**UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO**

UNIDAD ACADEMICA DE LOS CICLOS PROFESIONAL
Y DE POSGRADO
DEL COLEGIO DE CIENCIAS Y HUMANIDADES
INSTITUTO DE INVESTIGACIONES EN MATEMATICAS
APLICADAS Y SISTEMAS

# BUSQUEDA TABU PARA LA CONSTRUCCION DE DISEÑOS DE EXPERIMENTOS

## T E S I S

QUE PARA OBTENER EL GRADO DE :

MAESTRO EN CIENCIAS DE LA COMPUTACION

P R E S E N T A :

MIGUEL ZAMUDIO MONTAÑO

DIRECTOR: DR. LUIS B. MORALES MENDOZA

MEXICO, D. F. ENERO 1995


**UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso**

**Agradecimientos:**

- A mis padres Miguel Zamudio Chávez e Inés Montaño García, y a mis hermanos por todo el amor y el apoyo que siempre me han brindado.

- Al Dr. Luis B. Morales Mendoza, por haber dirigido esta tesis. Al Ing. Mario Rodríguez Manzanera por sus sugerencias. Al Dr. Sergio G. de los Cobos Silva, por ampliar nuestros conocimientos sobre la técnica de Búsqueda Tabú. Al Dr. Ramón Garduño Juárez y al M. en C. Felipe Maldonado por haber aceptado ser mis sinodales en el examen de grado.

- Al M. en C. J. Refugio Vallejo Gutiérrez, quien participó activamente en el seminario de paralelismo e inició la programación en la computadora CM-5. Al Act. Carlos Ordóñez Mondragón por haber realizado una revisión del manuscrito del trabajo.

  Muy especialmente a la Dra. Hanna Oktaba, quien ha apoyado al programa de la Maestría en Ciencias de la Computación y me alentó para obtener el grado.

- A mi amigo el Act. José Luis Meraz Ríos por haberme animado a estudiar la Maestría.

- A todos mis maestros y amigos de la Maestría.

*Miguel Zamudio Montaño*
México D.F., enero de 1995

# Contenido

# Capítulo 1

# Introducción

### 1.0.1 Problemas de optimización combinatoria

Muchos investigadores han estudiado el problema de buscar soluciones óptimas a problemas que pueden ser estructurados como una función de algunas *variables de control*, y tal vez con la presencia de algunas *restricciones*. Tales problemas pueden ser formulados de la siguiente manera:

$$\text{Minimizar } f(x) \text{ sujeta a:}$$

$$g_i(x) \geq b_i; \quad i = 1, \ldots, m;$$

$$h_j(x) = c_j; \quad j = 1, \ldots, n,$$

donde $x$ es un vector de variables de control y $f(\cdot), g_i(\cdot)$ y $h_j(\cdot)$ son funciones generales.

Existen muchas clases específicas de tales problemas, obtenidos aplicando restricciones sobre el tipo de funciones bajo consideración y en los valores que las variables de control pueden tomar. Quizás los problemas más conocidos son aquellos obtenidos restringiendo a que $f(\cdot)$, $g_i(\cdot)$ y $h_j(\cdot)$ sean funciones lineales de las variables de control, que están permitidas a tomar variables *continuas*, lo que nos conduce a problemas de *programación lineal*.

Nosotros consideraremos aquí otra clase de problemas: aquellos de naturaleza *combinatoria*. Este término es usualmente reservado para problemas en los cuales las variables de control son *discretas*. El problema de buscar

una solución óptima a tales problemas es conocido como *optimización combinatoria.*

Los problemas de optimización combinatoria tienen nexos cercanos a la programación lineal y muchos de los intentos para resolverlos utilizan estos métodos, generalmente introduciendo variables enteras que toman valores de 0 ó 1, para producir una formulación de *programación entera.* Tales formulaciones en ocasiones involucran un gran número de variables y restricciones, y no pueden hacer frente a problemas muy grandes.

Un ejemplo de un problema de optimización combinatoria es el siguiente:

**Problema de Asignación.** Un conjunto de $n$ personas está disponible para realizar $n$ tareas. Si la $i$-ésima persona realiza la $j$-ésima tarea, esto representa un costo de $C_{ij}$ unidades. Entonces el problema consiste en encontrar una asignación $\pi_1, \ldots, \pi_n$ la cual minimice:

$$\sum_{i=1}^{n} C_{i\pi_i},$$

aquí la solución está representada por la *permutación* $\pi_1, \ldots, \pi_n$ de los números $1, \ldots, n$.

Debe quedar claro que en el ejemplo anterior, utilizamos el término *problema* en un sentido genérico; dada una situación en la vida real, tendremos un problema *particular*, en el cual los símbolos utilizados para describirlo deben tomar valores numéricos específicos. Es común usar el término *instancia* para poder distinguir entre una situación particular de una general.

### 1.0.2 Optimos locales y globales

Una característica de la mayoría de problemas de optimización combinatoria es que pueden tener muchos óptimos globales, aunque tienen muchos más que son únicamente óptimos locales. Podemos hacer más concreta esta idea introduciendo el concepto de *vecindad.*

Estrictamente hablando, una vecindad $V(x, \sigma)$ de una solución $x$ es un conjunto de soluciones que pueden ser alcanzadas desde $x$ aplicando una simple operación $\sigma$. Tal operación podría ser el quitar o agregar un objeto a

la solución o hacer el intercambio de dos objetos en la solución. Particularmente en la técnica de la *Búsqueda Tabú* , a estas operaciones se les conoce con el nombre de *movimientos*. Si una solución $y$ es mejor que cualquier otra solución en su vecindad $V(y, \sigma)$, entonces $y$ es un óptimo local con respecto a su vecindad.

En algunos casos es posible encontrar un movimiento $\sigma$, tal que un óptimo local sea también un óptimo global. Hay muchos tratamientos existentes para la optimización combinatoria, pero éstos tienden a concentrarse en métodos que son *exactos* en vez de *heurísticos*. Esto es, ellos están principalmente relacionados con aquellas técnicas que *garantizan* encontrar una solución óptima al problema establecido. Estos métodos usualmente están ligados a la teoría de la programación lineal, o usan métodos de *enumeración implícita* tales como el branch and bound [28].

### 1.0.3 Técnicas heurísticas

La técnica de la *Búsqueda Tabú* se conoce como una *técnica heurística* [25]. Este término deriva del griego *heuriskein* que significa encontrar o descubrir. Sin embargo, las técnicas heurísticas no garantizan encontrar una solución óptima. El término heurístico es usado en contraste a los métodos que sí garantizan encontrar una solución óptima global y ha llegado a ser común en el contexto de la optimización combinatoria.

- **Definición**. Una técnica heurística es aquella que ayuda a guiar al proceso de búsqueda mejorando en cada intento de aproximación su eficiencia. Sin embargo, no nos garantiza que encontremos una solución óptima, pero sí nos proporciona buenas soluciones (cercanas a la óptima) con un costo computacional razonable.

Una gran cantidad de artículos tratan de como las técnicas heurísticas se han utilizado para resolver problemas de optimización combinatoria. Pareciendo doble la causa de este gran interés:

1. Por un lado, el desarrollo del concepto de *complejidad computacional* ha proporcionado las bases para explorar las técnicas heurísticas.

2. Por otro lado, han surgido nuevas técnicas muy eficientes para resolver problemas de optimización combinatoria en tiempos de cómputo razonables.

El método clásico para resolver una instancia de un problema de optimización combinatoria es simplemente listar todas las soluciones factibles de la instancia dada, evaluar su función objetivo y escoger la mejor solución. Podríamos pensar que la solución de un problema de optimización combinatoria únicamente se limita a buscar de manera exhaustiva el valor que minimice o maximice la función objetivo dentro de un conjunto finito de posibilidades y que utilizando una computadora muy rápida, el problema carecería de interés matemático, sin detenernos a pensar por un momento en el tamaño de este conjunto finito de posibilidades.

Sin embargo, es obvio que este método de enumeración completa es muy ineficiente conforme crece el tamaño de la entrada del problema debido a la *explosión combinatoria* del espacio de soluciones, es decir, dado un conjunto de elementos se pueden obtener diferentes arreglos ordenados de estos, permitiendo una gran cantidad de posibilidades para cualquier entrada de tamaño no muy reducido (por ejemplo del orden de $n!$).

Para ilustrar este punto, considérese el problema clásico del *Agente Viajero* (*Travelling Salesman Problem*), el cual ha fascinado a los investigadores en optimización combinatoria, probablemente porque es muy fácil de describir, pero muy difícil de resolver. Sin embargo, se han utilizado ya técnicas heurísticas junto con técnicas de paralelización para resolver grandes instancias de este problema con éxito [8].

**Problema del Agente Viajero.** Un vendedor tiene que buscar una ruta que visite cada una de $N$ ciudades una y sólo una vez, que minimice el total de la distancia recorrida, iniciando en cualquiera de ellas y volviendo a la misma ciudad. Como el punto inicial es arbitrario, hay $(N-1)!$ posibles soluciones o $\frac{(N-1)!}{2}$ si la distancia entre cada par de ciudades es la misma sin considerar la dirección del viaje.

Supóngase que tenemos una computadora que puede listar todas las posibles soluciones de una instancia; entonces utilizando la fórmula anterior, tendríamos como resultado la Tabla 1.1 (ver [25] p. 7) que nos muestra el crecimiento del tiempo de cómputo con respecto al tamaño de la entrada del problema, en el cual podemos observar que la enumeración completa es ineficiente para obtener una solución óptima, ya que por ejemplo un problema de 25 ciudades tomaría aproximadamente 6 siglos en ser resuelto.

| Número de Ciudades | Tiempo de Cómputo |
|---|---|
| 20 | 1 hora |
| 21 | 20 horas |
| 22 | 17.5 días |
| 23 | 1.05 años |
| 24 | 24.26 años |
| 25 | 5.82 siglos |

Tabla 1.1: Crecimiento del tiempo de cómputo

Varios algoritmos exactos fueron inventados para encontrar soluciones óptimas a problemas mucho más eficientemente que la enumeración completa. El algoritmo más famoso es el *Simplex* para problemas de programación lineal. Estos algoritmos fueron capaces de resolver pequeñas instancias, pero no lo fueron para encontrar soluciones óptimas a grandes instancias en una cantidad razonable de tiempo computacional. Como el poder computacional se ha incrementado en los últimos años, ha sido posible resolver grandes problemas, y los investigadores se han interesado en cómo el tiempo de solución varía con el tamaño de la entrada del problema.

Sin embargo, para problemas tales como el del agente viajero, el esfuerzo computacional sigue siendo exponencial. A finales de los años 60's los investigadores se hacían la siguiente pregunta: *¿hay un algoritmo de optimización en tiempo polinomial para un problema tal como el del agente viajero?*. Nadie ha sido capaz de responder a esta pregunta, pero en 1972, *Karp* mostró que si la respuesta a esta pregunta es *sí* para el problema del agente viajero, entonces hay también un algoritmo en tiempo polinomial para otros problemas equivalentes. Como ningún algoritmo ha sido encontrado para estos problemas, esto indica que la respuesta a la pregunta original es *probablemente no*. Sin embargo, la respuesta real es desconocida.

Hay problemas que tienen algoritmos polinomiales conocidos y se dice que están en la clase $P$. Pero *¿qué hay del problema del agente viajero y otros problemas equivalentes: son de complejidad exponencial?*. Muchos de estos problemas están en la clase $NP$, que es una abreviación de *non-deterministic polynomial* [9]. De hecho el problema del agente viajero es de los problemas más difíciles en $NP$ (si un algoritmo polinomial fuera encontrado para este problema, esto significaría que existe un algoritmo polinomial para todos

los problemas en $NP$); pero como ningún algoritmo polinomial exacto ha sido encontrado para cualquier problema en $NP$, hay fuerte evidencia de que $P \neq NP$, lo cual es un argumento para buscar formas alternativas de resolver problemas computacionalmente difíciles. Existe la posibilidad de que alguien llegue a probar que $P = NP$; pero mientras nadie encuentre tal prueba, el uso de las técnicas heurísticas tiene una justificación considerable.

Existen otros argumentos en favor del uso de las técnicas heurísticas: lo que realmente se está optimizando es un modelo de un problema del mundo real, por eso no hay garantía de que la mejor solución del modelo sea también la mejor solución para el problema del mundo real. Las técnicas heurísticas son usualmente más flexibles y son capaces de hacer lo mismo con funciones objetivo y/o restricciones más complicadas que los algoritmos exactos. Este es el caso de técnicas tales como el *Recocido Simulado* (*Simulated Annealing*) [17], la *Búsqueda Tabú* (*Tabu Search*) [10, 11] y los *Algoritmos Genéticos* (*Genetic Algorithms*) [14]; donde las funciones objetivo no necesitan cumplir hipótesis de linealidad. Esto nos permite modelar problemas del mundo real aún más precisamente que con el uso de algoritmos exactos.

Muchos problemas de optimización combinatoria son problemas específicos, de manera que un algoritmo de una técnica heurística que funciona para un problema, puede no ser útil para resolver un problema diferente. Sin embargo, hay un gran interés en técnicas que se han desarrollado en la última década y que pueden ser aplicables con mayor generalidad.

Algunas de estas técnicas han sido desarrolladas bajo la búsqueda local de vecinos (*local neighborhood search*). El proceso inicia con una solución para una instancia y busca una mejor solución dentro de una vecindad definida; habiendo encontrado una mejor solución, el proceso reinicia la búsqueda local con esta nueva solución y esto continúa iterativamente hasta que no pueda mejorar la solución actual encontrada. Esta solución final, probablemente sea un óptimo global, aunque con respecto a su vecindad, es un óptimo local.

Una variación a lo anterior que ha ganado reciente atención es el permitir movimientos *ascendentes*, con lo cual la solución con la que la búsqueda local reinicia puede ser peor que la anterior, considerando que debe haber algunas restricciones para aceptar tales movimientos (en otro caso el procedimiento se sumaría a la búsqueda del espacio completo de soluciones). Por lo tanto se da oportunidad de que el proceso pueda escapar o salir de óptimos locales

y busque una mejor solución.

La técnica heurística *Búsqueda Tabú* ha recibido considerable aceptación en los últimos años y ha habido muchas aplicaciones a varios problemas tales como: inventarios multiproducto [6], problema del agente viajero [8], problema de asignación cuadrática [27], problema de coloración de gráficas [13], problema de horarios [12], construcción de horarios para torneos [22]; los cuales han alcanzado considerable éxito. Esta técnica es un ejemplo de los movimientos ascendentes para los óptimos locales.

La idea de simular procesos naturales ha mostrado ser de considerable valor para resolver problemas complejos; el *recocido simulado* fue introducido originalmente como una analogía a los procesos termodinámicos, mientras que la *Búsqueda Tabú* trata de imitar procesos "*inteligentes*", en particular proporcionando a la búsqueda heurística la facilidad de utilizar un tipo de "*memoria*". Más recientemente, los *algoritmos genéticos* formulan los problemas como una analogía a las estructuras genéticas; mientras que las *redes neuronales* [20], tratan de usar una analogía a las propiedades del cerebro humano.

### 1.0.4 Diseño de experimentos

Dado que no existen métodos exactos para resolver el problema de la *construcción de diseños de experimentos con 2 concurrencias* [15], y la búsqueda exhaustiva no es factible debido al gran número de posibles soluciones (que depende de los parámetros del diseño) que habría que analizar, planteamos este problema como un problema de optimización combinatoria y tratamos de resolverlo usando la técnica heurística Búsqueda Tabú.

Este campo de las matemáticas y la estadística está relacionado con la construcción y el diseño de experimentos en los que existen algunas variables que son cambiadas simultáneamente. Originalmente el *diseño de bloques* fue utilizado para diseñar experimentos usando diferentes tipos de plantas en algunas parcelas. Si todas las plantas del mismo tipo fueran cultivadas en la misma parcela, los efectos del tipo de suelo en esa parcela en particular podrían afectar al experimento, por ello las plantas y parcelas deberían estar mezcladas. Los dos parámetros (parcelas y plantas) son llamados *bloques* y *variedades*. Las variedades serán denotadas por $v$ y los bloques por $b$.

Si todas las $v$ variedades son utilizadas en cada bloque, decimos que se trata de un *diseño de bloques completos*. Un diseño es llamado *diseño de bloques incompletos* [2, 24], si el número de variedades que contiene un bloque es menor al número total de variedades. En un diseño, ninguna variedad aparecerá más de una vez en un bloque y cada bloque contendrá el mismo número de variedades denotado por $k$, y cada una de las variedades aparecerá en el mismo número de bloques denotado por $r$. Contando la aparición de las variedades en los bloques de dos formas distintas, obtenemos la siguiente relación: $bk = vr$.

En un *diseño de bloques balanceados*, el número de veces que cualquier par de variedades aparecen juntas es una constante llamada $\lambda$. En un *diseño de bloques incompletos balanceados* (BIBD) [2], se puede mostrar que $r(k-1) = \lambda(v-1)$. Otra propiedad es la *resolubilidad* de los diseños, esto es, que los bloques puedan ser agrupados en $r$ clases paralelas (subconjuntos), donde cada clase paralela contiene $g$ bloques, en los cuales aparecen exactamente las $v$ variedades. Esta propiedad no acepta diseños en los que una variedad no aparece o aparece más de una vez en cualquier clase paralela.

Desafortunadamente, los diseños de bloques incompletos balanceados existen en un número limitado de casos, por lo que fue necesario introducir nuevos diseños. Bose y Nair [3], definieron los diseños de bloques incompletos parcialmente balanceados. Estos diseños tienen como característica principal que $\frac{r(k-1)}{v-1}$ no es un entero, por lo que se pueden obtener parámetros $n_i$ y $\lambda_i$, para $i = 1, \ldots, m$, tal que $\sum_{i=1}^{m} n_i = v - 1$ y $\sum_{i=1}^{m} n_i\lambda_i = r(k-1)$.

### 1.0.5 Objetivos

Las principales metas de esta tesis son:

1. Plantear el problema de la *construcción de diseños de experimentos resolubles con 2 concurrencias*, como un problema de optimización combinatoria.

2. Estudiar la técnica heurística conocida como Búsqueda Tabú y aplicarla a este problema.

Como uno de los resultados de esta tesis se pretende desarrollar un programa que sea altamente eficiente para resolver este problema. Debido a que

no existen métodos generales para construir este tipo de diseños para diferentes parámetros, esta formulación resulta ser importante ya que presenta una nueva forma de resolver el problema.

Se pretende aplicar la técnica de la *Búsqueda Tabú* para resolver varias *instancias* del problema de optimización combinatoria, reportando la experiencia computacional obtenida.

### 1.0.6 Contenido y aportación de la tesis

Se proporciona una aplicación concreta en la que se muestra la bondad e importancia de la técnica. Las aportaciones de esta tesis se pueden resumir en lo siguiente:

1. Una revisión ordenada de la técnica heurística: Búsqueda Tabú.
2. Desarrollo de dos programas de la Búsqueda Tabú para el problema de la construcción de diseños resolubles con 2 concurrencias, uno secuencial y otro en paralelo.
3. Se reporta la experiencia computacional sobre la eficiencia de ambos programas.

En el capítulo 1 se da la descripción de los problemas de optimización combinatoria, de las técnicas heurísticas y de los diseños de experimentos. En el capítulo 2 se describen los diseños de experimentos con $m$ concurrencias y se plantean como un problema de optimización combinatoria. En el capítulo 3 se introduce la técnica de la Búsqueda Tabú y se describen los elementos principales mediante el desarrollo de un ejemplo de diseño de experimentos. En el capítulo 4 se describe cómo se desarrollan los programas tanto secuencial como en paralelo para la Búsqueda Tabú. En el capítulo 5 se muestran las tablas de resultados obtenidas al aplicar ambos programas a una serie de instancias del problema de optimización combinatoria. En el capítulo 6 se presentan las conclusiones, así como posibles líneas de investigación. En los apéndices A y B se presentan los listados completos de los programas. Finalmente, en el apéndice C se muestran los resultados de 48 de los 76 diseños obtenidos, de los cuales según la literatura revisada se desconocía su existencia y no aparecen en las tablas de Clatworthy [5].

# Capítulo 2

# Diseños con m Concurrencias

## 2.1 Introducción

Los *diseños de bloques incompletos* se han desarrollado como herramientas estadísticas para la planeación, dirección, análisis e interpretación de experimentos científicos dirigidos en laboratorios, donde un alto grado del control experimental es deseado. Históricamente el uso estadístico de estos diseños inicia con los *diseños de bloques incompletos balanceados* [2, 24], introducidos por Yates [30], los cuales fueron usados en experimentos agrícolas y biológicos.

Bose y Nair [3], desarrollaron una clase muy general de diseños de bloques incompletos, a los que llamaron *diseños de bloques incompletos parcialmente balanceados con m clases asociadas* (PBIBD/m) [24]. Estos diseños incluyen a los diseños de bloques incompletos balanceados (BIBD) como un caso especial; generalmente ellos tienen la ventaja de proveer una gran variedad de arreglos experimentales que los investigadores pueden elegir según sus necesidades. Tablas de estos diseños están disponibles en [5, 18].

Por 1952 *Bose* y un grupo de sus estudiantes, habían desarrollado los *diseños de bloques incompletos parcialmente balanceados con 2 clases asociadas* (PBIBD/2), cuya clasificación está basada en el concepto de *esquemas de asociación* que fue desarrollado por Bose y Shimamoto [4]; desde entonces el concepto de esquemas de asociación ha sido tema de intensa investigación

por investigadores en estadística interesados en problemas combinatorios del diseño de experimentos.

Sin embargo, hay muchas combinaciones de parámetros para los cuales los diseños de bloques incompletos parcialmente balanceados con 2 clases asociadas no existen. Para resolver esto, Jarrett [15] definió los diseños con $m$ concurrencias, los cuales son una relajación de los PBIBD/m. Son de gran interés los diseños con 2 concurrencias; en particular si las concurrencias difieren solamente por una unidad. Es decir, $\lambda_2 = \lambda_1 + 1$. Tales diseños son conocidos como *diseños de gráficas regulares*.

## 2.2 Definiciones

Para definir los diseños con $m$ concurrencias, introducimos parámetros análogos a aquellos usados por los diseños de bloques incompletos parcialmente balanceados con $m$ clases asociadas (PBIBD/m) (ver por ejemplo [24] p. 121).

**Definición 2.2.1.** Un diseño de bloques incompletos con $v$ variedades es un ordenamiento de las variedades $1, \ldots, v$ en $b$ bloques de tamaño $k$, tal que cada variedad ocurre exactamente $r$ veces.

Si ninguna variedad ocurre más de una sola vez en cualquier bloque, el diseño se denomina un diseño binario. Nos referiremos a las variedades $\alpha$ y $\beta$ como $i$-ésima variedades asociadas si ellas aparecen juntas en un bloque $\lambda_i$ veces para $i = 1, \ldots, m$ donde las $\lambda_i$'s son distintas. Entonces tenemos:

**Definición 2.2.2.** Dadas $v$ variedades $1, \ldots, v$, definimos un diseño con $m$ concurrencias como un diseño binario de bloques incompletos si satisface las siguientes condiciones:

1. Cualesquiera dos variedades son $1^a, 2^a, \ldots, m$-ésima variedades asociadas, siendo la relación simétrica, y
2. Cada variedad $\alpha$ tiene $n_i$ $i$-ésima variedades asociadas, el número $n_i$ es independiente de $\alpha$.

El siguiente paso es considerar los parámetros $p^i_{jk}$ de asociación. Para los diseños con $m$ concurrencias, esto puede ser definido de la siguiente forma:

**Definición 2.2.3.** Si cualesquiera dos variedades $\alpha$ y $\beta$ tienen concurrencia $\lambda_i$, entonces el número de variedades comunes que tienen concurrencia $\lambda_j$ con $\alpha$ y $\lambda_k$ con $\beta$ es denotado por $p^i_{jk}(\alpha,\beta)$. El promedio de $p^i_{jk}(\alpha,\beta)$ sobre todos los pares $(\alpha,\beta)$ que tienen concurrencia $\lambda_i$, es denotado por $p^i_{jk}$.

Notamos que un diseño PBIB/m es un diseño con $m$ concurrencias con parámetros de asociación $p^i_{jk}$.

Shrikhande y Raghavarao [26], generalizaron el concepto de *resolubilidad* de Bose [1] a *$\alpha$-resolubilidad* como sigue:

**Definición 2.2.4.** Un diseño de bloques incompletos con los parámetros $(v,b,r,k)$ se dice que es *$\alpha$-resoluble* si los bloques pueden ser agrupados en $t$ clases paralelas (subconjuntos) $S_1$, $S_2$, ..., $S_t$, cada una con $\beta$ grupos, tal que en cada clase paralela cada variedad es repetida $\alpha$ veces. Entonces tenemos que:

$$v\alpha = k\beta, \qquad b = t\beta, \qquad r = t\alpha.$$

Un diseño de bloques incompletos 1-resoluble puede ser simplemente llamado *resoluble*, esto es, si los $b$ bloques son agrupados en $r$ clases paralelas (subconjuntos) de $g$ grupos cada uno, tal que cada clase paralela forme una réplica completa de todas las $v$ variedades. Una condición necesaria para la existencia de un diseño resoluble es que $k$ divida a $v$ y sea el mismo entero $r$ que divide a $b$: $\frac{v}{k} = \frac{b}{r}$. Entonces un diseño resoluble tiene:

$$v = kg, \qquad b = rg. \tag{2.1}$$

## 2.3 Relación entre los parámetros del diseño

Contando la aparición de las variedades en los bloques en dos formas diferentes, tenemos que hay $v$ variedades, cada una de las cuales aparece en $r$ bloques, y hay $b$ bloques, cada uno de los cuales contiene $k$ variedades:

$$vr = bk. \tag{2.2}$$

Cada variedad aparece en $r$ bloques y en cada uno de ellos aparece con $k-1$ variedades. Pero cada variedad debe aparecer con cada una de las restantes $v-1$ variedades exactamente $\lambda$ veces.

$$\lambda(v-1) = r(k-1). \tag{2.3}$$

En los diseños balanceados, la ecuación (2.3) se satisface si $\lambda$ es un entero, pero para los diseños parcialmente balanceados, $\lambda$ no es un entero; pero existen parámetros $n_i$ y $\lambda_i$, para $i = 1, \ldots, m$ que satisfacen las ecuaciones (2.4) y (2.5).

Con respecto a cada variedad $\alpha$, ya que cada una de las $v-1$ variedades restantes es clasificada como $1^a, 2^a, \ldots, m$-ésima variedad asociada y como cada variedad $\alpha$ tiene $n_i$ $i$-ésima variedades asociadas, entonces:

$$\sum_{i=1}^{m} n_i = v - 1, \tag{2.4}$$

$$n_1 + n_2 = v - 1.$$

Consideremos los $r$ bloques en los que una variedad particular $\alpha$ aparece. De estos bloques podemos formar $r(k-1)$ parejas de variedades, manteniendo a $\alpha$ como una de las variedades. Esas parejas de $i$-ésima variedades asociadas de $\alpha$ deben aparecer $\lambda_i$ veces, y como hay $n_i$ $i$-ésima variedades asociadas de $\alpha$. Entonces:

$$\sum_{i=1}^{m} n_i \lambda_i = r(k-1), \tag{2.5}$$

$$n_1\lambda_1 + n_2\lambda_2 = r(k-1).$$

Se puede observar que los parámetros $p_{jk}^i$ satisfacen las mismas ecuaciones que los diseños PBIB/m. Para una comparación con las propiedades de estos diseños, ver ([24] pp. 121-123). Si $\alpha$ y $\beta$ tienen concurrencia $\lambda_i$, entonces el número promedio de variedades comunes que tienen concurrencia $\lambda_k$ con $\alpha$ para $k = 1, 2, \ldots, m$, debería cubrir las $n_j$ variedades que tienen concurrencia $\lambda_j$ con $\beta$ para $j \neq i$. Cuando $i = j$, $\alpha$ será una de las variedades con concurrencia $\lambda_j$ con $\beta$. Entonces el número promedio de variedades comunes con concurrencia $\lambda_k$ con $\alpha$ para $k = 1, 2, \ldots, m$, debería cubrir las $n_j - 1$ variedades que tienen concurrencia $\lambda_j$ con $\beta$.

$$\sum_{k=1}^{m} p_{jk}^i = n_j - \delta_{ij}, \tag{2.6}$$

$$p_{11}^1 + p_{12}^1 = n_1 - 1, \qquad p_{21}^1 + p_{22}^1 = n_2,$$

$$p_{11}^2 + p_{12}^2 = n_1, \qquad p_{21}^2 + p_{22}^2 = n_2 - 1,$$

$$p_{jk}^i = p_{kj}^i, \tag{2.7}$$

$$p_{12}^1 = p_{21}^1, \qquad p_{12}^2 = p_{21}^2,$$

donde $\delta_{ij}$ es la *delta de Kronecker*, tomando el valor de 1 si $i = j$ y 0 en otro caso.

Sea $\alpha$ cualquier variedad, y sea $G_i$ el conjunto de variedades que tienen concurrencia $\lambda_i$ con $\alpha$ y $G_j$ el conjunto de variedades que tienen concurrencia $\lambda_j$ con $\alpha$. Cualquier variedad en $G_i$ tiene $p_{jk}^i$ variedades comunes con concurrencia $\lambda_k$ en $G_j$, y cualquier variedad en $G_j$ tiene $p_{ik}^j$ variedades comunes con concurrencia $\lambda_k$ en $G_i$. Si formamos parejas de variedades con concurrencia $\lambda_k$ de $G_i$ y $G_j$, entonces:

$$n_i p_{jk}^i = n_j p_{ik}^j, \tag{2.8}$$

$$n_1 p_{21}^1 = n_2 p_{11}^2, \qquad n_1 p_{22}^1 = n_2 p_{12}^2.$$

En los diseños con 2 concurrencias, las relaciones anteriores implican que con el conocimiento de cualquiera de los valores $p_{jk}^i$ es suficiente para poder determinar los demás parámetros.

## 2.4 Problema de optimización combinatoria

Para plantear el problema de la construcción de diseños de experimentos resolubles con 2 concurrencias como un problema de optimización combinatoria, vamos a tomar únicamente su definición sin tocar la parte estadística de éstos.

### 2.4.1 Planteamiento del problema

Un *diseño resoluble con m concurrencias*, se puede ver como el ordenamiento de $v = kg$ variedades en $b = rg$ bloques de tamaño $k$, tal que $k < v$ y formalmente se define de la siguiente manera:

1. Cada variedad aparece a lo más en un solo bloque.
2. Cada variedad aparece exactamente en $r$ bloques.
3. Los $b$ bloques se agrupan en $r$ clases paralelas de $g$ grupos cada uno.
4. Si dos variedades $\alpha$ y $\beta$ son $i$-ésima variedades asociadas, entonces ellas aparecen juntas en $\lambda_i$ bloques.

5. Cada variedad $\alpha$ tiene $n_i$ $i$-ésima variedades asociadas, el número $n_i$ es independiente de $\alpha$.

Tomando como base estas definiciones, nos enfocamos únicamente en los *diseños resolubles con 2 concurrencias*.

Podemos ver al problema de la construcción de diseños resolubles con 2 concurrencias como un problema de asignación de variedades a bloques. Para un diseño resoluble con 2 concurrencias definimos una matriz denominada matriz de concurrencia $C_{ij}$ de tamaño $v \times v$, la cual nos representa el número de veces que la variedad $i$ aparece junto a la variedad $j$ en ese diseño y que utilizaremos para medir su optimalidad. La matriz $C_{ij}$ es simétrica, es decir $C_{ij} = C_{ji}$, además $C_{ij} = 0$ para $i = j$ (diagonal principal), ya que una variedad no puede aparecer más de una vez en un solo bloque, es decir, una variedad no puede aparecer junto a ella misma. Como la matriz es simétrica, únicamente necesitamos revisar $\frac{v^2-v}{2}$ elementos por encima de la diagonal principal. Otra característica de la matriz de concurrencia $C_{ij}$ es que la suma de todos sus elementos es una constante sin considerar el diseño utilizado: $v(n_1\lambda_1 + n_2\lambda_2) = vr(k-1)$; debido a que esta suma representa dos veces el número que las variedades aparecen juntas, se divide entre dos: $\frac{v}{2}(n_1\lambda_1 + n_2\lambda_2)$.

Si una solución óptima para un *diseño resoluble con 2 concurrencias* existe, cada renglón de $C_{ij}$ debe consistir de $n_1$ veces $\lambda_1$ y de $n_2$ veces $\lambda_2$, con 0's en la diagonal principal. Entonces por encima de la diagonal principal, debe haber $\frac{vn_1}{2}$ veces $\lambda_1$ y $\frac{vn_2}{2}$ veces $\lambda_2$.

Los parámetros $g$, $n_1$, $n_2$, $\lambda_1$ y $\lambda_2$ se obtienen de la siguiente manera:

- A $g$ le asignamos la parte entera de $\frac{v}{k}$.
- A $n_2$ le asignamos el resto de la división $\frac{r(k-1)}{v-1}$.
- A $n_1$ le asignamos la diferencia de $(v-1)$ y $n_2$.
- A $\lambda_1$ le asignamos la parte entera de la división $\frac{r(k-1)}{v-1}$.
- A $\lambda_2$ le asignamos el valor de $\lambda_1$ más una unidad.

Ahora, el problema de la construcción de diseños resolubles con 2 concurrencias puede ser formulado como el problema de buscar un conjunto

ordenado de variedades en bloques (diseño) cuya matriz de concurrencia $C_{ij}$ minimice la siguiente función objetivo:

$$f = \sum_{i<j}^{v} (C_{ij} - \lambda)^2. \tag{2.9}$$

El costo de un diseño en particular es calculado sumando los cuadrados de la diferencia entre el número promedio de veces que dos variedades deberían aparecer juntas, llamado $\lambda$, y cada una de las entradas de la matriz de concurrencia por encima de la diagonal principal. De esta forma, el costo de un diseño decrece cuando la matriz $C_{ij}$ tiene más elementos que están cercanos al valor de $\lambda$.

Los diseños resolubles con 2 concurrencias cumplen la propiedad de que $\frac{r(k-1)}{v-1}$ no es un entero, pero existen enteros $\lambda_1$, $\lambda_2$, $n_1$ y $n_2$ que satisfacen las ecuaciones (2.4) y (2.5). Teniendo estos parámetros podemos establecer una cota mínima para nuestra función objetivo de la siguiente manera: Si $n_i = \min(n_1, n_2)$, entonces $\lambda = \lambda_j$ para $j \neq i$, donde $1 \leq i, j \leq 2$, entonces el mínimo valor posible de la función objetivo es: $f^* = \frac{vn_i}{2}$.

## 2.4.2 Búsqueda exhaustiva

La construcción de un diseño de experimentos óptimo no es una tarea trivial, debido a que hay un gran número de diseños a ser considerados. Aunque hay procedimientos para la construcción de los diseños de experimentos, esos procedimientos usualmente están restringidos a clases muy específicas de problemas (ver por ejemplo la clasificación de PBIBD/2 en [24] p. 127).

Antes de continuar, vamos a dar las siguientes definiciones:

**Combinaciones.** Cuando se dispone de $n$ elementos, hay $(n)_m = n(n-1)\cdots(n-m+1)$ posibilidades distintas de seleccionar muestras de tamaño $m$ sin reemplazo. Cada una de las muestras (que es una combinación de $m$ elementos diferentes) tiene $m!$ arreglos diferentes. Entonces $\frac{(n)_m}{m!}$ da el número de posibilidades para seleccionar una muestra de tamaño $m$ de $n$ elementos sin reemplazo cuando no se tiene en cuenta el orden dentro de cada muestra. Esta operación se representa simbólicamente por $C(n,m) = \binom{n}{m} = \frac{(n)_m}{m!}$, y también se le conoce como el *coeficiente binomial*.

| | | | | |
|---|---|---|---|---|
| $A_{11}$ | $A_{12}$ | $A_{13}$ | $\cdots$ | $A_{1g}$ |
| $A_{21}$ | $A_{22}$ | $A_{23}$ | $\cdots$ | $A_{2g}$ |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ |
| $A_{r1}$ | $A_{r2}$ | $A_{r3}$ | $\cdots$ | $A_{rg}$ |

Figura 2.1: $r$ clases paralelas con $g$ grupos y cada grupo con $k$ variedades

**Coeficiente Multinomial**: Dado un conjunto de $v$ elementos agrupados en $g$ grupos tales que $k_1 + k_2 + \cdots + k_g = v$, siendo $k_1$, $k_2$, ..., $k_g$ el número de elementos en cada grupo. Entonces hay $\frac{v!}{k_1!k_2!\cdots k_g!}$ formas diferentes de agrupar los $v$ elementos en estos $g$ grupos.

Para enumerar todos los posibles diseños, etiquetamos a los $g$ grupos en cada clase paralela como se muestra en la figura 2.1, donde cada grupo contiene $k$ variedades. El número de todas las posibles asignaciones de $v = kg$ variedades a esos $g$ grupos está dado por el coeficiente multinomial $\frac{v!}{k_1!k_2!\cdots k_g!}$, donde $k_i = k$. Esto representa el número de formas de elegir $k$ variedades para los $g$ grupos. Pero como las etiquetas en los grupos son muy arbitrarias, cada uno podría ser etiquetado en $g!$ formas diferentes, entonces el número de distintas posibilidades para cada clase paralela es el coeficiente multinomial dividido por $g!$. Como hay $r$ clases paralelas, el número de todos los posibles diseños está dado por las combinaciones de

$$C(\frac{v!}{k_1!\cdots k_g!g!}, r). \tag{2.10}$$

En el siguiente capítulo veremos como funciona la técnica de la Búsqueda Tabú a través del desarrollo de un ejemplo de diseño de experimentos.

# Capítulo 3

# Búsqueda Tabú

## 3.1 Introducción

La Búsqueda Tabú (BT) tiene sus antecedentes en métodos diseñados para cruzar límites de optimalidad local, normalmente tratados como barreras y manejados sistemáticamente para imponer y quitar restricciones que permitan la exploración de regiones de otra manera prohibidas. La forma moderna de la BT fue introducida por Glover [10, 11].

El diccionario *Webster* define *tabu* o *taboo* como: "... cargado con un peligroso poder sobrenatural y prohibido, para profanar el uso o contacto ..." o "... prohibido por la moral ...". La BT ciertamente no involucra referencias a consideraciones morales o sobrenaturales, pero en cambio está interesada en imponer restricciones que puedan guiar al proceso de búsqueda para negociar en otro caso regiones difíciles. Estas restricciones operan de varias formas; un ejemplo es por exclusión directa de ciertos movimientos clasificados como prohibidos.

La filosofía de la BT se basa en manejar y explotar una colección de principios inteligentes para la solución de problemas. Un elemento fundamental de la BT es el uso de la *memoria flexible*, la cual engloba los procesos de creación y explotación de estructuras computacionales para tomar ventaja de la historia de los movimientos realizados (esto es, la combinación de las actividades de adquisición y mejoramiento de información).

La BT se basa en tres características principales:

1. El uso de estructuras computacionales de memoria basadas en los atributos de un movimiento, nos permiten aplicar criterios de evaluación y obtener información histórica, para mejorar el proceso de búsqueda; lo cual no sucede para estructuras con pérdida de memoria (como algunos métodos aleatorios volátiles), es decir en los que no se recuerda.

2. Un mecanismo asociado de control, mediante el empleo de estructuras computacionales de memoria, basado entre las condiciones que restringen y liberan al proceso de búsqueda (restricciones tabú y criterio de aspiración).

3. La incorporación de funciones de memoria de distintos lapsos de tiempo o ciclos de vida, desde memoria corta hasta memoria larga, para implantar estrategias que refuercen la combinación de movimientos y las características de solución que históricamente se han encontrado buenas, mientras que las estrategias de diversificación manejan la búsqueda dentro de nuevas regiones.

La estructura de la memoria de la BT opera por referencia a 4 dimensiones principales:

1. Los movimientos más recientes (*recency*)

2. Frecuencia (*frequency*)

3. Calidad (*quality*)

4. Influencia (*influence*)

## 3.2 Marco teórico de la Búsqueda Tabú

### 3.2.1 Descripción

Para describir el funcionamiento de la técnica de BT, representamos el problema de optimización combinatoria de la siguiente manera:

$$\text{Minimizar } c(x)$$

$$\text{sujeta a } x \in X.$$

La función objetivo $c(x)$ puede ser lineal o no lineal, y la condición $x \in X$ supone que las restricciones especificadas de los componentes de $x$ sean valores discretos.

### 3.2.2 Búsqueda local de vecinos

La BT puede ser caracterizada como una forma de búsqueda local de vecinos como se describió brevemente en el primer capítulo. En la búsqueda local de vecinos cada solución $x \in X$ tiene asociado un conjunto de vecinos $V(x) \subset X$, llamado la vecindad de $x$. Cada solución $x' \in V(x)$, puede alcanzarse directamente desde la solución $x$ aplicando una operación llamada *movimiento*, se dice que $x$ cambia a $x'$ cuando tal operación se realiza. Normalmente en la BT, las vecindades son simétricas, es decir, $x'$ es un vecino de $x$ si y sólo si $x$ es un vecino de $x'$.

**Método de búsqueda local de vecinos**

1. Inicialización.
   - Seleccionar una solución inicial $x^{act} \in X$.
   - Registrar la mejor solución $x^{mej} = x^{act}$ y $mejor\_costo = c(x^{mej})$.
2. Selección y Terminación.
   - Escoger una solución $x^{sig} \in V(x^{act})$. Si el criterio de cambio empleado no puede ser satisfecho por ningún miembro de $V(x^{act})$ (ninguna solución califica a ser $x^{sig}$), o si otro criterio de terminación es aplicado (tal como el número de iteraciones), entonces el método se detiene.
3. Actualización.
   - Hacer $x^{act} = x^{sig}$, y si $c(x^{act}) < mejor\_costo$, realizar el segundo punto del primer paso. Regresar al segundo paso.

El método de la búsqueda local de vecinos puede ser alterado fácilmente para que nos dé algunos procedimientos clásicos. Los métodos descendentes, sólo permiten movimientos a soluciones vecinas que mejoren el valor actual de $c(x^{act})$, y terminan cuando la solución actual no puede ser mejorada.

**Método Descendente**

1. Inicialización.
   - Empezar con la inicialización de la búsqueda local de vecinos.

2. Selección y Terminación.

   - Escoger $x^{sig} \in V(x^{act})$ para satisfacer $c(x^{sig}) < c(x^{act})$ y termina si $x^{sig}$ no puede ser encontrado.

3. Actualización.

   - Realizar la actualización de la búsqueda local de vecinos.

El defecto evidente del método descendente es que al final la $x^{act}$ obtenida es un óptimo local, que en muchos de los casos no será un óptimo global.

Los procedimientos aleatorios tales como el método *Monte Carlo*, pueden ser representados de la siguiente manera:

**Método Monte Carlo**

1. Inicialización.

   - Empezar con la inicialización de la búsqueda local de vecinos.

2. Selección y Terminación.

   - Aleatoriamente seleccionar $x^{sig} \in V(x^{act})$.
   - Si $c(x^{sig}) \leq c(x^{act})$ aceptar $x^{sig}$ y proceder con el tercer paso.
   - [1]Si $c(x^{sig}) > c(x^{act})$ aceptar $x^{sig}$ con una probabilidad que decrece con incrementos de la diferencia $c(x^{sig}) - c(x^{act})$. Si $x^{sig}$ no es aceptado en la presente prueba por este criterio, regresar al primer punto de este paso. Si es aceptado proceder con el tercer paso.
   - Terminar por el número de iteraciones.

3. Actualización.

   - Realizar la actualización de la búsqueda local de vecinos.

[1]Este punto no pertenece al método Monte Carlo, pero es introducido porque el método recocido simulado utiliza la versión del método Monte Carlo agregando esta condición.

La versión del método *Monte Carlo* representada por el *recocido simulado* inicia con una probabilidad alta para aceptar aquellos movimientos que no mejoran en el tercer punto del segundo paso, el cual decrece en el tiempo como una función de un parámetro llamado *temperatura* disminuyendo a cero conforme aumenta el número de iteraciones. Tales métodos ofrecen una oportunidad para mejorar la búsqueda de un simple óptimo local, ya que ellos terminan únicamente cuando la probabilidad de aceptación de un movimiento que no mejora es tan pequeña que ningún movimiento es aceptado. Normalmente estos métodos utilizan la función exponencial para la definición de probabilidades. Otro método aleatorio para vencer las limitaciones de los métodos descendentes es simplemente reiniciar los métodos con distintas soluciones iniciales generadas aleatoriamente y correrlos en múltiples ocasiones.

### 3.2.3 Características de la Búsqueda Tabú

La BT, en contraste con los métodos mencionados, emplea una filosofía algo diferente para ir más allá del criterio de quedar atrapado en un óptimo local. La explotación de ciertas formas de memoria flexible para controlar el proceso de búsqueda es el tema fundamental de la BT. El efecto de tal memoria se obtiene estableciendo que la BT mantenga una historia selectiva $H$ de los estados encontrados durante la búsqueda y reemplazando $V(x^{act})$ por una vecindad modificada la cual puede ser denotada por $V(H, x^{act})$. La historia determina por lo tanto, cuales soluciones pueden ser alcanzadas por un movimiento a partir de la solución actual, seleccionando $x^{sig} \in V(H, x^{act})$.

En las estrategias de la memoria corta, la vecindad $V(H, x^{act})$ es típicamente un subconjunto de $V(x^{act})$ y la clasificación tabú sirve para identificar elementos de la vecindad $V(x^{act})$ excluídos de $V(H, x^{act})$. En las estrategias de la memoria intermedia y memoria larga, la vecindad $V(H, x^{act})$ puede contener soluciones que no están en $V(x^{act})$, generalmente consiste en escoger soluciones élite (óptimos locales de alta calidad) encontradas en diferentes puntos del proceso de búsqueda. Tales soluciones élite son identificadas como elementos de un sector regional en estrategias de intensificación en memoria intermedia y como elementos de sectores diferentes en estrategias de diversificación en memoria larga. Además, los componentes de las soluciones élite, en contraste a las soluciones mismas, están incluídos entre los elementos que pueden ser retenidos e integrados para proveer entradas al proceso de búsqueda.

La BT también utiliza la historia para crear una evaluación modificada de la solución accesible actual. Formalmente, esto puede ser expresado diciendo que la BT reemplaza la función objetivo $c(x)$ por una función $c(H, x)$, la cual tiene el propósito de evaluar la calidad relativa de la solución accesible actual. La relevancia de esta función modificada ocurre debido al uso de un criterio de elección agresivo para localizar un mejor $x^{sig}$, es decir, el mejor valor de $c(H, x^{sig})$, sobre el conjunto de candidatos derivados de $V(H, x^{act})$. La referencia a $c(x)$ es retenida para determinar si un movimiento está mejorando o nos conduce a una nueva mejor solución.

Para problemas grandes, donde $V(H, x^{act})$ puede tener muchos elementos, o para problemas donde es demasiado costoso examinar esos elementos, la BT hace muy importante el poder aislar un subconjunto de candidatos de la vecindad y examinarlo en vez de examinar la vecindad completa. Esto puede ser hecho en etapas, permitiendo al subconjunto candidato ser ampliado si las alternativas de satisfacción de los niveles de aspiración no son encontrados. Nos referimos a este subconjunto explícitamente por la notación $Candidato\_N(x^{act})$. Entonces el procedimiento de la BT puede ser expresado de la siguiente manera:

**Método Básico de la Búsqueda Tabú**

1. Inicialización.

   - Empezar con la inicialización de la búsqueda local de vecinos, y con el registro de la historia $H$ vacía.

2. Selección y Terminación.

   - Determinar el subconjunto $Candidato\_N(x^{act})$ como un subconjunto de $V(H, x^{act})$. Seleccionar $x^{sig} \in Candidato\_N(x^{act})$ para minimizar $c(H, x)$ sobre este subconjunto. ($x^{sig}$ es llamado el mejor elemento del subconjunto $Candidato\_N(x^{act})$). Terminar por el número de iteraciones si no se encuentra alguna solución óptima.

3. Actualización.

   - Realizar la actualización de la búsqueda local de vecinos, y adicionalmente actualizar el registro de la historia $H$.

La esencia del método depende de cómo el registro de la historia $H$ es definido y usado, y cómo la vecindad $Candidato\_N(x^{act})$ y la evaluación de la función $c(H,x)$ son determinadas. El método de la BT utilizado introduciendo el criterio de aspiración y la memoria larga es el siguiente:

**Método Mejorado de la Búsqueda Tabú**

1. Inicialización.

   - Empezar con la inicialización de la búsqueda local de vecinos, con el registro de la historia $H$ vacía y la matriz de frecuencia inicializada con ceros.

2. Selección y Terminación.

   - Determinar el subconjunto $Candidato\_N(x^{act})$ como un subconjunto de $V(x^{act})$. Inicializar $minimo$ y $minimop$ con un valor alto. Generar todos los vecinos $x^{sig} \in Candidato\_N(x^{act})$, cuyo costo $c(x^{sig}) \leq minimo$. Aplicar el criterio de aspiración si $c(x^{sig}) < mejor\_costo$, aceptar de inmediato a $x^{sig}$ y $minimop = minimo = x^{sig}$. Si no penalizar a $x^{sig}$ con la matriz de frecuencia, $pena = x^{sig} + frecuencia(i,j)$. Verificar que el movimiento no sea tabú y aceptarlo si $pena \leq minimop$. Si fue aceptado, entonces $minimop = pena$ y $minimo = x^{sig}$. Terminar si encuentra una solución óptima o por un número límite de iteraciones.

3. Actualización.

   - Realizar la actualización de la búsqueda local de vecinos, y adicionalmente actualizar el registro de la historia $H$, así como también la matriz de frecuencia.

### 3.2.4 Memoria de la Búsqueda Tabú

Un atributo de un movimiento de $x^{act}$ a $x^{sig}$, o más precisamente de un movimiento de prueba $x^{act}$ a una solución tentativa $x^{prue}$, puede abarcar cualquier aspecto que cambie como un resultado del movimiento. Por ejemplo, un movimiento que cambia el valor de dos variables simultáneamente.

El registro de los atributos de un movimiento son en ocasiones utilizados en la BT para imponer restricciones, llamadas *restricciones tabú*, que evitan

elegir movimientos que inviertan los cambios representados por esos atributos. Por ejemplo, un movimiento es tabú si: $x_j$ cambia de 1 a 0, donde $x_j$ previamente cambió de 0 a 1.

Las restricciones tabú también son utilizadas para evitar movimientos que conduzcan a repeticiones o ciclos en un camino de búsqueda. Estas restricciones tienen el papel de evitar el regreso a un movimiento cuyos atributos produzcan una solución previamente encontrada. Por lo tanto las restricciones tabú varían de acuerdo a como son definidas.

Una restricción tabú es activada únicamente en el caso en que sus atributos ocurran dentro de cierto número de iteraciones anterior a la iteración actual (creando la lista tabú de los movimientos más recientes), o que ocurran con cierta frecuencia a lo largo de las iteraciones (creando la matriz de frecuencia). Más precisamente, una restricción tabú es ejecutada únicamente cuando los atributos de su definición rebasen ciertos umbrales de frecuencia o sean iguales a los movimientos más recientes. Para entender esto, se define un atributo como *tabú-activo* cuando su atributo inverso asociado ha ocurrido en cierto intervalo de los movimientos más recientes. Un atributo que no es activo es llamado *tabú-inactivo*.

La condición de ser tabú-activo o tabú-inactivo es llamado el *estado tabú* de un atributo. En algunas ocasiones un atributo es llamado *tabú* o *no tabú* para indicar que es tabú-activo o tabú-inactivo.

Debemos señalar, sin embargo, que evitar el ciclado no es el último objetivo del proceso de BT. En algunos casos, un buen camino de búsqueda resultará en revisitar los atributos de un movimiento que nos conduzcan a una mejor solución que cualquiera encontrada anteriormente. El objetivo es continuar estimulando el descubrimiento de buenos movimientos que conduzcan a nuevas soluciones de alta calidad.

### 3.2.5 Criterios de aspiración

Los criterios de aspiración se introducen en la técnica de BT para poder determinar cuándo una restricción tabú puede ser rechazada, eliminando la clasificación tabú aplicada a cierto movimiento. El uso apropiado de este criterio de aspiración puede resultar muy importante para permitir así un buen nivel de desempeño de la BT.

Las primeras aplicaciones de la BT emplearon únicamente un tipo simple del criterio de aspiración, el cual consistía en rechazar la clasificación tabú de un movimiento de prueba cuando al evaluar la función objetivo se obtiene una mejor solución que la mejor solución encontrada hasta el momento. Este criterio de aspiración sigue siendo utilizado en muchas aplicaciones; sin embargo, otros criterios pueden mejorar también la efectividad de la BT.

Uno de esos criterios de aspiración surge introduciendo el concepto de *influencia*, el cual mide el grado de cambio inducido en la estructura de la solución (la influencia es en ocasiones asociado con la idea de la distancia del movimiento, es decir, un movimiento de mayor distancia es concebido como el de mayor influencia).

Los movimientos de gran influencia son importantes especialmente durante los intervalos para romper con la optimalidad local, debido a que una serie de movimientos que hacen solamente pequeños cambios estructurales es improbable que descubran un mejoramiento importante.

Las aspiraciones son de dos tipos: *movimientos de aspiración* y *atributos de aspiración*. Un movimiento de aspiración, cuando se satisface, rechaza la clasificación tabú del movimiento. Un atributo de aspiración, cuando se satisface, rechaza el estado tabú del atributo. En el último caso, el movimiento puede o no cambiar su clasificación tabú, dependiendo si la restricción tabú puede ser activada por más de un atributo.

## 3.3 Aplicación de la Búsqueda Tabú

### 3.3.1 Ejemplo: construcción de diseño de experimentos

Los problemas de permutaciones forman una clase muy importante de los problemas de optimización combinatoria y ofrecen un medio útil para demostrar algunas de las consideraciones que deben enfrentarse en el dominio combinatorio. Instancias clásicas de problemas de permutación incluyen al problema del agente viajero y al problema de asignación cuadrática, entre otros.

Como ejemplo para ilustrar la técnica de BT, consideramos el problema de horarios que fue presentado por un club de bridge a Elenbogen y Maxim [7]. Ellos formularon el problema como un problema de optimización combinatoria, utilizando las siguientes técnicas de optimización discreta para resolverlo: el algoritmo *greedy*, *branch and bound*, *steepest descent* y *simulated annealing*. Desafortunadamente, ninguna de estas técnicas obtuvo el horario requerido por el club. Ellos vieron a esta instancia como un *diseño resoluble de bloques incompletos parcialmente balanceados* [24]; pero el diseño de bloques requerido no está disponible en los catálogos de diseños de experimentos.

Este mismo problema, fue resuelto utilizando la BT en [21]. Además, utilizando esta misma técnica se ha estudiado el problema de una manera general y se han resuelto varias instancias de este problema de optimización combinatoria en [22].

**Problema del Club de Bridge**. El club de bridge consta de 12 parejas. En cada una de las 8 reuniones al año, el club se divide en 3 grupos de 4 parejas. Cada una de las 4 parejas compite contra las 3 restantes en su grupo, lo cual requiere de 6 juegos por grupo, 18 por cada reunión, y un total de 144 juegos. El club requiere un horario por año con las siguientes propiedades: el número de veces que cualquier pareja compita contra cualquier otra debe ser el mismo para todas las parejas. Si no existe tal horario, entonces se requiere uno donde el número de veces que cualquier pareja juegue contra cualquier otra debe ser lo más parecido posible para todas las parejas.

**Problema General**. Un club deportivo tiene $r$ reuniones por año. En cada reunión, los $t$ equipos del club están divididos en $g$ grupos de $k$ equipos ($t = kg$). Cada uno de los $k$ equipos juega contra los restantes $k - 1$ en su grupo. El club necesita un horario por año tal que el número de veces que cualquier equipo compita contra cualquier otro debe ser tan balanceado como sea posible para todos los equipos.

Existen $(k-1)!$ juegos en cada grupo y $g(k-1)!$ juegos por reunión. Entonces el número total de juegos por año es $rg(k-1)!$. Como cada equipo juega $r$ veces contra $k-1$ equipos, hay $r(k-1)$ equipos opuestos. Entonces cada equipo tiene $t-1$ posibles oponentes. Se desea un horario con la propiedad de que el número de veces que cualquier equipo juegue contra

cualquier otro sea muy cercano a $\lambda = \frac{r(k-1)}{t-1}$.

En general, podemos observar que existe una correspondencia entre este problema de horarios y la construcción de un diseño resoluble con 2 concurrencias como se planteó en la sección 2.4.1.

Entonces el problema de la construcción de diseños de experimentos es formulado como el problema de buscar la asignación de variedades a bloques, cuya matriz de concurrencia $C_{ij}$ minimice la función de costo dada por la ecuación (2.9).

En particular, para la instancia de Elenbogen y Maxim: $v = 12, b = 24$, $r = 8, g = 3$ y $k = 4$ el valor de $\lambda$ no es un entero, así que el diseño óptimo para cada variedad debería tener $r(k-1) = 24$ variedades opuestas, pero únicamente hay $v - 1 = 11$ posibles variedades distintas. De acuerdo con la sección 2.4.1, en el diseño óptimo cada variedad debe aparecer con otras 9 variedades 2 veces y con otras 2 variedades 3 veces, donde $n_1 = 9, n_2 = 2$, $\lambda_1 = 2$ y $\lambda_2 = 3$. Así que cada renglón de la matriz de concurrencia $C_{ij}$ debe consistir de 9 2's y 2 3's (con 0's en la diagonal). El costo óptimo para este diseño es 12, considerando a $n_i = n_2$ y a $\lambda = \lambda_1$.

**Método 1 (búsqueda exhaustiva).** Para encontrar la solución óptima mediante la búsqueda exhaustiva dada por la ecuación (2.10), para esta instancia en particular el número de posibles diseños diferentes es aproximadamente de $3.05 \times 10^{25}$. Por lo tanto, el método de la búsqueda exhaustiva no es factible computacionalmente.

**Método 2 (BT).** Debido a que el evaluar todas las posibilidades es un procedimiento muy caro, en este caso particular para el problema de la construcción de diseños resolubles con 2 concurrencias, necesitamos un método de búsqueda que sea capaz de buscar una solución óptima examinando solamente un pequeño subconjunto del número total de posibilidades. En este caso, una solución cercana a la óptima no tiene ningún interés, debido a que nuestro objetivo es probar la existencia de los diseños. Por ello usamos este problema para ilustrar los componentes básicos de la BT.

Primero suponemos que la solución inicial para el problema puede ser construida de alguna forma inteligente. Para el problema de optimización

| | | |
|---|---|---|
| $A_{11} = \{2,5,6,12\}$ | $A_{12} = \{1,3,4,10\}$ | $A_{13} = \{7,8,9,11\}$ |
| $A_{21} = \{2,4,6,12\}$ | $A_{22} = \{1,3,5,11\}$ | $A_{23} = \{7,8,9,10\}$ |
| $A_{31} = \{3,7,8,9\}$ | $A_{32} = \{2,5,11,12\}$ | $A_{33} = \{1,4,6,10\}$ |
| $A_{41} = \{2,6,9,12\}$ | $A_{42} = \{3,5,8,10\}$ | $A_{43} = \{1,4,7,11\}$ |
| $A_{51} = \{2,3,9,11\}$ | $A_{52} = \{5,6,7,12\}$ | $A_{53} = \{1,4,8,10\}$ |
| $A_{61} = \{2,3,7,10\}$ | $A_{62} = \{1,5,6,11\}$ | $A_{63} = \{4,8,9,12\}$ |
| $A_{71} = \{4,5,7,8\}$ | $A_{72} = \{3,10,11,12\}$ | $A_{73} = \{1,2,6,9\}$ |
| $A_{81} = \{3,4,5,11\}$ | $A_{82} = \{6,8,9,10\}$ | $A_{83} = \{1,2,7,12\}$ |

Figura 3.1: Solución inicial

combinatoria, construimos la solución inicial "aleatoriamente", la cual se muestra en la figura 3.1.

El orden en el arreglo de la figura 3.1 especifica que las variedades del grupo 1 de la clase paralela 1 ($A_{11}$) son: 2, 5, 6, 12, lo cual significa que aparecen los siguientes pares de variedades juntas: $(2,5),(2,6),(2,12),(5,6)$, $(5,12),(6,12)$, seguido por el grupo 2 de la clase paralela 1 ($A_{12}$), y así sucesivamente hasta el grupo 3 de la clase paralela 8 ($A_{83}$). A partir de la solución aleatoria inicial, se genera la matriz de concurrencia $C_{ij}$ que representa el número de veces que la variedad $i$ aparece junto a la variedad $j$, la cual se muestra en la iteración 0 del siguiente ejemplo. Calculando el valor de la función objetivo para esta solución inicial dada por la ecuación (2.9), da un valor de 88 unidades.

La técnica de BT opera bajo la hipótesis de que se puede construir una *vecindad* para identificar *soluciones adyacentes* (conjunto de movimientos admisibles) que puedan ser alcanzadas desde cualquier solución actual. El intercambio de parejas es frecuentemente usado para definir las vecindades en los problemas de permutaciones, identificando así "*movimientos* que permitan pasar de una solución de prueba a otra". En este problema, los movimientos se realizan intercambiando la posición de dos variedades de grupos diferentes, pero de la misma clase paralela, representados por los atributos de la tripleta $(k,i,j)$ como se muestra en la figura 3.2.

Por lo tanto, la vecindad completa de una solución actual consiste de

| | | |
|---|---|---|
| $A_{11} = \{2,5,6,12\}$ | $A_{12} = \{1,3,4,10\}$ | $A_{13} = \{7,8,9,11\}$ |
| $A_{21} = \{2,4,6,12\}$ | $A_{22} = \{1,3,8,11\}$ | $A_{23} = \{5,7,9,10\}$ |
| $A_{31} = \{3,7,8,9\}$ | $A_{32} = \{2,5,11,12\}$ | $A_{33} = \{1,4,6,10\}$ |
| $A_{41} = \{2,6,9,12\}$ | $A_{42} = \{3,5,8,10\}$ | $A_{43} = \{1,4,7,11\}$ |
| $A_{51} = \{2,3,9,11\}$ | $A_{52} = \{5,6,7,12\}$ | $A_{53} = \{1,4,8,10\}$ |
| $A_{61} = \{2,3,7,10\}$ | $A_{62} = \{1,5,6,11\}$ | $A_{63} = \{4,8,9,12\}$ |
| $A_{71} = \{4,5,7,8\}$ | $A_{72} = \{3,10,11,12\}$ | $A_{73} = \{1,2,6,9\}$ |
| $A_{81} = \{3,4,5,11\}$ | $A_{82} = \{6,8,9,10\}$ | $A_{83} = \{1,2,7,12\}$ |

**Figura 3.2: Intercambio de las variedades 5 y 8 de la clase paralela 2**

384 soluciones adyacentes que pueden ser obtenidas por tales movimientos y está representada por $r\binom{g}{2}kk$, esto es, existen $r$ distintas clases paralelas, y en cada clase paralela hay $\binom{g}{2}$ formas de elegir los dos grupos involucrados en el intercambio de variedades, y hay $k$ formas de elegir una variedad de cada uno de los grupos.

Asociado a cada movimiento está el valor del movimiento *vm*, que nos muestra el cambio en el valor de la función objetivo como resultado del intercambio propuesto y está representado por la diferencia entre el valor de la función objetivo antes y después del movimiento. Generalmente, el valor del movimiento nos proporciona las bases fundamentales para evaluar la *calidad* de un movimiento, aunque otros criterios también son importantes. El mecanismo principal para la explotación de la memoria en la BT es clasificar un subconjunto de los movimientos en una vecindad como prohibidos (o tabú). Por supuesto que la clasificación depende de la historia de la búsqueda, particularmente de los movimientos más *recientes* o de la *frecuencia* de ciertos atributos de un movimiento, que han participado en la generación de soluciones pasadas. En este caso, los atributos de un movimiento están dados por la tripleta $(k,i,j)$ de elementos que intercambian posiciones (de la clase paralela $k$, las variedades $i$ y $j$ intercambian de grupo).

Como base para prevenir repeticiones o ciclos de movimientos realizados recientemente, se anulan los efectos de movimientos previos que por el intercambio puedan regresar a posiciones anteriores. Clasificaremos como movimientos tabú a todos los "intercambios compuestos" por atributos de

cualquier tripleta $(k, i, j)$ más recientes dentro de la lista tabú; en este caso, de las 4 iteraciones más recientes, ya que experimentalmente con esta longitud se obtuvieron los mejores resultados. Esto significa que una tripleta (atributos del movimiento) será considerada tabú por una duración de 4 iteraciones (profundidad del arreglo). La estructura de datos utilizada para la lista tabú es un arreglo de registros de tamaño 4, donde cada registro está formado por tres enteros, para representar a la clase paralela y a las dos variedades que se van a intercambiar.

Las restricciones tabú no son inviolables bajo todas las circunstancias. Cuando un movimiento tabú resulte ser una mejor solución que cualquiera de todas las obtenidas hasta el momento, su clasificación tabú puede ser rechazada y a esta condición se le conoce como el *criterio de aspiración*. En el siguiente ejemplo (que incluye 8 iteraciones) mostramos el procedimiento básico de la BT:

**Iteración 0.** La **solución inicial** se muestra en la figura 2.2.

El costo de la función objetivo es $f = 88$.

$C_{ij}=$

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 2 | 2 | 4 | 2 | 3 | 2 | 1 | 1 | 3 | 3 | 1 |
| 0 | 0 | 2 | 1 | 2 | 4 | 2 | 0 | 3 | 1 | 2 | 5 |
| 0 | 0 | 0 | 2 | 3 | 0 | 2 | 2 | 2 | 4 | 4 | 1 |
| 0 | 0 | 0 | 0 | 2 | 2 | 2 | 3 | 1 | 3 | 2 | 2 |
| 0 | 0 | 0 | 0 | 0 | 3 | 2 | 2 | 0 | 1 | 4 | 3 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 3 | 2 | 1 | 4 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4 | 3 | 2 | 2 | 2 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 5 | 4 | 1 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 2 | 2 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Movimientos

| r | $k_i$ | $k_j$ | vm | f |
|---|---|---|---|---|
| 1 | 1 | 2 | −8 | 80 |
| 1 | 1 | 6 | −10 | 78 |
| 1 | 2 | 9 | −12 | 76 |
| 1 | 4 | 12 | −14 | 74 |
| 1 | 10 | 12 | −16 | 72 |
| 2 | 5 | 8 | −18 | 70 |

Tabú

| r | 0 | 0 | 0 | 0 |
|---|---|---|---|---|
| $k_i$ | 0 | 0 | 0 | 0 |
| $k_j$ | 0 | 0 | 0 | 0 |

La estructura de datos de la lista tabú es inicializada con 0's, indicando así que ningún movimiento es prohibido al principio de la búsqueda. La matriz de solución $Xt$ se generó de manera "aleatoria". La parte superior de la matriz $C_{ij}$ muestra el número de veces que dos variedades aparecen juntas (*matriz de concurrencia*); mientras que la parte inferior muestra el número de veces que dos variedades se han intercambiado (*matriz de frecuencia*) y que inicialmente son 0's, al igual que la diagonal principal, debido a que una variedad no puede aparecer junto a ella misma. Después de evaluar la función objetivo de los 384 movimientos admisibles (vecindad), seleccionamos el mejor movimiento de la vecindad en función de su valor de movimiento y que resultó ser la tripleta $(2, 5, 8)$. En este ejemplo, en caso de que en una iteración dada más de un movimiento tenga el menor valor de la función objetivo en una iteración dada, elegimos el primero de ellos. Mostramos únicamente 6 de los movimientos candidatos en términos del valor del movimiento. Actualizamos la lista tabú con la tripleta seleccionada y la matriz de frecuencia muestra que las variedades 5 y 8 han sido intercambiadas una vez. La matriz de concurrencia únicamente se modificó en los renglones 5 y 8 de aquellas columnas cuyas variedades pertenecían al mismo grupo que las variedades 5 y 8. La función de aspiración también se actualiza con el mejor costo obtenido hasta ahora; $mejor = 70$. El total ganado del movimiento seleccionado equivale a un valor de $vm = -18$ unidades.

**Iteración 1. La solución actual** se muestra en la figura 2.3.

El costo de la función objetivo es $f = 70$.

$$C_{ij}= \begin{bmatrix} 0 & 2 & 2 & 4 & 1 & 3 & 2 & 2 & 1 & 3 & 3 & 1 \\ 0 & 0 & 2 & 1 & 2 & 4 & 2 & 0 & 3 & 1 & 2 & 5 \\ 0 & 0 & 0 & 2 & 2 & 0 & 2 & 3 & 2 & 4 & 4 & 1 \\ 0 & 0 & 0 & 0 & 2 & 2 & 2 & 3 & 1 & 3 & 2 & 2 \\ 0 & 0 & 0 & 0 & 0 & 3 & 3 & 2 & 1 & 2 & 3 & 3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 3 & 2 & 1 & 4 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 & 3 & 2 & 2 & 2 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 4 & 3 & 2 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 2 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Movimientos

| r | $k_i$ | $k_j$ | vm | f |
|---|---|---|---|---|
| 1 | 2 | 4 | −6 | 64 |
| 1 | 2 | 9 | −8 | 62 |
| 1 | 1 | 2 | −10 | 60 |
| 1 | 1 | 6 | −12 | 58 |
| 1 | 4 | 12 | −14 | 56 |
| 4 | 2 | 3 | −18 | 52 |

Tabú

| | | | | |
|---|---|---|---|---|
| r | 2 | 0 | 0 | 0 |
| $k_i$ | 5 | 0 | 0 | 0 |
| $k_j$ | 8 | 0 | 0 | 0 |

El nuevo valor de la función objetivo está representado por su valor anterior más el valor del movimiento seleccionado $f(x') = f(x) + vm$, en nuestro caso $70 = 88 + (-18)$. La lista tabú ahora muestra que el intercambio de las posiciones de las variedades 5 y 8 de la clase paralela 2 estará prohibida durante 4 iteraciones. Es importante mencionar que en cada iteración se revisa la vecindad completa; además todos y cada uno de los vecinos se generan con la misma solución con la que inició la iteración. Después de realizar cada uno de los movimientos de la vecindad, se calcula el costo de tales movimientos, luego se verifica el criterio de aspiración y de restricción tabú. Si cumple el criterio de aspiración, se acepta el movimiento aún cuando éste sea prohibido, ya que el criterio de aspiración rechaza su condición de ser tabú. Si no cumple el criterio de aspiración, penalizamos al movimiento sumándole la frecuencia con la que se han intercambiado las variedades que intervienen en el movimiento. Luego se verifica que no sea un movimiento tabú y aceptamos el movimiento siempre y cuando éste sea menor o igual al mínimo de las penalizaciones realizadas hasta el momento. El mejor movimiento de esta vecindad es la tripleta $(4, 2, 3)$ con un valor de movimiento de $vm = -18$ unidades y $mejor = 52$.

**Iteración 2. Solución actual**: con un costo de $f = 52$.

| | | |
|---|---|---|
| $A_{11} = \{2,5,6,12\}$ | $A_{12} = \{1,3,4,10\}$ | $A_{13} = \{7,8,9,11\}$ |
| $A_{21} = \{2,4,6,12\}$ | $A_{22} = \{1,3,8,11\}$ | $A_{23} = \{5,7,9,10\}$ |
| $A_{31} = \{3,7,8,9\}$ | $A_{32} = \{2,5,11,12\}$ | $A_{33} = \{1,4,6,10\}$ |
| $\mathbf{A_{41}} = \{\mathbf{3},6,9,12\}$ | $\mathbf{A_{42}} = \{\mathbf{2},5,8,10\}$ | $A_{43} = \{1,4,7,11\}$ |
| $A_{51} = \{2,3,9,11\}$ | $A_{52} = \{5,6,7,12\}$ | $A_{53} = \{1,4,8,10\}$ |
| $A_{61} = \{2,3,7,10\}$ | $A_{62} = \{1,5,6,11\}$ | $A_{63} = \{4,8,9,12\}$ |
| $A_{71} = \{4,5,7,8\}$ | $A_{72} = \{3,10,11,12\}$ | $A_{73} = \{1,2,6,9\}$ |
| $A_{81} = \{3,4,5,11\}$ | $A_{82} = \{6,8,9,10\}$ | $A_{83} = \{1,2,7,12\}$ |

$C_{ij}=$

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 2 | 2 | 4 | 1 | 3 | 2 | 2 | 1 | 3 | 3 | 1 |
| 0 | 0 | 2 | 1 | 3 | 3 | 2 | 1 | 2 | 2 | 2 | 4 |
| 0 | 1 | 0 | 2 | 1 | 1 | 2 | 2 | 3 | 3 | 4 | 2 |
| 0 | 0 | 0 | 0 | 2 | 2 | 2 | 3 | 1 | 3 | 2 | 2 |
| 0 | 0 | 0 | 0 | 0 | 3 | 3 | 2 | 1 | 2 | 3 | 3 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 3 | 2 | 1 | 4 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 3 | 2 | 2 | 2 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 4 | 3 | 2 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 2 | 2 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Movimientos Tabú

| r | $k_i$ | $k_j$ | vm | f |
|---|---|---|---|---|
| 1 | 3 | 5 | 0 | 52 |
| 1 | 1 | 5 | −2 | 50 |
| 1 | 2 | 3 | −4 | 48 |
| 1 | 1 | 2 | −6 | 46 |
| 1 | 1 | 6 | −8 | 44 |
| 1 | 4 | 12 | −10 | 42 |
| r | 2 | 4 | 0 | 0 |
| $k_i$ | 5 | 2 | 0 | 0 |
| $k_j$ | 8 | 3 | 0 | 0 |

Cada vez que se selecciona un mínimo local de la vecindad actual, los atributos del movimiento se guardan en un registro para saber que movimiento es el que produce el mínimo local después de haber revisado la vecindad completa. Luego con el movimiento seleccionado, actualizamos a la matriz de solución $Xt$ y a la matriz de concurrencia únicamente en las entradas que resultan ser modificadas. Se actualiza la lista tabú con los atributos del movimiento, la matriz de frecuencia se incrementa en una unidad en la entrada correspondiente a las dos variedades intercambiadas. Si el costo del movimiento seleccionado es menor que el mejor costo encontrado hasta el momento, entonces se actualiza la función aspiración. Hasta ahora ningún movimiento ha sido clasificado como tabú. Note que la tripleta $(4,2,3)$ permanecerá tres iteraciones más como movimiento prohibido. El mejor movimiento encontrado en esta iteración es la tripleta $(1,4,12)$ con un valor de movimiento de $vm = -10$ unidades y $mejor = 42$.

**Iteración 3. Solución actual:** con un costo de $f = 42$.

| | | |
|---|---|---|
| $A_{11} = \{2,4,5,6\}$ | $A_{12} = \{1,3,10,12\}$ | $A_{13} = \{7,8,9,11\}$ |
| $A_{21} = \{2,4,6,12\}$ | $A_{22} = \{1,3,8,11\}$ | $A_{23} = \{5,7,9,10\}$ |
| $A_{31} = \{3,7,8,9\}$ | $A_{32} = \{2,5,11,12\}$ | $A_{33} = \{1,4,6,10\}$ |
| $A_{41} = \{3,6,9,12\}$ | $A_{42} = \{2,5,8,10\}$ | $A_{43} = \{1,4,7,11\}$ |
| $A_{51} = \{2,3,9,11\}$ | $A_{52} = \{5,6,7,12\}$ | $A_{53} = \{1,4,8,10\}$ |
| $A_{61} = \{2,3,7,10\}$ | $A_{62} = \{1,5,6,11\}$ | $A_{63} = \{4,8,9,12\}$ |
| $A_{71} = \{4,5,7,8\}$ | $A_{72} = \{3,10,11,12\}$ | $A_{73} = \{1,2,6,9\}$ |
| $A_{81} = \{3,4,5,11\}$ | $A_{82} = \{6,8,9,10\}$ | $A_{83} = \{1,2,7,12\}$ |

$C_{ij}=$

$$\begin{bmatrix}
0 & 2 & 2 & 3 & 1 & 3 & 2 & 2 & 1 & 3 & 3 & 2 \\
0 & 0 & 2 & 2 & 3 & 3 & 2 & 1 & 2 & 2 & 2 & 3 \\
0 & 1 & 0 & 1 & 1 & 1 & 2 & 2 & 3 & 3 & 4 & 3 \\
0 & 0 & 0 & 0 & 3 & 3 & 2 & 3 & 1 & 2 & 2 & 2 \\
0 & 0 & 0 & 0 & 0 & 3 & 3 & 2 & 1 & 2 & 3 & 2 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 3 & 2 & 1 & 3 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 & 3 & 2 & 2 & 2 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 4 & 3 & 2 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 2 & 2 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}$$

Movimientos Tabú

| r | $k_i$ | $k_j$ | vm | f |
|---|---|---|---|---|
| 1 | 1 | 2 | 10 | 52 |
| 1 | 1 | 4 | 4 | 46 |
| 1 | 2 | 3 | 0 | 42 |
| 1 | 2 | 9 | −2 | 40 |
| 1 | 3 | 5 | −4 | 38 |
| 1 | 6 | 9 | −10 | 32 |
| r | 2 | 4 | 1 | 0 |
| $k_i$ | 5 | 2 | 4 | 0 |
| $k_j$ | 8 | 3 | 12 | 0 |

En el algoritmo no utilizamos un número fijo de iteraciones, debido a que en ocasiones el costo de la función seguía decrementando; pero el proceso se interrumpía por haber llegado al límite de iteraciones. Por ello optamos por hacer variable el número de iteraciones, es decir, usamos un máximo de iteraciones, pero en que no se produzca una mejor solución que la mejor encontrada hasta el momento. Cada vez que se actualiza la función de aspiración, este contador de iteraciones se inicializa de nuevo a cero. Es decir, las condiciones de paro del algoritmo son: que encuentre la cota mínima de la función objetivo o que transcurra el máximo número de iteraciones en que no se haya encontrado una mejor solución que la *mejor* encontrada hasta el momento. El mejor movimiento encontrado en esta iteración es la tripleta $(1,6,9)$ con un valor de movimiento de $vm = -10$ unidades y $mejor = 32$.

**Iteración 4. Solución actual:** con un costo de $f = 32$.

| | | |
|---|---|---|
| $A_{11} = \{2,4,5,9\}$ | $A_{12} = \{1,3,10,12\}$ | $A_{13} = \{6,7,8,11\}$ |
| $A_{21} = \{2,4,6,12\}$ | $A_{22} = \{1,3,8,11\}$ | $A_{23} = \{5,7,9,10\}$ |
| $A_{31} = \{3,7,8,9\}$ | $A_{32} = \{2,5,11,12\}$ | $A_{33} = \{1,4,6,10\}$ |
| $A_{41} = \{3,6,9,12\}$ | $A_{42} = \{2,5,8,10\}$ | $A_{43} = \{1,4,7,11\}$ |
| $A_{51} = \{2,3,9,11\}$ | $A_{52} = \{5,6,7,12\}$ | $A_{53} = \{1,4,8,10\}$ |
| $A_{61} = \{2,3,7,10\}$ | $A_{62} = \{1,5,6,11\}$ | $A_{63} = \{4,8,9,12\}$ |
| $A_{71} = \{4,5,7,8\}$ | $A_{72} = \{3,10,11,12\}$ | $A_{73} = \{1,2,6,9\}$ |
| $A_{81} = \{3,4,5,11\}$ | $A_{82} = \{6,8,9,10\}$ | $A_{83} = \{1,2,7,12\}$ |

$C_{ij}=$

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 2 | 2 | 3 | 1 | 3 | 2 | 2 | 1 | 3 | 3 | 2 |
| 0 | 0 | 2 | 2 | 3 | 2 | 2 | 1 | 3 | 2 | 2 | 3 |
| 0 | 1 | 0 | 1 | 1 | 1 | 2 | 2 | 3 | 3 | 4 | 3 |
| 0 | 0 | 0 | 0 | 3 | 2 | 2 | 3 | 2 | 2 | 2 | 2 |
| 0 | 0 | 0 | 0 | 0 | 2 | 3 | 2 | 2 | 2 | 3 | 2 |
| 0 | 0 | 0 | 0 | 0 | 0 | 2 | 2 | 3 | 2 | 2 | 3 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 2 | 2 | 2 | 2 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 3 | 3 | 2 | 1 |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 2 | 1 | 2 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 2 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Movimientos

| r | $k_i$ | $k_j$ | vm | f |
|---|---|---|---|---|
| 1 | 4 | 12 | 10 | 42 |
| 1 | 1 | 2 | 6 | 38 |
| 1 | 1 | 4 | 2 | 34 |
| 1 | 3 | 9 | 0 | 32 |
| 2 | 3 | 12 | −2 | 30 |
| 2 | 5 | 11 | −4 | 28 |

Tabú

| r | 2 | 4 | 1 | 1 |
|---|---|---|---|---|
| $k_i$ | 5 | 2 | 4 | 6 |
| $k_j$ | 8 | 3 | 12 | 9 |

La lista tabú nos indica que 4 movimientos se clasifican como tabú, los cuales permanecerán prohibidos durante algún número de iteraciones distinto. En la tercera posición de la lista, encontramos a la tripleta $(1,4,12)$, que representa al movimiento realizado en la iteración anterior, aplicando el criterio de restricción tabú, este movimiento no podría ser seleccionado; a menos que el valor de su función objetivo dé un valor inferior a cualquier valor previo, en cuyo caso utilizaríamos el criterio de aspiración para rechazar la condición tabú del movimiento y seleccionarlo. El mejor movimiento encontrado en esta iteración es la tripleta $(2,5,11)$ con un valor de movimiento de $vm = -4$ unidades y $mejor = 28$.

**Iteración 5. Solución actual:** con un costo de $f = 28$.

| | | |
|---|---|---|
| $A_{11} = \{2,4,5,9\}$ | $A_{12} = \{1,3,10,12\}$ | $A_{13} = \{6,7,8,11\}$ |
| $A_{21} = \{2,4,6,12\}$ | $A_{22} = \{1,3,5,8\}$ | $A_{23} = \{7,9,10,11\}$ |
| $A_{31} = \{3,7,8,9\}$ | $A_{32} = \{2,5,11,12\}$ | $A_{33} = \{1,4,6,10\}$ |
| $A_{41} = \{3,6,9,12\}$ | $A_{42} = \{2,5,8,10\}$ | $A_{43} = \{1,4,7,11\}$ |
| $A_{51} = \{2,3,9,11\}$ | $A_{52} = \{5,6,7,12\}$ | $A_{53} = \{1,4,8,10\}$ |
| $A_{61} = \{2,3,7,10\}$ | $A_{62} = \{1,5,6,11\}$ | $A_{63} = \{4,8,9,12\}$ |
| $A_{71} = \{4,5,7,8\}$ | $A_{72} = \{3,10,11,12\}$ | $A_{73} = \{1,2,6,9\}$ |
| $A_{81} = \{3,4,5,11\}$ | $A_{82} = \{6,8,9,10\}$ | $A_{83} = \{1,2,7,12\}$ |

$C_{ij}=$

$$\begin{bmatrix} 0 & 2 & 2 & 3 & 2 & 3 & 2 & 2 & 1 & 3 & 2 & 2 \\ 0 & 0 & 2 & 2 & 3 & 2 & 2 & 1 & 3 & 2 & 2 & 3 \\ 0 & 1 & 0 & 1 & 2 & 1 & 2 & 2 & 3 & 3 & 3 & 3 \\ 0 & 0 & 0 & 0 & 3 & 2 & 2 & 3 & 2 & 2 & 2 & 2 \\ 0 & 0 & 0 & 0 & 0 & 2 & 2 & 3 & 1 & 1 & 3 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & 2 & 3 & 2 & 2 & 3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 & 2 & 2 & 3 & 2 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 3 & 3 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 2 & 2 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 2 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Movimientos:

| r | $k_i$ | $k_j$ | vm | f |
|---|---|---|---|---|
| 1 | 4 | 12 | 10 | 38 |
| 1 | 1 | 2 | 8 | 36 |
| 1 | 1 | 4 | 4 | 32 |
| 1 | 2 | 7 | 2 | 30 |
| 3 | 1 | 3 | 0 | 28 |
| 3 | 5 | 8 | −4 | 24 |

Tabú:

| | | | | |
|---|---|---|---|---|
| r | 2 | 4 | 1 | 1 |
| $k_i$ | 5 | 2 | 4 | 6 |
| $k_j$ | 11 | 3 | 12 | 9 |

Algo que hace muy eficiente al algoritmo en cuanto al tiempo de cómputo, es el optimizar el cálculo de la función objetivo. La función que calcula el costo de un diseño únicamente es llamada cuando se genera la solución aleatoria inicial. Al calcular el costo del diseño para cada uno de los movimientos (vecinos), únicamente se calcula el costo de las entradas de la matriz de concurrencia que fueron modificadas por el movimiento realizado, es decir, al costo total se le resta el costo de las entradas a modificar y se le suma el costo de las entradas ya modificadas. Evitando de esta manera calcular innecesariamente el costo de las entradas que no se modificaron al realizar el movimiento. El mejor movimiento en esta iteración es la tripleta $(3,5,8)$ con un valor de movimiento de $vm = -4$ unidades y $mejor = 24$.

**Iteración 6. Solución actual:** con un costo de $f = 24$.

| | | |
|---|---|---|
| $A_{11} = \{2,4,5,9\}$ | $A_{12} = \{1,3,10,12\}$ | $A_{13} = \{6,7,8,11\}$ |
| $A_{21} = \{2,4,6,12\}$ | $A_{22} = \{1,3,5,8\}$ | $A_{23} = \{7,9,10,11\}$ |
| $\mathbf{A_{31}} = \{3,5,7,9\}$ | $\mathbf{A_{32}} = \{2,6,11,12\}$ | $A_{33} = \{1,4,6,10\}$ |
| $A_{41} = \{3,6,9,12\}$ | $A_{42} = \{2,5,8,10\}$ | $A_{43} = \{1,4,7,11\}$ |
| $A_{51} = \{2,3,9,11\}$ | $A_{52} = \{5,6,7,12\}$ | $A_{53} = \{1,4,8,10\}$ |
| $A_{61} = \{2,3,7,10\}$ | $A_{62} = \{1,5,6,11\}$ | $A_{63} = \{4,8,9,12\}$ |
| $A_{71} = \{4,5,7,8\}$ | $A_{72} = \{3,10,11,12\}$ | $A_{73} = \{1,2,6,9\}$ |
| $A_{81} = \{3,4,5,11\}$ | $A_{82} = \{6,8,9,10\}$ | $A_{83} = \{1,2,7,12\}$ |

$C_{ij} =$

$$\begin{bmatrix} 0 & 2 & 2 & 3 & 2 & 3 & 2 & 2 & 1 & 3 & 2 & 2 \\ 0 & 0 & 2 & 2 & 2 & 2 & 2 & 2 & 3 & 2 & 2 & 3 \\ 0 & 1 & 0 & 1 & 3 & 1 & 2 & 1 & 3 & 3 & 3 & 3 \\ 0 & 0 & 0 & 0 & 3 & 2 & 2 & 3 & 2 & 2 & 2 & 2 \\ 0 & 0 & 0 & 0 & 0 & 2 & 3 & 3 & 2 & 1 & 2 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & 2 & 3 & 2 & 2 & 3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 2 & 2 & 3 & 2 \\ 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 2 & 3 & 2 & 2 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 2 & 2 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 2 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Movs. Tabú

| r | $k_i$ | $k_j$ | vm | f |
|---|---|---|---|---|
| 1 | 6 | 9 | 10 | 34 |
| 2 | 5 | 11 | 8 | 32 |
| 3 | 5 | 8 | 4 | 28 |
| 1 | 3 | 5 | 2 | 26 |
| 3 | 1 | 3 | −2 | 22 |
| 7 | 3 | 5 | −8 | 16 |

| r | 2 | 3 | 1 | 1 |
|---|---|---|---|---|
| $k_i$ | 5 | 5 | 4 | 6 |
| $k_j$ | 11 | 8 | 12 | 9 |

Ahora encontramos que las tripletas $(1,6,9)$, $(2,5,11)$ y $(3,5,8)$ están entre los posibles movimientos, pero estos movimientos están prohibidos y ninguno de ellos mejora el valor de la función objetivo hasta ahora encontrado. En la matriz de frecuencia podemos observar que la posición $(8,5)$ tiene un valor de dos unidades, esto se debe a que los movimientos realizados en la primera y sexta iteraciones intercambiaron las variedades 5 y 8 de las clases paralelas 2 y 3 respectivamente. El mejor movimiento de esta iteración es la tripleta $(7,3,5)$ con un valor de movimiento de $vm = -8$ unidades y $mejor = 16$.

**Iteración 7. Solución actual:** con un costo de $f = 16$.

| | | |
|---|---|---|
| $A_{11} = \{2,4,5,9\}$ | $A_{12} = \{1,3,10,12\}$ | $A_{13} = \{6,7,8,11\}$ |
| $A_{21} = \{2,4,6,12\}$ | $A_{22} = \{1,3,5,8\}$ | $A_{23} = \{7,9,10,11\}$ |
| $A_{31} = \{3,5,7,9\}$ | $A_{32} = \{2,8,11,12\}$ | $A_{33} = \{1,4,6,10\}$ |
| $A_{41} = \{3,6,9,12\}$ | $A_{42} = \{2,5,8,10\}$ | $A_{43} = \{1,4,7,11\}$ |
| $A_{51} = \{2,3,9,11\}$ | $A_{52} = \{5,6,7,12\}$ | $A_{53} = \{1,4,8,10\}$ |
| $A_{61} = \{2,3,7,10\}$ | $A_{62} = \{1,5,6,11\}$ | $A_{63} = \{4,8,9,12\}$ |
| $A_{71} = \{3,4,7,8\}$ | $A_{72} = \{5,10,11,12\}$ | $A_{73} = \{1,2,6,9\}$ |
| $A_{81} = \{3,4,5,11\}$ | $A_{82} = \{6,8,9,10\}$ | $A_{83} = \{1,2,7,12\}$ |

$$C_{ij} = \begin{bmatrix} 0 & 2 & 2 & 3 & 2 & 3 & 2 & 2 & 1 & 3 & 2 & 2 \\ 0 & 0 & 2 & 2 & 2 & 2 & 2 & 2 & 3 & 2 & 2 & 3 \\ 0 & 1 & 0 & 2 & 3 & 1 & 3 & 2 & 3 & 2 & 2 & 2 \\ 0 & 0 & 0 & 0 & 2 & 2 & 2 & 3 & 2 & 2 & 2 & 2 \\ 0 & 0 & 1 & 0 & 0 & 2 & 2 & 2 & 2 & 2 & 3 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & 2 & 3 & 2 & 2 & 3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 2 & 2 & 3 & 2 \\ 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 2 & 3 & 2 & 2 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 2 & 2 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 2 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Movimientos

| r | $k_i$ | $k_j$ | vm | f |
|---|---|---|---|---|
| 3 | 5 | 8 | 12 | 28 |
| 1 | 6 | 9 | 10 | 26 |
| 2 | 5 | 11 | 8 | 24 |
| 2 | 2 | 3 | 6 | 22 |
| 2 | 3 | 12 | 4 | 20 |
| 3 | 1 | 3 | −4 | 12 |

Tabú

| r | 2 | 3 | 7 | 1 |
|---|---|---|---|---|
| $k_i$ | 5 | 5 | 3 | 6 |
| $k_j$ | 11 | 8 | 5 | 9 |

Nuevamente encontramos que las tripletas $(1,6,9)$, $(2,5,11)$ y $(3,5,8)$ están entre los posibles movimientos, pero todos ellos están prohibidos y ninguno mejora el valor de la función objetivo hasta ahora encontrado. El mejor movimiento encontrado en esta iteración ha sido la tripleta $(3,1,3)$ con un valor de movimiento de $vm = -4$ unidades y $mejor = 12$. Por lo tanto al realizar este último movimiento se obtiene el valor óptimo del diseño.

**Iteración 8. Solución actual:** con un costo de $f = 12$.

| | | |
|---|---|---|
| $A_{11} = \{2,4,5,9\}$ | $A_{12} = \{1,3,10,12\}$ | $A_{13} = \{6,7,8,11\}$ |
| $A_{21} = \{2,4,6,12\}$ | $A_{22} = \{1,3,5,8\}$ | $A_{23} = \{7,9,10,11\}$ |
| $A_{31} = \{1,5,7,9\}$ | $A_{32} = \{2,8,11,12\}$ | $A_{33} = \{3,4,6,10\}$ |
| $A_{41} = \{3,6,9,12\}$ | $A_{42} = \{2,5,8,10\}$ | $A_{43} = \{1,4,7,11\}$ |
| $A_{51} = \{2,3,9,11\}$ | $A_{52} = \{5,6,7,12\}$ | $A_{53} = \{1,4,8,10\}$ |
| $A_{61} = \{2,3,7,10\}$ | $A_{62} = \{1,5,6,11\}$ | $A_{63} = \{4,8,9,12\}$ |
| $A_{71} = \{3,4,7,8\}$ | $A_{72} = \{5,10,11,12\}$ | $A_{73} = \{1,2,6,9\}$ |
| $A_{81} = \{3,4,5,11\}$ | $A_{82} = \{6,8,9,10\}$ | $A_{83} = \{1,2,7,12\}$ |

$C_{ij} =$

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 2 | 2 | 2 | 3 | 2 | 3 | 2 | 2 | 2 | 2 | 2 |
| 0 | 0 | 2 | 2 | 2 | 2 | 2 | 2 | 3 | 2 | 2 | 3 |
| 1 | 1 | 0 | 3 | 2 | 2 | 2 | 2 | 2 | 3 | 2 | 2 |
| 0 | 0 | 0 | 0 | 2 | 2 | 2 | 3 | 2 | 2 | 2 | 2 |
| 0 | 0 | 1 | 0 | 0 | 2 | 2 | 2 | 2 | 2 | 3 | 2 |
| 0 | 0 | 0 | 0 | 0 | 0 | 2 | 2 | 3 | 2 | 2 | 3 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 2 | 2 | 3 | 2 |
| 0 | 0 | 0 | 0 | 2 | 0 | 0 | 0 | 2 | 3 | 2 | 2 |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 2 | 2 | 2 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 2 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 2 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Movimientos Tabú

| | r | $k_i$ | $k_j$ | vm | f |
|---|---|---|---|---|---|
| | 0 | 0 | 0 | 0 | 0 |
| | 0 | 0 | 0 | 0 | 0 |
| | 0 | 0 | 0 | 0 | 0 |
| | 0 | 0 | 0 | 0 | 0 |
| | 0 | 0 | 0 | 0 | 0 |
| | 0 | 0 | 0 | 0 | 0 |
| | r | 2 | 3 | 7 | 3 |
| | $k_i$ | 5 | 5 | 3 | 1 |
| | $k_j$ | 11 | 8 | 5 | 3 |

Como puede observarse, en esta iteración no hay movimientos debido a que se ha alcanzado el valor óptimo de la función objetivo $f = 12$. Por lo tanto en la matriz de concurrencia $C_{ij}$ que es simétrica, en cada renglón aparecen 9 2's y 2 3's, que hacen un total de 54 2's y 12 3's por encima de la diagonal principal, terminando así el método de BT. Entonces el arreglo ordenado de las 8 clases paralelas con sus 3 grupos de 4 variedades cada uno, cumple con las propiedades que se pedían al plantear el problema como un problema de optimización combinatoria para el diseño de experimentos resoluble con 2 concurrencias.

**Estructuras complementarias a la memoria tabú.** La lista tabú de los movimientos más recientes junto con la frecuencia de los movimientos agrega un componente que típicamente opera como la *memoria larga* en la BT. Para mostrar su uso, supongamos que de las 8 iteraciones que se han realizado, se ha almacenado en la *matriz de frecuencia* el número de veces que cada pareja de variedades se ha intercambiado. En la última iteración, la lista tabú (movimientos más recientes) indica que los últimos 4 movimientos realizados fueron las tripletas: $(3,1,3)$, $(7,3,5)$, $(3,5,8)$ y $(2,5,11)$. La frecuencia nos muestra la distribución de los movimientos a lo largo de las 8 iteraciones. Utilizamos la frecuencia para *diversificar* la búsqueda, conduciéndola a nuevas regiones no visitadas anteriormente.

La *influencia* de la diversificación está restringida a operar únicamente en ocasiones particulares. En este caso, cuando los movimientos admisibles no mejoran la solución actual, utilizando la información de la frecuencia para *penalizar* a esos movimientos asignándoles una *pena*. En el ejemplo simplemente sumamos la frecuencia al valor del movimiento asociado, $pena = vm + frecuencia(i,j)$ donde $i$ y $j$ son variedades distintas.

La lista de posibles movimientos admisibles en la iteración 7, muestra que los 3 primeros movimientos están prohibidos, pero la tripleta $(2,2,3)$ tiene un valor de movimiento de $vm = 6$ unidades y podría ser el siguiente movimiento elegido, excepto que sus variedades asociadas ya han sido intercambiadas en una ocasión durante la historia de la búsqueda. Por ello el movimiento es penalizado con un valor de movimiento de $vm = 7$ unidades, aunque no por ello deja de ser atractivo en el momento en que es analizado ya que ningún movimiento en su vecindad lo había mejorado aún con la penalización.

La estrategia de las penalizaciones únicamente bajo condiciones particulares se usa para preservar la agresividad de la búsqueda. Las frecuencias definidas sobre diferentes subconjuntos de soluciones pasadas, particularmente los subconjuntos de soluciones élite formados por óptimos locales de alta calidad, dan origen a las estrategias complementarias de *intensificación*. Las estrategias de intensificación y diversificación interactúan para proveer la motivación fundamental de la memoria larga en la BT.

En el siguiente capítulo veremos como se desarrolló el programa de la Búsqueda Tabú para resolver este problema.

# Capítulo 4

# Programación de la Búsqueda Tabú

Un algoritmo de la Búsqueda Tabú (BT) no puede ser aplicado a cualquier tipo de problema de optimización combinatoria; debido a que cada problema se plantea de forma diferente. A continuación explicamos cómo desarrollamos los programas para resolver el problema de la construcción de diseños resolubles con 2 concurrencias. Los listados de los programas aparecen en los apéndices A y B. En el apéndice C se muestra la solución de 48 de los 76 diseños obtenidos, los cuales según la literatura revisada, se desconocía su existencia.

El lenguaje de programación utilizado para desarrollar el programa fue el *Lenguaje C* [16], básicamente porque *C* es un lenguaje que nos permite correr nuestra aplicación en varios ambientes (*UNIX, MS-DOS*) y equipos prácticamente sin hacer ninguna modificación, es decir por su *portabilidad*. Cabe mencionar también que nos fue de gran utilidad para poder adaptar el programa secuencial a un programa en paralelo; los resultados que se muestran en el siguiente capítulo son una comparación de ambos programas: secuencial versus paralelo.

Los resultados del programa secuencial se obtuvieron al correr un programa ejecutable, generado por el compilador de *Borland C++ 3.1* en una computadora personal *PC 486DX a 33Mhz*, mientras que los resultados del programa en paralelo se obtuvieron al correr un programa ejecutable generado por el compilador de Unix *cc* junto con las bibliotecas de *CMMD* (sis-

tema de paso de mensajes) [29] de una *CM-5* (Connection Machine) con *32 procesadores.*

## 4.1 Programas

### 4.1.1 Generación de instancias

Conociendo los parámetros de un diseño resoluble con 2 concurrencias, no sabemos si existe un conjunto ordenado (de variedades en bloques) que cumpla con las condiciones de estos diseños. Por ello se generó una lista de 317 posibles diseños para probar su existencia y ver que tan eficiente resulta ser la BT para resolver este problema. La lista se generó de la siguiente manera:

El intervalo de los parámetros $(v, b, r, k)$ con los que fue generada la lista son los siguientes: $7 \leq v \leq 50$, $5 \leq r \leq 20$, $v+1 \leq b \leq 51$ y $3 \leq k \leq \frac{v}{2}$, además la lista aparece ordenada lexicográficamente por $k$ y $r$ (en ese orden).

Se realizan 4 ciclos anidados correspondientes a los parámetros anteriores y se verifica que se cumpla con la condición necesaria: $vr = bk$; si se cumple esta condición, entonces debemos ver si se trata de un diseño con 2 concurrencias verificando que la división $\frac{r(k-1)}{v-1}$ no sea entera, si esto se cumple, entonces verificamos que se trate de un diseño resoluble si $\frac{v}{k} = \frac{b}{r}$; es decir que sean el mismo entero.

Una vez que ya sabemos que con los parámetros $(v, b, r, k)$ se tiene un posible diseño resoluble con 2 concurrencias, procedemos a obtener los parámetros de las 2 concurrencias: $n_1$, $n_2$, $\lambda_1$ y $\lambda_2$ de la siguiente manera:

- A $g$ le asignamos la parte entera de $\frac{v}{k}$.
- A $n_2$ le asignamos el resto de la división $\frac{r(k-1)}{v-1}$.
- A $n_1$ le asignamos la diferencia de $(v-1)$ y $n_2$.
- A $\lambda_1$ le asignamos la parte entera de la división $\frac{r(k-1)}{v-1}$.
- A $\lambda_2$ le asignamos el valor de $\lambda_1$ más una unidad.

De esta manera obtenemos los diseños que difieren en una unidad entre $\lambda_1$ y $\lambda_2$; luego obtenemos la cota inferior de la función objetivo de la siguiente manera:

- Elegimos el mínimo de $n_1$ y $n_2$ y lo multiplicamos por $\frac{v}{2}$.

El valor de $\lambda$, utilizado en la función objetivo como el valor promedio que dos variedades deben aparecer juntas, lo elegimos de la siguiente manera:

- si $n_i = \min(n_1, n_2)$, entonces $\lambda = \lambda_j$ para $j \neq i$, donde $i$ y $j = 1$ ó $2$.

### 4.1.2 Búsqueda Tabú secuencial

Primero definimos las variables que se van a utilizar en el programa, algunas son de carácter global y otras sólo se utilizan localmente en algunos procedimientos. Las variables que son globales son las siguientes:

- **v** número de variedades del diseño.
- **b** número de bloques del diseño.
- **r** número de clases paralelas del diseño.
- **k** número de variedades en cada bloque del diseño.
- **g** número de grupos en cada clase paralela del diseño.
- **$n_1$** número de variedades uno asociadas.
- **$n_2$** número de variedades dos asociadas.
- **$l_1$** número de veces que aparecen juntas $n_1$ variedades.
- **$l_2$** número de veces que aparecen juntas $n_2$ variedades.
- **fmin** cota mínima (valor óptimo) de la función objetivo.
- **lambda** valor promedio de veces que dos variedades aparecen juntas.
- **itera** número máximo de iteraciones que no registre ningún mejoramiento.
- **ntabu** tamaño de la lista tabú.

- **costot** costo de la función objetivo para cada vecino de la solución actual.
- **mejor** mejor costo de la función objetivo obtenido hasta el momento.
- **nits** número de iteraciones transcurridas.
- **num** número de la instancia leída del archivo de entrada.
- **Cij[Maxv][Maxv]** Matriz de concurrencia (parte superior) y matriz de frecuencia (parte inferior).
- **Xt[Maxr][Maxv]** Matriz que representa al conjunto ordenado de la solución, donde *Maxv* y *Maxr* determinan el valor máximo de los parámetros $v$ y $r$ respectivamente, es decir la memoria se reserva de manera estática.
- **tabu[Maxt]** Vector (registro con tres índices: una clase paralela y dos variedades) que representa a la lista tabú (movimientos prohibidos). *Maxt* es el tamaño máximo de la lista, también la memoria se reserva de manera estática.

Las variables que se utilizan localmente en cada procedimiento o función, no se mencionan debido a que éstas son índices o variables temporales que son utilizadas como auxiliares para realizar los cálculos. El programa está compuesto de 6 procedimientos, dos funciones y el programa principal.

### Procedimientos

*Lee_Datos(Archivo)*. Recibe como parámetro el nombre del archivo de lectura, del cual se leen los siguientes parámetros: $num$, $v$, $b$, $r$, $k$, y con ellos genera los parámetros faltantes: $g$, $n_1$, $n_2$, $l_1$, $l_2$, $lambda$ y $fmin$ como se mencionó en la sección anterior. Además se valida la información para detectar si se trata de un posible diseño resoluble con 2 concurrencias, en caso contrario, termina la ejecución del programa.

*cambiomatriz*(). Este procedimiento se utiliza para crear la matriz de concurrencia (número de veces que dos variedades aparecen juntas en algún bloque) a partir de la información de la matriz de solución $Xt[r][v]$. Se inicializa con 0's la matriz de concurrencia; luego para $1 \leq k \leq r$, $1 \leq i < v$ y $i+1 \leq j \leq v$, se compara si $Xt[k][i] = Xt[k][j]$. Es decir, si de la matriz de

solución en la clase paralela $k$ la variedad $i$ y $j$ tienen como valor el mismo grupo $g$, entonces $Cij[i][j]$ se incrementa en una unidad, representando así que la pareja de variedades $i$ y $j$ aparecen juntas en un bloque.

*despsol(Archivo)*. Recibe como parámetro el nombre del archivo de escritura, en el que se guarda la solución del conjunto ordenado únicamente en el caso en que se alcance el valor óptimo de la función objetivo. Para cada clase paralela $1 \leq l \leq r$ y cada uno de sus grupos $1 \leq i \leq g$, se verifica cuáles son las $k$ variedades que pertenecen a ese bloque revisando para $1 \leq j \leq v$ si $Xt[l][j] = i$, es decir, verificando cuáles variedades $j$ de la matriz de solución para cada clase paralela $l$ pertenecen al mismo grupo $i$. También se guarda la matriz de concurrencia, en la que se muestra $\frac{vn_1}{2}$ veces $\lambda_1$, $\frac{vn_2}{2}$ veces $\lambda_2$; y la matriz de frecuencia en la que se muestra la distribución de las iteraciones entre las variedades intercambiadas durante el proceso de búsqueda.

*init()*. Se encarga de inicializar a ceros el vector de los movimientos prohibidos *tabu* y las matrices de concurrencia y de frecuencia $Cij$, así como también la matriz de solución $Xt$. Por eficiencia se utiliza la función *memset* $(s, 0, sizeof(s))$, esta función llena con el byte 0 todos los elementos del vector o matriz $s$. Además esta función genera una solución aleatoria inicial.

La solución aleatoria inicial, se obtiene de la siguiente manera: primero a cada columna de la matriz de solución $Xt$, se le asigna el valor correspondiente a esa variedad. $Xt[i][j] = j$, para $1 \leq i \leq r$ y $1 \leq j \leq v$; luego para $1 \leq i \leq r$ y $v \geq j \geq 1$ se genera un número aleatorio $j_0$ entre 1 y $j$. Se realiza un cambio entre la variedad $j$ y $j_0$: $cambio = Xt[i][j_0]$, $Xt[i][j_0] = Xt[i][j]$, $Xt[i][j] = cambio$. Finalmente para cada entrada de la matriz de solución, se le asigna el valor de la entrada $Xt[i][j]$ módulo $g$, con el fin de que el valor de cada entrada de la matriz represente al grupo $g$ al que pertenece. Los $r$ renglones de la matriz representan a las clases paralelas del diseño, mientras que las $v$ columnas representan a las variedades del diseño. Entonces el valor de cada entrada de la matriz de solución representa al grupo $g$ de la $i$-ésima clase paralela, al cual pertenece la $j$-ésima variedad.

Después de haber generado la matriz de solución aleatoria inicial, se hace la llamada al procedimiento cambiomatriz() para crear la matriz de concurrencia, también se obtiene el costo inicial del diseño llamando a la función Costo(), la cual se asigna a la variable *costot*.

*Tiempo*($t$, *Archivo*). Recibe como parámetros: el tiempo $t$ expresado en segundos y el archivo de escritura en el que se guarda el tiempo de ejecución expresado en horas, minutos y/o segundos.

*Tabu*(). Este procedimiento es el más importante, ya que éste contiene básicamente la técnica de BT, además los tiempos reportados en el capítulo siguiente son exclusivamente de este procedimiento. Las variables locales que se utilizan son las siguientes:

- **minimo** valor mínimo encontrado en la vecindad actual.
- **costop** valor mínimo de la iteración anterior con el que reinicia la siguiente iteración.
- **minimop** valor mínimo penalizado para diversificar la búsqueda.
- **sump** equivale al valor del movimiento *vm* de cada vecino visitado.
- **pena** costo penalizado de acuerdo a la matriz de frecuencia.
- **iter** indica en qué iteración ocurrió el último mejor movimiento.
- **p** es la diferencia entre el valor de la entrada $Cij[i][j]$ y el valor de *lambda*.
- **m** registro que guarda los tres atributos del movimiento correspondiente al mínimo valor encontrado en la vecindad.

Como hemos mencionado, un movimiento lo representamos por los atributos de la tripleta $(k, i, j)$, que indica que las variedades $i$ y $j$ de la clase paralela $k$ se intercambian de grupo. Así, el dominio de los movimientos son todas aquellas tripletas $(k, i, j)$ tales que las variedades $i$ y $j$ de cualquier clase paralela no están en el mismo grupo. Para el problema de la construcción de diseños resolubles con 2 concurrencias, el tamaño de la vecindad puede calcularse de la siguiente manera: $r\binom{g}{2}kk$, esto es, existen $r$ distintas clases paralelas, y en cada clase paralela hay $\binom{g}{2}$ formas de elegir los dos grupos involucrados en el intercambio de variedades, y hay $k$ formas de elegir una variedad de cada uno de los grupos.

A *costop* le asignamos el costo de la solución aleatoria inicial e inicializamos el número de iteraciones *nits* a cero. Empezamos el proceso iterativo en un ciclo que se repite mientras el mejor valor encontrado sea mayor al

óptimo de la función objetivo *fmin* y el número de iteraciones en que no se ha mejorado a la mejor solución sea menor que el máximo número de iteraciones dado como parámetro. *while*(*mejor* > *fmin* && (*nits* − *iter*) < *itera*).

Antes de iniciar la búsqueda del movimiento que produzca el *mínimo* dentro de la vecindad actual, incrementamos el número de iteraciones transcurridas, y a las variables *minimo* y *minimop* les asignamos un valor grande para que se acepte el valor de cualquier vecino y poder compararlo con los demás. Al registro $m$ lo inicializamos con ceros.

Para calcular todos los movimientos de la vecindad, realizamos tres ciclos anidados: el primero $1 \leq k_0 \leq r$ representa a las clases paralelas, mientras que los otros dos: $1 \leq i_0 < v$ y $i_0 + 1 \leq i_1 \leq v$ representan a las variedades. Estos tres índices forman los atributos de nuestro movimiento a realizar, es decir, el movimiento que vamos a utilizar en la técnica de BT consiste en intercambiar la posición de dos variedades $i_0$ e $i_1$ de la misma clase paralela $k_0$, pero de diferente grupo, o sea, que $Xt[k_0][i_0] \neq Xt[k_0][i_1]$. Entonces los atributos de nuestros movimientos son las tripletas: $(k_0, i_0, i_1)$, las cuales representan a la clase paralela y a las dos variedades que intercambian de grupo respectivamente.

Luego para cada movimiento admisible, si realizaramos el movimiento sobre la matriz de solución, deberíamos volver a crear la matriz de concurrencia y calcular su costo aplicando el criterio de aspiración y de restricción tabú. Pero como cada vecino debe ser calculado a partir de la misma solución, entonces deberíamos guardar la solución con la que inicia la iteración en alguna matriz auxiliar y al terminar de evaluar al movimiento admisible, también deberíamos copiar toda la matriz auxiliar a la matriz de solución, y así sucesivamente hasta obtener el movimiento que produzca el mínimo en la vecindad. Una vez obtenido el mínimo de la vecindad, deberíamos actualizar la matriz de solución y volver a crear la matriz de concurrencia y calcular su costo para la siguiente iteración. Esto no es muy complicado, pero es muy ineficiente debido a que la mayoría de los cálculos realizados son innecesarios, porque con los intercambios realizados en los movimientos, únicamente se producen pequeños cambios en los resultados anteriores, probablemente para diseños con parámetros pequeños no parezca ser muy costoso, pero conforme crece el tamaño de los parámetros crece también el tiempo de cómputo de la solución, por ello es que tratamos de realizar un programa altamente eficiente en cuanto a la calidad de la solución y por

supuesto en cuanto al tiempo de cómputo utilizado.

En el programa desarrollado, los cálculos están optimizados para calcular únicamente los datos que fueron modificados durante el movimiento anterior, es decir, con la información ya obtenida se realizan los cálculos necesarios sólo sobre los elementos de la matriz de concurrencia que resultaron modificados. Esto resulta ser más barato y más eficiente que realizar el cálculo completo de las matrices para cada uno de los miembros de la vecindad. Con la introducción de una condición que permite únicamente aquellos movimientos cuyos costos sean menores e iguales que el mínimo alcanzado hasta el momento, el programa es más eficiente. Lo que estamos buscando es el movimiento que produzca el mínimo costo dentro de la vecindad y no es necesario proseguir con aquellos movimientos que sean mayores que el mínimo. Estos aspectos que mencionamos hacen al programa altamente eficiente como se verá en el siguiente capítulo, comparando los resultados con otros en la literatura.

Durante la búsqueda del mejor movimiento en la vecindad, elegimos de la clase paralela $k_0$ dos variedades: $i_0$ e $i_1$, que pertenezcan a diferentes grupos, y sin realizar el intercambio, consideramos como atributos del movimiento a los índices actuales $(k_0, i_0, i_1)$. Inicializamos la variable *sump* a cero y procedemos a optimizar el cálculo del costo de este movimiento: para $1 \leq i \leq v$, se verifica que los índices de las variedades $i_0$ e $i_1$ sean distintas de $i$, es decir, una variedad no puede aparecer junto a ella misma. Los únicos valores de la matriz de concurrencia que van a ser modificados son aquéllos de los renglones $i_0$ e $i_1$, pero únicamente aquellas columnas cuyas variedades pertenezcan al mismo grupo que $i_0$ e $i_1$.

Si $Xt[k_0][i_0] = Xt[k_0][i]$, entonces la variedad $i$ pertenece al mismo grupo que la variedad $i_0$, y el cambio que registra la matriz de concurrencia es el siguiente: aquellas variedades $i$ que aparecían junto a la variedad $i_0$ dejan de aparecer una vez con ella, pero aparecerán una vez más con la variedad $i_1$; por lo tanto debemos descontar a *sump* el costo que producía la variedad $i$ al aparecer junto a la variedad $i_0$, y agregarle el costo que produce ahora la variedad $i$ al aparecer junto a la variedad $i_1$. Así el valor de *sump* corresponde realmente al valor del movimiento, es decir, que tan bueno resulta ser el movimiento para mejorar la solución. De igual forma se procede si $Xt[k_0][i_1] = Xt[k_0][i]$.

Cabe señalar que con la matriz de concurrencia se deben ordenar los índices, el índice menor corresponde a los renglones y el mayor a las columnas, debido a que se está utilizando únicamente la parte superior de la matriz $Cij$. De igual forma para la matriz de frecuencia que es la parte inferior de $Cij$, el índice mayor corresponde a los renglones y el menor a las columnas. Además no se modifican las matrices de solución ni de concurrencia cuando se revisa a los vecinos.

A la variable *costot* le asignamos el costo actual con el que inicia la búsqueda del mejor movimiento en la vecindad *costop* y le sumamos el valor del movimiento *sump*. Si el costo de este movimiento es menor o igual que el *minimo* encontrado hasta el momento dentro de la vecindad, aplicamos el criterio de aspiración: si el costo de este movimiento *costot* es menor que el *mejor* movimiento encontrado hasta el momento, no importando si es movimiento tabú, aceptamos de inmediato este movimiento, el cual se guarda en el registro $m$, y al *minimo* y *minimop* les asignamos el *costot*, y continuamos analizando a los vecinos. Si el costo del movimiento *costot* es mayor o igual que el *mejor* movimiento, penalizamos al movimiento, es decir, si dos variedades ya se han intercambiado muchas veces, la penalización va a ser alta, de lo contrario será muy baja; esto nos permite diversificar la búsqueda a regiones no visitadas anteriormente.

La penalización resulta ser el costo del movimiento más la frecuencia con la que dos variedades se han intercambiado. Ya obtenida la penalización *pena*, se verifica si el movimiento es tabú llamando a la función estabu($k_0, i_0, i_1$); si es un movimiento tabú, no se toma en cuenta y continúa la búsqueda. De otra forma si no es un movimiento tabú y el costo penalizado es menor o igual que el mínimo penalizado *minimop*, entonces aceptamos el movimiento y lo guardamos en el registro $m$, al *minimo* le asignamos el costo del movimiento y al *minimop* le asignamos el costo penalizado. Esto se repite hasta revisar a todos los vecinos.

Terminado el proceso de búsqueda del mínimo movimiento en una iteración, se efectúa el cambio del movimiento en la matriz de solución realizando el cambio entre ambas variedades $i_0$ e $i_1$ del conjunto $k_0$. También se modifica la matriz de concurrencia para que queden actualizadas con el mínimo movimiento encontrado y continuar la siguiente iteración. Actualizamos la matriz de concurrencia con el movimiento seleccionado: para $1 \leq i \leq v$, verificar que los índices del movimiento $i_0$ e $i_1$ sean distintos

de $i$. Los únicos valores de la matriz de concurrencia que van a ser modificados son aquéllos de los renglones $i_0$ e $i_1$, pero únicamente aquellas columnas cuyas variedades pertenezcan al mismo grupo que $i_0$ e $i_1$.

Si $Xt[k_0][i_0] = Xt[k_0][i]$, entonces la variedad $i$ pertenece al mismo grupo que la variedad $i_0$, y el cambio que registra la matriz de concurrencia es el siguiente: aquellas variedades $i$ que aparecen junto a la variedad $i_0$, aparecen una vez más con ella, pero dejan de aparecer una vez con la variedad $i_1$, esto es debido a que ya se efectuó el movimiento en la matriz de solución $Xt$; por lo tanto debemos agregar a $Cij[i_1][i]$ una unidad para $i_1 < i$, y descontar a $Cij[i_0][i]$ una unidad para $i_0 < i$. De igual forma se procede si $Xt[k_0][i_1] = Xt[k_0][i]$. De esta manera se tienen actualizadas las matrices de solución y de concurrencia de acuerdo al mejor movimiento encontrado en la vecindad. Al *costop* le asignamos el *minimo* encontrado en la actual iteración; debemos actualizar también lo siguiente: la lista tabú, la matriz de frecuencia y la función de aspiración para iniciar con la siguiente iteración.

Actualizamos a la lista tabú aplicando la operación módulo sobre el número de iteraciones y el tamaño de la lista tabú, tomando el resultado de esta operación como el índice de la lista en el que se insertan los atributos del movimiento: tabu[nits *mod* ntabu]=m. La matriz de frecuencia también es actualizada agregando una unidad a la entrada correspondiente a las dos variedades intercambiadas. $Cij[i_0][i_1] + +$ para $i_0 > i_1$.

La función aspiración se actualiza únicamente cuando el mínimo encontrado en una vecindad sea menor que el mejor movimiento encontrado hasta el momento *mejor* = *minimo*; y a la variable *iter* le asignamos el número de iteraciones *nits*, el cual nos indica el número de iteraciones en el que no se ha registrado ningún movimiento que mejore al *mejor* movimiento encontrado.

*main*(*argc*, *argv*). Nos permite dar los parámetros necesarios en la línea de comandos para iniciar la ejecución del programa, verificando que no falten o sobren parámetros. Lo primero que se hace es llamar a la función *srand*((*unsigned*)*time*(*NULL*)), la cual inicializa al generador de números aleatorios tomando como semilla el resultado de la función *time* que nos proporciona el total de segundos transcurridos desde el 1° de Enero de 1970. Luego los parámetros dados en la línea de comandos se asignan a sus respectivas variables: archivo de lectura, iniciales del archivo de salida, número de instancias a realizar, número de estadísticas por cada instancia, máximo

número de iteraciones en que no se registre un mejor movimiento y el tamaño de la lista tabú a considerar; dejando desde el inicio abierto el archivo de lectura.

Se realizan dos ciclos anidados, el primero corresponde al número de instancias y el segundo al número de estadísticas. Dentro del primer ciclo se hace la llamada al procedimiento de lectura de los datos Lee_Datos(Entrada). Luego para cada instancia se abre un archivo de salida utilizando sus iniciales y el número de instancia que le corresponde, y se realiza el segundo ciclo, llamando al procedimiento init() para generar la solución aleatoria inicial, enseguida se llama al procedimiento Tabu() tomando el tiempo en segundos que tarda en ser ejecutado.

Al término del ciclo de las estadísticas, se obtienen los promedios correspondientes al número de iteraciones y al tiempo promedio únicamente de aquéllas que alcanzaron el óptimo de la función objetivo, cerrando el archivo correspondiente a cada instancia, y así sucesivamente hasta terminar con todas las instancias.

**Funciones**

*Costo*(). Esta función regresa el valor de la función objetivo asociado a la solución aleatoria inicial. Construida ya la matriz de concurrencia y conociendo el valor promedio de veces que dos variedades aparecen juntas: *lambda*, calculamos para cada entrada de la matriz de concurrencia el cuadrado de la diferencia entre el valor de la entrada $Cij[i][j]$ y el valor de *lambda*, el cual se va sumando en una variable y el resultado de la función resulta ser el valor total de la suma, que representa el costo del conjunto ordenado de bloques.

*estabu*$(i, j, k)$. Esta función regresa cierto o falso (1 ó 0). Se verifica si los atributos $i$, $j$ y $k$ del movimiento realizado se encuentran en la lista tabú, es decir, si es un movimiento tabú, entonces la función regresa como resultado 1, en caso contrario, regresa como resultado 0.

### 4.1.3 Búsqueda Tabú en paralelo

La tarea de diseñar algoritmos para procesamiento en paralelo presenta retos que son considerablemente más difíciles que aquéllos encontrados en el

dominio secuencial. Algo que está recibiendo una gran atención en la optimización combinatoria, es precisamente el desarrollo de algoritmos para procesamiento paralelo. Por ello nos enfocamos en la arquitectura de computadoras *MPP* (Massively Parallel Processing), con *MIMD* (Multiple Instruction, Multiple Data) [19], lo cual indica que existen múltiples flujos de control, de esta forma, cada procesador realiza simultáneamente diferentes tipos de instrucciones posiblemente sobre diferentes datos. Para intercambiar información entre dos procesadores que trabajan asíncronamente, alguna operación de paso de mensajes y sincronización deben ser realizadas. Los procesadores en un sistema de memoria distribuida no tienen acceso directo a la memoria de los demás procesadores. Para utilizar múltiples procesadores para una tarea, es necesario intercambiar la información entre los procesadores enviando paquetes de datos (*mensajes*) entre ellos a través de la red de comunicación disponible. Las bibliotecas de software para facilitar tales intercambios de datos se conocen como ambientes de paso de mensajes.

El paso de mensajes *MPP* consiste de $P$ programas secuenciales, uno corriendo en cada procesador. Cada programa secuencial utiliza las instrucciones del paso de mensajes para sincronizarse y accesar la memoria de otro programa, entonces el ambiente de programación consiste de un lenguaje de programación secuencial tal como *C* o *Fortran* que no tienen conceptos de paralelismo, acompañados por una biblioteca de paso de mensajes o comunicaciones primitivas de bajo nivel. El sistema de paso de mensajes utilizado en la *CM-5* es el de *Thinking Machines Corporation*, conocido como *CMMD* [29], que además de la comunicación estándar *send/receive*, soporta otros modos de comunicación, como el basado en *canales*, en el que una vez establecida la comunicación al realizar llamadas repetidas en el mismo canal, tiene un costo muy bajo; también han reducido el tiempo de comunicación de los mensajes con mecanismos de comunicación a muy bajo costo llamados *mensajes activos* también soportados por *CMMD*. Por ejemplo, para establecer la comunicación de un mensaje con la comunicación estándar send/receive, tarda aproximadamente 35 $\mu s$ (microsegundos), pero sólo 14 $\mu s$ con los canales de comunicación.

Una tarea en paralelo consiste de dos partes: un proceso controlador (*host*) y un conjunto de procesos nodo (*node*). El proceso *controlador* se ejecuta sobre la partición administradora de la *CM-5*, mientras que los procesos *nodo*, aunque ejecutan procesos idénticos, llevan flujos de control indepen-

dientes. El proceso controlador y los procesos nodo, son administrados por el sistema operativo *CMOST*. La forma más común de paso de mensajes, pero la más costosa utilizada en las máquinas de procesamiento en paralelo, es el protocolo send/receive, su costo se debe al protocolo requerido antes de la transmisión de datos, esto implica mandar una petición de envío seguida por una respuesta de aceptación por parte del receptor. Una vez que ambos procesos están de acuerdo, se realiza la transferencia de datos.

*CMMD_send_block*() y *CMMD_receive_block*() son utilizadas cuando se desea evitar hacer copias innecesarias de los datos o cuando es ventajoso utilizar los efectos de la sincronización de la comunicación.

En técnicas de búsqueda iterativas, y en particular en la técnica de BT, se pueden aplicar algunos tipos de paralelismo [8].

Primero, la búsqueda del siguiente movimiento a realizar puede ser paralelizada, generalmente esto requiere hacer una *partición* del conjunto de movimientos admisibles en $p$ subconjuntos, cada uno de ellos será asignado a un procesador el cual calcula su mejor movimiento. El mejor de todos esos $p$ movimientos es determinado y es el que debe aplicarse a la solución actual de todos los procesadores. Esta técnica requiere de una amplia comunicación ya que una sincronización es requerida en cada paso. Por lo tanto este es el precio aplicado a problemas en los cuales la búsqueda del mejor movimiento es relativamente complejo y consumidor de mucho tiempo.

Otro tipo de paralelismo que podríamos considerar para el problema de diseños resolubles con 2 concurrencias, es particionarlo de la siguiente manera: a cada procesador le podemos asignar una de las $r$ clases paralelas y que calcule el mejor movimiento de esa subvecindad, el caso ideal es que $r = \#nodos$; pero si $r > \#nodos$ se tendrían que repartir las $r$ clases paralelas entre el $\#nodos$, lo cual significaría que algunos procesadores trabajarían mucho más que los otros, o si $r < \#nodos$; tendríamos que algunos procesadores no tendrían nada que hacer. Además en este caso también necesitamos de la sincronización en cada paso. Por ello el primer tipo de paralelismo nos pareció más conveniente.

Aplicamos el primer tipo de paralelismo mencionado a la técnica de BT, es decir, paralelizando la búsqueda del siguiente movimiento dentro de una vecindad. La forma en que se planteó esta paralelización es muy sencilla,

utilizando el modelo controlador-nodo (host-node) de la CM-5, necesitamos dos programas, uno para el controlador y otro para los nodos.

El hecho de utilizar el lenguaje de programación *C*, nos facilitó el poder adaptar el programa secuencial en uno en paralelo, debido a que la CM-5 cuenta únicamente con los compiladores de C y Fortran, además del CMFortran y el C* que son lenguajes de tipo vectorial; pero utilizando las bibliotecas de paso de mensajes CMMD, se paralelizó el programa.

Los procedimientos y funciones utilizados en el programa secuencial son los mismos, excepto que algunos de ellos sufrieron algunas modificaciones, como fue el procedimiento Tabu() y el procedimiento main(), para el programa del controlador y de los nodos. El programa para el ***controlador***, cuenta con los siguientes procedimientos: Lee_Datos(), cambiomatriz(), despsol(), init(), Tabu(), main() y con la función Costo(). Mientras que el programa para los nodos cuenta con los procedimientos: cambiomatriz(), Tabu(), main() y con la función estabu().

### Controlador de procesos

En el programa para el controlador, se introdujo una nueva variable global *nprocs* que representa al número de procesadores con el que cuenta la computadora. En el procedimiento Tabu(), se utilizan las siguientes variables adicionales localmente:

- **el_li, elements[32]** son registros con 4 elementos: los tres atributos de un movimiento y el valor mínimo de ese movimiento.
- **Minimoprocs[32]** es un arreglo de enteros para almacenar el valor mínimo encontrado por cada procesador.
- **MinimoT** es el mínimo (de la vecindad) de todos los procesadores.
- **iMinimo** representa al número del procesador que encontró al mínimo en la vecindad.

*Tabu*(). Se inicia el ciclo iterativo hasta encontrar la solución óptima o llegar al límite de iteraciones permitidas en que no ocurra ningún mejor movimiento. Pero el controlador es el que se va a encargar de administrar el trabajo de los demás, el no realiza la búsqueda de los vecinos (ya que ésta la realizan los nodos paralelamente), sino que espera a recibir de todos los

nodos el registro que contiene a los atributos del movimiento y el mínimo encontrado por cada procesador, y lo guarda en el arreglo *elements*, calculando después el mínimo local y obteniendo el número de procesador que lo encontró, para después hacer la asignación del mínimo local al registro *el_li*. Se continúa haciendo lo mismo que en el programa secuencial después de terminado el ciclo de búsqueda del mínimo movimiento en la vecindad actual, excepto actualizar la lista tabú ni actualizar la matriz de concurrencia, ya que no es necesario aquí. Finalmente, después de actualizar la función aspiración se envía a cada procesador el registro *el_li* que contiene los atributos del movimiento que resultó ser el mejor movimiento de la vecindad actual, con el fin de que cada uno actualice sus datos y todos tengan la misma información para proceder con la siguiente iteración, sincronizándose con los nodos.

*main*(). Este procedimiento también sufrió algunas modificaciones. Antes de iniciar los ciclos anidados, se obtiene el número de procesadores de la computadora y se habilita el controlador. Se hace lo mismo que en el caso secuencial hasta el segundo ciclo correspondiente a las estadísticas de cada instancia, después de llamar al procedimiento init(), se envía a cada uno de los procesadores la siguiente información: la matriz de solución Xt y el costo inicial *costot*, los parámetros del diseño $(v, b, r, k, \lambda)$, el valor óptimo de la función *fmin*, el máximo número de iteraciones en que no mejore la solución *itera*, el tamaño de la lista tabú *ntabu*, el número de instancias a procesar *prueba* y el número de estadísticas a realizar por cada instancia *estad*, sincronizándose con los nodos. Luego realiza el procedimiento Tabu(), el tiempo no lo calcula el controlador, sino que lo hacen los nodos, entonces el controlador recibe del nodo 0 el tiempo que tardó en realizar el procedimiento; después se realizan las estadísticas como en el caso secuencial.

### Nodos

En el programa para los nodos, también se introduce una variable global, *pn* que representa el número del procesador de cada nodo. En el procedimiento Tabu(), se utiliza una variable más localmente: *nv* que representa al número total de vecinos, y además se utiliza la siguiente estructura de datos:

- **el_li** es un registro con 4 elementos: los tres atributos de un movimiento y el valor mínimo de ese movimiento.

*Tabu*(). Al inicio de cada iteración se inicia a cero la variable *nv*. En el ciclo de la búsqueda del mínimo movimiento de la vecindad, cada vez que se eligen dos variedades de distintos grupos se incrementa en una unidad la variable *nv*, que indica que un vecino más ha sido visitado. Ahora bien para distribuir la carga a todos los procesadores y realicen la búsqueda del siguiente movimiento de la vecindad paralelamente, se hace lo siguiente: como cada procesador tiene una copia del programa de los nodos, y es independiente de los demás, entonces realizamos una operación muy sencilla para determinar qué procesador va a realizar el siguiente movimiento: si el número de vecinos *nv* módulo el número de procesadores *nprocs* es igual al número de procesador *pn*, entonces ese movimiento es revisado por el procesador, es decir, *nprocs* vecinos son revisados paralelamente; de esta forma cada procesador únicamente revisa un subconjunto de la vecindad. Una vez terminado el proceso de búsqueda de todos los vecinos, los tres atributos y el valor del mínimo movimiento encontrado por cada procesador está en su registro *el_li*, el cual debe ser enviado al controlador para que él determine cual de todos los mínimos encontrados es el mínimo de la vecindad y esperan a que éste les regrese esa información sincronizándose con el controlador.

*main*(). Antes de iniciar los ciclos anidados, todos los nodos obtienen su número de procesador y se habilitan con el controlador. En el segundo ciclo correspondiente a las estadísticas de cada instancia, cada nodo recibe del controlador la siguiente información: la matriz de solución Xt y el costo inicial *costot*, los parámetros del diseño $(v, b, r, k, \lambda)$, el valor óptimo de la función *fmin*, el máximo número de iteraciones en que no mejore la solución *itera*, el tamaño de la lista tabú *ntabu*, el número de instancias a procesar *prueba* y el número de estadísticas a realizar por cada instancia *estad*, sincronizándose con el controlador. Como los nodos no tienen el procedimiento init(), después de recibir la información inicializan a ceros la lista tabú, la matriz de concurrencia y la matriz de frecuencia, luego generan su propia matriz de concurrencia llamando al procedimiento cambiomatriz(). Utilizando unas funciones especiales de la biblioteca CMMD, cada nodo obtiene el tiempo de proceso del procedimiento Tabu(), pero únicamente es enviado al controlador por el nodo 0.

En el siguiente capítulo se muestran los resultados obtenidos al aplicar la Búsqueda Tabú a una serie de instancias del problema de la construcción de diseños de experimentos.

# Capítulo 5

# Resultados

Para cada una de las 317 instancias que se generaron de posibles diseños resolubles con 2 concurrencias para realizar pruebas, se realizaron 10 corridas para poder obtener una estadística sobre el número promedio de iteraciones y el tiempo promedio que se tardó en encontrar la solución, así como también el porcentaje de las 10 corridas en las que se encontró la solución. Cabe mencionar que los promedios se obtuvieron únicamente de aquellas corridas que alcanzaron el valor óptimo de la función objetivo. Sólo 76 de ellas, las cuales representan el 24% del total de instancias, alcanzaron el óptimo correspondiente a su función objetivo de acuerdo a la forma en que se planteó el problema de optimización combinatoria. Todas las corridas fueron realizadas con una solución inicial generada aleatoriamente.

Las 10 corridas para las 317 instancias se realizaron tanto de manera secuencial como en paralelo. El programa secuencial se corrió en una PC 486DX a 33 Mhz; mientras que el programa en paralelo, se corrió en una CM-5 con 32 procesadores. Los resultados obtenidos con ambos programas de la Búsqueda Tabú se presentan en las tablas 5.1, 5.2 y 5.3.

Para la instancia de Elenbogen y Maxim [7] con parámetros $(12, 24, 8, 4)$, el costo óptimo del diseño es 12, el cual se encontró en un 100% de las corridas realizadas, con un promedio de 95 iteraciones y un tiempo promedio de cómputo de 1.4890 segundos en la PC; mientras que en la CM-5 se obtuvo en un promedio de 136 iteraciones y un tiempo promedio de cómputo de 0.6539 segundos. El tamaño de la lista tabú utilizado fue de 20, ya que experimentalmente con este valor se obtuvieron mejores resultados que con

| | | | | | | | | | | PC | | CM5 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| # | $v$ | $b$ | $r$ | $k$ | $n_1$ | $n_2$ | $\lambda_1$ | $\lambda_2$ | $f^*$ | $it$ | $tm$ | $it$ | $tm$ |
| 1 | 9 | 15 | 5 | 3 | 6 | 2 | 1 | 2 | 9 | 4 | 0.0109 | 4 | 0.0206 |
| 3 | 15 | 25 | 5 | 3 | 4 | 10 | 0 | 1 | 30 | 10 | 0.1703 | 9 | 0.0461 |
| 4 | 18 | 30 | 5 | 3 | 7 | 10 | 0 | 1 | 63 | 8 | 0.2307 | 8 | 0.0408 |
| 5 | 21 | 35 | 5 | 3 | 10 | 10 | 0 | 1 | 105 | 8 | 0.3571 | 8 | 0.0473 |
| 6 | 24 | 40 | 5 | 3 | 13 | 10 | 0 | 1 | 120 | 7 | 0.5054 | 8 | 0.0490 |
| 7 | 27 | 45 | 5 | 3 | 16 | 10 | 0 | 1 | 135 | 8 | 0.7802 | 8 | 0.0531 |
| 8 | 30 | 50 | 5 | 3 | 19 | 10 | 0 | 1 | 150 | 8 | 1.0934 | 8 | 0.0580 |
| 9 | 9 | 18 | 6 | 3 | 4 | 4 | 1 | 2 | 18 | 4 | 0.0164 | 4 | 0.0212 |
| 11 | 15 | 30 | 6 | 3 | 2 | 12 | 0 | 1 | 15 | 94 | 2.0659 | 132 | 0.6314 |
| 12 | 18 | 36 | 6 | 3 | 5 | 12 | 0 | 1 | 45 | 13 | 0.4725 | 14 | 0.0722 |
| 13 | 21 | 42 | 6 | 3 | 8 | 12 | 0 | 1 | 84 | 12 | 0.6758 | 13 | 0.0712 |
| 14 | 24 | 48 | 6 | 3 | 11 | 12 | 0 | 1 | 132 | 12 | 0.9835 | 11 | 0.0714 |
| 15 | 9 | 21 | 7 | 3 | 2 | 6 | 1 | 2 | 9 | 7 | 0.0274 | 6 | 0.0292 |
| 16 | 12 | 28 | 7 | 3 | 8 | 3 | 1 | 2 | 18 | 9 | 0.1208 | 11 | 0.0555 |
| 17 | 18 | 42 | 7 | 3 | 3 | 14 | 0 | 1 | 27 | 49 | 2.1263 | 56 | 0.2997 |
| 18 | 21 | 49 | 7 | 3 | 6 | 14 | 0 | 1 | 63 | 18 | 1.1868 | 18 | 0.1063 |
| 19 | 12 | 32 | 8 | 3 | 6 | 5 | 1 | 2 | 30 | 8 | 0.1208 | 9 | 0.0441 |
| 20 | 15 | 40 | 8 | 3 | 12 | 2 | 1 | 2 | 15 | 39 | 1.1098 | 41 | 0.2279 |
| 22 | 9 | 27 | 9 | 3 | 6 | 2 | 2 | 3 | 9 | 8 | 0.0549 | 7 | 0.0359 |
| 23 | 12 | 36 | 9 | 3 | 4 | 7 | 1 | 2 | 24 | 10 | 0.1648 | 10 | 0.0506 |
| 24 | 15 | 45 | 9 | 3 | 10 | 4 | 1 | 2 | 30 | 15 | 0.4780 | 15 | 0.0827 |
| 25 | 9 | 30 | 10 | 3 | 4 | 4 | 2 | 3 | 18 | 6 | 0.0604 | 6 | 0.0319 |
| 26 | 12 | 40 | 10 | 3 | 2 | 9 | 1 | 2 | 12 | 28 | 0.5329 | 31 | 0.1512 |
| 27 | 15 | 50 | 10 | 3 | 8 | 6 | 1 | 2 | 45 | 14 | 0.5000 | 15 | 0.0810 |
| 28 | 9 | 33 | 11 | 3 | 2 | 6 | 2 | 3 | 9 | 8 | 0.0769 | 8 | 0.0410 |

**Tabla 5.1: Resultados de diseños resolubles con 2 concurrencias, con $ntabu$ = 20 y el porcentaje de las corridas que llegaron al óptimo es del 100%**

otros tamaños de la lista. Por ejemplo, con tamaños más pequeños de la lista y con un máximo de 500 iteraciones, la solución óptima no se obtenía siempre.

Comparando los resultados de nuestros experimentos con algunos resultados obtenidos en la literatura, para el problema mencionado, Elenbogen y Maxim [7] utilizando la técnica del recocido simulado no encontraron la solución óptima. Mientras que Morales [21] utilizando la técnica de la Búsqueda Tabú, encontró la solución en un 100% de 20 corridas, con un promedio de 170 iteraciones y un tiempo promedio de cómputo de 28 segundos. Con respecto a este último, la solución es la misma, pero en cuanto al tiempo de cómputo necesario para obtener la solución, los programas aquí presentados son más eficientes.

Esto resulta interesante porque así podemos realizar más experimentos con diseños de distintos parámetros en un tiempo relativamente reducido para probar su existencia. Utilizando el programa en paralelo de la Búsqueda Tabú en la CM-5 sería aún más rápido. Por ejemplo: la instancia #316, con parámetros (36,40,20,18), no aparece en las tablas porque en ninguna de las 10 corridas se obtuvo su valor óptimo de la función objetivo. Una corrida con un promedio de 1200 iteraciones con el programa secuencial tardó aproximadamente 17 minutos, mientras que el programa en paralelo tardó aproximadamente 17 segundos.

En las tablas podemos observar que cuando el número promedio de iteraciones en que se encontró la solución óptima es pequeño, el tiempo promedio que tardó en obtener la solución también es muy pequeño, y no se aprecia gran diferencia entre ambos programas. Pero cuando el número de iteraciones va creciendo, observamos que la diferencia es considerable, por ejemplo, en la instancia #88 podemos apreciar mejor la diferencia en cuanto al tiempo de cómputo. En la PC un promedio de 422 iteraciones se tardó aproximadamente 170 segundos; mientras que en la CM-5 un promedio de 516 iteraciones se tardó aproximadamente 6 segundos. Esto representa aproximadamente una relación de 1/35, es decir, por cada iteración que realiza la PC, la CM-5 realiza 35 iteraciones.

En la mayoría de las instancias en las que no se obtiene la solución óptima el 100% de las corridas, podemos observar que parece ser más difícil obtener la solución óptima cuando la diferencia entre los parámetros $n_1$ y $n_2$

| | | | | | | | | | | | PC | | CM5 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| # | $v$ | $b$ | $r$ | $k$ | $n_1$ | $n_2$ | $\lambda_1$ | $\lambda_2$ | $f^*$ | $it$ | $tm$ | $it$ | $tm$ |
| 29 | 12 | 48 | 12 | 3 | 9 | 2 | 2 | 3 | 12 | 16 | 0.3846 | 22 | 0.1343 |
| 30 | 9 | 39 | 13 | 3 | 6 | 2 | 3 | 4 | 9 | 9 | 0.0879 | 9 | 0.0418 |
| 31 | 9 | 42 | 14 | 3 | 4 | 4 | 3 | 4 | 18 | 8 | 0.0659 | 8 | 0.0397 |
| 32 | 9 | 45 | 15 | 3 | 2 | 6 | 3 | 4 | 9 | 11 | 0.1208 | 11 | 0.0523 |
| 33 | 9 | 51 | 17 | 3 | 6 | 2 | 4 | 5 | 9 | 10 | 0.1373 | 10 | 0.0479 |
| 36 | 20 | 25 | 5 | 4 | 4 | 15 | 0 | 1 | 40 | 102 | 4.4285 | 111 | 0.6224 |
| 37 | 24 | 30 | 5 | 4 | 8 | 15 | 0 | 1 | 96 | 29 | 2.1428 | 32 | 0.1897 |
| 38 | 28 | 35 | 5 | 4 | 12 | 15 | 0 | 1 | 168 | 19 | 2.1428 | 17 | 0.1119 |
| 39 | 32 | 40 | 5 | 4 | 16 | 15 | 0 | 1 | 240 | 17 | 2.8516 | 16 | 0.1203 |
| 40 | 36 | 45 | 5 | 4 | 20 | 15 | 0 | 1 | 270 | 16 | 3.7417 | 16 | 0.1482 |
| 41 | 40 | 50 | 5 | 4 | 24 | 15 | 0 | 1 | 300 | 16 | 5.2033 | 15 | 0.1603 |
| 42 | 8 | 12 | 6 | 4 | 3 | 4 | 2 | 3 | 12 | 4 | 0.0109 | 4 | 0.0203 |
| 44 | 16 | 24 | 6 | 4 | 12 | 3 | 1 | 2 | 24 | 30 | 0.8296 | 40 | 0.1998 |
| 47 | 28 | 42 | 6 | 4 | 9 | 18 | 0 | 1 | 126 | 43 | 5.9615 | 46 | 0.3104 |
| 48 | 32 | 48 | 6 | 4 | 13 | 18 | 0 | 1 | 208 | 29 | 5.8736 | 27 | 0.1995 |
| 50 | 16 | 28 | 7 | 4 | 9 | 6 | 1 | 2 | 48 | 22 | 0.6868 | 21 | 0.1245 |
| 51 | 20 | 35 | 7 | 4 | 17 | 2 | 1 | 2 | 20 | 498 | 30.3113 | 514 | 3.1293 |
| 54 | 8 | 16 | 8 | 4 | 4 | 3 | 3 | 4 | 12 | 4 | 0.0219 | 5 | 0.0232 |
| 55 | 12 | 24 | 8 | 4 | 9 | 2 | 2 | 3 | 12 | 95 | 1.4890 | 136 | 0.6539 |
| 56 | 16 | 32 | 8 | 4 | 6 | 9 | 1 | 2 | 48 | 28 | 1.0109 | 33 | 0.2108 |
| 57 | 20 | 40 | 8 | 4 | 14 | 5 | 1 | 2 | 50 | 129 | 8.9890 | 143 | 0.9017 |
| 60 | 12 | 27 | 9 | 4 | 6 | 5 | 2 | 3 | 30 | 16 | 0.2472 | 27 | 0.1319 |
| 61 | 16 | 36 | 9 | 4 | 3 | 12 | 1 | 2 | 24 | 701 | 28.5164 | – | —— |
| 62 | 20 | 45 | 9 | 4 | 11 | 8 | 1 | 2 | 80 | 37 | 2.8846 | 37 | 0.2396 |
| 64 | 12 | 30 | 10 | 4 | 3 | 8 | 2 | 3 | 18 | 83 | 1.6153 | 126 | 0.6326 |

Tabla 5.2: Resultados de diseños resolubles con 2 concurrencias, con $ntabu$ = 20 y el porcentaje de las corridas que llegaron al óptimo es del 100%, excepto para el diseño #51 en el que el porcentaje es del 30% en ambos casos y para el diseño #61 es del 30% en la PC y del 0% en la CM-5

es grande. Como es el caso de la instancia #61 donde $n_1 = 3$ y $n_2 = 12$, la cual no encontró la solución óptima con el programa en paralelo en la CM-5; mientras que en la PC con el programa secuencial obtuvo la solución en un 30%.

Los porcentajes de las corridas en la tabla 5.3 son del 100%, excepto para los siguientes diseños:

#67 es del 30% en la PC y del 20% en la CM-5.

#73 es del 40% en la PC y del 30% en la CM-5.

#88 es del 70% en la CM-5.

#91 es del 90% en la PC.

#97 es del 30% en la PC y en la CM-5.

#101 es del 60% en la PC y del 10% en la CM-5.

#110 es del 40% en la PC y del 30% en la CM-5.

#111 es del 10% en la PC y del 20% en la CM-5.

#118 es del 40% en la PC y en la CM-5.

#206 es del 90% en la CM-5.

| | | | | | | | | | | | PC | | CM5 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| # | $v$ | $b$ | $r$ | $k$ | $n_1$ | $n_2$ | $\lambda_1$ | $\lambda_2$ | $f^*$ | $it$ | $tm$ | $it$ | $tm$ |
| 65 | 20 | 50 | 10 | 4 | 8 | 11 | 1 | 2 | 80 | 38 | 3.2912 | 42 | 0.2840 |
| 67 | 16 | 44 | 11 | 4 | 12 | 3 | 2 | 3 | 24 | 773 | 38.5531 | 1155 | 7.0061 |
| 69 | 12 | 36 | 12 | 4 | 8 | 3 | 3 | 4 | 18 | 66 | 1.5384 | 111 | 0.5657 |
| 70 | 16 | 48 | 12 | 4 | 9 | 6 | 2 | 3 | 48 | 33 | 1.8406 | 38 | 0.2160 |
| 71 | 8 | 26 | 13 | 4 | 3 | 4 | 5 | 6 | 12 | 6 | 0.0219 | 8 | 0.0385 |
| 72 | 12 | 39 | 13 | 4 | 5 | 6 | 3 | 4 | 30 | 20 | 0.4890 | 25 | 0.1424 |
| 73 | 12 | 42 | 14 | 4 | 2 | 9 | 3 | 4 | 12 | 165 | 4.5467 | 218 | 1.0929 |
| 74 | 8 | 30 | 15 | 4 | 4 | 3 | 6 | 7 | 12 | 6 | 0.0494 | 9 | 0.0412 |
| 77 | 12 | 48 | 16 | 4 | 7 | 4 | 4 | 5 | 24 | 32 | 0.9725 | 35 | 0.2116 |
| 79 | 12 | 51 | 17 | 4 | 4 | 7 | 4 | 5 | 24 | 25 | 0.8461 | 27 | 0.1550 |
| 82 | 8 | 40 | 20 | 4 | 3 | 4 | 8 | 9 | 12 | 8 | 0.0879 | 12 | 0.0662 |
| 88 | 40 | 48 | 6 | 5 | 15 | 24 | 0 | 1 | 300 | 422 | 170.0879 | 516 | 5.8928 |
| 91 | 25 | 35 | 7 | 5 | 20 | 4 | 1 | 2 | 50 | 491 | 59.0781 | 479 | 3.4995 |
| 97 | 25 | 40 | 8 | 5 | 16 | 8 | 1 | 2 | 100 | 221 | 30.4761 | 687 | 5.3721 |
| 101 | 25 | 45 | 9 | 5 | 12 | 12 | 1 | 2 | 150 | 587 | 90.9707 | 1087 | 9.3017 |
| 110 | 15 | 36 | 12 | 5 | 8 | 6 | 3 | 4 | 45 | 282 | 12.8708 | 637 | 3.6574 |
| 111 | 20 | 48 | 12 | 5 | 9 | 10 | 2 | 3 | 90 | 69 | 7.3626 | 1038 | 7.3686 |
| 118 | 15 | 48 | 16 | 5 | 6 | 8 | 4 | 5 | 45 | 338 | 20.3571 | 430 | 2.6461 |
| 119 | 10 | 34 | 17 | 5 | 4 | 5 | 7 | 8 | 20 | 24 | 0.4175 | 20 | 0.0968 |
| 121 | 10 | 38 | 19 | 5 | 5 | 4 | 8 | 9 | 20 | 18 | 0.3736 | 17 | 0.1013 |
| 129 | 18 | 24 | 8 | 6 | 11 | 6 | 2 | 3 | 54 | 81 | 4.2692 | 85 | 0.5689 |
| 134 | 18 | 27 | 9 | 6 | 6 | 11 | 2 | 3 | 54 | 214 | 12.6593 | 221 | 1.3279 |
| 137 | 12 | 20 | 10 | 6 | 5 | 6 | 4 | 5 | 30 | 11 | 0.2087 | 13 | 0.0623 |
| 143 | 12 | 24 | 12 | 6 | 6 | 5 | 5 | 6 | 30 | 18 | 0.4011 | 22 | 0.1115 |
| 203 | 16 | 28 | 14 | 8 | 7 | 8 | 6 | 7 | 56 | 55 | 3.3516 | 61 | 0.3403 |
| 206 | 16 | 32 | 16 | 8 | 8 | 7 | 7 | 8 | 56 | 50 | 3.5494 | 87 | 0.5750 |

Tabla 5.3: Resultados de diseños resolubles con 2 concurrencias, con $ntabu$ = 20, excepto para los diseños #73 y #111 en la PC, con listas tabú de $ntabu$ = 7 y $ntabu$ = 22 respectivamente

# Capítulo 6

# Conclusiones

Debido a que no existen métodos generales para resolver el problema de la construcción de diseños resolubles con 2 concurrencias, este problema ha sido formulado como un problema de optimización combinatoria, y se ha aplicado la técnica heurística Búsqueda Tabú para construir 76 diseños de una lista de 317 posibles diseños resolubles con 2 concurrencias, de los cuales, de 48 según la literatura revisada, se desconocía su existencia y no están disponibles en los catálogos de diseños de experimentos de Clatworthy [5]. Por lo tanto esta técnica ha probado ser eficiente para la construcción de estos diseños y resulta ser importante porque se plantea una nueva forma de resolver el problema.

Debido a que muchas combinaciones de parámetros de los diseños PBIB/2 no existen, Jarrett [15] relajó las condiciones de los diseños parcialmente balanceados con $m$ clases asociadas para definir a los diseños con $m$ concurrencias. Un ejemplo de un diseño PBIB/2 es el de Elenbogen y Maxim [7], el cual no aparece en los catálogos de diseños de experimentos; pero aquí probamos su existencia como un diseño con 2 concurrencias. Una ventaja de este problema, es que una cota mínima para el valor de la función objetivo siempre es conocida, así que si al tratar de resolver el problema se alcanza esta cota en los experimentos, entonces la existencia de los diseños queda probada.

Desafortunadamente, el tiempo requerido de cómputo que se necesita para experimentar con algunos diseños con parámetros grandes y probar su existencia, es demasiado grande en una computadora personal; por ello se

utilizaron técnicas de paralelismo en un equipo de supercómputo, tal como la CM-5, para reducir el tiempo de cómputo. Además se implantó el algoritmo de la Búsqueda Tabú de tal forma que se optimizaran los cálculos de la función objetivo, lo cual hizo que el algoritmo fuera más eficiente.

De los resultados presentados pudimos observar en general que cuando la diferencia entre los parámetros $n_1$ y $n_2$ es grande, parece ser más difícil encontrar una solución óptima. Además cuando el número promedio de iteraciones para encontrar una solución es pequeño, no se aprecia gran diferencia en el tiempo entre ambos programas para encontrar una solución óptima; pero cuando el número de iteraciones aumenta, se aprecia una diferencia considerable en el tiempo de cómputo requerido para encontrar una solución óptima del programa en paralelo sobre el secuencial. Por ello, explotando las técnicas de paralelismo junto con las técnicas heurísticas, podemos tratar de resolver instancias con grandes parámetros.

Aunque el diseño de experimentos tiene mucho tiempo de ser estudiado, todavía es un tema de intensa investigación. Hay muchas instancias para las cuales se desconoce su existencia; pero explotando el paralelismo y técnicas más avanzadas de la Búsqueda Tabú, así como definir otro tipo de movimientos o restricciones, podrían utilizarse para obtener mejores resultados. Actualmente estamos estudiando de manera similar los diseños PBIB/2 y los diseños BIB resolubles y no resolubles [23].

# Apéndice A PROGRAMA SECUENCIAL

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <time.h>
#include <mem.h>
#include <dos.h>
#define mini(a,b) (a < b ? a : b)
#define Maxv 83
#define Maxr 40
#define Maxt 50

int v=0, b=0, r=0, k=0, g=0, lambda=0, itera=0,
int fmin=0, ntabu=0, costot=0, mejor=0;
int Xt[Maxr][Maxv], Cij[Maxv][Maxv];
int num=0, n1=0, n2=0, l1=0, l2=0, nits;

struct TABU { /* Lista Tabu */
 int i,j,k; } tabu[Maxt];

void Tiempo(t,F)
double t; FILE *F; {
 long x;
 if ((long)t < 60) fprintf(F,"%6.5f seg\n", t);
 else {
  x = (long)t/60;
  if ((long)t < 3600)
   fprintf(F,"%ld min %6.5f seg\n", x, t-(x*60));
  else
   fprintf(F,"%ld hrs %ld min %6.5f seg\n", x/60,
    x%60, t-(x*60));
  }
}

void Lee_Datos(File)
FILE *File; {
 int a=0, d=0;
 fscanf(File,"%4d", &num);
 fscanf(File,"%5d", &v);
 fscanf(File,"%5d", &b);
 fscanf(File,"%5d", &r);
 fscanf(File,"%5d\n", &k);
 if ((v*r) == (b*k)) {
  a=r*(k-1); d=v-1; n2=a%d;
  if (n2!=0) {
   n1=d-n2; l1=a/d; l2=l1+1;
   fmin=v*mini(n1,n2)/2;
   if (n1<n2) lambda=l2;
   else lambda=l1;
  }
  else {
   puts("El diseño es balanceado, no es parcial-
    mente balanceado");
   exit(1);
  }
  if ((v/k) != (b/r) || b%r != 0) {
   puts("El diseño no es resoluble");
   exit(1);
  }
  g=b/r;
 }
 else {
  puts("El diseño no cumple la condición nece-
   saria de un diseño: (v*r=b*k)");
  exit(1);
 }
}

int Costo() {
 int i, j, costo=0, p=0;
 for (i=1; i<v; i++)
  for (j=i+1; j<=v; j++) {
   p=(lambda-Cij[i][j]);
   costo+=p*p;
  }
 return costo;
}

void cambiomatriz() {
 int i,j,k;
 for(i=1; i<v; i++)
  for(j=i+1; j<=v; j++)
   Cij[i][j]=0;
 for (k=1; k<=r; k++)
  for (i=1; i<v; i++)
   for (j=i+1; j<=v; j++)
    if (Xt[k][i]==Xt[k][j])
     Cij[i][j]++;
}

 /* Despliega conjunto y solución actual */

void despsol(File, tiempo)
FILE *File; double tiempo; {
 int i, j, l, cont=0;
 fprintf(File,"\nPARAMETROS (v,b,r,k,g,n1,n2,l1,l2)
  =(%d,%d,%d,%d,%d,%d,%d,%d,%d,%d),
  fmin=%d\n",v,b,r,k,g,n1,n2,l1,l2,fmin);
 fprintf(File," Costo Itera Tabu Tiempo\n");
 fprintf(File,"%5d %5d %4d ",mejor,nits,ntabu);
 Tiempo(tiempo,File);
 fprintf(File,"\n");
 for (l=1; l<=r; l++) {
  for (i=1; i<=g; i++) {
   fprintf(File,"A%d%d = {",l,i);
   for (j=1; j<=v; j++)
    if (Xt[l][j]==i) {
     fprintf(File,"%3d",j);
     cont++;
     if (cont<k) fprintf(File,",");
     }
```

```
    fprintf(File,"] ");
    cont=0;
   }
  fprintf(File,"\n");
 }
 for (i=1; i<=v; i++) {
  for (j=1; j<=v; j++)
   fprintf(File,"%3d",Cij[i][j]);
   fprintf(File,"\n");
 }
}

void init() {
 int i, j, j0, grupo0;

/* Inicializa la lista tabú y ambas matrices */

 memset(tabu,0,sizeof(tabu));
 memset(Cij,0,Maxv*Maxv*sizeof(int));
 memset(Xt,0,Maxr*Maxv*sizeof(int));

/* Genera la solución inicial aleatoriamente*/

 for (i=1; i<=r; i++)
  for (j=1; j<=v; j++)
   Xt[i][j]=j;
 for (i=1; i<=r; i++)
  for (j=v; j>=1; j--) {
   j0=rand()%j; j0++; grupo0=Xt[i][j0];
   Xt[i][j0]=Xt[i][j]; Xt[i][j]=grupo0;
  }
 for (i=1; i<=r; i++)
  for (j=1; j<=v; j++) {
   Xt[i][j]%=g; Xt[i][j]++;
  }
 cambiomatriz();
 mejor=costot=Costo();
}

int estabu(i, j, k)
 int i, j, k; {
 int l;
 for (l=0; l<=ntabu; l++)
  if (tabu[l].i==i && tabu[l].j==j &&
   tabu[l].k==k) return 1;
 return 0;
}

void Tabu() {
 int p, pena, iter=0, minimo, minimop;
 int costop, sump, i, i0, i1, k0;
 TABU m;
 costop=costot; nits=0;

      /* Inicio del proceso iterativo */

 while (mejor > fmin && (nits-iter) < itera) {
  minimop=minimo=30000;
  nits++;
  m.i=m.j=m.k=0;

/* Búsqueda del mínimo movimiento en la vecindad
                actual */

  for (k0=1; k0<=r; k0++)
   for (i0=1; i0<v; i0++)
    for (i1=i0+1; i1<=v; i1++)
     if (Xt[k0][i0] != Xt[k0][i1]) {
      sump=0;
      for (i=1; i<=v; i++)
       if((i!=i0) && (i!=i1)) {
        if (Xt[k0][i0]==Xt[k0][i]) {
         if (i>i0) p=(lambda-Cij[i][i0][i]);
         else p=(lambda-Cij[i][i0]);
         sump-=p*p; p++;
         sump+=p*p;
         if (i>i1) p=(lambda-Cij[i1][i]);
         else p=(lambda-Cij[i][i1]);
         sump-=p*p; p--;
         sump+=p*p;
        }
        if (Xt[k0][i1]==Xt[k0][i]) {
         if (i>i1) p=(lambda-Cij[i1][i]);
         else p=(lambda-Cij[i][i1]);
         sump-=p*p; p++;
         sump+=p*p;
         if (i>i0) p=(lambda-Cij[i0][i]);
         else p=(lambda-Cij[i][i0]);
         sump-=p*p; p--;
         sump+=p*p;
        }
       }
      costot=costop+sump;
      if (costot<=minimo) {
       if (costot>=mejor) {
        pena=costot+Cij[i1][i0];
        if (!estabu(k0, i0, i1))
         if (pena<=minimop) {
          minimo=costot;
          minimop=pena;
          m.i=k0; m.j=i0; m.k=i1;
         }
        }
        else {
         minimop=minimo=costot;
         m.i=k0; m.j=i0; m.k=i1;
        }
       }
      }

  /* Fin del proceso de búsqueda del mínimo
  movimiento en una iteración. Cambia la solución
   actual por la mínima solución encontrada en la
                vecindad */

  p=Xt[m.i][m.j];
  Xt[m.i][m.j]=Xt[m.i][m.k];
  Xt[m.i][m.k]=p;
  for (i=1; i<=v; i++)
   if((i!=m.j) && (i!=m.k)) {
    if (Xt[m.i][m.j]==Xt[m.i][i]) {
```

```
      if (i>m.j) Cij[m.j][i]++;
      else Cij[i][m.j]++;
      if (i>m.k) Cij[m.k][i]--;
      else Cij[i][m.k]--;
     }
     if (Xt[m.i][m.k]==Xt[m.i][i]) {
      if (i>m.k) Cij[m.k][i]++;
      else Cij[i][m.k]++;
      if (i>m.j) Cij[m.j][i]--;
      else Cij[i][m.j]--;
     }
    }
   costop=minimo;

         /* Actualiza la lista tabú */

   tabu[nits%ntabu]=m;

    /* Actualiza la matriz de frecuencia */

   Cij[m.k][m.j]++;

    /* Actualiza la función de aspiración */

   if (costop<mejor) {
    mejor=costop;
    iter=nits;
   }
  } /* Fin del proceso iterativo */
}

void main (argc, argv)
 int argc; char *argv[]; {
 int i, j, prueba, estad, n=0;
 clock_t t_inicial, t_final;
 double tiempo_ej, tmprom=0, itprom=0;
 char file[13];
 FILE *File, *File1, *tabla;
 if (argc!=7) {
  puts("Datos incorrectos en línea de comandos!\a");
  puts("pbibd arch_entrada arch_salida instancias
    estad. iterac l_tabú");
  exit(1);
 }
 srand((unsigned) time(NULL));
 if ((File=fopen(argv[1],"r"))==NULL) {
  puts("Archivo inaccesible!\a");
  exit(1);
 }
 prueba=atoi(argv[3]);
 estad=atoi(argv[4]);
 itera=atoi(argv[5]);
 ntabu=atoi(argv[6]);
 for (j=1; j<=prueba; j++) {
  n=0; itprom=tmprom=0;
  Lee_Datos(File);
  sprintf(file,"%s%d.est",argv[2],num);
  tabla=fopen(file,"w");
  sprintf(file,"%s%d.sol",argv[2],num);
  File1=fopen(file,"w");
  printf("\nPBIBD (v,b,r,k,g,n1,n2,l1,l2)=
   (%d,%d,%d,%d,%d,%d,%d,%d,%d),
   fmin=%d\n",v,b,r,k,g,n1,n2,l1,l2,fmin);
  printf(" # Costo Itera Tabu Tiempo\n");
  fprintf(tabla,"\nPBIBD (v,b,r,k,g,n1,n2,l1,l2)=
   (%d,%d,%d,%d,%d,%d,%d,%d,%d),
   fmin=%d\n",v,b,r,k,g,n1,n2,l1,l2,fmin);
  fprintf(tabla," # Costo Itera Tabu Tiempo\n");
  for (i=1; i<=estad; i++) {
   init();
   t_inicial=clock();
   Tabu();
   t_final=clock();
   tiempo_ej=(double)(t_final-t_inicial)/
    (double)CLK_TCK;
   if (mejor==fmin) {
    n++;
    itprom+=(double)nits;
    tmprom+=(double)tiempo_ej;
    if (File1!=NULL) {
     despsol(File1,tiempo_ej);
     fclose(File1);
    }
    sound(2700);
    delay(100);
    nosound();
   }
   printf("%3d %5d %5d %4d ",i,mejor,nits,ntabu);
   Tiempo(tiempo_ej,stdout);
   fprintf(tabla,"%3d %5d %5d %4d ",
    ,mejor,nits,ntabu);
   Tiempo(tiempo_ej,tabla);
  }
  if (n!=0) {
   itprom/=(double)n;
   tmprom/=(double)n;
   printf("\nPro %5d %5.0f",fmin,itprom);
   Tiempo(tmprom,stdout);
   printf("%d %% llegó a la solución",(n*100)/estad);
   fprintf(tabla,"\nPro %5d %5.0f",fmin,itprom);
   Tiempo(tmprom,tabla);
   fprintf(tabla,"%d %% llegó a la solución",
    (n*100)/estad);
  }
  fclose(tabla);
 }
 fclose(File);
 sound(2700);
 delay(1000);
 nosound();
}
```

# Apéndice B PROGRAMA EN PARALELO

## Programa para el controlador de procesos

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <cm/cmmd.h>
#define mini(a,b) (a < b ? a : b)
#define Maxv 83
#define Maxr 40
#define Maxt 50

int v=0, b=0, r=0, k=0, g=0, lambda=0, itera=0,
int fmin=0, ntabu=0, costot=0, mejor=0, num=0;
int Xt[Maxr][Maxv], Cij[Maxv][Maxv];
int n1=0, n2=0, l1=0, l2=0, nits, nprocs;
int ntest=1, nstatis=1

struct TABU { /* Lista Tabu */
  int i,j,k,min; };

    /* Los procedimientos Tiempo(t,F),
   Lee_Datos(finput), despsol(fsal,tiempo),
cambiomatriz(), init() y la función Costo(), son
     idénticas a los del caso secuencial */

void Tiempo(t,F)
double t; FILE *F; {
  ⋮
}

void Lee_Datos(File)
FILE *File; {
  ⋮
}

int Costo() {
  ⋮
}

void cambiomatriz() {
  ⋮
}

  /* Despliega conjunto y solución actual */

void despsol(File, tiempo)
FILE *File; double tiempo; {
  ⋮
}

void init() {
  int i, j, j0, grupo0;

/* Inicializa ambas matrices, aquí no se maneja la
                lista tabú */

  memset(Cij,0,Maxv*Maxv*sizeof(int));
  memset(Xt,0,Maxr*Maxv*sizeof(int));

 /* Genera la solución inicial aleatoriamente*/

  ⋮

}

void Tabu() {
 int p, costop, iter=0, MinimoT, iMinimo;
 int ni, aux, Minimoprocs[32];
 TABU el_li, elements[32];
 costop=costot; nits=0;

        /* Inicio del proceso iterativo */

 while (mejor > fmin && (nits-iter) < itera) {
  nits++;

/* Búsqueda del mínimo movimiento en la vecindad
                actual */

  for(ni=0; ni<nprocs; ni++)
   CMMD_receive_block(ni,CMMD_
    DEFAULT_TAG,&elements[ni],sizeof(el_li));
  MinimoT=elements[0].min;
  for(ni=0; ni<nprocs; ni++)
   if(MinimoT>=elements[ni].min) {
    MinimoT=elements[ni].min;
    iMinimo=ni;
   }
  el_li=elements[iMinimo];

  /* Fin del proceso de búsqueda del mínimo
movimiento en una iteración. Cambia la solución
 actual por la mínima solución encontrada en la
                vecindad */

  p=Xt[el_li.i][el_li.j];
  Xt[el_li.i][el_li.j]=Xt[el_li.i][el_li.k];
  Xt[el_li.i][el_li.k]=p;

   /* Actualiza la matriz de frecuencia */

  Cij[el_li.k][el_li.j]++;

   /* Actualiza la función de aspiración */
```

```
  if (el_li.min<mejor) {
   mejor=el_li.min;
   iter=nits;
  }
  for(ni=0; ni<nproca; ni++)
   CMMD_send_block(ni,CMMD_
    DEFAULT_TAG,&el_li,sizeof(el_li));
  CMMD_sync_host_with_nodes();
 } /* Fin del proceso iterativo */
}

void main (argc, argv)
 int argc; char *argv[]; {
 int i, j, n, ni;
 double timep, avtimep, aviter;
 FILE *fsal, *finput;
 if (argc!=7) {
  fprintf(stderr,"Tiene que ser %s file de
    entrada, file de salida\n",argv[0]);
  fprintf(stderr,"%s %s %s %s %s %s\n",argv[1],
   argv[2],argv[3],argv[4],argv[5],argv[6]);
  fprintf(stderr,"%d \n",argc);
  exit(1);
 }
 srand((unsigned) time(NULL));
 if ((finput=fopen(argv[1],"r"))==NULL) {
  fprintf(stderr,"No se puede abrir
   el archivo %s",argv[1]);
  exit(1);
 }
 ntest=atoi(argv[3]);
 nstatis=atoi(argv[4]);
 itera=atoi(argv[5]);
 ntabu=atoi(argv[6]);
 CMMD_enable();
 nproca=CMMD_partition_size();
 for (j=1; j<=ntest; j++) {
  n=aviter=0; avtimep=0.0;
  Lee_Datos(finput);
  fsal=fopen(argv[2],"a");
  fprintf(fsal,"V = %4d ",v);
  fprintf(fsal,"B = %4d ",b);
  fprintf(fsal,"R = %3d ",r);
  fprintf(fsal,"K = %3d ",k);
  fprintf(fsal,"Lam = %3d\n ",lambda);
  fprintf(fsal,"Statis Iter Best Timep\n",nits,mejor);
  for (i=1; i<=nstatis; i++) {
   init();
   for(ni=0; ni<nproca; ni++) {
    CMMD_send_block(ni,CMMD_DEFAULT_TAG,
     Xt,Maxr*Maxv*sizeof(int));
    CMMD_send_block(ni,CMMD_DEFAULT_TAG,
     &costot,sizeof(costot));
    CMMD_send_block(ni,CMMD_DEFAULT_TAG,
     &v,sizeof(v));
    CMMD_send_block(ni,CMMD_DEFAULT_TAG,
     &b,sizeof(b));
    CMMD_send_block(ni,CMMD_DEFAULT_TAG,
     &r,sizeof(r));
    CMMD_send_block(ni,CMMD_DEFAULT_TAG,
     &k,sizeof(k));
    CMMD_send_block(ni,CMMD_DEFAULT_TAG,
     &lambda,sizeof(lambda));
    CMMD_send_block(ni,CMMD_DEFAULT_TAG,
     &fmin,sizeof(fmin));
    CMMD_send_block(ni,CMMD_DEFAULT_TAG,
     &itera,sizeof(itera));
    CMMD_send_block(ni,CMMD_DEFAULT_TAG,
     &ntabu,sizeof(ntabu));
    CMMD_send_block(ni,CMMD_DEFAULT_TAG,
     &ntest,sizeof(ntest));
    CMMD_send_block(ni,CMMD_DEFAULT_TAG,
     &nstatis,sizeof(nstatis));
   }
   CMMD_sync_host_with_nodes();
   Tabu();
   CMMD_receive_block(0,0,&timep,sizeof(timep));
   fprintf(fsal,"%6d %6d %6ld " ,j,nits,mejor);
   Tiempo(timep,fsal);
   if (mejor==fmin) {
    n++;
    aviter+=nits;
    avtimep+=timep;
    if (File1!=NULL) {
     despsol(File1,tiempo_ej);
     fclose(File1);
    }
   }
  }
  if (n!=0) {
   aviter/=n;
   avtimep/=(double)n;
  }
  fprintf(fsal,"Success %4d %% aviter %4d",
   ((n*100)/nstatis),aviter);
  Tiempo(avtimep,fsal);
  fclose(fsal);
 }
 fclose(finput);
}
```

## Programa para los nodos

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <cm/cmmd.h>
#define Maxv 83
#define Maxr 40
#define Maxt 50

int v=0, b=0, r=0, k=0, g=0, lambda=0, itera=0;
int fmin=0, ntabu=0, costot=0, mejor=0, pn;
int Xt[Maxr][Maxv], Cij[Maxv][Maxv];
int nl=0, n2=0, l1=0, l2=0, ntest=1, nstatis=1;

struct TABU { /* Lista Tabu */
  int i,j,k,min; } tabu[Maxt];

/* El procedimiento cambiomatriz() y la función
estabu(i,j,k) son idénticos al del caso secuencial. */

void cambiomatriz() {
  .
  .
}

int estabu(i,j,k)
  int i,j,k; {
  .
  .
}

void Tabu() {
  TABU el_li;
  int p, minimo, costop, sump, minimop;
  int i, j, i0, i1, k0, pena, iter=0, nits=0;
  long nv;

  mejor=costop=costot;

        /* Inicio del proceso iterativo */

  while (mejor > fmin && (nits-iter) < itera) {
    minimop=minimo=30000;
    nits++; nv=0;

  /* La búsqueda del mínimo movimiento en la
vecindad actual, es similar al caso secuencial solo que
        se agrega una condición más. */

    for (k0=1; k0<=r; k0++)
     for (i0=1; i0<v; i0++)
      for (i1=i0+1; i1<=v; i1++)
       if (Xt[k0][i0] != Xt[k0][i1]) {
        nv++;
        if(nv%32==pn) {
         sump=0;
         .
         .
         costot=costop+sump;
         if (costot<=minimo) {
          if (costot>=mejor) {
           pena=costot+Cij[i1][i0];
           if (!estabu(k0, i0, i1))
            if (pena<=minimop) {
             minimo=costot;
             minimop=pena;
             el_li.i=k0; el_li.j=i0; el_li.k=i1;
            }
          }
          else {
           minimop=minimo=costot;
           el_li.i=k0; el_li.j=i0; el_li.k=i1;
          }
         }
        }
       }

/* Fin del proceso de búsqueda del mínimo
        movimiento en una iteración. */

el_li.min=minimop;
CMMD_send_block(CMMD_host_node(),
  CMMD_DEFAULT_TAG,&el_li,sizeof(el_li));
CMMD_receive_block(CMMD_host_node(),
  CMMD_DEFAULT_TAG,&el_li,sizeof(el_li));
CMMD_sync_with_host();
minimop=el_li.min;
if(minimop>=mejor)
  el_li.min-=Cij[el_li.k][el_li.j];
costop=el_li.min;

/* Cambia la solución actual por la mínima solución
        encontrada en la vecindad */

p=Xt[el_li.i][el_li.j];
Xt[el_li.i][el_li.j]=Xt[el_li.i][el_li.k];
Xt[el_li.i][el_li.k]=p;
for (i=1; i<=v; i++)
 if((i!=el_li.j) && (i!=el_li.k)) {
  if (Xt[el_li.i][el_li.j]==Xt[el_li.i][i]) {
   if (i>el_li.j) Cij[el_li.j][i]++;
   else Cij[i][el_li.j]++;
   if (i>el_li.k) Cij[el_li.k][i]--;
   else Cij[i][el_li.k]--;
  }
  if (Xt[el_li.i][el_li.k]==Xt[el_li.i][i]) {
   if (i>el_li.k) Cij[el_li.k][i]++;
   else Cij[i][el_li.k]++;
   if (i>el_li.j) Cij[el_li.j][i]--;
   else Cij[i][el_li.j]--;
  }
 }

        /* Actualiza la lista tabú */

tabu[nits%ntabu]=el_li;
```

```
    /* Actualiza la matriz de frecuencia */

   Cij[el_Ji.k][el_li.j]++;

    /* Actualiza la función de aspiración */

   if (costop<mejor) {
    mejor=costop;
    iter=nits;
   }
  } /* Fin del proceso iterativo */
}

void main() {
 int i,j;
 double timep;
 srand((unsigned) time(NULL));
 pn=CMMD_self_address();
 CMMD_enable_host();
 for(i=1; i<=ntest; i++) {
  for(j=1; j<=nstatis; j++) {
   CMMD_receive_block(CMMD_host_node(),
    CMMD_DEFAULT_TAG,
    Xt,MaxreMaxv*sizeof(int));
   CMMD_receive_block(CMMD_host_node(),
    CMMD_DEFAULT_TAG,
    &costot,sizeof(costot));
   CMMD_receive_block(CMMD_host_node(),
    CMMD_DEFAULT_TAG,
    &v,sizeof(v));
   CMMD_receive_block(CMMD_host_node(),
    CMMD_DEFAULT_TAG,
    &b,sizeof(b));
   CMMD_receive_block(CMMD_host_node(),
    CMMD_DEFAULT_TAG,
    &r,sizeof(r));
   CMMD_receive_block(CMMD_host_node(),
    CMMD_DEFAULT_TAG,
    &k,sizeof(k));
   CMMD_receive_block(CMMD_host_node(),
    CMMD_DEFAULT_TAG,
    &lambda,sizeof(lambda));
   CMMD_receive_block(CMMD_host_node(),
    CMMD_DEFAULT_TAG,
    &fmin,sizeof(fmin));
   CMMD_receive_block(CMMD_host_node(),
    CMMD_DEFAULT_TAG,
    &itera,sizeof(itera));
   CMMD_receive_block(CMMD_host_node(),
    CMMD_DEFAULT_TAG,
    &ntabu,sizeof(ntabu));
   CMMD_receive_block(CMMD_host_node(),
    CMMD_DEFAULT_TAG,
    &ntest,sizeof(ntest));
   CMMD_receive_block(CMMD_host_node(),
    CMMD_DEFAULT_TAG,
    &nstatis,sizeof(nstatis));
   CMMD_sync_with_host();
   memset(Cij,0,MaxvoMaxr*sizeof(int));
   memset(tabu,0,sizeof(tabu));
   cambiomatriz();
   CMMD_node_timer_clear(0);
   CMMD_node_timer_start(0);
   Tabu();
   CMMD_node_timer_stop(0);
   timep=CMMD_node_timer_busy(0);
   if(pn==0) CMMD_send_block(CMMD_host_node(),
    0,&timep,sizeof(timep));
  }
 }
```

- Nota: Donde aparecen los puntos suspensivos : el código faltante es exactamente igual que en el caso secuencial.

# Apéndice C DISEÑOS RESOLUBLES CON 2 CONCURRENCIAS

En este apéndice se muestran 48 diseños que no aparecen en las tablas de Clatworthy. El número # representa a los diseños correspondientes mostrados en las tablas de resultados. Donde los renglones corresponden a las clases paralelas y las columnas a los grupos de cada clase, donde cada grupo está formado por $k$ variedades separadas por comas(,).

# 14

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 4,9,14 | 16,21,24 | 1,13,22 | 3,12,23 | 6,7,20 | 10,11,15 | 8,18,19 | 2,5,17 |
| 1,7,15 | 13,18,20 | 2,6,16 | 9,19,23 | 5,8,22 | 12,14,24 | 3,10,21 | 4,11,17 |
| 8,11,24 | 15,16,18 | 1,3,9 | 4,7,12 | 5,21,23 | 10,17,20 | 2,13,19 | 6,14,22 |
| 6,12,21 | 15,22,24 | 2,7,8 | 4,13,23 | 1,10,14 | 9,17,18 | 5,11,20 | 3,16,19 |
| 1,12,19 | 11,18,21 | 16,20,22 | 9,13,15 | 5,6,10 | 3,8,14 | 2,4,24 | 7,17,23 |
| 1,17,24 | 7,14,16 | 2,10,18 | 9,12,22 | 3,11,13 | 15,20,21 | 4,5,19 | 6,8,23 |

# 206

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1,2,7,8,11,13,14,15 | 3,4,5,6,9,10,12,16 | 1,4,7,10 | 2,9,11,12 | 3,5,6,8 |
| 3,5,7,8,10,11,14,16 | 1,2,4,6,9,12,13,15 | 1,3,9,11 | 2,6,7,10 | 4,5,8,12 |
| 3,4,6,8,11,12,13,14 | 1,2,5,7,9,10,15,16 | 4,5,10,12 | 1,2,3,6 | 7,8,9,11 |
| 2,6,10,11,13,14,15,16 | 1,3,4,5,7,8,9,12 | 2,4,6,7 | 3,8,10,12 | 1,5,9,11 |
| 1,2,8,9,11,12,14,16 | 3,4,5,6,7,10,13,15 | 2,5,9,10 | 1,6,8,11 | 3,4,7,12 |
| 1,3,5,11,12,13,15,16 | 2,4,6,7,8,9,10,14 | 2,3,5,12 | 1,4,6,11 | 7,8,9,10 |
| 2,5,6,7,8,12,13,16 | 1,3,4,9,10,11,14,15 | 1,2,4,9 | 3,8,10,11 | 5,6,7,12 |
| 1,2,3,4,8,10,13,16 | 5,6,7,9,11,12,14,15 | 1,3,7,12 | 4,6,10,11 | 2,5,8,9 |
| 4,7,9,10,11,12,13,16 | 1,2,3,5,6,8,14,15 | 2,4,11,12 | 1,5,7,8 | 3,6,9,10 |
| 4,5,8,9,13,14,15,16 | 1,2,3,6,7,10,11,12 | 3,4,6,9 | 5,7,10,11 | 1,2,8,12 |
| 1,5,6,8,9,10,11,13 | 2,3,4,7,12,14,15,16 | 6,10,11,12 | 1,2,7,8 | 3,4,5,9 |
| 1,2,4,5,6,11,14,16 | 3,7,8,9,10,12,13,15 | 4,8,11,12 | 2,6,7,9 | 1,3,5,10 |
| 2,4,5,8,10,11,12,15 | 1,3,6,7,9,13,14,16 | 2,3,8,10 | 4,7,9,11 | 1,5,6,12 |
| 2,3,4,5,7,9,11,13 | 1,6,8,10,12,14,15,16 | 4,5,6,8 | 1,9,10,12 | 2,3,7,11 |
| 1,4,6,7,8,11,15,16 | 2,3,5,9,10,12,13,14 | 7,9,10,12 | 1,3,4,8 | 2,5,6,11 |
| 2,3,6,8,9,11,15,16 | 1,4,5,7,10,12,13,14 | 1,2,4,10 | 3,5,7,11 | 6,8,9,12 |

# 203

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 2,4,5,6,7,9,13,16 | 1,3,8,10,11,12,14,15 | 3,6,8,9 | 1,2,11,12 | 4,5,7,10 |
| 2,3,4,11,12,13,15,16 | 1,5,6,7,8,9,10,14 | 7,8,9,12 | 3,4,5,11 | 1,2,6,10 |
| 1,2,4,6,8,9,11,15 | 3,5,7,10,12,13,14,16 | 2,7,9,11 | 1,5,8,12 | 3,4,6,10 |
| 2,6,7,10,11,13,14,15 | 1,3,4,5,8,9,12,16 | 6,7,8,11 | 2,4,10,12 | 1,3,5,9 |
| 1,3,4,7,9,11,13,14 | 2,5,6,8,10,12,15,16 | 1,4,6,11 | 2,5,7,9 | 3,8,10,12 |
| 1,6,9,12,13,14,15,16 | 2,3,4,5,7,8,10,11 | 2,3,8,11 | 1,4,5,9 | 6,7,10,12 |
| 4,5,6,8,11,12,13,14 | 1,2,3,7,9,10,15,16 | 2,3,4,8 | 1,7,9,10 | 5,6,11,12 |
| 1,2,5,9,10,11,12,13 | 3,4,6,7,8,14,15,16 | 5,6,7,8 | 2,4,9,10 | 1,3,11,12 |
| 1,3,4,5,6,10,13,15 | 2,7,8,9,11,12,14,16 | 4,8,9,12 | 1,2,10,11 | 3,5,6,7 |
| 3,6,8,9,10,11,13,16 | 1,2,4,5,7,12,14,15 | 1,3,4,7 | 8,9,10,11 | 2,5,6,12 |
| 1,2,3,5,6,11,14,16 | 4,7,8,9,10,12,13,15 | 4,6,9,11 | 1,2,7,8 | 3,5,10,12 |
| 1,5,7,8,11,13,15,16 | 2,3,4,6,9,10,12,14 | 1,5,8,10 | 2,3,6,9 | 4,7,11,12 |
| 4,5,9,10,11,14,15,16 | 1,2,3,6,7,8,12,13 | 3,7,10,11 | 1,6,9,12 | 2,4,5,8 |
| 1,4,6,7,10,11,12,16 | 2,3,5,8,9,13,14,15 | 1,4,6,8 | 2,3,7,12 | 5,9,10,11 |

# 77

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1,4,7,10 | 2,9,11,12 | 3,5,6,8 |
| 1,3,9,11 | 2,6,7,10 | 4,5,8,12 |
| 4,5,10,12 | 1,2,3,6 | 7,8,9,11 |
| 2,4,6,7 | 3,8,10,12 | 1,5,9,11 |
| 2,5,9,10 | 1,6,8,11 | 3,4,7,12 |
| 2,3,5,12 | 1,4,6,11 | 7,8,9,10 |
| 1,2,4,9 | 3,8,10,11 | 5,6,7,12 |
| 1,3,7,12 | 4,6,10,11 | 2,5,8,9 |
| 2,4,11,12 | 1,5,7,8 | 3,6,9,10 |
| 3,4,6,9 | 5,7,10,11 | 1,2,8,12 |
| 6,10,11,12 | 1,2,7,8 | 3,4,5,9 |
| 4,8,11,12 | 2,6,7,9 | 1,3,5,10 |
| 2,3,8,10 | 4,7,9,11 | 1,5,6,12 |
| 4,5,6,8 | 1,9,10,12 | 2,3,7,11 |
| 7,9,10,12 | 1,3,4,8 | 2,5,6,11 |
| 1,2,4,10 | 3,5,7,11 | 6,8,9,12 |

# 73

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 3,6,8,9 | 1,2,11,12 | 4,5,7,10 |
| 7,8,9,12 | 3,4,5,11 | 1,2,6,10 |
| 2,7,9,11 | 1,5,8,12 | 3,4,6,10 |
| 6,7,8,11 | 2,4,10,12 | 1,3,5,9 |
| 1,4,6,11 | 2,5,7,9 | 3,8,10,12 |
| 2,3,8,11 | 1,4,5,9 | 6,7,10,12 |
| 2,3,4,8 | 1,7,9,10 | 5,6,11,12 |
| 5,6,7,8 | 2,4,9,10 | 1,3,11,12 |
| 4,8,9,12 | 1,2,10,11 | 3,5,6,7 |
| 1,2,3,4,7 | 8,9,10,11 | 2,5,6,12 |
| 4,6,9,11 | 1,2,7,8 | 3,5,10,12 |
| 1,5,8,10 | 2,3,6,9 | 4,7,11,12 |
| 3,7,10,11 | 1,6,9,12 | 2,4,5,8 |
| 1,4,6,8 | 2,3,7,12 | 5,9,10,11 |

# 38

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 4,6,16,26 | 1,5,8,19 | 2,9,10,22 | 3,17,21,27 | 7,11,13,23 | 12,15,18,28 | 14,20,24,25 |
| 3,5,12,25 | 1,11,18,20 | 14,16,17,22 | 10,19,23,24 | 2,8,21,26 | 6,13,15,27 | 4,7,9,28 |
| 9,23,25,26 | 10,15,17,20 | 2,16,27,28 | 5,13,18,21 | 1,3,7,24 | 6,8,12,14 | 4,11,19,22 |
| 4,12,23,27 | 8,18,22,24 | 2,7,19,20 | 1,10,16,25 | 5,6,11,17 | 3,13,26,28 | 9,14,15,21 |
| 1,21,23,28 | 2,4,13,17 | 7,10,14,27 | 12,20,22,26 | 3,8,9,11 | 5,15,16,24 | 6,18,19,25 |

| # 74 | | # 32 | | | # 31 | | |
|---|---|---|---|---|---|---|---|
| 2,5,7,8 | 1,3,4,6 | 5,7,8 | 1,2,3 | 4,6,9 | 7,8,9 | 1,4,5 | 2,3,6 |
| 2,3,4,6 | 1,5,7,8 | 5,6,7 | 2,4,9 | 1,3,8 | 4,7,8 | 3,5,9 | 1,2,6 |
| 1,3,4,5 | 2,6,7,8 | 2,4,6 | 3,5,9 | 1,7,8 | 1,6,8 | 2,4,5 | 3,7,9 |
| 2,4,6,8 | 1,3,5,7 | 1,2,5 | 4,8,9 | 3,6,7 | 1,7,9 | 3,4,6 | 2,5,8 |
| 2,3,6,7 | 1,4,5,8 | 5,6,7 | 1,2,9 | 3,4,8 | 1,3,8 | 5,6,9 | 2,4,7 |
| 3,4,6,8 | 1,2,5,7 | 2,4,7 | 3,5,9 | 1,6,8 | 2,8,9 | 1,3,4 | 5,6,7 |
| 3,4,7,8 | 1,2,5,6 | 1,4,7 | 2,5,8 | 3,6,9 | 4,6,9 | 3,7,8 | 1,2,5 |
| 1,2,3,8 | 4,5,6,7 | 3,4,7 | 2,6,8 | 1,5,9 | 4,5,8 | 3,6,7 | 1,2,9 |
| 2,3,4,5 | 1,6,7,8 | 1,3,4 | 7,8,9 | 2,5,6 | 2,3,5 | 1,6,7 | 4,8,9 |
| 2,4,5,7 | 1,3,6,8 | 2,3,7 | 1,6,9 | 4,5,8 | 1,2,8 | 5,6,7 | 3,4,9 |
| 1,2,3,7 | 4,5,6,8 | 4,7,8 | 5,6,9 | 1,2,3 | 1,3,5 | 2,7,9 | 4,6,8 |
| 3,4,7,8 | 1,2,5,6 | 2,8,9 | 3,4,6 | 1,5,7 | 2,3,6 | 5,8,9 | 1,4,7 |
| 3,5,6,8 | 1,2,4,7 | 3,6,8 | 2,4,5 | 1,7,9 | 2,6,8 | 1,4,9 | 3,5,7 |
| 1,2,4,8 | 3,5,6,7 | 3,5,8 | 1,4,6 | 2,7,9 | 5,6,9 | 1,3,8 | 2,4,7 |
| 1,4,6,7 | 2,3,5,8 | 2,6,8 | 3,7,9 | 1,4,5 | | | |

| # 71 | | # 30 | | | # 72 | | |
|---|---|---|---|---|---|---|---|
| 3,4,7,8 | 1,2,5,6 | 2,8,9 | 4,5,7 | 1,3,6 | 1,2,6,7 | 3,5,9,12 | 4,8,10,11 |
| 2,5,7,8 | 1,3,4,6 | 2,3,9 | 4,6,7 | 1,5,8 | 3,7,8,9 | 5,6,10,11 | 1,2,4,12 |
| 2,3,4,6 | 1,5,7,8 | 2,4,9 | 1,7,8 | 3,5,6 | 2,4,10,12 | 1,5,7,9 | 3,6,8,11 |
| 1,4,6,8 | 2,3,5,7 | 1,3,9 | 4,6,8 | 2,5,7 | 2,8,9,12 | 1,3,4,11 | 5,6,7,10 |
| 2,3,6,8 | 1,4,5,7 | 1,2,4 | 3,8,9 | 5,6,7 | 1,3,4,10 | 5,9,11,12 | 2,6,7,8 |
| 1,2,3,7 | 4,5,6,8 | 4,5,9 | 2,6,8 | 1,3,7 | 1,6,9,12 | 2,4,7,11 | 3,5,8,10 |
| 4,5,6,7 | 1,2,3,8 | 1,2,6 | 7,8,9 | 3,4,5 | 6,7,10,12 | 1,5,8,11 | 2,3,4,9 |
| 3,4,5,8 | 1,2,6,7 | 1,4,7 | 2,6,9 | 3,5,8 | 2,5,10,11 | 4,7,8,9 | 1,3,6,12 |
| 1,3,5,6 | 2,4,7,8 | 2,3,7 | 4,6,8 | 1,5,9 | 3,7,11,12 | 4,6,9,10 | 1,2,5,8 |
| 2,3,4,5 | 1,6,7,8 | 1,3,6 | 4,7,9 | 2,5,8 | 1,4,5,6 | 2,3,9,10 | 7,8,11,12 |
| 2,5,6,8 | 1,3,4,7 | 5,6,9 | 2,3,4 | 1,7,8 | 2,6,9,11 | 4,5,8,12 | 1,3,7,10 |
| 1,2,4,8 | 3,5,6,7 | 6,7,9 | 1,2,5 | 3,4,8 | 4,6,9,11 | 1,8,10,12 | 2,3,5,7 |
| 1,2,4,5 | 3,6,7,8 | 5,6,8 | 2,3,7 | 1,4,9 | 2,3,6,8 | 1,9,10,11 | 4,5,7,12 |

| # 110 | | | # 143 | |
|---|---|---|---|---|
| 1,4,8,10,15 | 2,5,7,11,14 | 3,6,9,12,13 | 2,4,6,8,10,11 | 1,3,5,7,9,12 |
| 4,7,9,11,12 | 2,3,10,13,15 | 1,5,6,8,14 | 3,6,7,8,9,10 | 1,2,4,5,11,12 |
| 3,4,10,12,14 | 1,5,11,13,15 | 2,6,7,8,9 | 1,4,5,6,7,10 | 2,3,8,9,11,12 |
| 3,5,8,13,14 | 4,6,7,9,15 | 1,2,10,11,12 | 1,2,3,7,10,11 | 4,5,6,8,9,12 |
| 2,4,6,13,14 | 3,7,8,10,11 | 1,5,9,12,15 | 1,2,4,7,8,9 | 3,5,6,10,11,12 |
| 2,6,11,14,15 | 3,5,7,9,10 | 1,4,8,12,13 | 2,4,7,9,10,12 | 1,3,5,6,8,11 |
| 1,6,7,10,13 | 2,4,5,8,12 | 3,9,11,14,15 | 5,7,8,10,11,12 | 1,2,3,4,6,9 |
| 1,2,3,7,15 | 4,6,9,11,13 | 5,8,10,12,14 | 3,4,6,7,11,12 | 1,2,5,8,9,10 |
| 7,12,13,14,15 | 4,5,6,10,11 | 1,2,3,8,9 | 1,6,9,10,11,12 | 2,3,4,5,7,8 |
| 5,9,10,11,13 | 1,3,4,7,14 | 2,6,8,12,15 | 1,2,6,7,8,12 | 3,4,5,9,10,11 |
| 8,9,10,14,15 | 1,3,6,11,12 | 2,4,5,7,13 | 1,4,7,8,9,11 | 2,3,5,6,10,12 |
| 7,8,11,12,13 | 1,2,9,10,14 | 3,4,5,6,15 | 2,5,6,7,9,11 | 1,3,4,8,10,12 |

| # 6 | | | | | | |
|---|---|---|---|---|---|---|
| 9,15,24 | 1,11,23 | 6,14,21 | 8,16,19 | 3,4,17 | 2,5,22 | 10,12,20 | 7,13,18 |
| 3,21,22 | 14,15,16 | 10,11,13 | 5,17,24 | 12,19,23 | 6,18,20 | 2,7,8 | 1,4,9 |
| 4,12,22 | 2,11,21 | 9,10,16 | 1,3,20 | 8,23,24 | 14,17,18 | 6,7,19 | 5,13,15 |
| 2,3,24 | 13,14,20 | 4,18,23 | 6,8,10 | 1,15,21 | 7,12,17 | 11,16,22 | 5,9,19 |
| 1,16,18 | 4,5,6 | 7,14,24 | 2,19,20 | 12,13,21 | 10,15,17 | 3,8,11 | 9,22,23 |

# 79

| | | |
|---|---|---|
| 1,4,7,12 | 2,5,9,11 | 3,6,8,10 |
| 1,3,8,12 | 2,5,7,10 | 4,6,9,11 |
| 3,7,8,10 | 5,6,11,12 | 1,2,4,9 |
| 1,3,5,11 | 2,4,8,12 | 6,7,9,10 |
| 5,6,7,8 | 2,9,11,12 | 1,3,4,10 |
| 3,6,9,12 | 2,4,5,10 | 1,7,8,11 |
| 2,3,6,8 | 4,5,10,12 | 1,7,9,11 |
| 4,5,9,11 | 2,3,7,8 | 1,9,10,12 |
| 4,6,7,10 | 2,3,5,12 | 1,8,9,11 |
| 5,8,11,12 | 1,2,6,7 | 3,4,9,10 |
| 3,5,7,12 | 4,6,8,9 | 1,2,10,11 |
| 6,10,11,12 | 2,4,7,8 | 1,3,5,9 |
| 1,2,3,6 | 4,7,11,12 | 5,8,9,10 |
| 2,3,4,11 | 5,6,8,9 | 1,7,10,12 |
| 8,9,10,12 | 3,4,7,11 | 1,2,5,6 |
| 2,8,10,11 | 3,5,7,9 | 1,4,6,12 |
| 2,7,9,12 | 3,6,10,11 | 1,4,5,8 |

# 33

| | | | |
|---|---|---|---|
| 5,6,9 | 1,7,8 | 2,3,4 | |
| 6,7,8 | 3,4,5 | 1,2,9 | |
| 3,6,8 | 1,5,7 | 2,4,9 | |
| 1,2,5 | 3,7,9 | 4,6,8 | |
| 1,3,8 | 2,6,7 | 4,5,9 | |
| 6,7,9 | 2,5,8 | 1,3,4 | |
| 1,8,9 | 2,4,7 | 3,5,6 | |
| 3,5,7 | 4,8,9 | 1,2,6 | |
| 2,6,7 | 1,5,8 | 3,4,9 | |
| 2,3,5 | 4,7,9 | 1,6,8 | |
| 1,4,6 | 3,7,8 | 2,5,9 | |
| 3,5,8 | 2,7,9 | 1,4,6 | |
| 2,3,6 | 1,4,7 | 5,8,9 | |
| 2,4,8 | 1,3,9 | 5,6,7 | |
| 1,3,7 | 2,8,9 | 4,5,6 | |
| 1,2,5 | 4,7,8 | 3,6,9 | |
| 1,6,9 | 4,5,7 | 2,3,8 | |

# 119

| | | |
|---|---|---|
| 2,4,6,9,10 | 1,3,5,7,8 | |
| 2,3,5,7,9 | 1,4,6,8,10 | |
| 1,3,7,9,10 | 2,4,5,6,8 | |
| 2,4,6,7,8 | 1,3,5,9,10 | |
| 3,4,6,7,9 | 1,2,5,8,10 | |
| 5,6,7,8,9 | 1,2,3,4,10 | |
| 1,4,7,8,10 | 2,3,5,6,9 | |
| 1,4,5,6,9 | 2,3,7,8,10 | |
| 2,5,6,7,10 | 1,3,4,8,9 | |
| 3,4,6,7,10 | 1,2,5,8,9 | |
| 3,5,6,8,10 | 1,2,4,7,9 | |
| 2,7,8,9,10 | 1,3,4,5,6 | |
| 4,5,7,9,10 | 1,2,3,6,8 | |
| 1,5,6,7,10 | 2,3,4,8,9 | |
| 1,2,3,6,7 | 4,5,8,9,10 | |
| 3,6,8,9,10 | 1,2,4,5,7 | |
| 1,6,7,8,9 | 2,3,4,5,10 | |

# 82

| | | |
|---|---|---|
| 1,3,4,7 | 2,5,6,8 | |
| 1,3,6,7 | 2,4,5,8 | |
| 1,2,7,8 | 3,4,5,6 | |
| 1,2,4,8 | 3,5,6,7 | |
| 2,4,7,8 | 1,3,5,6 | |
| 3,4,5,7 | 1,2,6,8 | |
| 2,3,7,8 | 1,4,5,6 | |
| 2,3,5,8 | 1,4,6,7 | |
| 1,5,7,8 | 2,3,4,6 | |
| 2,5,6,7 | 1,3,4,8 | |
| 4,6,7,8 | 1,2,3,5 | |
| 5,6,7,8 | 1,2,3,4 | |
| 1,2,5,6 | 3,4,7,8 | |
| 2,3,4,5 | 1,6,7,8 | |
| 4,5,6,8 | 1,2,3,7 | |
| 1,3,6,8 | 2,4,5,7 | |
| 1,2,4,6 | 3,5,7,8 | |
| 1,2,5,7 | 3,4,6,8 | |
| 1,4,5,7 | 2,3,6,8 | |
| 2,4,6,7 | 1,3,5,8 | |

# 121

| | | |
|---|---|---|
| 1,4,5,6,10 | 2,3,7,8,9 | |
| 1,2,4,6,10 | 3,5,7,8,9 | |
| 1,2,4,7,8 | 3,5,6,9,10 | |
| 1,3,4,8,9 | 2,5,6,7,10 | |
| 4,5,8,9,10 | 1,2,3,6,7 | |
| 1,2,3,5,10 | 4,6,7,8,9 | |
| 1,5,7,8,10 | 2,3,4,6,9 | |
| 6,7,8,9,10 | 1,2,3,4,5 | |
| 1,2,5,6,9 | 3,4,7,8,10 | |
| 1,3,6,9,10 | 2,4,5,7,8 | |
| 1,5,6,7,8 | 2,3,4,9,10 | |
| 1,4,7,9,10 | 2,3,5,6,8 | |
| 1,2,6,8,9 | 3,4,5,7,10 | |
| 1,2,3,8,10 | 4,5,6,7,9 | |
| 2,4,6,8,10 | 1,3,5,7,9 | |
| 1,2,7,9,10 | 3,4,5,6,8 | |
| 1,3,4,6,7 | 2,5,8,9,10 | |
| 2,3,6,7,10 | 1,4,5,8,9 | |
| 2,4,5,7,9 | 1,3,6,8,10 | |

# 118

| | | |
|---|---|---|
| 6,7,11,12,15 | 1,2,10,13,14 | 3,4,5,8,9 |
| 1,3,5,7,10 | 8,9,13,14,15 | 2,4,6,11,12 |
| 3,5,6,13,15 | 1,2,9,10,12 | 4,7,8,11,14 |
| 1,3,4,5,15 | 2,6,8,10,14 | 7,9,11,12,13 |
| 2,4,5,13,14 | 1,8,11,12,15 | 3,6,7,9,10 |
| 2,5,7,8,15 | 4,6,9,12,13 | 1,3,10,11,14 |
| 8,9,11,14,15 | 3,4,10,12,13 | 1,2,5,6,7 |
| 2,3,7,8,9 | 4,10,11,13,15 | 1,5,6,12,14 |
| 2,5,7,11,13 | 1,3,8,12,14 | 4,6,9,10,15 |
| 3,5,9,11,14 | 4,6,7,8,10 | 1,2,12,13,15 |
| 7,10,13,14,15 | 2,3,4,8,12 | 1,5,6,9,11 |
| 1,6,7,8,13 | 5,9,10,12,14 | 2,3,4,11,15 |
| 1,4,7,9,11 | 2,3,6,13,14 | 5,8,10,12,15 |
| 1,4,8,9,13 | 2,5,6,10,11 | 3,7,12,14,15 |
| 1,4,6,14,15 | 5,8,10,11,13 | 2,3,7,9,12 |
| 1,2,9,10,15 | 4,5,7,12,14 | 3,6,8,11,13 |

# 111

| | | |
|---|---|---|
| 3,4,13,16,18 | 1,10,11,14,15 | 2,7,9,17,20 |
| 6,12,13,17,18 | 9,10,11,14,16 | 1,3,5,19,20 |
| 3,12,13,14,20 | 1,2,9,18,19 | 8,11,15,16,17 |
| 2,8,11,13,20 | 1,7,10,16,19 | 5,9,12,15,18 |
| 1,2,4,6,11 | 5,8,14,17,18 | 9,10,13,19,20 |
| 4,8,9,16,19 | 1,5,7,13,17 | 3,10,11,12,18 |
| 6,7,15,18,19 | 1,3,9,11,17 | 4,8,10,12,20 |
| 2,3,15,17,19 | 1,6,8,10,13 | 7,9,12,14,16 |
| 6,11,13,16,19 | 1,4,9,12,14 | 5,10,15,17,20 |
| 6,7,9,11,20 | 2,3,5,10,16 | 4,14,17,18,19 |
| 1,15,16,18,20 | 3,5,6,8,9 | 7,11,13,14,19 |
| 7,8,10,14,18 | 3,4,9,13,15 | 2,5,11,12,19 |

# 69

| | | |
|---|---|---|
| 2,4,5,9 | 6,7,8,10 | 1,3,11,12 |
| 1,3,4,10 | 6,9,11,12 | 2,5,7,8 |
| 5,8,10,12 | 1,2,4,6 | 3,7,9,11 |
| 2,4,8,9 | 1,6,10,11 | 3,5,7,12 |
| 3,4,6,12 | 5,7,10,11 | 1,2,8,9 |
| 1,4,7,12 | 2,3,8,11 | 5,6,9,10 |
| 2,3,6,12 | 4,8,10,11 | 1,5,7,9 |
| 1,4,5,6 | 8,9,11,12 | 2,3,7,10 |
| 3,5,6,8 | 1,2,10,12 | 4,7,9,11 |
| 4,5,10,12 | 2,6,7,11 | 1,3,8,9 |
| 2,9,10,12 | 1,6,7,8 | 3,4,5,11 |
| 1,2,5,11 | 3,6,9,10 | 4,7,8,12 |

# 129

| | | |
|---|---|---|
| 5,8,9,10,11,17 | 3,6,12,15,16,18 | 1,2,4,7,13,14 |
| 1,5,6,10,13,18 | 2,7,9,15,16,17 | 3,4,8,11,12,14 |
| 4,5,14,16,17,18 | 1,7,8,10,12,15 | 2,3,6,9,11,13 |
| 4,5,6,7,11,15 | 9,10,12,13,14,16 | 1,2,3,8,17,18 |
| 3,5,7,12,13,17 | 2,10,11,14,15,18 | 1,4,6,8,9,16 |
| 2,4,6,10,12,17 | 7,8,11,13,16,18 | 1,3,5,9,14,15 |
| 1,3,10,11,16,17 | 6,7,9,12,14,18 | 2,4,5,8,13,15 |
| 1,2,5,11,12,16 | 6,8,13,14,15,17 | 3,4,7,9,10,18 |

# 55

| | | |
|---|---|---|
| 2,7,10,12 | 3,6,9,11 | 1,4,5,8 |
| 2,5,6,8 | 4,9,10,11 | 1,3,7,12 |
| 5,7,10,11 | 3,4,8,9 | 1,2,6,12 |
| 1,5,7,9 | 3,6,8,10 | 2,4,11,12 |
| 1,3,10,11 | 5,6,9,12 | 2,4,7,8 |
| 6,7,8,11 | 1,2,9,10 | 3,4,5,12 |
| 8,9,10,12 | 1,4,6,7 | 2,3,5,11 |
| 1,8,11,12 | 2,3,7,9 | 4,5,6,10 |

# 134

| | | |
|---|---|---|
| 4,6,8,10,16,18 | 1,2,3,12,14,17 | 5,7,9,11,13,15 |
| 8,9,12,13,14,18 | 1,2,6,7,10,11 | 3,4,5,15,16,17 |
| 4,7,9,10,14,17 | 2,3,5,6,13,18 | 1,8,11,12,15,16 |
| 3,4,8,13,14,15 | 6,7,11,12,17,18 | 1,2,5,9,10,16 |
| 2,4,5,7,8,12 | 1,10,13,15,17,18 | 3,6,9,11,14,16 |
| 1,4,5,11,14,16 | 3,7,10,12,13,16 | 2,6,8,9,15,17 |
| 2,9,10,11,12,18 | 5,8,13,14,16,17 | 1,3,4,6,7,15 |
| 5,6,10,12,14,15 | 1,3,7,8,9,18 | 2,4,11,13,16,17 |
| 2,7,14,15,16,18 | 3,5,8,10,11,17 | 1,4,6,9,12,13 |

# 60

| | | |
|---|---|---|
| 2,3,6,7 | 4,8,9,10 | 1,5,11,12 |
| 3,5,10,12 | 1,2,4,8 | 6,7,9,11 |
| 6,8,9,12 | 3,4,5,11 | 1,2,7,10 |
| 2,10,11,12 | 3,5,7,8 | 1,4,6,9 |
| 3,4,6,10 | 1,7,11,12 | 2,5,8,9 |
| 3,4,9,12 | 7,8,10,11 | 1,2,5,6 |
| 6,8,10,12 | 1,4,5,7 | 2,3,9,11 |
| 1,3,9,10 | 2,4,7,12 | 5,6,8,11 |
| 5,7,9,10 | 1,3,8,12 | 2,4,6,11 |

# 47

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 7,9,10,23 | 3,11,26,27 | 14,16,18,28 | 2,17,20,24 | 4,6,8,22 | 12,13,15,21 | 1,5,19,25 |
| 12,14,17,19 | 13,24,27,28 | 10,16,20,26 | 1,3,4,18 | 9,11,22,25 | 2,7,8,15 | 5,6,21,23 |
| 3,9,13,20 | 4,5,16,27 | 8,18,21,26 | 14,15,22,23 | 2,12,25,28 | 1,7,11,17 | 6,10,19,24 |
| 1,6,13,26 | 5,7,12,20 | 2,18,19,22 | 11,15,16,24 | 9,14,21,27 | 3,8,23,28 | 4,10,17,25 |
| 10,12,18,27 | 1,8,14,20 | 2,4,11,21 | 9,15,19,28 | 23,24,25,26 | 5,13,17,22 | 3,6,7,16 |
| 15,20,25,27 | 8,11,13,19 | 2,5,10,14 | 3,21,22,24 | 1,12,16,23 | 6,9,17,18 | 4,7,26,28 |

# 29

| | | | | |
|---|---|---|---|---|
| 2,3,9 | 1,5,10 | 6,7,8 | 4,11,12 | |
| 1,2,6 | 4,5,12 | 7,10,11 | 3,8,9 | |
| 1,7,12 | 3,6,11 | 4,9,10 | 2,5,8 | |
| 4,10,12 | 1,6,9 | 2,5,7 | 3,8,11 | |
| 1,5,9 | 3,7,10 | 4,6,11 | 2,8,12 | |
| 5,9,10 | 6,8,12 | 1,7,11 | 2,3,4 | |
| 3,5,6 | 2,4,7 | 9,11,12 | 1,8,10 | |
| 4,7,9 | 3,10,12 | 5,8,11 | 1,2,6 | |
| 5,6,12 | 7,8,9 | 1,3,4 | 2,10,11 | |
| 1,2,11 | 4,5,8 | 3,7,12 | 6,9,10 | |
| 6,10,11 | 1,4,8 | 3,5,7 | 2,9,12 | |
| 4,6,7 | 1,3,12 | 2,8,10 | 5,9,11 | |

# 28

| | | | | |
|---|---|---|---|---|
| 2,4,8 | 3,6,9 | 1,5,7 | | |
| 2,6,8 | 3,4,7 | 1,5,9 | | |
| 1,2,9 | 4,7,8 | 3,5,6 | | |
| 2,3,8 | 1,6,7 | 4,5,9 | | |
| 3,8,9 | 2,6,7 | 1,4,5 | | |
| 4,6,9 | 1,2,3 | 5,7,8 | | |
| 2,5,7 | 6,8,9 | 1,3,4 | | |
| 2,3,5 | 4,6,7 | 1,8,9 | | |
| 4,5,8 | 1,2,6 | 3,7,9 | | |
| 3,5,6 | 2,4,9 | 1,7,8 | | |
| 2,7,9 | 1,3,4 | 5,6,8 | | |

# 23

| | | | | |
|---|---|---|---|---|
| 8,10,12 | 2,5,9 | 3,6,11 | 1,4,7 | |
| 1,3,12 | 5,7,8 | 4,6,9 | 2,10,11 | |
| 4,6,10 | 1,8,9 | 5,11,12 | 2,3,7 | |
| 8,9,12 | 3,5,10 | 6,7,11 | 1,2,4 | |
| 2,10,12 | 3,7,9 | 4,8,11 | 1,5,6 | |
| 4,7,10 | 3,5,8 | 6,9,12 | 1,2,11 | |
| 7,9,11 | 2,6,8 | 1,5,10 | 3,4,12 | |
| 3,6,10 | 1,8,11 | 4,5,9 | 2,7,12 | |
| 3,4,11 | 5,7,12 | 2,6,8 | 1,9,10 | |

# 70

| | | | | |
|---|---|---|---|---|
| 1,3,8,11 | 4,5,6,13 | 2,10,15,16 | 7,9,12,14 | |
| 3,9,10,14 | 4,5,7,8 | 12,13,15,16 | 1,2,6,11 | |
| 4,7,9,15 | 1,3,12,16 | 6,11,13,14 | 2,5,6,10 | |
| 3,7,10,13 | 2,4,8,12 | 1,6,14,15 | 5,9,11,16 | |
| 1,4,13,14 | 3,5,12,15 | 6,7,10,11 | 2,8,9,16 | |
| 2,5,7,14 | 3,9,11,15 | 4,8,10,16 | 1,6,12,13 | |
| 2,7,11,12 | 1,5,9,10 | 4,6,14,16 | 3,8,13,15 | |
| 6,8,9,15 | 7,10,13,16 | 5,11,12,14 | 1,2,3,4 | |
| 2,3,14,16 | 1,9,10,13 | 6,7,8,12 | 4,5,11,15 | |
| 1,2,7,15 | 8,9,14,16 | 3,5,6,13 | 4,10,11,12 | |
| 6,11,15,16 | 2,9,12,13 | 3,4,7,14 | 1,5,8,10 | |
| 3,4,6,9 | 2,8,11,13 | 1,5,7,16 | 10,12,14,15 | |

# 67

| | | | |
|---|---|---|---|
| 7,10,12,13 | 4,5,11,15 | 1,2,8,14 | 3,6,9,16 |
| 2,8,13,16 | 1,4,5,9 | 3,7,11,12 | 6,10,14,15 |
| 1,10,12,14 | 2,3,4,7 | 6,11,13,16 | 5,8,9,15 |
| 5,6,7,14 | 1,3,15,16 | 2,9,10,11 | 4,8,12,13 |
| 2,4,7,9 | 1,5,6,12 | 10,13,15,16 | 3,8,11,14 |
| 5,7,12,16 | 3,8,9,10 | 1,2,6,15 | 4,11,13,14 |
| 6,7,8,11 | 4,12,14,15 | 1,3,9,13 | 2,5,10,16 |
| 2,6,8,12 | 3,5,10,14 | 1,4,11,16 | 7,9,13,15 |
| 4,6,8,10 | 1,3,5,13 | 2,7,14,16 | 9,11,12,15 |
| 1,7,8,15 | 3,4,6,10 | 2,5,11,13 | 9,12,14,16 |
| 6,9,13,14 | 4,5,8,16 | 1,7,10,11 | 2,3,12,15 |

# 37

| | | | | | | |
|---|---|---|---|---|---|---|
| 4, 7, 11, 13 | 3, 12, 18, 21 | 9, 22, 23, 24 | 5, 6, 8, 15 | 1, 16, 17, 19 | 2, 10, 14, 20 |
| 10, 16, 18, 23 | 1, 5, 20, 21 | 3, 6, 13, 17 | 2, 7, 12, 22 | 8, 9, 11, 14 | 4, 15, 19, 24 |
| 4, 17, 21, 23 | 5, 18, 19, 22 | 1, 9, 10, 13 | 11, 12, 15, 20 | 2, 3, 8, 24 | 6, 7, 14, 16 |
| 1, 3, 4, 14 | 8, 12, 19, 23 | 7, 18, 20, 24 | 6, 10, 11, 22 | 13, 15, 16, 21 | 2, 5, 9, 17 |
| 6, 9, 19, 21 | 3, 5, 11, 23 | 12, 13, 14, 24 | 4, 16, 20, 22 | 7, 8, 10, 17 | 1, 2, 15, 18 |

# 44

| | | | | |
|---|---|---|---|---|
| 2, 8, 13, 16 | 1, 3, 5, 6 | 4, 10, 11, 12 | 7, 9, 14, 15 | |
| 7, 11, 12, 13 | 3, 4, 8, 15 | 5, 6, 10, 14 | 1, 2, 9, 16 | |
| 3, 12, 14, 16 | 1, 4, 9, 11 | 6, 7, 8, 10 | 2, 5, 13, 15 | |
| 1, 4, 13, 14 | 6, 11, 15, 16 | 2, 3, 7, 10 | 5, 8, 9, 12 | |
| 1, 10, 12, 15 | 4, 5, 7, 16 | 2, 8, 11, 14 | 3, 6, 9, 13 | |
| 9, 10, 13, 16 | 2, 4, 6, 12 | 1, 7, 8, 14 | 3, 5, 11, 15 | |

# 4

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1, 12, 16 | 2, 15, 17 | 11, 13, 18 | 3, 7, 14 | 6, 9, 10 | 4, 5, 8 | | |
| 5, 10, 17 | 1, 3, 13 | 8, 14, 18 | 2, 9, 12 | 11, 15, 16 | 4, 6, 7 | | |
| 4, 10, 11 | 9, 15, 18 | 1, 2, 8 | 12, 14, 17 | 3, 6, 16 | 5, 7, 13 | | |
| 1, 5, 18 | 8, 16, 17 | 10, 14, 15 | 6, 12, 13 | 2, 7, 11 | 3, 4, 9 | | |
| 10, 16, 18 | 3, 11, 12 | 2, 5, 14 | 7, 8, 9 | 1, 6, 15 | 4, 13, 17 | | |

# 57

| | | | | | |
|---|---|---|---|---|---|
| 2, 4, 13, 20 | 1, 10, 11, 12 | 5, 6, 9, 19 | 3, 15, 16, 18 | 7, 8, 14, 17 | |
| 5, 11, 14, 16 | 1, 9, 15, 20 | 2, 3, 12, 19 | 6, 7, 13, 18 | 4, 8, 10, 17 | |
| 1, 14, 17, 19 | 3, 4, 11, 13 | 2, 7, 9, 16 | 6, 8, 10, 15 | 5, 12, 18, 20 | |
| 1, 6, 12, 16 | 2, 10, 14, 18 | 4, 7, 15, 19 | 3, 8, 9, 13 | 5, 11, 17, 20 | |
| 3, 9, 17, 18 | 1, 2, 5, 8 | 7, 12, 13, 14 | 10, 16, 19, 20 | 4, 6, 11, 15 | |
| 2, 6, 16, 17 | 4, 9, 12, 14 | 1, 3, 7, 20 | 8, 11, 18, 19 | 5, 10, 13, 15 | |
| 1, 13, 17, 19 | 4, 9, 16, 18 | 6, 8, 12, 20 | 2, 11, 14, 15 | 3, 5, 7, 10 | |
| 8, 13, 16, 19 | 1, 4, 5, 18 | 7, 9, 10, 11 | 3, 6, 14, 20 | 2, 12, 15, 17 | |

# 51

| | | | | |
|---|---|---|---|---|
| 1, 13, 14, 19 | 3, 9, 16, 18 | 7, 11, 12, 15 | 2, 5, 8, 20 | 4, 6, 10, 17 |
| 3, 11, 17, 19 | 2, 4, 7, 13 | 6, 9, 15, 20 | 1, 5, 12, 16 | 8, 10, 14, 18 |
| 1, 6, 15, 18 | 2, 9, 17, 20 | 10, 11, 13, 16 | 3, 5, 7, 14 | 4, 8, 12, 19 |
| 4, 14, 16, 20 | 6, 7, 18, 19 | 1, 8, 9, 11 | 2, 3, 10, 12 | 5, 13, 15, 17 |
| 5, 9, 10, 19 | 7, 8, 16, 17 | 1, 3, 4, 15 | 12, 13, 18, 20 | 2, 6, 11, 14 |
| 3, 6, 8, 13 | 9, 12, 14, 17 | 2, 15, 16, 19 | 4, 5, 11, 15 | 1, 7, 10, 20 |
| 5, 6, 12, 16 | 3, 11, 19, 20 | 4, 7, 9, 13 | 1, 2, 17, 18 | 8, 10, 14, 15 |

# 65

| | | | | | |
|---|---|---|---|---|---|
| 1, 6, 15, 17 | 2, 8, 10, 20 | 3, 4, 12, 14 | 5, 9, 13, 19 | 7, 11, 16, 18 | |
| 3, 6, 10, 17 | 5, 11, 19, 20 | 12, 13, 15, 16 | 1, 2, 9, 14 | 4, 7, 8, 18 | |
| 3, 10, 16, 20 | 5, 9, 12, 17 | 6, 8, 14, 18 | 2, 7, 11, 13 | 1, 4, 15, 19 | |
| 6, 7, 9, 20 | 4, 5, 8, 13 | 2, 3, 15, 18 | 1, 10, 11, 12 | 14, 16, 17, 19 | |
| 3, 11, 13, 14 | 5, 15, 18, 20 | 1, 2, 7, 12 | 4, 6, 9, 16 | 8, 10, 17, 19 | |
| 1, 8, 11, 16 | 5, 7, 10, 14 | 6, 12, 15, 19 | 2, 3, 13, 18 | 4, 9, 17, 20 | |
| 9, 12, 18, 19 | 7, 10, 14, 15 | 3, 8, 11, 17 | 2, 5, 6, 16 | 1, 4, 13, 20 | |
| 7, 13, 15, 17 | 2, 6, 11, 19 | 4, 5, 10, 18 | 12, 14, 16, 20 | 1, 3, 8, 9 | |
| 8, 9, 15, 16 | 3, 7, 19, 20 | 1, 10, 13, 18 | 4, 6, 11, 14 | 2, 5, 12, 17 | |
| 2, 4, 16, 19 | 1, 3, 5, 7 | 9, 10, 11, 15 | 14, 17, 18, 20 | 6, 8, 12, 13 | |

# 62

| | | | | | |
|---|---|---|---|---|---|
| 7, 8, 16, 20 | 6, 10, 12, 13 | 1, 2, 4, 14 | 3, 5, 18, 19 | 9, 11, 15, 17 | |
| 6, 7, 16, 19 | 3, 4, 8, 10 | 2, 12, 17, 18 | 5, 14, 15, 20 | 1, 9, 11, 13 | |
| 5, 11, 18, 19 | 6, 9, 12, 15 | 1, 2, 3, 8 | 4, 7, 13, 20 | 10, 14, 16, 17 | |
| 13, 15, 16, 18 | 1, 12, 14, 19 | 3, 7, 9, 10 | 2, 6, 8, 11 | 4, 5, 17, 20 | |
| 2, 3, 7, 15 | 5, 8, 9, 12 | 4, 10, 11, 19 | 6, 13, 14, 17 | 1, 16, 18, 20 | |
| 8, 10, 13, 18 | 3, 5, 6, 14 | 4, 12, 15, 16 | 2, 9, 19, 20 | 1, 7, 11, 17 | |
| 2, 5, 9, 16 | 3, 13, 17, 19 | 4, 7, 12, 18 | 8, 11, 14, 15 | 1, 6, 10, 20 | |
| 3, 11, 12, 20 | 5, 7, 10, 17 | 2, 13, 14, 16 | 1, 8, 15, 19 | 4, 6, 9, 15 | |
| 1, 5, 12, 13 | 6, 8, 17, 19 | 7, 9, 14, 18 | 3, 4, 11, 16 | 2, 10, 15, 20 | |

# 17

| | | | | | |
|---|---|---|---|---|---|
| 9, 15, 18 | 3, 10, 16 | 5, 8, 17 | 1, 6, 13 | 2, 4, 7 | 11, 12, 14 |
| 9, 13, 14 | 6, 15, 17 | 4, 12, 16 | 1, 2, 8 | 10, 11, 18 | 3, 5, 7 |
| 7, 14, 18 | 5, 6, 9 | 4, 10, 13 | 2, 11, 17 | 8, 15, 16 | 1, 3, 12 |
| 1, 4, 15 | 6, 11, 16 | 3, 8, 14 | 5, 13, 18 | 2, 9, 12 | 7, 10, 17 |
| 8, 10, 12 | 3, 6, 18 | 7, 9, 16 | 4, 14, 17 | 2, 13, 15 | 1, 5, 11 |
| 2, 16, 18 | 4, 9, 11 | 1, 10, 14 | 5, 12, 15 | 3, 13, 17 | 6, 7, 8 |
| 2, 6, 10 | 3, 11, 15 | 4, 8, 18 | 1, 9, 17 | 7, 12, 13 | 5, 14, 16 |

# 13

| | | | | | | |
|---|---|---|---|---|---|---|
| 5, 17, 20 | 4, 6, 11 | 1, 7, 14 | 2, 15, 16 | 5, 10, 21 | 3, 13, 19 | 9, 12, 13 |
| 8, 12, 15 | 5, 14, 17 | 7, 9, 18 | 3, 6, 13 | 10, 16, 20 | 1, 2, 4 | 11, 19, 21 |
| 11, 15, 18 | 4, 12, 16 | 1, 9, 21 | 3, 5, 8 | 7, 13, 17 | 2, 6, 10 | 14, 19, 20 |
| 7, 8, 19 | 1, 13, 20 | 10, 11, 12 | 3, 9, 17 | 2, 14, 18 | 6, 16, 21 | 4, 5, 15 |
| 4, 9, 14 | 12, 18, 20 | 8, 11, 16 | 2, 5, 13 | 1, 10, 19 | 3, 7, 21 | 6, 15, 17 |
| 2, 17, 21 | 3, 15, 20 | 4, 7, 10 | 8, 13, 14 | 1, 16, 18 | 6, 12, 19 | 5, 9, 11 |

# 8

8, 17, 23 20, 22, 30 16, 21, 24 6, 19, 27 5, 25, 28 12, 18, 29 3, 10, 26 2, 4, 9 1, 14, 15 7, 11, 13
13, 15, 24 23, 27, 30 11, 12, 20 6, 8, 10 4, 5, 22 16, 17, 28 1, 2, 3 18, 21, 26 9, 14, 29 7, 19, 25
9, 24, 30 3, 7, 23 2, 18, 25 6, 16, 20 10, 12, 21 1, 4, 28 15, 22, 27 13, 17, 26 11, 14, 19 5, 8, 29
3, 20, 21 9, 13, 23 4, 12, 19 10, 11, 25 8, 16, 30 2, 26, 28 6, 7, 15 5, 18, 27 14, 17, 24 1, 22, 29
5, 12, 26 3, 17, 18 14, 25, 27 2, 10, 23 28, 29, 30 4, 20, 24 6, 11, 22 9, 15, 16 7, 8, 21 1, 13, 19

# 48

10, 15, 17, 31 16, 18, 23, 32 4, 9, 27, 29 3, 11, 20, 25 5, 8, 19, 22 1, 6, 12, 28 2, 7, 24, 26 13, 14, 21, 30
11, 13, 29, 32 7, 16, 22, 25 14, 26, 28, 31 4, 12, 18, 19 2, 5, 17, 21 3, 6, 8, 30 1, 10, 24, 27 9, 15, 20, 23
1, 20, 26, 30 6, 7, 11, 17 4, 21, 22, 24 3, 12, 14, 23 5, 13, 18, 28 8, 9, 10, 32 15, 19, 25, 27 2, 16, 29, 31
3, 5, 26, 32 2, 4, 13, 15 12, 17, 22, 29 1, 9, 25, 31 8, 16, 21, 27 10, 11, 14, 19 6, 18, 20, 24 7, 23, 28, 30
1, 13, 16, 19 3, 17, 18, 27 9, 12, 24, 30 6, 22, 23, 31 4, 5, 7, 20 2, 14, 25, 32 8, 11, 15, 28 10, 21, 26, 29
4, 30, 31, 32 12, 13, 20, 27 2, 10, 22, 28 1, 8, 14, 29 5, 6, 15, 16 9, 11, 18, 26 3, 7, 19, 21 17, 23, 24, 25

# 39

2, 10, 25, 28 15, 24, 27, 30 6, 17, 18, 22 12, 14, 19, 21 1, 16, 26, 29 9, 13, 20, 31 5, 7, 11, 32 3, 4, 8, 23
4, 5, 10, 19 3, 28, 30, 32 11, 13, 18, 26 1, 2, 14, 23 6, 8, 15, 20 9, 12, 24, 29 15, 17, 25, 31 7, 21, 22, 27
16, 23, 25, 27 5, 13, 14, 22 2, 8, 11, 15 4, 9, 26, 28 7, 12, 18, 20 1, 10, 21, 32 17, 19, 29, 30 3, 6, 24, 31
16, 21, 30, 31 2, 4, 12, 22 6, 13, 28, 29 8, 14, 25, 32 11, 17, 20, 27 1, 5, 18, 24 3, 7, 10, 26 9, 15, 19, 23
17, 24, 26, 32 1, 3, 11, 22 8, 19, 27, 31 2, 5, 6, 9 10, 12, 13, 15 20, 21, 23, 29 14, 16, 18, 28 4, 7, 25, 30

# 40

5, 21, 22, 23 7, 14, 27, 30 6, 28, 32, 33 1, 8, 19, 20 3, 13, 34, 35 9, 15, 29, 31 10, 11, 17, 24 2, 12, 16, 25 4, 18, 26, 36
4, 6, 16, 21 10, 12, 14, 18 5, 7, 8, 24 2, 20, 28, 31 9, 23, 35, 36 11, 27, 29, 32 13, 19, 22, 30 15, 17, 33, 34 1, 3, 25, 26
5, 9, 18, 34 6, 10, 19, 31 13, 20, 23, 32 7, 15, 26, 35 16, 22, 29, 33 8, 12, 28, 30 1, 4, 11, 14 2, 3, 24, 36 17, 21, 25, 27
13, 14, 25, 28 8, 15, 22, 32 24, 26, 27, 34 12, 17, 20, 36 4, 5, 31, 35 1, 10, 16, 23 3, 6, 9, 11 2, 7, 19, 29 18, 21, 30, 33
1, 2, 6, 34 3, 7, 10, 28 18, 22, 27, 31 11, 15, 30, 36 8, 25, 33, 35 9, 21, 24, 32 5, 12, 13, 29 4, 17, 19, 23 14, 16, 20, 26

# 88

8, 21, 32, 37, 39 13, 17, 19, 25, 35 7, 14, 34, 36, 40 9, 18, 22, 33, 38 2, 3, 6, 20, 30 1, 5, 10, 23, 29 11, 12, 15, 24, 28 4, 16, 25, 27, 31
10, 19, 34, 37, 38 7, 12, 23, 25, 35 11, 22, 26, 30, 31 2, 17, 24, 39, 40 1, 3, 8, 13, 28 5, 15, 27, 33, 36 4, 9, 20, 29, 32 6, 14, 16, 18, 21
1, 19, 24, 30, 36 5, 21, 25, 28, 38 11, 17, 18, 23, 32 2, 9, 14, 27, 35 12, 16, 20, 26, 37 4, 6, 8, 33, 34 7, 10, 13, 22, 39 3, 15, 29, 31, 40
7, 8, 18, 20, 31 1, 6, 26, 27, 40 2, 4, 11, 13, 38 10, 16, 28, 32, 36 9, 15, 19, 21, 23 3, 22, 24, 35, 37 5, 12, 17, 30, 34 14, 25, 29, 33, 39
18, 26, 28, 29, 34 5, 6, 19, 31, 39 3, 7, 16, 17, 33 14, 15, 30, 32, 38 13, 20, 23, 24, 27 1, 9, 11, 25, 37 4, 10, 21, 35, 40 2, 8, 12, 22, 36
1, 4, 12, 18, 39 14, 19, 20, 22, 28 6, 29, 35, 36, 38 9, 13, 16, 30, 40 3, 11, 21, 27, 34 8, 10, 15, 17, 25 2, 23, 31, 33, 37 5, 7, 24, 26, 32

# 41

5, 8, 28, 36 7, 19, 34, 37 17, 24, 32, 35 11, 15, 21, 25 1, 18, 33, 38 13, 30, 31, 40 3, 10, 23, 39 4, 14, 20, 27 2, 12, 16, 26 6, 9, 22, 29
15, 20, 22, 31 8, 13, 19, 33 4, 26, 28, 30 10, 12, 29, 40 17, 27, 37, 38 7, 9, 14, 23 1, 16, 32, 34 2, 3, 5, 11 18, 24, 25, 36 6, 21, 35, 39
5, 12, 18, 20 3, 21, 32, 40 14, 33, 34, 39 2, 13, 22, 35 4, 11, 19, 31 7, 17, 25, 28 8, 9, 26, 38 10, 16, 24, 30 1, 15, 27, 29 6, 23, 36, 37
4, 8, 24, 40 7, 26, 31, 39 11, 20, 29, 38 16, 22, 30, 34 9, 15, 16, 36 12, 14, 21, 37 5, 13, 23, 25 1, 10, 17, 19 3, 27, 28, 35 2, 6, 32, 33
8, 17, 22, 39 12, 23, 31, 38 1, 2, 4, 21 3, 6, 19, 30 10, 13, 18, 37 5, 7, 15, 32 14, 26, 29, 36 11, 16, 27, 40 20, 24, 28, 33 9, 25, 34, 35

## Bibliografía


[1] Bose, R. C., (1942). "A note on the resolvability of balanced incomplete block designs", *Sankhya*, **6**, 105-110.

[2] Bose, R. C. and Manvel B., (1984). *Introduction to combinatorial theory*, John Wiley & Sons, NY.

[3] Bose, R. C., and Nair, K. R., (1939). "Partially balanced incomplete block designs", *Sankhya*, **4**, 337-372.

[4] Bose, R. C., and Shimamoto, T., (1952). "Classification and analysis of partially balanced incomplete block designs with two associate classes", *J. Am. Stat. Assn.*, **47**, 151-184.

[5] Clatworthy, W. H., (1973). *Tables of two-associate class partially balanced designs. Appl. Math. Series*, **63**. Washington: National Bureau of Standards.

[6] de los Cobos Silva, S., (1994). *La técnica de la búsqueda tabú y sus aplicaciones*, Tesis Doctoral, Facultad de Ingeniería UNAM.

[7] Elenbogen, B. S. and Maxim, B. R., (1992). "Scheduling a bridge club (a case study in discrete optimization)", *Mathematics Magazine*, **65**:1, 18-26.

[8] Fiechter, C. N., (1994). "A parallel tabu search algorithm for large traveling salesman problems", *Discrete Applied Mathematics*, **51**, 243-267.

[9] Garey, M. R. and Johnson D. S., (1979). *Computers and intractability*, Freeman and Co., NY.

[10] Glover, F., (1989). "Tabu search part I", *ORSA Journal on Computing*, **1**:3, 190-206.

[11] Glover, F., (1990). "Tabu search part II", *ORSA Journal on Computing*, **2**:1, 4-32.

[12] Hertz, A., (1991). "Tabu search for large scale timetabling problems", *European Journal of Operational Research*, **54**, 39-47.

[13] Hertz, A. and de Werra, D., (1987). "Using tabu search techniques for graph coloring", *Computing*, **39**, 345-351.

[14] Holland, J., (1975). *Adaptation in natural and artificial systems*, University of Michigan Press, Ann Arbor, MI.

[15] Jarrett, R. G., (1983). "Definitions and properties for m-concurrence designs", *J. R. Statist. Soc.* B, **45**:1, 1-10.

[16] Kernighan, B. W. and Ritchie D. M., (1989). *El lenguaje de programación C*, Prentice-Hall, México.

[17] Kirkpatrick, S., Gelatt, C. D., Jr., and Vecchi, M., (1983). "Optimization by simulated annealing", *Science*, **220**, 671-680.

[18] Mathon, R. and Rosa, A., (1990). "Tables of parameters of BIBDs with $r \leq 41$ including existence, enumeration and resolvability results: an update", *ARS Combinatoria*, **30**, 65-96.

[19] McBryan, O., (1994). "An overview of message passing environments", *Parallel Computing*, **20:4**, 417-444.

[20] McCulloch, W. S., and Pitts, W., (1943). "A logical calculus of the ideas immanent in nervous activity", *Bulletin of Mathematical Biophysics*, **5**, 115-133.

[21] Morales, L. B., (1995). "Solution for a scheduling problem of a bridge club", *Mathematics Magazine*, (aceptado).

[22] Morales, L. B., (1995). "Constructing optimal schedules for certain types of tournaments using tabu search", *Investigación Operativa*, (aceptado).

[23] Morales, L. B., Ordoñez C., Zamudio, M., "Parallel tabu search to construct incomplete block designs", (en preparación).

[24] Raghavarao, D., (1971). *Constructions and combinatorial problems in design of experiments*, John Wiley & Sons, NY.

[25] Reeves, C. R., (1993). *Modern heuristics techniques for combinatorial problems*, John Wiley & Sons, NY.

[26] Shrikhande, S. S., and Raghavarao, D., (1963). "Affine $\alpha$-resolvable incomplete block designs", *Contributions to Statistics*. Presented to Prof. P. C. Mahalanobis on the occasion of his 70th birthday. Pergamon Press, pp. 471-480.

[27] Skorin-Kapov, J., (1990). "Tabu search applied to the quadratic assignment problem", *ORSA, Journal on Computing*, **2:1**, 33-45.

[28] Tucker, A., (1985). *Applied Combinatorics*, 2nd edition, John Wiley & Sons, NY.

[29] Tucker, L. W. and Mainwaring, A., (1994). "CMMD: active messages on the CM-5", *Parallel Computing*, **20:4**, 481-496.

[30] Yates, F., (1936). "Incomplete randomized blocks", *Ann. Eugen.*, **7**, 121-140.