Autor: Juan Francisco Rodríguez Herrera
Título: Enfoque multihebrado del algoritmo de resolución del problema de bi-mezcla
Fecha: Septiembre de 2011
Director: Dr. D. Leocadio González Casado

Enfoque multihebrado del algoritmo de resolución del problema de bi-mezcla

Juan F. R. Herrera[1] y Leocadio G. Casado[2]

Resumen— **Los algoritmos de diseño de mezclas tienen como objetivo determinar las mezclas de ingredientes que se ajustan a las restricciones de diseño impuestas para el producto en cuestión. Estas restricciones pueden ser lineales y/o cuadráticas. Se quiere minimizar el coste y el número de ingredientes utilizados, nótese que con más ingredientes se puede obtener menor coste. Los fabricantes elaboran una serie de productos a partir de un conjunto dado de materias primas. La escasez en la disponibilidad de estas materias primas introduce restricciones de disponibilidad que alteran la solución Pareto-óptima. Los autores han desarrollado algoritmos de Ramificación y Acotación para resolver problemas de mezcla en donde la complejidad computacional se incrementa con el número de posibles ingredientes para elaborar el producto. Debido a esta complejidad, se abordará el problema de mezcla para la obtención de sólo dos productos.**

El diseño de mezclas para dos productos es más difícil que para un único producto porque además de que el diseño de cada producto está sometido a unas restricciones, el frente de Pareto así como la disponibilidad de materias primas pasan a ser comunes a ambos productos. Se debe realizar una combinación final entre todas las soluciones encontradas del primer y el segundo producto para eliminar las combinaciones de mezclas que no satisfacen los criterios impuestos. El conjunto resultante puede ser usado como dato de entrada del mismo algoritmo cuando se requieran resultados más precisos. El coste computacional de la fase de combinación dependerá del número de elementos del conjunto final de cada producto.

Aquí, estudiaremos el coste computacional de las diferentes fases del algoritmo de mezcla para dos productos y proporcionaremos dos versiones hebradas para la fase más costosa. Los experimentos se han realizado en una máquina de ocho núcleos con memoria compartida, usando un problema de tamaño medio para evitar largos tiempos de ejecución. Los experimentos muestran que la computación paralela es una herramienta necesaria para hacer una búsqueda exhaustiva en problemas de grandes dimensiones y de más de un producto.

Palabras clave— **Memoria compartida, Procesamiento paralelo , Multihebrado, Ramificación y Acotación, Optimización Global.**

I. Introducción

En esta primera sección se introducirá el concepto de Optimización Global, concepto al que después se hará referencia en posteriores secciones. También se describirá el algoritmo de Ramificación y Acotación, una de las técnicas utilizadas para dar solución al problema de la Optimización Global.

A. Optimización Global

Uno de los principios más fundamentales en nuestro mundo es la búsqueda del valor óptimo. Muchos avances recientes en campos tales como la Ciencia, la Economía o la Ingeniería dependen de técnicas numéricas para calcular las mejores soluciones globales en problemas de optimización. El objetivo de la Optimización Global es encontrar la mejor solución global de un modelo (posiblemente no lineal) en presencia (o no) de múltiples óptimos locales, aunque puede que tal solución no exista. Modelos no lineales se encuentran presentes en muchas aplicaciones tales como avanzados diseños de ingeniería, biotecnología, análisis de datos, gestión medioambiental, planificación financiera, control de procesos, gestión del riesgo, modelado científico, etc.

La formulación del problema de Optimización Global tiene como forma general

$$\begin{array}{ll} \text{minimizar o maximizar} & f(x) \\ \text{sujeto a} & x \in D \end{array}$$

donde $D \subset \mathbb{R}^n$ es el dominio de factibilidad y $f : \mathbb{R}^n \to \mathbb{R}$ es la función objetivo.

En problemas de minimización, se dice que un punto $x^\star \in D$ es un mínimo local si $f(x^\star) \leq f(x)$ para todo $x \in D$ que satisfaga que $\|x - x^\star\| \leq \varepsilon$, siendo $\varepsilon > 0$ y $\|\cdot\|$ la norma euclídea. Decimos que $x^\star$ es un mínimo global si $f(x^\star) \leq f(x),\ \forall x \in D$.

Asimismo, en problemas de maximización, se dice que un punto $x^\star \in D$ es un máximo local si $f(x^\star) \geq f(x)$ para todo $x \in D$ que satisfaga que $\|x - x^\star\| \leq \varepsilon$, siendo $\varepsilon > 0$. Decimos que $x^\star$ es un máximo global si $f(x^\star) \geq f(x),\ \forall x \in D$.

La figura 1 ilustra una función f definida en un espacio bidimensional $\mathrm{X} = (X_1, X_2)$. Como se indica en la figura, se distinguen los óptimos locales de los óptimos globales. Un óptimo global es un óptimo de todo el dominio X, mientras que un óptimo local es un óptimo de sólo un subconjunto de X.

Un ejemplo es la búsqueda del punto más alto de la Tierra sobre el nivel del mar. Un óptimo local en el continente europeo será el monte Elbrus, en el continente africano será el Kilimanjaro y en el continente asiático el monte Everest. El óptimo global, sin embargo, es el monte Everest, ya que es el punto más alto de la superficie terráquea. Se puede ver que el óptimo local para Asia es también el óptimo global para la Tierra. Esto muestra que un óptimo global pertenece al conjunto de los óptimos locales.

A partir de ahora, el problema de Optimización Global será considerado de la forma

$$\min_{x \in D} f(x), \tag{1}$$

donde $D = \{x \in \mathbb{R}^n : g_i(x) \leq 0,\ i = 1, \ldots, p\}$ con restricciones $g_i : A \to \mathbb{R}$ en un conjunto $A \supseteq D$,

[1]Dpto. de Arquitectura de Computadores y Electrónica, Univ. Almería, e-mail: juanfrh@ual.es.

[2]Dpto. de Arquitectura de Computadores y Electrónica, Univ. Almería, e-mail: leo@ual.es.

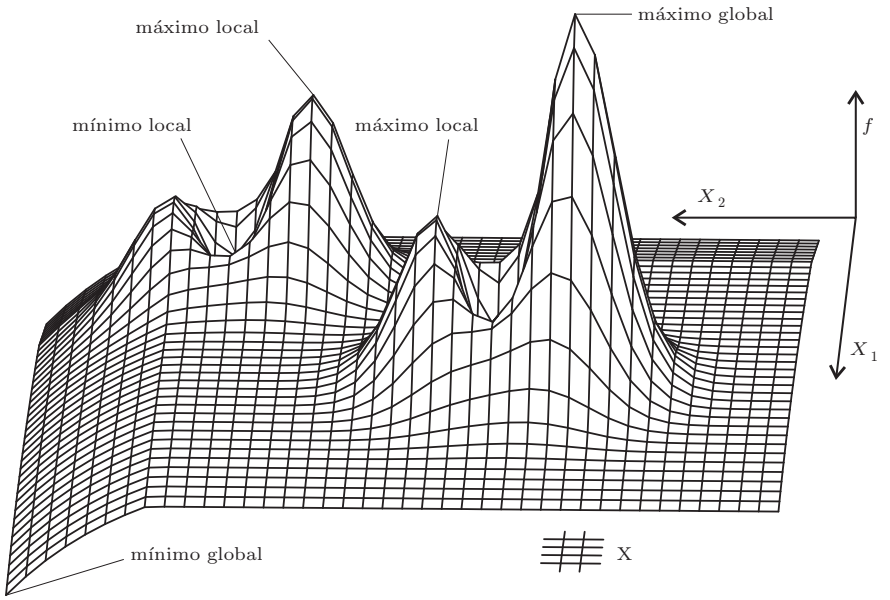


Fig. 1: Óptimos locales y globales en una función con dos variables

siendo a menudo $A = \mathbb{R}^n$. Los problemas de maximización también están considerados en (1) porque $\max\{f(x) : x \in D\} = -\min\{f(x) : x \in D\}$. Además, ya que $g_i(x) \geq 0$ es equivalente a $-g_i(x) \leq 0$, y $g_i(x) = 0$ es equivalente a $g_i(x) \leq 0$ y $-g_i(x) \leq 0$, se observa que la definición (1) considera muchos otros tipos de restricciones.

Si todas las funciones definidas anteriormente son continuas y $D \neq \emptyset$, el conjunto de soluciones óptimas para el problema (1) es no vacío. Para resolver (1), la mayoría de las investigaciones realizadas se han enfocado al caso especial en el que la estructura del modelo matemático tiene una serie de características, con lo que se puede establecer que sólo existe una única solución y, por lo tanto, un único mínimo, que es a su vez local y global. Esto se cumple, por ejemplo, si f es una función convexa y D es un conjunto convexo. Los métodos desarrollados para problemas convexos usan solamente información local. Con la información de uno o más puntos, se construye una aproximación de la solución del problema original, la cual se usa para calcular un nuevo punto de prueba en la siguiente iteración. Estos métodos garantizan la convergencia al mínimo global si se da la propiedad de convexidad. De otra forma, dicha convergencia no está asegurada.

Aunque muchos tipos de problemas pertenecen a la clase anterior, existe una amplia variedad de problemas donde la existencia de un solo mínimo no puede ser postulada o verificada, con lo que su resolución se vuelve más difícil.

Si la región de factibilidad está delimitada por restricciones lineales y la función objetivo también es lineal, el problema se puede resolver utilizando programación lineal. Por el contrario, si alguna función involucrada en el problema no es lineal, la programación lineal no se puede aplicar como método de resolución. Todas las técnicas en optimización no lineal pueden al menos localizar óptimos locales; sin embargo, no existe un criterio para decidir si una solución es global. El hecho de que el problema de optimización no sea lineal, o no sea convexo, conlleva la posible existencia de múltiples óptimos locales. Por lo general, el número de óptimos locales es desconocido y puede ser bastante grande. Además, la calidad entre los óptimos locales y los óptimos globales puede diferir significativamente. Debido a esto, las soluciones locales no son válidas en la mayoría de los casos. Incluso para diferencias pequeñas entre los valores en los óptimos locales y globales, el hecho de no encontrar el óptimo global puede traducirse, especialmente en aplicaciones económicas, en millones de euros de pérdidas. Por lo tanto, la Optimización Global puede llegar a ser extremadamente importante.

Siguiendo con la analogía de la búsqueda del punto más alto de la Tierra, si se busca en Europa y se determina que el punto más alto de dicho continente es el monte Elbrus, se podría pensar que es el punto más alto del mundo, ya que en sus alrededores no existe ninguna montaña más alta que el monte Elbrus.

Una manera de obtener los mínimos globales consiste en determinar todos los mínimos locales y de ellos quedarse con los mejores; pero esta aproximación es impracticable, debido a que muchos problemas se caracterizan por tener un gran número de mínimos locales. Además, incluso la determinación de un mínimo local no siempre es fácil. La mayoría de las aproximaciones clásicas no pueden aplicarse directamente para resolver estos problemas, dificultando así la resolución de los mismos.

Naturalmente, ante tales circunstancias, es esencial utilizar una estrategia de búsqueda global apropiada. Además, en lugar de soluciones «exactas», normalmente se tienen que aceptar diversas aproximaciones numéricas a la solución óptima global o al conjunto de éstas.

Estas estructuras de mínimos pueden dar lugar a representaciones parecidas a paisajes montañosos. A modo de ejemplo, véase la figura 2, que representa gráficamente una, relativamente simple, composición de funciones trigonométricas con argumentos polinomiales $f(x,y) = 0.2 \times (\sin(x+4y) - 2 \times \cos(2x+3y) - 3 \times \sin(2x-y) + 4 \times \cos(x-2y))$ sobre un espacio de búsqueda bidimensional. La función posee múltiples mínimos locales.

Los primeros y esporádicos trabajos sobre Optimización Global surgieron a finales de los cincuenta. La evolución desde entonces ha sido muy grande, de forma que el *estado del arte* en la actualidad se caracteriza por varias decenas de monográficos, una revista internacional (*Journal of Global Optimization*) y varios miles de artículos de investigación dedicados exclusivamente a este tema.

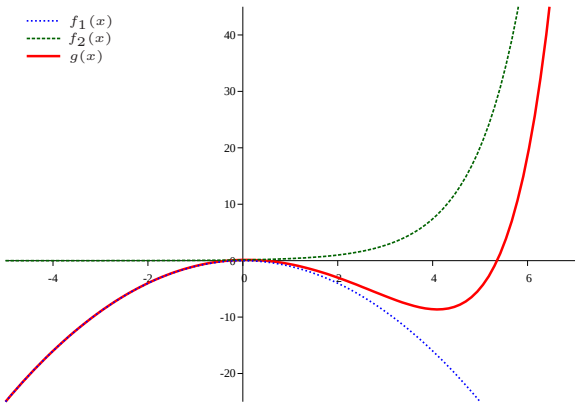


Fig. 3: Dos funciones no aptas para la suma ponderada

A.1 Número de criterios

Los problemas de optimización pueden ser divididos en los que intentan encontrar la solución óptima de una función objetivo y en los que optimizan un conjunto F de funciones objetivo. A estos segundos problemas se les conoce como problemas de optimización multiobjetivo.

En el caso de optimizar una única función objetivo f, un óptimo es su máximo o su mínimo global, dependiendo de lo que se esté buscando.

Las técnicas de Optimización Global no se utilizan solamente para encontrar el máximo o el mínimo de una función f. En muchos problemas de diseño o de toma de decisiones, dichas técnicas son aplicadas a un conjunto F con $p = |F|$ funciones objetivo f_i, cada una representando un criterio a ser optimizado:

$$F = \{f_i : \mathbb{R}^n \to \mathbb{R};\ i = 1, \ldots, p\}.$$

Método de la suma ponderada

El método más simple para hallar el óptimo en un problema de optimización multiobjetivo es calcular una suma ponderada $g(x)$ de todas las funciones $f_i \in F$. Cada función objetivo f_i es multiplicada por un peso w_i que representa su importancia. Usar pesos con signo permite minimizar una función objetivo y maximizar otra. Se puede, por ejemplo, aplicar un peso $w_a = 1$ a una función objetivo f_a y el peso $w_b = -1$ a f_b. Al minimizar $g(x)$, se minimiza f_a y se maximiza f_b. Si se maximiza $g(x)$, el efecto sería el inverso: f_b sería minimizada y f_a sería maximizada. De esta forma, los problemas multiobjetivo pueden ser reducidos a problemas con un único objetivo:

$$g(x) = \sum_{i=1}^{p} w_i f_i(x) = \sum_{\forall f_i \in F} w_i f_i(x).$$

Este método tiene el inconveniente de que no es válido para funciones que crecen o decrecen a un ritmo diferente. En la figura 3 se representa la suma $g(x)$ de las dos funciones objetivo $f_1(x) = -x^2$ y $f_2(x) = e^{x-2}$. Cuando se minimiza o maximiza la función $g(x)$, siempre se ignora una de las dos funciones, dependiendo del intervalo elegido. Para valores pequeños de x, f_2 es insignificante comparada con f_1. Para valores de x mayores que 5, f_2 comienza superando a f_1, que ahora se vuelve insignificante. Estas funciones no pueden ser sumadas adecuadamente usando pesos constantes. Por lo tanto, la suma ponderada es apropiada para optimizar funciones que al menos compartan la misma cota superior asintótica. A menudo, no es obvio cómo crecen o decrecen las funciones objetivo. Aun cuando la forma de las funciones objetivo y su cota superior asintótica son claras, la cuestión de cómo establecer los pesos adecuadamente sigue abierta en muchos casos [1]. En [1], también se muestra que utilizando sumas ponderadas no se encuentran necesariamente todos los elementos considerados óptimos en términos de la dominancia de Pareto.

Optimalidad de Pareto

Los fundamentos matemáticos para la optimización multiobjetivo que consideran criterios en conflicto de una manera equitativa fueron establecidos por Vilfredo Pareto a finales del siglo XIX. La optimalidad de Pareto se convirtió en un concepto importante en la Economía, la Teoría de Juegos, la Ingeniería y las Ciencias Sociales.

Un elemento x_1 se dice que domina a un elemento x_2 si x_1 es mejor que x_2 en al menos una función objetivo y no es peor con respecto al resto de funciones objetivo. Un elemento $x^\star$ se dice que es Pareto-óptimo si no es dominado por ningún otro elemento, es decir, si no existe algún otro elemento que haga mejorar una de las funciones objetivo sin que empeore de forma simultánea alguna de las otras. En general, la solución al problema de la optimización multiobjetivo no será única: la solución estará formada por el conjunto de todos los vectores no dominados, a los que se conoce con el nombre de conjunto, frente o frontera de Pareto.

A.2 Métodos de resolución

Las cualidades deseadas en un algoritmo de Optimización Global se encuentran en la siguiente lista:

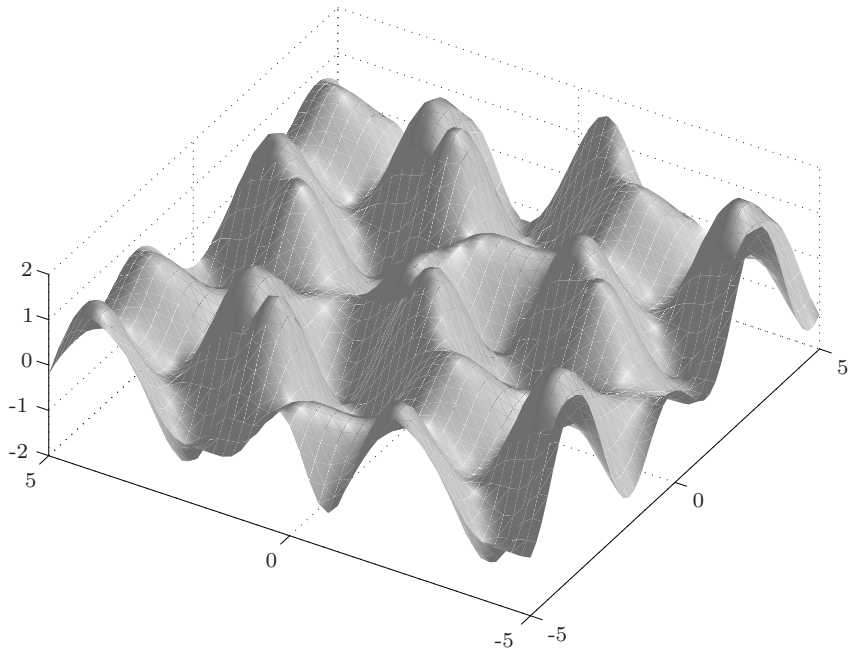


Fig. 2: Representación en 3D de una función con múltiples mínimos locales

- Corrección: no producir resultados incorrectos.
- Completitud: encontrar todas las posibles soluciones.
- Finitud: garantizar la convergencia.
- Certeza: probar la existencia o inexistencia de soluciones.

Sólo unas pocas implementaciones existentes garantizan todas las cualidades mostradas.

No existe un método universal de resolución de problemas de optimización, eligiéndose uno u otro dependiendo de las características del problema, de los requerimientos de calidad en los resultados y del tiempo de respuesta del algoritmo. Una clasificación general de los algoritmos de optimización hace una distinción entre algoritmos deterministas y probabilistas.

Los algoritmos deterministas no toman decisiones aleatorias en sus algoritmos y sus demostraciones de convergencia no dependen de la probabilidad, al contrario que los algoritmos probabilistas. Algunos ofrecen un tiempo de terminación finito para problemas específicos y otros convergen cuando el número de iteraciones se aproxima a infinito. Se caracterizan porque varias ejecuciones del mismo algoritmo realizan la misma computación y obtienen los mismos resultados. Se utilizan con mayor frecuencia si existe una definición matemática del problema. Dentro de los algoritmos deterministas, los algoritmos de Ramificación y Acotación realizan una búsqueda exhaustiva del espacio de búsqueda del problema.

Si el problema no está definido analíticamente o las dimensiones del espacio de búsqueda son muy extensas, se hace más difícil resolver el problema de forma determinista. En espacios relativamente grandes, la búsqueda exhaustiva sería computacionalmente impracticable. Aquí es donde los algoritmos probabilistas entran en juego. Los primeros trabajos en este área, que ahora se ha convertido en uno de los campos de investigación más importantes en la optimización, se iniciaron a mediados del siglo XX.

Todos los métodos probabilistas usan algún factor aleatorio en sus algoritmos y la demostración de su convergencia depende de argumentaciones estadísticas. Normalmente, los métodos probabilistas se aplican a problemas sin restricciones y no necesitan ningún conocimiento sobre la función objetivo, dando a menudo resultados óptimos o cercanos a éstos. Se basan en obtener valores de la función objetivo en puntos de la región de búsqueda elegidos aleatoriamente, pero de forma guiada. Las desventajas de estos métodos es que no garantizan que se encuentre la solución en un número finito de pasos, ni que el mínimo encontrado sea el global.

Una familia de especial relevancia de algoritmos probabilistas son los algoritmos basados en el método de Montecarlo. Dan una solución aproximada a cambio de un tiempo de ejecución más corto. Esto no quiere decir que los resultados obtenidos sean incorrectos, sólo que no se garantiza que sean los óptimos globales. Por otra parte, una solución un poco inferior a la óptima es mejor que una que necesite mucho tiempo ser encontrada.

Las heurísticas utilizadas en Optimización Global son esquemas que ayudan a decidir qué parte de un conjunto de posibles soluciones es examinado a continuación. Por un lado, los algoritmos deterministas suelen emplear heurísticas para definir el orden de procesamiento de los candidatos a solución, pero todos son evaluados. Los métodos probabilísticos, por otro lado, consideran sólo aquellos elementos del espacio de búsqueda que hayan sido seleccionados heurísticamente.

Los diferentes algoritmos también pueden ser clasificados de acuerdo al grado de rigor con que se en-

cuentra el óptimo global [2]:

- Un método incompleto utiliza heurísticas intuitivas para la búsqueda pero es susceptible de que se quede atascado en un óptimo local.
- Un método asintóticamente completo alcanza un óptimo global con certeza (o por lo menos con probabilidad uno) si se ejecuta durante un periodo de tiempo indefinidamente largo, pero no tiene medios para saber si un óptimo global ha sido encontrado.
- Un método completo alcanza un óptimo global con certeza, suponiendo cálculos exactos y un tiempo de ejecución indefinidamente largo, y sabe después de un tiempo finito que se ha encontrado una solución global aproximada.
- Un método riguroso alcanza un óptimo global con certeza incluso en la presencia de errores de redondeo.

A menudo, las dos últimas categorías de algoritmos se caracterizan como deterministas; sin embargo, esta caracterización es ligeramente confusa, ya que muchos métodos incompletos y asintóticamente completos son deterministas también.

Los métodos completos (y con mayor razón los rigurosos) garantizan (en aritmética exacta) encontrar el óptimo global con una cantidad de trabajo previsible en función de las características del problema. El límite de la cantidad de trabajo suele ser muy pesimista, exponencial en muchos casos. Es sólo una garantía débil que no asegura que el algoritmo sea eficiente en ningún sentido, sino que garantiza la ausencia de deficiencias sistemáticas que impidan encontrar, en definitiva, un óptimo global.

El método completo más simple para los problemas con restricciones es la búsqueda en rejilla (del inglés *grid search*), donde el espacio de búsqueda se divide en forma de rejilla y cada punto es analizado en búsqueda de un óptimo global. Dado que el número de puntos en una rejilla crece exponencialmente con la dimensión, la búsqueda en rejilla es eficaz sólo cuando el número de dimensiones del problema es pequeño. Métodos completos más eficientes generalmente combinan técnicas de Ramificación y Acotación con una o varias técnicas de optimización local: Análisis Convexo, Análisis de Intervalos y Programación con Restricciones.

En general, los métodos completos (incluidos los métodos de aproximación que reducen el problema a un tratamiento por métodos completos) son más fiables que los métodos incompletos.

Una buena heurística junto con elecciones probabilistas (similares, pero por lo general más simples que las de los métodos incompletos) también juegan un papel importante en los métodos completos, principalmente para proporcionar de forma asequible buenos puntos factibles en beneficio de la búsqueda completa.

Centraremos nuestra atención en los métodos rigurosos, particularmente en los métodos de Ramificación y Acotación, que son los más utilizados entre los métodos completos y los métodos rigurosos para problemas no lineales.

B. Algoritmos de Ramificación y Acotación

Los primeros algoritmos de Ramificación y Acotación (también llamados algoritmos de «Ramificación y Poda») aparecieron en artículos de los cincuenta y primeros de los sesenta, donde los investigadores describieron esquemas enumerativos para resolver, lo que posteriormente se denominaron, problemas NP-completos. Debido a la generalidad del método y a la efectividad que aportaba, fue ampliamente usado. De hecho, todavía es una de las mejores herramientas para resolver problemas difíciles. Los primeros que dieron el nombre de «Ramificación y Acotación» (*Branch and Bound*, B&B) a este método fueron Little, Murty, Sweeny y Karel [3] (1963), en su artículo innovador sobre el problema del viajante de comercio. Lawler y Wood [4] estudiaron los algoritmos de Ramificación y Acotación, obteniendo una descripción independiente del problema, siendo así el primer artículo que presenta un modelo general de Ramificación y Acotación.

Los métodos de Ramificación y Acotación son algoritmos basados en árboles de búsqueda. La idea básica del algoritmo B&B consiste en descomponer recursivamente el problema original en subproblemas disjuntos hasta que la solución óptima es encontrada. El método evita visitar aquellos subproblemas en los que se conoce que no se va a encontrar una solución.

B.1 Reglas básicas

Según Mitten e Ibaraki [5], [6], un algoritmo B&B, generalmente, consiste en cuatro partes o reglas: regla de ramificación, regla de acotación, regla de selección y regla de eliminación. Dependiendo del tipo de problema, se añade o no la regla de finalización. La regla de selección depende del algoritmo de búsqueda que se quiera aplicar, las restantes reglas dependen del problema a resolver. Llamaremos a estas reglas las «reglas básicas» de los algoritmos de Ramificación y Acotación.

Una buena comprensión de la estructura del problema a resolver, es decir, hacer una buena elección de las reglas básicas, da lugar a una reducción del tiempo de ejecución y a poder resolver problemas más complejos en cuanto a su tamaño.

Regla de ramificación

La regla de ramificación (*branch*) nos indica el método de división de los subproblemas, que pueden ser divididos en dos o más subproblemas disjuntos.

Regla de acotación

La regla de acotación (*bound*) calcula el límite inferior del valor de la solución óptima de un subproblema dado, en problemas de minimización.

Regla de selección

La regla de selección es un factor importante para el rendimiento del algoritmo diseñado. No afecta a

la convergencia del algoritmo, pero sí a su eficacia y al espacio de memoria requerido [7]. Define cuál subproblema va a ser el próximo en ser dividido. El recorrido que se haga en el árbol de búsqueda depende del criterio de selección elegido. Los cuatro criterios de selección más importantes son la búsqueda en anchura, la búsqueda en profundidad, la búsqueda «primero el mejor» y la búsqueda al azar. Entre ellos, la búsqueda en profundidad y la búsqueda «primero el mejor» son dos métodos eficientes y comúnmente utilizados.

- Búsqueda en anchura: criterio también conocido como *Breadth-First Search*. Recorre el árbol por niveles, seleccionando el problema que tenga un mayor tamaño. Este tipo de estrategia no es recomendable ni desde el punto de vista del tiempo de computación ni desde el del ahorro de memoria.
- Búsqueda en profundidad: criterio también conocido como *Depth-First Search*. Se selecciona, de entre aquellos subproblemas que estén a más profundidad en el árbol de búsqueda, el que tenga un menor límite inferior. Dicho de otro modo, se va seleccionando el problema con el tamaño más pequeño.
  Una desventaja de este criterio es que el número de subproblemas inspeccionados es normalmente mayor que el realizado en otros tipos de selecciones, como la de «primero el mejor». Además, si se entra en una rama del árbol de búsqueda que no lleva a la solución óptima, se necesita mucho tiempo para salir de esa ramificación.
  Como ventaja tenemos que es el método de selección más económico desde el punto de vista del espacio de memoria requerido. Además, conduce más rápidamente a la obtención de una mejor cota superior de la solución.
- Búsqueda «primero el mejor»: criterio también conocido como *Best-First Search*. El subproblema con el mejor límite inferior es seleccionado primero. Con este método, difícilmente un subproblema es descompuesto innecesariamente. Aunque tiene como inconveniente que la mejor cota superior de la solución suele obtenerse en las fases finales del algoritmo.
- Búsqueda al azar: se selecciona el subproblema sin prestar atención a ninguna característica del problema.

También existe un modelo híbrido. Es una combinación de la búsqueda en profundidad y la búsqueda «primero el mejor». Se basa en aplicar alternativamente cada una de las estrategias anteriores. Primero, se realiza una búsqueda en profundidad hasta que no se pueda proseguir la búsqueda por la rama del árbol generada. De entre los nodos activos generados en la búsqueda en profundidad, se selecciona uno siguiendo una estrategia «primero el mejor» y a partir de él se realiza una nueva búsqueda en profundidad, repitiendo el proceso hasta la finalización del algoritmo. Este criterio de selección trata de obtener las ventajas de los dos criterios en los que está basado.

Regla de eliminación

La regla de eliminación reconoce y elimina subproblemas que no contienen una solución óptima para el problema original.

Regla de finalización

La regla de finalización es incorporada en problemas donde la solución está determinada con una precisión deseada. Determina cuándo un subproblema pertenece al conjunto solución.

B.2 Modelo general

Los algoritmos de Ramificación y Acotación consisten en una secuencia de iteraciones en las que se aplican las reglas básicas a una estructura de datos Λ, que puede verse como una lista ordenada de subproblemas, y a los subproblemas que se extraen de esta lista de trabajo. Estas reglas seleccionan un subproblema de Λ (inicialmente el problema completo), lo descomponen y eventualmente insertan los subproblemas creados en Λ. A cada subproblema se le asocia una solución factible, resolviendo el subproblema si es suficientemente simple o asignándole una solución, elegida entre las posibles dentro del subproblema (no necesariamente la mejor). La mejor solución factible encontrada durante la ejecución del algoritmo de Ramificación y Acotación será un límite superior de la solución final y se utilizará para eliminar aquellos subproblemas generados que no pueden contener una solución factible mejor que la que ya se ha encontrado.

La ejecución del algoritmo comienza con la estructura de datos en su estado inicial $\left(\{S\}, \overline{f^\star}\right)$, donde $\overline{f^\star} \geq f^\star$ representa el límite superior inicial de la solución óptima $f^\star$ (posiblemente infinito), y se termina en el estado final $(\{\emptyset\}, f^\star)$.

Como puede observarse, no existe una regla más importante que las demás, ya que todas ellas cooperan conjuntamente para la resolución del problema. El objetivo principal es reducir el árbol de búsqueda para obtener rápidamente la mejor solución. Aunque la regla de eliminación es la que realiza la poda del árbol en última instancia, será aplicada en mayor o menor medida dependiendo de las demás reglas. Una regla de acotación que obtenga un buen límite inferior de la posible solución de un subproblema permitirá caracterizar mejor a los subproblemas que no contienen la mejor solución para eliminarlos. Para poder eliminar subproblemas, también hay que encontrar un buen límite superior de la solución. Este límite superior se consigue inspeccionando primero los nodos más prometedores, ya que son los que pueden aportar mejores soluciones. Este orden se establece mediante la regla de selección, de la que también dependen los requerimientos de memoria del problema. Hay que distinguir entre el número total de subproblemas inspeccionados y el número máximo de subproblemas almacenados en la estructura de datos Λ. La gestión de esta estructura de datos orde-

nada será más costosa cuanto mayor sea el número de subproblemas que contenga. La regla de ramificación también juega un papel importante, ya que de ella dependerá el número de ramas generadas a partir de un nodo del árbol de búsqueda. Por un lado, generar muchas ramas permite obtener información más precisa debido a que los subproblemas generados son menores en tamaño pero, por otro lado, hay que inspeccionar más nodos del árbol en cada división. Por lo tanto, la regla de ramificación debe estar orientada a reducir el espacio de búsqueda pero sin generar un número excesivo de nodos en cada división.

II. El problema de diseño de mezclas

En la industria alimentaria, las materias primas son mezcladas y procesadas para obtener diversos productos. Estos procesos de mezcla no ocurren solamente en el sector alimenticio, sino también en otros tipos de industrias, tales como la petroquímica.

Los productos que son producidos en grandes cantidades requieren extensas pruebas y un cuidadoso diseño, donde muchos aspectos tales como la robustez, el coste, la elección y disponibilidad de las materias primas, etc. juegan un papel importante. Las compañías deben tener en cuenta estos aspectos antes de la fabricación de un producto. En situaciones prácticas, tales problemas son resueltos a diario en la industria, donde los requisitos suelen ser modelados a través de restricciones cuadráticas.

El diseño de mezclas es un problema que se clasifica dentro de los problemas de Optimización Global.

A. Definición

El problema del diseño de mezclas para obtener un producto consiste en identificar mezclas, cada una representada por un vector $x \in \mathbb{R}^n$, que cumplan unos ciertos requisitos. El conjunto de posibles mezclas se define matemáticamente a través de un símplex unidad

$$S = \left\{ x \in \mathbb{R}^n : \sum_{i=1}^{n} x_i = 1.0;\ x_i \geq 0 \right\}, \qquad (2)$$

donde cada variable x_i representa el porcentaje de la materia prima i en la mezcla x. Un n-símplex es la envoltura convexa de $n+1$ puntos afínmente independientes $x_0, \ldots, x_n$ en un espacio euclídeo de dimensión n o mayor. Un conjunto de $r+1$ vectores $v_0, \ldots, v_r \in \mathbb{R}^n$ es llamado afínmente independiente cuando $(v_1 - v_0), \ldots, (v_r - v_0)$ son linealmente independientes. Su representación (2) es única, lo cual obviamente no es cierto para polítopos arbitrarios. Un 0-símplex es un punto, un 1-símplex es un segmento de una línea, un 2-símplex es un triángulo, un 3-símplex es un tetraedro, etc. En la figura 4 se puede ver una representación gráfica de varios n-símplices.

Los símplices tienen una serie de propiedades interesantes [8]. Es evidente que, al establecer $0 \leq k \leq n$ de las componentes de x a cero en (2), se obtiene una cara $(d = n - k)$-dimensional F de S. Una cara d-dimensional es una envoltura convexa de un subconjunto no vacío de d elementos de los $n+1$ puntos que definen un n-símplex. Una cara F es también un símplex en sí, en el menor subespacio afín que contenga a F. Una cara 0-dimensional es un vértice, una cara 1-dimensional es una arista, una cara $(d-1)$-dimensional es una faceta y una cara d-dimensional es el n-símplex en sí. Un n-símplex tiene $\binom{n+1}{n-d} = \binom{n+1}{d+1}$ caras d-dimensionales $(d = 0, \ldots, n)$. De la anterior ecuación se obtiene que el número de caras d-dimensionales equivale al número de caras $(n-d)$-dimensionales. En particular, un n-símplex tiene $n+1$ facetas.

Otra consecuencia inmediata de la definición de un símplex es que los n-símplices tienen el menor número de vértices de entre todos los polítopos n-dimensionales.

En los problemas de diseño de mezclas, el objetivo es encontrar una mezcla x que minimice el coste del producto, $f(x) = c^T x$, donde el vector c representa el coste de las materias primas. En situaciones prácticas, tales problemas son resueltos a diario en la industria donde a menudo los requisitos son modelados por restricciones cuadráticas [9], [10], [11]. Se han realizado estudios sobre cómo tratar con restricciones cuadráticas, con semicontinuidad en las variables y cómo generar soluciones robustas [12], [13].

No sólo el coste del producto debe ser minimizado, sino también el número de materias primas involucradas en la mezcla x dado por $\sum_{i=1}^{n} \delta_i(x)$, donde

$$\delta_i(x) = \begin{cases} 1 & \textbf{si} \quad x_i > 0, \\ 0 & \textbf{si} \quad x_i = 0. \end{cases}$$

Observando las características del problema, parece más apropiado adoptar un enfoque multiobjetivo (véase la sección I-A.1). La solución debe proporcionar el mínimo coste posible para cada número de materias primas utilizadas.

Los productos deben satisfacer ciertos requisitos. Para problemas de mezcla relativamente sencillos, los límites y restricciones lineales (véanse [9], [10] y [11])

$$h_i(x) \leq 0; \quad i = 1, \ldots, l, \qquad (3)$$

definen el espacio de búsqueda $X \subset S$.

Sin embargo, en la práctica aparecen restricciones cuadráticas [12], [13]. Las restricciones cuadráticas se describen como

$$g_i(x) = x^T A_i x + b_i^T x + d_i \leq 0; \quad i = 1, \ldots, m, \quad (4)$$

en donde A_i es una matriz simétrica de orden $n \times n$, b_i es un vector de dimensión n and d_i es un escalar. Se define Q como el espacio donde se satisfacen tales restricciones cuadráticas:

$$Q = \{x \in S\ :\ g_i(x) \leq 0;\ i = 1, \ldots, m\}.$$

Encontrar un punto $x \in X \cap Q$ define el problema del diseño de mezclas cuadráticas (*Quadratic Mixture Design Problem*, QMDP), que es estudiado en [14].

La semicontinuidad de las variables está relacionada con la dosis mínima aceptable *md* (*minimal dose*) de cada materia prima i, de forma que $x_i = 0$

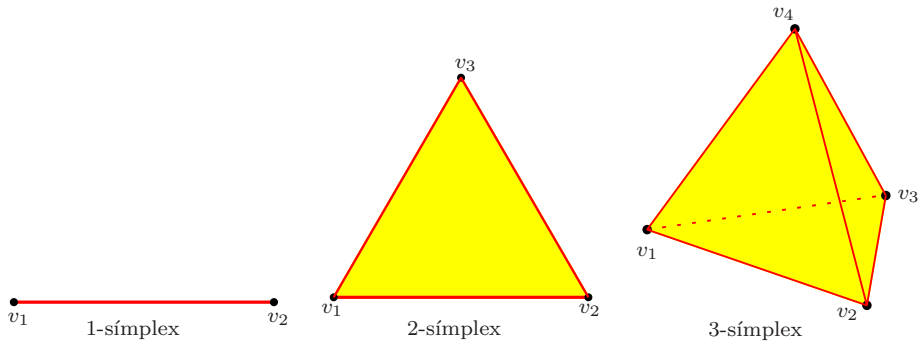


Fig. 4: Representación gráfica de varios n-símplices

ó $x_i \geq md$. La figura 5 muestra un ejemplo gráfico en el espacio de búsqueda en 2D (izquierda) y en 3D (derecha) que consiste en símplices donde se ha eliminado la región que determina la dosis mínima. El número de subsímplices resultantes (caras) es

$$\sum_{t=1}^{n} \binom{n}{t} = 2^n - 1, \tag{5}$$

donde t denota el número de materias primas en cada subsímplex. Todos los puntos x en un símplex inicial $P_u,\ u = 1, \ldots, 2^n - 1$, son mezclas del mismo grupo de materias primas. El índice u representa el grupo de materias primas correspondiente a un símplex inicial P_u:

$$u = \sum_{i=1}^{n} 2^{i-1} \delta_i(x), \quad \forall x \in P_u.$$

Una manera común de formular la semicontinuidad es la siguiente:

$$\delta_i \cdot md \leq x_i \leq \delta_i; \quad i = 1, \ldots, n,$$

donde δ es una variable binaria que representa si la materia prima i está presente en la mezcla x o no. El carácter semicontinuo de las variables también podría definirse a través de restricciones cuadráticas:

$$x_i \cdot (md - x_i) \leq 0; \quad i = 1, \ldots, n.$$

Desde una consideración práctica, sería deseable que pequeños errores en el proceso de producción no dieran lugar a productos que no cumpliesen las restricciones de diseño impuestas, es decir, la producción debería seguir cumpliendo los requisitos de diseño pese a pequeñas variaciones ($< \varepsilon$) en el proceso. Este concepto se conoce con el nombre de robustez. La robustez $R(x)$ de un diseño $x \in Q$ con respecto a Q puede ser definida como

$$\begin{aligned} R(x) = \text{máx}\{&R \in \mathbb{R}^+ : (x + r) \in Q, \\ &\forall r \in \mathbb{R}^n,\ \|r\| \leq R\}. \end{aligned} \tag{6}$$

Téngase en cuenta que, para los problemas de mezcla, $x+h$ se proyecta sobre el símplex inicial. En [15], la robustez es calculada analíticamente para problemas de diseño de mezclas lineales y se demuestra que el cálculo no es sencillo para restricciones cuadráticas. No existe una expresión analítica para determinar el punto no factible más cercano a x, así que es necesario el algoritmo descrito en [12]. Además, [15] muestra que determinar el punto de máxima robustez se convierte en un problema de Optimización Global. El interés principal es generar soluciones ε-robustas, es decir, un elemento de $\{x \in Q : R(x) \geq \varepsilon\}$.

Teniendo en cuenta las consideraciones anteriores, se define SQMDP como:

$$\begin{array}{ll} \text{mín} & f(x),\ \sum_{i=1}^{n} \delta_i(x) \\ \text{s.a.} & x \in X \cap Q \\ & R(x) \geq \varepsilon \end{array}$$

En [12] se describen tests basados en las llamadas «esferas de no factibilidad», que identifican áreas donde no se puede encontrar una mezcla factible. En [13] se describe un algoritmo de Ramificación y Acotación para resolver SQMDP usando tests de rechazo basados en restricciones lineales, cuadráticas y de robustez.

El problema de encontrar la mejor solución robusta se convierte en un problema de Optimización Global, donde resulta complejo garantizar que la solución obtenida es la global, ya que pueden existir múltiples óptimos locales y el área factible puede no ser convexa o incluso estar dividida en varios compartimentos.

En [16] se presentó una versión multihebrada del algoritmo para resolver SQMDP mediante un esquema *Asynchronous Multiple Pool* [17], donde se siguió una estrategia similar a la utilizada en el algoritmo paralelo de Optimización Global basado en Aritmética de Intervalos [18].

B. Algoritmo para resolver el problema de mezcla

Teniendo restricciones lineales, una función objetivo lineal y variables semicontinuas, se podría aplicar un método de programación lineal entera mixta (*Mixed Integer Linear Programming*, MILP). Pero debido a la existencia de restricciones cuadráticas, no se puede aplicar dicho método de resolución.

La industria aplica diferentes enfoques para generar soluciones aceptablemente buenas. Los métodos usados se suelen basar en conceptos tales como generar soluciones aleatorias, algoritmos de agrupamiento (*clustering*), programación no lineal basada en búsqueda local y búsqueda en rejilla (*grid search*).

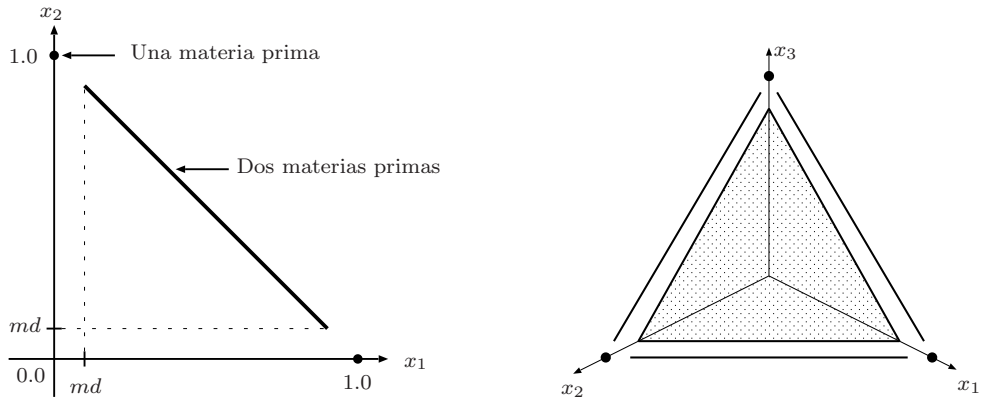


Fig. 5: Símplices 2D y 3D sin la región de dosis mínima

La estructura matemática del problema requiere tratar con variables binarias y restricciones lineales y cuadráticas (no convexas). Desde el punto de vista de la complejidad, encontrar soluciones es ciertamente un reto cuando se buscan enfoques que apliquen una búsqueda exhaustiva en lugar de aplicar una búsqueda heurística (véase la sección I-A.2).

El objetivo es diseñar un algoritmo para encontrar las soluciones ε-robustas más baratas. Este algoritmo debería ser capaz de identificar problemas que no contengan soluciones ε-robustas. Por otra parte, hay que hacer un seguimiento del número de materias primas que utiliza en cada diseño. Se ha desarrollado un algoritmo específico para este caso. Este algoritmo está basado en el concepto de Ramificación y Acotación (véase la sección I-B). Se han descrito y probado varias opciones que permiten comprobar la factibilidad de un subconjunto y determinar los límites inferiores de la robustez.

En [12] se realiza una comparación de la eficiencia entre un algoritmo de búsqueda en rejilla y un algoritmo que utiliza la técnica de Ramificación y Acotación. Para dos problemas elegidos, el número de vértices evaluados por el algoritmo B&B es entre 340 y 2300 veces menor que el número de vértices evaluados con el algoritmo que realiza la búsqueda en rejilla.

El algoritmo 1 muestra el esquema de Ramificación y Acotación utilizado para la resolución del problema de mezcla. Se denotará a un símplex por C_k, donde k determina el orden en que el símplex fue generado. Se denota t_k al número de materias primas de C_k. En el algoritmo, todos los t_k vértices de un símplex C_k, denotados por $v_{k,j},\ j = 1, \ldots, t_k$, son evaluados, es decir, son calculados los valores de las restricciones cuadráticas $g_i(v_{k,j}),\ i = 1, \ldots, m$, las restricciones lineales $h_i(v_{k,j}),\ i = 1, \ldots, l$, y la función de coste $f(v_{k,j})$ (línea 6 del algoritmo 1). El valor del coste es utilizado para actualizar los límites superiores globales $f_t^U,\ t = 1, \ldots, n$. Si $v_{k,j}$ es factible así como ε-robusto, es almacenado como un diseño Pareto-óptimo si $f(v_{k,j}) \leq f_{t_k}^U$ y los límites superiores del frente de Pareto son actualizados en consecuencia.

El algoritmo 1 comienza generando el conjunto inicial de $2^n - 1$ subsímplices (5) que son definidos eliminando la región de dosis mínima desde el símplex original (véase la figura 5). Este conjunto inicial de símplices es almacenado en la lista de trabajo Λ (línea 2). Mientras la lista de trabajo no se encuentre vacía, un símplex C_k de Λ es seleccionado y evaluado (líneas 5 y 6). Si C_k no puede ser eliminado (línea 8) y tampoco satisface el criterio de finalización (línea 9), éste es dividido mediante bisección. De los dos símplices generados, el más caro es almacenado en la lista de trabajo Λ, mientras el algoritmo procede con el más barato (búsqueda en profundidad). Aquellos símplices que satisfacen el criterio de finalización son almacenados en la lista final Ω, que determina el conjunto de todos los símplices donde una mezcla óptima ε-robusta global puede ser localizada, si la hubiere. El algoritmo también almacena las mejores mezclas ε-robustas encontradas para cada número de materias primas $t = 1, \ldots, n$.

En las siguientes subsecciones se proporcionará una descripción más detallada de las reglas del

Algoritmo 1 B&B

```
1:  Inicializar ns := 2^n − 1
2:  Inicializar la lista Λ := {C_1, ..., C_n}
3:  Inicializar la lista Ω := {}
4:  while Λ ≠ {} do
5:    Seleccionar un símplex C = C_k de Λ
6:    Evaluar C
7:    Calcular un límite inferior f^L(C) de f en C
8:    if C no puede ser eliminado then
9:      if C satisface la regla de terminación then
10:       Almacenar C en Ω
11:     else
12:       Dividir C en C_{ns+1}, C_{ns+2}
13:       C := arg mín{f^L(C_{ns+1}), f^L(C_{ns+2})}
14:       Almacenar {C_{ns+1}, C_{ns+2}} \ C en Λ_j
15:       ns := ns + 2
16:       Ir a 6
17:     end if
18:   end if
19: end while
20: return Ω
```

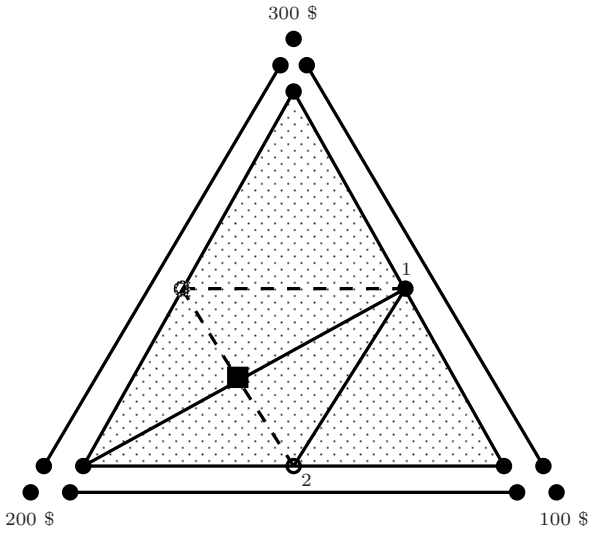


Fig. 6: Ejemplo de división según el coste de las materias primas

algoritmo 1.

B.1 Regla de ramificación

La regla de ramificación aplicada en el algoritmo 1 consiste en realizar una bisección al lado más largo de un símplex C_k dado, con $t_k \geq 2$, generando dos nuevos símplices. Usando esta regla de ramificación, la longitud del lado más largo es a lo sumo dos veces la longitud de su lado más corto. Por lo tanto, los sucesivos símplices nunca tendrán una forma de aguja [19], [20]. Si todos los lados son de la misma longitud, el lado formado por el vértice más caro y el vértice más barato es dividido. Así, la regla de ramificación genera un símplex más caro y un símplex más barato, en promedio. Esto ayuda a la regla de selección (véase la sección II-B.4), la cual otorga más prioridad a los símplices más baratos. Observe que esta regla de ramificación no siempre genera un nuevo vértice, ya que un vértice generado puede ser compartido por diferentes símplices.

La figura 6 muestra un ejemplo de la regla de ramificación del algoritmo 1. En primer lugar, el vértice etiquetado como 1 es generado teniendo en cuenta el valor del coste de los vértices del símplex equilátero. El vértice 2 es generado bisecando el lado más largo. Las líneas discontinuas representan futuras subdivisiones que terminan en un vértice (dibujado como un cuadrado) compartido por cuatro símplices.

El número de símplices que el algoritmo 1 genera (y almacena) en el peor de los casos depende de muchos aspectos. En el peor caso, las reglas conducen a la división y almacenamiento de símplices que tienen un tamaño ligeramente mayor al valor de precisión α. Después de ir $n(n-1)/2$ niveles de profundidad en el árbol de búsqueda, al menos todos los lados se han reducido a la mitad y el tamaño del símplex es menor o igual que la mitad de su tamaño original. Supongamos que $Size(C_1) = 1$ (distancia en la norma infinita). El número máximo K de reducir a la mitad los símplices es dado por $1/2^K \leq \alpha$, tal que $K = \lceil(-\ln(\alpha)/\ln(2))\rceil$, donde $\lceil y\rceil$ es el menor entero mayor o igual a y. Dado el número de lados por símplex $n(n-1)/2$, la profundidad máxima del árbol de búsqueda es $K \times n(n-1)/2$. El nivel final no es almacenado, ya que los símplices cumplen la regla de finalización (véase la sección II-B.3). Una sobreestimación del número de símplices en el árbol en el peor de los casos es

$$2^{K\times n(n-1)/2-1},$$

donde $K = \lceil(-\ln(\alpha)/\ln(2))\rceil$. Este análisis proporciona una garantía de que el algoritmo es finito dada una precisión α. Se observa que en la práctica el número de símplices que el algoritmo genera y evalúa es mucho menor. El éxito de la técnica B&B depende de a qué niveles se han podido podar las ramas del árbol de búsqueda.

B.2 Regla de acotación

El límite inferior de la función lineal de coste f en C_k en la regla de acotación del algoritmo 1 puede ser evaluado como

$$f_k^L = \min_{j=1,\ldots,t_k} \{f(v_{k,j})\},$$

debido a la linealidad de f y a la convexidad de C_k. Este límite inferior es utilizado por el test de rechazo por dominancia de Pareto, descrito en la sección II-B.5.

B.3 Regla de finalización

Un símplex no es dividido más veces cuando su tamaño es menor que el valor de precisión α. El tamaño es dado por la longitud del lado más largo, es decir,

$$\text{Size}(C_k) = \max_{v,w\in C_k} \|v - w\|.$$

Si el símplex no ha sido rechazado por la regla de eliminación (véase la sección II-B.5), es almacenado en la lista final Ω. El algoritmo 1 finaliza cuando la lista de trabajo Λ se encuentra vacía. La lista final Ω proporciona el conjunto de símplices donde una solución ε-robusta óptima puede ser encontrada, si la hubiere.

B.4 Regla de selección

La regla de selección ha sido diseñada para satisfacer dos objetivos: facilitar la eliminación de símplices con un alto número de materias primas y reducir los requisitos de memoria. El primer objetivo se cumple dando prioridad a los símplices con un número bajo de materias primas y con un coste bajo. Símplices con más materias primas y/o mayor coste pueden ser dominados por aquellos con menor número de materias primas, lo que mejora la eficiencia del test de rechazo por dominancia de Pareto (véase la sección II-B.5). En el algoritmo 1, el coste del símplex es medido por la suma de los costes de sus vértices, es decir,

$$\text{Cost}(C_k) = \sum_{j=1}^{t_k} f(v_{k,j}).$$

El segundo objetivo se cumple aplicando un criterio de selección en profundidad. Una vez que un símplex es seleccionado y dividido, su hijo más barato será el nuevo símplex seleccionado hasta que no se permitan más subdivisiones. Esto reduce la memoria requerida por el algoritmo.

B.5 Regla de eliminación

La regla de eliminación está basada en un conjunto de tests que tienen en cuenta las condiciones de factibilidad asociadas a las restricciones lineales y cuadráticas y los requisitos relacionados con la robustez de la solución, así como la minimización del número de materias primas y de la función de coste. Los siguientes tests de rechazo han sido diseñados para ser aplicados al símplex seleccionado. El orden en que estas pruebas son ejecutadas no afecta al resultado final del algoritmo en términos de la solución final, pero sí puede influenciar en la eficiencia relativa al esfuerzo computacional (tiempo).

Factibilidad lineal

Si para una de las restricciones lineales $h_i(x) \leq 0$ todos los vértices del símplex C_k no son factibles $(h_i(v_{k,j}) > 0,\ j = 1, \ldots, t_k)$, entonces C_k es descartado porque no satisface $h_i(x) \leq 0$.

Factibilidad cuadrática

Dado un símplex C_k y un conjunto de restricciones cuadráticas $\{g_i(x) \leq 0,\ i = 1, \ldots, m\}$, C_k puede ser rechazado si $\forall x \in C_k\ \exists i \mid g_i(x) > 0$. Esto es equivalente a

$$\min_{x \in X} \left\{ \max_i g_i(x) \right\}. \tag{7}$$

Resolver (7) es un problema de Optimización Global. Esto significa que aplicando un método estándar, que utilice varios puntos de partida, se podrían obtener soluciones que son óptimos locales y todas mayores que cero, es decir, no se alcanzaría ninguna solución factible de (4). En este caso, no se puede asegurar que no exista una solución factible.

Después de verificar que ningún vértice $v_{k,j} \in C_k,\ j = 1, \ldots, t_k$, es factible, habría que comprobar si C_k es completamente no factible. En lugar de considerar las restricciones cuadráticas individualmente, sería más inteligente considerarlas de forma simultánea. Consideremos el caso en que todos los vértices no son factibles por al menos una de las funciones, $g_i(v_{k,j}) > 0$. En torno a cada vértice $v_{k,j}$, se define una esfera de no factibilidad $B_{k,j}$ tal que no contenga un punto factible:

$$B_{k,j} = \{x \in \mathbb{R}^n : \|x - v_{k,j}\|^2 < \rho_{k,j}^2\}. \tag{8}$$

Dos posibles maneras de calcular el valor de $\rho_{k,j}$ se discuten a continuación. Una de ellas se ilustra en la figura 7 y está basada en sobrestimar la derivada proyectada a partir de una constante

$$\Gamma_i(C_k) = \max_{v \in C_k} \left\| \nabla g_i(v) - \frac{\mathbf{1}^T \nabla g_i(v) \mathbf{1}}{n} \right\|,$$

obteniéndose un radio de no factibilidad

$$\rho_{k,j}^l = \max_i \frac{g_i(v_{k,j})}{\Gamma_i(C_k)}. \tag{9}$$

La figura 7 muestra (9) para un caso bidimensional con dos restricciones cuadráticas $g_i(x) \leq 0$. Aunque es sabido que todos los vértices no son factibles, el símplex no es cubierto completamente por $B_{k,1} \cup B_{k,2}$, como se define en (8).

Una segunda manera de calcular el radio de no factibilidad se basa en el siguiente teorema.

Teorema 1: Dado una función cuadrática $g(x) = x^T Ax + b^T x + d$ con una matriz simétrica A de orden $n \times n$ con el valor propio más negativo η, b un n-vector y d un escalar. Si $g(v) > 0$, entonces $g(v + r) > 0$ para $\|r\| < \frac{\|\nabla g(v)\| - \sqrt{\|\nabla g(v)\|^2 - 4\eta g(v)}}{2\eta}$.

Demostración:

$$\begin{aligned} g(v+r) &= (v+r)^T A(v+r) + b^T(v+r) + d \\ &= g(x) + r^T(2Av + b) + r^T Ar \\ &\leq g(x) + \|r\|\|\nabla g(x)\| + \eta\|r\|^2 \end{aligned} \tag{10}$$

Resolviendo sobre ρ,

$$g(v) - \rho\|\nabla g(v)\| + \eta\rho^2 = 0$$

proporciona una raiz positiva

$$\rho = \frac{\|\nabla g(v)\| - \sqrt{\|\nabla g(v)\|^2 - 4\eta g(v)}}{2\eta}.$$

Dado $g(v) > 0$ conduce a $g(v + r) > 0$ en (10) para $\|r\| < \rho$. ■

Para obtener esferas de no factibilidad usando el teorema 1, hay que determinar el menor valor propio $\eta_1, \ldots, \eta_m$ de $A_1, \ldots, A_m$. Tenga en cuenta que si la función g_i es convexa, el menor valor propio es positivo y, por lo tanto, el teorema 1 no puede ser aplicado. El radio de no factibilidad cuadrática viene dado por $\rho_{k,j}^q(C_k) =$

$$\max_{i\ :\ \eta_i < 0} \frac{\|\nabla g_i(v_{k,j})\| - \sqrt{\|\nabla g_i(v_{k,j})\|^2 - 4\eta_i g_i(v_{k,j})}}{2\eta_i}.$$

Cuanto más negativa sea la curvatura, es decir, cuanto más negativo sea η_i, el radio será menor. Por lo tanto, son mejores los valores negativos pequeños de η. Similar a la proyección del gradiente, dicho valor se puede mejorar porque la desviación del punto $x + r$, o su proyección, se mueve sobre el símplex inicial que contiene a C_k. Esto significa que es suficiente considerar el valor propio más negativo de la matriz $P^T AP$, cuando la matriz de proyección P es dada por $P =$

$$\frac{1}{n} \begin{pmatrix} n-1 & -1 & . & . & -1 \\ -1 & n-1 & \ddots & . & . \\ . & \ddots & \ddots & \ddots & . \\ . & . & \ddots & n-1 & -1 \\ -1 & . & . & -1 & n-1 \end{pmatrix} \tag{11}$$

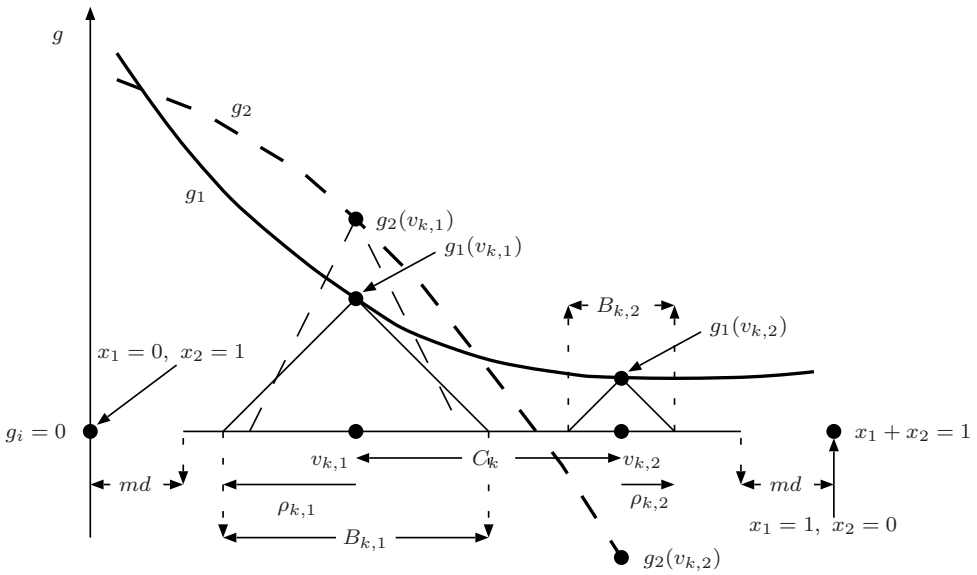


Fig. 7: Ejemplos de radios de no factibilidad cuadrática

Finalmente, para obtener la esfera de no factibilidad cuadrática más grande en el vértice $v_{k,j}$, se toma

$$\rho_{k,j} = \text{máx}\{\rho^l_{k,j}(C_k), \rho^q_{k,j}\}.$$

La idea de las esferas de no factibilidad soporta bien el concepto de considerar varias restricciones al mismo tiempo. Para ir un paso más adelante, también se podría considerar las esferas de no factibilidad de todos los vértices simultáneamente. La comprobación de la no factibilidad de un símplex estaría basada en comprobar si $C_k \subset \cup_j B_{k,j}$. No es fácil realizar esta comprobación. Para realizarla, se podría buscar un punto en C_k que no pertenezca a $\cup_j B_{k,j}$. Sin embargo, el siguiente teorema muestra que es suficiente buscar un punto que esté cubierto por todas las esferas de no factibilidad, justo lo contrario que la anterior búsqueda. La demostración de este teorema se puede consultar en [12].

Teorema 2: Dado $V = \text{conv}\{v_1, \ldots, v_h\}$ con $v_1, \ldots, v_h$ los puntos extremos de V (vértices). Supongamos que cada vértice v_j es no factible, es decir, $\text{máx}_i\, g_i(v_j) > 0$ con su correspondiente esfera de no factibilidad B_j definida por (8). Si $\exists\, x \in \cap_j B_j \cap V$, entonces $V \subset \cup_j B_j$ y, por consiguiente, no existe ningún punto factible en V.

El teorema 2 muestra que para demostrar la no factibilidad de C_k, es suficiente encontrar un punto $x \in C_k$ que para cada vértice $v_{k,j}$ sea más cercano que $\rho_{k,j}$. Un buen candidato es la media ponderada

$$\xi_k = \frac{1}{\sum_j \frac{1}{\rho_{k,j}}} \sum_j \frac{v_{k,j}}{\rho_{k,j}} \tag{12}$$

y comprobar si $\|\xi_k - v_{k,j}\|^2 < \rho^2_{k,j}$ para cada vértice $v_{k,j}$. Si ξ_k no es cubierto por la esfera de menor longitud, es decir, $\|\xi_k - v_{k,s}\|^2 \geq \rho^2_{k,s}$ con $s = \arg\text{mín}_j\, \rho_j$, se podría tomar otro punto de prueba. Este punto se puede obtener heurísticamente siguiendo la dirección de $v_{k,s}$ y generando un punto θ_k que, al menos, esté cubierto por la esfera más pequeña:

$$\theta_k = v_{k,s} + (\rho_{k,s} - \delta)\frac{\xi_k - v_{k,s}}{\|\xi_k - v_{k,s}\|}, \tag{13}$$

donde δ es un valor positivo pequeño tal que θ_k este dentro de la esfera de menor longitud.

En la figura 8 se puede observar un ejemplo de funcionamiento del teorema 2. Cada restricción ha sido representada con una línea discontinua diferente y un color diferente. Cada esfera de no factibilidad (8) ha sido representada con la misma línea discontinua y el mismo color que su correspondiente restricción cuadrática. Los símplices de color gris se encuentran en la lista de trabajo Λ. Centrando nuestra atención en el símplex C_k (de color blanco), que está siendo evaluado, se puede observar que ninguna de las esferas cubre totalmente el símplex. El punto ξ_k tampoco es cubierto por todas las esferas. En cambio, el punto θ_k, dentro de la esfera menor, es cubierto por todas las esferas, así que este símplex no contiene ningún punto factible y puede ser eliminado.

Toda esta teoría es utilizada por los siguientes tests, que prueban la no factibilidad de un símplex C_k:

- SCTest (*Single Cover Test*): prueba que un símplex C_k no es factible en el caso de que esté completamente cubierto por una esfera de no factibilidad. Así que SCTest consiste en determinar un vértice $v_{k,j}$ que satisfaga que $\rho_{k,j} > \text{máx}_{x \in C_k} \|v_{k,j} - x\|$.
- NCTest (*N spheres Cover Test*): intenta determinar si un símplex C_k está completamente cubierto por el conjunto de todas las esferas de no factibilidad $B_{k,j}$, es decir, $C_k \subset \cup_j B_{k,j}$. Es básicamente la aplicación práctica del teorema 2 usando el punto definido en (12) y (13).
- PCTest (*Point Cover Test*): es aplicado cuando SCTest y NCTest fallan. Se genera una esfera de no factibilidad centrada en ξ_k o θ_k para comprobar si dicha esfera cubre por completo el símplex

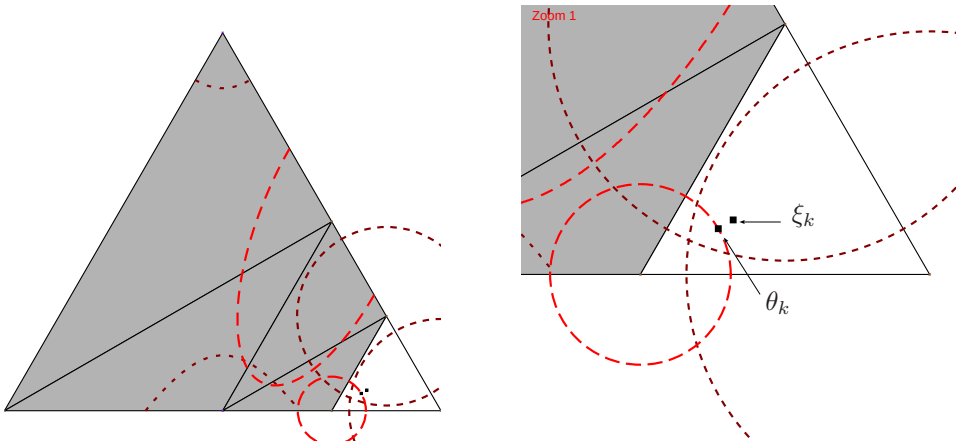


Fig. 8: Ejemplo de rechazo de C_k usando θ_k en lugar de ξ_k

C_k. Se sabe que ξ_k o θ_k no es factible para al menos una de las restricciones cuadráticas. La ventaja de utilizar un punto interior es que su distancia al vértice más lejano es menor que la mayor distancia entre los vértices.

Robustez

El requisito de la ε-robustez hace rechazar un símplex C_k que está lo suficientemente cerca de una solución no factible. Un símplex C_k con un punto no factible x tal que $\max_j \|x - v_{k,j}\| < \varepsilon$, no puede contener una solución ε-robusta. La prueba de ε-robustez comprueba esta condición para $x = v_{k,j}$ $(j = 1, \ldots, t_k)$, $x = \xi_k$ y posiblemente θ_k.

El algoritmo 1 ha sido diseñado para tratar con problemas de diseño de mezclas donde la solución (si existe) ha de ser ε-robusta. La robustez $R(x)$ de un diseño x con respecto a D ha sido definida en (6). Para restricciones cuadráticas, $R(x)$ no es fácil de determinar. El algoritmo aplica una subestimación $R^L(x)$. Si la cota inferior $R^L(x)$ es mayor que ε, el punto x es ε-robusto. Similar a (9), se puede definir para un punto factible x con $g_i(x) < 0$, $i = 1, \ldots, m$,

$$R^L_\Gamma(x) = \min_i \frac{-g_i(x)}{\Gamma_i}.$$

Siempre que $\|h\|$ sea menor que $R^L_\Gamma(x)$, entonces $g_i(x+h) \le g_i(x) + \Gamma_i\|h\| \le 0$ y el punto $x+h$ cumple todas las restricciones. Para Γ_i se puede tomar la distancia euclídea máxima del gradiente proyectado sobre el símplex inicial que contiene a C_k.

Alternativamente, en la línea del teorema 1, se puede establecer una cota inferior de la forma cuadrática de la función

$$\begin{aligned} g(x+h) &= (x+h)^T A(x+h) + b^T(x+h) + d \\ &= g(x) + h^T(2Ax + b) + h^T Ah \\ &\le g(x) + \|h\|\|\nabla g(x)\| + \mu\|h\|^2, \end{aligned}$$

donde μ es el mayor valor propio de A. Se determinan los valores propios μ_i de A_i y se calcula la cota inferior de la robustez basada en consideraciones cuadráticas como $R^L_Q(x) =$

$$\min_{i:\ \mu_i > 0} \frac{-\|\nabla g_i(x)\| + \sqrt{\|\nabla g_i(x)\|^2 - 4\mu g_i(x)}}{2\mu}.$$

Esto muestra claramente que si la segunda derivada es mayor, la estimación de la robustez es peor. Por lo tanto, son mejores los valores más pequeños de μ. Similar a la proyección del gradiente, se puede mejorar ligeramente ese valor considerando que el punto $x + h$ (o su proyección) se mueve sobre el símplex que contiene a C_k. Dicho de otro modo, es suficiente considerar el mayor valor propio de la matriz $P^T AP$, donde la matriz de proyección P viene dada por (11). El límite inferior de la robustez $R^L(x)$ se obtiene como el $\max\{R^L_\Gamma, R^L_Q\}$.

La solución proporcionada como salida del algoritmo 1 (si existe) es el conjunto de puntos $\{x : x \in X \cap Q;\ R^L(x) \ge \varepsilon\}$ con el menor coste encontrado. Además, se obtiene el conjunto de símplices que pueden contener una solución mejor que las encontradas con la precisión establecida.

Optimalidad de Pareto

Se está tratando con la minimización del número de materias primas y del coste de la mezcla. Si una solución ε-robusta ha sido encontrada con t_k materias primas y un coste menor que f^L_k, entonces C_k puede ser eliminado, es decir, se descarta C_k si $f^U_{t_k} < f^L_k$. Esta idea se refleja en la figura 9, donde se muestra cómo una solución es rechazada porque está dominada por otra solución con menor coste y con un número de materias primas que es menor o igual. El algoritmo realiza un seguimiento de f^U_t, el cual es inicializado a infinito. Cuando se encuentra una mezcla ε-robusta $v_{k,j}$ con $f(v_{k,j}) < f^U_{t_k}$, el valor de f^U_t es actualizado para $t = t_k, \ldots, n$. El algoritmo 1 devuelve el vector de Pareto f^U y sus correspondientes mezclas.

Cada vez que f^U es actualizado, es necesario realizar una criba en la lista final Ω de los símplices cuyo coste superan el límite superior f^U_t, $t = t_k, \ldots, n$ (véase la figura 9).

III. El problema de bi-mezcla

Encontrar un diseño robusto y barato para un producto que satisfaga una serie de restricciones cuadráticas es una tarea ardua. En la industria, las compañías pueden utilizar las mismas materias pri-

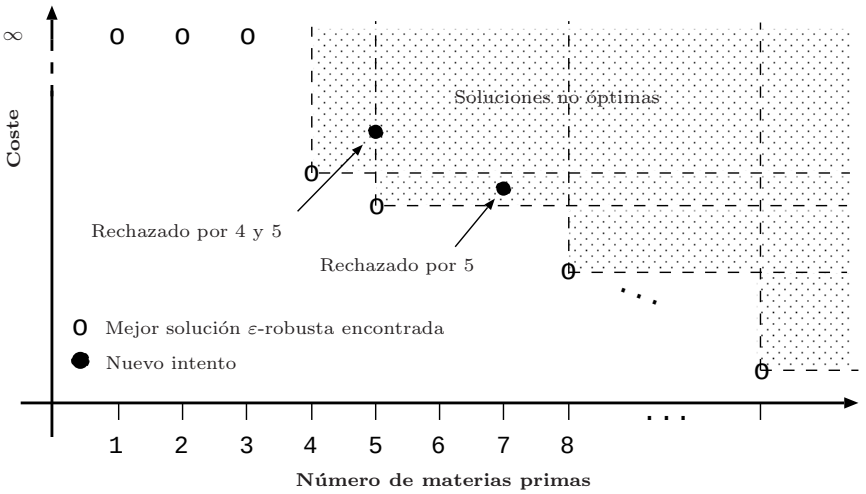


Fig. 9: Dominancia de Pareto

mas para producir varios productos. Se pueden encontrar descripciones de casos prácticos en [9] y [10], entre otros. Esto complica el proceso de búsqueda de soluciones factibles y robustas si se pretende garantizar la optimalidad y robustez de la solución final.

A. Definición

Las compañías utilizan materias primas para elaborar productos. De hecho, suelen elaborar varios productos usando (parcialmente) las mismas materias primas. Cada producto tiene su propia demanda y sus propias restricciones de diseño. En algunas ocasiones, la industria se enfrenta al problema de la escasez de materias primas para los productos que desea elaborar. Este inconveniente puede ser resuelto mediante el uso de una dosis mayor de otro ingrediente, aunque esto haga que la solución óptima para cada producto no sea siempre la solución del problema completo. La escasez de las materias primas viene descrita a través de las restricciones de disponibilidad.

Tal y como se describe en [21], cada producto tiene su propia demanda y sus propias especificaciones de diseño en forma de restricciones lineales y/o cuadráticas. Una manera común de describir el problema es usar un índice j para cada producto, que tiene una cierta demanda D_j. La cantidad de materia prima i disponible viene dada por B_i. Ahora, la variable de decisión principal $x_{i,j}$ es la fracción de la materia prima i presente en el producto j.

Sea $x_{*,j}$ la columna j de la matriz $\mathbf{x}$ de variables de decisión. Entonces definimos las restricciones lineales por producto: $x_{*,j} \in X_j$; así como las restricciones cuadráticas por producto: $x_{*,j} \in Q_j = \{y \in S : g_i(y) \leq 0;\ i = 1, \ldots, m_j\}$.

En principio, todos los productos $x_{*,j},\ j = 1, 2$, pueden hacer uso de todas las n materias primas; $x_{*,j} \in \mathbb{R}^n,\ j = 1, 2$. Esto significa que $x_{i,1}$ y $x_{i,2}$ hacen referencia a fracciones del mismo ingrediente para los productos 1 y 2. En la práctica, existe una preselección de qué materias primas pueden ser usadas para cada producto. Una descripción alternativa es que el tipo y el número de las materias primas n_j varíen según el producto final y definir, a través de conjuntos de índices, qué materias primas están compartidas.

Las reservas de materias primas se describen a través de las restricciones de disponibilidad, que vienen dadas por

$$\sum_{j=1}^{2} D_j x_{i,j} \leq B_i; \quad i = 1, \ldots, n. \tag{14}$$

Con las restricciones de disponibilidad añadidas al problema, la función de coste cambia, ya que ahora hay que tener en cuenta dos productos con sus demandas individuales. La función de coste puede ser escrita como:

$$f_{bi}(x_{*,1}, x_{*,2}) = \sum_{j=1}^{2} D_j f(x_{*,j}).$$

El otro criterio a minimizar es el número de materias primas distintas utilizado en el diseño de los dos productos:

$$\omega(x_{*,1}, x_{*,2}) = \sum_{i=1}^{n} \delta_i(x_{*,1}) \vee \delta_i(x_{*,2}), \tag{15}$$

donde $\vee$ denota la operación *or* bit a bit.

Por lo tanto, se define el problema de bi-mezcla como:

$$\begin{array}{ll} \text{mín} & f_{bi}(x_{*,1}, x_{*,2}),\ \omega(x_{*,1}, x_{*,2}) \\ \text{s.a.} & x_{*,1} \in X_1 \cap Q_1,\ x_{*,2} \in X_2 \cap Q_2 \\ & R(x_{*,j}) \geq \varepsilon,\ j = 1, 2 \\ & \sum_{j=1}^{2} D_j x_{i,j} \leq B_i;\ i = 1, \ldots, n \end{array}$$

B. Algoritmo para resolver el problema de bi-mezcla

Se pueden generar óptimos locales del problema de bi-mezcla, descrito en la sección anterior usando software como GAMS/BARON o *solvers* en MATLAB, aunque la robustez no puede ser modelada de una forma sencilla. Además, el problema hereda la dificultad de tratar con las restricciones cuadráticas tal

Algoritmo 2 B&B

1: Inicializar $ns := 2 \times (2^n - 1)$
2: Inicializar la lista $\Lambda_1 := \{C_1, \ldots, C_{2^n-1}\}$
3: Inicializar la lista $\Lambda_2 := \{C_{2^n}, \ldots, C_{ns}\}$
4: Inicializar las listas $\Omega_1 := \{\}$ y $\Omega_2 := \{\}$
5: **while** $\Lambda_1, \Lambda_2 \neq \{\}$ **do**
6: Seleccionar un símplex $C = C_k$ de Λ_j
7: Evaluar C
8: Calcular $f^L(C)$ y $b_i^L(C),\ i = 1, \ldots, t_k$
9: **if** C no puede ser eliminado **then**
10: **if** C satisface la regla de terminación **then**
11: Almacenar C en Ω_j
12: **else**
13: Dividir C en C_{ns+1}, C_{ns+2}
14: $C := \arg\min\{f^L(C_{ns+1}), f^L(C_{ns+2})\}$
15: Almacenar $\{C_{ns+1}, C_{ns+2}\} \setminus C$ en Λ_j
16: $ns := ns + 2$
17: Ir a 7
18: **end if**
19: **end if**
20: $j := (j \bmod 2) + 1$
21: **end while**
22: **return** Ω_1 y Ω_2

y como se describe en [12]. Resolver el problema de bi-mezcla de una manera exhaustiva (el método obtiene todas las soluciones globales dentro de la precisión establecida) requiere el diseño de un algoritmo de Ramificación y Acotación específico.

Una descripción detallada del algoritmo 2 se puede encontrar en [21]. El algoritmo 2 usa una lista de trabajo Λ_j y una lista final Ω_j para cada producto.

El algoritmo 2 muestra el esquema de Ramificación y Acotación utilizado para la resolución del problema de bi-mezcla. Se denotará a un símplex por C_k, donde k determina el orden en que el símplex fue generado. Se denota t_k al número de materias primas de C_k. En el algoritmo, todos los t_k vértices de un símplex C_k son evaluados, es decir, a todos los vértices $v \in C_k$ se les calculan los valores de las restricciones cuadráticas $g_i(v),\ i = 1, \ldots, m_j$, las restricciones lineales $h_i(v),\ i = 1, \ldots, l_j$ y la función de coste $f(v)$ (línea 7 del algoritmo 2). Si se encuentra un vértice que es factible así como ε-robusto, se combina con todos los vértices factibles y ε-robustos del otro producto con el fin de encontrar una pareja de vértices que cumplan la restricción de disponibilidad y mejore $f_p^U,\ p = 1, \ldots, n$ (véase la sección III-B.5).

El algoritmo 2 comienza generando el conjunto inicial de $2^n - 1$ subsímplices para cada producto j (véase la ecuación 5), que son definidos eliminando la región de dosis mínima en el símplex original (véase la figura 5). Estos dos conjuntos iniciales de subsímplices son almacenados en las listas de trabajo Λ_1 y Λ_2 (líneas 2 y 3, respectivamente). Mientras ambas listas de trabajo no se encuentre vacías, un símplex C_k de Λ_j es seleccionado y evaluado (líneas 6 y 7). Si C_k no puede ser eliminado (línea 9) y tampoco satisface el criterio de finalización (línea 10), éste es dividido. De los dos símplices generados, el más caro es almacenado en la lista de trabajo Λ_j, mientras el algoritmo procede con el más barato (búsqueda en profundidad). Aquellos símplices que satisfacen el criterio de finalización son almacenados en la lista final Ω_j, que determina el conjunto de todos los símplices donde una mezcla óptima ε-robusta global puede ser localizada, si la hubiere. Se alterna de producto (línea 20) cuando el símplex seleccionado cumple todas las restricciones y no puede ser dividido más o bien es eliminado porque no satisface alguna restricción. El algoritmo también almacena las mejores mezclas ε-robustas encontradas para cada número de materias primas $i = 1, \ldots, n$.

B.1 Regla de ramificación

Se divide el símplex por el lado más largo o por aquel lado definido por los vértices más caro y más barato. Para profundizar más en esta parte del algoritmo, véase la sección II-B.1.

B.2 Regla de acotación

Para cada símplex se calculan dos límites inferiores:

Coste: $f^L(C_k)$ es el límite inferior del coste de un símplex C_k y es proporcionado por el vértice cuyo coste es menor, debido a que los símplices son convexos y la función de coste es lineal.

Cantidad de cada materia prima: $b_i^L(C_k)$ es el límite inferior del uso de cada materia prima i en un símplex C_k. Este límite inferior es obtenido de manera análoga al límite inferior del coste.

B.3 Regla de terminación

Los símplices no rechazados que alcanzan el tamaño requerido α son almacenados en Ω_j. Un estudio más detenido de esta regla se encuentra en la sección II-B.3.

B.4 Regla de selección

Se realiza una búsqueda híbrida: combinación de la búsqueda en profundidad y la búsqueda «primero el mejor». Se selecciona el símplex más barato, basado en la suma de los costes de sus vértices, y se realiza una selección en profundidad hasta que el símplex no pueda ser dividido más (véase el algoritmo 2, las líneas 6, 14 y 17). De esta forma, se pretende reducir el consumo de memoria del algoritmo. En la sección II-B.4, todos estos aspectos son explicados con más detalle.

B.5 Regla de eliminación

Se aplica un conjunto de tests basados en restricciones lineales, cuadráticas y de robustez a los símplices de un producto, véase la sección II-B.5.

Además, existen otros tests de rechazo donde hay que tener en cuenta ambos productos. Estos tests son los siguientes:

Disponibilidad de materias primas

El diseño de mezclas para la elaboración de dos productos tiene la dificultad añadida de la escasez de materias primas. Dos mezclas $x_{*,1}$ y $x_{*,2}$ que sean óptimas pueden no ser solución del problema si no hay la cantidad suficiente de materias primas disponible.

Para cada producto j, habrá un límite inferior de uso de cada materia prima i

$$\beta_{i,j}^L = D_j \times \min_{x \in C \in \Lambda_j \cup \Omega_j} x_i, \tag{16}$$

donde $x = v,\ \forall v \in C$, son los vértices del símplex C. También se guarda una referencia al vértice que proporciona $\beta_{i,j}^L$. Si tal vértice es eliminado porque el símplex en donde se encontraba es descartado y no existe ningún otro símplex en la lista de trabajo Λ_j o en la lista final Ω_j que contenga dicho vértice, se vuelve a inspeccionar todos los vértices hasta encontrar un nuevo límite inferior $\beta_{i,j}^L$. Existen símplices en Λ_j que no han sido eliminados y que contienen vértices no factibles. Por lo tanto, estos vértices no factibles pueden determinar el límite inferior de la materia prima i en el producto j.

En la regla de acotación vista en la sección III-B.2, al símplex se le calcula un límite inferior de uso de la materia prima i. Para cada materia prima i presente en el símplex C_k del producto j, se cumple la restricción de disponibilidad si

$$D_j \times b_i^L(C_k) + \beta_{i,j'}^L \leq B_i, \tag{17}$$

donde j' hace referencia al otro producto.

Optimalidad de Pareto

El límite superior f_p^U de la función de coste varía con respecto al problema de diseñar un solo producto. En lugar de tener un límite superior individual para cada producto, se pasa a tener un límite superior f_p^U común, donde p será el número de materias primas utilizadas para la fabricación de ambos productos.

Al comienzo del algoritmo, $f_p^U = \infty$, donde $p = 1, \ldots, n$. La pareja de vértices que actualice f^U debe cumplir todas las restricciones individuales (factibilidad lineal (3), factibilidad cuadrática (4) y ε-robustez (6)) así como la restricción de disponibilidad de materias primas (14). Si se encuentra un par de vértices $v_1 \in C_k \in \Lambda_1 \cup \Omega_1$ y $v_2 \in C_{k'} \in \Lambda_2 \cup \Omega_2$ que satisfaga que $f(v_1, v_2) < f_{\omega(v_1,v_2)}^U$, se actualiza $f_p^U,\ p = \omega(v_1, v_2), \ldots, n$. El valor $\omega(v_1, v_2)$ es el número de materias primas distintas empleadas en v_1 y v_2, véase la ecuación 15.

Cada vez que se genera un nuevo vértice por división, si dicho vértice cumple las restricciones individuales citadas anteriormente, habría que combinarlo con todos los vértices del otro producto que también cumplan las restricciones individuales, para comprobar si una combinación que satisfaga la restricción de disponibilidad puede mejorar el frente de Pareto. Entonces, es necesario tener para cada producto el conjunto de vértices factibles y ε-robustos para así agilizar la posible actualización de f_p^U. Se podría tener un conjunto con los vértices más baratos y utilizarlos para ver si alguna combinación actualiza f_p^U, pero puede que los vértices más baratos no satisfagan la restricción de disponibilidad (14).

Para cada subsímplex inicial $P_{u,j},\ u = 1, \ldots, 2^n - 1$, (véase la figura 5) del producto j se tendrá un límite inferior del coste $\varphi_{u,j}^L =$

$$\min \{ f(v) : v \in C \subset P_{u,j},\ C \in \Lambda_j \cup \Omega_j \}. \tag{18}$$

El vector $\varphi_{u,j}$ contiene el valor del coste del vértice (factible o no) más barato para cada subsímplex inicial del producto j. No es necesario que los vértices que actualizan $\varphi_{u,j}$ pertenezca a un par que satisfaga la restricción de disponibilidad (14), lo único que se pretende es que $\varphi_{u,j}$ sea un límite inferior válido del coste en el producto j. Hay que comprobar la actualización de $\varphi_{u,j}$ cuando se genera un nuevo vértice o se eliminan todos los símplices que contienen el vértice que actualizó $\varphi_{u,j}$. Esto queda claro en (18), ya que sólo es válido para los vértices que pertenecen a un símplex que no haya sido eliminado, es decir, que se encuentre en la lista de trabajo Λ_j o en la lista final Ω_j.

El motivo de que se guarde un límite inferior del coste por subsímplex tiene el fin de considerar las soluciones que utilicen un número distinto de materias primas. En cada subsímplex puede haber un vértice ε-robusto que en combinación con el vértice ε-robusto encontrado en el otro producto pueda actualizar una posición diferente de f_p^U en función del número de materias primas empleado en las dos mezclas.

Un símplex C_k no es rechazado por Pareto si para algún subsímplex inicial $P_{u,j}$ se cumple que

$$f^L(C_k) + \varphi_{u,j'}^L \leq f_{\omega(x,y)}^U; \quad x \in C_k,\ y \in P_{u,j'}. \tag{19}$$

Filtrado de las listas finales

El resultado del algoritmo 2 es un conjunto de mezclas Pareto-óptimas (dentro de la precisión α) y las listas finales $\Omega_j,\ j = 1, 2$, que, además de las mezclas, contienen los símplices que no han sido rechazados. Durante la ejecución del algoritmo, se actualizan los límites inferiores $\beta_{i,j}^L$ y $\varphi_{u,j}^L$ basándose en los vértices no rechazados para descartar símplices que no satisfagan (17) o (19). Se utilizan estos límites inferiores para evitar la ardua tarea de evaluar todas las combinaciones de los símplices de ambos productos.

Una vez finalizado el algoritmo 2, existen símplices en $\Omega_j,\ j = 1, 2$, que no contienen ninguna solución. El algoritmo 3 lleva a cabo la combinación de los símplices de ambas listas para reducir el área solución. El conjunto de combinaciones restantes después de la ejecución del algoritmo 3 puede ser utilizado como entrada para una segunda ejecución donde se requieran resultados con una mayor precisión.

Por lo tanto, se debe realizar una combinación final entre los símplices de Ω_1 y Ω_2 para rechazar aquellos

Algoritmo 3 Comb

1: **for** $j = 1, 2$ **do**
2: **for all** $C \in \Omega_j$ no marcado como válido **do**
3: **if** $\exists\, C' \in \Omega_{j'}$ que satisfaga (20) y (21) **then**
4: Marcar C' como válido
5: Continuar con el siguiente símplex C
{*Los restantes* $C' \in \Omega_{j'}$ *no son visitados*}
6: **else**
7: Eliminar C
8: **end if**
9: **end for**
10: **end for**

símplices C que no contengan una solución Pareto-óptima:

$$f^L(C) + f^L(C') > f^U_{\omega(x,y)}; \quad x \in C,\ y \in C', \quad (20)$$

o no satisfagan la restricción de disponibilidad:

$$b^L_i(C) + b^L_i(C') > B_i; \quad i = 1, \ldots, n, \quad (21)$$

para todos los símplices $C' \in \Omega_{j'}$. En caso contrario, C es rechazado. En la primera iteración ($j = 1$) del bucle externo del algoritmo 3, si se encuentra un símplex $C' \in \Omega_2$ que junto con C satisfaga (20) y (21), C' es marcado como válido y no será procesado en la línea 2 de la siguiente iteración ($j = 2$).

C. Evaluación

En este apartado se discutirán los resultados numéricos de la evaluación realizada al algoritmo de bi-mezcla. Estas pruebas ayudarán a saber si el algoritmo propuesto es capaz de resolver problemas reales en un tiempo razonable. Se comprobará qué eficacia tienen el nuevo test de optimalidad de Pareto y el test de disponibilidad de materias primas. Para probar el algoritmo, se utilizará un conjunto de problemas de mezcla de tres dimensiones. La ventaja de los casos tridimensionales es que la ejecución del algoritmo puede ser representada gráficamente. No existe aún en la literatura un conjunto de pruebas estándar para la comparación del rendimiento de diferentes casos. Las ejecuciones del algoritmo se han llevado a cabo en un procesador Intel® Core™ 2 Duo T7250 2.0 GHz con 4 GBytes de RAM sobre una distribución GNU/Linux. Se llevará un registro de:

1. Estadísticas sobre la cantidad de símplices y vértices manejados por el algoritmo.
2. Información sobre el número de símplices descartados durante la búsqueda y la causa de rechazo.
3. Información acerca del número de símplices filtrados por el algoritmo 3 en las listas $\Omega_j,\ j = 1, 2$.
4. Estadísticas generales del algoritmo: tiempo de ejecución, memoria utilizada y el número de soluciones encontradas.

La notación utilizada es:

- NSimplex: Número de símplices evaluados.
- NVertex: Número de vértices evaluados.
- NVertexF: Número de vértices factibles y ε-robustos encontrados.
- EndNSimplex: Número de símplices en las listas finales $\Omega_j,\ j = 1, 2$.
- EndNVertex: Número de vértices asociados a los símplices en $\Omega_j,\ j = 1, 2$.
- ε-Infeas.: Número de símplices rechazados por el test de ε-robustez aplicado a los vértices.
- Pareto: Número de símplices rechazados por el test de Pareto (19).
- SCTest: Número de símplices rechazados por SCTest.
- MCTest: Número de símplices rechazados por MCTest.
- ξ-θ-ε-Infeas.: Número de símplices rechazados por el test de ε-robustez test aplicado a ξ o θ.
- PCTest: Número de símplices rechazados por PCTest.
- LC: Número de símplices rechazados por no ser factibles linealmente (3).
- Capacity: Número de símplices rechazados por el test de disponibilidad (17).
- FinalPareto: Número de símplices rechazados por dominancia de Pareto (20) en $\Omega_j,\ j = 1, 2$.
- FinalCapacity: Número de símplices rechazados por escasez de materias primas (21) en $\Omega_j,\ j = 1, 2$.
- Time: Tiempo de ejecución.
- Memory: Memoria requerida por el algoritmo.
- N. Sol.: Un vector binario que muestra si, para un número dado de materias primas, el algoritmo ha encontrado una solución.

C.1 *RumCoke* & *Case2*

Se comenzará la evaluación utilizando dos productos extraídos de [13]. El primero (*RumCoke*) tiene dos restricciones lineales y dos restricciones cuadráticas. El segundo (*Case2*) fue tomado de un ejemplo de la industria con cinco restricciones cuadráticas. Se puede encontrar una descripción detallada de estos dos problemas de mezcla en el apéndice.

Las materias primas involucradas en *Case2* ($j = 2$) son RM1 ($i = 1$), RM2 ($i = 2$) y RM3 ($i = 3$). Las materias primas involucradas en *RumCoke* ($j = 1$) son RM1, RM2 y RM4. Por lo tanto, las materias primas compartidas son RM1 y RM2. En contraste con el ejemplo mostrado en la sección III-A, la componente $x_{3,1}$ hace referencia a RM4 y no a RM3.

Se ha analizado este caso de prueba con cuatro especificaciones distintas (entornos). La semicontinuidad se establece en una dosis mínima de 0.03 y la precisión en $\alpha = \varepsilon$. La demanda de cada producto es $D^T = (1, 1)$. La robustez y la disponibilidad de materias primas varía en cada entorno:

E1) Robustez $\varepsilon = \sqrt{2}/100$.
Sin restricciones de disponibilidad.
E2) Robustez $\varepsilon = \sqrt{2}/150$.
Sin restricciones de disponibilidad.
E3) Robustez $\varepsilon = \sqrt{2}/150$. $B_1 = 0.86$ unidades.

E4) Robustez $\varepsilon = \sqrt{2}/150$. $B_1 = 0.80$ unidades.

La tabla I muestra los resultados obtenidos tras ejecutar el algoritmo para el conjunto de entornos descritos previamente. El tiempo empleado en ser resueltos todos los entornos es menor de un segundo.

Para E1, no existe ninguna solución ya que no existe ninguna mezcla en *RumCoke* con una robustez de $\varepsilon = \sqrt{2}/100$. Debido a la ausencia de soluciones robustas, f_p^U no es actualizado y, por lo tanto, no llega a ejecutarse el test de Pareto.

En E2, con una robustez requerida menor ($\varepsilon = \sqrt{2}/150$), se obtiene una solución empleando cuatro materias primas. Tan pronto como una pareja de vértices ε-robustos actualiza el frente de Pareto, el test de Pareto es capaz de descartar extensas regiones del espacio de búsqueda y el número de vértices evaluados se reduce. Por otra parte, el test de ε-robustez es eficiente sólo cuando los símplices son lo suficientemente pequeños, es decir, a suficiente profundidad del árbol de búsqueda. Se debe tener en cuenta que el orden en que se aplican los tests es el mismo en que aparecen en la tabla I. Por lo tanto, no se puede determinar la eficiencia individual de cada test porque depende de los tests aplicados con anterioridad. La solución para E2 es

$$\mathbf{x}^\star = \begin{array}{c} \\ \text{RM1} \\ \text{RM2} \\ \text{RM3} \\ \text{RM4} \end{array} \begin{array}{c} \begin{array}{cc} \textit{RumCoke} & \textit{Case2} \end{array} \\ \begin{pmatrix} 0.563203 & 0.570312 \\ 0.357031 & 0.282383 \\ 0.0 & 0.147305 \\ 0.079766 & 0.0 \end{pmatrix} \end{array}$$

y el coste $f_{bi}(x^\star_{*,1}, x^\star_{*,2}) = 0.625305 + 0.402004 = 1.027309$. En ambos productos, se utiliza la máxima cantidad de RM1 ($0.563203 + 0.570312 = 1.133515$) ya que es el ingrediente más barato y su disponibilidad no está limitada. Esto significa que, para los productos *RumCoke* y *Case2*, pueden ser fabricadas las mezclas más baratas. En caso de que no haya restricciones de disponibilidad, el problema de bi-mezcla puede ser escrito como dos problemas de mezclas separados.

Cuando las restricciones de disponibilidad entran en juego, el problema es más desafiante. En E3, $B_1 = 0.86 < 1.133515$. El número de vértices evaluados se incrementa porque los vértices con un coste menor, evaluados en primer lugar de acuerdo a la regla de selección (véase la sección III-B.4), utilizan una gran cantidad de la materia prima más barata, la escasa materia prima RM1. La solución para E3 es

$$\mathbf{x}^\star = \begin{array}{c} \\ \text{RM1} \\ \text{RM2} \\ \text{RM3} \\ \text{RM4} \end{array} \begin{array}{c} \begin{array}{cc} \textit{RumCoke} & \textit{Case2} \end{array} \\ \begin{pmatrix} 0.513438 & 0.346367 \\ 0.378359 & 0.456563 \\ 0.0 & 0.197070 \\ 0.108203 & 0.0 \end{pmatrix} \end{array}$$

y el coste $f_{bi}(x^\star_{*,1}, x^\star_{*,2}) = 0.749008 + 0.551301 = 1.300309$. La solución satisface la restricción de disponibilidad impuesta ya que $0.513438 + 0.346367 = 0.859805 < 0.86$.

La figura 10 muestra el espacio de búsqueda de E3 una vez terminado el algoritmo. Cada restricción cuadrática (4) se representa con una línea discontinua de diferente trazo y color. Los símplices finales (pertenecientes a Ω_j, $j = 1, 2$) tienen un color verde. Los símplices restantes han sido descartados de acuerdo a la regla de eliminación (véase la sección III-B.5). El significado de cada color de los símplices rechazados se da a continuación:

- Terracota (naranja amarronado): Símplices rechazados por no ser factibles linealmente.
- Tonalidades azules: Símplices rechazados por no ser factibles cuadráticamente.
- Tonalidades púrpuras: Símplices rechazados por la existencia de un punto no factible cuadráticamente que produce la falta de robustez.
- Tonalidades marrones: Símplices rechazados por dominancia de Pareto. Aquellos con una tonalidad más fuerte han sido rechazados por el algoritmo 3.
- Tonalidades naranjas: Símplices rechazados por escasez de materias primas. Aquellos con una tonalidad más fuerte han sido rechazados por el algoritmo 3.

Los vértices de color blanco son factibles (véanse las ecuaciones 3 y 4) y ε-robustos (6). Los vértices solución están rodeados por una circunferencia de tamaño ε. Cada par de vértices solución tendrá un color distinto para distinguirse del resto (si los hubiere): el primer par tendrá un color amarillo, el segundo tendrá un color magenta, etc.

Se puede observar en la figura 10d unos cuantos símplices de un tamaño considerable rechazados por escasez de materias primas. Por otra parte, existe un gran número de símplices de un tamaño similar a α rechazados por este mismo motivo en el filtrado de las listas Ω_j, $j = 1, 2$, a causa de límites inferiores subestimados en el uso de materias primas.

Finalmente, E4 no tiene solución debido a la escasez de la materia prima RM1. La única diferencia entre E3 y E4 es que la cantidad disponible de la materia prima RM1 para E4 es menor que para E3.

C.2 *Case3* & *Case4*

El caso de prueba anterior sólo tenía una solución para cuatro materias primas. En este apartado se evaluará un nuevo caso de prueba donde existe más de una solución que emplea un número distinto de materias primas. El caso de prueba consiste en dos productos: *Case3* y *Case4*. Ambos tienen cuatro restricciones cuadráticas. *Case3* hace uso de RM1, RM2 y RM4; *Case4* hace uso de RM2, RM3 y RM4. La disponibilidad de RM2 está restringida a 1.35 unidades, mientras que las demás materias primas no estarán limitadas. Se describen más detalles acerca de estos productos en el apéndice. La dosis mínima se establece en $md = 0.03$, la robustez es $\varepsilon = \sqrt{2}/150$, la demanda $D = (1, 1)^T$ y la precisión $\alpha = \varepsilon$. La tabla I muestra los resultados numéricos obtenidos por el algoritmo.

A diferencia del caso de prueba anterior, se han encontrado dos soluciones que utilizan un número dis-

TABLA I: Resultados numéricos de dos casos de prueba distintos

Caso de prueba	*RumCoke & Case2*				*Case3 & Case4*
Entorno	E1	E2	E3	E4	–
NSimplex	2258	1616	2822	2178	3088
NVertex	685	515	874	690	912
NVertexF	44	77	99	79	528
EndNSimplex	363	324	206	0	115
EndNVertex	255	220	144	0	100
ε-Infeas.	455	263	485	404	194
Pareto	0	38	2	0	135
SCTest	228	141	265	243	97
MCtest	84	36	84	77	43
ξ-θ-ε-Infeas.	4	3	3	3	0
PCTest	0	0	0	0	0
LC	2	2	2	2	0
Capacity	0	0	26	27	4
FinalPareto	0	8	0	0	265
FinalCapacity	0	0	345	394	698
Time	< 1 s	< 1 s	< 1 s	< 1 s	< 1 s
Memory	321 KB	250 KB	405 KB	313 KB	548 KB
N.Sol.	0, 0, 0, 0	0, 0, 0, 1	0, 0, 0, 1	0, 0, 0, 0	0, 0, 1, 1

tinto de materias primas: $(x^{\star[1]}_{*,1}, x^{\star[1]}_{*,2})$ y $(x^{\star[2]}_{*,1}, x^{\star[2]}_{*,2})$. La primera de ellas (de color amarillo) emplea tres materias primas (RM2, RM3 y RM4):

$$\mathbf{x}^\star = \begin{array}{c} \\ \text{RM1} \\ \text{RM2} \\ \text{RM3} \\ \text{RM4} \end{array} \begin{array}{c} \begin{array}{cc} \textit{Case3} & \textit{Case4} \end{array} \\ \begin{pmatrix} 0.0 & 0.0 \\ 0.823125 & 0.524101 \\ 0.0 & 0.374805 \\ 0.176875 & 0.101094 \end{pmatrix} \end{array}$$

Su coste es de $f(x^{\star[1]}_{*,1}, x^{\star[1]}_{*,2}) = 1.283688 + 1.146051 = 2.429739$. La segunda solución (de color magenta) emplea las cuatro materias primas:

$$\mathbf{x}^\star = \begin{array}{c} \\ \text{RM1} \\ \text{RM2} \\ \text{RM3} \\ \text{RM4} \end{array} \begin{array}{c} \begin{array}{cc} \textit{Case3} & \textit{Case4} \end{array} \\ \begin{pmatrix} 0.224609 & 0.0 \\ 0.775391 & 0.573867 \\ 0.0 & 0.349922 \\ 0.0 & 0.076211 \end{pmatrix} \end{array}$$

Su coste es de $f(x^{\star[2]}_{*,1}, x^{\star[2]}_{*,2}) = 0.565234 + 1.056473 = 1.621707$.

Existen dos soluciones porque el coste de la solución de tres materias primas es mayor que el coste de la solución de cuatro materias primas. Ambas soluciones son Pareto-óptimas porque el algoritmo minimiza el coste y el número de materias primas empleadas.

La figura 11 muestra el espacio de búsqueda para este caso de prueba. En esta figura, también se puede observar que muchos símplices de un tamaño cercano a α han sido rechazados por el algoritmo 3. Los límites inferiores del coste y de la cantidad de materia prima RM2 usados durante la ejecución del algoritmo han sido muy subestimados.

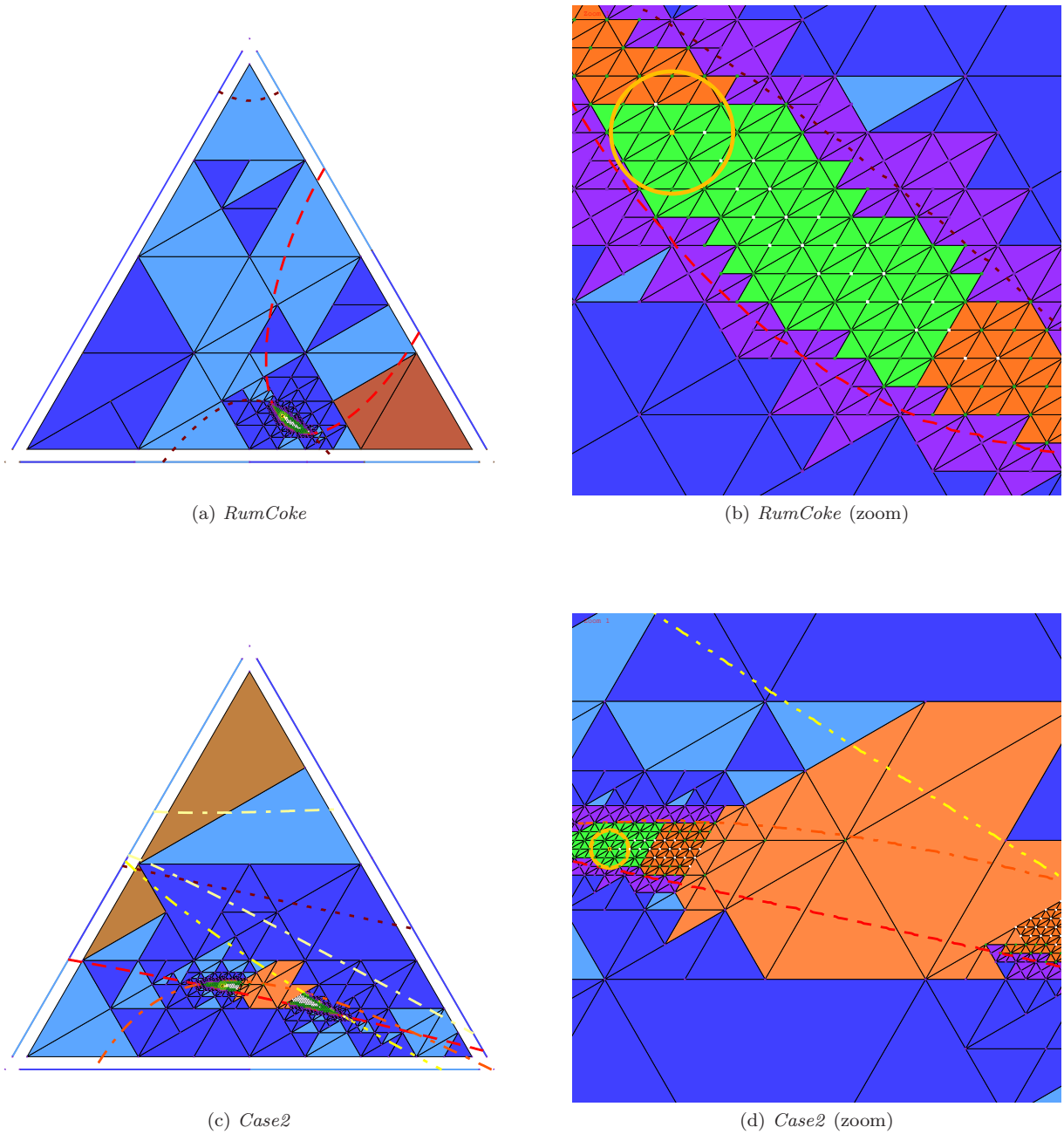
IV. Estrategia paralela

El problema de bi-mezcla se resuelve en dos fases independientes: en la fase B&B se obtienen las listas Ω_1 y Ω_2 con todos los símplices que alcanzaron el criterio de finalización (algoritmo 2) y en la fase Comb. se filtran aquellos símplices que no pueden contener ninguna solución (algoritmo 3). Las características computacionales de los algoritmos 2 y 3 son completamente diferentes y, además, el algoritmo 3 no puede empezar a ejecutarse hasta que el algoritmo 2 haya finalizado. Por lo tanto, los modelos paralelos de ambos algoritmos serán analizados por separado.

El número de símplices finales obtenidos por el algoritmo 2 depende de varios factores: la dimensión del problema, la precisión α de la regla de terminación y el tamaño de la región factible de cada producto. Experimentos preliminares muestran que este número puede ser relativamente alto. Por lo tanto, el algoritmo 3 puede ser computacionalmente más costoso que el algoritmo 2, siendo la paralelización de esta fase la principal prioridad de este trabajo.

El algoritmo 3 hace uso de un bucle anidado y de dos listas Ω_1 y Ω_2. Para cada símplex $C \in \Omega_j$, se busca un símplex $C' \in \Omega_{j'}$ que satisfaga (20) y (21) de forma que C no se elimine. En el peor de los casos (cuando el símplex C se elimina), la lista $\Omega_{j'}$ se explora completamente (todos los símplices $C' \in \Omega_{j'}$ son visitados).

Para evitar contención entre hebras en el acceso a la lista enlazada Ω_j, los símplices no son rechazados (véase la línea 7 del algoritmo 3) sino marcados para ser eliminados posteriormente. De otro modo, la lista podría ser modificada por varias hebras cuando los símplices son eliminados, haciendo necesario el uso



(a) *RumCoke* (b) *RumCoke* (zoom)

(c) *Case2* (d) *Case2* (zoom)

Fig. 10: *RumCoke* & *Case2* con una restricción de disponibilidad en RM1 ($B_1 = 0.86$)

de exclusión mutua.

La siguiente notación es necesaria para la descripción de las distintas estrategias:

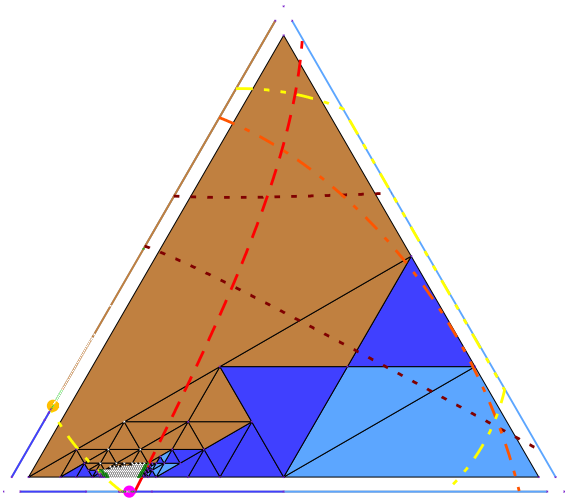
- $\text{Pos}(C, \Omega_j)$: posición del símplex C en Ω_j.
- NTh: número total de hebras.
- Th: número identificador de la hebra. Los números de identificación son consecutivos y comienzan en cero.

Existen varias formas de paralelizar el algoritmo 3 mediante el uso de hebras. En este artículo abordaremos dos estrategias [22]:

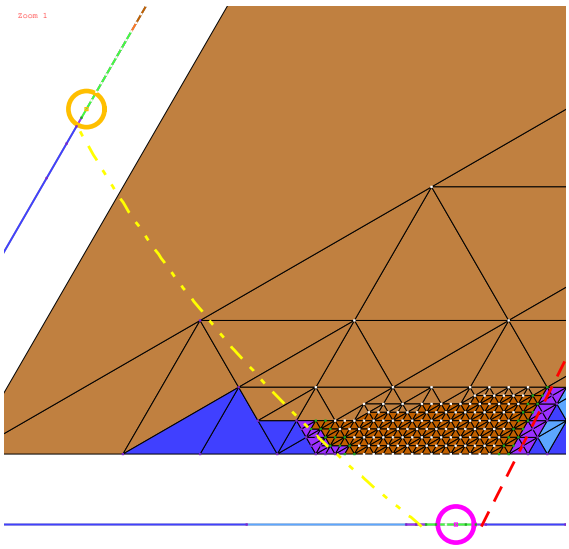
Estrategia 1: Asignar $NTh/2$ hebras a cada lista Ω_j, $j = 1, 2$. De este modo, las iteraciones 1 y 2 del bucle exterior se llevan a cabo en paralelo, trabajando sobre las dos listas simultáneamente. Esta estrategia requiere que $NTh \geq 2$. Cada hebra Th analiza los símplices en $\{C \in \Omega_j : \text{Pos}(C, \Omega_j) \text{ mód } (NTh/2) = Th \text{ mód } (NTh/2)\}$. Después de recorrer ambas listas, el borrado de los nodos de las listas Ω_j, $j = 1, 2$, es realizado por una hebra para cada lista.

Estrategia 2: Asignar NTh hebras al bucle interior para llevar a cabo una iteración del bucle exterior. Cada hebra Th analiza los símplices $C \in \Omega_j$ que satisfacen $\text{Pos}(C, \Omega_j) \text{ mód } NTh = Th$. Ahora se pretende procesar sólo una lista en paralelo, borrando los símplices no factibles antes de procesar la siguiente lista en paralelo. El borrado de los nodos (marcados para tal fin) de Ω_j es realizado por una sola hebra al final de cada iteración j.

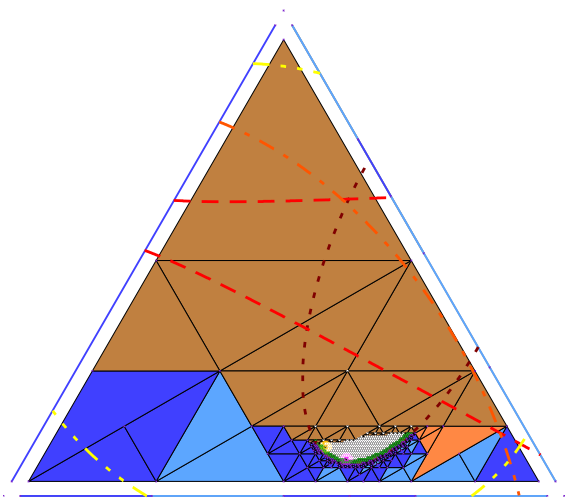
Una dificultad de paralelizar el algoritmo 2 es que



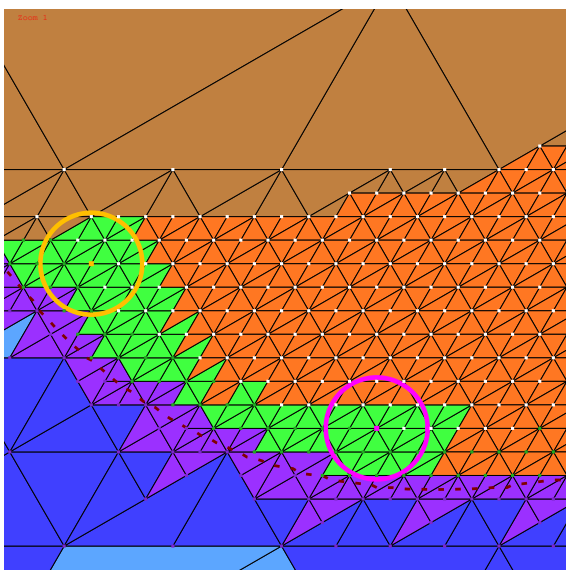
(a) *Case3*



(b) *Case3* (zoom)



(c) *Case4*



(d) *Case4* (zoom)

Fig. 11: *Case3* & *Case4* con una restricción de disponibilidad en RM2 ($B_2 = 1.35$)

el trabajo computacional pendiente de la búsqueda B&B en un producto no es conocido de antemano. La búsqueda de un producto está influenciada por la información compartida con el otro producto. Además, el coste computacional de la búsqueda en cada producto puede ser bastante desigual debido a los distintos requisitos de diseño. Un estudio de la predicción del trabajo en algoritmos de Ramificación y Acotación para problemas de Optimización Global basado en Aritmética de Intervalos se puede encontrar en [23]. Aunque los autores de este trabajo presentan algoritmos paralelos de Ramificación y Acotación en [18], [24], [25] y [26], estos artículos estudian la paralelización de un único algoritmo B&B. Sin embargo, la resolución del problema de bi-mezcla se puede ver como dos instancias del mismo algoritmo B&B, una para cada producto, que comparten $\beta^L_{i,j}$, $\varphi^L_{u,j}$ y f^U_p (véanse las ecuaciones 16, 17, 18 y 19). El problema consiste en determinar cuántas hebras se dedican a cada producto si queremos que ambas ejecuciones B&B paralelas consuman aproximadamente el mismo (o similar) tiempo computacional. Esto será resuelto en trabajos futuros.

Resultados preliminares muestran que la fase B&B es insignificante comparada con la fase Comb. Por lo tanto, utilizaremos una hebra por producto para mostrar la dificultad de la fase B&B. Esto nos permitirá ilustrar el reto del balanceo de carga.

V. Resultados experimentales

Para evaluar el rendimiento de las distintas versiones del algoritmo paralelo, se han utilizado un par de productos de cinco dimensiones, llamados UniSpec1-5 y UniSpec5b-5. Ambos han sido adapta-

dos de dos productos de siete dimensiones (UniSpec1 y UniSpec5b, respectivamente) tomados de [13], eliminando las materias primas 6 y 7. Este problema ha sido resuelto con una robustez $\varepsilon = \sqrt{2}/100$, una precisión $\alpha = \varepsilon$ y una dosis mínima $md = 0.03$. La demanda de cada producto es $D^T = (1, 1)$. La disponibilidad de la materia prima uno (RM1) y de RM3 está restringida a 0.62 y 0.6, respectivamente; mientras las otras materias no tienen limitación alguna.

Para el problema de mezcla descrito anteriormente (UniSpec1-5 & UniSpec5b-5), se han encontrado dos soluciones con un número diferente de materias primas involucrado: $(x^{\star[1]}_{*,1}, x^{\star[1]}_{*,2})$ y $(x^{\star[2]}_{*,1}, x^{\star[2]}_{*,2})$. La primera de ellas hace uso de cuatro materias primas (RM1, RM3, RM4 y RM5):

$$\mathbf{x}^{\star[1]} = \begin{array}{c} \\ \text{RM1} \\ \text{RM2} \\ \text{RM3} \\ \text{RM4} \\ \text{RM5} \end{array} \begin{array}{c} \begin{array}{cc} \text{UniSpec1-5} & \text{UniSpec5b-5} \end{array} \\ \begin{pmatrix} 0.428125 & 0.146875 \\ 0.0 & 0.0 \\ 0.435234 & 0.164063 \\ 0.0 & 0.232812 \\ 0.136641 & 0.456250 \end{pmatrix} \end{array}$$

El coste de esta solución es $f_{bi}(x^{\star[1]}_{*,1}, x^{\star[1]}_{*,2}) = 111.09 + 116.33 = 227.42$. La segunda solución involucra las cinco materias primas:

$$\mathbf{x}^{\star[2]} = \begin{array}{c} \\ \text{RM1} \\ \text{RM2} \\ \text{RM3} \\ \text{RM4} \\ \text{RM5} \end{array} \begin{array}{c} \begin{array}{cc} \text{UniSpec1-5} & \text{UniSpec5b-5} \end{array} \\ \begin{pmatrix} 0.428125 & 0.156172 \\ 0.0 & 0.03 \\ 0.442344 & 0.152852 \\ 0.0 & 0.212617 \\ 0.129531 & 0.448359 \end{pmatrix} \end{array}$$

El coste de esta solución es $f_{bi}(x^{\star[2]}_{*,1}, x^{\star[2]}_{*,2}) = 111.03 + 116.17 = 227.20$.

Los algoritmos se han codificado en C y han sido evaluados en una máquina Dell PowerEdge R810 con un procesador Intel Xeon 1.87 GHz de ocho núcleos, 16 GBytes de RAM y sistema operativo Linux con kernel 2.6. Para la creación y el manejo de las hebras, se ha utilizado la librería de POSIX Threads.

La tabla II proporciona información sobre el esfuerzo computacional realizado por el algoritmo secuencial (BiBlendSeq) y el algoritmo paralelo (BiBlendPar) en términos de: número de símplices (NEvalS) y vértices (NEvalV) evaluados, número de símplices rechazados por restricciones lineales, cuadráticas o de robustez (QLR), test de Pareto (Pareto) y test de disponibilidad (Capacity). $|\Omega_S|$ y $|\Omega_V|$ dan, respectivamente, la suma del número de símplices y vértices en las listas $\Omega_j,\ j = 1, 2$.

Se puede apreciar que las diferencias existentes entre la versión secuencial y la paralela son insignificantes, así que no existen anomalías decrementales ni incrementales en la fase B&B. La fase Comb. es capaz de reducir drásticamente (casi un 50 %) el espacio de búsqueda (número de símplices). Esto alimenta la idea de usar el algoritmo completo de una forma iterativa para refinar las soluciones del problema de bi-mezcla usando decrementos en la precisión solicitada hasta alcanzar el valor de α.

TABLA II: Esfuerzo computacional

Fase B&B

	BiBlendSeq	BiBlendPar
NEvalS	2536862	2537430
NEvalV	168186	168299
QLR	887609	888004
Pareto	54050	54050
Capacity	18277	18211
$\|Q_S\|$	308443	308465
$\|Q_V\|$	49317	49324

Fase Comb

	BiBlendSeq	BiBlendPar
Pareto	27284	27284
Capacity	105499	105521
$\|Q_S\|$	175660	175660
$\|Q_V\|$	24861	24861

La tabla III muestra el tiempo de ejecución de la fase Comb. en BiBlendSeq y BiBlendPar ($NTh = 2, 4, 8$) aplicando las estrategias descritas en la sección IV. En la Estrategia 2, se dan los resultados cuando se empieza el filtrado de Ω_1 y después el de Ω_2 (Ω_1 – Ω_2) o viceversa (Ω_2 – Ω_1). La aceleración con respecto al tiempo de ejecución de un algoritmo paralelo con p unidades de proceso se define como $S(p) = t(1)/t(p)$, donde $t(p)$ es el tiempo de ejecución cuando p hebras son utilizadas.

La Estrategia 1 exhibe una aceleración pobre en comparación con la Estrategia 2. La Estrategia 1 se basa en hebras trabajando en los elementos de Ω_1 y Ω_2, en donde para cada elemento de una lista, la comparación es ejecutada con elementos de la otra lista hasta que una combinación válida es encontrada o la otra lista ha sido comparada por completo (el peor caso). Esto requiere que muchos elementos de ambas listas tengan que ser almacenados en caché, ocasionando fallos de caché.

Por otra parte, la Estrategia 2 utiliza todas las hebras para evaluar los elementos de una lista. De esta manera, sólo el número de elementos de la lista actual (igual a NTh) y los elementos de la otra lista tienen que estar en caché para compararse. Esto reduce el número de fallos de caché y como consecuencia el tiempo de ejecución, más aún cuando la otra lista tiene un tamaño reducido. Este hecho se ilustra cuando la Estrategia 2 empieza con la lista Ω_2 de tamaño $|\Omega_2| = 286475$. La otra lista tiene un tamaño reducido $|\Omega_1| = 21990$, lo que puede incrementar la posibilidad de tener los mismos símplices para ser comparados en caché. El tiempo de ejecución se reduce en más del 25 %.

En cuanto a la fase B&B, que es la misma en ambas estrategias, BiBlendPar utiliza $NTh = 2$. En esta fase, se obtiene una ligera aceleración igual a 1.03: BiBlendSeq tarda 7.23 segundos y BiBlendPar tarda 7 segundos. No se alcanza una aceleración lineal debido a la diferencia de complejidad entre los dos productos: UniSpec1-5 tiene una región factible definida por unas restricciones cuadráticas de me-

TABLA III: Aceleración obtenida en la fase Comb.

	Estrategia 1		Estrategia 2 ($\Omega_1 - \Omega_2$)		Estrategia 2 ($\Omega_2 - \Omega_1$)	
NTh	Time	Speedup	Time	Speedup	Time	Speedup
–	1991.46	–	1991.46	–	1453.52	–
2	1428.88	1.39	956.01	2.08	693.57	2.09
4	696.39	2.86	465.02	4.28	340.44	4.27
8	338.92	5.88	228.63	8.71	168.44	8.63

nor complejidad que UniSpec5b-5; la hebra $Th = 1$ sólo tarda 0.86 segundos en explorar todo el espacio de búsqueda de UniSpec1-5, mientras que la hebra $Th = 2$ tarda 7 segundos en finalizar la exploración del espacio de búsqueda de UniSpec5b-5.

VI. Conclusiones

Se ha estudiado la paralelización de un algoritmo con la finalidad de resolver el problema de diseño de dos productos obtenidos mediante mezcla de materias primas para un problema de tamaño medio. Este caso particular muestra las dificultades de este tipo de algoritmos. El algoritmo de bi-mezcla incrementa los retos de la paralelización debido a la utilización de dos algoritmos B&B que comparten información. Además, en los algoritmos de bi-mezcla, se ha de realizar una combinación de símplices finales después de la fase B&B para descartar regiones no factibles. Esta fase de combinación es computacionalmente varios órdenes de magnitud más costosa que la fase B&B. Por ello, aquí sólo se ha utilizado una hebra por producto en la fase B&B y varias hebras en la fase de combinación. Los resultados muestran aceleraciones lineales en una máquina de ocho núcleos con memoria compartida cuando se paraleliza el recorrido de una lista final y después el de la otra, en vez de realizar ambos recorridos en paralelo.

Nuestra intención es experimentar con problemas de mayor dimensión, intentando reducir su coste computacional. Otra línea de investigación a continuar es desarrollar el algoritmo de multi-mezcla y su versión paralela, problema que resulta de interés para la industria.

Apéndice

RumCoke

Dimensión = 3;
Materias primas presentes = {RM1, RM2, RM4};
Coste de las materias primas = $(0.1, 0.7, 4.0)$;
Restricciones lineales:
$h_1(x) = -1.5x_1 + 0.5x_2 - 0.5x_3 \geq 0.0$
$h_2(x) = 0.3x_1 - 0.5x_2 - 0.3x_3 \geq 0.0$
Restricciones cuadráticas
$\big(g_i(x) = x^T A_i x + b_i^T x + d_i \leq 0;\ i = 1, 2\big)$.

Case2

Dimensión = 3;
Materias primas presentes = {RM1, RM2, RM3};
Coste de las materias primas = $(0.1, 0.7, 1.0)$;
Restricciones cuadráticas
$\big(g_i(x) = x^T A_i x + b_i^T x + d_i \leq 0;\ i = 3, \ldots, 7\big)$.

Case3

Dimensión = 3;
Materias primas presentes = {RM1, RM2, RM4};
Coste de las materias primas = $(0.1, 0.7, 4.0)$;
Restricciones cuadráticas
$\big(g_i(x) = x^T A_i x + b_i^T x + d_i \leq 0;\ i = 7, \ldots, 10\big)$.

Case4

Dimensión = 3;
Materias primas presentes = {RM2, RM3, RM4};
Coste de las materias primas = $(0.7, 1.0, 4.0)$;
Restricciones cuadráticas
$\big(g_i(x) = x^T A_i x + b_i^T x + d_i \leq 0;\ i = 2, 7, 9, 11\big)$.

UniSpec1-5

Dimensión = 7;
Materias primas presentes =
{RM5, RM6, RM7, RM8, RM9};
Coste de las materias primas =
$(114, 115, 107, 127, 115)$;
Restricción lineal:
$h_1(x) = 0.1493x_1 + 0.6927x_2 + 0.4643x_3 + 0.7975x_4 + 0.5967x_5 \geq 0.35$
Restricciones cuadráticas
$\big(g_i(x) = x^T A_i x + b_i^T x + d_i \leq 0;\ i = 12, \ldots, 14\big)$.

UniSpec5b-5

Dimensión = 7;
Materias primas presentes =
{RM5, RM6, RM7, RM8, RM9};
Coste de las materias primas =
$(114, 115, 107, 127, 115)$;
Restricciones cuadráticas
$\big(g_i(x) = x^T A_i x + b_i^T x + d_i \leq 0;\ i = 15, \ldots, 18\big)$.

Restricciones cuadráticas

$A_1[3 \times 3] = (0, -16, 0, -16, 0, 0, 0, 0, 0)$;
$b_1[3 \times 1] = (8, 8, 0)$; $d_1 = -1$;
$A_2[3 \times 3] = (10, 0, 2, 0, 0, 0, 2, 0, 2)$;
$b_2[3 \times 1] = (-12, 0, -4)$; $d_2 = 3.7$;
$A_3[3 \times 3] = (0.001,\ -0.001,\ 0.0085,\ -0.001,\ 0.008,\ -0.0105,\ 0.0085,\ -0.0105,\ -0.021)$;
$b_3[3 \times 1] = (-0.0145, -0.0205, 0.073)$; $d_3 = -0.0165$;
$A_4[3 \times 3] = (-0.004,\ 0.0005,\ 0.002,\ 0.0005,\ -0.001,\ -0.003,\ 0.002,\ -0.003,\ 0.014)$;
$b_4[3 \times 1] = (0.0155, 0.0515, -0.121)$; $d_4 = -0.006$;
$A_5[3 \times 3] = (20.605,\ -5.087,\ -10.9885,\ -5.087,\ 32.003,\ -43.476,\ -10.9885,\ -43.476,\ -81.278)$;
$b_5[3 \times 1] = (0.1995,\ -0.097,\ 126.7685)$;
$d_5 = -20.5063$;
$A_6[3 \times 3] = (0.766,\ -0.1205,\ 2.4735,\ -0.1205,\ 0.528,$

1.9835, 2.4735, 1.9835, −7.822);
$b_6[3 \times 1] = (-2.432, -15.191, 10.712)$; $d_6 = 3.21125$;
$A_7[3 \times 3] = (116.75, -3.09, 168.553, -3.09, -67.424, 515.114, 168.553, 515.114, -845.215)$;
$b_7[3 \times 1] = (-287.43, -645.926, 354.537)$;
$d_7 = 115.0953$;
$A_8[3 \times 3] = (1.0, 3.0, -0.5, 3.0, -5.0, -3.5, -0.5, -3.5, -2.0)$;
$b_8[3 \times 1] = (0.832, 0.832, 0.832)$; $d_8 = 0.968$;
$A_9[3 \times 3] = (2.0, -1.5, 1.0, -1.5, 1.0, -1.0, 1.0, -1.0, 3.0)$;
$b_9[3 \times 1] = (0.12, 0.12, 0.12)$; $d_9 = -1.60$;
$A_{10}[3 \times 3] = (4.0, -1.5, -1.5, -1.5, 4.0, -2.5, -1.5, -2.5, 4.0)$;
$b_{10}[3 \times 1] = (-0.026, -0.026, -0.026)$; $d_{10} = -2.141$;
$A_{11}[3 \times 3] = (4.0, -1.0, -2.0, -1.0, 5.0, -3.0, -2.0, -3.0, 4.0)$;
$b_{11}[3 \times 1] = (-0.019, -0.019, -0.019)$; $d_{11} = -2.631$;
$A_{12}[5 \times 5] = (-1.473, 8.215, -27.204, 46.119, 2.059, -11.929, -12.768, 8.215, 37.733346, 5.127, 95.691, 34.954, 20.165, 19.445, -27.204, 5.127, -21.743, 36.843, -7.126, 4.029, -4.152, 46.119, 95.691, 36.843, 189.643, 93.359, 52.904, 54.802, 2.059, 34.954, -7.126, 93.356, 31.885, 7.528, 10.248, -11.929, 20.165, 4.029, 52.904, 7.528, 11.951, 10.964, -12.768, 19.445, -4.152, 54.802, 10.248, 10.964, 7.197)$;
$b_{12}[5 \times 1] = (4.5675, 34.7289, 70.5707, -82.2761, 29.3169)$; $d_{12} = -35$;
$A_{13}[5 \times 5] = (1.35, -4.41, 17.60, -92.45, 2.74, -29.94, -14.05, -4.41, -39.13, -6.11, -126.38, -29.81, -63.42, -43.97, 17.60, -6.11, 15.45, -76.60, 5.93, -44.05, -20.54, -92.45, -126.38, -76.60, -240.64, -117.46, -125.18, -114.98, 2.74, -29.81, 5.93, -117.46, -22.90, -47.37, -30.68, -29.94, -63.42, -44.05, -125.18, -47.37, -73.39, -73.99, -14.05, -43.97, -20.54, -114.98, -30.68, -73.99, -55.33)$;
$b_{13}[5 \times 1] = (-2.1232, -9.0403, -42.2072, 190.5292, -9.9529)$; $d_{13} = 10$;
$A_{14}[5 \times 5] = (-0.670, 4.284, -12.837, 23.708, 1.677, -8.964, -4.859, 4.284, 21.380, -1.188, 28.990, 13.216, 17.177, 16.620, -12.837, -1.189, -21.376, 9.841, -7.298, -10.043, -8.981, 23.708, 28.990, 9.841, 49.385, 25.574, 15.561, 21.666, 1.677, 13.216, -7.298, 25.574, 8.419, 4.149, 6.595, -8.965, 17.177, -10.043, 15.561, 4.149, 1.090, 6.292, -4.859, 16.620, -8.981, 21.666, 6.594, 6.292, 5.906)$;
$b_{14}[5 \times 1] = (0.7097, -13.0982, 27.5078, -49.1608, -7.3725)$; $d_{14} = -2$;
$A_{15} = -A_{12}$; $b_{15} = -B_{12}$; $d_{15} = 45$;
$A_{16} = -A_{13}$; $b_{16} = -B_{13}$; $d_{16} = -21$;
$A_{17}[5 \times 5] = (0.0, -11.556, -1.114, 14.690, -11.411, 0.121, -0.150, -11.556, -3.316, -2.116, 7.313, -8.800, 19.897, 9.051, -1.114, -2.116, 4.728, 16.250, -4.535, 18.319, 11.537, 14.690, 7.313, 16.250, 40.428, 9.766, 21.512, 15.266, -11.412, -8.800, -4.535, 9.766, -10.165, 10.088, 1.889, 0.121, 19.897, 18.319, 21.511, 10.088, 28.569, 27.239, -0.150, 9.051, 11.537, 15.266, 1.889, 27.239, 19.965)$;
$b_{17}[5 \times 1] = (1.7278, 23.5166, 5.6724, -32.0798, 19.0154)$; $d_{17} = -5$;
$A_{18} = A_{14}$; $b_{18} = B_{14}$; $d_{18} = -1$

Referencias

[1] I. Das and J.E. Dennis, "A closer look at drawbacks of minimizing weighted sums of objectives for Pareto set generation in multicriteria optimization problems," *Structural and Multidisciplinary Optimization*, vol. 14, no. 1, pp. 63–69, 1997.

[2] A. Neumaier, "Complete search in continuous global optimization and constraint satisfaction," *Acta Numerica*, vol. 13, pp. 271–369, 2004.

[3] J.D.C. Little, K.G. Murty, D.W. Sweeney, and C. Karel, "An algorithm for the traveling salesman problem," *Operations Research*, vol. 11, no. 6, pp. 972–989, 1963.

[4] E.L. Lawler and D.E. Wood, "Branch-and-bound methods: a survey," *Operations Research*, vol. 14, no. 4, pp. 699–719, 1966.

[5] L.G. Mitten, "Branch and bound methods: general formulation and properties," *Oper. Res.*, vol. 18, no. 1, pp. 24–34, 1970.

[6] T. Ibaraki, "Theoretical comparisons of search strategies in branch and bound algorithms," *Int. J. Comput. Inf. Sci.*, vol. 5, no. 4, pp. 315–344, 1976.

[7] F.C.G. López, *Programación en paralelo y técnicas algorítmicas*, Ph.D. thesis, Universidad de La Laguna, 1995.

[8] R. Horst, P.M. Pardalos, and N.V. Thoai, *Introduction to Global Optimization*, vol. 3 of *Noncovex Optimization and its Applications*, Kluwer Academic Publishers, Dordrecht, 1995.

[9] J. Ashayeri, A.G.M. van Eijs, and P. Nederstigt, "Blending modelling in a process manufacturing: A case study," *Eur. J. Oper. Res.*, vol. 72, no. 3, pp. 460–468, 1994.

[10] J.W.M. Bertrand and W.G.M.M. Rutten, "Evaluation of three production planning procedures for the use of recipe flexibility," *Eur. J. Oper. Res.*, vol. 115, no. 1, pp. 179–194, 1999.

[11] H.P. Williams, *Model Building in Mathematical Programming*, Wiley & Sons, Chichester, 1993.

[12] L.G. Casado, E.M.T. Hendrix, and I. García, "Infeasibility spheres for finding robust solutions of blending problems with quadratic constraints," *J. Global Optim.*, vol. 39, no. 4, pp. 577–593, 2007.

[13] E.M.T. Hendrix, L.G. Casado, and I. García, "The semi-continuous quadratic mixture design problem: Description and branch-and-bound approach," *Eur. J. Oper. Res.*, vol. 191, no. 3, pp. 803–815, 2008.

[14] E.M.T. Hendrix and J.D. Pintér, "An application of Lipschitzian global optimization to product design," *Journal of Global Optimization*, vol. 1, no. 4, pp. 389–401, 1991.

[15] E.M.T. Hendrix, C.J. Mecking, and T.H.B. Hendriks, "Finding robust solutions for product design problems," *Eur. J. Oper. Res.*, vol. 92, no. 1, pp. 28–36, 1996.

[16] L.G. Casado, I. García, J.A. Martínez, and E.M.T. Hendrix, "Shared memory parallel exhaustive search of epsilon-robust mixture design solutions," in *Volume of Abstracts of 22nd European Conference on Operational Research (EURO XXII)*, 2007, p. 178.

[17] B. Gendron and T.G. Crainic, "Parallel branch-and-bound algorithms: Survey and synthesis," *Oper. Res.*, vol. 42, no. 6, pp. 1042–1066, 1994.

[18] L.G. Casado, J.A. Martínez, I. García, and E.M.T. Hendrix, "Branch-and-bound interval global optimization on shared memory multiprocessors," *Optim. Method. Softw.*, vol. 23, pp. 689–701, 2008.

[19] J. Claussen and A. Zilinskas, "Subdivision, sampling and initialization strategies for simplicial branch and bound in global optimization," *Comput. Math. Appl.*, vol. 44, pp. 943–955, 2002.

[20] R. Horst, "On generalized bisection of n-simplices," *Math. Comput.*, vol. 66, no. 218, pp. 691–698, 1997.

[21] J.F.R. Herrera, L.G. Casado, E.M.T. Hendrix, and I. García, "Pareto optimality and robustness in bi-blending problems," *TOP*, 2011, Submitted.

[22] J.F.R. Herrera, L.G. Casado, E.M.T. Hendrix, and I. García, "A threaded approach of the quadratic bi-blending algorithm," *J. Supercomput.*, 2011, Submitted.

[23] J.L. Berenguel, L.G. Casado, I. García, and E.M.T. Hendrix, "On estimating workload in interval branch-and-

bound global optimization algorithms," *J. Global Optim.*, 2011, Submitted.

[24] J.F.S. Estrada, L.G. Casado, and I. García, "Adaptive parallel interval global optimization algorithms based on their performance for non-dedicated multicore architectures," in *Parallel, Distributed and Network-Based Processing (PDP), 2011 19th Euromicro International Conference on*, February 2011, pp. 252–256.

[25] J.A. Martínez, L.G. Casado, J.A. Alvarez, and I. García, "Interval parallel global optimization with Charm++," in *Applied Parallel Computing. State of the Art in Scientific Computing*, Jack Dongarra, Kaj Madsen, and Jerzy Wasniewski, Eds., vol. 3732 of *Lecture Notes in Computer Science*, pp. 161–168. Springer Berlin / Heidelberg, 2006.

[26] J.F. Sanjuan-Estrada, L.G. Casado, and I. García, "Adaptive parallel interval branch and bound algorithms based on their performance for multicore architectures," *J. Supercomput.*, pp. 1–9, 2011.