriz de cargas $\boldsymbol{B}$, ya que el objetivo es calcular componentes ortogonales en el modelo Tucker1-B. Si se quita la restricción, el cálculo en general es de componentes oblicuos. El objetivo es minimizar la suma de los cuadrados de las entradas del tensor de residuos $\underline{\mathbf{E}}$, lo que equivale a minimizar $\|\boldsymbol{E_B}\|^2$. Cualquier algoritmo que se use para resolver este modelo de optimización debe encontrar la matriz $\boldsymbol{B}$ y el core que hacen que $\|\boldsymbol{E_B}\|^2$ tome el valor más pequeño posible.

Modelo de Optimización 2.6.8: Tucker1-C con restricciones de ortogonalidad

$$\min_{\boldsymbol{C},\boldsymbol{G_C}} f(\boldsymbol{C},\boldsymbol{G_C}) = \|\boldsymbol{X_C} - \boldsymbol{C}\boldsymbol{G_C}\|^2$$
$$sujeto\ a\text{:}$$
$$\boldsymbol{C}^T\boldsymbol{C} = \boldsymbol{I}_{R\times R}$$

Para el **Modelo de Optimización 2.6.8** se utilizó la **Ecuación 2.5.5** del modelo multivariante Tucker1-C. Se trata de un modelo de optimización con una restricción de ortogonalidad para la matriz de cargas $\boldsymbol{C}$, ya que el objetivo es calcular componentes ortogonales en el modelo Tucker1-C. Si se quita la restricción, el cálculo en general es de componentes oblicuos. El objetivo es minimizar la suma de los cuadrados de las entradas del tensor de residuos $\underline{\mathbf{E}}$, lo que equivale a minimizar $\|\boldsymbol{E_C}\|^2$. Cualquier algoritmo que se use para resolver este modelo de optimización debe encontrar la matriz $\boldsymbol{C}$ y el core que hacen que $\|\boldsymbol{E_C}\|^2$ tome el valor más pequeño posible.

Desde el **Modelo de Optimización 2.6.1** hasta el **Modelo de Optimización 2.6.8** se han mostrado los 8 modelos matemáticos de optimización relacionados con los 8 modelos multivariantes para el análisis de componentes principales en tablas de 3 vías que son, respectivamente: El modelo PARAFAC, el Modelo Tucker3, el modelo Tucker2-AB, el modelo Tucker2-AC, el modelo Tucker2-BC, el modelo Tucker1-A, el modelo Tucker1-B y el modelo Tucker1-C.

2.7 COMPLEJIDAD DE UN MODELO

Para seleccionar el modelo multivariante de componentes a utilizar para analizar una tabla de 3 vías, se deben considerar algunos factores. Entre ellos, el número de parámetros del modelo, el fit del modelo, los recursos computacionales consumidos por los algoritmos correspondientes (memoria y tiempo de procesador), la presencia del problema de degeneración, la convergencia, la unicidad de la solución, la libertad rotacional, entre otros (ver Smilde et al. 2004). En la práctica, puede ser bueno más de un modelo.

Cuando un modelo usa más parámetros el fit es mejor, pero al tener más parámetros en el modelo se debe estudiar la variabilidad de las estimaciones de cada parámetro (por ejemplo a través de un bootstrap). Además los algoritmos tardan más en estimar los parámetros ya que consumen más recursos computacionales. Por tal razón, en ocasiones puede ser más conveniente un modelo con menos parámetros y sacrificar algo de fit (ver Bro 1998). A continuación en la **Tabla 2.7.1** se muestra el número de parámetros a estimar en cada uno de los 8 modelos para realizar un análisis de componentes para tablas de 3 vías:

Tabla 2.7.1: Número de parámetros por modelo

MODELO	NÚMERO DE PARÁMETROS
PARAFAC	$R(I+J+K)$
Tucker3	$PQR+IP+JQ+KR$
Tucker2-AB	$PQK+IP+JQ$
Tucker2-AC	$PJR+IP+KR$
Tucker2-BC	$IQR+JQ+KR$
Tucker1-A	$PJK+IP$
Tucker1-B	$IQK+JQ$
Tucker1-C	$IJR+KR$

Para ilustrar las expresiones anteriores, supongamos que tenemos una tabla de 3 vías $\underline{\mathbf{X}}$ de tamaño $100 \times 20 \times 15$. Si elegimos 2 componentes en cada modo en el que deseamos reducir la dimensión, tendríamos los valores mostrados en la **Tabla 2.7.2**:

Tabla 2.7.2: Ilustración de la Tabla 2.7.1 con 2 componentes en cada modo

MODELO	NÚMERO DE PARÁMETROS
PARAFAC	$R(I+J+K)=270$
Tucker3	$PQR+IP+JQ+KR=278$
Tucker2-AB	$PQK+IP+JQ=300$
Tucker2-AC	$PJR+IP+KR=310$
Tucker2-BC	$IQR+JQ+KR=470$
Tucker1-A	$PJK+IP=800$
Tucker1-B	$IQK+JQ=3040$
Tucker1-C	$IJR+KR=4030$

Por otro lado, si elegimos 3 componentes en cada modo en el que deseamos reducir la dimensión, tendríamos los valores mostrados en la **Tabla 2.7.3** que se muestra a continuación:

Tabla 2.7.3: Ilustración de la Tabla 2.7.1 con 3 componentes en cada modo

MODELO	NÚMERO DE PARÁMETROS
PARAFAC	$R(I+J+K)=405$
Tucker3	$PQR+IP+JQ+KR=432$
Tucker2-AB	$PQK+IP+JQ=495$
Tucker2-AC	$PJR+IP+KR=525$
Tucker2-BC	$IQR+JQ+KR=1005$
Tucker1-A	$PJK+IP=1200$
Tucker1-B	$IQK+JQ=4560$
Tucker1-C	$IJR+KR=6045$

Si se tiene más de un modelo con igual fit (o un fit similar), la recomendación es utilizar el modelo que usa el menor número de parámetros, es decir, hacer uso del modelo más sencillo. Para un análisis en particular, se puede utilizar más de un modelo para entre ellos complementar el estudio de la tabla de 3 vías. Ante un igual número de componentes en la reducción dimensional, el modelo PARAFAC es el más sencillo. Luego, en complejidad viene el modelo Tucker3, posteriormente los modelos Tucker2 y finalmente los modelos Tucker1. Al aumentar el número de parámetros el fit del modelo será mejor, pero habrá que considerar la variabilidad de las estimaciones de cada parámetro, y el algoritmo usado para la estimación de dichos parámetros consumirá más recursos computacionales. Para seleccionar un modelo en particular, y el número de componentes a utilizar en dicho modelo, la complejidad del modelo es un factor a considerar. Por lo anteriormente expuesto, la complejidad de un modelo multivariante de componentes se define como el número de parámetros a estimar.

2.8 SELECCIÓN DEL NÚMERO DE COMPONENTES

Para determinar el número de componentes en un modelo multivariante para tablas de 3 vías se pueden emplear los siguientes métodos:

1.- El método de "diferencia en fit" o método de "diferencia en ajuste del modelo" (DIFFIT, por sus siglas en inglés). Es propuesto por Timmerman and Kiers 2000 y es un método que permite determinar el número de componentes a utilizar en un modelo Tucker3. Este método encuentra un equilibrio entre el número total de componentes del modelo Tucker3 y el fit del modelo.

Para P componentes en el primer modo, Q componentes en el segundo modo y R componentes en el tercer modo, el número total de componentes en un modelo Tucker3 está dado por $S = P + Q + R$. La cantidad de modelos Tucker3 a considerar dependerá del analista o investigador, pero los autores sugieren que sólo deben tomarse en cuenta aquellos modelos para los cuales $PQ \geq R$, $PR \geq Q$ y $QR \geq P$.

El número máximo de componentes en este método, denotado S_{max}, para una tabla de 3 vías con I individuos, J variables y K situaciones (o tiempos) está dado por la expresión:

$$S_{max} = max(I, JK) + max(J, IK) + max(K, IJ)$$

Una desventaja importante del método DIFFIT es el tiempo de procesamiento, ya que no sólo debe calcular el fit de todos los modelos Tucker3 considerados, sino que también debe realizar cálculos adicionales de diferencia de fit y de factor de selección (ver el detalle del algoritmo en Timmerman and Kiers 2000).

2.- El método numérico de la "envolvente convexa" o "escudo convexo" (NCH, por sus siglas en inglés). Este método es propuesto por Ceulemans and Kiers 2006 y utiliza el número de parámetros libres del modelo y el fit o ajuste del modelo para determinar el modelo más adecuado y el número de componentes dentro de dicho modelo (ver el detalle del algoritmo en Ceulemans and Kiers 2006).

Para cada uno de los 8 modelos multivariantes de análisis de componentes vistos en este capítulo, el número de parámetros libres (ver Weesie and Houwelingen 1983, Smilde et al. 2004) se presenta en la **Tabla 2.8.1**:

Tabla 2.8.1: Número de parámetros libres en los modelos

MODELO	NÚMERO DE PARÁMETROS LIBRES
PARAFAC	$R(I+J+K)-2R$
Tucker3	$PQR+IP+JQ+KR-P^2-Q^2-R^2$
Tucker2-AB	$PQK+IP+JQ-P^2-Q^2$
Tucker2-AC	$PJR+IP+KR-P^2-R^2$
Tucker2-BC	$IQR+JQ+KR-Q^2-R^2$
Tucker1-A	$PJK+IP-P^2$
Tucker1-B	$IQK+JQ-Q^2$
Tucker1-C	$IJR+KR-R^2$

Igual que con el método DIFFIT, se pueden considerar únicamente ciertos modelos fijando cotas superiores para los valores de P, Q y R. El cálculo del fit, las estrategias de eliminación de modelos y el cálculo del factor de selección, hacen que el algoritmo asociado a este método consuma recursos computacionales de forma importante. Mientras más modelos se tomen en cuenta para aplicar este método NCH, una mayor demanda de memoria principal y de tiempo de procesador habrá. El número de modelos a considerar queda a criterio del analista de los datos.

3.- El método genérico de la "envolvente convexa" o "escudo convexo" (C-HULL, por sus siglas en inglés). Este método propuesto por Wilderjans et al. 2013 es similar al método NCH, pero en lugar de usar el número de parámetros libres del modelo utiliza el número de parámetros a estimar (ver **Tabla 2.7.1**). Permite además para los cálculos la flexibilidad de escoger como variable dependiente (eje vertical):

a) El fit del modelo, o

b) La variabilidad no explicada por el modelo (el valor de desajuste del modelo).

En ambos casos la variable independiente (eje horizontal) es el número de parámetros a estimar del modelo. Ver el detalle del algoritmo en Wilderjans et al. 2013. Se trata también de un método que consume una cantidad significativa de recursos computacionales (memoria y tiempo de procesador).

2.9 ANÁLISIS COMPARATIVO ENTRE EL MODELO PARAFAC Y EL MODELO TUCKER3

A continuación se presenta la **Tabla 2.9.1** que resume comparativamente los modelos PARAFAC y Tucker3, considerados los modelos multivariantes más populares para el análisis de componentes en tablas de 3 vías:

Tabla 2.9.1: Análisis comparativo de los modelos PARAFAC y Tucker3

MODELO PARAFAC	MODELO TUCKER3
Puede presentar soluciones degeneradas.	No presenta soluciones degeneradas.
Unicidad en la solución del modelo.	Tiene muchas soluciones (no unicidad).
No tiene libertad rotacional (propiedad de los ejes intrínsecos), usa la técnica de escalado como una opción para alcanzar estructura simple en las matrices de cargas.	Tiene libertad rotacional. Con la intención de alcanzar una estructura simple, se pueden implementar rotaciones oblicuas o rotaciones ortogonales, tanto para el core como para las matrices de cargas.
Usa principalmente el producto de Khatri-Rao en su modelo matemático.	Usa principalmente el producto de Kronecker en su modelo matemático.
Modelo matemático más sencillo (menos complejo), pero con menor fit.	Modelo matemático más complejo, pero con mayor fit.
Modelo de optimización sin restricciones, en general ejes oblicuos (se recomienda no imponer restricciones de ortogonalidad).	Es conveniente un modelo de optimización con restricciones de ortogonalidad en las matrices de cargas.
Es un caso particular del modelo Tucker3.	Es el modelo matemático que generaliza el análisis de componentes para tablas de 3 vías.
No soporta interacción entre todos los componentes.	Soporta interacción entre todos los componentes.
ParafacALS suele consumir mucho tiempo de procesamiento y es muy propenso a caer en óptimos locales.	TuckALS3 suele alcanzar el óptimo global en unas pocas iteraciones (menor tiempo de procesamiento).
Modelo más fácil de interpretar (no tiene core).	Modelo más difícil de interpretar por la presencia del core.

CAPÍTULO TRES: "COMPONENTES ORTOGONALES DISJUNTOS"

En este capítulo se presenta la definición de matriz ortogonal disjunta, y además se justifica cómo pueden ser útiles este tipo de matrices en el análisis de componentes para tablas de 3 vías. Luego se muestran los distintos modelos de optimización que se deben resolver para el cálculo de componentes ortogonales disjuntos en los 8 modelos (PARAFAC y Tucker) vistos en el **Capítulo 2**. Posteriormente mostramos el principal beneficio de un algoritmo denominado CBPSO-DC, el cual es la obtención de una matriz de cargas con estructura simple, mediante el cálculo de componentes ortogonales disjuntos en tablas de 2 vías o matrices de datos. Este algoritmo sirve como punto de inicio de los algoritmos propuestos en el **Capítulo 4** para el cálculo de componentes ortogonales disjuntos en tablas de 3 vías. Finalmente el capítulo termina con una propuesta metodológica a la que se ha denominado "la metodología disjunta" donde se recomienda en qué parte del análisis de una tabla de 3 vías realizar el cálculo de los componentes ortogonales disjuntos.

3.1 MATRIZ ORTOGONAL DISJUNTA

Sea $\boldsymbol{A}$ una matriz de tamaño $I \times J$. Se dice que $\boldsymbol{A}$ es una matriz disjunta si y sólo si se cumplen las siguientes propiedades:

- $\forall i \in \{1, \dots, I\}\ \exists!\ j \in \{1, \dots, J\}$: $a_{ij} \neq 0$
- $\forall j \in \{1, \dots, J\}\ \exists i \in \{1, \dots, I\}$: $a_{ij} \neq 0$

La primera propiedad nos dice que por cada renglón de $\boldsymbol{A}$ debe existir una <u>única</u> entrada de la matriz que sea diferente de cero.

La segunda propiedad nos dice que por cada columna de $\boldsymbol{A}$ debe existir <u>al menos una</u> entrada de la matriz que sea diferente de cero.

Una matriz $\boldsymbol{A}$ de tamaño $I \times J$ es una matriz ortogonal disjunta si y sólo si:

- $\boldsymbol{A}$ es disjunta
- $\boldsymbol{A}^T\boldsymbol{A} = \boldsymbol{I}_{J\times J}$ ($\boldsymbol{A}$ es ortogonal)

Donde $\boldsymbol{I}_{J\times J}$ es la matriz identidad de tamaño $J \times J$. Se muestra una ilustración de una matriz $\boldsymbol{A}$ de tamaño 6×3 que es ortogonal disjunta:

$$\boldsymbol{A} = \begin{pmatrix} \frac{1}{\sqrt{2}} & 0 & 0 \\ 0 & 0 & \frac{1}{\sqrt{3}} \\ 0 & 0 & \frac{1}{\sqrt{3}} \\ \frac{-1}{\sqrt{2}} & 0 & 0 \\ 0 & 0 & \frac{1}{\sqrt{3}} \\ 0 & -1 & 0 \end{pmatrix}$$

¿En qué nos pueden ayudar las matrices disjuntas en el análisis de componentes para tablas de 3 vías? Suponga que $\boldsymbol{A}$ es una matriz de cargas en cualquiera de los modelos multivariantes de componentes para tablas de 3 vías del **Capítulo 2**. Sin pérdida de generalidad, suponga además que se trata de la matriz de cargas en el modo de los individuos. En términos interpretativos, se conoce que a nivel de los renglones se tienen los individuos originales, mientras que a nivel de columnas se tienen los individuos latentes, tal como podemos apreciar en la **Tabla 3.1.1**:

Tabla 3.1.1: Matriz de cargas "ortogonal disjunta" de individuos

	$\boldsymbol{sy_1}$	$\boldsymbol{sy_2}$	$\boldsymbol{sy_3}$
$\boldsymbol{sx_1}$	$\frac{1}{\sqrt{2}}$	0	0
$\boldsymbol{sx_2}$	0	0	$\frac{1}{\sqrt{3}}$
$\boldsymbol{sx_3}$	0	0	$\frac{1}{\sqrt{3}}$
$\boldsymbol{sx_4}$	$\frac{-1}{\sqrt{2}}$	0	0
$\boldsymbol{sx_5}$	0	0	$\frac{1}{\sqrt{3}}$
$\boldsymbol{sx_6}$	0	-1	0

Tenemos 6 individuos originales $sx_1, \dots, sx_6$ pero 3 individuos latentes sy_1, sy_2, sy_3 (la matriz $\boldsymbol{A}$ es de tamaño 6×3). Es decir, el modo de los individuos se ha reducido a 3 componentes. Debido a que la matriz ortogonal $\boldsymbol{A}$, es también disjunta, se tiene una estructura simple en ella que facilita su interpretación. De forma sencilla, al observar la matriz de cargas del ejemplo, podemos concluir lo siguiente:

- El individuo latente sy_1 representa a los individuos originales sx_1 y sx_4
- El individuo latente sy_2 representa al individuo original sx_6
- El individuo latente sy_3 representa a los individuos originales sx_2, sx_3 y sx_5

Al imponer, en una matriz de cargas, la restricción de que sea una matriz ortogonal disjunta, se facilita la interpretación de la matriz debido a su estructura simple. En general, con tal restricción, se obliga a que cada entidad original sea representada por una y sólo una entidad latente, eliminando de esta manera cualquier ambigüedad. Además, se obliga a participar a todas las entidades latentes, asignándoles al menos una entidad original del modo.

Esta es la propuesta principal de esta tesis, el diseñar e implementar algoritmos que permitan obtener una, dos o hasta tres matrices de cargas disjuntas, dependiendo del modelo multivariante de componentes para tablas de 3 vías. De esta manera se puede facilitar el análisis de tablas de 3 vías al trabajar con matrices de estructura simple.

3.2 MODELOS DE OPTIMIZACIÓN CON RESTRICCIONES DE MATRICES ORTOGONALES DISJUNTAS EN EL ANÁLISIS DE COMPONENTES PARA TABLAS DE 3 VÍAS

Para el cálculo de componentes ortogonales disjuntos en las matrices de cargas de los modelos multivariantes de componentes para tablas de 3 vías, se necesita imponer restricciones de matrices ortogonales disjuntas en los correspondientes modelos de optimización.

En esta tesis se proponen algoritmos heurísticos (ver **Capítulo 4**) para el cálculo de componentes ortogonales disjuntos en el modelo PARAFAC, en el modelo Tucker3, en los 3 modelos Tucker2 y en los 3 modelos Tucker1. Los algoritmos propuestos resuelven modelos de optimización que tienen restricciones de matrices ortogonales disjuntas. Para todos los modelos multivariantes (ver los experimentos computacionales del **Capítulo 5**), mientras más matrices de cargas son ortogonales disjuntas en un modelo en particular, se pierde más fit en dicho modelo.

El número de matrices de cargas que pueden ser ortogonales disjuntas varía según el modelo multivariante de componentes, lo que se aprecia en la **Tabla 3.2.1**:

Tabla 3.2.1: Número de matrices ortogonales disjuntas por modelo multivariante

MODELO	NÚMERO DE MATRICES DE CARGAS ORTOGONALES DISJUNTAS
PARAFAC	$1, 2\ o\ 3$
Tucker3	$1, 2\ o\ 3$
Tucker2-AB	$1\ o\ 2$
Tucker2-AC	$1\ o\ 2$
Tucker2-BC	$1\ o\ 2$
Tucker1-A	1
Tucker1-B	1
Tucker1-C	1

La tabla anterior nos dice que el modelo PARAFAC puede tener hasta 3 matrices de cargas ortogonales disjuntas, igual que el modelo Tucker3. Los 3 modelos Tucker2 pueden tener hasta 2 matrices de cargas ortogonales disjuntas. Finalmente, cualquiera de los 3 modelos Tucker1 puede tener únicamente una matriz de cargas ortogonal disjunta.

3.2.1 MODELO PARAFAC CON MATRICES ORTOGONALES DISJUNTAS

Un algoritmo que desee calcular componentes ortogonales disjuntos únicamente en la matriz de cargas $\boldsymbol{A}$ de un modelo PARAFAC, debe resolver el **Modelo de Optimización 3.2.1.1** que se muestra:

Modelo de Optimización 3.2.1.1: PARAFAC donde $\boldsymbol{A}$ es ortogonal disjunta

$$\min_{\boldsymbol{A},\boldsymbol{B},\boldsymbol{C}} f(\boldsymbol{A},\boldsymbol{B},\boldsymbol{C}) = \|\boldsymbol{X_A} - \boldsymbol{A}(\boldsymbol{C} \odot \boldsymbol{B})^T\|^2$$
$$sujeto\ a\text{:}$$
$$\boldsymbol{A}^T\boldsymbol{A} = \boldsymbol{I}_{R\times R}$$
$$\boldsymbol{A}\ disjunta$$

Donde $\boldsymbol{I}_{R\times R}$ es la matriz identidad de tamaño $R \times R$. Para construir el **Modelo de Optimización 3.2.1.1** se incorporó la restricción de que $\boldsymbol{A}$ sea una matriz ortogonal disjunta en el **Modelo de Optimización 2.6.1** del **Capítulo 2**.

Un algoritmo que desee calcular componentes ortogonales disjuntos únicamente en la matriz de cargas $\boldsymbol{B}$ de un modelo PARAFAC, debe resolver el **Modelo de Optimización 3.2.1.2** que se muestra:

Modelo de Optimización 3.2.1.2: PARAFAC donde $\boldsymbol{B}$ es ortogonal disjunta

$$\min_{\boldsymbol{A},\boldsymbol{B},\boldsymbol{C}} f(\boldsymbol{A},\boldsymbol{B},\boldsymbol{C}) = \|\boldsymbol{X_A} - \boldsymbol{A}(\boldsymbol{C} \odot \boldsymbol{B})^T\|^2$$
$$sujeto\ a\text{:}$$
$$\boldsymbol{B}^T\boldsymbol{B} = \boldsymbol{I}_{R\times R}$$
$$\boldsymbol{B}\ disjunta$$

Para construir el **Modelo de Optimización 3.2.1.2** se incorporó la restricción de que $\boldsymbol{B}$ sea una matriz ortogonal disjunta en el **Modelo de Optimización 2.6.1** del **Capítulo 2**.

Un algoritmo que desee calcular componentes ortogonales disjuntos únicamente en la matriz de cargas $\boldsymbol{C}$ de un modelo PARAFAC, debe resolver el **Modelo de Optimización 3.2.1.3** que se muestra:

Modelo de Optimización 3.2.1.3: PARAFAC donde $\boldsymbol{C}$ es ortogonal disjunta

$$\min_{\boldsymbol{A},\boldsymbol{B},\boldsymbol{C}} f(\boldsymbol{A},\boldsymbol{B},\boldsymbol{C}) = \|\boldsymbol{X_A} - \boldsymbol{A}(\boldsymbol{C} \odot \boldsymbol{B})^T\|^2$$
$$sujeto\ a:$$
$$\boldsymbol{C}^T\boldsymbol{C} = \boldsymbol{I}_{R\times R}$$
$$\boldsymbol{C}\ disjunta$$

Para construir el **Modelo de Optimización 3.2.1.3** se incorporó la restricción de que $\boldsymbol{C}$ sea una matriz ortogonal disjunta en el **Modelo de Optimización 2.6.1** del **Capítulo 2**.

Un algoritmo que desee calcular componentes ortogonales disjuntos en las matrices de cargas $\boldsymbol{A}$ y $\boldsymbol{B}$ de un modelo PARAFAC, debe resolver el **Modelo de Optimización 3.2.1.4** que se muestra:

Modelo de Optimización 3.2.1.4: PARAFAC donde $\boldsymbol{A}$ y $\boldsymbol{B}$ son ortogonales disjuntas

$$\min_{\boldsymbol{A},\boldsymbol{B},\boldsymbol{C}} f(\boldsymbol{A},\boldsymbol{B},\boldsymbol{C}) = \|\boldsymbol{X_A} - \boldsymbol{A}(\boldsymbol{C} \odot \boldsymbol{B})^T\|^2$$
$$sujeto\ a:$$
$$\boldsymbol{A}^T\boldsymbol{A} = \boldsymbol{I}_{R\times R}$$
$$\boldsymbol{B}^T\boldsymbol{B} = \boldsymbol{I}_{R\times R}$$
$$\boldsymbol{A},\boldsymbol{B}\ disjuntas$$

Para construir el **Modelo de Optimización 3.2.1.4** se incorporó la restricción de que $\boldsymbol{A}$ y $\boldsymbol{B}$ sean ortogonales disjuntas en el **Modelo de Optimización 2.6.1** del **Capítulo 2**.

Un algoritmo que desee calcular componentes ortogonales disjuntos en las matrices de cargas $\boldsymbol{A}$ y $\boldsymbol{C}$ de un modelo PARAFAC, debe resolver el **Modelo de Optimización 3.2.1.5** que se muestra:

Modelo de Optimización 3.2.1.5: PARAFAC donde $\boldsymbol{A}$ y $\boldsymbol{C}$ son ortogonales disjuntas

$$\min_{\boldsymbol{A},\boldsymbol{B},\boldsymbol{C}} f(\boldsymbol{A},\boldsymbol{B},\boldsymbol{C}) = \|\boldsymbol{X_A} - \boldsymbol{A}(\boldsymbol{C} \odot \boldsymbol{B})^T\|^2$$
$$sujeto\ a:$$
$$\boldsymbol{A}^T\boldsymbol{A} = \boldsymbol{I}_{R\times R}$$
$$\boldsymbol{C}^T\boldsymbol{C} = \boldsymbol{I}_{R\times R}$$
$$\boldsymbol{A},\boldsymbol{C}\ disjuntas$$

Para construir el **Modelo de Optimización 3.2.1.5** se incorporó la restricción de que $\boldsymbol{A}$ y $\boldsymbol{C}$ sean ortogonales disjuntas en el **Modelo de Optimización 2.6.1** del **Capítulo 2**.

Un algoritmo que desee calcular componentes ortogonales disjuntos en las matrices de cargas $\boldsymbol{B}$ y $\boldsymbol{C}$ de un modelo PARAFAC, debe resolver el **Modelo de Optimización 3.2.1.6** que se muestra:

Modelo de Optimización 3.2.1.6: PARAFAC donde $\boldsymbol{B}$ y $\boldsymbol{C}$ son ortogonales disjuntas

$$\min_{\boldsymbol{A},\boldsymbol{B},\boldsymbol{C}} f(\boldsymbol{A},\boldsymbol{B},\boldsymbol{C}) = \|\boldsymbol{X_A} - \boldsymbol{A}(\boldsymbol{C} \odot \boldsymbol{B})^T\|^2$$
$$sujeto\ a:$$
$$\boldsymbol{B}^T\boldsymbol{B} = \boldsymbol{I}_{R\times R}$$
$$\boldsymbol{C}^T\boldsymbol{C} = \boldsymbol{I}_{R\times R}$$
$$\boldsymbol{B},\boldsymbol{C}\ disjuntas$$

Para construir el **Modelo de Optimización 3.2.1.6** se incorporó la restricción de que $\boldsymbol{B}$ y $\boldsymbol{C}$ sean ortogonales disjuntas en el **Modelo de Optimización 2.6.1** del **Capítulo 2**.

Para terminar, un algoritmo que desee calcular componentes ortogonales disjuntos en las matrices de cargas $\boldsymbol{A}$, $\boldsymbol{B}$ y $\boldsymbol{C}$ de un modelo PARAFAC, debe resolver el **Modelo de Optimización 3.2.1.7** que se muestra:

Modelo de Optimización 3.2.1.7: PARAFAC donde $\boldsymbol{A}$, $\boldsymbol{B}$ y $\boldsymbol{C}$ son ortogonales disjuntas

$$\min_{\boldsymbol{A},\boldsymbol{B},\boldsymbol{C}} f(\boldsymbol{A},\boldsymbol{B},\boldsymbol{C}) = \|\boldsymbol{X_A} - \boldsymbol{A}(\boldsymbol{C} \odot \boldsymbol{B})^T\|^2$$

$$sujeto\ a\text{:}$$

$$\boldsymbol{A}^T\boldsymbol{A} = \boldsymbol{I}_{R\times R}$$

$$\boldsymbol{B}^T\boldsymbol{B} = \boldsymbol{I}_{R\times R}$$

$$\boldsymbol{C}^T\boldsymbol{C} = \boldsymbol{I}_{R\times R}$$

$$\boldsymbol{A},\boldsymbol{B},\boldsymbol{C}\ disjuntas$$

Para construir el **Modelo de Optimización 3.2.1.7** se incorporó la restricción de que $\boldsymbol{A}$, $\boldsymbol{B}$ y $\boldsymbol{C}$ sean ortogonales disjuntas en el **Modelo de Optimización 2.6.1** del **Capítulo 2**.

3.2.2 MODELO TUCKER3 CON MATRICES ORTOGONALES DISJUNTAS

Un algoritmo que desee calcular componentes ortogonales disjuntos únicamente en la matriz de cargas $\boldsymbol{A}$ de un modelo Tucker3, debe resolver el **Modelo de Optimización 3.2.2.1** que se muestra:

Modelo de Optimización 3.2.2.1: Tucker3 donde $\boldsymbol{A}$ es ortogonal disjunta

---

$$\min_{\boldsymbol{A},\boldsymbol{B},\boldsymbol{C},\boldsymbol{G_A}} f(\boldsymbol{A},\boldsymbol{B},\boldsymbol{C},\boldsymbol{G_A}) = \|\boldsymbol{X_A} - \boldsymbol{A}\boldsymbol{G_A}(\boldsymbol{C} \otimes \boldsymbol{B})^T\|^2$$

$$sujeto\ a\text{:}$$

$$\boldsymbol{A}^T\boldsymbol{A} = \boldsymbol{I}_{P\times P}$$

$$\boldsymbol{B}^T\boldsymbol{B} = \boldsymbol{I}_{Q\times Q}$$

$$\boldsymbol{C}^T\boldsymbol{C} = \boldsymbol{I}_{R\times R}$$

$$\boldsymbol{A}\ disjunta$$

Donde $\boldsymbol{I}_{P\times P}$ es la matriz identidad de tamaño $P \times P$, $\boldsymbol{I}_{Q\times Q}$ es la matriz identidad de tamaño $Q \times Q$, $\boldsymbol{I}_{R\times R}$ es la matriz identidad de tamaño $R \times R$. Para construir el **Modelo de Optimización 3.2.2.1** se incorporó la restricción de que $\boldsymbol{A}$ sea una matriz ortogonal disjunta en el **Modelo de Optimización 2.6.2** del **Capítulo 2**.

Un algoritmo que desee calcular componentes ortogonales disjuntos únicamente en la matriz de cargas $\boldsymbol{B}$ de un modelo Tucker3, debe resolver el **Modelo de Optimización 3.2.2.2** que se muestra:

Modelo de Optimización 3.2.2.2: Tucker donde $\boldsymbol{B}$ es ortogonal disjunta

---

$$\min_{\boldsymbol{A},\boldsymbol{B},\boldsymbol{C},\boldsymbol{G_A}} f(\boldsymbol{A},\boldsymbol{B},\boldsymbol{C},\boldsymbol{G_A}) = \|\boldsymbol{X_A} - \boldsymbol{A}\boldsymbol{G_A}(\boldsymbol{C} \otimes \boldsymbol{B})^T\|^2$$

$$sujeto\ a\text{:}$$

$$\boldsymbol{A}^T\boldsymbol{A} = \boldsymbol{I}_{P\times P}$$

$$\boldsymbol{B}^T\boldsymbol{B} = \boldsymbol{I}_{Q\times Q}$$

$$\boldsymbol{C}^T\boldsymbol{C} = \boldsymbol{I}_{R\times R}$$

$$\boldsymbol{B}\ disjunta$$

Para construir el **Modelo de Optimización 3.2.2.2** se incorporó la restricción de que $\boldsymbol{B}$ sea una matriz ortogonal disjunta en el **Modelo de Optimización 2.6.2** del **Capítulo 2**.

Un algoritmo que desee calcular componentes ortogonales disjuntos únicamente en la matriz de cargas $\boldsymbol{C}$ de un modelo Tucker3, debe resolver el **Modelo de Optimización 3.2.2.3** que se muestra:

Modelo de Optimización 3.2.2.3: Tucker3 donde $\boldsymbol{C}$ es ortogonal disjunta

$$\min_{\boldsymbol{A},\boldsymbol{B},\boldsymbol{C},\boldsymbol{G_A}} f(\boldsymbol{A},\boldsymbol{B},\boldsymbol{C},\boldsymbol{G_A}) = \|\boldsymbol{X_A} - \boldsymbol{A}\boldsymbol{G_A}(\boldsymbol{C} \otimes \boldsymbol{B})^T\|^2$$

$$sujeto\ a\text{:}$$

$$\boldsymbol{A}^T\boldsymbol{A} = \boldsymbol{I}_{P\times P}$$

$$\boldsymbol{B}^T\boldsymbol{B} = \boldsymbol{I}_{Q\times Q}$$

$$\boldsymbol{C}^T\boldsymbol{C} = \boldsymbol{I}_{R\times R}$$

$$\boldsymbol{C}\ disjunta$$

Para construir el **Modelo de Optimización 3.2.2.3** se incorporó la restricción de que $\boldsymbol{C}$ sea una matriz ortogonal disjunta en el **Modelo de Optimización 2.6.2** del **Capítulo 2**.

Un algoritmo que desee calcular componentes ortogonales disjuntos en las matrices de cargas $\boldsymbol{A}$ y $\boldsymbol{B}$ de un modelo Tucker3, debe resolver el **Modelo de Optimización 3.2.2.4** que se muestra:

Modelo de Optimización 3.2.2.4: Tucker3 donde $\boldsymbol{A}$ y $\boldsymbol{B}$ son ortogonales disjuntas

$$\min_{\boldsymbol{A},\boldsymbol{B},\boldsymbol{C},\boldsymbol{G_A}} f(\boldsymbol{A},\boldsymbol{B},\boldsymbol{C},\boldsymbol{G_A}) = \|\boldsymbol{X_A} - \boldsymbol{A}\boldsymbol{G_A}(\boldsymbol{C} \otimes \boldsymbol{B})^T\|^2$$

$$sujeto\ a\text{:}$$

$$\boldsymbol{A}^T\boldsymbol{A} = \boldsymbol{I}_{P\times P}$$

$$\boldsymbol{B}^T\boldsymbol{B} = \boldsymbol{I}_{Q\times Q}$$

$$\boldsymbol{C}^T\boldsymbol{C} = \boldsymbol{I}_{R\times R}$$

$$\boldsymbol{A},\boldsymbol{B}\ disjuntas$$

Para construir el **Modelo de Optimización 3.2.2.4** se incorporó la restricción de que $\boldsymbol{A}$ y $\boldsymbol{B}$ sean ortogonales disjuntas en el **Modelo de Optimización 2.6.2** del **Capítulo 2**.

Un algoritmo que desee calcular componentes ortogonales disjuntos en las matrices de cargas $\boldsymbol{A}$ y $\boldsymbol{C}$ de un modelo Tucker3, debe resolver el **Modelo de Optimización 3.2.2.5** que se muestra:

Modelo de Optimización 3.2.2.5: Tucker3 donde $\boldsymbol{A}$ y $\boldsymbol{C}$ son ortogonales disjuntas

$$\min_{\boldsymbol{A},\boldsymbol{B},\boldsymbol{C},\boldsymbol{G_A}} f(\boldsymbol{A},\boldsymbol{B},\boldsymbol{C},\boldsymbol{G_A}) = \|\boldsymbol{X_A} - \boldsymbol{A}\boldsymbol{G_A}(\boldsymbol{C} \otimes \boldsymbol{B})^T\|^2$$

$$sujeto\ a:$$

$$\boldsymbol{A}^T\boldsymbol{A} = \boldsymbol{I}_{P\times P}$$

$$\boldsymbol{B}^T\boldsymbol{B} = \boldsymbol{I}_{Q\times Q}$$

$$\boldsymbol{C}^T\boldsymbol{C} = \boldsymbol{I}_{R\times R}$$

$$\boldsymbol{A},\boldsymbol{C}\ disjuntas$$

Para construir el **Modelo de Optimización 3.2.2.5** se incorporó la restricción de que $\boldsymbol{A}$ y $\boldsymbol{C}$ sean ortogonales disjuntas en el **Modelo de Optimización 2.6.2** del **Capítulo 2**.

Un algoritmo que desee calcular componentes ortogonales disjuntos en las matrices de cargas $\boldsymbol{B}$ y $\boldsymbol{C}$ de un modelo Tucker3, debe resolver el **Modelo de Optimización 3.2.2.6** que se muestra:

Modelo de Optimización 3.2.2.6: Tucker3 donde $\boldsymbol{B}$ y $\boldsymbol{C}$ son ortogonales disjuntas

$$\min_{\boldsymbol{A},\boldsymbol{B},\boldsymbol{C},\boldsymbol{G_A}} f(\boldsymbol{A},\boldsymbol{B},\boldsymbol{C},\boldsymbol{G_A}) = \|\boldsymbol{X_A} - \boldsymbol{A}\boldsymbol{G_A}(\boldsymbol{C} \otimes \boldsymbol{B})^T\|^2$$

$$sujeto\ a:$$

$$\boldsymbol{A}^T\boldsymbol{A} = \boldsymbol{I}_{P\times P}$$

$$\boldsymbol{B}^T\boldsymbol{B} = \boldsymbol{I}_{Q\times Q}$$

$$\boldsymbol{C}^T\boldsymbol{C} = \boldsymbol{I}_{R\times R}$$

$$\boldsymbol{B},\boldsymbol{C}\ disjuntas$$

Para construir el **Modelo de Optimización 3.2.2.6** se incorporó la restricción de que $\boldsymbol{B}$ y $\boldsymbol{C}$ sean ortogonales disjuntas en el **Modelo de Optimización 2.6.2** del **Capítulo 2**.

Finalmente, un algoritmo que desee calcular componentes ortogonales disjuntos en las matrices de cargas $\boldsymbol{A}$, $\boldsymbol{B}$ y $\boldsymbol{C}$ de un modelo Tucker3, debe resolver el **Modelo de Optimización 3.2.2.7** que se muestra:

Modelo de Optimización 3.2.2.7: Tucker3 donde $\boldsymbol{A}$, $\boldsymbol{B}$ y $\boldsymbol{C}$ son ortogonales disjuntas

$$\min_{\boldsymbol{A},\boldsymbol{B},\boldsymbol{C},\boldsymbol{G_A}} f(\boldsymbol{A},\boldsymbol{B},\boldsymbol{C},\boldsymbol{G_A}) = \|\boldsymbol{X_A} - \boldsymbol{A}\boldsymbol{G_A}(\boldsymbol{C} \otimes \boldsymbol{B})^T\|^2$$

$$sujeto\ a\text{:}$$

$$\boldsymbol{A}^T\boldsymbol{A} = \boldsymbol{I}_{P\times P}$$

$$\boldsymbol{B}^T\boldsymbol{B} = \boldsymbol{I}_{Q\times Q}$$

$$\boldsymbol{C}^T\boldsymbol{C} = \boldsymbol{I}_{R\times R}$$

$$\boldsymbol{A},\boldsymbol{B},\boldsymbol{C}\ disjuntas$$

Para construir el **Modelo de Optimización 3.2.2.7** se incorporó la restricción de que $\boldsymbol{A}$, $\boldsymbol{B}$ y $\boldsymbol{C}$ sean ortogonales disjuntas en el **Modelo de Optimización 2.6.2** del **Capítulo 2**.

3.2.3 MODELOS TUCKER2 CON MATRICES ORTOGONALES DISJUNTAS

Un algoritmo que desee calcular componentes ortogonales disjuntos únicamente en la matriz de cargas $\boldsymbol{A}$ de un modelo Tucker2-AB, debe resolver el **Modelo de Optimización 3.2.3.1** que se muestra:

Modelo de Optimización 3.2.3.1: Tucker2-AB donde $\boldsymbol{A}$ es ortogonal disjunta

---

$$\min_{\boldsymbol{A},\boldsymbol{B},\boldsymbol{G_C}} f(\boldsymbol{A},\boldsymbol{B},\boldsymbol{G_C}) = \|\boldsymbol{X_C} - \boldsymbol{G_C}(\boldsymbol{B} \otimes \boldsymbol{A})^T\|^2$$

$$sujeto\ a\text{:}$$

$$\boldsymbol{A}^T\boldsymbol{A} = \boldsymbol{I}_{P\times P}$$

$$\boldsymbol{B}^T\boldsymbol{B} = \boldsymbol{I}_{Q\times Q}$$

$$\boldsymbol{A}\ disjunta$$

Donde $\boldsymbol{I}_{P\times P}$ es la matriz identidad de tamaño $P \times P$, $\boldsymbol{I}_{Q\times Q}$ es la matriz identidad de tamaño $Q \times Q$. Para construir el **Modelo de Optimización 3.2.3.1** se incorporó la restricción de que $\boldsymbol{A}$ sea una matriz ortogonal disjunta en el **Modelo de Optimización 2.6.3** del **Capítulo 2**.

Un algoritmo que desee calcular componentes ortogonales disjuntos únicamente en la matriz de cargas $\boldsymbol{B}$ de un modelo Tucker2-AB, debe resolver el **Modelo de Optimización 3.2.3.2** que se muestra:

Modelo de Optimización 3.2.3.2: Tucker2-AB donde $\boldsymbol{B}$ es ortogonal disjunta

---

$$\min_{\boldsymbol{A},\boldsymbol{B},\boldsymbol{G_C}} f(\boldsymbol{A},\boldsymbol{B},\boldsymbol{G_C}) = \|\boldsymbol{X_C} - \boldsymbol{G_C}(\boldsymbol{B} \otimes \boldsymbol{A})^T\|^2$$

$$sujeto\ a\text{:}$$

$$\boldsymbol{A}^T\boldsymbol{A} = \boldsymbol{I}_{P\times P}$$

$$\boldsymbol{B}^T\boldsymbol{B} = \boldsymbol{I}_{Q\times Q}$$

$$\boldsymbol{B}\ disjunta$$

Para construir el **Modelo de Optimización 3.2.3.2** se incorporó la restricción de que $\boldsymbol{B}$ sea una matriz ortogonal disjunta en el **Modelo de Optimización 2.6.3** del **Capítulo 2**.

Un algoritmo que desee calcular componentes ortogonales disjuntos en las matrices de cargas $\boldsymbol{A}$ y $\boldsymbol{B}$ de un modelo Tucker2-AB, debe resolver el **Modelo de Optimización 3.2.3.3** que se muestra:

Modelo de Optimización 3.2.3.3: Tucker2-AB donde $\boldsymbol{A}$ y $\boldsymbol{B}$ son ortogonales disjuntas

$$\min_{\boldsymbol{A},\boldsymbol{B},\boldsymbol{G_C}} f(\boldsymbol{A},\boldsymbol{B},\boldsymbol{G_C}) = \|\boldsymbol{X_C} - \boldsymbol{G_C}(\boldsymbol{B} \otimes \boldsymbol{A})^T\|^2$$

$$sujeto\ a\text{:}$$

$$\boldsymbol{A}^T\boldsymbol{A} = \boldsymbol{I}_{P\times P}$$

$$\boldsymbol{B}^T\boldsymbol{B} = \boldsymbol{I}_{Q\times Q}$$

$$\boldsymbol{A},\boldsymbol{B}\ disjuntas$$

Para construir el **Modelo de Optimización 3.2.3.3** se incorporó la restricción de que $\boldsymbol{A}$ y $\boldsymbol{B}$ sean ortogonales disjuntas en el **Modelo de Optimización 2.6.3** del **Capítulo 2**.

Un algoritmo que desee calcular componentes ortogonales disjuntos únicamente en la matriz de cargas $\boldsymbol{A}$ de un modelo Tucker2-AC, debe resolver el **Modelo de Optimización 3.2.3.4** que se muestra:

Modelo de Optimización 3.2.3.4: Tucker2-AC donde $\boldsymbol{A}$ es ortogonal disjunta

$$\min_{\boldsymbol{A},\boldsymbol{C},\boldsymbol{G_B}} f(\boldsymbol{A},\boldsymbol{C},\boldsymbol{G_B}) = \|\boldsymbol{X_B} - \boldsymbol{G_B}(\boldsymbol{C} \otimes \boldsymbol{A})^T\|^2$$

$$sujeto\ a\text{:}$$

$$\boldsymbol{A}^T\boldsymbol{A} = \boldsymbol{I}_{P\times P}$$

$$\boldsymbol{C}^T\boldsymbol{C} = \boldsymbol{I}_{R\times R}$$

$$\boldsymbol{A}\ disjunta$$

Donde $\boldsymbol{I}_{P\times P}$ es la matriz identidad de tamaño $P \times P$, $\boldsymbol{I}_{R\times R}$ es la matriz identidad de tamaño $R \times R$. Para construir el **Modelo de Optimización 3.2.3.4** se incorporó la restricción de que $\boldsymbol{A}$ sea una matriz ortogonal disjunta en el **Modelo de Optimización 2.6.4** del **Capítulo 2**.

Un algoritmo que desee calcular componentes ortogonales disjuntos únicamente en la matriz de cargas $\boldsymbol{C}$ de un modelo Tucker2-AC, debe resolver el **Modelo de Optimización 3.2.3.5** que se muestra:

Modelo de Optimización 3.2.3.5: Tucker2-AC donde $\boldsymbol{C}$ es ortogonal disjunta

$$\min_{\boldsymbol{A},\boldsymbol{C},\boldsymbol{G_B}} f(\boldsymbol{A},\boldsymbol{C},\boldsymbol{G_B}) = \|\boldsymbol{X_B} - \boldsymbol{G_B}(\boldsymbol{C} \otimes \boldsymbol{A})^T\|^2$$

$$sujeto\ a\text{:}$$

$$\boldsymbol{A}^T\boldsymbol{A} = \boldsymbol{I}_{P\times P}$$

$$\boldsymbol{C}^T\boldsymbol{C} = \boldsymbol{I}_{R\times R}$$

$$\boldsymbol{C}\ disjunta$$

Para construir el **Modelo de Optimización 3.2.3.5** se incorporó la restricción de que $\boldsymbol{C}$ sea una matriz ortogonal disjunta en el **Modelo de Optimización 2.6.4** del **Capítulo 2**.

Un algoritmo que desee calcular componentes ortogonales disjuntos en las matrices de cargas $\boldsymbol{A}$ y $\boldsymbol{C}$ de un modelo Tucker2-AC, debe resolver el **Modelo de Optimización 3.2.3.6** que se muestra:

Modelo de Optimización 3.2.3.6: Tucker2-AC donde $\boldsymbol{A}$ y $\boldsymbol{C}$ son ortogonales disjuntas

$$\min_{\boldsymbol{A},\boldsymbol{C},\boldsymbol{G_B}} f(\boldsymbol{A},\boldsymbol{C},\boldsymbol{G_B}) = \|\boldsymbol{X_B} - \boldsymbol{G_B}(\boldsymbol{C} \otimes \boldsymbol{A})^T\|^2$$

$$sujeto\ a\text{:}$$

$$\boldsymbol{A}^T\boldsymbol{A} = \boldsymbol{I}_{P\times P}$$

$$\boldsymbol{C}^T\boldsymbol{C} = \boldsymbol{I}_{R\times R}$$

$$\boldsymbol{A},\boldsymbol{C}\ disjuntas$$

Para construir el **Modelo de Optimización 3.2.3.6** se incorporó la restricción de que $\boldsymbol{A}$ y $\boldsymbol{C}$ sean ortogonales disjuntas en el **Modelo de Optimización 2.6.4** del **Capítulo 2**.

Un algoritmo que desee calcular componentes ortogonales disjuntos únicamente en la matriz de cargas $\boldsymbol{B}$ de un modelo Tucker2-BC, debe resolver el **Modelo de Optimización 3.2.3.7** que se muestra:

Modelo de Optimización 3.2.3.7: Tucker2-BC donde $\boldsymbol{B}$ es ortogonal disjunta

$$\min_{\boldsymbol{B},\boldsymbol{C},\boldsymbol{G_A}} f(\boldsymbol{B},\boldsymbol{C},\boldsymbol{G_A}) = \|\boldsymbol{X_A} - \boldsymbol{G_A}(\boldsymbol{C} \otimes \boldsymbol{B})^T\|^2$$

$$sujeto\ a\text{:}$$

$$\boldsymbol{B}^T\boldsymbol{B} = \boldsymbol{I}_{Q\times Q}$$

$$\boldsymbol{C}^T\boldsymbol{C} = \boldsymbol{I}_{R\times R}$$

$$\boldsymbol{B}\ disjunta$$

Donde $\boldsymbol{I}_{Q\times Q}$ es la matriz identidad de tamaño $Q \times Q$, $\boldsymbol{I}_{R\times R}$ es la matriz identidad de tamaño $R \times R$. Para construir el **Modelo de Optimización 3.2.3.7** se incorporó la restricción de que $\boldsymbol{B}$ sea una matriz ortogonal disjunta en el **Modelo de Optimización 2.6.5** del **Capítulo 2**.

Un algoritmo que desee calcular componentes ortogonales disjuntos únicamente en la matriz de cargas $\boldsymbol{C}$ de un modelo Tucker2-BC, debe resolver el **Modelo de Optimización 3.2.3.8** que se muestra:

Modelo de Optimización 3.2.3.8: Tucker2-BC donde $\boldsymbol{C}$ es ortogonal disjunta

$$\min_{\boldsymbol{B},\boldsymbol{C},\boldsymbol{G_A}} f(\boldsymbol{B},\boldsymbol{C},\boldsymbol{G_A}) = \|\boldsymbol{X_A} - \boldsymbol{G_A}(\boldsymbol{C} \otimes \boldsymbol{B})^T\|^2$$

$$sujeto\ a\text{:}$$

$$\boldsymbol{B}^T\boldsymbol{B} = \boldsymbol{I}_{Q\times Q}$$

$$\boldsymbol{C}^T\boldsymbol{C} = \boldsymbol{I}_{R\times R}$$

$$\boldsymbol{C}\ disjunta$$

Para construir el **Modelo de Optimización 3.2.3.8** se incorporó la restricción de que $\boldsymbol{C}$ sea una matriz ortogonal disjunta en el **Modelo de Optimización 2.6.5** del **Capítulo 2**.

Un algoritmo que desee calcular componentes ortogonales disjuntos en las matrices de cargas $\boldsymbol{B}$ y $\boldsymbol{C}$ de un modelo Tucker2-BC, debe resolver el **Modelo de Optimización 3.2.3.9** que se muestra:

Modelo de Optimización 3.2.3.9: Tucker2-BC donde $\boldsymbol{B}$ y $\boldsymbol{C}$ son ortogonales disjuntas

$$\min_{\boldsymbol{B},\boldsymbol{C},\boldsymbol{G_A}} f(\boldsymbol{B},\boldsymbol{C},\boldsymbol{G_A}) = \|\boldsymbol{X_A} - \boldsymbol{G_A}(\boldsymbol{C} \otimes \boldsymbol{B})^T\|^2$$

$$sujeto\ a\text{:}$$

$$\boldsymbol{B}^T\boldsymbol{B} = \boldsymbol{I}_{Q\times Q}$$

$$\boldsymbol{C}^T\boldsymbol{C} = \boldsymbol{I}_{R\times R}$$

$$\boldsymbol{B},\boldsymbol{C}\ disjuntas$$

Para construir el **Modelo de Optimización 3.2.3.9** se incorporó la restricción de que $\boldsymbol{B}$ y $\boldsymbol{C}$ sean ortogonales disjuntas en el **Modelo de Optimización 2.6.5** del **Capítulo 2**.

3.2.4 MODELOS TUCKER1 CON MATRICES ORTOGONALES DISJUNTAS

Un algoritmo que desee calcular componentes ortogonales disjuntos únicamente en la matriz de cargas $\boldsymbol{A}$ de un modelo Tucker1-A, debe resolver el **Modelo de Optimización 3.2.4.1** que se muestra:

Modelo de Optimización 3.2.4.1: Tucker1-A donde $\boldsymbol{A}$ es ortogonal disjunta

---

$$\min_{\boldsymbol{A},\boldsymbol{G_A}} f(\boldsymbol{A},\boldsymbol{G_A}) = \|\boldsymbol{X_A} - \boldsymbol{A}\boldsymbol{G_A}\|^2$$

$$sujeto\ a\text{:}$$

$$\boldsymbol{A}^T\boldsymbol{A} = \boldsymbol{I}_{P\times P}$$

$$\boldsymbol{A}\ disjunta$$

Donde $\boldsymbol{I}_{P\times P}$ es la matriz identidad de tamaño $P \times P$. Para construir el **Modelo de Optimización 3.2.4.1** se incorporó la restricción de que $\boldsymbol{A}$ sea una matriz ortogonal disjunta en el **Modelo de Optimización 2.6.6** del **Capítulo 2**.

Un algoritmo que desee calcular componentes ortogonales disjuntos únicamente en la matriz de cargas $\boldsymbol{B}$ de un modelo Tucker1-B, debe resolver el **Modelo de Optimización 3.2.4.2** que se muestra:

Modelo de Optimización 3.2.4.2: Tucker1-B donde $\boldsymbol{B}$ es ortogonal disjunta

---

$$\min_{\boldsymbol{B},\boldsymbol{G_B}} f(\boldsymbol{B},\boldsymbol{G_B}) = \|\boldsymbol{X_B} - \boldsymbol{B}\boldsymbol{G_B}\|^2$$

$$sujeto\ a\text{:}$$

$$\boldsymbol{B}^T\boldsymbol{B} = \boldsymbol{I}_{Q\times Q}$$

$$\boldsymbol{B}\ disjunta$$

Donde $\boldsymbol{I}_{Q\times Q}$ es la matriz identidad de tamaño $Q \times Q$. Para construir el **Modelo de Optimización 3.2.4.2** se incorporó la restricción de que $\boldsymbol{B}$ sea una matriz ortogonal disjunta en el **Modelo de Optimización 2.6.7** del **Capítulo 2**.

Un algoritmo que desee calcular componentes ortogonales disjuntos únicamente en la matriz de cargas $\boldsymbol{C}$ de un modelo Tucker1-C, debe resolver el **Modelo de Optimización 3.2.4.3** que se muestra:

Modelo de Optimización 3.2.4.3: Tucker1-C donde $\boldsymbol{C}$ es ortogonal disjunta

$$\min_{\boldsymbol{C},\boldsymbol{G_C}} f(\boldsymbol{C},\boldsymbol{G_C}) = \|\boldsymbol{X_C} - \boldsymbol{C}\boldsymbol{G_C}\|^2$$
$$sujeto\ a\text{:}$$
$$\boldsymbol{C}^T\boldsymbol{C} = \boldsymbol{I}_{R\times R}$$
$$\boldsymbol{C}\ disjunta$$

Donde $\boldsymbol{I}_{R\times R}$ es la matriz identidad de tamaño $R \times R$. Para construir el **Modelo de Optimización 3.2.4.3** se incorporó la restricción de que $\boldsymbol{C}$ sea una matriz ortogonal disjunta en el **Modelo de Optimización 2.6.8** del **Capítulo 2**.

El cálculo de componentes ortogonales disjuntos en los 3 modelos Tucker1 es equivalente al cálculo de componentes ortogonales disjuntos en las tablas de 2 vías $\boldsymbol{X_A}$, $\boldsymbol{X_B}$ y $\boldsymbol{X_C}$ que representan, respectivamente, las matrices de cortes frontales, horizontales y verticales de la tabla de datos $\underline{\mathbf{X}}$ de 3 vías.

3.3 EL ALGORITMO CBPSO-DC

El algoritmo para la optimización por enjambre de partículas (PSO, por sus siglas en inglés) aparece con el trabajo conjunto de James Kennedy, un sociólogo, y Russel Eberhart, un ingeniero eléctrico (ver Kennedy and Eberhart 1995). En sus inicios el algoritmo PSO fue utilizado en el campo de la computación gráfica con la intención de simular el movimiento de una bandada de pájaros o de un banco de peces que buscan proveerse de alimento. Luego el algoritmo PSO sufrió una adaptación y tomó la forma de un algoritmo evolutivo estocástico con el objetivo de hacer su mejor esfuerzo e intentar encontrar el óptimo en problemas de optimización, tal como podemos ver en Poli et al. 2007 y García-Gonzalo and Fernández-Martínez 2012.

Diferentes tipos de algoritmos PSO han sido usados para realizar descomposiciones tensoriales, en vez de los algoritmos ALS, tal como podemos ver en Borckmans et al. 2010. En Nekouie and Moattar 2019 los autores presentan una metodología que emplea una descomposición tensorial con un algoritmo PSO para manejar un faltante de datos o datos perdidos. Fan and Wang 2017 usan un algoritmo PSO para evitar que una descomposición tensorial quede atrapada en un óptimo local.

El modelo de optimización que permite obtener, en un análisis de componentes principales (PCA, por sus siglas en inglés), una matriz de scores $\boldsymbol{A}$ y una matriz de cargas ortogonal $\boldsymbol{B}$, para una tabla de 2 vías o matriz de datos $\boldsymbol{X}$, es el siguiente:

Modelo de Optimización 3.3.1: PCA donde $\boldsymbol{B}$ es una matriz ortogonal

---

$$\min_{\boldsymbol{A},\boldsymbol{B}} f(\boldsymbol{A},\boldsymbol{B}) = \|\boldsymbol{X} - \boldsymbol{A}\boldsymbol{B}^T\|^2$$

$$sujeto\ a:$$

$$\boldsymbol{B}^T\boldsymbol{B} = \boldsymbol{I}_{Q\times Q}$$

La matriz $\boldsymbol{I}_{Q\times Q}$ es la identidad de tamaño $Q \times Q$. Suponemos que $\boldsymbol{X}$ es una matriz de datos con I individuos y J variables. El modo de las variables se reduce a Q componentes, por lo que $Q < J$. La matriz de scores $\boldsymbol{A}$ es de tamaño $I \times Q$ y contiene las coordenadas de cada uno de los I individuos en el espacio de dimensión reducida. La matriz ortogonal $\boldsymbol{B}$ de tamaño $J \times Q$ es la matriz de cargas correspondiente que nos permite relacionar el espacio original con el espacio de dimensión reducida.

Si al **Modelo de Optimización 3.3.1** le imponemos la restricción adicional de que la matriz de cargas $\boldsymbol{B}$ sea disjunta, obtenemos el **Modelo de Optimización 3.3.2** que se muestra a continuación y que permite un análisis de componentes principales disjuntos (DPCA, por sus siglas en inglés):

Modelo de Optimización 3.3.2: DPCA donde $\boldsymbol{B}$ es una matriz ortogonal disjunta

$$\min_{\boldsymbol{A},\boldsymbol{B}} f(\boldsymbol{A},\boldsymbol{B}) = \|\boldsymbol{X} - \boldsymbol{A}\boldsymbol{B}^T\|^2$$
$$sujeto\ a:$$
$$\boldsymbol{B}^T\boldsymbol{B} = \boldsymbol{I}_{Q\times Q}$$
$$\boldsymbol{B}\ disjunta$$

Si un algoritmo desea calcular componentes ortogonales disjuntos en una matriz de datos $\boldsymbol{X}$, debe resolver el **Modelo de Optimización 3.3.2** para poder obtener una matriz de cargas $\boldsymbol{B}$ ortogonal disjunta que facilite su interpretación.

En Ramirez-Figueroa et al. 2021 se propone un algoritmo denominado CBPSO-DC (optimización por enjambre de partículas binaria y con restricciones para el cálculo de componentes ortogonales disjuntos, por sus siglas en inglés) que resuelve el **Modelo de Optimización 3.3.2** para calcular componentes ortogonales disjuntos en una tabla de 2 vías $\boldsymbol{X}$. Se trata de un algoritmo PSO que es binario y con restricciones, debido a la estructura que tienen las soluciones factibles empleadas para la búsqueda del óptimo global. El **Algoritmo 3.3.1** muestra una implementación del algoritmo CBPSO-DC.

Algoritmo 3.3.1: Algoritmo CBPSO-DC

Datos de Entrada:
$\boldsymbol{X}, Q, PSOMaxIter, nPar, minIner, maxIner, wCognition, wSocial$
Inicialización aleatoria de $nPar$ número de partículas
$Inercia \leftarrow maxIner$
Para cada i desde 1 hasta $PSOMaxIter$:
  Para cada p desde 1 hasta $nPar$:
    $\boldsymbol{B_p} \leftarrow Paso(\boldsymbol{B_p}, \boldsymbol{B_p^*}, \boldsymbol{B}^*, wCognition, wSocial, Inercia)$
    $\boldsymbol{A_p} \leftarrow \boldsymbol{X}\boldsymbol{B_p}$
    Si $f(\boldsymbol{A_p}, \boldsymbol{B_p}) < f(\boldsymbol{X}\boldsymbol{B_p^*}, \boldsymbol{B_p^*})$: $\boldsymbol{B_p^*} \leftarrow \boldsymbol{B_p}$
    Si $f(\boldsymbol{A_p}, \boldsymbol{B_p}) < f(\boldsymbol{X}\boldsymbol{B}^*, \boldsymbol{B}^*)$: $\boldsymbol{B}^* \leftarrow \boldsymbol{B_p}$
  Fin del "Para cada p"
  Actualizar $Inercia$
Fin del "Para cada i"
$\boldsymbol{B} \leftarrow \boldsymbol{B}^*$
$\boldsymbol{A} \leftarrow \boldsymbol{X}\boldsymbol{B}$
$SCE \leftarrow f(\boldsymbol{A}, \boldsymbol{B})$
Datos de salida: $\boldsymbol{A}, \boldsymbol{B}, SCE$

La primera tarea que efectúa el **Algoritmo 3.3.1** es inicializar de forma aleatoria las partículas dentro del espacio de las soluciones factibles. Cada partícula p tiene asociada una matriz de cargas $\boldsymbol{B_p}$ (solución factible, es decir, una matriz ortogonal disjunta). El algoritmo opera con un total de $nPar$ número de partículas. Luego, por cada iteración, cada partícula da "un paso", es decir, se mueve de la solución factible en la que se encuentra, a otra solución factible (La matriz de cargas original sufre una pequeña perturbación y se obtiene una nueva matriz de cargas ortogonal disjunta). Cada vez que se "mueve" una partícula a una nueva posición, se analiza si la partícula pudo encontrar una "mejor" solución, es decir, una solución con un menor valor de SCE (suma de los cuadrados de los errores) y, de ser así, se actualiza la mejor solución encontrada por esa partícula hasta el momento representada en la matriz $\boldsymbol{B}_{\boldsymbol{p}}^{*}$. Si además, la nueva solución encontrada por la partícula es la "mejor" solución encontrada por el cúmulo de partículas, se actualiza la matriz $\boldsymbol{B}^{*}$. Por cada iteración todas las partículas dan un paso, luego de ello se actualiza la variable $Inercia$. Esta variable inicia con el valor $maxIner$ y disminuye linealmente hasta alcanzar el valor $minIner$ en la última iteración. Los valores de los parámetros de entrada $wCognition$ y $wSocial$ se escogen como en todo algoritmo PSO (ver Poli et al. 2007 y García-Gonzalo and Fernández-Martínez 2012).

En este trabajo de tesis se utiliza el algoritmo CBPSO-DC con el objetivo de extenderlo para el cálculo de componentes ortogonales disjuntos en tablas de 3 vías. Para cumplir con tal propósito, se han diseñado e implementado algoritmos que se proponen en el **Capítulo 4** y que permiten obtener matrices de cargas ortogonales disjuntas en cualquiera de los modos de una tabla de 3 vías $\underline{\mathbf{X}}$.

El principal beneficio del algoritmo CBPSO-DC es que permite calcular una matriz de cargas $\boldsymbol{B}$ ortogonal disjunta, lo que facilita su interpretación. Para ilustrarlo, suponga que tenemos la matriz de datos $\boldsymbol{X}$ que se muestra en la **Tabla 3.3.1**. A esta matriz de datos le vamos a calcular la matriz de cargas, de acuerdo a los dos modelos de optimización vistos anteriormente.

Se trata de una matriz de datos con 15 individuos y 8 variables originales. La tabla almacena la calificación obtenida en una escala del 1 al 10, por el individuo $i \in \{1, \dots, 15\}$ (se trata de un estudiante) en la materia $j \in \{1, \dots, 8\}$. Al aplicar, por ejemplo, una reducción dimensional a $Q = 3$ componentes, esperamos que el modelo multivariante de componentes agrupe las materias (variables originales) según su tipo, es decir, si son materias de ciencias, materias de letras o materias relacionadas con alguna actividad física como son la gimnasia y el deporte. Las materias de izquierda a derecha son: gramática, matemática, física, inglés, literatura, historia, química y gimnasia. En este ejemplo ilustrativo se espera que las materias se distribuyan en 3 grupos. En un primer grupo las materias de ciencias como matemática, física y química. En un segundo grupo las materias de letras como gramática, inglés, literatura e historia. Finalmente, en el tercer grupo debe estar únicamente la materia de gimnasia.

Tabla 3.3.1: Matriz de DATOS con calificaciones de 15 alumnos en 8 materias

	GRAM.	MAT.	FÍSICA	INGLÉS	LITERAT.	HIST.	QUÍMICA	GIMNASIA
S1	5	5	5	5	5	5	5	5
S2	7	4	3	8	4	7	3	8
S3	5	8	7	6	5	6	7	5
S4	7	2	4	8	7	7	3	6
S5	8	9	10	8	8	7	9	4
S6	4	9	8	4	3	4	7	5
S7	6	4	4	6	5	5	3	7
S8	4	7	8	3	3	2	8	3
S9	5	5	4	5	6	5	5	1
S10	7	4	5	7	8	8	4	6
S11	7	8	8	7	7	6	7	9
S12	4	3	3	4	3	2	1	4
S13	7	4	4	7	8	7	4	5
S14	3	5	5	2	3	3	5	7
S15	5	6	6	5	5	5	6	6

En la **Tabla 3.3.2** se muestra la matriz de cargas $\boldsymbol{B}$ ortogonal que se obtiene al realizar un análisis tradicional de componentes usando el software R. Antes del cálculo se le aplicó un centrado clásico a la matriz de datos, es decir, a cada dato se le restó la media de cada columna (centrado por variable).

Tabla 3.3.2: Matriz de cargas $\boldsymbol{B}$ ortogonal con PCA

	COMP1	COMP2	COMP3
GRAMÁTICA	-0.10537175	**0.40711401**	-0.03314967
MATEMÁTICA	**0.56246248**	0.11900302	0.13283045
FÍSICA	**0.53363953**	0.18912541	0.06515149
INGLÉS	-0.16878408	**0.49000490**	-0.01967903
LITERATURA	-0.08821408	**0.48702046**	-0.30534414
HISTORIA	-0.14562687	**0.49715281**	-0.02560583
QUÍMICA	**0.55931708**	0.17184302	-0.04106607
GIMNASIA	-0.13209482	0.17419349	**0.93864161**

En la matriz de cargas $\boldsymbol{B}$ se han resaltado en negrita las entradas más altas en valor absoluto (mayor a 0.4). Al observar la matriz podemos apreciar que el primer componente etiquetado como COMP1 es la variable latente "materia de ciencias" ya que las entradas más altas en valor absoluto corresponden a matemática, física, química. El componente etiquetado como COMP2 es la variable latente "materia de letras" ya que las entradas más altas en valor absoluto corresponden a gramática, inglés, literatura, historia. El componente etiquetado como COMP3 es la variable latente "materia de actividad física" ya que la entrada más alta en valor absoluto corresponde a la materia gimnasia.

Por otra parte, al realizar el cálculo de componentes ortogonales disjuntos con el algoritmo CBPSO-DC, se obtiene una matriz de cargas $\boldsymbol{B}$ ortogonal disjunta (ver definición en la **Sección 3.1**) que presenta una estructura simple. En la **Tabla 3.3.3** se muestra la matriz de cargas $\boldsymbol{B}$ que se obtiene con el algoritmo CBPSO-DC. Igual que en el experimento computacional anterior, se aplicó un centrado por variable a la matriz de datos.

Tabla 3.3.3: Matriz de cargas B ortogonal disjunta con DPCA

	COMP1	COMP2	COMP3
GRAMÁTICA	0	**0.42683051**	0
MATEMÁTICA	**0.58009486**	0	0
FÍSICA	**0.56675657**	0	0
INGLÉS	0	**0.52692660**	0
LITERATURA	0	**0.51315878**	0
HISTORIA	0	**0,52614840**	0
QUÍMICA	**0.58504440**	0	0
GIMNASIA	0	0	**1**

En esta matriz de cargas $\boldsymbol{B}$ se han resaltado en negrita los valores no nulos. La estructura simple de esta matriz de cargas es el principal beneficio del algoritmo CBPSO-DC, ya que la interpretación se facilita de manera importante. Pero este beneficio tiene un costo: pérdida de fit en el modelo. En el PCA, la función objetivo tomó como valor mínimo 0.057 y el modelo atrapó un 94.27% de la variabilidad de los datos. Por otro lado, en el DPCA, la función objetivo tomó como valor mínimo 0.067 y el modelo atrapó un 93.30% de la variabilidad de los datos. Debido a experimentos computacionales que se han llevado a cabo en esta tesis, se afirma que estas pérdidas son mayores con matrices de datos más grandes.

El algoritmo CBPSO-DC usa los siguientes 8 parámetros de entrada para poder realizar el cálculo de los Q componentes ortogonales disjuntos en la matriz de datos $\boldsymbol{X}$:

- $\boldsymbol{X}$: Matriz de datos con I individuos y J variables.
- Q: Número de componentes en la reducción dimensional.
- $PSOMaxIter$: Número de iteraciones.
- $nPar$: Número de partículas utilizadas en la búsqueda.
- $minIner$: Inercia mínima.
- $maxIner$: Inercia máxima.
- $wCognition$: Peso cognitivo.
- $wSocial$: Peso social.

En el **Capítulo 5**, para indicar que cierto tipo de instrucción debe ejecutarse en los algoritmos heurísticos que se proponen, se usa la siguiente notación:

$$\boldsymbol{B} \leftarrow CBPSO(\boldsymbol{X}, Q, PSOMaxIter, nPar, minIner, maxIner, wCognition, wSocial)$$

El significado de esta instrucción es que el algoritmo CBPSO-DC, usando los valores fijados para los parámetros de entrada, calcula Q componentes ortogonales disjuntos en la matriz de datos $\boldsymbol{X}$ y entrega como resultado la matriz de cargas ortogonal disjunta $\boldsymbol{B}$. Para más detalles relacionados al algoritmo CBPSO-DC ver Ramirez-Figueroa et al. 2021.

Para más detalles sobre el análisis de componentes principales en tablas de 2 vías, la descomposición en valores singulares y otras técnicas multivariantes para matrices de datos, como por ejemplo el escalamiento multidimensional, consulte Pearson 1901, Ten Berge 1993, Cuadras 2014, Borg and Groenen 2005.

El algoritmo CBPSO-DC se diseñó con la intención de mejorar (en tiempos de procesamiento y en la calidad de las soluciones obtenidas) un algoritmo heurístico, propuesto por Vichi and Saporta 2009, que calcula componentes ortogonales disjuntos en tablas de 2 vías. El **Algoritmo 3.3.2** presenta una implementación del algoritmo que permite realizar un DPCA.

Algoritmo 3.3.2: Algoritmo de Vichi and Saporta 2009 (para realizar un DPCA)

Datos de Entrada: $\boldsymbol{X}, Q, Tol$
Inicialización aleatoria de una matriz ortogonal disjunta (binaria) de cargas $\boldsymbol{V}$
$k \leftarrow 0$
Iterar:
    $k \leftarrow k + 1$
    Para cada i desde 1 hasta I:
        $MejorSCE \leftarrow +\infty$
        Para cada q desde 1 hasta Q:
            Perturbar $\boldsymbol{V}$
            $\boldsymbol{B} \leftarrow AplicarSVD(\boldsymbol{X}, \boldsymbol{V})$
            $\boldsymbol{A} \leftarrow \boldsymbol{XB}$
            $fSCE \leftarrow f(\boldsymbol{A}, \boldsymbol{B})$
            Si $(fSCE < MejorSCE)$: $MejorSCE \leftarrow fSCE, Pos \leftarrow q$
        Fin del "Para cada q"
        Asignar la variable original i al componente Pos
    Fin del "Para cada i"
    $SCE_k \leftarrow f(\boldsymbol{A}, \boldsymbol{B})$
Hasta que $|SCE_k - SCE_{k-1}| < Tol$
Datos de salida: $\boldsymbol{A}, \boldsymbol{B}, SCE_k$

El **Algoritmo 3.3.2** inicia con una matriz binaria aleatoria $\boldsymbol{V}$ que además es ortogonal disjunta. Luego, por cada iteración k, se debe realizar un barrido fijando cada variable original i (donde I es el número de variables originales) y asignándola a cada componente q desde el primero hasta el último (donde Q representa el número total de componentes). La idea es que, por cada variable original, se pueda detectar el componente que la representa mejor (esto involucra evaluar en la función objetivo f). Este procedimiento se repite para cada una de las variables originales.

La perturbación de la matriz binaria $\boldsymbol{V}$ consiste entonces en que una variable original deje de ser representada por el componente actual, para que sea representada por el siguiente componente. Luego, mediante descomposiciones SVD, se calcula la nueva matriz de cargas ortogonal disjunta $\boldsymbol{B}$. Posterior a eso se calcula la matriz de "scores" $\boldsymbol{A}$ y, finalmente, el valor de la función objetivo que se almacena en la variable $fSCE$. Cuando se ha llegado hasta la última variable original se repite el proceso hasta que la diferencia en la SCE de dos iteraciones consecutivas sea inferior a una tolerancia Tol que es ingresada por el usuario. La instrucción $MejorSCE \leftarrow +\infty$ significa que se le asigna el valor de punto flotante más alto posible a la variable $MejorSCE$, según el lenguaje de programación utilizado.

La **Tabla 3.3.1** presenta algunas diferencias entre el **Algoritmo 3.3.1** y el **Algoritmo 3.3.2**. Estas diferencias se han enunciado en base a experimentos computacionales realizados y también en base al diseño de ambos algoritmos. Una discusión más detallada la podemos ver en Ramirez-Figueroa et al. 2021. ¿Qué tienen en común estos algoritmos? Ambos algoritmos son heurísticas que permiten calcular componentes ortogonales disjuntos en tablas de 2 vías. El algoritmo CBPSO-DC presenta un mejor diseño, y a pesar de que su implementación es más complicada, es recomendable su uso en comparación con el algoritmo clásico, porque encuentra mejores soluciones.

Tabla 3.3.1: Análisis comparativo de los Algoritmos 3.3.1 y 3.3.2

Algoritmo CBPSO-DC	**Algoritmo de Vichi and Saporta 2009**
La implementación en un lenguaje de programación es más difícil.	La implementación en un lenguaje de programación es más sencilla.
Ofrece mejores soluciones y mejores tiempos de procesamiento en matrices de gran tamaño (número de individuos y número de variables por encima de 100).	Ofrece mejores soluciones y mejores tiempos de procesamiento en matrices pequeñas (número de individuos y número de variables que no exceda de 100).
Durante su ejecución tiene muchas partículas buscando la mejor solución (por ende trabaja con muchas matrices de cargas) y esto aumenta la probabilidad de encontrar soluciones de mejor calidad.	Durante su ejecución sólo trabaja con una matriz de cargas y esto disminuye la probabilidad de encontrar soluciones de mejor calidad.
Realiza una búsqueda inteligente basada en la optimización por enjambre de partículas (PSO).	Realiza una búsqueda casi exhaustiva (aplica un criterio de fuerza bruta).
Se recomienda ejecutarlo hasta 3 veces para encontrar la solución de un problema.	Se recomienda ejecutarlo al menos 5 veces para encontrar la solución de un problema.
Tiene muchos parámetros de entrada y su configuración es más difícil.	Tiene menos parámetros de entrada y su configuración es más sencilla.
Es menos propenso a quedarse atrapado en óptimos locales.	Es más propenso a quedarse atrapado en óptimos locales.

3.4 LA METODOLOGÍA DISJUNTA EN TABLAS DE 3 VÍAS

En esta tesis, además de proponer algoritmos heurísticos (ver **Capítulo 4**) para el cálculo de componentes ortogonales disjuntos en los modelos multivariantes para el análisis de tablas de 3 vías, también proponemos, en forma algorítmica, bajo qué condiciones utilizarlos. En otras palabras, en esta tesis se proponen no sólo los algoritmos que realizan el cálculo sino también una sugerencia de su utilización. A esto se le ha denominado la "metodología disjunta" en tablas de 3 vías.

3.4.1 PROPUESTA PARA EL MODELO PARAFAC

En el **Algoritmo 3.4.1** se muestra paso a paso la recomendación para el cálculo de matrices de cargas ortogonales disjuntas en el modelo PARAFAC que consiste en un total de 10 pasos:

Algoritmo 3.4.1: Metodología disjunta para el modelo PARAFAC

---

1. Recolectar los datos (observaciones) y almacenarlos en una tabla de 3 vías $\underline{\mathbf{X}}$ de tamaño $I \times J \times K$ con I individuos, J variables y K situaciones o tiempos.
2. Preprocesar $\underline{\mathbf{X}}$ de acuerdo al criterio del analista de los datos (ver **Sección 2.1**)
3. Si un modelo PARAFAC con R componentes (ver **Sección 2.2**) es escogido para analizar los datos, continuar al **Paso 4** (ver **Sección 2.7** y **Sección 2.8**). Si no, use el **Algoritmo 3.4.2**
4. Calcular el fit del modelo PARAFAC sin restricciones (ver **Modelo de Optimización 2.6.1**) usando el **Algoritmo 2.2** por ejemplo. Si se sospecha la presencia del problema de "degeneración" (ver **Sección 2.2.2**) continuar al **Paso 5**, caso contrario continuar al **Paso 8**.
5. Imponer la restricción ortogonal en al menos una de las matrices de cargas, y calcular las matrices de cargas $\boldsymbol{A}$, $\boldsymbol{B}$ y $\boldsymbol{C}$. Aplicar, si se considera necesario, la técnica de escalado (ver **Sección 2.2.1**) para obtener matrices de cargas con estructura simple.
6. Imponer la restricción ortogonal disjunta (ver **Modelo de Optimización 3.2.1.1** hasta **Modelo de Optimización 3.2.1.7**) en al menos una de las matrices de cargas, y calcular las matrices de cargas $\boldsymbol{A}$, $\boldsymbol{B}$ y $\boldsymbol{C}$. Es decir, calcular componentes ortogonales disjuntos por lo menos en una de las matrices de cargas (aplicar la técnica disjunta).
7. Comparar los resultados obtenidos en **Paso 5** y **Paso 6**. Ir al **Paso 10**.
8. Calcular las matrices de cargas $\boldsymbol{A}$, $\boldsymbol{B}$ y $\boldsymbol{C}$ usando el **Algoritmo 2.2** por ejemplo. Aplicar la técnica de escalado, si se considera necesario, para mejorar la interpretación (se mantiene el fit del modelo).
9. Ir al **Paso 6**. Comparar con los resultados obtenidos en el **Paso 8**.
10. Análisis de resultados y conclusiones.

Calcular componentes ortogonales disjuntos simultáneamente en todas las matrices de cargas del modelo PARAFAC queda a criterio del investigador o analista de los datos. Sin embargo, no se recomienda hacerlo debido a la pérdida importante de fit que se ha observado en los experimentos computaciones (ver **Capítulo 5**) llevados a cabo en esta tesis. Se sugiere en el modelo PARAFAC, calcular una, o a lo mucho, dos matrices de cargas ortogonales disjuntas.

Otro asunto importante es que no se necesita la presencia del problema de "degeneración" para realizar el cálculo de componentes ortogonales disjuntos. Se puede directamente calcular sólo una matriz, o dos matrices de cargas ortogonales disjuntas, en un modelo PARAFAC que no tenga el problema de degeneración con el propósito de ganar en interpretación, claro está, con la correspondiente pérdida de fit del modelo. En otras palabras, el algoritmo para el cálculo de componentes ortogonales disjuntos que proponemos para el modelo PARAFAC (ver **Capítulo 4**) se puede utilizar para resolver el problema de degeneración si se presenta, pero si no se presenta, también se puede emplear de forma directa con la intención de obtener una estructura simple que permita facilidad de interpretación.

En el **Paso 6** del **Algoritmo 3.4.1** se propone el cálculo de componentes ortogonales disjuntos como una técnica complementaria, es decir, si un modelo PARAFAC con R componentes se ajusta bien a los datos pero las matrices de cargas, antes o después de un escalado, no son fácilmente interpretables, se recomienda calcular matrices de cargas ortogonales disjuntas. Sin embargo, si antes o después de un escalado, las matrices de cargas son interpretables, se recomienda también el cálculo de matrices ortogonales disjuntas con la intención de hacer un análisis comparativo y verificar que se obtiene la misma interpretación. A continuación se muestra un diagrama de flujo del **Algoritmo 3.4.1**:

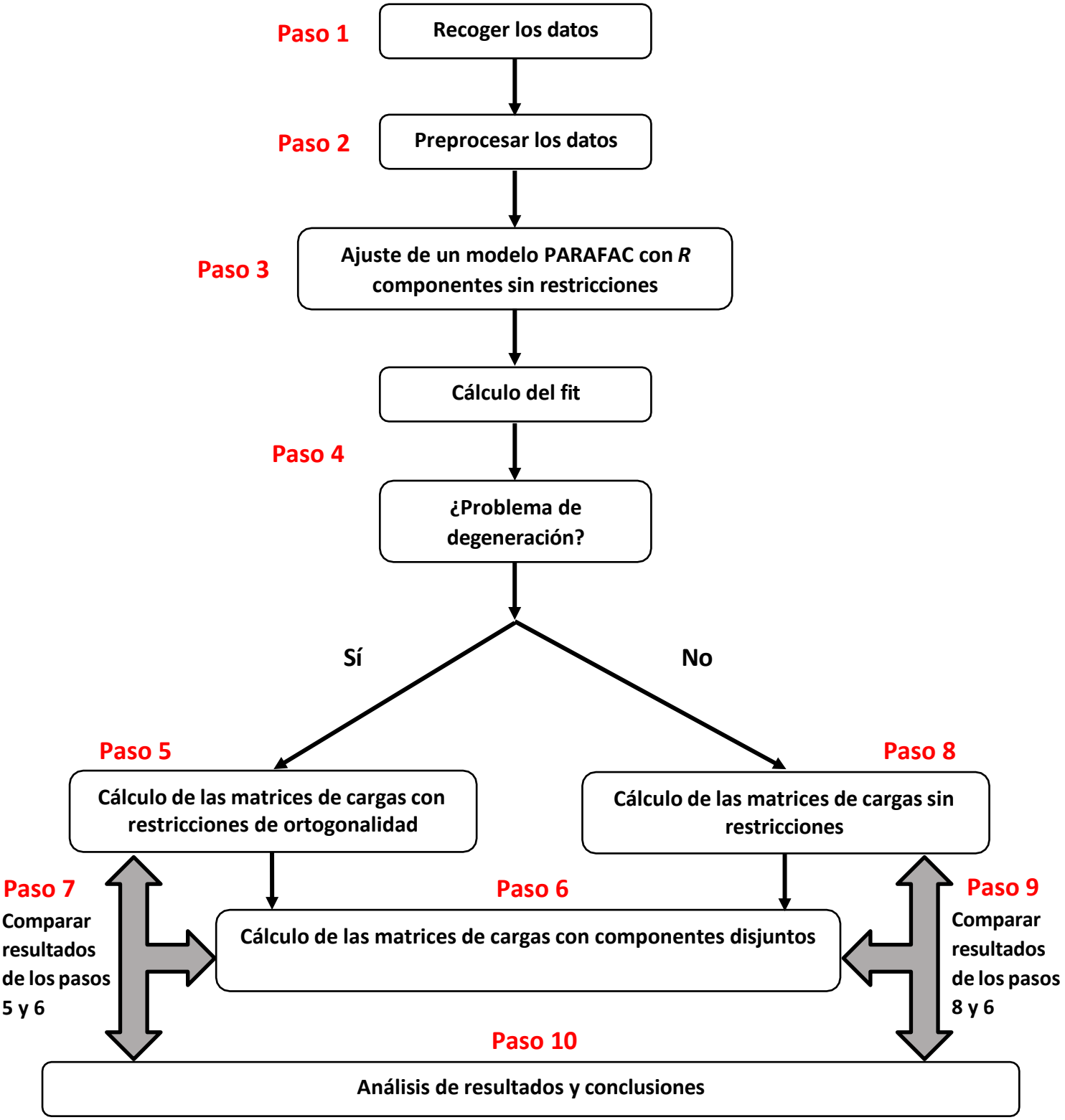


Figura 3.4.1: Diagrama de flujo del Algoritmo 3.4.1

3.4.2 PROPUESTA PARA LOS MODELOS TUCKER

En el **Algoritmo 3.4.2** se muestra paso a paso la recomendación para el cálculo de matrices de cargas ortogonales disjuntas en los modelos Tucker que consiste en un total de 9 pasos:

Algoritmo 3.4.2: Metodología disjunta para los modelos Tucker

---

1. Recolectar los datos (observaciones) y almacenarlos en una tabla de 3 vías $\underline{\mathbf{X}}$ de tamaño $I \times J \times K$ con I individuos, J variables y K situaciones o tiempos.
2. Preprocesar $\underline{\mathbf{X}}$ de acuerdo al criterio del analista de los datos (ver **Sección 2.2**)
3. Determinar el modelo Tucker que es más conveniente para $\underline{\mathbf{X}}$ o escoger el modelo Tucker según las preferencias del investigador (ver **Sección 2.3**, **Sección 2.4** y **Sección 2.5**). Este paso implica también determinar el número de componentes por cada modo que se desea reducir (ver **Sección 2.7** y **Sección 2.8**). Por ejemplo, si se escoge el modelo Tucker3, determinar los valores de P, Q y R.
4. Calcular el fit del modelo seleccionado en el **Paso 3** y calcular las matrices de cargas de dicho modelo con el correspondiente algoritmo ALS (ver **Sección 2.3**, **Sección 2.4** y **Sección 2.5**). Por ejemplo, si se escoge el modelo Tucker3, emplear el **Algoritmo 2.3** para realizar el cálculo de las matrices de cargas $\boldsymbol{A}$, $\boldsymbol{B}$ y $\boldsymbol{C}$. Si las matrices de cargas del modelo tienen una estructura simple (son fáciles de interpretar) ir al **Paso 8**, caso contrario continúe al **Paso 5**.
5. Aplicar técnicas de escalado y técnicas de rotación (ver **Sección 2.3.1**) para mantener el fit. Si las matrices de cargas tienen una estructura simple ir al **Paso 8.**
6. Aplicar técnicas "sparse", se pierde algo de fit pero se puede alcanzar una estructura simple. Si las matrices de cargas tienen una estructura simple ir al **Paso 8**.
7. Calcular componentes ortogonales disjuntos en aquellas matrices de cargas que considere el investigador (es decir, aplicar la técnica disjunta) según el modelo. Ir al **Paso 8**.
8. Análisis de resultados y conclusiones.

Más de una técnica puede ser utilizada con el objetivo de alcanzar una estructura simple en las matrices de cargas. Lo primero que se propone en esta tesis (**Paso 5** del **Algoritmo 3.4.2**) es aplicar escalado y/o rotación, ya que al emplear cualquiera de estas dos técnicas se mantiene el fit del modelo. Este debe ser el primer intento en el análisis, tratar de no perder fit en el modelo. Pero no hay garantía con estas técnicas de que se obtengan matrices de cargas fáciles de interpretar.

Si se fracasa en el primer intento, un segundo intento es aplicar una técnica "sparse" (ver por ejemplo Yokota and Cichocki 2014 y Perros et al. 2015). Con cualquier técnica "sparse" habrá pérdida de fit en el modelo (**Paso 6** del **Algoritmo 3.4.2**), pero se puede alcanzar estructura simple en las matrices de cargas. Importante es mencionar que con esta técnica tampoco hay garantía de obtener matrices de cargas fáciles de interpretar.

Si en los dos intentos anteriores no se obtienen matrices de cargas con estructura simple, en el **Paso 7** del **Algoritmo 3.4.2** se propone el cálculo de componentes ortogonales disjuntos en las matrices de cargas. Si se está trabajando con un modelo Tucker3 que usa tres matrices de cargas, se pueden calcular desde una hasta tres matrices de cargas ortogonales disjuntas. Sin embargo, igual que con el modelo PARAFAC, no se recomienda calcular componentes ortogonales disjuntos en las tres matrices de cargas simultáneamente porque la pérdida de fit es importante (esta decisión queda a criterio del investigador). Es lo que se sugiere en esta tesis por lo que se ha podido observar en los experimentos computacionales realizados. Para el modelo Tucker3 se recomienda calcular componentes ortogonales disjuntos en una o máximo dos matrices de cargas. Si se está trabajando con un modelo Tucker2 se pueden calcular componentes ortogonales disjuntos en una matriz de cargas o en ambas y, si se está trabajando con un modelo Tucker1 entonces se deben calcular los componentes ortogonales disjuntos en la única matriz de cargas del modelo.

Todas estas técnicas no tienen que ser excluyentes, es decir, no porque se emplee una de ellas se van a descartar las demás. En el estudio de una tabla de 3 vías el análisis debe ser complementario, las distintas técnicas pueden brindar diferentes perspectivas por lo que únicamente aplicando todas se puede tener una visión completa de los datos. Se pueden aplicar todas las técnicas (algunas con pérdida de fit y otras no), incluida la técnica disjunta, y realizar un análisis comparativo en el **Paso 8** del **Algoritmo 3.4.2** como parte del análisis de los resultados y las conclusiones. A continuación se muestra un diagrama de flujo del **Algoritmo 3.4.2**:

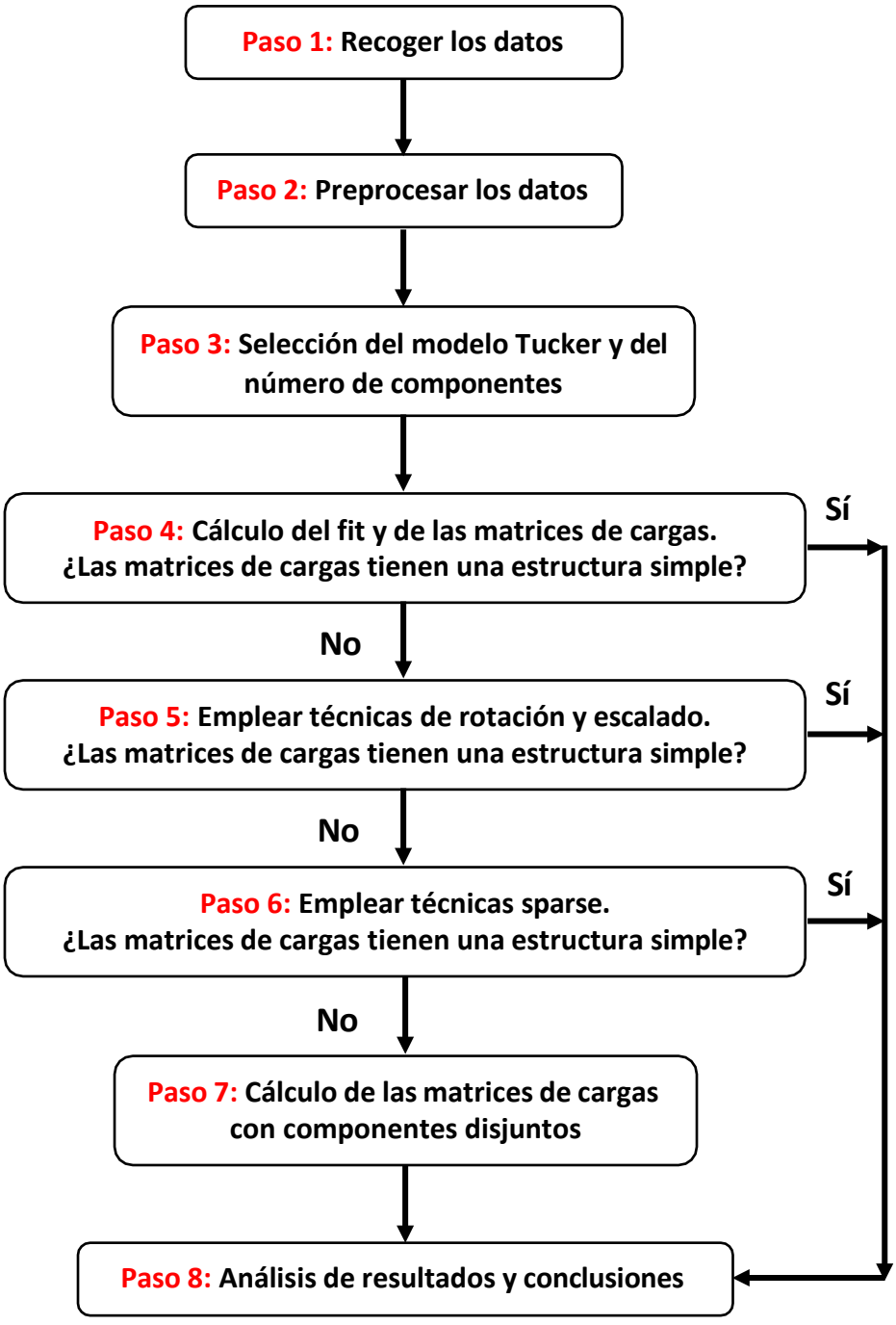


Figura 3.4.2: Diagrama de flujo del Algoritmo 3.4.2

CAPÍTULO CUATRO: "ALGORITMOS HEURÍSTICOS PARA EL CÁLCULO DE COMPONENTES ORTOGONALES DISJUNTOS EN TABLAS DE 3 VÍAS"

En este capítulo se hace una breve introducción a los algoritmos heurísticos y su importancia en la resolución de problemas de optimización con una alta complejidad computacional. Luego se proponen algoritmos heurísticos que realizan el cálculo de componentes ortogonales disjuntos en tablas de 3 vías para cada uno de los 8 modelos multivariantes de componentes que se discuten en el **Capítulo 2**. Estos algoritmos propuestos resuelven los modelos de optimización presentados en la **Sección 3.2** de este documento de tesis. En el **Capítulo 5** se presentan experimentos computacionales que se han llevado a cabo utilizando estos algoritmos heurísticos.

4.1 INTRODUCCIÓN A LOS ALGORITMOS HEURÍSTICOS

El significado del término “heurística” es hallar, descubrir. Por tanto, un algoritmo heurístico es aquel algoritmo que hace su mejor esfuerzo por encontrar al menos un elemento (que cumple con ciertas características) dentro de un conjunto o espacio de búsqueda. No hay garantía de que el algoritmo tenga éxito (pero podría tenerlo), sin embargo se lo diseña de manera tal que haga todo lo posible por encontrar dicho elemento. Si fracasa en el intento, el algoritmo entrega como resultado, dentro de aquellos elementos con los que se topó mientras realizaba la búsqueda, aquel elemento que “más se parece” al que busca. Se debe tener en cuenta que el conjunto donde el algoritmo heurístico busca al elemento puede ser finito pero de una alta cardinalidad, infinito contable o incluso infinito no enumerable. Por tanto la tarea de búsqueda del elemento en el conjunto no es trivial, es decir, no es en general una tarea sencilla. En muchos problemas de la matemática se utilizan algoritmos heurísticos. El proceso de búsqueda de un algoritmo heurístico está basado en un conjunto de reglas claramente definidas, en otras palabras, se trata de un procedimiento organizado, sistemático.

Los algoritmos heurísticos se usan por ejemplo para resolver problemas de optimización (ver **Sección 2.6** y **Sección 3.2**). Cuando el objetivo es resolver un problema de optimización, un algoritmo exacto es aquel algoritmo que, bajo ciertas condiciones y escenarios, garantiza que puede encontrar el óptimo del problema. Es deseable que un algoritmo resuelva un problema de optimización en un tiempo razonable. Para muchos problemas de optimización no existe un algoritmo exacto que pueda resolver el problema en un tiempo razonable. Si tal es el caso, los algoritmos heurísticos son una alternativa práctica y viable, ya que si no pueden encontrar el óptimo del problema de optimización en un tiempo razonable, al menos pueden entregar una “buena solución”. Los recursos computacionales (tiempo de procesador y memoria principal) son un factor importante que se debe considerar al utilizar un algoritmo para resolver un problema de optimización en un tiempo prudencial.

En resumen, las heurísticas, contrarias a los métodos exactos, son métodos inexactos para resolver problemas de optimización, ya que no ofrecen ningún tipo de garantía para encontrar el óptimo del problema, pero son capaces de ofrecer buenas soluciones y en un tiempo razonable. Una buena solución, a pesar de no ser la óptima, puede ser suficiente en la práctica para el estudio y análisis de un problema de optimización. A pesar de no haber garantía de encontrar el óptimo del problema, eso no quiere decir que la heurística no pueda encontrarlo. Lo dicho anteriormente es la principal razón por la que los algoritmos heurísticos son populares para resolver muchos problemas difíciles (complejidad computacional) de optimización.

A continuación en la **Tabla 4.1.1** se presentan algunas de las ventajas y desventajas más importantes al emplear algoritmos heurísticos en la resolución de problemas difíciles de optimización:

Tabla 4.1.1: Ventajas y desventajas de los algoritmos heurísticos

VENTAJAS	DESVENTAJAS
Normalmente pueden encontrar buenas soluciones (en algunos casos el óptimo).	En ocasiones pueden no encontrar buenas soluciones.
Ejecutan en un tiempo razonable.	Pueden quedar atrapados en óptimos locales (Pero usan estrategias para intentar evitarlos).
Consumen en general menos recursos computacionales que cualquier método exacto (si existe) que resuelve el mismo problema.	Algunos algoritmos heurísticos no son fáciles de implementar y requieren de un gran esfuerzo de programación.
Suelen ser más sencillos de diseñar que los algoritmos exactos.	No existe garantía de encontrar el óptimo del problema de optimización.
Las reglas usadas por los algoritmos heurísticos son más fáciles de entender que la justificación matemática de los algoritmos exactos.	En ocasiones no se conoce qué tan cerca se encuentra la solución encontrada de la solución óptima.

Para el cálculo de componentes ortogonales disjuntos en matrices de datos, no existe un método exacto que pueda resolver el correspondiente problema de optimización en un tiempo razonable (ver **Sección 3.3**). En Vichi and Saporta 2009 se propone un algoritmo heurístico para un DPCA en matrices de datos y en Macedo and Freitas 2015 una implementación de dicho algoritmo. Ferrara et al. 2016 propone algunos métodos para el cálculo de componentes ortogonales disjuntos en tablas de 2 vías.

El objetivo principal de la tesis es diseñar e implementar 4 algoritmos heurísticos para el cálculo de componentes ortogonales disjuntos en tablas de 3 vías. Hasta la fecha, no existe en la literatura de la Estadística Multivariante, ningún algoritmo que realice este cálculo.

4.2 EL ALGORITMO CBPSO-ParafacALS

A continuación se presenta un algoritmo heurístico denominado CBPSO-ParafacALS que permite una reducción dimensional en los tres modos con el mismo número R de componentes según el modelo PARAFAC (ver **Sección 2.2**), y permite además calcular desde una hasta tres matrices de cargas ortogonales disjuntas.

Para resolver el **Modelo de Optimización 3.2.1.1** se propone el **Algoritmo 4.2.1** que se muestra y que calcula la matriz de cargas ortogonal disjunta $\boldsymbol{A}$:

Algoritmo 4.2.1: CBPSO-ParafacALS donde $\boldsymbol{A}$ es ortogonal disjunta

---

Datos de Entrada:

$\underline{\mathbf{X}}, R, ALSMaxIter, PSOMaxIter, nPar, minIner, maxIner, wCognition, wSocial$

A partir de $\underline{\mathbf{X}}$ obtener las supermatrices: $\boldsymbol{X_A}, \boldsymbol{X_B}, \boldsymbol{X_C}$

Etapa 1.- Calcular la matriz ortogonal disjunta $\boldsymbol{A}$:

$\boldsymbol{A} \leftarrow CBPSO(\boldsymbol{X_A}^T, R, PSOMaxIter, nPar, minIner, maxIner, wCognition, wSocial)$

Etapa 2.- Calcular las matrices ortogonales $\boldsymbol{B}$ y $\boldsymbol{C}$:

Inicializar la matriz $\boldsymbol{C}$

Repetir

$\boldsymbol{B} \leftarrow \boldsymbol{X_B}[(\boldsymbol{C} \odot \boldsymbol{A})^T]^+$

$\boldsymbol{C} \leftarrow \boldsymbol{X_C}[(\boldsymbol{B} \odot \boldsymbol{A})^T]^+$

Hasta que se cumpla $CriterioParada4.2.1$

Calcular el fit del Modelo

Datos de salida: $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$, fit

---

El criterio de parada etiquetado como $CriterioParada4.2.1$ se cumple cuando se alcanza el número máximo de iteraciones (almacenado en el parámetro de entrada $ALSMaxIter$) o cuando las matrices $\boldsymbol{B}$ y $\boldsymbol{C}$ no difieren de forma significativa en dos iteraciones consecutivas. La inicialización de la matriz $\boldsymbol{C}$ es aleatoria. En la **Sección 2.2** se indica la expresión que permite calcular el fit del modelo. Ver la **Sección 3.3** para los datos de entrada y la instrucción de la **Etapa 1**.

Para resolver el **Modelo de Optimización 3.2.1.2** se propone el **Algoritmo 4.2.2** que se muestra y que calcula la matriz de cargas ortogonal disjunta $\boldsymbol{B}$:

Algoritmo 4.2.2: CBPSO-ParafacALS donde $\boldsymbol{B}$ es ortogonal disjunta

---

Datos de Entrada:

$\underline{\mathbf{X}}, R, ALSMaxIter, PSOMaxIter, nPar, minIner, maxIner, wCognition, wSocial$

A partir de $\underline{\mathbf{X}}$ obtener las supermatrices: $\boldsymbol{X_A}, \boldsymbol{X_B}, \boldsymbol{X_C}$

Etapa 1.- Calcular la matriz ortogonal disjunta $\boldsymbol{B}$:

$\boldsymbol{B} \leftarrow CBPSO(\boldsymbol{X_B}^T, R, PSOMaxIter, nPar, minIner, maxIner, wCognition, wSocial)$

Etapa 2.- Calcular las matrices ortogonales $\boldsymbol{A}$ y $\boldsymbol{C}$:

Inicializar la matriz $\boldsymbol{C}$

Repetir

$\boldsymbol{A} \leftarrow \boldsymbol{X_A}[(\boldsymbol{C} \odot \boldsymbol{B})^T]^+$

$\boldsymbol{C} \leftarrow \boldsymbol{X_C}[(\boldsymbol{B} \odot \boldsymbol{A})^T]^+$

Hasta que se cumpla $CriterioParada4.2.2$

Calcular el fit del Modelo

Datos de salida: $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$, fit

---

El criterio de parada etiquetado como $CriterioParada4.2.2$ se cumple cuando se alcanza el número máximo de iteraciones (almacenado en el parámetro de entrada $ALSMaxIter$) o cuando las matrices $\boldsymbol{A}$ y $\boldsymbol{C}$ no difieren de forma significativa en dos iteraciones consecutivas. La inicialización de la matriz $\boldsymbol{C}$ es aleatoria. En la **Sección 2.2** se indica la expresión que permite calcular el fit del modelo. Ver la **Sección 3.3** para los datos de entrada y la instrucción de la **Etapa 1**.

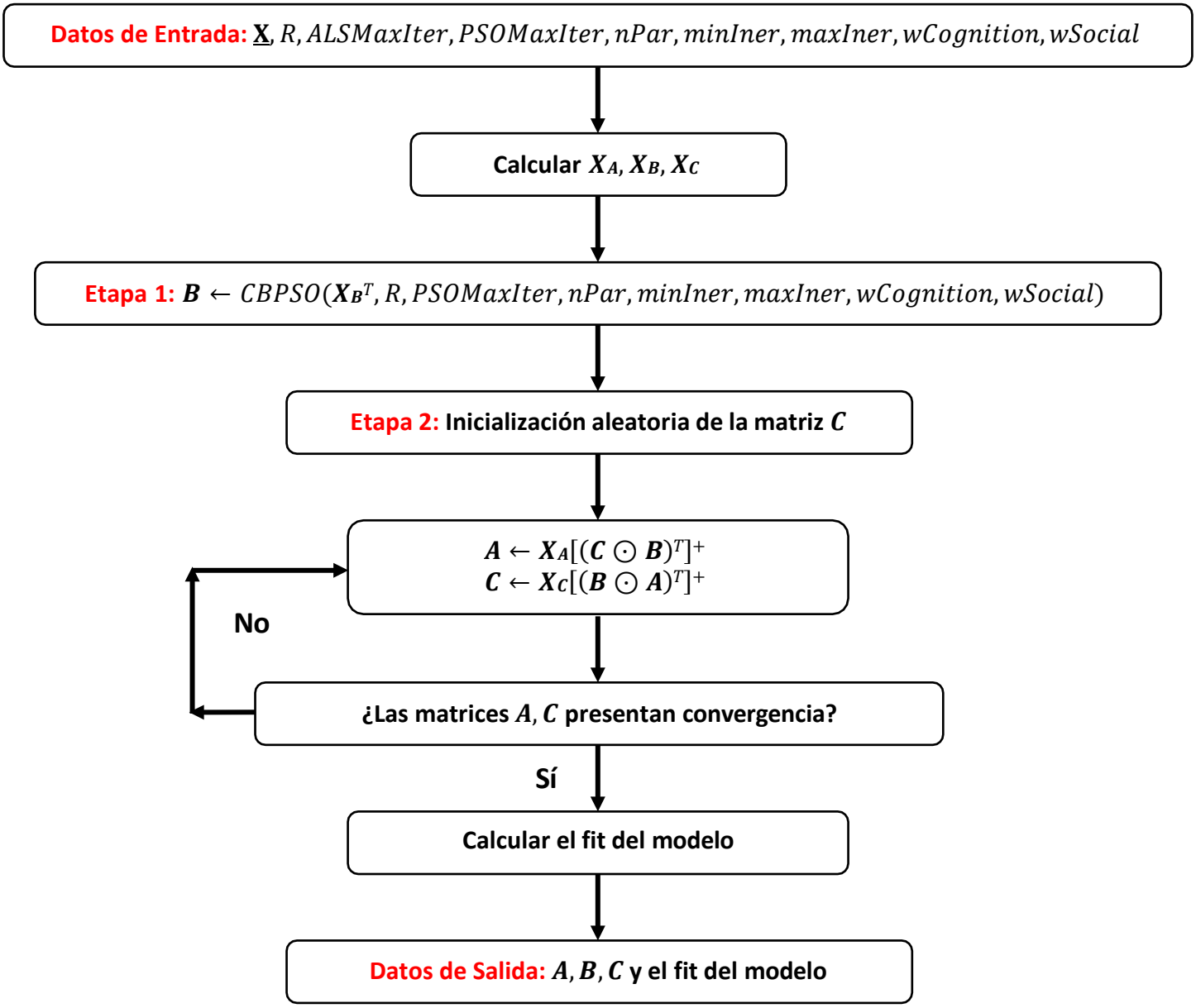


Figura 4.2.1: Diagrama de flujo del Algoritmo 4.2.2

Para resolver el **Modelo de Optimización 3.2.1.3** se propone el **Algoritmo 4.2.3** que se muestra y que calcula la matriz de cargas ortogonal disjunta $\boldsymbol{C}$:

Algoritmo 4.2.3: CBPSO-ParafacALS donde $\boldsymbol{C}$ es ortogonal disjunta

---

Datos de Entrada:

$\underline{\mathbf{X}}, R, ALSMaxIter, PSOMaxIter, nPar, minIner, maxIner, wCognition, wSocial$

A partir de $\underline{\mathbf{X}}$ obtener las supermatrices: $\boldsymbol{X_A}, \boldsymbol{X_B}, \boldsymbol{X_C}$

Etapa 1.- Calcular la matriz ortogonal disjunta $\boldsymbol{C}$:

$\boldsymbol{C} \leftarrow CBPSO(\boldsymbol{X_C}^T, R, PSOMaxIter, nPar, minIner, maxIner, wCognition, wSocial)$

Etapa 2.- Calcular las matrices ortogonales $\boldsymbol{A}$ y $\boldsymbol{B}$:

Inicializar la matriz $\boldsymbol{B}$

Repetir

$\boldsymbol{A} \leftarrow \boldsymbol{X_A}[(\boldsymbol{C} \odot \boldsymbol{B})^T]^+$

$\boldsymbol{B} \leftarrow \boldsymbol{X_B}[(\boldsymbol{C} \odot \boldsymbol{A})^T]^+$

Hasta que se cumpla $CriterioParada4.2.3$

Calcular el fit del Modelo

Datos de salida: $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$, fit

---

El criterio de parada etiquetado como $CriterioParada4.2.3$ se cumple cuando se alcanza el número máximo de iteraciones (almacenado en el parámetro de entrada $ALSMaxIter$) o cuando las matrices $\boldsymbol{A}$ y $\boldsymbol{B}$ no difieren de forma significativa en dos iteraciones consecutivas. La inicialización de la matriz $\boldsymbol{B}$ es aleatoria. En la **Sección 2.2** se indica la expresión que permite calcular el fit del modelo. Ver la **Sección 3.3** para los datos de entrada y la instrucción de la **Etapa 1**.

Para resolver el **Modelo de Optimización 3.2.1.4** se propone el **Algoritmo 4.2.4** que se muestra y que calcula las matrices de cargas ortogonales disjuntas $\boldsymbol{A}$ y $\boldsymbol{B}$:

Algoritmo 4.2.4: CBPSO-ParafacALS donde $\boldsymbol{A}$ y $\boldsymbol{B}$ son ortogonales disjuntas

---

Datos de Entrada:

$\underline{\mathbf{X}}, R, PSOMaxIter, nPar, minIner, maxIner, wCognition, wSocial$

A partir de $\underline{\mathbf{X}}$ obtener las supermatrices: $\boldsymbol{X_A}, \boldsymbol{X_B}, \boldsymbol{X_C}$

Etapa 1.- Calcular las matrices ortogonales disjuntas $\boldsymbol{A}$ y $\boldsymbol{B}$:

$\boldsymbol{A} \leftarrow CBPSO(\boldsymbol{X_A}^T, R, PSOMaxIter, nPar, minIner, maxIner, wCognition, wSocial)$

$\boldsymbol{B} \leftarrow CBPSO(\boldsymbol{X_B}^T, R, PSOMaxIter, nPar, minIner, maxIner, wCognition, wSocial)$

Etapa 2.- Calcular la matriz ortogonal $\boldsymbol{C}$:

$\boldsymbol{C} \leftarrow \boldsymbol{X_C}[(\boldsymbol{B} \odot \boldsymbol{A})^T]^+$

Calcular el fit del Modelo

Datos de salida: $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$, fit

---

En la **Sección 2.2** se indica la expresión que permite calcular el fit del modelo. Ver la **Sección 3.3** para los datos de entrada y las instrucciones de la **Etapa 1**.

Para resolver el **Modelo de Optimización 3.2.1.5** se propone el **Algoritmo 4.2.5** que se muestra y que calcula las matrices de cargas ortogonales disjuntas $\boldsymbol{A}$ y $\boldsymbol{C}$:

Algoritmo 4.2.5: CBPSO-ParafacALS donde $\boldsymbol{A}$ y $\boldsymbol{C}$ son ortogonales disjuntas

---

Datos de Entrada:

$\underline{\mathbf{X}}, R, PSOMaxIter, nPar, minIner, maxIner, wCognition, wSocial$

A partir de $\underline{\mathbf{X}}$ obtener las supermatrices: $\boldsymbol{X_A}, \boldsymbol{X_B}, \boldsymbol{X_C}$

Etapa 1.- Calcular las matrices ortogonales disjuntas $\boldsymbol{A}$ y $\boldsymbol{C}$:

$\boldsymbol{A} \leftarrow CBPSO(\boldsymbol{X_A}^T, R, PSOMaxIter, nPar, minIner, maxIner, wCognition, wSocial)$

$\boldsymbol{C} \leftarrow CBPSO(\boldsymbol{X_C}^T, R, PSOMaxIter, nPar, minIner, maxIner, wCognition, wSocial)$

Etapa 2.- Calcular la matriz ortogonal $\boldsymbol{B}$:

$\boldsymbol{B} \leftarrow \boldsymbol{X_B}[(\boldsymbol{C} \odot \boldsymbol{A})^T]^+$

Calcular el fit del Modelo

Datos de salida: $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$, fit

---

En la **Sección 2.2** se indica la expresión que permite calcular el fit del modelo. Ver la **Sección 3.3** para los datos de entrada y las instrucciones de la **Etapa 1**.

Para resolver el **Modelo de Optimización 3.2.1.6** se propone el **Algoritmo 4.2.6** que se muestra y que calcula las matrices de cargas ortogonales disjuntas $\boldsymbol{B}$ y $\boldsymbol{C}$:

Algoritmo 4.2.6: CBPSO-ParafacALS donde $\boldsymbol{B}$ y $\boldsymbol{C}$ son ortogonales disjuntas

---

Datos de Entrada:

$\underline{\mathbf{X}}, R, PSOMaxIter, nPar, minIner, maxIner, wCognition, wSocial$

A partir de $\underline{\mathbf{X}}$ obtener las supermatrices: $\boldsymbol{X_A}, \boldsymbol{X_B}, \boldsymbol{X_C}$

Etapa 1.- Calcular las matrices ortogonales disjuntas $\boldsymbol{B}$ y $\boldsymbol{C}$:

$\boldsymbol{B} \leftarrow CBPSO(\boldsymbol{X_B}^T, R, PSOMaxIter, nPar, minIner, maxIner, wCognition, wSocial)$

$\boldsymbol{C} \leftarrow CBPSO(\boldsymbol{X_C}^T, R, PSOMaxIter, nPar, minIner, maxIner, wCognition, wSocial)$

Etapa 2.- Calcular la matriz ortogonal $\boldsymbol{A}$:

$\boldsymbol{A} \leftarrow \boldsymbol{X_A}[(\boldsymbol{C} \odot \boldsymbol{B})^T]^+$

Calcular el fit del Modelo

Datos de salida: $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$, fit

---

En la **Sección 2.2** se indica la expresión que permite calcular el fit del modelo. Ver la **Sección 3.3** para los datos de entrada y las instrucciones de la **Etapa 1**.

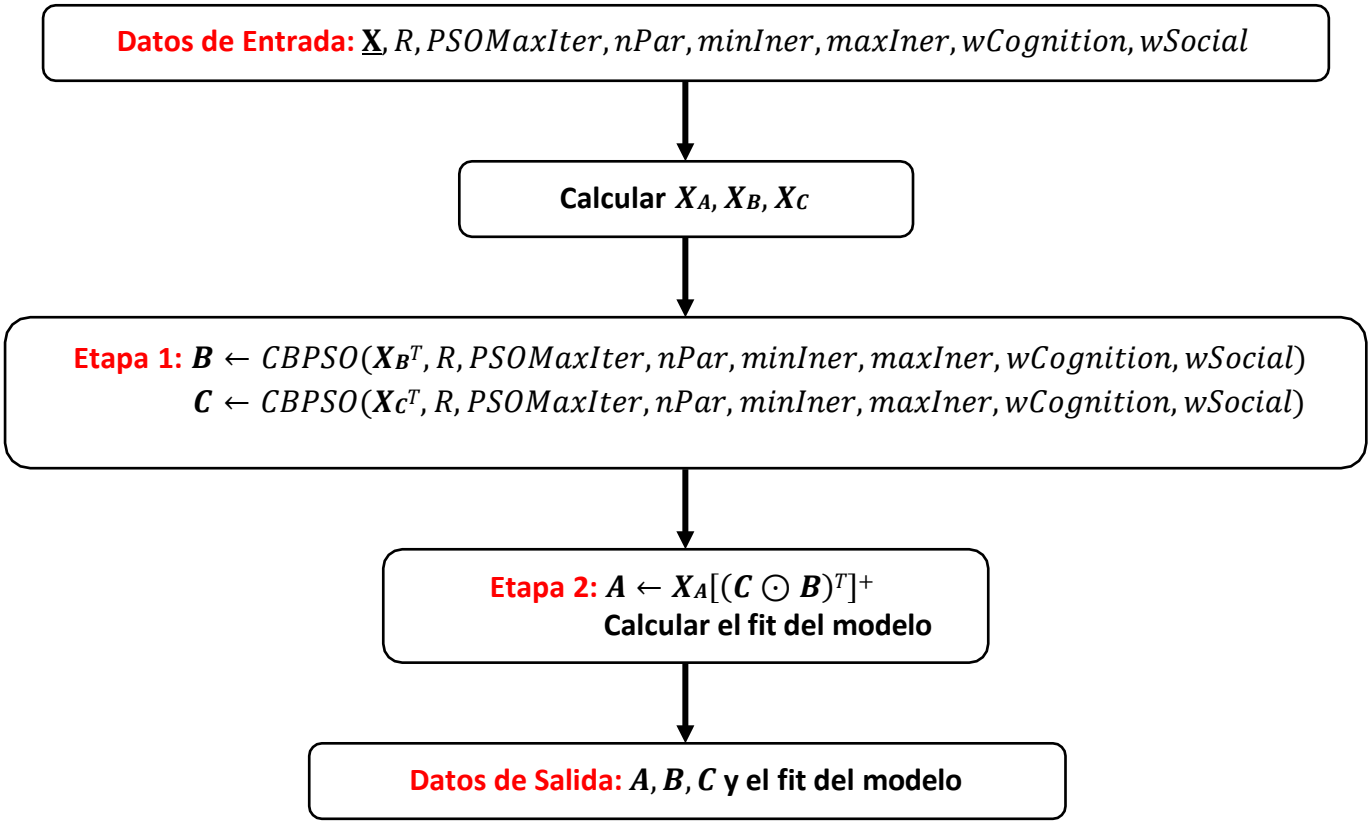


Figura 4.2.2: Diagrama de flujo del Algoritmo 4.2.6

Para resolver el **Modelo de Optimización 3.2.1.7** se propone el **Algoritmo 4.2.7** que se muestra y que calcula las matrices de cargas ortogonales disjuntas $\boldsymbol{A}$, $\boldsymbol{B}$ y $\boldsymbol{C}$:

Algoritmo 4.2.7: CBPSO-ParafacALS donde $\boldsymbol{A}$, $\boldsymbol{B}$ y $\boldsymbol{C}$ son ortogonales disjuntas

---

Datos de Entrada:

$\underline{\mathbf{X}}, R, PSOMaxIter, nPar, minIner, maxIner, wCognition, wSocial$

A partir de $\underline{\mathbf{X}}$ obtener las supermatrices: $\boldsymbol{X_A}, \boldsymbol{X_B}, \boldsymbol{X_C}$

Etapa 1.- Calcular las matrices ortogonales disjuntas $\boldsymbol{A}$, $\boldsymbol{B}$ y $\boldsymbol{C}$:

$\boldsymbol{A} \leftarrow CBPSO(\boldsymbol{X_A}^T, R, PSOMaxIter, nPar, minIner, maxIner, wCognition, wSocial)$

$\boldsymbol{B} \leftarrow CBPSO(\boldsymbol{X_B}^T, R, PSOMaxIter, nPar, minIner, maxIner, wCognition, wSocial)$

$\boldsymbol{C} \leftarrow CBPSO(\boldsymbol{X_C}^T, R, PSOMaxIter, nPar, minIner, maxIner, wCognition, wSocial)$

Etapa 2.- Calcular el fit del Modelo

Datos de salida: $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$, fit

---

En la **Sección 2.2** se indica la expresión que permite calcular el fit del modelo. Ver la **Sección 3.3** para los datos de entrada y las instrucciones de la **Etapa 1**.

Al observar los algoritmos de esta sección se aprecia que el cálculo de matrices de cargas ortogonales disjuntas en el modelo PARAFAC se realiza calculando la matriz de cargas ortogonal disjunta en un DPCA para una matriz de datos o tablas de 2 vías. En particular, para obtener la matriz de cargas ortogonal disjunta $\boldsymbol{A}$ en el modelo PARAFAC se aplica el algoritmo CBPSO-DC de la **Sección 3.3** a la supermatriz de cortes frontales $\boldsymbol{X_A}$. Para obtener la matriz de cargas ortogonal disjunta $\boldsymbol{B}$ en el modelo PARAFAC se aplica el algoritmo CBPSO-DC a la supermatriz de cortes horizontales $\boldsymbol{X_B}$ y, finalmente, para obtener la matriz de cargas ortogonal disjunta $\boldsymbol{C}$ en el modelo PARAFAC se aplica el algoritmo CBPSO-DC a la supermatriz de cortes verticales $\boldsymbol{X_C}$.

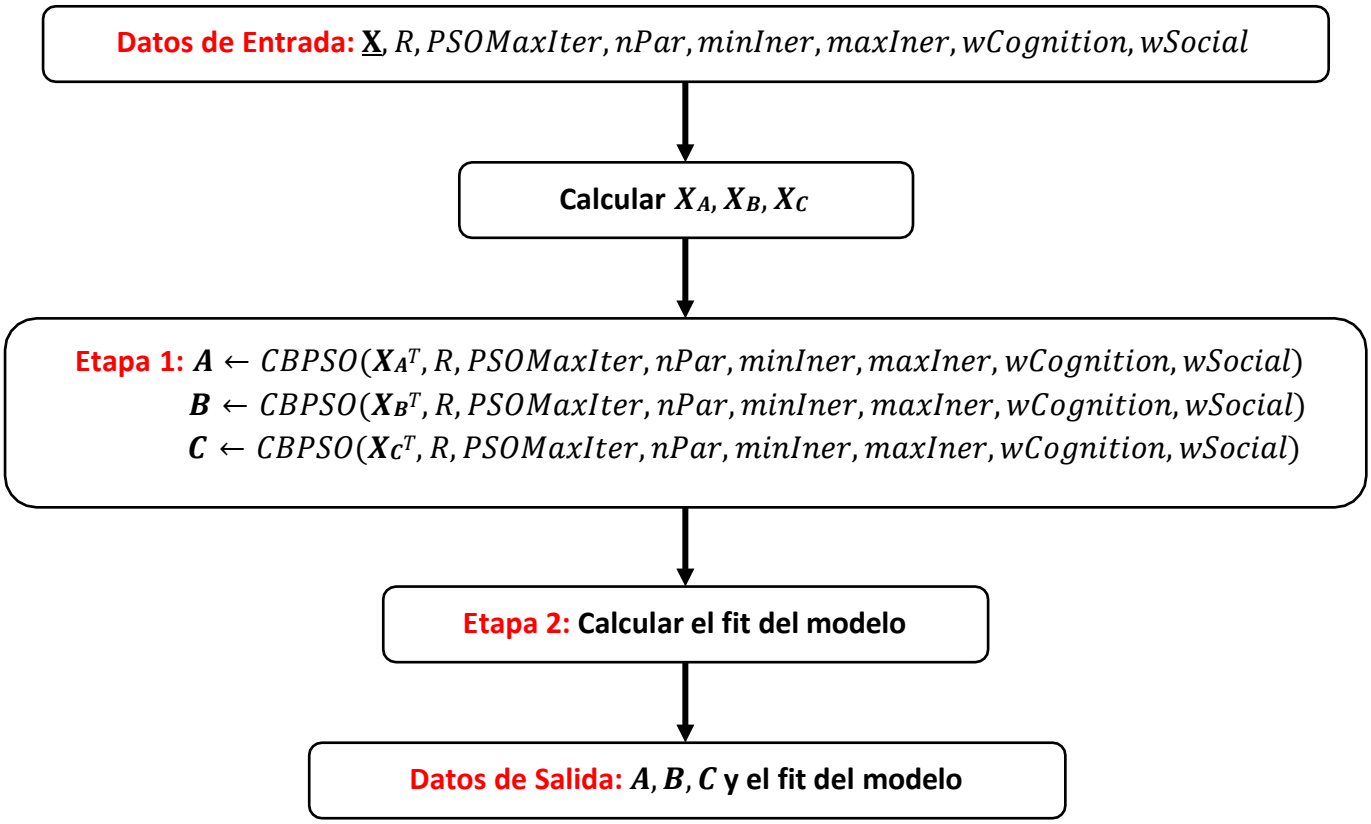


Figura 4.2.3: Diagrama de flujo del Algoritmo 4.2.7

4.3 EL ALGORITMO CBPSO-TuckALS3

En esta sección se presenta un algoritmo heurístico denominado CBPSO-TuckALS3 que permite una reducción dimensional en los tres modos, con P componentes para el primer modo, Q componentes para el segundo modo y R componentes para el tercer modo, según el modelo Tucker3 (ver **Sección 2.3**), y permite además calcular desde una hasta tres matrices de cargas ortogonales disjuntas.

Antes de realizar el cálculo de los componentes ortogonales disjuntos, el algoritmo CBPSO-TuckALS3 necesita calcular las matrices que se definen a continuación:

$$\boldsymbol{Y_A} = \boldsymbol{X_A}(\boldsymbol{C} \otimes \boldsymbol{B}) \in M_{I\times QR}$$

$$\boldsymbol{Y_B} = \boldsymbol{X_B}(\boldsymbol{C} \otimes \boldsymbol{A}) \in M_{J\times PR}$$

$$\boldsymbol{Y_C} = \boldsymbol{X_C}(\boldsymbol{B} \otimes \boldsymbol{A}) \in M_{K\times PQ}$$

Estas matrices sirven como punto de inicio para obtener matrices de cargas ortogonales disjuntas. A continuación en el **Algoritmo 4.3.1** se propone entonces un algoritmo "TuckALS3 Adaptado" para el cálculo de las matrices $\boldsymbol{Y_A}$, $\boldsymbol{Y_B}$ y $\boldsymbol{Y_C}$ necesarias en los algoritmos CBPSO-TuckALS3.

Algoritmo 4.3.1: Algoritmo TuckALS3 Adaptado (algoritmo auxiliar de los algoritmos CBPSO-TuckALS3)

Datos de entrada: $\underline{\mathbf{X}} \in T_{I\times J\times K}, P, Q, R$
Inicio
A partir de $\underline{\mathbf{X}}$ obtener la terna: $(I \quad J \quad K)$
A partir de $\underline{\mathbf{X}}$ obtener las supermatrices: $\boldsymbol{X_A}, \boldsymbol{X_B}, \boldsymbol{X_C}$
Construir la matriz $\boldsymbol{A}$ de tamaño $I \times P$
Construir la matriz $\boldsymbol{B}$ de tamaño $J \times Q$
Construir la matriz $\boldsymbol{C}$ de tamaño $K \times R$
$\boldsymbol{B} \leftarrow svd(\boldsymbol{X_B}, Q)$
$\boldsymbol{C} \leftarrow svd(\boldsymbol{X_C}, R)$
Repetir
$\boldsymbol{Y_A} \leftarrow \boldsymbol{X_A}(\boldsymbol{C} \otimes \boldsymbol{B})$
$\boldsymbol{A} \leftarrow svd(\boldsymbol{Y_A}, P)$
$\boldsymbol{Y_B} \leftarrow \boldsymbol{X_B}(\boldsymbol{C} \otimes \boldsymbol{A})$
$\boldsymbol{B} \leftarrow svd(\boldsymbol{Y_B}, Q)$
$\boldsymbol{Y_C} \leftarrow \boldsymbol{X_C}(\boldsymbol{B} \otimes \boldsymbol{A})$
$\boldsymbol{C} \leftarrow svd(\boldsymbol{Y_C}, R)$
Hasta que se cumpla $CriterioParadaTuckALS3$
$\boldsymbol{Y_A} \leftarrow \boldsymbol{X_A}(\boldsymbol{C} \otimes \boldsymbol{B})$
$\boldsymbol{Y_B} \leftarrow \boldsymbol{X_B}(\boldsymbol{C} \otimes \boldsymbol{A})$
$\boldsymbol{Y_C} \leftarrow \boldsymbol{X_C}(\boldsymbol{B} \otimes \boldsymbol{A})$
Fin
Datos de salida: $\boldsymbol{X_A}, \boldsymbol{X_B}, \boldsymbol{X_C}, \boldsymbol{Y_A}, \boldsymbol{Y_B}, \boldsymbol{Y_C}$

Para el criterio de parada etiquetado $CriterioParadaTuckALS3$ y para la notación de las instrucciones del **Algoritmo 4.3.1** ver la **Sección 2.3** en la que se encuentra el detalle del algoritmo TuckALS3.

Para resolver el **Modelo de Optimización 3.2.2.1** se propone el **Algoritmo 4.3.2** que se muestra y que calcula la matriz de cargas ortogonal disjunta $\boldsymbol{A}$:

Algoritmo 4.3.2: CBPSO-TuckALS3 donde $\boldsymbol{A}$ es ortogonal disjunta

---

Datos de Entrada:
$\underline{\mathbf{X}}, P, Q, R, ALSMaxIter, PSOMaxIter, nPar, minIner, maxIner, wCognition, wSocial$
Etapa 1.- Calcular las matrices $\boldsymbol{Y_A}, \boldsymbol{Y_B}, \boldsymbol{Y_C}$:
Aplicar el **Algoritmo 4.3.1** pasando como parámetros: $\underline{\mathbf{X}}, P, Q, R$
Etapa 2.- Calcular la matriz ortogonal disjunta $\boldsymbol{A}$:
$\boldsymbol{A} \leftarrow CBPSO(\boldsymbol{Y_A}^T, P, PSOMaxIter, nPar, minIner, maxIner, wCognition, wSocial)$
Etapa 3.- Calcular las matrices ortogonales $\boldsymbol{B}$ y $\boldsymbol{C}$, y el core $\underline{\mathbf{G}}$:
$\boldsymbol{C} \leftarrow svd(\boldsymbol{X_C}, R)$
Repetir
$\boldsymbol{B} \leftarrow svd(\boldsymbol{X_B}(\boldsymbol{C} \otimes \boldsymbol{A}), Q)$
$\boldsymbol{C} \leftarrow svd(\boldsymbol{X_C}(\boldsymbol{B} \otimes \boldsymbol{A}), R)$
Hasta que se cumpla $CriterioParada4.3.2$
$\boldsymbol{G_A} \leftarrow \boldsymbol{A}^T\boldsymbol{X_A}(\boldsymbol{C} \otimes \boldsymbol{B})$
Calcular el fit del Modelo
Datos de salida: $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$, $\boldsymbol{G_A}$, fit

---

El criterio de parada etiquetado como $CriterioParada4.3.2$ se cumple cuando se alcanza el número máximo de iteraciones (almacenado en el parámetro de entrada $ALSMaxIter$) o cuando las matrices $\boldsymbol{B}$ y $\boldsymbol{C}$ no difieren de forma significativa en dos iteraciones consecutivas. Se usa la **Ecuación 2.3.6** para calcular el fit del modelo. Ver la **Sección 3.3** para los datos de entrada y la instrucción de la **Etapa 2**.

Para resolver el **Modelo de Optimización 3.2.2.2** se propone el **Algoritmo 4.3.3** que se muestra y que calcula la matriz de cargas ortogonal disjunta $\boldsymbol{B}$:

Algoritmo 4.3.3: CBPSO-TuckALS3 donde $\boldsymbol{B}$ es ortogonal disjunta

---

Datos de Entrada:
$\underline{\mathbf{X}}, P, Q, R, ALSMaxIter, PSOMaxIter, nPar, minIner, maxIner, wCognition, wSocial$
Etapa 1.- Calcular las matrices $\boldsymbol{Y_A}, \boldsymbol{Y_B}, \boldsymbol{Y_C}$:
Aplicar el **Algoritmo 4.3.1** pasando como parámetros: $\underline{\mathbf{X}}, P, Q, R$
Etapa 2.- Calcular la matriz ortogonal disjunta $\boldsymbol{B}$:
$\boldsymbol{B} \leftarrow CBPSO(\boldsymbol{Y_B}^T, Q, PSOMaxIter, nPar, minIner, maxIner, wCognition, wSocial)$
Etapa 3.- Calcular las matrices ortogonales $\boldsymbol{A}$ y $\boldsymbol{C}$, y el core $\underline{\mathbf{G}}$:
$\boldsymbol{C} \leftarrow svd(\boldsymbol{X_C}, R)$
Repetir
$\boldsymbol{A} \leftarrow svd(\boldsymbol{X_A}(\boldsymbol{C} \otimes \boldsymbol{B}), P)$
$\boldsymbol{C} \leftarrow svd(\boldsymbol{X_C}(\boldsymbol{B} \otimes \boldsymbol{A}), R)$
Hasta que se cumpla $CriterioParada4.3.3$
$\boldsymbol{G_A} \leftarrow \boldsymbol{A}^T\boldsymbol{X_A}(\boldsymbol{C} \otimes \boldsymbol{B})$
Calcular el fit del Modelo
Datos de salida: $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$, $\boldsymbol{G_A}$, fit

---

El criterio de parada etiquetado como $CriterioParada4.3.3$ se cumple cuando se alcanza el número máximo de iteraciones (almacenado en el parámetro de entrada $ALSMaxIter$) o cuando las matrices $\boldsymbol{A}$ y $\boldsymbol{C}$ no difieren de forma significativa en dos iteraciones consecutivas. Se usa la **Ecuación 2.3.6** para calcular el fit del modelo. Ver la **Sección 3.3** para los datos de entrada y la instrucción de la **Etapa 2**.

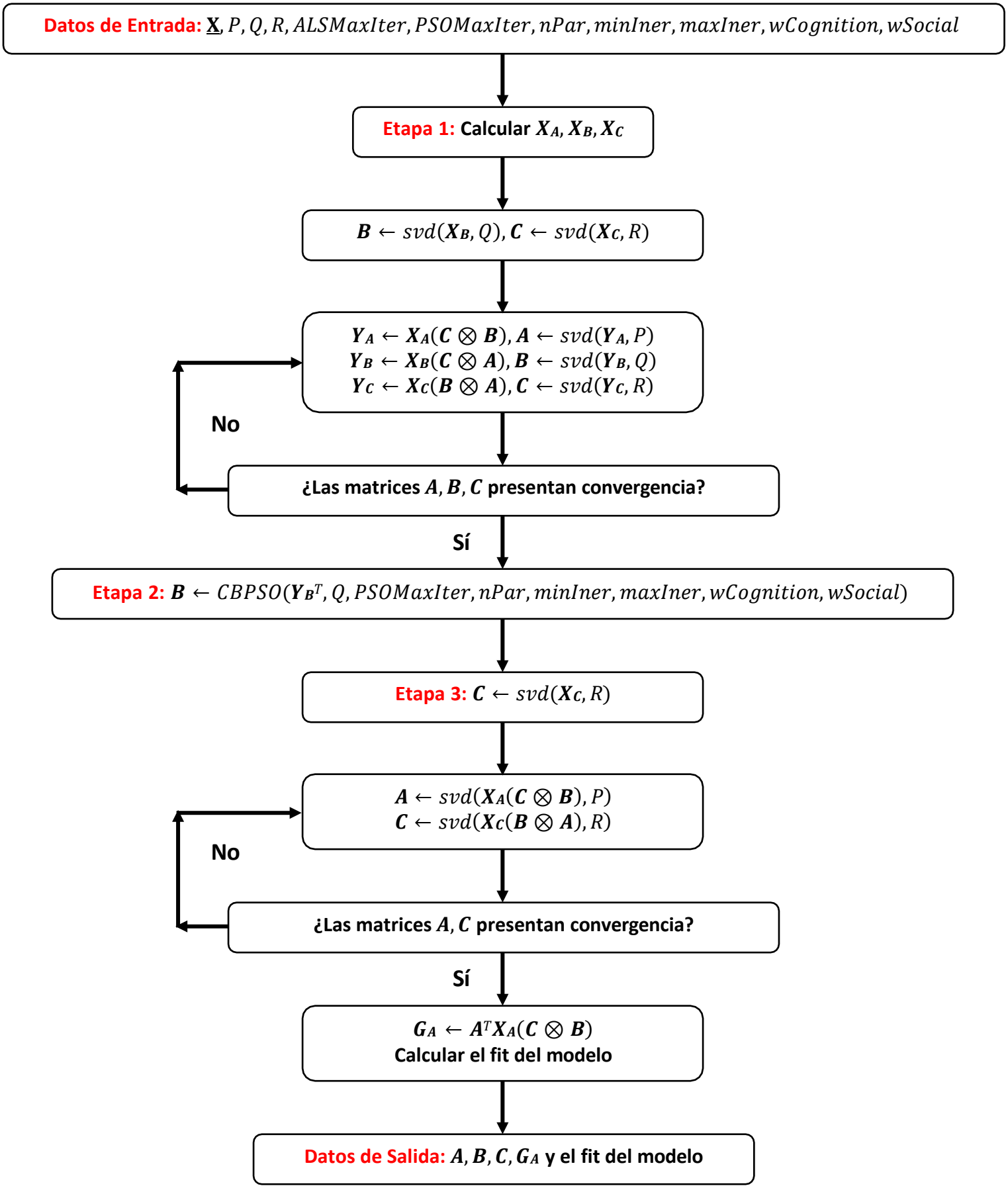


Figura 4.3.1: Diagrama de flujo del Algoritmo 4.3.3

Para resolver el **Modelo de Optimización 3.2.2.3** se propone el **Algoritmo 4.3.4** que se muestra y que calcula la matriz de cargas ortogonal disjunta $\boldsymbol{C}$:

Algoritmo 4.3.4: CBPSO-TuckALS3 donde $\boldsymbol{C}$ es ortogonal disjunta

---

Datos de Entrada:
$\underline{\mathbf{X}}, P, Q, R, ALSMaxIter, PSOMaxIter, nPar, minIner, maxIner, wCognition, wSocial$
Etapa 1.- Calcular las matrices $\boldsymbol{Y_A}, \boldsymbol{Y_B}, \boldsymbol{Y_C}$:
Aplicar el **Algoritmo 4.3.1** pasando como parámetros: $\underline{\mathbf{X}}, P, Q, R$
Etapa 2.- Calcular la matriz ortogonal disjunta $\boldsymbol{C}$:
$\boldsymbol{C} \leftarrow CBPSO(\boldsymbol{Y_C}^T, R, PSOMaxIter, nPar, minIner, maxIner, wCognition, wSocial)$
Etapa 3.- Calcular las matrices ortogonales $\boldsymbol{A}$ y $\boldsymbol{B}$, y el core $\underline{\mathbf{G}}$:
$\boldsymbol{B} \leftarrow svd(\boldsymbol{X_B}, Q)$
Repetir
$\boldsymbol{A} \leftarrow svd(\boldsymbol{X_A}(\boldsymbol{C} \otimes \boldsymbol{B}), P)$
$\boldsymbol{B} \leftarrow svd(\boldsymbol{X_B}(\boldsymbol{C} \otimes \boldsymbol{A}), Q)$
Hasta que se cumpla $CriterioParada4.3.4$
$\boldsymbol{G_A} \leftarrow \boldsymbol{A}^T\boldsymbol{X_A}(\boldsymbol{C} \otimes \boldsymbol{B})$
Calcular el fit del Modelo
Datos de salida: $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$, $\boldsymbol{G_A}$, fit

---

El criterio de parada etiquetado como $CriterioParada4.3.4$ se cumple cuando se alcanza el número máximo de iteraciones (almacenado en el parámetro de entrada $ALSMaxIter$) o cuando las matrices $\boldsymbol{A}$ y $\boldsymbol{B}$ no difieren de forma significativa en dos iteraciones consecutivas. Se usa la **Ecuación 2.3.6** para calcular el fit del modelo. Ver la **Sección 3.3** para los datos de entrada y la instrucción de la **Etapa 2**.

Para resolver el **Modelo de Optimización 3.2.2.4** se propone el **Algoritmo 4.3.5** que se muestra y que calcula las matrices de cargas ortogonales disjuntas $\boldsymbol{A}$ y $\boldsymbol{B}$:

Algoritmo 4.3.5: CBPSO-TuckALS3 donde $\boldsymbol{A}$ y $\boldsymbol{B}$ son ortogonales disjuntas

---

Datos de Entrada:
$\underline{\mathbf{X}}, P, Q, R, ALSMaxIter, PSOMaxIter, nPar, minIner, maxIner, wCognition, wSocial$
Etapa 1.- Calcular las matrices $\boldsymbol{Y_A}, \boldsymbol{Y_B}, \boldsymbol{Y_C}$:
Aplicar el **Algoritmo 4.3.1** pasando como parámetros: $\underline{\mathbf{X}}, P, Q, R$
Etapa 2.- Calcular la matrices ortogonales disjuntas $\boldsymbol{A}$ y $\boldsymbol{B}$:
$\boldsymbol{A} \leftarrow CBPSO(\boldsymbol{Y_A}^T, P, PSOMaxIter, nPar, minIner, maxIner, wCognition, wSocial)$
$\boldsymbol{B} \leftarrow CBPSO(\boldsymbol{Y_B}^T, Q, PSOMaxIter, nPar, minIner, maxIner, wCognition, wSocial)$
Etapa 3.- Calcular la matriz ortogonal $\boldsymbol{C}$ y el core $\underline{\mathbf{G}}$:
$\boldsymbol{C} \leftarrow svd(\boldsymbol{X_C}(\boldsymbol{B} \otimes \boldsymbol{A}), R)$
$\boldsymbol{G_A} \leftarrow \boldsymbol{A}^T \boldsymbol{X_A}(\boldsymbol{C} \otimes \boldsymbol{B})$
Calcular el fit del Modelo
Datos de salida: $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$, $\boldsymbol{G_A}$, fit

---

Se usa la **Ecuación 2.3.6** para calcular el fit del modelo. Ver la **Sección 3.3** para los datos de entrada y las instrucciones de la **Etapa 2**.

Para resolver el **Modelo de Optimización 3.2.2.5** se propone el **Algoritmo 4.3.6** que se muestra y que calcula las matrices de cargas ortogonales disjuntas $\boldsymbol{A}$ y $\boldsymbol{C}$:

Algoritmo 4.3.6: CBPSO-TuckALS3 donde $\boldsymbol{A}$ y $\boldsymbol{C}$ son ortogonales disjuntas

---

Datos de Entrada:
$\underline{\mathbf{X}}, P, Q, R, ALSMaxIter, PSOMaxIter, nPar, minIner, maxIner, wCognition, wSocial$
Etapa 1.- Calcular las matrices $\boldsymbol{Y_A}, \boldsymbol{Y_B}, \boldsymbol{Y_C}$:
Aplicar el **Algoritmo 4.3.1** pasando como parámetros: $\underline{\mathbf{X}}, P, Q, R$
Etapa 2.- Calcular la matrices ortogonales disjuntas $\boldsymbol{A}$ y $\boldsymbol{C}$:
$\boldsymbol{A} \leftarrow CBPSO(\boldsymbol{Y_A}^T, P, PSOMaxIter, nPar, minIner, maxIner, wCognition, wSocial)$
$\boldsymbol{C} \leftarrow CBPSO(\boldsymbol{Y_C}^T, R, PSOMaxIter, nPar, minIner, maxIner, wCognition, wSocial)$
Etapa 3.- Calcular la matriz ortogonal $\boldsymbol{B}$ y el core $\underline{\mathbf{G}}$:
$\boldsymbol{B} \leftarrow svd(\boldsymbol{X_B}(\boldsymbol{C} \otimes \boldsymbol{A}), Q)$
$\boldsymbol{G_A} \leftarrow \boldsymbol{A}^T \boldsymbol{X_A}(\boldsymbol{C} \otimes \boldsymbol{B})$
Calcular el fit del Modelo
Datos de salida: $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{G_A}$, fit

---

Se usa la **Ecuación 2.3.6** para calcular el fit del modelo. Ver la **Sección 3.3** para los datos de entrada y las instrucciones de la **Etapa 2**.

Para resolver el **Modelo de Optimización 3.2.2.6** se propone el **Algoritmo 4.3.7** que se muestra y que calcula las matrices de cargas ortogonales disjuntas $\boldsymbol{B}$ y $\boldsymbol{C}$:

Algoritmo 4.3.7: CBPSO-TuckALS3 donde $\boldsymbol{B}$ y $\boldsymbol{C}$ son ortogonales disjuntas

---

Datos de Entrada:
$\underline{\mathbf{X}}, P, Q, R, ALSMaxIter, PSOMaxIter, nPar, minIner, maxIner, wCognition, wSocial$
Etapa 1.- Calcular las matrices $\boldsymbol{Y_A}, \boldsymbol{Y_B}, \boldsymbol{Y_C}$:
Aplicar el **Algoritmo 4.3.1** pasando como parámetros: $\underline{\mathbf{X}}, P, Q, R$
Etapa 2.- Calcular la matrices ortogonales disjuntas $\boldsymbol{B}$ y $\boldsymbol{C}$:
$\boldsymbol{B} \leftarrow CBPSO(\boldsymbol{Y_B}^T, Q, PSOMaxIter, nPar, minIner, maxIner, wCognition, wSocial)$
$\boldsymbol{C} \leftarrow CBPSO(\boldsymbol{Y_C}^T, R, PSOMaxIter, nPar, minIner, maxIner, wCognition, wSocial)$
Etapa 3.- Calcular la matriz ortogonal $\boldsymbol{A}$ y el core $\underline{\mathbf{G}}$:
$\boldsymbol{A} \leftarrow svd(\boldsymbol{X_A}(\boldsymbol{C} \otimes \boldsymbol{B}), P)$
$\boldsymbol{G_A} \leftarrow \boldsymbol{A}^T \boldsymbol{X_A}(\boldsymbol{C} \otimes \boldsymbol{B})$
Calcular el fit del Modelo
Datos de salida: $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$, $\boldsymbol{G_A}$, fit

---

Se usa la **Ecuación 2.3.6** para calcular el fit del modelo. Ver la **Sección 3.3** para los datos de entrada y las instrucciones de la **Etapa 2**.

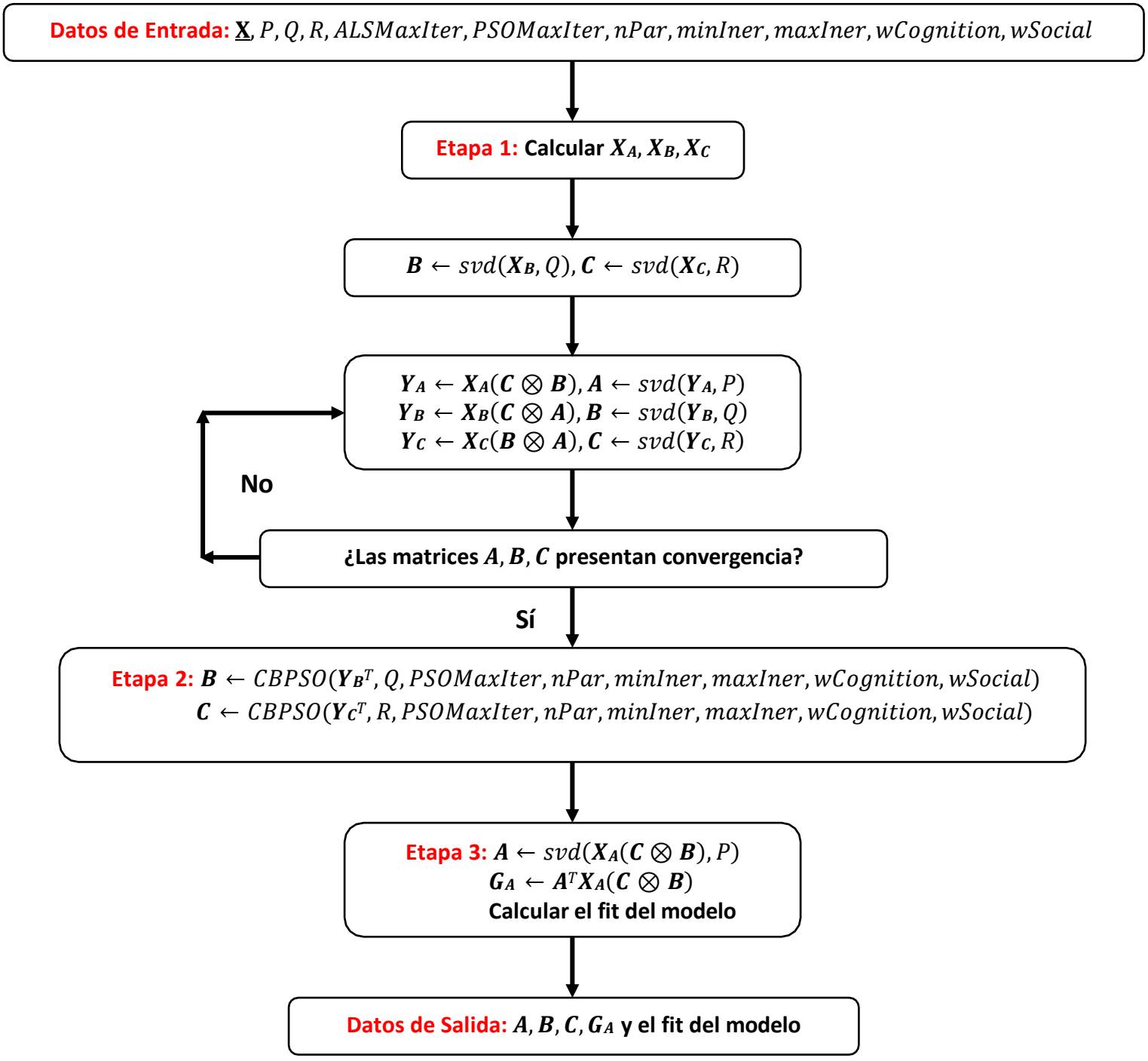


Figura 4.3.2: Diagrama de flujo del Algoritmo 4.3.7

Para resolver el **Modelo de Optimización 3.2.2.7** se propone el **Algoritmo 4.3.8** que se muestra y que calcula las matrices de cargas ortogonales disjuntas $\boldsymbol{A}$, $\boldsymbol{B}$ y $\boldsymbol{C}$:

Algoritmo 4.3.8: CBPSO-TuckALS3 donde $\boldsymbol{A}$, $\boldsymbol{B}$ y $\boldsymbol{C}$ son ortogonales disjuntas

---

Datos de Entrada:
$\underline{\mathbf{X}}, P, Q, R, ALSMaxIter, PSOMaxIter, nPar, minIner, maxIner, wCognition, wSocial$
Etapa 1.- Calcular las matrices $\boldsymbol{Y_A}, \boldsymbol{Y_B}, \boldsymbol{Y_C}$:
Aplicar el **Algoritmo 4.3.1** pasando como parámetros: $\underline{\mathbf{X}}, P, Q, R$
Etapa 2.- Calcular la matrices ortogonales disjuntas $\boldsymbol{A}$, $\boldsymbol{B}$ y $\boldsymbol{C}$:
$\boldsymbol{A} \leftarrow CBPSO(\boldsymbol{Y_A}^T, P, PSOMaxIter, nPar, minIner, maxIner, wCognition, wSocial)$
$\boldsymbol{B} \leftarrow CBPSO(\boldsymbol{Y_B}^T, Q, PSOMaxIter, nPar, minIner, maxIner, wCognition, wSocial)$
$\boldsymbol{C} \leftarrow CBPSO(\boldsymbol{Y_C}^T, R, PSOMaxIter, nPar, minIner, maxIner, wCognition, wSocial)$
Etapa 3.- Calcular el core $\underline{\mathbf{G}}$:
$\boldsymbol{G_A} \leftarrow \boldsymbol{A}^T \boldsymbol{X_A} (\boldsymbol{C} \otimes \boldsymbol{B})$
Calcular el fit del Modelo
Datos de salida: $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$, $\boldsymbol{G_A}$, fit

---

Se usa la **Ecuación 2.3.6** para calcular el fit del modelo. Ver la **Sección 3.3** para los datos de entrada y las instrucciones de la **Etapa 2**.

Al observar los algoritmos de esta sección se aprecia que el cálculo de matrices de cargas ortogonales disjuntas en el modelo Tucker3 se realiza calculando la matriz de cargas ortogonal disjunta en un DPCA para una matriz de datos o tablas de 2 vías. En particular, para obtener la matriz de cargas ortogonal disjunta $\boldsymbol{A}$ en el modelo Tucker3 se aplica el algoritmo CBPSO-DC de la **Sección 3.3** a la matriz $\boldsymbol{Y_A}$. Para obtener la matriz de cargas ortogonal disjunta $\boldsymbol{B}$ en el modelo Tucker3 se aplica el algoritmo CBPSO-DC a la matriz $\boldsymbol{Y_B}$ y, finalmente, para obtener la matriz de cargas ortogonal disjunta $\boldsymbol{C}$ en el modelo Tucker3 se aplica el algoritmo CBPSO-DC a la matriz $\boldsymbol{Y_C}$.

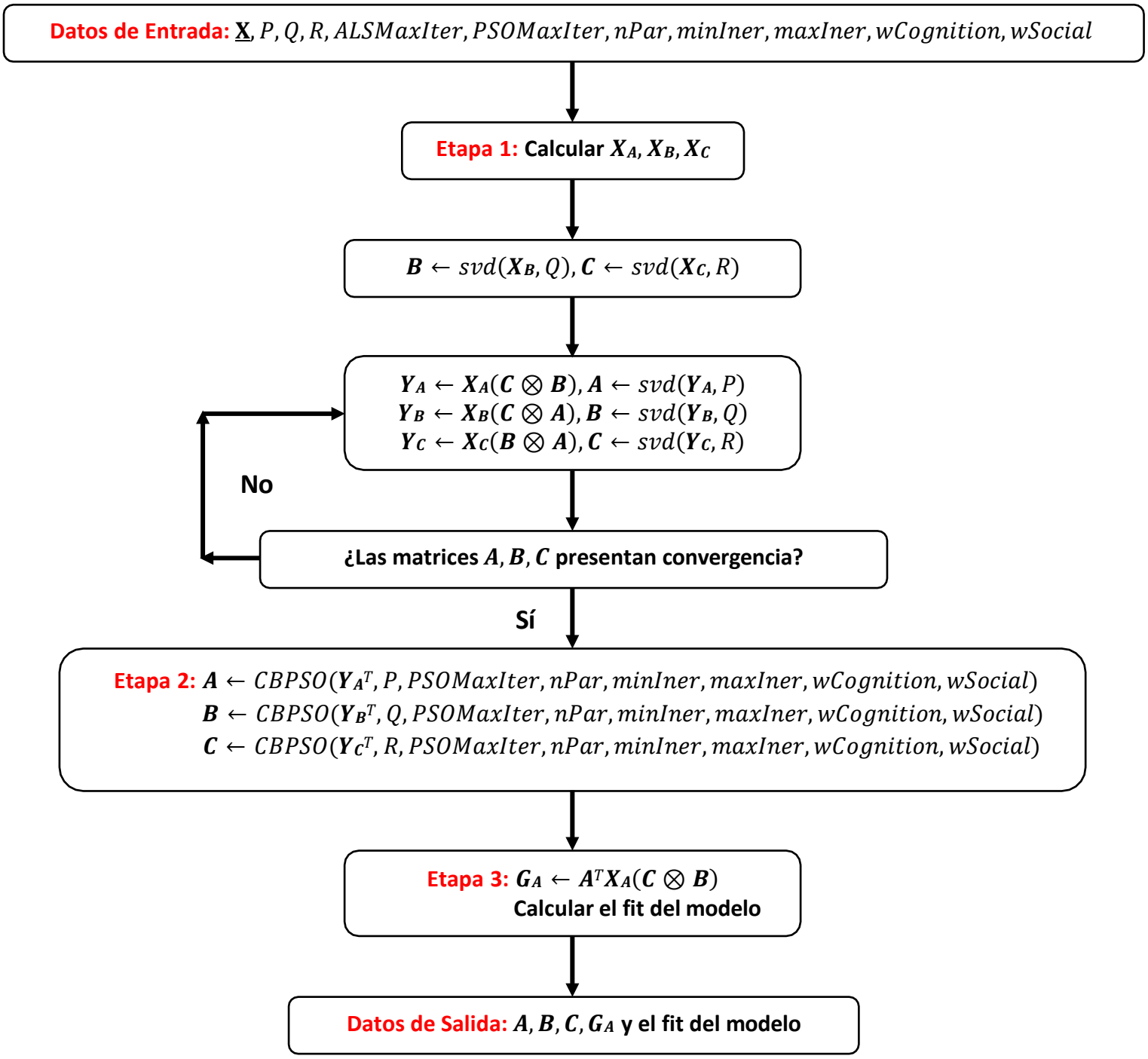


Figura 4.3.3: Diagrama de flujo del Algoritmo 4.3.8

4.4 EL ALGORITMO CBPSO-TuckALS2

A continuación se presenta un algoritmo heurístico denominado CBPSO-TuckALS2 que permite una reducción dimensional en dos de los tres modos de una tabla de 3 vías según los modelos Tucker2 (ver **Sección 2.4**), y permite además calcular una o dos matrices de cargas ortogonales disjuntas.

Antes de realizar el cálculo de los componentes ortogonales disjuntos para el modelo Tucker2-AB, el algoritmo CBPSO-TuckALS2 necesita calcular las matrices que se definen a continuación:

$$\boldsymbol{Y_A} = \boldsymbol{X_A}(\boldsymbol{I}_{K\times K} \otimes \boldsymbol{B}) \in M_{I\times QK}$$

$$\boldsymbol{Y_B} = \boldsymbol{X_B}(\boldsymbol{I}_{K\times K} \otimes \boldsymbol{A}) \in M_{J\times PK}$$

Estas matrices sirven como punto de inicio para obtener matrices de cargas ortogonales disjuntas. A continuación en el **Algoritmo 4.4.1** se propone entonces un algoritmo "TuckALS2-AB Adaptado" para calcular las matrices $\boldsymbol{Y_A}$ y $\boldsymbol{Y_B}$ necesarias en los algoritmos CBPSO-TuckALS2 para el modelo Tucker2-*AB*.

Algoritmo 4.4.1: Algoritmo TuckALS2-AB Adaptado (algoritmo auxiliar de los algoritmos CBPSO-TuckALS2, modelo Tucker2-*AB*)

Datos de entrada: $\underline{\mathbf{X}} \in T_{I\times J\times K}, P, Q$
Inicio
A partir de $\underline{\mathbf{X}}$ obtener la terna: $(I \quad J \quad K)$
A partir de $\underline{\mathbf{X}}$ obtener las supermatrices: $\boldsymbol{X_A}, \boldsymbol{X_B}$
Construir la matriz $\boldsymbol{A}$ de tamaño $I \times P$
Construir la matriz $\boldsymbol{B}$ de tamaño $J \times Q$
$\boldsymbol{B} \leftarrow svd(\boldsymbol{X_B}, Q)$
Repetir
$\boldsymbol{Y_A} \leftarrow \boldsymbol{X_A}(\boldsymbol{I}_{K\times K} \otimes \boldsymbol{B})$
$\boldsymbol{A} \leftarrow svd(\boldsymbol{Y_A}, P)$
$\boldsymbol{Y_B} \leftarrow \boldsymbol{X_B}(\boldsymbol{I}_{K\times K} \otimes \boldsymbol{A})$
$\boldsymbol{B} \leftarrow svd(\boldsymbol{Y_B}, Q)$
Hasta que se cumpla $CriterioParadaTuckALS2 - AB$
$\boldsymbol{Y_A} \leftarrow \boldsymbol{X_A}(\boldsymbol{I}_{K\times K} \otimes \boldsymbol{B})$
$\boldsymbol{Y_B} \leftarrow \boldsymbol{X_B}(\boldsymbol{I}_{K\times K} \otimes \boldsymbol{A})$
Fin
Datos de salida: $\boldsymbol{X_A}, \boldsymbol{X_B}, \boldsymbol{Y_A}, \boldsymbol{Y_B}$

La matriz $\boldsymbol{I}_{K\times K}$ es la identidad de $K \times K$. Para el criterio de parada etiquetado $CriterioParadaTuckALS2-AB$ y para la notación de las instrucciones del **Algoritmo 4.4.1** ver la **Sección 2.4** en la que se encuentra el detalle del algoritmo TuckALS2AB.

Para resolver el **Modelo de Optimización 3.2.3.1** se propone el **Algoritmo 4.4.2** que se muestra y que calcula la matriz de cargas ortogonal disjunta $\boldsymbol{A}$:

Algoritmo 4.4.2: CBPSO-TuckALS2, modelo Tucker2-AB donde $\boldsymbol{A}$ es ortogonal disjunta

---

Datos de Entrada:
$\underline{\mathbf{X}}, P, Q, ALSMaxIter, PSOMaxIter, nPar, minIner, maxIner, wCognition, wSocial$
Etapa 1.- Calcular las matrices $\boldsymbol{Y_A}, \boldsymbol{Y_B}$:
Aplicar el **Algoritmo 4.4.1** pasando como parámetros: $\underline{\mathbf{X}}, P, Q$
Etapa 2.- Calcular la matriz ortogonal disjunta $\boldsymbol{A}$:
$\boldsymbol{A} \leftarrow CBPSO(\boldsymbol{Y_A}^T, P, PSOMaxIter, nPar, minIner, maxIner, wCognition, wSocial)$
Etapa 3.- Calcular la matriz ortogonal $\boldsymbol{B}$ y el core $\underline{\mathbf{G}}$:
$\boldsymbol{B} \leftarrow svd(\boldsymbol{X_B}(\boldsymbol{I}_{K\times K} \otimes \boldsymbol{A}), Q)$
$\boldsymbol{G_A} \leftarrow \boldsymbol{A}^T\boldsymbol{X_A}(\boldsymbol{I}_{K\times K} \otimes \boldsymbol{B})$
Calcular el fit del Modelo
Datos de salida: $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{G_A}$, fit

---

La matriz $\boldsymbol{G_A}$ es la supermatriz de cortes frontales del core $\underline{\mathbf{G}}$. Se usa la **Ecuación 2.3.6** para calcular el fit del modelo. Ver la **Sección 3.3** para los datos de entrada y la instrucción de la **Etapa 2**.

Para resolver el **Modelo de Optimización 3.2.3.2** se propone el **Algoritmo 4.4.3** que se muestra y que calcula la matriz de cargas ortogonal disjunta $\boldsymbol{B}$:

Algoritmo 4.4.3: CBPSO-TuckALS2, modelo Tucker2-AB donde $\boldsymbol{B}$ es ortogonal disjunta

---

Datos de Entrada:

$\underline{\mathbf{X}}, P, Q, ALSMaxIter, PSOMaxIter, nPar, minIner, maxIner, wCognition, wSocial$

Etapa 1.- Calcular las matrices $\boldsymbol{Y_A}, \boldsymbol{Y_B}$:

Aplicar el **Algoritmo 4.4.1** pasando como parámetros: $\underline{\mathbf{X}}, P, Q$

Etapa 2.- Calcular la matriz ortogonal disjunta $\boldsymbol{B}$:

$\boldsymbol{B} \leftarrow CBPSO(\boldsymbol{Y_B}^T, Q, PSOMaxIter, nPar, minIner, maxIner, wCognition, wSocial)$

Etapa 3.- Calcular la matriz ortogonal $\boldsymbol{A}$ y el core $\underline{\mathbf{G}}$:

$\boldsymbol{A} \leftarrow svd(\boldsymbol{X_A}(\boldsymbol{I}_{K\times K} \otimes \boldsymbol{B}), P)$

$\boldsymbol{G_A} \leftarrow \boldsymbol{A}^T\boldsymbol{X_A}(\boldsymbol{I}_{K\times K} \otimes \boldsymbol{B})$

Calcular el fit del Modelo

Datos de salida: $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{G_A}$, fit

---

La matriz $\boldsymbol{G_A}$ es la supermatriz de cortes frontales del core $\underline{\mathbf{G}}$. Se usa la **Ecuación 2.3.6** para calcular el fit del modelo. Ver la **Sección 3.3** para los datos de entrada y la instrucción de la **Etapa 2**.

Para resolver el **Modelo de Optimización 3.2.3.3** se propone el **Algoritmo 4.4.4** que se muestra y que calcula las matrices de cargas ortogonales disjuntas $\boldsymbol{A}$ y $\boldsymbol{B}$:

Algoritmo 4.4.4: CBPSO-TuckALS2, modelo Tucker2-AB donde $\boldsymbol{A}$ y $\boldsymbol{B}$ son ortogonales disjuntas

---

Datos de Entrada:
$\underline{\mathbf{X}}, P, Q, ALSMaxIter, PSOMaxIter, nPar, minIner, maxIner, wCognition, wSocial$
Etapa 1.- Calcular las matrices $\boldsymbol{Y_A}, \boldsymbol{Y_B}$:
Aplicar el **Algoritmo 4.4.1** pasando como parámetros: $\underline{\mathbf{X}}, P, Q$
Etapa 2.- Calcular la matrices ortogonales disjuntas $\boldsymbol{A}$ y $\boldsymbol{B}$:
$\boldsymbol{A} \leftarrow CBPSO(\boldsymbol{Y_A}^T, P, PSOMaxIter, nPar, minIner, maxIner, wCognition, wSocial)$
$\boldsymbol{B} \leftarrow CBPSO(\boldsymbol{Y_B}^T, Q, PSOMaxIter, nPar, minIner, maxIner, wCognition, wSocial)$
Etapa 3.- Calcular el core $\underline{\mathbf{G}}$:
$\boldsymbol{G_A} \leftarrow \boldsymbol{A}^T \boldsymbol{X_A} (\boldsymbol{I}_{K\times K} \otimes \boldsymbol{B})$
Calcular el fit del Modelo
Datos de salida: $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{G_A}$, fit

---

La matriz $\boldsymbol{G_A}$ es la supermatriz de cortes frontales del core $\underline{\mathbf{G}}$. Se usa la **Ecuación 2.3.6** para calcular el fit del modelo. Ver la **Sección 3.3** para los datos de entrada y las instrucciones de la **Etapa 2**.

Antes de realizar el cálculo de los componentes ortogonales disjuntos para el modelo Tucker2-AC, el algoritmo CBPSO-TuckALS2 necesita calcular las matrices que se definen a continuación:

$$\boldsymbol{Y_A} = \boldsymbol{X_A}(\boldsymbol{C} \otimes \boldsymbol{I}_{J\times J}) \in M_{I\times JR}$$

$$\boldsymbol{Y_C} = \boldsymbol{X_C}(\boldsymbol{I}_{J\times J} \otimes \boldsymbol{A}) \in M_{K\times PJ}$$

Estas matrices sirven como punto de inicio para obtener matrices de cargas ortogonales disjuntas. A continuación en el **Algoritmo 4.4.5** se propone entonces un algoritmo "TuckALS2-AC Adaptado" para calcular las matrices $\boldsymbol{Y_A}$ y $\boldsymbol{Y_C}$ necesarias en los algoritmos CBPSO-TuckALS2 para el modelo Tucker2-*AC*.

Algoritmo 4.4.5: Algoritmo TuckALS2-AC Adaptado (algoritmo auxiliar de los algoritmos CBPSO-TuckALS2, modelo Tucker2-*AC*)

Datos de entrada: $\underline{\mathbf{X}} \in T_{I\times J\times K}, P, R$
Inicio
A partir de $\underline{\mathbf{X}}$ obtener la terna: $(I \quad J \quad K)$
A partir de $\underline{\mathbf{X}}$ obtener las supermatrices: $\boldsymbol{X_A}, \boldsymbol{X_C}$
Construir la matriz $\boldsymbol{A}$ de tamaño $I \times P$
Construir la matriz $\boldsymbol{C}$ de tamaño $K \times R$
$\boldsymbol{C} \leftarrow svd(\boldsymbol{X_C}, R)$
Repetir
$\boldsymbol{Y_A} \leftarrow \boldsymbol{X_A}(\boldsymbol{C} \otimes \boldsymbol{I}_{J\times J})$
$\boldsymbol{A} \leftarrow svd(\boldsymbol{Y_A}, P)$
$\boldsymbol{Y_C} \leftarrow \boldsymbol{X_C}(\boldsymbol{I}_{J\times J} \otimes \boldsymbol{A})$
$\boldsymbol{C} \leftarrow svd(\boldsymbol{Y_C}, R)$
Hasta que se cumpla $CriterioParadaTuckALS2 - AC$
$\boldsymbol{Y_A} \leftarrow \boldsymbol{X_A}(\boldsymbol{C} \otimes \boldsymbol{I}_{J\times J})$
$\boldsymbol{Y_C} \leftarrow \boldsymbol{X_C}(\boldsymbol{I}_{J\times J} \otimes \boldsymbol{A})$
Fin
Datos de salida: $\boldsymbol{X_A}, \boldsymbol{X_C}, \boldsymbol{Y_A}, \boldsymbol{Y_C}$

La matriz $\boldsymbol{I}_{J\times J}$ es la identidad de $J \times J$. Para el criterio de parada etiquetado $CriterioParadaTuckALS2 - AC$ y para la notación de las instrucciones del **Algoritmo 4.4.5** ver la **Sección 2.4** en la que se encuentra el detalle del algoritmo TuckALS2-AC.

Para resolver el **Modelo de Optimización 3.2.3.4** se propone el **Algoritmo 4.4.6** que se muestra y que calcula la matriz de cargas ortogonal disjunta $\boldsymbol{A}$:

Algoritmo 4.4.6: CBPSO-TuckALS2, modelo Tucker2-AC donde $\boldsymbol{A}$ es ortogonal disjunta

---

Datos de Entrada:

$\underline{\mathbf{X}}, P, R, ALSMaxIter, PSOMaxIter, nPar, minIner, maxIner, wCognition, wSocial$

Etapa 1.- Calcular las matrices $\boldsymbol{Y_A}, \boldsymbol{Y_C}$:

Aplicar el **Algoritmo 4.4.5** pasando como parámetros: $\underline{\mathbf{X}}, P, R$

Etapa 2.- Calcular la matriz ortogonal disjunta $\boldsymbol{A}$:

$\boldsymbol{A} \leftarrow CBPSO(\boldsymbol{Y_A}^T, P, PSOMaxIter, nPar, minIner, maxIner, wCognition, wSocial)$

Etapa 3.- Calcular la matriz ortogonal $\boldsymbol{C}$ y el core $\underline{\mathbf{G}}$:

$\boldsymbol{C} \leftarrow svd(\boldsymbol{X_C}(\boldsymbol{I}_{J\times J} \otimes \boldsymbol{A}), R)$

$\boldsymbol{G_A} \leftarrow \boldsymbol{A}^T\boldsymbol{X_A}(\boldsymbol{C} \otimes \boldsymbol{I}_{J\times J})$

Calcular el fit del Modelo

Datos de salida: $\boldsymbol{A}$, $\boldsymbol{C}$, $\boldsymbol{G_A}$, fit

---

La matriz $\boldsymbol{G_A}$ es la supermatriz de cortes frontales del core $\underline{\mathbf{G}}$. Se usa la **Ecuación 2.3.6** para calcular el fit del modelo. Ver la **Sección 3.3** para los datos de entrada y la instrucción de la **Etapa 2**.

Para resolver el **Modelo de Optimización 3.2.3.5** se propone el **Algoritmo 4.4.7** que se muestra y que calcula la matriz de cargas ortogonal disjunta $\boldsymbol{C}$:

Algoritmo 4.4.7: CBPSO-TuckALS2, modelo Tucker2-AC donde $\boldsymbol{C}$ es ortogonal disjunta

---

Datos de Entrada:

$\underline{\mathbf{X}}, P, R, ALSMaxIter, PSOMaxIter, nPar, minIner, maxIner, wCognition, wSocial$

Etapa 1.- Calcular las matrices $\boldsymbol{Y_A}, \boldsymbol{Y_C}$:

Aplicar el **Algoritmo 4.4.5** pasando como parámetros: $\underline{\mathbf{X}}, P, R$

Etapa 2.- Calcular la matriz ortogonal disjunta $\boldsymbol{C}$:

$\boldsymbol{C} \leftarrow CBPSO(\boldsymbol{Y_C}^T, R, PSOMaxIter, nPar, minIner, maxIner, wCognition, wSocial)$

Etapa 3.- Calcular la matriz ortogonal $\boldsymbol{A}$ y el core $\underline{\mathbf{G}}$:

$\boldsymbol{A} \leftarrow svd(\boldsymbol{X_A}(\boldsymbol{C} \otimes \boldsymbol{I}_{J\times J}), P)$

$\boldsymbol{G_A} \leftarrow \boldsymbol{A}^T\boldsymbol{X_A}(\boldsymbol{C} \otimes \boldsymbol{I}_{J\times J})$

Calcular el fit del Modelo

Datos de salida: $\boldsymbol{A}$, $\boldsymbol{C}$, $\boldsymbol{G_A}$, fit

---

La matriz $\boldsymbol{G_A}$ es la supermatriz de cortes frontales del core $\underline{\mathbf{G}}$. Se usa la **Ecuación 2.3.6** para calcular el fit del modelo. Ver la **Sección 3.3** para los datos de entrada y la instrucción de la **Etapa 2**.

Para resolver el **Modelo de Optimización 3.2.3.6** se propone el **Algoritmo 4.4.8** que se muestra y que calcula las matrices de cargas ortogonales disjuntas $\boldsymbol{A}$ y $\boldsymbol{C}$:

Algoritmo 4.4.8: CBPSO-TuckALS2, modelo Tucker2-AC donde $\boldsymbol{A}$ y $\boldsymbol{C}$ son ortogonales disjuntas

---

Datos de Entrada:

$\underline{\mathbf{X}}, P, R, ALSMaxIter, PSOMaxIter, nPar, minIner, maxIner, wCognition, wSocial$

Etapa 1.- Calcular las matrices $\boldsymbol{Y_A}, \boldsymbol{Y_C}$:

Aplicar el **Algoritmo 4.4.5** pasando como parámetros: $\underline{\mathbf{X}}, P, R$

Etapa 2.- Calcular la matrices ortogonales disjuntas $\boldsymbol{A}$ y $\boldsymbol{C}$:

$\boldsymbol{A} \leftarrow CBPSO(\boldsymbol{Y_A}^T, P, PSOMaxIter, nPar, minIner, maxIner, wCognition, wSocial)$

$\boldsymbol{C} \leftarrow CBPSO(\boldsymbol{Y_C}^T, R, PSOMaxIter, nPar, minIner, maxIner, wCognition, wSocial)$

Etapa 3.- Calcular el core $\underline{\mathbf{G}}$:

$\boldsymbol{G_A} \leftarrow \boldsymbol{A}^T \boldsymbol{X_A}(\boldsymbol{C} \otimes \boldsymbol{I}_{J\times J})$

Calcular el fit del Modelo

Datos de salida: $\boldsymbol{A}, \boldsymbol{C}, \boldsymbol{G_A}$, fit

---

La matriz $\boldsymbol{G_A}$ es la supermatriz de cortes frontales del core $\underline{\mathbf{G}}$. Se usa la **Ecuación 2.3.6** para calcular el fit del modelo. Ver la **Sección 3.3** para los datos de entrada y las instrucciones de la **Etapa 2**.

Antes de realizar el cálculo de los componentes ortogonales disjuntos para el modelo Tucker2-BC, el algoritmo CBPSO-TuckALS2 necesita calcular las matrices que se definen a continuación:

$$\boldsymbol{Y_B} = \boldsymbol{X_B}(\boldsymbol{C} \otimes \boldsymbol{I}_{I\times I}) \in M_{J\times IR}$$

$$\boldsymbol{Y_C} = \boldsymbol{X_C}(\boldsymbol{B} \otimes \boldsymbol{I}_{I\times I}) \in M_{K\times IQ}$$

Estas matrices sirven como punto de inicio para obtener matrices de cargas ortogonales disjuntas. A continuación en el **Algoritmo 4.4.9** se propone entonces un algoritmo "TuckALS2-BC Adaptado" para calcular las matrices $\boldsymbol{Y_B}$ y $\boldsymbol{Y_C}$ necesarias en los algoritmos CBPSO-TuckALS2 para el modelo Tucker2-*BC*.

Algoritmo 4.4.9: Algoritmo TuckALS2-BC Adaptado (algoritmo auxiliar de los algoritmos CBPSO-TuckALS2, modelo Tucker2-*BC*)

Datos de entrada: $\underline{\mathbf{X}} \in T_{I\times J\times K}, Q, R$
Inicio
A partir de $\underline{\mathbf{X}}$ obtener la terna: $(I \quad J \quad K)$
A partir de $\underline{\mathbf{X}}$ obtener las supermatrices: $\boldsymbol{X_B}, \boldsymbol{X_C}$
Construir la matriz $\boldsymbol{B}$ de tamaño $J \times Q$
Construir la matriz $\boldsymbol{C}$ de tamaño $K \times R$
$\boldsymbol{C} \leftarrow svd(\boldsymbol{X_C}, R)$
Repetir
$\boldsymbol{Y_B} \leftarrow \boldsymbol{X_B}(\boldsymbol{C} \otimes \boldsymbol{I}_{I\times I})$
$\boldsymbol{B} \leftarrow svd(\boldsymbol{Y_B}, Q)$
$\boldsymbol{Y_C} \leftarrow \boldsymbol{X_C}(\boldsymbol{B} \otimes \boldsymbol{I}_{I\times I})$
$\boldsymbol{C} \leftarrow svd(\boldsymbol{Y_C}, R)$
Hasta que se cumpla $CriterioParadaTuckALS2-BC$
$\boldsymbol{Y_B} \leftarrow \boldsymbol{X_B}(\boldsymbol{C} \otimes \boldsymbol{I}_{I\times I})$
$\boldsymbol{Y_C} \leftarrow \boldsymbol{X_C}(\boldsymbol{B} \otimes \boldsymbol{I}_{I\times I})$
Fin
Datos de salida: $\boldsymbol{X_B}, \boldsymbol{X_C}, \boldsymbol{Y_B}, \boldsymbol{Y_C}$

La matriz $\boldsymbol{I}_{I\times I}$ es la identidad de tamaño $I \times I$. Para el criterio de parada etiquetado $CriterioParadaTuckALS2-BC$ y para la notación de las instrucciones del **Algoritmo 4.4.9** ver la **Sección 2.4** en la que se encuentra el detalle del algoritmo TuckALS2-BC.

Para resolver el **Modelo de Optimización 3.2.3.7** se propone el **Algoritmo 4.4.10** que se muestra y que calcula la matriz de cargas ortogonal disjunta $\boldsymbol{B}$:

Algoritmo 4.4.10: CBPSO-TuckALS2, modelo Tucker2-BC donde $\boldsymbol{B}$ es ortogonal disjunta

---

Datos de Entrada:
$\underline{\mathbf{X}}, Q, R, ALSMaxIter, PSOMaxIter, nPar, minIner, maxIner, wCognition, wSocial$
Etapa 1.- Calcular las matrices $\boldsymbol{Y_B}, \boldsymbol{Y_C}$:
Aplicar el **Algoritmo 4.4.9** pasando como parámetros: $\underline{\mathbf{X}}, Q, R$
Etapa 2.- Calcular la matriz ortogonal disjunta $\boldsymbol{B}$:
$\boldsymbol{B} \leftarrow CBPSO(\boldsymbol{Y_B}^T, Q, PSOMaxIter, nPar, minIner, maxIner, wCognition, wSocial)$
Etapa 3.- Calcular la matriz ortogonal $\boldsymbol{C}$ y el core $\underline{\mathbf{G}}$:
$\boldsymbol{C} \leftarrow svd(\boldsymbol{X_C}(\boldsymbol{B} \otimes \boldsymbol{I}_{I\times I}), R)$
$\boldsymbol{G_B} \leftarrow \boldsymbol{B}^T\boldsymbol{X_B}(\boldsymbol{C} \otimes \boldsymbol{I}_{I\times I})$
Calcular el fit del Modelo
Datos de salida: $\boldsymbol{B}$, $\boldsymbol{C}$, $\boldsymbol{G_B}$, fit

---

La matriz $\boldsymbol{G_B}$ es la supermatriz de cortes horizontales del core $\underline{\mathbf{G}}$. Se usa la **Ecuación 2.3.6** para calcular el fit del modelo. Ver la **Sección 3.3** para los datos de entrada y la instrucción de la **Etapa 2**.

Para resolver el **Modelo de Optimización 3.2.3.8** se propone el **Algoritmo 4.4.11** que se muestra y que calcula la matriz de cargas ortogonal disjunta $\boldsymbol{C}$:

Algoritmo 4.4.11: CBPSO-TuckALS2, modelo Tucker2-BC donde $\boldsymbol{C}$ es ortogonal disjunta

---

Datos de Entrada:

$\underline{\mathbf{X}}, Q, R, ALSMaxIter, PSOMaxIter, nPar, minIner, maxIner, wCognition, wSocial$

Etapa 1.- Calcular las matrices $\boldsymbol{Y_B}, \boldsymbol{Y_C}$:

Aplicar el **Algoritmo 4.4.9** pasando como parámetros: $\underline{\mathbf{X}}, Q, R$

Etapa 2.- Calcular la matriz ortogonal disjunta $\boldsymbol{C}$:

$\boldsymbol{C} \leftarrow CBPSO(\boldsymbol{Y_C}^T, R, PSOMaxIter, nPar, minIner, maxIner, wCognition, wSocial)$

Etapa 3.- Calcular la matriz ortogonal $\boldsymbol{B}$ y el core $\underline{\mathbf{G}}$:

$\boldsymbol{B} \leftarrow svd(\boldsymbol{X_B}(\boldsymbol{C} \otimes \boldsymbol{I}_{I\times I}), Q)$

$\boldsymbol{G_B} \leftarrow \boldsymbol{B}^T\boldsymbol{X_B}(\boldsymbol{C} \otimes \boldsymbol{I}_{I\times I})$

Calcular el fit del Modelo

Datos de salida: $\boldsymbol{B}$, $\boldsymbol{C}$, $\boldsymbol{G_B}$, fit

---

La matriz $\boldsymbol{G_B}$ es la supermatriz de cortes horizontales del core $\underline{\mathbf{G}}$. Se usa la **Ecuación 2.3.6** para calcular el fit del modelo. Ver la **Sección 3.3** para los datos de entrada y la instrucción de la **Etapa 2**.

Para resolver el **Modelo de Optimización 3.2.3.9** se propone el **Algoritmo 4.4.12** que se muestra y que calcula las matrices de cargas ortogonales disjuntas $\boldsymbol{B}$ y $\boldsymbol{C}$:

Algoritmo 4.4.12: CBPSO-TuckALS2, modelo Tucker2-BC donde $\boldsymbol{B}$ y $\boldsymbol{C}$ son ortogonales disjuntas

---

Datos de Entrada:

$\underline{\mathbf{X}}, Q, R, ALSMaxIter, PSOMaxIter, nPar, minIner, maxIner, wCognition, wSocial$

Etapa 1.- Calcular las matrices $\boldsymbol{Y_B}$, $\boldsymbol{Y_C}$:

Aplicar el **Algoritmo 4.4.9** pasando como parámetros: $\underline{\mathbf{X}}, Q, R$

Etapa 2.- Calcular la matrices ortogonales disjuntas $\boldsymbol{B}$ y $\boldsymbol{C}$:

$\boldsymbol{B} \leftarrow CBPSO(\boldsymbol{Y_B}^T, Q, PSOMaxIter, nPar, minIner, maxIner, wCognition, wSocial)$

$\boldsymbol{C} \leftarrow CBPSO(\boldsymbol{Y_C}^T, R, PSOMaxIter, nPar, minIner, maxIner, wCognition, wSocial)$

Etapa 3.- Calcular el core $\underline{\mathbf{G}}$:

$\boldsymbol{G_B} \leftarrow \boldsymbol{B}^T \boldsymbol{X_B}(\boldsymbol{C} \otimes \boldsymbol{I}_{I\times I})$

Calcular el fit del Modelo

Datos de salida: $\boldsymbol{B}$, $\boldsymbol{C}$, $\boldsymbol{G_B}$, fit

---

La matriz $\boldsymbol{G_B}$ es la supermatriz de cortes horizontales del core $\underline{\mathbf{G}}$. Se usa la **Ecuación 2.3.6** para calcular el fit del modelo. Ver la **Sección 3.3** para los datos de entrada y las instrucciones de la **Etapa 2**.

4.5 EL ALGORITMO CBPSO-TuckALS1

Para finalizar este capítulo se presenta el último algoritmo heurístico que se propone en esta tesis denominado CBPSO-TuckALS1 que permite una reducción dimensional en un único modo de la tabla de 3 vías según los modelos Tucker1 (ver **Sección 2.5**), y calcula una única matriz de cargas ortogonal disjunta.

Para resolver el **Modelo de Optimización 3.2.4.1** se propone el **Algoritmo 4.5.1** que se muestra y que calcula la matriz de cargas ortogonal disjunta $\boldsymbol{A}$:

Algoritmo 4.5.1: CBPSO-TuckALS1, modelo Tucker1-A donde $\boldsymbol{A}$ es ortogonal disjunta

---

Datos de Entrada:

$\underline{\mathbf{X}}, P, PSOMaxIter, nPar, minIner, maxIner, wCognition, wSocial$

Etapa 1.- Calcular la matriz ortogonal disjunta $\boldsymbol{A}$:

A partir de $\underline{\mathbf{X}}$ obtener la supermatriz $\boldsymbol{X_A}$

$\boldsymbol{A} \leftarrow CBPSO(\boldsymbol{X_A}^T, P, PSOMaxIter, nPar, minIner, maxIner, wCognition, wSocial)$

Etapa 2.- Calcular el core $\underline{\mathbf{G}}$:

$\boldsymbol{G_A} \leftarrow \boldsymbol{A}^T\boldsymbol{X_A}$

Calcular el fit del Modelo

Datos de salida: $\boldsymbol{A}$, $\boldsymbol{G_A}$, fit

La matriz $\boldsymbol{G_A}$ es la supermatriz de cortes frontales del core $\underline{\mathbf{G}}$. Se usa la **Ecuación 2.3.6** para calcular el fit del modelo. Ver la **Sección 3.3** para los datos de entrada y la instrucción de la **Etapa 1**.

Para resolver el **Modelo de Optimización 3.2.4.2** se propone el **Algoritmo 4.5.2** que se muestra y que calcula la matriz de cargas ortogonal disjunta $\boldsymbol{B}$:

Algoritmo 4.5.2: CBPSO-TuckALS1, modelo Tucker1-B donde $\boldsymbol{B}$ es ortogonal disjunta

---

Datos de Entrada:

$\underline{\mathbf{X}}, Q, PSOMaxIter, nPar, minIner, maxIner, wCognition, wSocial$

Etapa 1.- Calcular la matriz ortogonal disjunta $\boldsymbol{B}$:

A partir de $\underline{\mathbf{X}}$ obtener la supermatriz $\boldsymbol{X_B}$

$\boldsymbol{B} \leftarrow CBPSO(\boldsymbol{X_B}^T, Q, PSOMaxIter, nPar, minIner, maxIner, wCognition, wSocial)$

Etapa 2.- Calcular el core $\underline{\mathbf{G}}$:

$\boldsymbol{G_B} \leftarrow \boldsymbol{B}^T\boldsymbol{X_B}$

Calcular el fit del Modelo

Datos de salida: $\boldsymbol{B}$, $\boldsymbol{G_B}$, fit

La matriz $\boldsymbol{G_B}$ es la supermatriz de cortes horizontales del core $\underline{\mathbf{G}}$. Se usa la **Ecuación 2.3.6** para calcular el fit del modelo. Ver la **Sección 3.3** para los datos de entrada y la instrucción de la **Etapa 1**.

Para resolver el **Modelo de Optimización 3.2.4.3** se propone el **Algoritmo 4.5.3** que se muestra y que calcula la matriz de cargas ortogonal disjunta $\boldsymbol{C}$:

Algoritmo 4.5.3: CBPSO-TuckALS1, modelo Tucker1-C donde $\boldsymbol{C}$ es ortogonal disjunta

---

Datos de Entrada:

$\underline{\mathbf{X}}, R, PSOMaxIter, nPar, minIner, maxIner, wCognition, wSocial$

Etapa 1.- Calcular la matriz ortogonal disjunta $\boldsymbol{C}$:

A partir de $\underline{\mathbf{X}}$ obtener la supermatriz $\boldsymbol{X_C}$

$\boldsymbol{C} \leftarrow CBPSO(\boldsymbol{X_C}^T, R, PSOMaxIter, nPar, minIner, maxIner, wCognition, wSocial)$

Etapa 2.- Calcular el core $\underline{\mathbf{G}}$:

$\boldsymbol{G_C} \leftarrow \boldsymbol{C}^T\boldsymbol{X_C}$

Calcular el fit del Modelo

Datos de salida: $\boldsymbol{C}$, $\boldsymbol{G_C}$, fit

La matriz $\boldsymbol{G_C}$ es la supermatriz de cortes verticales del core $\underline{\mathbf{G}}$. Se usa la **Ecuación 2.3.6** para calcular el fit del modelo. Ver la **Sección 3.3** para los datos de entrada y la instrucción de la **Etapa 1**.

CAPÍTULO CINCO: "APLICACIÓN DE LOS ALGORITMOS HEURÍSTICOS PROPUESTOS A DATOS REALES Y SIMULADOS"

En este capítulo se presentan algunos experimentos computacionales con los algoritmos heurísticos propuestos en el **Capítulo 4**. Estos experimentos se dividen en dos partes. En la primera parte se realiza un estudio de simulación en el que se generan de forma aleatoria dos tablas de 3 vías, la primera con una estructura disjunta según el modelo PARAFAC, y la segunda con una estructura disjunta según el modelo Tucker3. Se construyen las correspondientes matrices de cargas de los modelos de forma tal que se esconde la estructura disjunta en cada caso. El objetivo es determinar si los algoritmos son capaces de detectar si está presente dicha estructura disjunta en los datos. En la segunda parte se realizan experimentos con datos no simulados y tomados de la literatura científica donde se realizan análisis de componentes para tablas de 3 vías. El objetivo es poner en evidencia el principal beneficio de los algoritmos propuestos: en los modelos multivariantes de componentes para el análisis de tablas de 3 vías, los algoritmos calculan matrices de cargas que tienen una estructura simple, lo que facilita el análisis y la interpretación de los datos.

5.1 ENTORNO COMPUTACIONAL DE LOS EXPERIMENTOS

Los experimentos computacionales se llevaron a cabo en un ordenador con las siguientes características de hardware:

- **Sistema operativo:** Windows 10 para 64 bits
- **Memoria principal (RAM):** 8 Gigabytes
- **Procesador:** Intel Core i7-4510U 2-2.60 GHZ

Respecto del software se utilizaron las siguientes herramientas y lenguajes de programación:

- **Herramienta de desarrollo (IDE):** Visual Studio Express de Microsoft
- **Leguaje de programación:** C#.NET
- **Software estadístico:** R
- **Middleware:** R.NET y COM+

En esta tesis se construyó un software para realizar todos los experimentos computacionales. Se usó C#.NET principalmente para:

- Implementación de la interfaz gráfica de usuario (GUI) para la entrada de datos, el control de los cálculos y la entrega de resultados
- Implementación del algoritmo CBPSO-DC
- Implementación de los algoritmos ALS
- Implementación de los algoritmos de simulación
- Implementación de los algoritmos heurísticos propuestos

La entrada de datos y la presentación de resultados se llevaron a cabo con hojas de Excel. La comunicación entre C#.NET y Excel se estableció mediante un conector conocido como COM+.

Algunas partes del software que se desarrolló también fueron implementadas usando el lenguaje de programación de R, entre las que destacan:

- Generación de números aleatorios
- Descomposiciones matriciales SVD
- La matriz inversa generalizada de Moore-Penrose

Para la comunicación entre C#.NET y R se usó como conector R.NET el cual se puede instalar en Visual Studio como un paquete “NuGet”. Se decidió utilizar el software R para generar números aleatorios porque al tomar muestras de números y realizar las

correspondientes pruebas estadísticas de uniformidad, independencia, media, varianza y autocorrelación, los resultados obtenidos fueron satisfactorios. Por tal razón se tomó la decisión de generar los números aleatorios con R. En particular, se utilizó la función "runif" del paquete **stats**.

En R se instaló el paquete **irlba** para las descomposiciones SVD y se utilizó la función "irlba". Esta función calcula de una forma rápida las descomposiciones SVD parciales, es decir, aquellas descomposiciones matriciales SVD que usan los primeros w valores singulares (donde w es un parámetro predeterminado). Por tanto, esta función no usa ni calcula los demás valores singulares (a partir del valor singular $w + 1$ ya no realiza cálculos), por lo que computa la mencionada descomposición parcial SVD de manera rápida para matrices de gran tamaño (ideal por ejemplo para las supermatrices de cortes frontales, horizontales y verticales de una tabla de 3 vías). Además se usó la función "ginv" del paquete **MASS** para calcular la matriz inversa generalizada de Moore-Penrose.

En la **Figura 5.1.1** se muestra la arquitectura que se propone para el software multivariante construido en esta tesis cuyo principal propósito es el cálculo de componentes ortogonales disjuntos en las matrices de cargas de los modelos multivariantes de componentes.

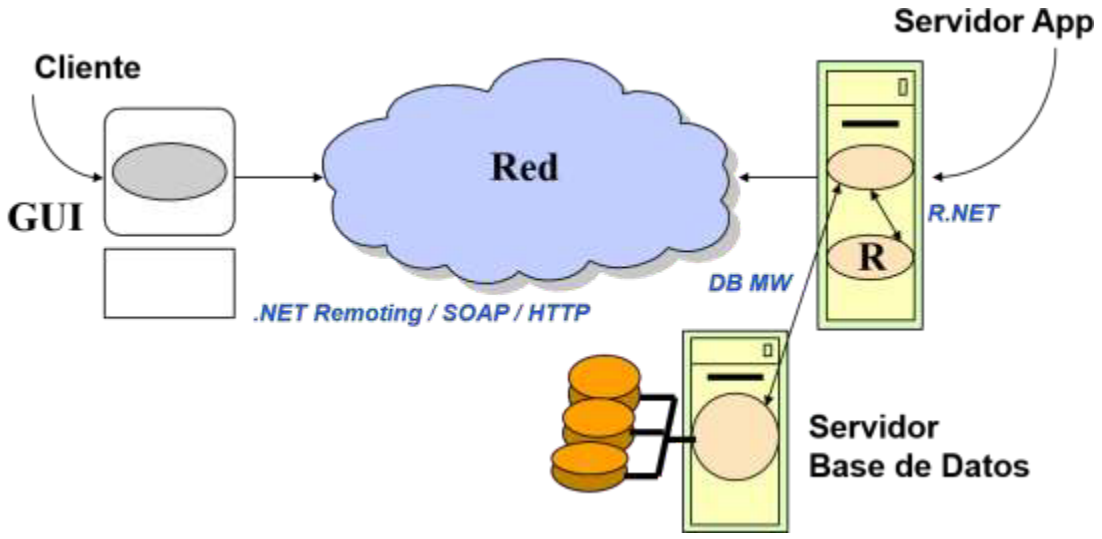


Figura 5.1.1: Arquitectura del Software

El proceso cliente implementa la interfaz gráfica para poder interactuar con el usuario, además envía la petición por el cálculo de componentes ortogonales disjuntos. El proceso servidor recibe el requerimiento y realiza el cálculo, se comunica con R vía R.NET para generar números aleatorios y para las descomposiciones parciales SVD. Una vez que el servidor finaliza el cálculo, los resultados se envían de regreso al cliente para la visualización y análisis del usuario. Los resultados se pueden almacenar en un repositorio vía algún conector DB MW para bases de datos.

5.2 APLICACIÓN CON DATOS SIMULADOS DE 3 VÍAS

En esta sección se realiza un estudio de simulación que tiene dos partes. El objetivo de la primera parte es determinar si el algoritmo CBPSO-ParafacALS puede detectar la estructura disjunta oculta en una tabla de 3 vías, de acuerdo al modelo PARAFAC. Para tal efecto se propone un primer algoritmo de simulación que implementa una función $\boldsymbol{\varphi_1}$ con la siguiente entrada y salida:

$$\boldsymbol{\varphi_1}: \mathbb{N}^4 \times \mathbb{N}^R \times \mathbb{N}^R \times \mathbb{N}^R \rightarrow T_{I\times J\times K}$$

$$(\{I,J,K,R\},\{\alpha_r\}_{r=1}^{R},\{\beta_r\}_{r=1}^{R},\{\gamma_r\}_{r=1}^{R}) \mapsto \boldsymbol{\varphi_1}(\{I,J,K,R\},\{\alpha_r\}_{r=1}^{R},\{\beta_r\}_{r=1}^{R},\{\gamma_r\}_{r=1}^{R})$$

El primer algoritmo de simulación recibe como entrada:

- Las dimensiones originales I, J, K de cada modo
- El número R de componentes del modelo PARAFAC. Por tal motivo se tienen las restricciones $R < I, R < J, R < K$
- Una sucesión finita $\{\alpha_r\}_{r=1}^{R}$ para el primer modo (individuos) cuyo $r-ésimo$ término α_r indica el número de entidades originales que van a estar representadas por la entidad latente r del modelo. Por tal motivo se tiene la restricción $I = \sum_{r=1}^{R} \alpha_r$
- Una sucesión finita $\{\beta_r\}_{r=1}^{R}$ para el segundo modo (variables) cuyo $r-ésimo$ término β_r indica el número de entidades originales que van a estar representadas por la entidad latente r del modelo. Por tal motivo se tiene la restricción $J = \sum_{r=1}^{R} \beta_r$
- Una sucesión finita $\{\gamma_r\}_{r=1}^{R}$ para el tercer modo (situaciones o tiempos) cuyo $r-ésimo$ término γ_r indica el número de entidades originales que van a estar representadas por la entidad latente r del modelo. Por tal motivo se tiene la restricción $K = \sum_{r=1}^{R} \gamma_r$

El primer algoritmo de simulación construye aleatoriamente una tabla de 3 vías $\underline{\mathbf{X}}$ (y la entrega como salida) de tamaño $I \times J \times K$ con una estructura disjunta que se puede reducir a R componentes en cada modo, según el modelo PARAFAC.

El objetivo de la segunda parte del estudio de simulación es determinar si el algoritmo CBPSO-TuckALS3 puede detectar la estructura disjunta oculta en una tabla de 3 vías, de

acuerdo al modelo Tucker3. Para tal efecto se propone un segundo algoritmo de simulación que implementa una función $\boldsymbol{\varphi_2}$ con la siguiente entrada y salida:

$$\boldsymbol{\varphi_2}: \mathbb{N}^6 \times \mathbb{N}^P \times \mathbb{N}^Q \times \mathbb{N}^R \rightarrow T_{I\times J\times K}$$

$$(\{I,J,K,P,Q,R\}, \{\alpha_p\}_{p=1}^{P}, \{\beta_q\}_{q=1}^{Q}, \{\gamma_r\}_{r=1}^{R}) \mapsto \boldsymbol{\varphi_2}(\{I,J,K,P,Q,R\}, \{\alpha_p\}_{p=1}^{P}, \{\beta_q\}_{q=1}^{Q}, \{\gamma_r\}_{r=1}^{R})$$

El segundo algoritmo de simulación recibe como entrada:

- Las dimensiones originales I, J, K de cada modo
- Para un modelo Tucker3, el número P de componentes en el primer modo, el número Q de componentes en el segundo modo y el número R de componentes en el tercer modo. Por tal motivo se tienen las restricciones $P < I, Q < J, R < K$
- Una sucesión finita $\{\alpha_p\}_{p=1}^{P}$ para el primer modo (individuos) cuyo $p-$ésimo término α_p indica el número de entidades originales que van a estar representadas por la entidad latente p del modelo. Por tal motivo se tiene la restricción $I = \sum_{p=1}^{P} \alpha_p$
- Una sucesión finita $\{\beta_q\}_{q=1}^{Q}$ para el segundo modo (variables) cuyo $q-$ésimo término β_q indica el número de entidades originales que van a estar representadas por la entidad latente q del modelo. Por tal motivo se tiene la restricción $J = \sum_{q=1}^{Q} \beta_q$
- Una sucesión finita $\{\gamma_r\}_{r=1}^{R}$para el tercer modo (situaciones o tiempos) cuyo $r-$ésimo término γ_r indica el número de entidades originales que van a estar representadas por la entidad latente r del modelo. Por tal motivo se tiene la restricción $K = \sum_{r=1}^{R} \gamma_r$

El segundo algoritmo de simulación construye aleatoriamente una tabla de 3 vías $\underline{\mathbf{X}}$ (y la entrega como salida) de tamaño $I \times J \times K$ con una estructura disjunta que se puede reducir a P, Q, R componentes en el primer, segundo y tercer modo, respectivamente, según el modelo Tucker3.

Para las dos partes del estudio de simulación, los valores de los parámetros del algoritmo CBPSO-ParafacALS y del algoritmo CBPSO-TuckALS3 que se usaron para los resultados que se presentan en la **Sección 5.2.1** y en la **Sección 5.2.2**, son los mismos:

- $ALSMaxIter = 8000$
- $Tol = 10^{-6}$
- $PSOMaxIter = 20$
- $nPar = 40$
- $minIner = 0.8$
- $maxIner = 1.2$
- $wCognition = 2$
- $wSocial = 2$

Se probaron 5 configuraciones diferentes para los valores de los parámetros. La mostrada anteriormente es la configuración común para ambas partes del estudio de simulación que presentó los mejores resultados.

Las otras 4 configuraciones utilizadas se resumen en la **Tabla 5.2.1**:

Tabla 5.2.1: Configuraciones para los parámetros en el estudio de simulación

PARÁMETROS	CONFIGURACIÓN 1	CONFIGURACIÓN 2	CONFIGURACIÓN 3	CONFIGURACIÓN 4
$ALSMaxIter$	8000	8000	10000	10000
Tol	10^{-5}	10^{-6}	10^{-5}	10^{-6}
$PSOMaxIter$	30	20	30	20
$nPar$	40	50	40	50
$minIner$	0.5	0.75	0.5	0.75
$maxIner$	1.5	1.25	1.5	1.25
$wCognition$	1.5	1.8	1.5	1.8
$wSocial$	2.5	3	2.5	3

5.2.1 APLICANDO EL ALGORITMO CBPSO-ParafacALS A DATOS SIMULADOS CON UNA ESTRUCTURA DISJUNTA PARA EL MODELO PARAFAC

Sea $\underline{\mathbf{X}}$ una tabla de 3 vías con I individuos, J variables y K situaciones o tiempos. Suponga que el primer modo (relacionado con la matriz de cargas $\boldsymbol{A}$) tiene R individuos latentes ($R < I$), el segundo modo (relacionado con la matriz de cargas $\boldsymbol{B}$) tiene R variables latentes ($R < J$) y, finalmente, el tercer modo (relacionado con la matriz de cargas $\boldsymbol{C}$) tiene R situaciones o tiempos latentes ($R < K$).

Suponga que $sx_1, \dots, sx_I$ son los I individuos originales. Por otro lado $sy_1, \dots, sy_R$ son los R individuos latentes. Considere la combinación lineal:

$$sy_r = a_{1,r}sx_1 + \cdots + a_{I,r}sx_I;\ r = 1,2,3, \dots, R$$

Si se desea que los m individuos originales consecutivos sx_n, sx_{n+1}, … , $sx_{n+(m-1)}$ estén representados por el individuo latente sy_r entonces los correspondientes escalares $a_{n,r}, a_{n+1,r}, \dots, a_{n+(m-1),r}$ se definen como variables aleatorias uniformes discretas independientes cuyo conjunto soporte son los números enteros del 70 al 100 incluidos (pesos altos para ganar representatividad en el individuo latente). Los demás escalares de esa misma combinación lineal se definen como variables aleatorias uniformes discretas independientes cuyo conjunto soporte son los números enteros del 1 al 30 incluidos (pesos bajos para no tener representatividad en el individuo latente). Este procedimiento se debe realizar para cada r desde 1 hasta R. Al hacerlo, tener presente que cada individuo original debe tener una fuerte presencia en un único individuo latente. A la matriz de tamaño $I \times R$ que contiene los escalares de todas las combinaciones lineales se le aplica el proceso de ortonormalización vectorial de Gram-Schmidt para poder construir una matriz ortogonal. De esta manera se alcanza una reducción dimensional disjunta en la matriz de cargas $\boldsymbol{A}$. Este proceso de dos etapas se lo va a representar con las instrucciones:

$$\boldsymbol{A_{temp}} \leftarrow CLD(sx_1, \dots, sx_I;\ \{\alpha_r\}_{r=1}^{R})$$

$$\boldsymbol{A} \leftarrow GS(\boldsymbol{A_{temp}})$$

La primera instrucción corresponde a las combinaciones lineales y la segunda instrucción corresponde al proceso de ortonormalización. Para la matriz de cargas $\boldsymbol{B}$ se trabaja de la misma manera. Considere que $vx_1, \dots, vx_J$ son las J variables originales. Además $vy_1, \dots, vy_R$ son las R variables latentes. Sea la combinación lineal:

$$vy_r = b_{1,r}vx_1 + \cdots + b_{J,r}vx_J;\ r = 1,2,3, \dots, R$$

Para que las m variables originales consecutivas vx_n, vx_{n+1}, … , $vx_{n+(m-1)}$ estén representadas por la variable latente vy_r, entonces los correspondientes escalares $b_{n,r}, b_{n+1,r}, \ldots, b_{n+(m-1),r}$ se definen como variables aleatorias uniformes discretas independientes cuyo conjunto soporte son los números enteros del 70 al 100 incluidos (pesos altos para ganar representatividad en la variable latente). Los demás escalares de esa misma combinación lineal se definen como variables aleatorias uniformes discretas independientes cuyo conjunto soporte son los números enteros del 1 al 30 incluidos (pesos bajos para no tener representatividad en la variable latente). Este procedimiento se debe realizar para cada r desde 1 hasta R. Al hacerlo, tener presente que cada variable original debe tener una fuerte presencia en una única variable latente. A la matriz de tamaño $J \times R$ que contiene los escalares de todas las combinaciones lineales se le aplica el conocido proceso de ortonormalización vectorial de Gram-Schmidt para poder construir una matriz ortogonal. De esta manera se alcanza una reducción dimensional disjunta en la matriz de cargas $\boldsymbol{B}$. Este proceso de dos etapas se lo va a representar con las instrucciones:

$$\boldsymbol{B_{temp}} \leftarrow CLD(vx_1, \ldots, vx_J;\ \{\beta_r\}_{r=1}^{R})$$

$$\boldsymbol{B} \leftarrow GS(\boldsymbol{B_{temp}})$$

Para finalizar se procede con la matriz de cargas $\boldsymbol{C}$ tal como se hizo con las 2 matrices anteriores. Sean $tx_1, \ldots, tx_K$ las K situaciones o tiempos originales. Sean $ty_1, \ldots, ty_R$ las R situaciones o tiempos latentes. Dada la combinación lineal:

$$ty_r = c_{1,r}tx_1 + \cdots + c_{K,r}tx_K;\ r = 1,2,3, \ldots, R$$

Para que las m situaciones o tiempos originales consecutivos $tx_n, tx_{n+1}, \ldots, tx_{n+(m-1)}$ estén representados por la situación o tiempo latente ty_r, entonces los escalares $c_{n,r}$, $c_{n+1,r}$, … , $c_{n+(m-1),r}$ se definen como variables aleatorias uniformes discretas independientes cuyo conjunto soporte son los números enteros del 70 al 100 incluidos (pesos altos para ganar representatividad en la situación o tiempo latente). Los demás escalares de esa misma combinación lineal se definen como variables aleatorias uniformes discretas independientes cuyo conjunto soporte son los números enteros del 1 al 30 incluidos (pesos bajos para no tener representatividad en la situación o tiempo latente). Este procedimiento se debe realizar para cada r desde 1 hasta R. Al hacerlo, tener presente que cada situación o tiempo original debe tener una fuerte presencia en una única situación o tiempo latente.

A la matriz de tamaño $K \times R$ que contiene los escalares de todas las combinaciones lineales se le aplica el proceso de ortonormalización vectorial de Gram-Schmidt para poder construir una matriz ortogonal. De esta manera se alcanza una reducción dimensional disjunta en la matriz de cargas $\boldsymbol{C}$. Este proceso de dos etapas se lo va a representar con las instrucciones:

$$\boldsymbol{C_{temp}} \leftarrow CLD(tx_1, \dots, tx_K;\ \{\gamma_r\}_{r=1}^{R})$$

$$\boldsymbol{C} \leftarrow GS(\boldsymbol{C_{temp}})$$

Para completar la construcción de $\underline{\mathbf{X}}$ se usa la ecuación matricial:

$$\boldsymbol{X_A} = \boldsymbol{A}(\boldsymbol{C} \odot \boldsymbol{B})^T$$

La matriz $\boldsymbol{X_A}$ de tamaño $I \times JK$ contiene los cortes frontales de la tabla de 3 vías $\underline{\mathbf{X}}$. El **Algoritmo 5.2.1** resume este primer algoritmo de simulación:

Algoritmo 5.2.1: Algoritmo de simulación para modelo PARAFAC

Datos de entrada: $\{I, J, K, R\}, \{\alpha_r\}_{r=1}^{R}, \{\beta_r\}_{r=1}^{R}, \{\gamma_r\}_{r=1}^{R}$

Inicio

$\boldsymbol{A_{temp}} \leftarrow CLD(sx_1, \dots, sx_I;\ \{\alpha_r\}_{r=1}^{R})$

$\boldsymbol{A} \leftarrow GS(\boldsymbol{A_{temp}})$

$\boldsymbol{B_{temp}} \leftarrow CLD(vx_1, \dots, vx_J;\ \{\beta_r\}_{r=1}^{R})$

$\boldsymbol{B} \leftarrow GS(\boldsymbol{B_{temp}})$

$\boldsymbol{C_{temp}} \leftarrow CLD(tx_1, \dots, tx_K;\ \{\gamma_r\}_{r=1}^{R})$

$\boldsymbol{C} \leftarrow GS(\boldsymbol{C_{temp}})$

$\boldsymbol{X_A} \leftarrow \boldsymbol{A}(\boldsymbol{C} \odot \boldsymbol{B})^T$

Fin

Datos de salida: $\boldsymbol{X_A}$

Cabe mencionar que el algoritmo de simulación para el modelo PARAFAC genera tablas de 3 vías de forma aleatoria. En la **Tabla 5.2.1.1** se resume un estudio computacional con 50 ejecuciones:

Tabla 5.2.1.1: Estudio de simulación con el modelo PARAFAC

EXPERIMENTO	$\boldsymbol{\varphi_1}(\{15, 20, 10, 3\}, \{5, 6, 4\}, \{7, 8, 5\}, \{2, 5, 3\})$
Número de ejecuciones	50
Fit más bajo con ParafacALS	97.86%
Fit más alto con ParafacALS	99.57%
Fit más bajo con CBPSO-ParafacALS, $\boldsymbol{A}$ ortogonal disjunta	93.44%
Fit más alto con CBPSO-ParafacALS, $\boldsymbol{A}$ ortogonal disjunta	96.25%
Fit más bajo con CBPSO-ParafacALS, $\boldsymbol{B}$ ortogonal disjunta	93.38%
Fit más alto con CBPSO-ParafacALS, $\boldsymbol{B}$ ortogonal disjunta	95.42%
Fit más bajo con CBPSO-ParafacALS, $\boldsymbol{C}$ ortogonal disjunta	92.91%
Fit más alto con CBPSO-ParafacALS, $\boldsymbol{C}$ ortogonal disjunta	96.69%

El algoritmo de simulación para el modelo PARAFAC ejecutó la función $\boldsymbol{\varphi_1}(\{15, 20, 10, 3\}, \{5, 6, 4\}, \{7, 8, 5\}, \{2, 5, 3\})$ 50 veces y se obtuvieron 50 tablas distintas de 3 vías $\underline{\mathbf{X}}$ de tamaño $15 \times 20 \times 10$. En todas las ejecuciones el algoritmo CBPSO-ParafacALS pudo detectar la estructura disjunta en los datos. Se esperaban valores altos de "fit" (con componentes clásicos y con componentes disjuntos) debido a que los datos simulan una estructura disjunta basada en el modelo PARAFAC.

A continuación se van a mostrar los resultados de una de las ejecuciones. Al ejecutar el algoritmo ParafacALS con los datos simulados se obtuvo un fit del 99.31% en 12 iteraciones. En la **Tabla 5.2.1.2** se muestra la matriz de correlación de componentes $\boldsymbol{\Psi}$, por lo que se puede afirmar que no está presente el problema de degeneración del modelo PARAFAC en los datos.

Tabla 5.2.1.2: Matriz Ψ de correlación de componentes con datos simulados

	COMP1	COMP2	COMP3
COMP1	1	0.01936312	0.02429681
COMP2	0.01936312	1	0.01031314
COMP3	0.02429681	0.01031314	1

Posteriormente se ejecutó el algoritmo CBPSO-ParafacALS en 3 escenarios. En el primero de ellos se obtuvo un fit del 95.70% en el cual sólo la matriz de cargas $\boldsymbol{A}$ es ortogonal disjunta, en el segundo escenario se obtuvo un fit del 93.71% en el cual únicamente la matriz de cargas $\boldsymbol{B}$ es ortogonal disjunta y finalmente en el último y tercer escenario de ejecución se obtuvo un fit del 95.76% en el que sólo la matriz de cargas $\boldsymbol{C}$ es ortogonal disjunta. Estos resultados computacionales se resumen en la **Tabla 5.2.1.3** que se muestra a continuación:

Tabla 5.2.1.3: Comparación del fit en escenarios de ejecución

ESCENARIOS DE EJECUCIÓN	FIT (%)
ParafacALS	99.31
CBPSO-ParafacALS con $\boldsymbol{A}$ ortogonal disjunta	95.70
CBPSO-ParafacALS con $\boldsymbol{B}$ ortogonal disjunta	93.71
CBPSO-ParafacALS con $\boldsymbol{C}$ ortogonal disjunta	95.76

La **Tabla 5.2.1.4** muestra la matriz de cargas $\boldsymbol{A}$ ortogonal disjunta y se observa que efectivamente el algoritmo CBPSO-ParafacALS pudo detectar la estructura disjunta en el primer modo.

Tabla 5.2.1.4: Matriz de cargas A ortogonal disjunta con el modelo PARAFAC

	sy_1	sy_2	sy_3
sx_1	-0.39987415	0	0
sx_2	-0.38886439	0	0
sx_3	-0.46369775	0	0
sx_4	-0.47750129	0	0
sx_5	-0.49584480	0	0
sx_6	0	0.45327357	0
sx_7	0	0.40688744	0
sx_8	0	0.37939130	0
sx_9	0	0.34345457	0
sx_{10}	0	0.41482141	0
sx_{11}	0	0.44159947	0
sx_{12}	0	0	-0.54088342
sx_{13}	0	0	-0.41894439
sx_{14}	0	0	-0.53459196
sx_{15}	0	0	-0.49612716

La **Tabla 5.2.1.5** muestra la matriz de cargas $\boldsymbol{B}$ ortogonal disjunta y se observa que efectivamente el algoritmo CBPSO-ParafacALS pudo detectar la estructura disjunta en el segundo modo.

Tabla 5.2.1.5: Matriz de cargas $\boldsymbol{B}$ ortogonal disjunta con el modelo PARAFAC

	vy_1	vy_2	vy_3
vx_1	-0.40287714	0	0
vx_2	-0.38117182	0	0
vx_3	-0.38769074	0	0
vx_4	-0.34328899	0	0
vx_5	-0.31499905	0	0
vx_6	-0.34929550	0	0
vx_7	-0.45057171	0	0
vx_8	0	-0.39457824	0
vx_9	0	-0.33906833	0
vx_{10}	0	-0.37809945	0
vx_{11}	0	-0.30556330	0
vx_{12}	0	-0.32902385	0
vx_{13}	0	-0.32439850	0
vx_{14}	0	-0.33306471	0
vx_{15}	0	-0.41059636	0
vx_{16}	0	0	-0.46109260
vx_{17}	0	0	-0.43821079
vx_{18}	0	0	-0.41637465
vx_{19}	0	0	-0.38723790
vx_{20}	0	0	-0.52157826

Finalmente la **Tabla 5.2.1.6** muestra la matriz de cargas $\boldsymbol{C}$ ortogonal disjunta y se observa que efectivamente el algoritmo CBPSO-ParafacALS pudo detectar la estructura disjunta en el tercer modo.

Tabla 5.2.1.6: Matriz de cargas $\boldsymbol{C}$ ortogonal disjunta con el modelo PARAFAC

	$\boldsymbol{ty_1}$	$\boldsymbol{ty_2}$	$\boldsymbol{ty_3}$
$\boldsymbol{tx_1}$	-0.72171067	0	0
$\boldsymbol{tx_2}$	-0.69219484	0	0
$\boldsymbol{tx_3}$	0	-0.43505908	0
$\boldsymbol{tx_4}$	0	-0.39485340	0
$\boldsymbol{tx_5}$	0	-0.49233526	0
$\boldsymbol{tx_6}$	0	-0.42701150	0
$\boldsymbol{tx_7}$	0	-0.47966818	0
$\boldsymbol{tx_8}$	0	0	-0.59150295
$\boldsymbol{tx_9}$	0	0	-0.53266172
$\boldsymbol{tx_{10}}$	0	0	-0.60530633

Tal como se observa en las 3 tablas anteriores, la estructura disjunta facilita la interpretación. Este beneficio tiene como costo una pérdida en el fit del modelo (ver **Tabla 5.2.1.3**). Con los mismos datos simulados de 3 vías se realizó un experimento adicional, se ejecutó el algoritmo CBPSO-ParafacALS y se impuso la restricción de que las matrices de cargas $\boldsymbol{A}$ y $\boldsymbol{B}$ sean ortogonales disjuntas (esto como una ilustración de un experimento computacional con 2 matrices ortogonales disjuntas). Se obtuvo un fit del 87.41%. Es decir, al tener 2 matrices ortogonales disjuntas se tiene una mayor pérdida de fit en el modelo. La recomendación es minimizar el número de matrices ortogonales disjuntas.

Con este primer algoritmo de simulación se realizó un estudio de simulación con 50 ejecuciones. Este que se ha presentado detalladamente es uno de ellos y fue seleccionado aleatoriamente, pues en todos los casos el algoritmo CBPSO-ParafacALS pudo detectar la estructura disjunta en los datos. En conclusión, si existe una estructura disjunta en los datos (podría no estar presente) según el modelo PARAFAC, el algoritmo denominado CBPSO-ParafacALS tiene la capacidad de detectarla. Esta afirmación se apoya en las pruebas efectuadas. Una recomendación importante, producto de la experimentación, es que el número de matrices de cargas ortogonales disjuntas en el modelo PARAFAC sea de una o dos.

Al ejecutar el algoritmo ParafacALS las 3 matrices de cargas que se obtienen permiten comprobar que el algoritmo CBPSO-ParafacALS pudo detectar la estructura disjunta en los datos. En las 3 tablas que se presentan a continuación se muestran las matrices de cargas obtenidas por el algoritmo ParafacALS. Se han resaltado en negrita los valores más altos (en valor absoluto) por cada fila. Estos valores coinciden en interpretación con los valores no nulos de las respectivas matrices de cargas ortogonales disjuntas que se obtuvieron con el algoritmo CBPSO-ParafacALS.

Tabla 5.2.1.7: Matriz de cargas A obtenida con el algoritmo ParafacALS

	sy_1	sy_2	sy_3
sx_1	**0.39058407**	0.04220063	0.00624486
sx_2	**0.38497927**	0.00589775	-0.01409814
sx_3	**0.45314340**	0.04101933	-0.00084005
sx_4	**0.44438271**	0.05919143	-0.11318175
sx_5	**0.46616460**	0.10905944	-0.02663146
sx_6	0.07803826	**0.42911772**	-0.06213948
sx_7	0.01701127	**0.40785304**	-0.01877318
sx_8	0.12564583	**0.34031041**	-0.06153307
sx_9	0.02234233	**0.33679831**	-0.04125205
sx_{10}	0.14002076	**0.37172543**	-0.06519013
sx_{11}	0.03076091	**0.43973833**	-0.01412528
sx_{12}	0.06424171	0.08655876	**-0.51474080**
sx_{13}	0.02448708	0.03291099	**-0.40988417**
sx_{14}	0.07820388	0.00102732	**-0.51853307**
sx_{15}	0.10232589	0.02237328	**-0.47036366**

Tabla 5.2.1.8: Matriz de cargas B obtenida con el algoritmo ParafacALS

	vy_1	vy_2	vy_3
vx_1	**-0.38577448**	-0.09573379	-0.01340660
vx_2	**-0.36550301**	0.00461886	-0.11510760
vx_3	**-0.37281420**	-0.08252101	-0.01518721
vx_4	**-0.32972661**	-0.02205547	-0.06949772
vx_5	**-0.30739516**	-0.05760671	-0.00157004
vx_6	**-0.32919372**	-0.02982258	-0.09117871
vx_7	**-0.41583329**	-0.07709632	-0.11611009
vx_8	-0.05106728	**-0.39380244**	-0.05713416
vx_9	-0.11703646	**-0.32543151**	-0.00454256
vx_{10}	-0.10630324	**-0.36110697**	-0.04900409
vx_{11}	-0.04127823	**-0.30387530**	-0.04698212
vx_{12}	-0.03871558	**-0.32102351**	-0.09184524
vx_{13}	-0.03489164	**-0.32456782**	-0.05552073
vx_{14}	-0.04225277	**-0.33563141**	-0.03368986
vx_{15}	-0.06979258	**-0.40168848**	-0.07437269
vx_{16}	-0.07078440	-0.01769632	**-0.45544347**
vx_{17}	-0.12060576	-0.00032693	**-0.41942883**
vx_{18}	-0.13227102	-0.00982828	**-0.39210433**
vx_{19}	-0.12308955	-0.02046642	**-0.36322682**
vx_{20}	-0.05668153	-0.08006329	**-0.51439813**

Tabla 5.2.1.9: Matriz de cargas C obtenida con el algoritmo ParafacALS

	ty_1	ty_2	ty_3
tx_1	**-0.69410545**	-0.07895547	0.07407109
tx_2	**-0.67420621**	-0.07206525	0.02160259
tx_3	-0.03657895	**-0.44774093**	0.02929262
tx_4	-0.06956667	**-0.37817806**	0.10534648
tx_5	-0.03052856	**-0.49968461**	0.07643172
tx_6	-0.18447609	**-0.39820966**	0.02110541
tx_7	-0.04107722	**-0.47430464**	0.11684362
tx_8	-0.11594920	-0.00759120	**0.57844005**
tx_9	-0.06716494	-0.04098604	**0.52454901**
tx_{10}	-0.05366541	-0.10061829	**0.59363847**

5.2.2 APLICANDO EL ALGORITMO CBPSO-TuckALS3 A DATOS SIMULADOS CON UNA ESTRUCTURA DISJUNTA PARA EL MODELO TUCKER3

Sea $\underline{\mathbf{X}}$ una tabla de 3 vías con I individuos, J variables y K situaciones o tiempos. Suponga que el primer modo (relacionado con la matriz de cargas $\boldsymbol{A}$) tiene P individuos latentes ($P < I$), el segundo modo (relacionado con la matriz de cargas $\boldsymbol{B}$) tiene Q variables latentes ($Q < J$) y, finalmente, el tercer modo (relacionado con la matriz de cargas $\boldsymbol{C}$) tiene R situaciones o tiempos latentes ($R < K$).

Suponga que $sx_1, \dots, sx_I$ son los I individuos originales. Por otro lado $sy_1, \dots, sy_P$ son los P individuos latentes. Considere la combinación lineal:

$$sy_p = a_{1,p}sx_1 + \cdots + a_{I,p}sx_I;\ p = 1,2,3, \dots, P$$

Si se desea que los m individuos originales consecutivos sx_n, sx_{n+1}, … , $sx_{n+(m-1)}$ estén representados por el individuo latente sy_p entonces los correspondientes escalares $a_{n,r}$, $a_{n+1,r}$, … , $a_{n+(m-1),r}$ se definen como variables aleatorias uniformes discretas independientes cuyo conjunto soporte son los números enteros del 70 al 100 incluidos. Los demás escalares de esa misma combinación lineal se definen como variables aleatorias uniformes discretas independientes cuyo conjunto soporte son los números enteros del 1 al 30 incluidos. Este procedimiento se debe realizar para cada p desde 1 hasta P. Al hacerlo, tener presente que cada individuo original debe tener una fuerte presencia en un único individuo latente. A la matriz de tamaño $I \times P$ que contiene los escalares de todas las combinaciones lineales se le aplica el proceso de ortonormalización vectorial de Gram-Schmidt para poder construir una matriz ortogonal. De esta manera se alcanza una reducción dimensional disjunta en la matriz de cargas $\boldsymbol{A}$. Este proceso de dos etapas se lo va a representar con las instrucciones:

$$\boldsymbol{A_{temp}} \leftarrow CLD\ (sx_1, \dots, sx_I;\ \{\alpha_p\}_{p=1}^{P})$$

$$\boldsymbol{A} \leftarrow GS(\boldsymbol{A_{temp}})$$

Para la matriz de cargas $\boldsymbol{B}$ se trabaja de la misma manera. Considere que vx_1, … , vx_J son las J variables originales. Además vy_1, … , vy_Q son las Q variables latentes. Sea la combinación lineal:

$$vy_q = b_{1,q}vx_1 + \cdots + b_{J,q}vx_J;\ q = 1,2,3, \dots, Q$$

Para que las m variables originales consecutivas vx_n, vx_{n+1}, … , $vx_{n+(m-1)}$ estén representadas por la variable latente vy_q, entonces los correspondientes escalares $b_{n,r}$, $b_{n+1,r}$, … , $b_{n+(m-1),r}$ se definen como variables aleatorias uniformes discretas independientes cuyo conjunto soporte son los números enteros del 70 al 100 incluidos. Los demás escalares de esa misma combinación lineal se definen como variables aleatorias uniformes discretas independientes cuyo conjunto soporte son los números enteros del 1 al 30 incluidos. Este procedimiento se debe realizar para cada q desde 1 hasta Q. Al hacerlo, tener presente que cada variable original debe tener una fuerte presencia en una única variable latente. A la matriz de tamaño $J \times Q$ que contiene los escalares de todas las combinaciones lineales se le aplica el conocido proceso de ortonormalización vectorial de Gram-Schmidt para poder construir una matriz ortogonal. De esta manera se alcanza una reducción dimensional disjunta en la matriz de cargas $\boldsymbol{B}$. Este proceso de dos etapas se lo va a representar con las instrucciones:

$$\boldsymbol{B_{temp}} \leftarrow CLD\ (vx_1, \dots, vx_J;\ \{\beta_q\}_{q=1}^{Q})$$

$$\boldsymbol{B} \leftarrow GS(\boldsymbol{B_{temp}})$$

Para finalizar se procede con la matriz de cargas $\boldsymbol{C}$ tal como se hizo con las 2 matrices anteriores. Sean $tx_1, \dots, tx_K$ las K situaciones o tiempos originales. Sean $ty_1, \dots, ty_R$ las R situaciones o tiempos latentes. Dada la combinación lineal:

$$ty_r = c_{1,r}tx_1 + \cdots + c_{K,r}tx_K;\ r = 1,2,3, \dots, R$$

Para que las m situaciones o tiempos originales consecutivos $tx_n, tx_{n+1}, \dots, tx_{n+(m-1)}$ estén representados por la situación o tiempo latente ty_r, entonces los escalares $c_{n,r}$, $c_{n+1,r}$, … , $c_{n+(m-1),r}$ se definen como variables aleatorias uniformes discretas independientes cuyo conjunto soporte son los números enteros del 70 al 100 incluidos. Los demás escalares de esa misma combinación lineal se definen como variables aleatorias uniformes discretas independientes cuyo conjunto soporte son los números enteros del 1 al 30 incluidos. Este procedimiento se debe realizar para cada r desde 1 hasta R. Al hacerlo, tener presente que cada situación o tiempo original debe tener una fuerte presencia en una única situación o tiempo latente. A la matriz de tamaño $K \times R$ que contiene los escalares de todas las combinaciones lineales se le aplica el proceso de ortonormalización vectorial de Gram-Schmidt para poder construir una matriz ortogonal. De esta manera se alcanza una reducción dimensional disjunta en la matriz de cargas $\boldsymbol{C}$. Este proceso de dos etapas se lo va a representar con las instrucciones:

$$\boldsymbol{C_{temp}} \leftarrow CLD(tx_1, \dots, tx_K;\ \{\gamma_r\}_{r=1}^{R})$$

$$\boldsymbol{C} \leftarrow GS(\boldsymbol{C_{temp}})$$

El core $\underline{\mathbf{G}}$ es de tamaño $P \times Q \times R$. Sin pérdida de generalidad supondremos que las entradas de dicha matriz de 3 vías son variables aleatorias independientes con distribución uniforme continua en el intervalo $[-50,\ 50]$. Esta tarea se la va a representar con la instrucción $\boldsymbol{G_A} \leftarrow \mathbf{U}\,[-50, 50]$, donde la supermatriz $\boldsymbol{G_A}$ de tamaño $P \times QR$ contiene los cortes frontales de $\underline{\mathbf{G}}$.

Para completar la construcción de $\underline{\mathbf{X}}$ se usa la ecuación matricial:

$$\boldsymbol{X_A} = \boldsymbol{A G_A (C \otimes B)^T}$$

La matriz $\boldsymbol{X_A}$ de tamaño $I \times JK$ contiene los cortes frontales de $\underline{\mathbf{X}}$. El **Algoritmo 5.2.2** resume este segundo algoritmo de simulación:

Algoritmo 5.2.2: Algoritmo de simulación para modelo Tucker3

Datos de entrada: $\{I, J, K, P, Q, R\}, \{\alpha_p\}_{p=1}^{P}, \{\beta_q\}_{q=1}^{Q}, \{\gamma_r\}_{r=1}^{R}$

Inicio

$\boldsymbol{A_{temp}} \leftarrow CLD\ (sx_1, \ldots, sx_I;\ \{\alpha_p\}_{p=1}^{P})$

$\boldsymbol{A} \leftarrow GS(\boldsymbol{A_{temp}})$

$\boldsymbol{B_{temp}} \leftarrow CLD\ (vx_1, \ldots, vx_J;\ \{\beta_q\}_{q=1}^{Q})$

$\boldsymbol{B} \leftarrow GS(\boldsymbol{B_{temp}})$

$\boldsymbol{C_{temp}} \leftarrow CLD(tx_1, \ldots, tx_K;\ \{\gamma_r\}_{r=1}^{R})$

$\boldsymbol{C} \leftarrow GS(\boldsymbol{C_{temp}})$

$\boldsymbol{G_A} \leftarrow \mathbf{U}\,[-50, 50]$

$\boldsymbol{X_A} \leftarrow \boldsymbol{A G_A (C \otimes B)^T}$

Fin

Datos de salida: $\boldsymbol{X_A}$

Cabe mencionar que el algoritmo de simulación para el modelo Tucker3 genera tablas de 3 vías de forma aleatoria. En la **Tabla 5.2.2.1** se resume un estudio computacional con 50 ejecuciones:

Tabla 5.2.2.1: Estudio de simulación con el modelo Tucker3

EXPERIMENTO	$\boldsymbol{\varphi_2}(\{20,18,17,3,4,5\},\{5,7,8\},\{3,4,5,6\},\{2,3,3,4,5\})$
Número de ejecuciones	50
Fit más bajo con TuckALS3	93.83%
Fit más alto con TuckALS3	96.41%
Fit más bajo con CBPSO-TuckALS3 y las 3 matrices de cargas ortogonales disjuntas	91.62%
Fit más alto con CBPSO-TuckALS3 y las 3 matrices de cargas ortogonales disjuntas	93.15%

El algoritmo de simulación para el modelo Tucker3 ejecutó la función $\boldsymbol{\varphi_2}(\{20,18,17,3,4,5\},\{5,7,8\},\{3,4,5,6\},\{2,3,3,4,5\})$ 50 veces y se obtuvieron 50 tablas diferentes de 3 vías $\underline{\mathbf{X}}$ de tamaño $20 \times 18 \times 17$. En todas las ejecuciones el algoritmo CBPSO-TuckALS3 pudo detectar la estructura disjunta en los datos. Se esperaban valores altos de "fit" (con componentes clásicos y con componentes disjuntos) debido a que los datos simulan una estructura disjunta basada en el modelo Tucker3.

A continuación se van a mostrar los resultados de una de las ejecuciones. Al ejecutar el algoritmo TuckALS3 con los datos simulados se obtuvo un fit del 94.73%. Con los mismos datos posteriormente se ejecutó el algoritmo CBPSO-TuckALS3 y se obtuvo un fit del 92.79%. En este último se impuso la restricción de que las 3 matrices de cargas sean ortogonales disjuntas.

Observe en la **Tabla 5.2.2.2** el fit del modelo obtenido en cada escenario usando la misma matriz de 3 vías generada por el algoritmo de simulación.

Tabla 5.2.2.2: Comparación del fit en escenarios de ejecución

ESCENARIOS DE EJECUCIÓN	FIT (%)
TuckALS3	94.73
CBPSO-TuckALS3 con $\boldsymbol{A}$, $\boldsymbol{B}$ y $\boldsymbol{C}$ ortogonales disjuntas	92.79

Como se esperaba se pierde fit en el modelo, pero se gana en interpretación debido a que se obtiene una estructura simple en las 3 matrices de cargas. La **Tabla 5.2.2.3** muestra la matriz de cargas ortogonal disjunta $\boldsymbol{A}$. Se observa que el algoritmo heurístico CBPSO-TuckALS3 es capaz de identificar la estructura disjunta en el primer modo.

Tabla 5.2.2.3: Matriz de cargas A ortogonal disjunta con el modelo Tucker3

	sy_1	sy_2	sy_3
sx_1	0.48932151	0	0
sx_2	0.44138634	0	0
sx_3	0.38687825	0	0
sx_4	0.40775562	0	0
sx_5	0.49980310	0	0
sx_6	0	0.39376793	0
sx_7	0	0.36369995	0
sx_8	0	0.39165864	0
sx_9	0	0.37564528	0
sx_{10}	0	0.43001693	0
sx_{11}	0	0.31146907	0
sx_{12}	0	0.36910128	0
sx_{13}	0	0	-0.32559414
sx_{14}	0	0	-0.33366963
sx_{15}	0	0	-0.31117803
sx_{16}	0	0	-0.36099787
sx_{17}	0	0	-0.34156470
sx_{18}	0	0	-0.38518722
sx_{19}	0	0	-0.37839893
sx_{20}	0	0	-0.38377130

Se tienen 5 individuos originales representados por el primer individuo latente. Los 7 siguientes individuos originales representados por el segundo individuo latente y, finalmente, los últimos 8 individuos originales representados por el tercer individuo latente del modelo.

En la **Tabla 5.2.2.4** se muestra la matriz de cargas ortogonal disjunta $\boldsymbol{B}$. Se resalta también el hecho de que el algoritmo CBPSO-TuckALS3 es capaz de identificar la estructura disjunta en el segundo modo. El algoritmo CBPSO-TuckALS3 logra reconocer la forma en que las variables latentes agrupan a las variables originales.

Tabla 5.2.2.4: Matriz de cargas B ortogonal disjunta con el modelo Tucker3

	vy_1	vy_2	vy_3	vy_4
vx_1	-0.63185677	0	0	0
vx_2	-0.53495062	0	0	0
vx_3	-0.56087864	0	0	0
vx_4	0	-0.54058450	0	0
vx_5	0	-0.51277877	0	0
vx_6	0	-0.54593785	0	0
vx_7	0	-0.38311642	0	0
vx_8	0	0	-0.43631239	0
vx_9	0	0	-0.46432965	0
vx_{10}	0	0	-0.45681443	0
vx_{11}	0	0	-0.45592570	0
vx_{12}	0	0	-0.42128590	0
vx_{13}	0	0	0	0.48405851
vx_{14}	0	0	0	0.47246029
vx_{15}	0	0	0	0.36140660
vx_{16}	0	0	0	0.40851941
vx_{17}	0	0	0	0.35273551
vx_{18}	0	0	0	0.34719369

Se tienen 3 variables originales representadas en la primera variable latente. Las 4 siguientes variables originales representadas por la segunda variable latente, las 5 siguientes variables originales representadas por la tercera variable latente y, finalmente, las últimas 6 variables originales representadas por la cuarta variable latente del modelo.

Finalmente, en la **Tabla 5.2.2.5** se muestra la matriz de cargas ortogonal disjunta $\boldsymbol{C}$. Se observa nuevamente que el algoritmo CBPSO-TuckALS3 logra identificar la estructura disjunta en el tercer modo.

Tabla 5.2.2.5: Matriz de cargas C ortogonal disjunta con el modelo Tucker3

	ty_1	ty_2	ty_3	ty_4	ty_5
tx_1	0.56826144	0	0	0	0
tx_2	0.82284806	0	0	0	0
tx_3	0	0.50066252	0	0	0
tx_4	0	0.57846089	0	0	0
tx_5	0	0.64398760	0	0	0
tx_6	0	0	0.62935973	0	0
tx_7	0	0	0.65899658	0	0
tx_8	0	0	0.41186143	0	0
tx_9	0	0	0	0.40455206	0
tx_{10}	0	0	0	0.52191470	0
tx_{11}	0	0	0	0.47595483	0
tx_{12}	0	0	0	0.58086976	0
tx_{13}	0	0	0	0	0.50933537
tx_{14}	0	0	0	0	0.44032157
tx_{15}	0	0	0	0	0.34018158
tx_{16}	0	0	0	0	0.36337834
tx_{17}	0	0	0	0	0.54674224

El algoritmo CBPSO-TuckALS3 también pudo reconocer la manera en la cual los tiempos o situaciones latentes agrupan a los tiempos o situaciones originales.

Las 3 matrices de cargas son de fácil interpretación debido a que tienen una estructura simple. Al ejecutar el algoritmo TuckALS3 las 3 matrices de cargas que se obtienen no permiten realizar una interpretación al analista de los datos ya que no presentan una estructura simple. Si bien existe la posibilidad de realizar rotaciones o aplicar técnicas "sparse" para mejorar la interpretación, la técnica que se propone en esta tesis respecto de calcular componentes ortogonales disjuntos en las matrices de cargas, es una opción importante también a considerar para el investigador que va a realizar la interpretación. Con este segundo algoritmo de simulación se realizó un estudio computacional con 50 ejecuciones. Este que se ha presentado en detalle es uno de ellos y fue seleccionado aleatoriamente ya que en todas las ejecuciones el algoritmo CBPSO-TuckALS3 pudo detectar la estructura disjunta en los datos. En conclusión, si existe una estructura disjunta en los datos (podría no estar presente) según el modelo Tucker3, el algoritmo denominado CBPSO-TuckALS3 tiene la capacidad de detectarla. Esta afirmación se apoya en las pruebas realizadas.

Al ejecutar el algoritmo TuckALS3 las 3 matrices de cargas obtenidas no tienen una estructura simple, por lo que no son interpretables. En las tablas que se presentan a continuación se muestran las matrices de cargas que se obtienen con el algoritmo TuckALS3.

Tabla 5.2.2.6: Matriz de cargas A obtenida con el algoritmo TuckALS3

	sy_1	sy_2	sy_3
sx_1	-0.25550639	0.33207905	-0.29920712
sx_2	-0.24112058	0.25032068	-0.23392021
sx_3	-0.20452465	0.26228010	-0.20409113
sx_4	-0.21407607	0.28240369	-0.22212734
sx_5	-0.26827842	0.32782346	-0.23357305
sx_6	-0.25469368	0.04778989	0.28235193
sx_7	-0.23111660	0.03698121	0.30874972
sx_8	-0.25363293	0.03793761	0.28031540
sx_9	-0.24360226	0.02581209	0.26849677
sx_{10}	-0.27705587	0.04592063	0.32239751
sx_{11}	-0.19825172	0.02811158	0.26186236
sx_{12}	-0.23808287	0.03579710	0.27473801
sx_{13}	-0.17728504	-0.25072929	-0.12853757
sx_{14}	-0.18268289	-0.25224853	-0.12829981
sx_{15}	-0.16988024	-0.23715544	-0.12302062
sx_{16}	-0.19450684	-0.28929989	-0.14192651
sx_{17}	-0.19183855	-0.23674038	-0.10819945
sx_{18}	-0.21118999	-0.28865109	-0.15200623
sx_{19}	-0.21446330	-0.25451052	-0.11049715
sx_{20}	-0.20820552	-0.30090205	-0.14565185

Tabla 5.2.2.7: Matriz de cargas B obtenida con el algoritmo TuckALS3

	vy_1	vy_2	vy_3	vy_4
vx_1	0.25416883	-0.30708947	-0.28653465	0.38729181
vx_2	0.21504909	-0.23627929	-0.28672351	0.33938693
vx_3	0.22693048	-0.25470217	-0.29983328	0.29874680
vx_4	0.23536331	-0.38614901	0.16325123	-0.24628101
vx_5	0.22936645	-0.32817755	0.19215660	-0.25495229
vx_6	0.24450855	-0.34009967	0.23881762	-0.26270040
vx_7	0.17537379	-0.22669159	0.16162324	-0.17667930
vx_8	0.22567460	0.16807404	-0.22858586	-0.24492592
vx_9	0.23628651	0.20342888	-0.28078027	-0.19185361
vx_{10}	0.23787730	0.16887076	-0.23316352	-0.25241859
vx_{11}	0.23967951	0.13990504	-0.28194359	-0.18762750
vx_{12}	0.21251971	0.20801905	-0.19579082	-0.26850673
vx_{13}	0.29545744	0.21077663	0.27507818	0.15303647
vx_{14}	0.28584792	0.20285861	0.29370070	0.19533530
vx_{15}	0.21442157	0.18223619	0.22482264	0.17798954
vx_{16}	0.25677389	0.20298517	0.07402967	0.13925084
vx_{17}	0.21649632	0.15931267	0.16176850	0.14173487
vx_{18}	0.21299839	0.12148956	0.22727338	0.13835590

Tabla 5.2.2.8: Matriz de cargas C obtenida con el algoritmo TuckALS3

	ty_1	ty_2	ty_3	ty_4	ty_5
tx_1	-0.26188615	0.30791207	-0.16883523	0.21968121	-0.34408797
tx_2	-0.36811534	0.50443581	-0.19452124	0.47601581	0.00596657
tx_3	-0.24044381	-0.01034069	-0.29546639	-0.27418556	0.14166011
tx_4	-0.28032485	0.10223534	-0.26873318	-0.31408828	0.20807680
tx_5	-0.30679112	0.03790557	-0.33556683	-0.36207103	0.32407194
tx_6	-0.27393547	0.17286749	0.47588168	-0.24376720	-0.15980802
tx_7	-0.29314840	0.20848408	0.39661789	-0.27536860	-0.25241240
tx_8	-0.19711092	0.01549482	0.19468211	-0.15678510	-0.14868881
tx_9	-0.15233754	-0.11830512	0.13882409	0.26057808	0.28163088
tx_{10}	-0.21257247	-0.13173771	0.17534744	0.18963260	0.29370801
tx_{11}	-0.18057502	-0.10702518	0.30185822	0.12863466	0.37298690
tx_{12}	-0.24022042	-0.13135585	0.18042097	0.27507367	0.25378392
tx_{13}	-0.22850244	-0.34015395	-0.10920221	0.12810094	-0.34264273
tx_{14}	-0.18907134	-0.39502750	-0.03636739	-0.03999028	-0.18866260
tx_{15}	-0.16215750	-0.24200366	0.02618379	-0.05649640	-0.00594387
tx_{16}	-0.17779067	-0.22911924	0.03963155	-0.09458493	-0.03598895
tx_{17}	-0.24695845	-0.34968194	-0.22509993	0.18485547	-0.27997985

5.3 APLICACIÓN CON DATOS NO SIMULADOS

En esta sección se reportan los experimentos computacionales que usan datos que no han sido simulados. Son 3 ejemplos muy conocidos, tomados de la literatura científica, que se indican a continuación:

- **Experimento # 1:** Se trabaja con una tabla de 3 vías $\underline{\mathbf{X}}$ de tamaño $30 \times 16 \times 15$. Para este primer experimento se ha escogido el ejemplo de los datos de TV (Lundy, Harshman and Kruskal 1989) donde $I = 30$ estudiantes universitarios clasifican $K = 15$ programas estadounidenses empleando $J = 16$ escalas bipolares. A estos datos se les aplica una reducción dimensional según el modelo PARAFAC. Tal como lo hacen Giordani, Kiers and Del Ferraro 2014 el primer modo corresponde a los estudiantes, el segundo modo a las escalas bipolares y el tercer modo a los diferentes programas de TV. Se escogió además un modelo PARAFAC con $R = 3$ componentes como en Lundy, Harshman and Kruskal 1989. Se usan los algoritmos ParafacALS y CBPSO-ParafacALS.

- **Experimento # 2:** Se trabaja con una tabla de 3 vías $\underline{\mathbf{X}}$ de tamaño $6 \times 4 \times 4$. Para este segundo experimento se ha escogido el ejemplo ilustrativo de los niveles de comportamiento (Amaya and Pacheco 2002, Kiers and Van Mechelen 2001) donde $I = 6$ individuos son evaluados mediante $J = 4$ variables: emocionalidad, sensibilidad, concentrabilidad y racionalidad en $K = 4$ situaciones diferentes: haciendo un examen, dando un discurso, compartiendo en un picnic familiar y compartiendo en una cita romántica. A estos datos se les aplica una reducción dimensional según el modelo Tucker3. El primer modo corresponde a los sujetos, el segundo modo a las variables y el tercer modo a las situaciones. Se escoge un modelo Tucker3 con $P = Q = R = 2$ componentes tal como en el artículo de Amaya and Pacheco 2002. Se usan los algoritmos TuckALS3 y CBPSO-TuckALS3.

- **Experimento # 3:** Se trabaja con una tabla de 3 vías $\underline{\mathbf{X}}$ de tamaño $24 \times 20 \times 38$. Para este tercer experimento se ha escogido el ejemplo de los preludios de Chopin (Murakami and Kroonenberg 2003) donde $K = 38$ estudiantes japoneses universitarios (21 hombres y 17 mujeres) evalúan $I = 24$ preludios de Chopin usando $J = 20$ escalas bipolares. A estos datos se les aplica una reducción dimensional según el modelo Tucker3. En el primer modo se encuentran los preludios de Chopin, en el segundo modo tenemos las escalas bipolares y en el tercer modo se ubican los estudiantes que realizan la evaluación. Tal como en el artículo de Murakami and Kroonenberg 2003 se escoge un modelo Tucker3 con $P = 2$ componentes, $Q = 3$ componentes y $R = 2$ componentes. Se usan los algoritmos TuckALS3 y CBPSO-TuckALS3.

5.3.1 APLICANDO EL ALGORITMO CBPSO-ParafacALS A DATOS REALES RELACIONADOS A PROGRAMAS DE TV

Esta sección está dedicada al **Experimento # 1**. Como parte del preprocesamiento, la tabla de 3 vías es centrada primero respecto de las escalas (segundo modo) y luego es centrada respecto de los programas de TV (tercer modo). Finalmente, se aplica una normalización respecto de las escalas bipolares, tal como en Giordani, Kiers and Del Ferraro 2014. Los datos de este experimento se pueden tomar de un paquete del software R llamado "ThreeWay".

Se ejecutó el algoritmo ParafacALS con $R = 3$ componentes y se obtuvo un fit del 47.93% después de más de 7 mil iteraciones. En general, el valor del fit depende de los datos y su contexto (si se trata de un estudio de datos económicos, datos psicológicos o datos ambientales). En Kroonenberg 2008 se afirma, por ejemplo, que en psicometría un valor aceptable para el fit debe ser al menos del 40%. En la **Tabla 5.3.1.1** se muestra la matriz $\boldsymbol{\Psi}$ de correlación de componentes:

Tabla 5.3.1.1: Matriz Ψ de correlación de componentes en Experimento # 1

	COMP1	COMP2	COMP3
COMP1	1	0.05652381	-0.97495543
COMP2	0.05652381	1	-0.11506567
COMP3	-0.97495543	-0.11506567	1

Se sospecha la presencia del problema de degeneración del modelo PARAFAC debido al alto número de iteraciones que se necesitaron (7231 iteraciones) para alcanzar el criterio de parada del algoritmo ParafacALS. Además la matriz $\boldsymbol{\Psi}$ de este primer escenario de ejecución nos dice que el primer componente y el tercer componente tienen una alta correlación lineal inversa (ver **Sección 2.2.2** para más detalles).

Para corregir el problema de degeneración (ver Giordani, Kiers and Del Ferraro 2014) se impuso la restricción de ortogonalidad en la matriz de cargas $\boldsymbol{B}$ y se obtuvo un fit del 47.27% en 239 iteraciones. La matriz de correlación de componentes $\boldsymbol{\Psi}$ en este segundo escenario de ejecución se presenta en la **Tabla 5.3.1.2**:

Tabla 5.3.1.2: Matriz Ψ de correlación de componentes en Experimento # 1 con B ortogonal

	COMP1	COMP2	COMP3
COMP1	1	0	0
COMP2	0	1	0
COMP3	0	0	1

En esta última ejecución se decidió, con el propósito de ayudar a la interpretación, que las columnas de las matrices de cargas $\boldsymbol{B}$ y $\boldsymbol{C}$ sean vectores unitarios. Se compensó tal restricción en la matriz de cargas $\boldsymbol{A}$ (ver **Sección 2.2.1**).

Después de las 2 ejecuciones con el algoritmo ParafacALS, se ejecutó también el algoritmo CBPSO-ParafacALS con $R = 3$ componentes, en 2 escenarios distintos. En el primero de ellos sólo la matriz de cargas $\boldsymbol{B}$ tiene componentes ortogonales disjuntos, en el segundo escenario sólo la matriz de cargas $\boldsymbol{C}$ tiene componentes ortogonales disjuntos. El fit obtenido en todas las 4 ejecuciones se resume a continuación en la **Tabla 5.3.1.3** que se muestra:

Tabla 5.3.1.3: Comparación del fit en escenarios de ejecución del Experimento # 1

ESCENARIOS DE EJECUCIÓN	FIT (%)
ParafacALS con problema de degeneración	47.93
ParafacALS con $\boldsymbol{B}$ ortogonal	47.27
CBPSO-ParafacALS con $\boldsymbol{B}$ ortogonal disjunta	41.12
CBPSO-ParafacALS con $\boldsymbol{C}$ ortogonal disjunta	41.21

Se observa que la restricción de tener componentes ortogonales disjuntos afecta al fit, pero se gana en interpretación, como ilustraremos más adelante. Tal como se afirma en Lundy, Harshman and Kruskal 1989 los 3 componentes se pueden interpretar como "humor", "sensibilidad" y "violencia". Cabe indicar que en la matriz de cargas $\boldsymbol{B}$, la interpretación de un valor negativo en una entrada significa que el componente de esa columna está asociado con el lado izquierdo de la escala bipolar ubicada en esa fila.

La **Tabla 5.3.1.4** muestra el componente "humor" de la matriz de cargas $\boldsymbol{B}$ y la **Tabla 5.3.1.5** muestra el componente "humor" de la matriz de cargas $\boldsymbol{C}$, en los 3 últimos escenarios de ejecución de la **Tabla 5.3.1.3**.

Además, la **Tabla 5.3.1.6** muestra el componente "sensibilidad" de la matriz de cargas $\boldsymbol{B}$ y la **Tabla 5.3.1.7** muestra el componente "sensibilidad" de la matriz de cargas $\boldsymbol{C}$, en los 3 últimos escenarios de ejecución de la **Tabla 5.3.1.3**.

Finalmente, la **Tabla 5.3.1.8** muestra el componente "violencia" de la matriz de cargas $\boldsymbol{B}$ y la **Tabla 5.3.1.9** muestra el componente "violencia" de la matriz de cargas $\boldsymbol{C}$, en los 3 últimos escenarios de ejecución de la **Tabla 5.3.1.3**.

En todas estas 6 tablas se presentan resaltados en "negrita" los valores mayores o iguales a 0.30 en valor absoluto, para facilitar el análisis. Es de observar que coinciden en interpretación las ejecuciones en todos los escenarios.

Tabla 5.3.1.4: Componente HUMOR en la matriz de cargas B

ESCALAS BIPOLARES	PARAFAC con B ortogonal	PARAFAC con B ortogonal disjunta	PARAFAC con C ortogonal disjunta
(1) Thrilling-Boring	-0.09	0.00	-0.05
(2) Intelligent-Idiotic	**0.30**	**0.39**	**0.35**
(3) Erotic-Not Erotic	-0.25	**-0.35**	-0.20
(4) Sensitive-Insensitive	0.03	0.00	-0.13
(5) Interesting-Uninteresting	0.00	0.00	0.12
(6) Fast-Slow	-0.08	0.00	-0.02
(7) Intell. Stimul.-Intell. Dull	0.27	**0.39**	**0.36**
(8) Violent-Peaceful	0.08	0.00	0.11
(9) Caring-Callous	0.04	0.00	-0.15
(10) Satirical-Not Satirical	**-0.46**	**-0.39**	**-0.35**
(11) Informative-Uninformative	**0.31**	**0.40**	**0.38**
(12) Touching-Leave Me Cold	-0.07	0.00	-0.23
(13) Deep-Shallow	0.21	0.00	0.16
(14) Tasteful-Crude	0.19	0.00	0.05
(15) Real-Fantasy	**0.38**	**0.36**	**0.33**
(16) Funny-Not Funny	**-0.46**	**-0.38**	**-0.42**

Tabla 5.3.1.5: Componente HUMOR en la matriz de cargas C

PROGRAMAS DE TV	PARAFAC con B ortogonal	PARAFAC con B ortogonal disjunta	PARAFAC con C ortogonal disjunta
(1) Mash	0.14	0.13	0.17
(2) Charlie's angels	0.21	0.26	0.00
(3) All in the family	0.20	0.18	0.28
(4) 60 minutes	**-0.35**	**-0.37**	**-0.51**
(5) The tonight show	0.16	0.10	0.00
(6) Let's make a deal	0.19	0.19	0.00
(7) The Waltons	-0.13	-0.03	0.00
(8) Saturday night live	**0.35**	**0.30**	0.00
(9) News	**-0.39**	**-0.42**	**-0.63**
(10) Kojak	0.09	0.10	0.00
(11) Mork and Mindy	**0.38**	**0.37**	**0.49**
(12) Jacques Cousteau	**-0.40**	**-0.42**	0.00
(13) Football	-0.07	-0.08	0.00
(14) Little house on the prairie	-0.06	0.02	0.00
(15) Wild kingdom	**-0.31**	**-0.32**	0.00

Tabla 5.3.1.6: Componente SENSIBILIDAD en la matriz de cargas B

ESCALAS BIPOLARES	PARAFAC con B ortogonal	PARAFAC con B ortogonal disjunta	PARAFAC con C ortogonal disjunta
(1) Thrilling-Boring	0.29	**0.40**	**0.33**
(2) Intelligent-Idiotic	0.17	0.00	0.03
(3) Erotic-Not Erotic	-0.06	0.00	0.11
(4) Sensitive-Insensitive	**-0.43**	**-0.50**	**-0.44**
(5) Interesting-Uninteresting	**0.37**	**0.37**	0.27
(6) Fast-Slow	0.19	0.00	0.26
(7) Intell. Stimul.-Intell. Dull	0.20	0.00	0.05
(8) Violent-Peaceful	0.04	0.00	0.17
(9) Caring-Callous	**-0.47**	**-0.52**	**-0.44**
(10) Satirical-Not Satirical	0.09	0.00	0.18
(11) Informative-Uninformative	0.20	0.00	0.05
(12) Touching-Leave Me Cold	**-0.31**	**-0.41**	**-0.30**
(13) Deep-Shallow	-0.14	0.00	-0.23
(14) Tasteful-Crude	**-0.31**	0.00	**-0.35**
(15) Real-Fantasy	0.06	0.00	-0.05
(16) Funny-Not Funny	0.06	0.00	0.12

Tabla 5.3.1.7: Componente SENSIBILIDAD en la matriz de cargas C

PROGRAMAS DE TV	PARAFAC con B ortogonal	PARAFAC con B ortogonal disjunta	PARAFAC con C ortogonal disjunta
(1) Mash	0.03	0.04	0.00
(2) Charlie's angels	0.13	0.09	0.00
(3) All in the family	0.04	0.07	0.00
(4) 60 minutes	-0.19	-0.17	0.00
(5) The tonight show	-0.23	-0.24	-0.24
(6) Let's make a deal	0.00	0.00	0.00
(7) The Waltons	**0.55**	**0.55**	**0.60**
(8) Saturday night live	**-0.31**	**-0.32**	**-0.44**
(9) News	-0.27	-0.22	0.00
(10) Kojak	-0.04	-0.06	0.00
(11) Mork and Mindy	0.12	0.07	0.00
(12) Jacques Cousteau	-0.10	-0.07	0.00
(13) Football	**-0.32**	**-0.37**	**-0.31**
(14) Little house on the prairie	**0.54**	**0.54**	**0.55**
(15) Wild kingdom	0.04	0.09	0.00

Tabla 5.3.1.8: Componente VIOLENCIA en la matriz de cargas B

ESCALAS BIPOLARES	PARAFAC con B ortogonal	PARAFAC con B ortogonal disjunta	PARAFAC con C ortogonal disjunta
(1) Thrilling-Boring	-0.12	0.00	-0.05
(2) Intelligent-Idiotic	0.12	0.00	**0.32**
(3) Erotic-Not Erotic	-0.27	0.00	**-0.38**
(4) Sensitive-Insensitive	0.12	0.00	0.03
(5) Interesting-Uninteresting	0.26	0.00	0.22
(6) Fast-Slow	-0.29	**-0.52**	-0.24
(7) Intell. Stimul.-Intell. Dull	0.25	0.00	**0.37**
(8) Violent-Peaceful	**-0.68**	**-0.47**	**-0.34**
(9) Caring-Callous	0.02	0.00	-0.02
(10) Satirical-Not Satirical	0.18	0.00	-0.25
(11) Informative-Uninformative	0.19	0.00	**0.38**
(12) Touching-Leave Me Cold	0.15	0.00	0.04
(13) Deep-Shallow	0.16	**0.51**	0.24
(14) Tasteful-Crude	0.02	**0.50**	0.12
(15) Real-Fantasy	-0.06	0.00	0.28
(16) Funny-Not Funny	**0.30**	0.00	-0.17

Tabla 5.3.1.9: Componente VIOLENCIA en la matriz de cargas C

PROGRAMAS DE TV	PARAFAC con B ortogonal	PARAFAC con B ortogonal disjunta	PARAFAC con C ortogonal disjunta
(1) Mash	-0.11	0.01	0.00
(2) Charlie's angels	**0.54**	**0.30**	**0.53**
(3) All in the family	-0.18	0.01	0.00
(4) 60 minutes	-0.18	-0.14	0.00
(5) The tonight show	-0.19	0.02	0.00
(6) Let's make a deal	0.25	0.27	**0.36**
(7) The Waltons	-0.23	**-0.51**	0.00
(8) Saturday night live	0.11	**0.31**	0.00
(9) News	0.00	0.05	0.00
(10) Kojak	**0.37**	0.28	0.25
(11) Mork and Mindy	-0.10	0.01	0.00
(12) Jacques Cousteau	-0.29	-0.29	**-0.58**
(13) Football	**0.39**	**0.30**	0.00
(14) Little house on the prairie	-0.18	**-0.41**	0.00
(15) Wild kingdom	-0.21	-0.20	**-0.44**

En la **Tabla 5.3.1.10** y en la **Tabla 5.3.1.11** se muestran las matrices de cargas $\boldsymbol{B}$ y $\boldsymbol{C}$ respectivamente, en el escenario de ejecución con el algoritmo CBPSO-ParafacALS en el cual la matriz $\boldsymbol{B}$ tiene componentes ortogonales disjuntos. Se resaltan los valores usando "negritas" a partir de 0.30 en valor absoluto para facilitar el análisis.

Tabla 5.3.1.10: Matriz de cargas $\boldsymbol{B}$ ortogonal disjunta con CBPSO-ParafacALS

ESCALAS BIPOLARES	HUMOR	SENSIBILIDAD	VIOLENCIA
(1) Thrilling-Boring	0.00	**0.40**	0.00
(2) Intelligent-Idiotic	**0.39**	0.00	0.00
(3) Erotic-Not Erotic	**-0.35**	0.00	0.00
(4) Sensitive-Insensitive	0.00	**-0.50**	0.00
(5) Interesting-Uninteresting	0.00	**0.37**	0.00
(6) Fast-Slow	0.00	0.00	**-0.52**
(7) Intell. Stimul.-Intell. Dull	**0.39**	0.00	0.00
(8) Violent-Peaceful	0.00	0.00	**-0.47**
(9) Caring-Callous	0.00	**-0.52**	0.00
(10) Satirical-Not Satirical	**-0.39**	0.00	0.00
(11) Informative-Uninformative	0.40	0.00	0.00
(12) Touching-Leave Me Cold	0.00	**-0.41**	0.00
(13) Deep-Shallow	0.00	0.00	**0.51**
(14) Tasteful-Crude	0.00	0.00	**0.50**
(15) Real-Fantasy	**0.36**	0.00	0.00
(16) Funny-Not Funny	**-0.38**	0.00	0.00

Tabla 5.3.1.11: Matriz de cargas $\boldsymbol{C}$ no disjunta con CBPSO-ParafacALS

PROGRAMAS DE TV	HUMOR	SENSIBILIDAD	VIOLENCIA
(1) Mash	0.13	0.04	0.01
(2) Charlie's angels	0.26	0.09	**0.30**
(3) All in the family	0.18	0.07	0.01
(4) 60 minutes	**-0.37**	-0.17	-0.14
(5) The tonight show	0.10	-0.24	0.02
(6) Let's make a deal	0.19	0.00	0.27
(7) The Waltons	-0.03	**0.55**	**-0.51**
(8) Saturday night live	**0.30**	**-0.32**	**0.31**
(9) News	**-0.42**	-0.22	0.05
(10) Kojak	0.10	-0.06	0.28
(11) Mork and Mindy	**0.37**	0.07	0.01
(12) Jacques Cousteau	**-0.42**	-0.07	-0.29
(13) Football	-0.08	**-0.37**	**0.30**
(14) Little house on the prairie	0.02	**0.54**	**-0.41**
(15) Wild kingdom	**-0.32**	0.09	-0.20

En la **Tabla 5.3.1.12** y en la **Tabla 5.3.1.13** se muestran las matrices de cargas $\boldsymbol{B}$ y $\boldsymbol{C}$ respectivamente, en el escenario de ejecución con el algoritmo CBPSO-ParafacALS en el cual la matriz $\boldsymbol{C}$ tiene componentes ortogonales disjuntos. Se resaltan los valores usando "negritas" a partir de 0.30 en valor absoluto para facilitar el análisis.

Tabla 5.3.1.12: Matriz de cargas $\boldsymbol{B}$ no disjunta con CBPSO-ParafacALS

ESCALAS BIPOLARES	HUMOR	SENSIBILIDAD	VIOLENCIA
(1) Thrilling-Boring	-0.05	**0.33**	-0.05
(2) Intelligent-Idiotic	**0.35**	0.03	**0.32**
(3) Erotic-Not Erotic	-0.20	0.11	**-0.38**
(4) Sensitive-Insensitive	-0.13	**-0.44**	0.03
(5) Interesting-Uninteresting	0.12	0.27	0.22
(6) Fast-Slow	-0.02	0.26	-0.24
(7) Intell. Stimul.-Intell. Dull	**0.36**	0.05	**0.37**
(8) Violent-Peaceful	0.11	0.17	**-0.34**
(9) Caring-Callous	-0.15	**-0.44**	-0.02
(10) Satirical-Not Satirical	**-0.35**	0.18	-0.25
(11) Informative-Uninformative	**0.38**	0.05	**0.38**
(12) Touching-Leave Me Cold	-0.23	**-0.30**	0.04
(13) Deep-Shallow	0.16	-0.23	0.24
(14) Tasteful-Crude	0.05	**-0.35**	0.12
(15) Real-Fantasy	**0.33**	-0.05	0.28
(16) Funny-Not Funny	**-0.42**	0.12	-0.17

Tabla 5.3.1.13: Matriz de cargas $\boldsymbol{C}$ ortogonal disjunta con CBPSO-ParafacALS

PROGRAMAS DE TV	HUMOR	SENSIBILIDAD	VIOLENCIA
(1) Mash	0.17	0.00	0.00
(2) Charlie's angels	0.00	0.00	**0.53**
(3) All in the family	0.28	0.00	0.00
(4) 60 minutes	**-0.51**	0.00	0.00
(5) The tonight show	0.00	-0.24	0.00
(6) Let's make a deal	0.00	0.00	**0.36**
(7) The Waltons	0.00	**0.60**	0.00
(8) Saturday night live	0.00	**-0.44**	0.00
(9) News	**-0.63**	0.00	0.00
(10) Kojak	0.00	0.00	**0.25**
(11) Mork and Mindy	**0.49**	0.00	0.00
(12) Jacques Cousteau	0.00	0.00	**-0.58**
(13) Football	0.00	**-0.31**	0.00
(14) Little house on the prairie	0.00	**0.55**	0.00
(15) Wild kingdom	0.00	0.00	**-0.44**

Al comparar los resultados entregados por el algoritmo heurístico CBPSO-ParafacALS con los resultados del algoritmo tradicional ParafacALS se observa que la interpretación es la misma, a excepción de pequeñas diferencias, para los 3 componentes del modelo PARAFAC: humor, sensibilidad y violencia. La metodología disjunta que se propone en esta tesis permite identificar la presencia de estos 3 componentes, tanto en las escalas bipolares como en los programas de TV (ver Lundy, Harshman and Kruskal 1989).

Los valores de los parámetros del algoritmo CBPSO-ParafacALS que se usaron para los resultados presentados en este **Experimento # 1**, son:

- $ALSMaxIter = 10000$
- $Tol = 10^{-7}$
- $PSOMaxIter = 30$
- $nPar = 30$
- $minIner = 0.8$
- $maxIner = 1.2$
- $wCognition = 2$
- $wSocial = 2$

El valor de 10000 en el parámetro $ALSMaxIter$ suele ser el valor utilizado en el algoritmo ParafacALS (paquetes del software R). Se probaron 5 configuraciones diferentes para los valores de los parámetros y esta que se muestra es la que presentó los mejores resultados. Las otras 4 configuraciones usadas se presentan a continuación:

Tabla 5.3.1.14: Configuraciones para los parámetros en el Experimento # 1

PARÁMETROS	CONFIGURACIÓN 1	CONFIGURACIÓN 2	CONFIGURACIÓN 3	CONFIGURACIÓN 4
$ALSMaxIter$	10000	10000	10000	10000
Tol	10^{-7}	10^{-7}	10^{-7}	10^{-7}
$PSOMaxIter$	30	20	30	20
$nPar$	30	40	30	40
$minIner$	0.5	0.75	0.5	0.75
$maxIner$	1.5	1.25	1.5	1.25
$wCognition$	1.5	1.8	1.5	1.8
$wSocial$	2.5	3	2.5	3

5.3.2 APLICANDO EL ALGORITMO CBPSO-TuckALS3 A DATOS RELACIONADOS CON NIVELES DE COMPORTAMIENTO

Esta sección está dedicada al **Experimento # 2**. Se muestran a continuación los 4 cortes frontales de una tabla $\underline{\mathbf{X}}$ de 3 vías con $I = 6$ individuos, $J = 4$ variables y $K = 4$ situaciones diferentes, sobre las que se miden niveles de comportamiento.

Tabla 5.3.2.1: Corte frontal de la situación "HACIENDO UN EXAMEN"

	EMOCIONALIDAD	SENSIBILIDAD	CONCENTRABILIDAD	RACIONALIDAD
ANNE	0.0	0.0	4.0	4.0
BERT	0.0	0.0	2.0	2.0
CLAUS	0.0	0.0	2.0	2.0
DOLLY	0.0	0.0	4.0	4.0
EDNA	0.0	0.0	2.5	2.5
FRANCES	0.0	0.0	4.0	4.0

Tabla 5.3.2.2: Corte frontal de la situación "DANDO UN DISCURSO"

	EMOCIONALIDAD	SENSIBILIDAD	CONCENTRABILIDAD	RACIONALIDAD
ANNE	0.6	0.6	2.4	2.4
BERT	0.2	0.2	1.8	1.8
CLAUS	0.2	0.2	1.8	1.8
DOLLY	0.6	0.6	2.4	2.4
EDNA	0.4	0.4	2.1	2.1
FRANCES	0.6	0.6	2.4	2.4

Tabla 5.3.2.3: Corte frontal de la situación "COMPARTIENDO EN UN PICNIC FAMILIAR"

	EMOCIONALIDAD	SENSIBILIDAD	CONCENTRABILIDAD	RACIONALIDAD
ANNE	4.0	4.0	0.0	0.0
BERT	1.0	1.0	1.0	1.0
CLAUS	1.0	1.0	1.0	1.0
DOLLY	4.0	4.0	0.0	0.0
EDNA	2.0	2.0	0.5	0.5
FRANCES	4.0	4.0	0.0	0.0

Tabla 5.3.2.4: Corte frontal de la situación "COMPARTIENDO EN UNA CITA ROMÁNTICA"

	EMOCIONALIDAD	SENSIBILIDAD	CONCENTRABILIDAD	RACIONALIDAD
ANNE	4.6	4.6	0.9	0.9
BERT	1.2	1.2	1.8	1.8
CLAUS	1.2	1.2	1.8	1.8
DOLLY	4.6	4.6	0.9	0.9
EDNA	2.4	2.4	1.4	1.4
FRANCES	4.6	4.6	0.9	0.9

Desde la **Tabla 5.3.2.1** hasta la **Tabla 5.3.2.4** se muestran, respectivamente, las 4 matrices de tamaño 6×4 que corresponden a las distintas situaciones en las cuales se evalúan los niveles de comportamiento: "HACIENDO UN EXAMEN", "DANDO UN DISCURSO", "COMPARTIENDO EN UN PICNIC FAMILIAR" y "COMPARTIENDO EN UNA CITA ROMÁNTICA".

Tal como en el artículo de Amaya and Pacheco 2002 se escogen $P = Q = R = 2$ componentes y se ejecuta el algoritmo TuckALS3 tal como están los datos, es decir, sin realizar ningún tipo de preprocesado. La salida del algoritmo nos dice que el fit del modelo es 99.8403%. Por otro lado, el fit del modelo es 98.1048% al ejecutar el algoritmo CBPSO-TuckALS3. Se calcularon componentes ortogonales disjuntos en todas las matrices de cargas.

Al interpretar la salida del algoritmo TuckALS3 en el primer modo, se puede afirmar que el primer componente de la matriz de cargas $\boldsymbol{A}$ de individuos representa feminidad, y el segundo componente masculinidad, tal como en la **Tabla 5.3.2.5** se observa, de acuerdo a los valores resaltados en "negritas".

Tabla 5.3.2.5: Matriz de cargas A ortogonal con el algoritmo TuckALS3

	FEMINIDAD	MASCULINIDAD
ANNE	**-0.518560994**	0.228268998
BERT	-0.216781213	**-0.619921659**
CLAUS	-0.216781213	**-0.619921659**
DOLLY	**-0.518560994**	0.228268998
EDNA	**-0.315111562**	-0.273996475
FRANCES	**-0.518560994**	0.228268998

En la **Tabla 5.3.2.6** se observa la matriz de cargas $\boldsymbol{A}$ ortogonal disjunta obtenida con el algoritmo CBPSO-TuckALS3. La interpretación es la misma que con el algoritmo tradicional TuckALS3, pero es más fácil debido a la estructura simple que presenta la matriz de cargas.

Tabla 5.3.2.6: Matriz de cargas A ortogonal disjunta con el algoritmo CBPSO-TuckALS3

	FEMININITY	MASCULINITY
ANNE	-0.545737149	0
BERT	0	-0.707106781
CLAUS	0	-0.707106781
DOLLY	-0.545737149	0
EDNA	-0.326363131	0
FRANCES	-0.545737149	0

El algoritmo CBPSO-TuckALS3 logra identificar la estructura disjunta que se encuentra en el modo de los individuos (primer modo).

Al interpretar la salida del algoritmo TuckALS3 en el segundo modo, se puede afirmar que el primer componente de la matriz de cargas $\boldsymbol{B}$ de las variables representa la parte emocional, y el segundo componente la parte consciente o racional, tal como se observa en la **Tabla 5.3.2.7**, de acuerdo a los valores resaltados en "negritas".

Tabla 5.3.2.7: Matriz de cargas B ortogonal con el algoritmo TuckALS3

	VARIABLE LATENTE EMOCIONAL	VARIABLE LATENTE CONSCIENTE
EMOCIONALIDAD	**-0.588715219**	-0.391681492
SENSIBILIDAD	**-0.588715219**	-0.391681492
CONCENTRABILIDAD	-0.391681492	**0.588715219**
RACIONALIDAD	-0.391681492	**0.588715219**

En la **Tabla 5.3.2.8** se observa la matriz de cargas $\boldsymbol{B}$ ortogonal disjunta obtenida con el algoritmo CBPSO-TuckALS3. La interpretación es la misma que con el algoritmo tradicional TuckALS3, pero es más fácil debido a la estructura simple que presenta la matriz de cargas.

Tabla 5.3.2.8: Matriz de cargas B ortogonal disjunta con el algoritmo CBPSO-TuckALS3

	VARIABLE LATENTE EMOCIONAL	VARIABLE LATENTE CONSCIENTE
EMOCIONALIDAD	-0.707106781	0
SENSIBILIDAD	-0.707106781	0
CONCENTRABILIDAD	0	-0.707106781
RACIONALIDAD	0	-0.707106781

El algoritmo CBPSO-TuckALS3 logra identificar la estructura disjunta que se encuentra en el modo de las variables (segundo modo).

Al interpretar la salida del algoritmo TuckALS3 en el tercer modo, se puede afirmar que el primer componente de la matriz de cargas $\boldsymbol{C}$ de las situaciones representa una situación social, y el segundo componente una situación de rendimiento, tal como se observa en la **Tabla 5.3.2.9**, de acuerdo a los valores resaltados en "negritas".

Tabla 5.3.2.9: Matriz de cargas C ortogonal con el algoritmo TuckALS3

	SITUACIÓN SOCIAL	SITUACIÓN DE RENDIMIENTO
HACIENDO UN EXAMEN	-0.333198693	**0.770591276**
DANDO UN DISCURSO	-0.310425514	**0.435143574**
COMPARTIENDO EN UN PICNIC FAMILIAR	**-0.538191124**	-0.387207268
COMPARTIENDO EN UNA CITA ROMÁNTICA	**-0.709200215**	-0.258669069

En la **Tabla 5.3.2.10** se observa la matriz de cargas $\boldsymbol{C}$ ortogonal disjunta obtenida con el algoritmo CBPSO-TuckALS3. La interpretación es la misma que con el algoritmo tradicional TuckALS3, pero es más fácil debido a la estructura simple que presenta la matriz de cargas.

Tabla 5.3.2.10: Matriz de cargas C ortogonal disjunta con el algoritmo CBPSO-TuckALS3

	SITUACIÓN SOCIAL	SITUACIÓN DE RENDIMIENTO
HACIENDO UN EXAMEN	0	-0.827376571
DANDO UN DISCURSO	0	-0.561647585
COMPARTIENDO EN UN PICNIC FAMILIAR	-0.633810357	0
COMPARTIENDO EN UNA CITA ROMÁNTICA	-0.773488482	0

El algoritmo CBPSO-TuckALS3 logra identificar la estructura disjunta que se encuentra en el modo de las situaciones (tercer modo).

En la **Tablas 5.3.2.11** y en la **Tabla 5.3.2.12** se observa la matriz core $\underline{\mathbf{G}}$ obtenida con el algoritmo TuckALS3 y con el algoritmo CBPSO-TuckALS3, respectivamente. Se puede interpretar, al observar las dos matrices, que las mujeres en situaciones sociales son principalmente emocionales y poco conscientes. Los hombres en situaciones sociales son, por el contrario, más conscientes que emocionales. Además, en situaciones formales y de rendimiento, las mujeres son principalmente conscientes. Los hombres en situaciones formales y de rendimiento son conscientes más que emocionales. Es decir, se obtiene la misma interpretación del core. Al realizar el cálculo de componentes ortogonales disjuntos en las tres matrices de cargas $\boldsymbol{A}$, $\boldsymbol{B}$ y $\boldsymbol{C}$, los valores del core $\underline{\boldsymbol{G}}$ son distintos, pero en este experimento computacional el analista de los datos llega a las mismas conclusiones.

Tabla 5.3.2.11: Core $\underline{\mathbf{G}}$ con el algoritmo TuckALS3

	SITUACIÓN SOCIAL		SITUACIÓN DE RENDIMIENTO	
	VAR. LATENTE EMOCIONAL	VAR. LATENTE CONSCIENTE	VAR. LATENTE EMOCIONAL	VAR. LATENTE CONSCIENTE
FEMINIDAD	-17.09837769	-0.489491858	0.50606476	-12.25957342
MASCULINIDAD	-0.623702797	4.106153763	0.54320104	0.72835096

Tabla 5.3.2.12: Core $\underline{\mathbf{G}}$ con el algoritmo CBPSO-TuckALS3

	SITUACIÓN SOCIAL		SITUACIÓN DE RENDIMIENTO	
	VAR. LATENTE EMOCIONAL	VAR. LATENTE CONSCIENTE	VAR. LATENTE EMOCIONAL	VAR. LATENTE CONSCIENTE
FEMINIDAD	-15.55006689	-2.25788715	-0.883942787	-12.28278827
MASCULINIDAD	-3.12399307	-4.052179249	-0.224659034	-5.33143759

Importante tener presente que con el algoritmo TuckALS3 no siempre las matrices de cargas $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$ son fácilmente interpretables, tal como se afirma en Kiers and Van Mechelen 2001. En muchas situaciones estas matrices necesitan ser rotadas (las tres matrices pueden ser rotadas de forma separada o de manera conjunta, y compensar dichas rotaciones en el core $\underline{\mathbf{G}}$) con el objetivo de alcanzar una estructura simple que permita realizar una interpretación. Sin embargo, el realizar rotaciones, no garantiza que esa estructura simple pueda ser alcanzada.

Para buscar una estructura simple y mejorar la interpretación, se puede rotar sólo la matriz $\boldsymbol{B}$ o se puede elegir rotar las matrices $\boldsymbol{B}$ y $\boldsymbol{C}$. En algunos casos el investigador puede optar por rotar las tres matrices de carga. En el algoritmo CBPSO-TuckALS3 existe la misma analogía, se puede elegir componentes ortogonales disjuntos en una única matriz ($\boldsymbol{B}$ por ejemplo), en dos de las matrices ($\boldsymbol{B}$ y $\boldsymbol{C}$ por ejemplo) o incluso en las tres matrices de cargas.

Al buscar una estructura simple mediante rotaciones se mantiene el fit, pero al buscar una estructura simple por medio de componentes ortogonales disjuntos en las matrices de cargas se debe tener presente que se pierde fit y por ende variabilidad explicada por el modelo.

Se ejecutó el algoritmo CBPSO-TuckALS3 varias veces con los mismos datos de esta sección considerando todas las combinaciones posibles de componentes ortogonales disjuntos en las matrices de cargas $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$. Se presenta a continuación la **Tabla 5.3.2.13** que resume todos los resultados del fit del modelo:

Tabla 5.3.2.13: Comparación del fit en escenarios de ejecución del Experimento # 2

ESCENARIOS DE EJECUCIÓN COMPONENTES ORTOGONALES DISJUNTOS	FIT (%)
Ninguna matriz de cargas (TuckALS3)	99.8403
$\boldsymbol{A}$ (CBPSO-TuckALS3)	99.2496
$\boldsymbol{B}$ (CBPSO-TuckALS3)	99.8403
$\boldsymbol{C}$ (CBPSO-TuckALS3)	98.6659
$\boldsymbol{A}, \boldsymbol{B}$ (CBPSO-TuckALS3)	99.2496
$\boldsymbol{A}, \boldsymbol{C}$ (CBPSO-TuckALS3)	98.1048
$\boldsymbol{B}, \boldsymbol{C}$ (CBPSO-TuckALS3)	98.6659
$\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$ (CBPSO-TuckALS3)	98.1048

Lo más costoso en términos de fit es un modelo Tucker3 en el cual las tres matrices de cargas tienen componentes ortogonales disjuntos. Se llegó a esta conclusión en base a las decenas de experimentos computacionales llevados a cabo no sólo con los datos de esta sección sino también con otros datos.

Los valores de los parámetros del algoritmo CBPSO-TuckALS3 que se usaron para los resultados presentados en este **Experimento # 2**, son:

- $ALSMaxIter = 100$
- $Tol = 10^{-7}$
- $PSOMaxIter = 20$
- $nPar = 30$
- $minIner = 0.8$
- $maxIner = 1.2$
- $wCognition = 2$
- $wSocial = 2$

El valor de 100 en el parámetro $ALSMaxIter$ suele ser el valor utilizado en el algoritmo TuckALS3 (paquetes del software R). En el algoritmo ParafacALS se emplea $ALSMaxIter = 10000$ debido a la posible presencia del problema de "degeneración". Se probaron 5 configuraciones diferentes para los valores de los parámetros y esta que se muestra es la que presentó los mejores resultados. Las otras 4 configuraciones usadas las podemos observar a continuación:

Tabla 5.3.2.14: Configuraciones para los parámetros en el Experimento # 2

PARÁMETROS	CONFIGURACIÓN 1	CONFIGURACIÓN 2	CONFIGURACIÓN 3	CONFIGURACIÓN 4
$ALSMaxIter$	100	100	100	100
Tol	10^{-7}	10^{-7}	10^{-7}	10^{-7}
$PSOMaxIter$	30	20	30	20
$nPar$	30	40	30	40
$minIner$	0.5	0.75	0.5	0.75
$maxIner$	1.5	1.25	1.5	1.25
$wCognition$	1.5	1.8	1.5	1.8
$wSocial$	2.5	3	2.5	3

5.3.3 APLICANDO EL ALGORITMO CBPSO-TuckALS3 A DATOS REALES SOBRE LOS PRELUDIOS DE CHOPIN

Esta sección está dedicada al **Experimento # 3**. Como parte del preprocesado se realiza un centrado a través del primer modo y luego un normalizado en el segundo modo (ver Murakami and Kroonenberg 2003). El primero modo corresponde a los preludios de Chopin, el segundo modo a las escalas bipolares y el tercer modo es el de los estudiantes universitarios. El modelo escogido por los autores del artículo es un modelo Tucker3 con $P = 2$ componentes, $Q = 3$ componentes y $R = 2$ componentes. Los datos de este experimento se pueden tomar de http://three-mode.leidenuniv.nl.

Se ejecutó el algoritmo TuckALS3 y se obtuvo un fit del 42.63% en el modelo en 9 iteraciones. Cabe indicar que la salida del algoritmo no es directamente interpretable, ya que las matrices de cargas no presentan una estructura simple, por lo que los autores del mencionado artículo se ven obligados a realizar rotaciones en las matrices de cargas y también en el core, para luego poder realizar la correspondiente interpretación (en la rotación se mantiene el fit). En esta tesis se realizaron ejecuciones en 2 escenarios diferentes usando el algoritmo CBPSO-TuckALS3. En la primera ejecución se puso como requerimiento que únicamente la matriz de cargas $\boldsymbol{A}$ tenga componentes ortogonales disjuntos, y se obtuvo un fit del 38.92% en el modelo. En la segunda ejecución el requisito fue que las matrices de cargas $\boldsymbol{A}$ y $\boldsymbol{B}$ tengan componentes ortogonales disjuntos, y se obtuvo en el modelo un fit del 36.79%. En total se trata de 3 escenarios de ejecución, los cuales se resumen en la **Tabla 5.3.3.1** que se muestra:

Tabla 5.3.3.1: Comparación del fit en escenarios de ejecución del Experimento # 3

ESCENARIOS DE EJECUCIÓN COMPONENTES ORTOGONALES DISJUNTOS	FIT (%)
Ninguna matriz de cargas (TuckALS3)	42.63
$\boldsymbol{A}$ (CBPSO-TuckALS3)	38.92
$\boldsymbol{A}$, $\boldsymbol{B}$(CBPSO-TuckALS3)	36.79

Se procede a mostrar sólo la salida del último escenario de ejecución, solución que es la más interpretable de todas. Se presenta en la **Tabla 5.3.3.2** la matriz de cargas $\boldsymbol{A}$ ortogonal disjunta y en la **Tabla 5.3.3.3** se presenta la matriz de cargas $\boldsymbol{B}$ ortogonal disjunta.

Para la matriz de cargas ortogonal disjunta $\boldsymbol{A}$ el análisis es el siguiente: Los preludios de Chopin que están de color rojo son aquellos que Murakami and Kroonenberg 2003 afirman que se encuentran en el primer componente, etiquetado como "FAST+MINOR, SLOW+MAJOR". Por otro lado, los que están de color verde son los que se encuentran, según los autores, en el segundo componente etiquetado como "FAST+MAJOR, SLOW+MINOR". Los que están de color café son aquellos preludios que los autores logran identificar que no tienen cargas representativas en ninguno de los 2 componentes. Se observa que la misma interpretación la podemos realizar con la salida del algoritmo CBPSO-TuckALS3.

Tabla 5.3.3.2: Matriz de cargas A ortogonal disjunta con el algoritmo CBPSO-TuckALS3

PRELUDIOS DE CHOPIN	FAST+MINOR, SLOW+MAJOR	FAST+MAJOR, SLOW+MINOR
(1) C major Agitato	0.091495243	0
(2) a minor Lento	0	-0.349499568
(3) G major Vivace	0	0.380083836
(4) e minor Largo	0	-0.216381495
(5) D major Allegro	0	0.322563112
(6) b minor Lento assai	0	-0.306829783
(7) A major Andantino	-0.391407119	0
(8) f# minor Molto agitato	0.170783607	0
(9) E major Largo	0	-0.308357327
(10) c# minor Allegro molto	0	0.130529712
(11) B major Vivace	-0.292993592	0
(12) g# minor Presto	0.231153448	0
(13) F# major Lento	-0.191308850	0
(14) eb minor Allegro	0.252789215	0
(15) Db major Sostenuto	-0.393591840	0
(16) bb minor Presto con fuoco	0.254710019	0
(17) Ab major Allegretto	-0.071449806	0
(18) f minor Allegro molto	0.352314932	0
(19) Eb major Vivace	0	0.300266062
(20) c minor Largo	0	-0.359417233
(21) Bb major Cantabile	-0.178560495	0
(22) g minor Molto agitato	0.308258927	0
(23) F major Moderato	0	0.396120182
(24) d minor Allegro appasionato	0.305864985	0

Para la matriz de cargas ortogonal disjunta $\boldsymbol{B}$ el análisis es el siguiente: Las escalas bipolares que están de rojo son aquellas que Murakami and Kroonenberg 2003 afirman que se encuentran en el primer componente, etiquetado como "SOOTHING-AGITATING". Por otro lado, las escalas que están de color verde son las que se ubican, según los autores, en el segundo componente etiquetado como "CHEERFUL-MOURNFUL". Finalmente, las escalas de color morado son las que, a criterio de los autores, se encuentran en un tercer componente que ellos llaman "DISLIKE-LIKE". Las escalas bipolares que están de color café son aquellas que los autores identifican que no tienen cargas representativas en ninguno de los 3 componentes. Se observa que una interpretación similar la podemos realizar con la salida del algoritmo CBPSO-TuckALS3, a excepción de lo que ocurre en el tercer componente.

Tabla 5.3.3.3: Matriz de cargas B ortogonal disjunta con el algoritmo CBPSO-TuckALS3

ESCALAS BIPOLARES	SOOTHING-AGITATING	CHEERFUL-MOURNFUL	DISLIKE-LIKE
(1) bright-dark	0	0.415123535	0
(2) slow-fast	0	0	**0.755252562**
(3) uninteresting-interesting	0	**-0.119274696**	0
(4) gentle-severe	0.391410866	0	0
(5) light-heavy	0	0.470232892	0
(6) clear-cloudy	0	0.373873628	0
(7) weak-strong	0.327769110	0	0
(8) warm-cold	0.273596319	0	0
(9) tranquil-vehement	0.364019736	0	0
(10) small-large	0.222702331	0	0
(11) soft-hard	0	0.344695525	0
(12) still-loud	0.349870226	0	0
(13) happy-sad	0	0.380865702	0
(14) calm-restless	0.376896650	0	0
(15) delicate-coarse	0.228397330	0	0
(16) thin-thick	0.240310465	0	0
(17) quiet-noisy	0	0	**0.655433878**
(18) cheerful-gloomy	0	0.419326213	0
(19) lyric-dramatic	0.328677017	0	0
(20) unattractive-attractive	0	**-0.113302141**	0

Al comparar la matriz de cargas $\boldsymbol{B}$ ortogonal disjunta obtenida con el algoritmo CBPSO-TuckALS3, con la matriz de cargas $\boldsymbol{B}$ que se presenta en el mencionado artículo, encontramos que existen 4 diferencias que en la **Tabla 5.3.3.3** están resaltadas de color negro. Recordemos que por buscar una estructura simple para facilitar la interpretación, hubo pérdida de fit. Es importante tener presente que habrá que lidiar con esto cada vez que usemos el algoritmo CBPSO-TuckALS3. Sin embargo, si consideramos que puede existir otra perspectiva, la matriz de cargas $\boldsymbol{B}$ ortogonal disjunta nos puede hacer pensar que quizás ese tercer componente propuesto por los autores no se trata de un componente de agrado o desagrado de los preludios de Chopin. Podría tratarse de un componente que permite clasificar a los preludios como rápidos y ruidosos o, por el contrario, como lentos y silenciosos, debido a las cargas altas de las escalas bipolares "slow-fast" y "quiet-noisy". Podríamos etiquetar ese componente por ejemplo como "RELAX-TENSE". En resumen, se recomienda utilizar el algoritmo heurístico CBPSO-TuckALS3 de manera complementaria (no excluyente) con otros algoritmos existentes para el análisis de componentes en tablas de 3 vías, porque puede darnos otra perspectiva en el análisis de los datos.

A continuación en la **Tabla 5.3.3.4** se muestra la matriz de cargas $\boldsymbol{C}$ ortogonal pero no disjunta que se obtiene en el tercer escenario de ejecución y también la matriz de cargas $\boldsymbol{C}$ que presenta Murakami and Kroonenberg 2003. Respecto de esta matriz, los autores del artículo llaman al primer componente "CONSENSUS" y al segundo componente "CONTRAST". Se observa que en el primer componente "CONSENSUS", al comparar ambas matrices de cargas $\boldsymbol{C}$, los valores son muy cercanos en los 38 sujetos. Por otro lado, en el segundo componente "CONTRAST", al comparar ambas matrices $\boldsymbol{C}$, los valores son cercanos en algunos de los sujetos, pero en otros no.

Tabla 5.3.3.4: Matriz de cargas C ortogonal disjunta con el algoritmo CBPSO-TuckALS3

SUJETOS	CONSENSUS CBPSO-TuckALS3	CONSENSUS Murakami and Kroonenberg	CONTRAST CBPSO-TuckALS3	CONTRAST Murakami and Kroonenberg
S1	0.168	0.169	-0.062	0.214
S2	0.160	0.162	-0.080	-0.022
S3	0.208	0.209	-0.215	-0.128
S4	0.123	0.126	-0.059	0.030
S5	0.185	0.188	0.030	-0.020
S6	0.152	0.147	-0.096	-0.034
S7	0.189	0.182	0.224	0.104
S8	0.151	0.144	0.183	0.163
S9	0.129	0.128	0.240	0.104
S10	0.162	0.162	-0.306	-0.213
S11	0.166	0.171	-0.081	-0.147
S12	0.197	0.202	-0.308	-0.163
S13	0.112	0.111	0.058	0.158
S14	0.088	0.088	0.257	0.181
S15	0.148	0.149	0.269	0.293
S16	0.134	0.138	0.085	-0.062
S17	0.192	0.196	-0.193	-0.047
S18	0.101	0.101	-0.180	-0.190
S19	0.205	0.209	0.001	-0.441
S20	0.136	0.140	-0.059	-0.131
S21	0.126	0.121	0.099	0.141
S22	0.117	0.118	0.124	-0.033
S23	0.070	0.068	0.139	-0.013
S24	0.148	0.143	0.030	-0.049
S25	0.130	0.127	0.086	0.191
S26	0.231	0.235	0.185	0.103
S27	0.111	0.112	0.133	-0.257
S28	0.166	0.175	-0.044	0.012
S29	0.077	0.079	0.048	0.037
S30	0.186	0.190	-0.232	-0.235
S31	0.222	0.221	0.018	0.073
S32	0.199	0.196	0.138	-0.008
S33	0.137	0.129	0.129	0.110
S34	0.194	0.183	-0.112	0.041
S35	0.197	0.193	-0.293	-0.152
S36	0.208	0.208	0.253	0.371
S37	0.109	0.111	-0.081	0.064
S38	0.220	0.222	0.061	0.136

En la **Tabla 5.3.3.5** se muestran los elementos del core $\mathbf{G}$ para el primer sujeto latente etiquetado "CONSENSUS" y en la **Tabla 5.3.3.6** se presentan los elementos del core $\mathbf{G}$ para el segundo sujeto latente etiquetado "CONTRAST", ambos se obtuvieron con el algoritmo CBPSO-TuckALS3.

Tabla 5.3.3.5: Primer corte frontal del core $\mathbf{G}$ con el algoritmo CBPSO-TuckALS3

CONSENSUS	SOOTHING-AGITATING	CHEERFUL-MOURNFUL	DISLIKE-LIKE
FAST+MINOR, SLOW+MAJOR	**53.68358291**	23.28593443	22.1713583
FAST+MAJOR, SLOW+MINOR	-19.52804336	**-42.52677906**	20.9586346

En el primer corte frontal del core $\mathbf{G}$ los dos valores más altos en valor absoluto, los cuales están resaltados en "negritas", permiten afirmar que las interacciones de mayor peso entre los componentes son: primero la del preludio latente "FAST+MINOR, SLOW+MAJOR" con la escala latente "SOOTHING-AGITATING" en el sujeto latente "CONSENSUS", y segundo la interacción del preludio latente "FAST+MAJOR, SLOW+MINOR" con la escala latente "CHEERFUL-MOURNFUL" en el sujeto latente "CONSENSUS". A estas mismas conclusiones se llega usando los elementos del core $\mathbf{G}$ que presentan los autores en su artículo.

Tabla 5.3.3.6: Segundo corte frontal del core $\mathbf{G}$ con el algoritmo CBPSO-TuckALS3

CONTRAST	SOOTHING-AGITATING	CHEERFUL-MOURNFUL	DISLIKE-LIKE
FAST+MINOR, SLOW+MAJOR	-6.204414563	**-5.97027323**	-0.6622205
FAST+MAJOR, SLOW+MINOR	-3.030454311	-8.300481792	**3.55983706**

En el segundo corte frontal del core $\mathbf{G}$ observamos unos valores resaltados en "negritas". En esas ubicaciones se encuentran los valores más altos en valor absoluto según el core $\mathbf{G}$ que presentan los autores en su artículo (lo cual no ocurre con el algoritmo CBPSO-TuckALS3). Pero, como se ha dicho antes, se ha perdido fit en el modelo. A pesar de ello, la salida del algoritmo CBPSO-TuckALS3 nos puede dar una perspectiva en el análisis de los datos que, con otros algoritmos, no podríamos apreciar. Esto concuerda (ver **Tabla 5.3.3.4**) con las diferencias encontradas en el sujeto latente "CONTRAST".

El core $\underline{\mathbf{G}}$ obtenido por el algoritmo CBPSO-TuckALS3, en su segundo corte frontal, nos permite concluir que la interacción más importante es la que ocurre entre el segundo preludio latente "FAST+MAJOR, SLOW+MINOR" con la segunda escala latente "CHEERFUL-MOURNFUL" y en el segundo sujeto latente "CONTRAST".

Los valores de los parámetros del algoritmo CBPSO-TuckALS3 que se usaron para los resultados presentados en este **Experimento # 3**, son:

- $ALSMaxIter = 100$
- $Tol = 10^{-7}$
- $PSOMaxIter = 25$
- $nPar = 50$
- $minIner = 0.8$
- $maxIner = 1.2$
- $wCognition = 2$
- $wSocial = 2$

Se probaron 5 configuraciones diferentes para los valores de los parámetros y esta que se muestra es la que presentó los mejores resultados. Las otras 4 configuraciones utilizadas son las que se observan en la **Tabla 5.3.2.14** del **Experimento # 2**. En Martin-Barreiro et al. 2021 podemos observar algunas áreas de aplicación donde se puede emplear el algoritmo CBPSO-TuckALS3.

5.4 ANÁLISIS DE LOS VALORES ASIGNADOS A LOS PARÁMETROS DE ENTRADA DE LOS ALGORITMOS CBPSO-ParafacALS y CBPSO-TuckALS3

En esta sección se explica el porqué de los valores escogidos para todos los parámetros de entrada en los experimentos computacionales realizados con los algoritmos propuestos: CBPSO-ParafacALS y CBPSO-TuckALS3.

El parámetro $ALSMaxIter$ sirve para indicar el máximo número de iteraciones en el módulo ALS de ambos algoritmos, por lo tanto es utilizado como un criterio de parada. En el caso del algoritmo CBPSO-ParafacALS se experimentó con valores que van desde 8000 iteraciones hasta 12000 iteraciones. En el caso del algoritmo CBPSO-TuckALS3 se experimentó con valores que van desde 50 iteraciones hasta 120 iteraciones. ¿Por qué tanta diferencia entre ambos algoritmos? La razón es que en el modelo PARAFAC existe la posibilidad de tener la presencia del problema de degeneración en los datos (ver **Sección 2.2.2**), mientras que en el modelo Tucker3 no. Como se puede ver en Giordani, Kiers and Del Ferraro 2014, el problema de degeneración consume una cantidad importante de iteraciones.

El parámetro Tol sirve como medida de tolerancia en el fit de dos iteraciones consecutivas de ambos algoritmos y se utiliza también como un criterio de parada del módulo ALS. Si nos encontramos en la iteración n del módulo ALS, y si se satisface la desigualdad $|Fit_n - Fit_{n-1}| < Tol$, entonces se detienen las iteraciones. Por tanto, el módulo ALS finaliza cuando una de las dos condiciones se cumple, ya sea que se alcance la tolerancia deseada o si se alcanza el número máximo de iteraciones. Para ambos algoritmos se experimentó con los siguientes valores: 10^{-5}, 10^{-6}, 10^{-7}, 10^{-8}. Estos son valores muy típicos tanto para el modelo PARAFAC como para el modelo Tucker3 como podemos ver en Smilde et al. 2004 y en Giordani, Kiers and Del Ferraro 2014.

El parámetro $PSOMaxIter$ sirve para controlar el número máximo de iteraciones en el módulo CBPSO de ambos algoritmos, y se usa como único criterio de parada. En Ramirez-Figueroa et al. 2021 se observa que se puede trabajar con valores que van desde unas pocas iteraciones (10 por ejemplo) hasta un número importante de iteraciones (por ejemplo 500). En los experimentos realizados con ambos algoritmos se usaron valores que van desde 20 iteraciones hasta 50 iteraciones. Este parámetro trabaja de forma directa con el parámetro $nPar$ que sirve para indicar el número de partículas que van a realizar la búsqueda del óptimo. Por cada iteración del módulo CBPSO existe $nPar$ número de partículas activas realizando la inspección del espacio de soluciones con el objetivo de encontrar la "mejor" solución. A valores altos del parámetro $PSOMaxIter$ se recomiendan valores bajos del parámetro $nPar$, ya que si ambos parámetros tienen valores altos cada iteración consumirá una cantidad importante de recursos computacionales, por lo que cada iteración resultará muy costosa y esto se verá reflejado en el tiempo de ejecución (un valor alto para el módulo CBPSO está por encima de 50). Otra alternativa es realizar experimentos con valores por

debajo de 50 para ambos parámetros. En los experimentos realizados en esta tesis se usó para el parámetro $nPar$ valores que van desde 20 hasta 50 partículas.

En el algoritmo CBPSO-DC existen tres factores que ayudan a las partículas a moverse por el espacio de búsqueda: (*i*) el factor de inercia, (*ii*) el factor cognitivo y (*iii*) el factor social. Estos factores son los que permiten que la partícula pase a una nueva posición. Los valores escogidos para los parámetros correspondientes, en los experimentos computacionales realizados en esta tesis, son los que usualmente se utilizan en los algoritmos PSO (ver Poli et al. 2007, García-Gonzalo and Fernández-Martínez 2012, Fan and Wang 2017, Nekouie and Moattar 2019). El factor de inercia considera la posición actual de la partícula para poder determinar la nueva posición. En el algoritmo CBPSO-DC la inercia se controla mediante un coeficiente que disminuye en cada iteración, inicia con el valor más alto que se almacena en $maxIner$ (primera iteración) y termina con el valor más bajo que se almacena en $minIner$ (última iteración). Para los experimentos se probaron los siguientes decrementos: $1.5 \rightarrow 0.5$, $1.25 \rightarrow 0.75$, $1.2 \rightarrow 0.8$, $1.75 \rightarrow 0.9$, $2 \rightarrow 1$. Vale indicar que en cada iteración del algoritmo CBPSO-DC todas las partículas dan un paso, el valor de inercia se mantiene para todas las partículas durante una misma iteración.

El factor cognitivo aporta en la determinación de la nueva posición de la partícula mediante el parámetro $wCognition$, el cual es un coeficiente que afecta a la distancia que existe entre la mejor solución encontrada por la actual partícula y la posición de la actual partícula. Los valores que se probaron en los experimentos son los siguientes: 1.5, 1.6, 1.7, 1.8, 1.9, 2. El factor social aporta en la determinación de la nueva posición de la partícula mediante el parámetro $wSocial$, el cual es un coeficiente que afecta a la distancia que existe entre la mejor solución encontrada por el cúmulo total de partículas y la posición de la actual partícula. Los valores que se probaron en los experimentos son los siguientes: 2.5, 2.6, 2.7, 2.8, 2.9, 3.

Los valores de $wSocial$ son ligeramente mayores a los valores de $wCognition$ porque, como parte del diseño del algoritmo CBPSO-DC, la mejor solución encontrada por el cúmulo de partículas influye un poco más en determinar la nueva posición de la partícula que lo que influye la mejor solución encontrada por dicha partícula. Otra consideración importante respecto del diseño del algoritmo CBPSO-DC es el decaimiento lineal de la inercia. Para más detalles del algoritmo CBPSO-DC ver Ramirez-Figueroa et al. 2021.

En la **Sección 5.2** se pueden ver los valores de los parámetros usados en los experimentos con datos simulados. En la **Sección 5.3** se pueden ver los valores de los parámetros utilizados en los experimentos con datos reales.

CONCLUSIONES

Como resultado del trabajo de investigación realizado se ha llegado a las siguientes conclusiones:

1. La implementación de los algoritmos de Vichi and Saporta 2009, Ramirez-Figueroa et al. 2021, ParafacALS y TuckALS3 sirvió como pilar para proponer una metodología que aporta al estudio de tablas de 3 vías. Esta metodología consiste en extender el cálculo de componentes ortogonales disjuntos desde las tablas de 2 vías hasta las tablas de 3 vías, lo que permite obtener matrices de cargas de estructura simple (fáciles de interpretar) en el modelo PARAFAC y en los modelos Tucker. La metodología propuesta se puede utilizar de forma conjunta con otras metodologías existentes.

2. Se han propuesto los siguientes algoritmos heurísticos: **(a)** el algoritmo denominado CBPSO-ParafacALS, para el cálculo de componentes ortogonales disjuntos en el modelo PARAFAC, **(b)** el algoritmo denominado CBPSO-TuckALS3, para el cálculo de componentes ortogonales disjuntos en el modelo Tucker3, **(c)** el algoritmo denominado CBPSO-TuckALS2, para el cálculo de componentes ortogonales disjuntos en el modelo Tucker2, y, **(d)** el algoritmo denominado CBPSO-TuckALS1, para el cálculo de componentes ortogonales disjuntos en el modelo Tucker1.

3. El algoritmo CBPSO-ParafacALS permite resolver el problema de degeneración que suele estar presente en el modelo PARAFAC. Es decir, el algoritmo CBPSO-ParafacALS también permite analizar una tabla de 3 vías con el modelo PARAFAC cuando el problema de degeneración está presente.

4. Respecto del diseño, cada algoritmo heurístico propuesto está basado en una combinación de un algoritmo ALS y el algoritmo CBPSO-DC. La mezcla exitosa de estos algoritmos permitió construir nuevos algoritmos que entregan buenas soluciones, es decir, soluciones útiles en la práctica.

5. En las pruebas computacionales se pudo observar que los algoritmos heurísticos propuestos ejecutan en tiempos de respuesta razonables, pero pueden quedar atrapados en óptimos locales. No hay garantía de alcanzar el óptimo global. En la práctica se recomienda realizar al menos 5 ejecuciones y estudiar la calidad de las soluciones obtenidas.

6. La metodología propuesta facilita la interpretación de las matrices de cargas en el modelo PARAFAC y en los modelos Tucker, pero el costo de utilizarla es la pérdida de fit en el modelo.

7. Se realizaron propuestas metodológicas que facilitan el uso de los algoritmos heurísticos al analista de una tabla de 3 vías. Estas propuestas se diseñaron como algoritmos que proporcionan condiciones bajo las cuales se recomiendan calcular componentes ortogonales disjuntos en tablas de 3 vías.

FUTURAS LÍNEAS DE INVESTIGACIÓN

Trabajos de investigación a futuro que pueden realizarse son los siguientes:

1. Un análisis bootstrap con las entradas de las matrices de cargas calculadas con los algoritmos heurísticos propuestos. Esto con la finalidad de estudiar su variabilidad mediante intervalos de confianza.

2. Los algoritmos heurísticos propuestos calculan una estructura disjunta únicamente para las matrices de cargas. Se sugiere realizar un estudio que permita obtener una estructura disjunta en el core de los modelos Tucker para facilitar su interpretación.

3. Un estudio de los parámetros de entrada de los algoritmos heurísticos propuestos que permita determinar la combinación de valores con los que se obtienen las mejores soluciones.

4. Respecto del tiempo de ejecución, el cuello de botella de los algoritmos heurísticos propuestos es el algoritmo CBPSO-DC. Se sugiere realizar un estudio que permita mejorar esos tiempos de procesamiento.

5. En esta tesis se ha empleado la optimización por enjambre de partículas. Se sugiere un estudio usando otros algoritmos heurísticos como el recocido simulado, la búsqueda dispersa o los algoritmos genéticos.

Las dos primeras líneas planteadas conducen a estudios que permitan precisar los posibles beneficios del uso de los algoritmos heurísticos en el análisis de tablas de tres vías, mientras que las 3 últimas pretenden mejoras computacionales de los algoritmos que se proponen en esta tesis.

REFERENCIAS BIBLIOGRÁFICAS

Amaya J, Pacheco P (2002). “Análisis Factorial Dinámico mediante el Método Tucker3”. Revista Colombiana de Estadística, **25**, 43-57.

Borckmans P, Ishteva M, Absil P (2010). "A Modified Particle Swarm Optimization Algorithm for the Best Low Multilinear Rank Approximation of Higher- Order Tensors". Lecture Notes in Computer Science, **6234**, 13-23.

Borg I, Groenen PJF (2005). Modern Multidimensional Scaling: Theory and Applications. Springer Series in Statistics, New York, USA.

Bro R (1998). Multi-Way Analysis in the Food Industry - Models, Algorithms and Applications. Ph.D. thesis, University of Amsterdam.

Bro R, Smilde AK (2003). "Centering and Scaling in Component Analysis". Journal of Chemometrics, **17**, 16-33.

Carroll JD, Chang JJ (1970). "Analysis of Individual Differences in Multidimensional Scaling via an *n*-Way Generalization of Eckart-Young Decomposition". Psychometrika, **35**, 283-319.

Ceulemans E, Kiers HAL (2006). "Selecting among Three-Mode Principal Component Models of Different Types and Complexities: A Numerical Convex Hull Based Method". British Journal of Mathematical and Statistical Psychology, **59**, 133-150.

Cuadras CM (2014). "Nuevos Métodos de Análisis Multivariante". CMC Editions, Barcelona, Spain.

Eckart C, Young G (1936). "The Approximation of One Matrix by Another of Lower Rank". Psychometrika, **1**, 211-218.

Eckart C, Young G (1939). "A Principal Axis Transformation for Non-Hermitian Matrices". Am.Math.Soc.Bull, **45**, 118-121.

Fan J, Wang J (2017). "A Collective Neurodynamic Optimization Approach to Nonnegative Tensor Decomposition". Advances in Neural Networks, **10262**, 207-213.

Ferrara C, Martella F, Vichi M (2016). "Dimensions of Well-Being and Their Statistical Measurements". Topics in Theoretical and Applied Statistics, 85-99.

García-Gonzalo E, Fernández-Martínez J (2012). "A Brief Historical Review of Particle Swarm Optimization". Journal of Bioinformatics and Intelligent Control, **1**, 3-16.

Giordani P, Kiers HAL, Del Ferraro MA (2014). "Three-Way Component Analysis Using the *R* Package". Journal of Statistical Software, **57**(7), 1-23.

Giordani P, Rocci R (2013). "Constrained Candecomp/Parafac via the Lasso". Psychometrika, **78**, 669-684.

Goossens M, Mittelbach F, Samarin A (2002). "Principal Component Analysis". Encyclopedia of Statistics in Behavioral Science, Second Edition, Volumen 30.

Harshman RA (1970). "Foundations of the Parafac Procedure: Models and Conditions for an 'Explanatory' Multimodal Factor Analysis". UCLA Working Papers in Phonetics, **16**, 1-84.

Harshman RA, Lundy ME (1984). "Data Preprocessing and the Extended PARAFAC Model". In HG Law, CWS Jr, JA Hattie, RP McDonald (eds.), Research Methods for Multimode Data Analysis, pp. 216-284. Praeger, New York.

Henrion R (1994). "N-Way Principal Componente Analysis: Theory, Algorithms and Applications". Chemometrics and Intelligent Laboratory Systems, **25**, 1-23.

Hitchcock FL (1927). "The Expression of a Tensor or a Polyadic as a Sum of Products". Journal of Mathematics and Physics, **6**, 164-189.

Hotelling H (1933). "Analysis of a Complex of Statistical Variables Into Principal Components". J. Educ. Psychol., **24**, 417-441.

Husson F, Pages J (2006). "INDSCAL Model: Geometrical Interpretation and Methodology". Computational Statictics & Data Analysis, **50**, 358-378.

Kennedy J, Eberhart R (1995). "Particle Swarm Optimization". Proceedings of ICNN95 - International Conference on Neural Networks, **4**, 1942-1948.

Kiers HAL (1998). "Joint Orthomax Rotation of the Core and Component Matrices Resulting from Three-Mode Principal Component Analysis". Journal of Classification, **15**, 245-263.

Kiers HAL (2000). "Towards a Standardized Notation and Terminology in Multiway Analysis". Journal of Chemometrics, **14**, 105-22.

Kiers HAL, Van Mechelen I (2001). "Three-Way Component Analysis: Principles and Illustrative Applications". Psychological Methods, **6**, 84-110.

Kolda TG (2001). "Orthogonal Tensor Decompositions". SIAM Journal on Matrix Analysis and Applications. **23**(1), 243-255.

Kolda TG, Bader BW (2009). "Tensor Decompositions and Applications". SIAM Review, **51**(3), 455-500.

Kroonenberg PM (1983). Three-Mode Principal Component Analysis: Theory and Applications. DSWO, Leiden.

Kroonenberg PM (1984). Three-Mode Principal Component Analysis: Illustrated with an Example from Attachment Theory, in Research Methods for Multimode Data Analysis, Law HG, Snyder CW, Hattie JA and McDonald RP (eds), Praeger, New York, pp 64–103.

Kroonenberg PM, de Leeuw J (1980). "Principal Component Analysis of Three-Mode Data by Means of Alternating Least Squares Algorithms", Psychometrika, **45**, 69-97.

Kroonenberg PM (2008). “Applied Multiway Data Analysis”. Wiley, New York, USA.

Kruskal JB (1964a). "Multidimensional Scaling by Optimizing Goodness of Fit to a NonMetric Hypothesis". Psychometrika, **29**, 1-27.

Kruskal JB (1964b). "Nonmetric Multidimensional Scaling: A Numerical Method". Psychometrika, **29**, 115-129.

Kruskal JB (1976). "More Factors than Subjects, Tests and Treatments: An Indeterminacy Theorem for Canonical Decomposition and Individual Differences Scaling". Phychometrika, **41**, 281-293.

Kruskal JB (1977). "Three-Way Arrays: Rank and Uniqueness of Trilinear Decompositions, with Applications to Arithmetic Complexity and Statistics". Linear Algebra and Its Applications, **18**, 95-138.

Kruskal JB (1989). Rank, Decomposition, and Uniqueness for 3-Way and N-Way Arrays, in Multiway Data Analysis, Coppi R, Bolasco S (Eds), Elsevier, Amsterdam, pp. 8-18.

Leibovici DG (2010). "Spatio-Temporal Multiway Decompositions Using Principal Tensor Analysis on *k*-Modes: The *R* Package PTAk". Journal of Statistical Software, **34**(10), 1-34.

Levin J (1965). "Three-Mode Factor Analysis". Psychological Bulletin, **64**, 442-452.

Lundy M, Harshman R, Kruskal J (1989). A Two Stage Procedure Incorporating Good Features of Both Trilinear and Quadrilinear Models. In Coppi R, Bolasco S (eds.), *Multiway Data Analysis*, pp. 123-130. Elsevier, Amsterdan.

Macedo E, Freitas A (2015). "The Alternating Least Squares Algorithm for CDPCA". Optimization in the Natural Sciences, Springer International Publishing, 173-191.

Magnus JR, Neudecker H (1988). Matrix Differential Calculus with Applications in Statistics and Econometrics, John Wiley & Sons, Chichester.

Martin-Barreiro C, Ramirez-Figueroa JA, Nieto-Librero AB, Leiva V, Martin-Casado A, Galindo-Villardón MP (2021). “A New Algorithm for Computing Disjoint Orthogonal Components in the Three-Way Tucker Model”. MDPI Mathematics, **9**(3), 203.

Murakami T, Kroonenberg PM (2003). “Three-Mode Models and Individual Differences in Semantic Differential Data”. Multivariate Behavioral Research, **38**(2), 247-283.

Nekouie A, Moattar M (2019). "Missing Value Imputation for Breast Cancer Diagnosis Data Using Tensor Factorization Improved by Enhanced Reduced Adaptive Particle Swarm Optimization". Journal of King Saud University - Computer and Information Sciences, **31**(3), 287-294.

Papalexakis E, Faloutsos C, Sidiropoulos N (2012). "Parcube: Sparse Parallelizable Tensor Decompositions". Annalen der Physik, 521-536.

Pearson K (1901). "On Lines and Planes of Closest Fit to Points in Space". Philos. Mag., **2**, 559-572.

Perros I, Chen R, Vuduc R, Sun J (2015). "Sparse Hierarchical Tucker Factorization and Its Application to Healthcare". IEEE International Conference on Data Mining, 943-948.

Poli R, Kennedy J, Blackwell T (2007). "Particle Swarm Optimization". Swarm Intelligence, **1**, 33-57.

Ramirez-Figueroa JA, Martin-Barreiro C, Nieto-Librero AB, Leiva V, Galindo-Villardón MP (2021). “A new principal component analysis by particle swarm optimization with an environmental application for data science”. Stochastic Environmental Research and Risk Assessment, **1**, 1-16.

Rocci R, Giordani P (2010). "A Weak Degeneracy Decomposition for the CANDECOMP/PARAFAC Model". Journal of Chemometrics, **24**, 57-66.

Sánchez E, Kowalski BR (1990). "Tensorial Resolution: A Direct Trilinear Decomposition". Journal of Chemometrics, **4**, 29-45.

Schott JR (1997). Matrix Analysis for Statistics. John Wiley & Sons, Inc., New York.

Shepard RN (1962). "The Analysis of Proximities: Multidimensional Scaling with an Unknown Distance Function. Part I, Part II". Psychometrika, **27**, 125-140, 219-246.

Silva VD, Lim LH (2008). "Tensor Rank and the ILL-POSEDNESS of the Best Low-Rank Approximation Problem". SIAM Journal on Matrix Analysis and Applications, **30**, 1084-1127.

Smilde A, Bro R, Geladi P (2004). Multi-way Analysis with Applications in the Chemical Sciences. John Wiley & Sons, Chichester.

Stegeman A (2006). "Degeneracy in Candecomp/Parafac Explained for p x p x 2 Arrays of Rank p + 1 or Higher". Psychometrika, **71**, 483-501.

Stegeman A (2007). "Degeneracy in Candecomp/Parafac and Indscal Explained for Several Three-Sliced Arrays with a Two-Valued Typical Rank". Psychometrika, **72**, 601-619.

Sun WW, Junwei L, Han L, Guang C (2017). "Provable Sparse Tensor Decomposition". Journal of the Royal Statistical, **79**(3), 899-916.

Ten Berge JMF (1993). Least Squares Optimization in Multivariate Analysis. DSWO Press, Leiden.

Ten Berge JMF, Kiers HAL (1996). "Optimality Criteria for Principal Components Analysis and Generalizations". British Journal of Mathematical and Statistical Psychology, **49**, 335-345.

Ten Berge JMF, Kiers HAL (1997). "Are All Variaties of PCA the same? A Reply to Cadima & Jolliffe". British Journal of Mathematical and Statistical Psychology, **50**, 368.

Ten Berge JMF, Kiers HAL, Krijnen WP (1993). "Computational Solutions for the Problem of Negative Saliences and Nonsymmetry in INDSCAL". Journal of Classification, **10**, 115-124.

Timmerman ME, Kiers HAL (2000). "Three-Mode Principal Component Analysis: Choosing the Numbers of Components and Sensitivity to Local Optima". British Journal of Mathematical and Statistical Psychology, **53**, 1-16.

Tomasi G, Bro R (2006). "A Comparison of Algorithms for Fitting the Parafac Model". Computational Statistics & Data Analysis, **50**, 1700-1734.

Torgerson WS (1952). "Multidimensional Scaling: I. Theory and Methods". Psychometrika, **17**, 401-419.

Tucker LR (1963). "Implications of Factor Analysis of Three-Way Matrices for Measurement of Change". University of Wisconsin Press, Madison WI, 122-137.

Tucker LR (1966). "Some Mathematical Notes on Three-Mode Factor Analysis". Psychometrika, **31**, 279-311.

Vichi M, Saporta G (2009). "Clustering and Disjoint Principal Component Analysis". Computational Statistics and Data Analysis, **53**(8), 3194-3208.

Weesie J, Houwelingen HV (1983). "GEPCAMP User's Manual". Utrecht, The Netherlands: Institute of Mathematical Statistics, State University of Utrecht.

Wilderjans TF, Ceulemans E, Meers K (2013). "A Generic Convex-Hull-Based Model Selection Method". Behavior Research Methods, **45**, 1-15.

Yokota T, Cichocki A (2014). "Multilinear Tensor Rank Estimation via Sparse Tucker Decomposition". 2014 Joint 7th International Conference on Soft Computing and Intelligent Systems, and 15th International Symposium on Advanced Intelligent Systems.

Young G, Householder AS (1938). "Discussion of a Set of Points in Terms of their Mutual Distances". Psychometrika, **3**, 19-22.

Zou H, Hastie T, Tibshirani R (2006). “Sparse Principal Component Analysis”. Journal of Computational and Graphical Statistics. **15**(2), 265-286.