

Índice

1. Problemas de optimización	4
1.1. Espacio de búsqueda	4
1.2. Problema de la mochila	5
1.3. Mochila multidimensional	6
1.4. Dependencia y Exclusión	7
1.5. Set covering - bomberos	7
1.6. Set partitioning	10
1.7. Set packing	11
1.8. Generación de columnas	12
1.8.1. AA Crew Scheduling	13
1.9. Travelling Salesman Problem	14
1.10. Preguntas de Examen/Control	15
2. Complejidad computacional	17
2.1. Transformación polinomial	17
2.2. NP completo y NP difícil	17
2.3. Preguntas de Examen/Control	18
3. Problemas de asignación	19
3.1. Asignación 1 a 1	19
3.2. Asignación cuadrática (Mall layout)	20
3.3. Job Shop	21
3.4. VRP	22
3.5. Preguntas de Examen/Control	23
4. CSP y CSOP	24
4.1. Variable encapsuladora	24
4.2. N-Reinas	25
4.2.1. Modelo binario	25
4.2.2. Modelo casilla	26
4.2.3. Modelo fila	26
4.2.4. Modelo columna	27
4.3. Sudoku	28
4.3.1. Fase de transición	28
4.4. Buscaminas	29
4.5. Car Sequencing Problem	30
4.6. Coloreo de Grafos	31
4.7. 2d space planning	32
4.8. UCTP	33
4.9. Preguntas de Examen/Control	34
5. Técnicas de Filtro y Consistencia	38
5.1. Nodo consistencia	39
5.2. Arco consistencia	40
5.2.1. Procedimiento REVISE	41
5.2.2. AC-1	41

5.2.3. AC-3	42
5.3. Camino consistencia	44
5.4. K consistencia	45
5.5. Preguntas de Examen/Control	45
6. Técnicas Completas	49
6.1. Look-Back: Backtracking	49
6.2. Trashing	52
6.3. BT+GBJ	52
6.4. BT+CBJ	53
6.5. Look Ahead	54
6.6. Look-Ahead: FC	54
6.7. Look-Ahead: RFLA	57
6.8. Heurísticas de selección de Variables/Valores	58
6.9. Preguntas de Examen/Control	59
7. Técnicas incompletas	63
7.1. Algoritmos constructivos	64
7.2. Algoritmos reparadores	65
7.2.1. Preguntas del Examen/Control	66
7.3. Hill Climbing	67
7.3.1. Hill Climbing Mejor Mejora	67
7.3.2. Hill Climbing AM	69
7.3.3. Hill Climbing + Restart	70
7.3.4. Preguntas del Examen/Control	72
7.4. Tabu Search	76
7.4.1. Preguntas del Examen/Control	79
7.5. Simulated Annealing	83
7.5.1. Preguntas del Examen/Control	85
7.6. Heurísticas K-Opt	86
7.6.1. Preguntas del Examen/Control	88
8. Algoritmos genéticos	88
8.1. Algoritmo Genético (GA)	92
8.2. Algoritmo Genético Modificado	93
8.3. Programación Evolutiva (EP)	97
8.4. Estrategias Evolutivas (ES)	98
8.5. Programación Genética (PG)	99
8.6. Preguntas del Examen/Control	99
9. Asignación de Parámetros	107
9.1. Sintonización de Parámetros	108
9.1.1. F-Race	109
9.1.2. ParamILS	111
9.2. Control de Parámetros	112
9.2.1. Control en Estrategias Evolutivas	114
9.2.2. Control en Simmulated Annealing	115
9.2.3. Control en Tabu Search para Asignación Cuadratica (QAP)	115

9.3. Preguntas del Examen/Control	116
---	-----

1. Problemas de optimización

Optimización - determinación de una alternativa de decisión que es mejor que cualquier otra bajo algún criterio (es la mínima, etc).

En optimización se trabaja con:

- Función objetivo - medida cuantitativa que permite evaluar el funcionamiento del sistema analizado
- Variables - decisiones que pueden afectar el valor de función objetivo
- Dominios - valores posibles de las variables
- Restricciones - relaciones que deben cumplir las variables
- Parámetros o Constantes - atributos de problema conocidos apriori que permiten simplificar el problema

Modelo - es una interpretación del problema en lenguaje matemático que implica entender el problema. Sirve como mecanismo de comunicación y permite identificar semejanzas entre los problemas.

El modelado muy común es modelado con variables **enteras y binarias**

- Permiten modelar cantidad discretas
- Modelado de decisiones que implican costo fijo (adquirir un activo o no, poner en marcha un proceso o no)
- Modelado de restricciones no lineales
- Modelado de condiciones lógicas
- Modelado de decisiones dependientes (toma de una decisión permite hacer otra decisión - si se compra un aparato, podemos decidir sobre su uso)

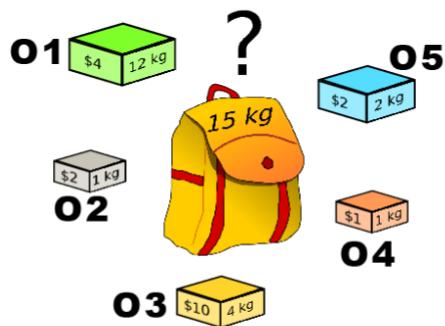
1.1. Espacio de búsqueda

EDB o Espacio de búsqueda contiene todas las soluciones para el problema, tanto factibles como infactibles.

El fenómeno de **explosión combinatorial** corresponde al crecimiento desmedido del espacio de búsqueda al aumentar el tamaño de problema (cantidad de variables o sus dominios).

1.2. Problema de la mochila

Problema de la mochila (Knapsack problem) modela una situación análoga al llenado de una mochila, con una capacidad determinada en peso. La idea es seleccionar un subconjunto de objetos, donde cada objeto cuenta con un peso y ganancia específicos. Los objetos colocados en la mochila deben **maximizar la ganancia total** sin exceder la capacidad de la mochila.



Modelo:

Variables: para $i \in \{1 \dots n\}$

$$X_i = \begin{cases} 1, & \text{si se agrega el objeto } i \text{ a la mochila} \\ 0, & \text{en el caso contrario} \end{cases}$$

Parámetros/Constantes:

- P_i - peso de objeto i
- g_i - ganancia obtenida por agregar el objeto i a la mochila
- n - cantidad de objetos
- C - capacidad de la mochila

Función objetivo: maximizar la ganancia

$$\max \sum_{i=1}^n x_i g_i$$

Sujeta a las restricción:

- Capacidad de la mochila: $\sum_{i=1}^n x_i P_i \leq C$

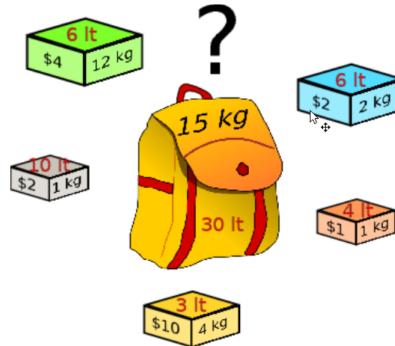
Hay n variables, donde cada una de estas puede tomar 2 valores: 1 o 0. El espacio de búsqueda en este problema corresponde a: 2^n . Se observa que al aumentar la cantidad de variables n el problema sufre de la **explosión combinatorial**.

Tabla: Explosión Combinatorial - Ejemplo KP

#Variables	Tamano EDB
5	$2^5 = 32$
6	$2^6 = 64$
7	$2^7 = 128$
20	$2^{20} = 1.048.576$
50	$2^{50} = 1.12 * 10^{15}$
100	$2^{100} = 1,26 * 10^{30}$

1.3. Mochila multidimensional

En mochila multidimensional se agrega la capacidad por una dimensión específica.



Modelo:

Variables: para $i \in \{1 \dots n\}$

$$X_i = \begin{cases} 1, & \text{si se agrega el objeto } i \text{ a la mochila} \\ 0, & \text{en el caso contrario} \end{cases}$$

Parámetros/Constantes:

- P_{ij} - peso de objeto i en la dimensión j
- g_i - ganancia obtenida por agregar el objeto i a la mochila
- n - cantidad de objetos
- C_j - capacidad de la mochila en la dimensión j

Función objetivo: maximizar la ganancia

$$\max \sum_{i=1}^n x_i g_i$$

Sujeta a las restricción:

- Capacidad de la mochila: $\sum_{i=1}^n x_i P_{ij} \leq C_j, \forall j$

1.4. Dependencia y Exclusión

- **Actividad 4 no se puede realizar si se hizo la actividad 5 y viceversa**

La expresión es

$$x_4 + x_5 \leq 1$$

para permitir que se realice solo una actividad, o ninguna de las dos. Expresiones como $x_4 + x_5 = 1$ y $x_4! = x_5$ quitan la posibilidad de realizar ninguna actividad (ambas 0).

- **Las actividad 3,4,5 son excluyentes entre si**

$$x_3 + x_4 + x_5 \leq 1$$

No se pone $x_3 + x_4 + x_5 = 1$ ya que si bien se debe realizarse solo una actividad, esta condición no permite no realizar ninguna actividad. También es cierto:

$$x_3 + x_4 + x_5 < 2$$

- **Actividad 11 requiere que la actividad 2 se realice**

$$x_{11} \leq x_2$$

De esta manera solo si la actividad 2 se realiza, podemos realizar la actividad 11. Sin embargo la actividad 2 queda libre y puede realizarse o no, ya que no depende de la 11. Usar $x_{11} = x_2$ es incorrecto porque introduce dependencia de que actividad 2 no se realiza sin 11. $x_{11} + x_2 = 2$ nuevamente es incorrecto porque implica que ambas deben realizarse, lo cual no es cierto.

- **Actividad 1 depende de que se realice actividad 3 o 4** Podemos escribir 'x3 o x4' como $x_3 + x_4$. Entonces si la x_1 depende de alguna de estas podemos anotar:

$$x_1 \leq x_3 + x_4$$

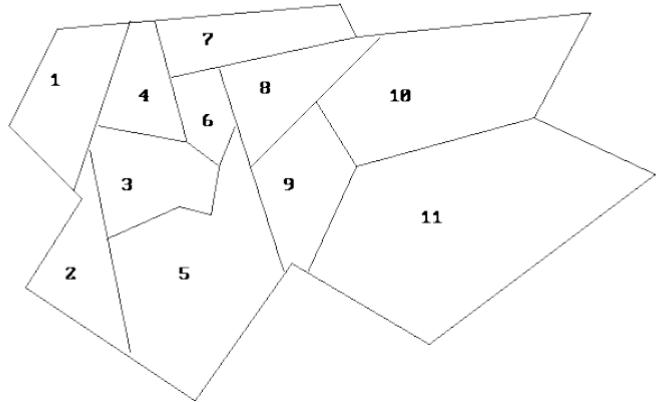
O tambien es cierto que:

$$x_1 < x_3 + x_4 + 1$$

1.5. Set covering - bomberos

Set covering trabaja con un conjunto de elementos. Se requiere que cada elemento pertenezca a **por lo menos un conjunto**. Puede pertenecer a varios.

Instalar la menor cantidad de Estaciones de Bomberos de manera que puedan satisfacer las demandas de las 11 comunas de una ciudad. Se considera que una Estación de Bomberos es capaz de satisfacer las demandas de la comuna en la que se encuentra y de las inmediatamente adyacentes a dicha comuna.



Modelo:

Variables: para $i \in \{1 \dots n\}$

$$X_i = \begin{cases} 1, & \text{si se instala un estación de bomberos en comuna } i \\ 0, & \text{en el caso contrario} \end{cases}$$

Matriz de adyacencia:

$$a_{ij} = \begin{cases} 1, & \text{comuna } i \text{ es adyacente (vecina) a } j \\ 0, & \text{en el caso contrario} \end{cases}$$

Función objetivo: minimizar la cantidad de estaciones instaladas

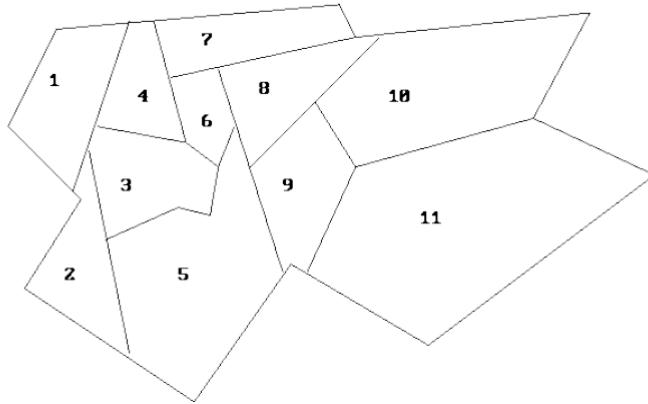
$$\min \sum_{i=1}^n x_i$$

Restricciones:

- Entre las comunas adyacentes debe haber por lo menos una estación de bomberos:
 $\sum_{i=1}^n a_{ij} x_i \geq 1$

También existe la versión de Problema de Bomberos que usa **la prioridad de comunas en base a la población**. Además se tiene el número **fijo** de estaciones (igual a 2):

Instalar dos Estaciones de Bomberos de manera que puedan satisfacer las demandas de las comunas más importantes de una ciudad. La importancia de la comuna viene dada por la cantidad de ciudadanos que viven en dicha comuna. Se considera que una Estación de Bomberos es capaz de satisfacer las demandas de la comuna en la que se encuentra y de las inmediatamente adyacentes a dicha comuna.



Variables: para $i \in \{1 \dots n\}$

$$X_i = \begin{cases} 1, & \text{si se instala un estación de bomberos en comuna } i \\ 0, & \text{en el caso contrario} \end{cases}$$

$$Y_i = \begin{cases} 1, & \text{si la comuna } i \text{ NO está cubierta por estación} \\ 0, & \text{en el caso contrario} \end{cases}$$

Parámetros:

- P_i - prioridad de la comuna i
- Matriz de adyacencia:

$$a_{ij} = \begin{cases} 1, & \text{comuna } i \text{ es adyacente (vecina) a } j \\ 0, & \text{en el caso contrario} \end{cases}$$

Función objetivo: minimizar

$$\min \sum_{i=1}^n P_i y_i$$

Restricciones:

- Si la comuna j tiene una estación adyacente, entonces el valor de la variable y_j está obligado a ser 0 (ya que la suma dará 1 si es que hay estación) $\sum_{i=1}^n a_{ij} x_i + y_j \leq 1$
- Hay exactamente 2 estaciones de bomberos: $\sum_{i=1}^n x_i = 2$

Otro problema es la selección de equipos con traslape (con posible repetición de jugadores):

Considere un conjunto S de personas:

$$S = \{1, 2, 3, 4, 5\}$$

Suponga que desea organizar dichas personas en varios equipos conocidos:

- El conjunto de equipos posibles es s
- Por ejemplo: $s = \{\{1, 2\}, \{4, 5\}, \{1, 3, 5\}, \{2, 4, 5\}, \{1\}, \{3\}\}$
- Cada equipo tiene un costo c_j
- Se desea elegir los equipos que permiten tener ocupadas a **todas** las personas con un costo mínimo
 - No importa que la misma persona este en más de un equipo

Modelo:

Variables: para $j \in \{1 \dots n\}$

$$X_j = \begin{cases} 1, & \text{eschoje el equipo } j \\ 0, & \text{en el caso contrario} \end{cases}$$

Matriz de adyacencia:

$$a_{ij} = \begin{cases} 1, & \text{persona } i \text{ es parte de equipo } j \\ 0, & \text{en el caso contrario} \end{cases}$$

Parámetros:

- c_j - costo de equipo j

Función objetivo: minimizar el costo

$$\min \sum_{j=1}^n x_j c_j$$

Restricciones:

- Persona i debe estar en por lo menos un conjunto $\sum_{i=1}^n a_{ij} x_i \geq 1$

1.6. Set partitioning

Set covering trabaja con un conjunto de elementos. Se requiere que cada elemento pertenezca a **exactamente un conjunto**. No deben quedar elementos sin un grupo asignado.

Considere un conjunto S de personas:

$$S = \{1, 2, 3, 4, 5\}$$

Suponga que desea organizar dichas personas en varios equipos conocidos:

- El conjunto de equipos posibles es s
- Por ejemplo: $s = \{\{1, 2\}, \{4, 5\}, \{1, 3, 5\}, \{2, 4, 5\}, \{1\}, \{3\}\}$
- Cada equipo tiene un costo c_j
- Se desea elegir los equipos que permiten que cada persona esté **exactamente en un equipo** con un beneficio máximo

Modelo:

Variables: para $j \in \{1 \dots n\}$

$$X_j = \begin{cases} 1, & \text{es elige el equipo } j \\ 0, & \text{en el caso contrario} \end{cases}$$

Matriz de adyacencia:

$$a_{ij} = \begin{cases} 1, & \text{persona } i \text{ es parte de equipo } j \\ 0, & \text{en el caso contrario} \end{cases}$$

Parámetros:

- c_j - costo de equipo j

Función objetivo: minimizar el costo

$$\min \sum_{j=1}^n x_j c_j$$

Restricciones:

- Persona i debe estar en exactamente un grupo $\sum_{j=1}^n a_{ij} x_j = 1$

1.7. Set packing

Set covering trabaja con un conjunto de elementos. Se requiere que cada elemento pertenezca a **a lo más un conjunto**. Puede pertenecer a ninguno.

Considere un conjunto S de personas:

$$S = \{1, 2, 3, 4, 5\}$$

Suponga que desea organizar dichas personas en varios equipos conocidos:

- El conjunto de equipos posibles es s
- Por ejemplo: $s = \{\{1, 2\}, \{4, 5\}, \{1, 3, 5\}, \{2, 4, 5\}, \{1\}, \{3\}\}$
- Cada equipo tiene un beneficio b_j
- Se desea elegir los equipos que proporcionen el beneficio máximo **sin que se traslapen**.
 - Que las personas no se repitan entre equipos diferentes, no importa que una persona no esté en ningún equipo

Modelo:

Variables: para $j \in \{1 \dots n\}$

$$X_j = \begin{cases} 1, & \text{eschoje el equipo j} \\ 0, & \text{en el caso contrario} \end{cases}$$

Matriz de adyacencia:

$$a_{ij} = \begin{cases} 1, & \text{persona i es parte de equipo j} \\ 0, & \text{en el caso contrario} \end{cases}$$

Parámetros:

- b_j - beneficio de equipo j

Función objetivo: maximizar beneficio

$$\max \sum_{j=1}^n x_j b_j$$

Restricciones:

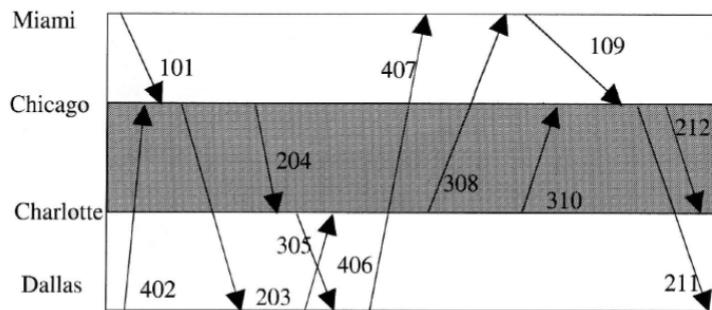
- Persona i debe estar en a lo más un grupo: $\sum_{i=1}^n a_{ij} x_i \leq 1$

1.8. Generación de columnas

Es una técnica de dos pasos que se usa para problemas combinatorias altamente complejas. Consistente en generación de columnas donde cada columna representa una alternativa factible siendo una parte posible de la solución.

1.8.1. AA Crew Scheduling

- Problema: Encontrar la secuencia de vuelos para cada tripulación sobre un período de tiempo. Cada secuencia debe comenzar y terminar en la ciudad donde vive la tripulación. Se deben realizar TODOS los vuelos.
- Suponga la siguiente secuencia de viajes de American Airlines.



Nota: Números de vuelos: Miami (100), Chicago (200), Charlotte (300), Dallas (400)

Los vuelos de Miami son 1XX, Chicago 2XX, etc.

Un posible viaje (columna) es: $C1 = [101, 294, 497]$. Otro posible viaje es $C2 = [308, 109, 212]$. Entonces, el **primer paso de algoritmo** consiste en generar todas las columnas:

Paso 1:
Generación de secuencias de vuelos (columnas)

i	Secuencia de Vuelos	Costo
1	101 - 203 - 406 - 308	2900
2	101 - 203 - 407	2700
3	101 - 204 - 305 - 407	2600
4	101 - 204 - 308	3000
5	203 - 406 - 310	2600
6	203 - 407 - 109	3150
7	204 - 305 - 407 - 109	2550
8	204 - 308 - 109	2500
9	305 - 407 - 109 - 212	2600
10	308 - 109 - 212	2300
11	310 - 212	2000
12	402 - 203	2100
13	402 - 204 - 305	2400
14	402 - 204 - 310 - 211	2550
15	406 - 308 - 109 - 211	2750
16	406 - 310 - 211	2600
17	407 - 109 - 211	2550

El **segundo paso** corresponde a la selección de conjunto óptimo usando Set Partitioning (o también puede servir covering)

Paso 2:

Resolver el problema como un Set [Covering/ Partitioning]

Variables:

$$x_i = \begin{cases} 1 & \text{Si se elige la columna } i \\ 0 & \text{si no} \end{cases}$$

Constantes:

$$a_{ij} = \begin{cases} 1 & \text{Si el vuelo } j \text{ está en la columna } i \\ 0 & \text{si no} \end{cases}$$

C_i = Costo de la columna i

Objetivo:

$$\text{Mín } \sum_{j=1}^{17} C_j \cdot x_i$$

Restricciones:

$$\sum_{i=1}^{17} a_{ij} \cdot x_i = 1 \quad \forall j$$

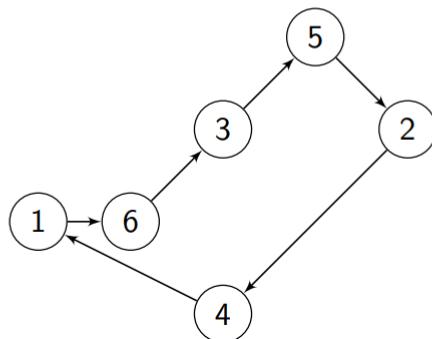
Solución $x_1 = x_9 = x_{14} = 1$ a un costo total de 8050



La ventaja de este método es la flexibilidad, y la desventaja es la necesidad de generar y enumerar las columnas.

1.9. Travelling Salesman Problem

El problema del vendedor viajero (Traveling Salesman Problem (TSP)) consiste en encontrar un circuito de costo mínimo que pase una sola vez por cada ciudad que debe visitar el vendedor y que le permita volver a su ciudad de origen al final del día.



Modelo:

Variables: para $j \in \{1 \dots n\}$

$$X_{ij} = \begin{cases} 1, & \text{se toma ruta de } i \text{ a } j \\ 0, & \text{en el caso contrario} \end{cases}$$

u_i - orden en que se visita la ciudad i

Parámetros:

- C_{ij} - costo de viajar de i a j

Función objetivo: minimizar distancia

$$\min \sum_{j=1}^n \sum_{i=1}^n x_{ij} b_{ij}$$

Restricciones:

- Salir una vez de una ciudad: $\sum_j x_{ij} = 1, \forall i$
- Entrar una vez en una ciudad: $\sum_j x_{ji} = 1, \forall i$
- Evitar ciclos:
 - Dos ciudades no pueden estar en misma posición del tour: $u_i! = u_j$ para $i! = j$
 - Ciudad inicial arbitraria 1: $u_1 = 1$
 - Restricción de distancia de orden $u_j - u_i \leq 1 + (n - 1)(1 - x_{ij}), \forall i, j \neq 1$

$(1 - x_{ij})$ activa o relaja las restricciones y x_{ij} se conoce como **variable de activación**. Por ejemplo, si se viaja de Ciudad 3 a Ciudad 5 en orden seguido $u_3 = 2, u_5 = 3, x_{35} = 1$.

$$u_5 - u_3 \leq 1 + (n - 1)(1 - 1)$$

$u_5 - u_3 \leq 1$ - Orden restringido si se viaja de 3 a 5

1.10. Preguntas de Examen/Control

-
1. Los problemas de set-covering y set-partitioning:

V Comparten la definición de variables

Sí. Los tres problemas definen la misma variable binaria - se escoge grupo j o no.

V Comparten la definición de función objetivo

Sí, los tres problemas buscan minizar el costo o maximizar el beneficio, según la definición del problema. En clases vimos solo Set Packing con beneficio b_j en vez de costo c_j .

V Comparten la definición de constantes/parámetros

Sí, los tres problemas usan como parámetro costode agregar persona i en grupo j (o el beneficio en versión alternativa). Además los 3 probelmas usan la matriz de adycancia para ver si persona i está en grupo j o no

F Comparten la definición de restricciones

No. Set covering tiene restricción de que persona tiene que estar en por lo menos un conjunto. Set partitioning tiene restricción de que persona tiene que estar en un conjunto si o si. En caso de set packing, persona puede estar en a lo más un grupo.

■ **Tienen el mismo tamaño de espacios de búsqueda**

Sí, los tres problemas tienen el mismo espacio de búsqueda. Existen n variables para la cantidad de equipos $j \in \{1...n\}$. Cada variable toma 2 valores, por lo que el espacio es 2^n

2. En el problema de localización de Estaciones de Bomberos:

V **El tamaño del espacio de búsqueda crece a medida que aumenta la cantidad de comunas**

Sí, ya que se definen N variables X_i , una por cada columna. Mientras más variables hay, mayor es el espacio de búsqueda.

F La cantidad de restricciones aumenta a medida que aumenta el número de comunas adyacentes

No. Sigue siendo una restricción por cada comuna, ya que es una sumatoria.

F Se define una variable por Estación de Bomberos

No, es una variable por comuna.

F Se define una restricción por Estación de Bomberos

No, es una sola restricción para todas las estaciones.

■ La explosión combinatoria se produce al aumentar la cantidad de Estaciones de Bomberos

No, se produce al aumentar el número de comunas.

3. En la mochila multidimensional

V **El tamaño de espacio de búsqueda aumenta al aumentar la cantidad de objetos**

Sí, mientras más objetos hay, mayor es el espacio de búsqueda. El espacio de búsqueda sería 2^N , donde N - es la cantidad de objetos

4. ¿Cómo se escribe que las actividad 3,4 y 5 sean excluyentes en el modelo de presupuesto?

$x_3 + x_4 + x_5 \leq 1$ o también $x_3 + x_4 + x_5 < 2$. Esto permite o que se cumpla a lo más una de las tres actividades, o ninguna de las tres.

5. En el modelo de generación de columnas

■ Cada columna representa una solución

No, son secuencias factibles

■ Cada columna representa una secuencia que satisface todas las restricciones del problema

No, no necesariamente.

- Cada columna representa una secuencia de vuelos factibles a realizar por una tripulación
Sí, por definición
 - Cada columna representa una secuencia que satisface las restricciones de localidad y tiempo
Sí, por definición
- V Cada secuencia de vuelos debe comenzar y terminar en la misma ciudad
Sí, por definición
- F Las secuencias de vuelos son de largo fijo
No, no necesariamente
- F Las secuencias deben incluir mínimo 3 vuelos
No, no necesariamente
- F Se selecciona una cantidad fija de columnas en el modelo
No, no necesariamente

2. Complejidad computacional

Problema P - se dice que el problema es polinomial si existe un algoritmo de complejidad polinomial que permite responder la pregunta del problema con cualquier dato

Problema NP - se dice que el problema es no-determinístico polinomial si para cualquier instancia de este problema en un tiempo polinomial respecto al tamaño de instancia se puede comprobar si una solución adivinada es solución del problema o no.

Dilema: No existe certeza si clases P y NP coinciden o si P está estrictamente incluido en NP.

2.1. Transformación polinomial

Si se tienen 2 problemas de reconocimiento D_1, D_2

- Existe una transformación $f()$ que transforma cualquier instancia I de D_1 en una instancia $f(I)$ de D_2 . El algoritmo para calcular $f(I)$ debe ser polinomial
- Existe equivalencia entre dos enunciados. Si D_1 responde 'si' para instancia I , D_2 debe responder 'si' para instancia $f(I)$

Si D_1 se puede transformar en D_2 ($D_1 \prec D_2$) y D_2 en D_3 ($D_2 \prec D_3$) entonces existe una relación transitiva: $D_1 \prec D_3$

2.2. NP completo y NP difícil

El problema es NP-completo - Q es un problema de reconocimiento NP completo si es de clase NP y para todo problema Q' de la clase NP se tiene que $Q' \prec Q$.

Nos interesa demostrar que un problema es NP-completo para no buscar un algoritmo polinomial que resuelva el problema (porque no hay) y para justificar el uso de las heurísticas.

El problema es NP-difícil - sea O un problema de optimización. Si su problema de reconocimiento asociado es NP-completo, entonces O es NP-difícil.

Por ejemplo, RTSP (*¿existe un grafo hamiltoniano de largo k en el grafo?*) es un problema de reconocimiento y es NP-completo. Por ello TSP (determinar el mínimo largo de ciclo hamiltoniano), problema de optimización, es NP-difícil.

2.3. Preguntas de Examen/Control

1. Un problema NP-difícil:

F Corresponde a un problema de reconocimiento cuyas respuestas pueden verificarse en tiempos NO polinomiales

No. Problema NP-difícil es un problema de optimización.

V Corresponde a un problema de reconocimiento cuyo problema de optimización asociado es NP-completo

Sí, es la definición

F Corresponde a un problema de reconocimiento cuyo problema de optimización asociado es NP-completo

No. Los problemas NP-completos son los problemas de reconocimiento.

F Corresponde a un problema de optimización que puede ser verificado en tiempos polinomiales

No. Problemas NP-difíciles no verifican, sino optimizan.

2. Un problema NP-completo:

V Corresponde a un problema de reconocimiento cuyas respuestas pueden verificarse en tiempos polinomiales

Sí, es la definición.

F Corresponde a un problema de optimización que puede ser respondido en tiempos polinomiales

No, es un problema de reconocimiento.

V Corresponde a un problema de reconocimiento cuyo problema de optimización asociado se clasifica como NP-difícil

Sí, por definición.

F Corresponde a un problema de optimización cuyo problema de reconocimiento asociado se clasifica como NP-difícil

No, es un problema de reconocimiento con problema de optimización asociado NP-difícil

3. Una transformación polinomial permite transformar un problema D1 en D2:

V Siempre que haya una función polinomial f que transforme cada instancia de I de D1 en una instancia f(I) de D2

Sí, es definición

F Siempre que haya una función polinomial f que transforme un algoritmo que resuelve I de D1 en un algoritmo que resuelve f(I) de D2

No, se transforman las instancias, no los algoritmos

V Siempre que D1 responda sí para I y D2 responda sí para f(I)

Sí, si D1 responde a I entonces D2 debe responder a f(I)

F Siempre que D1 responda sí para I y también responda sí para f(I) No, D1 debe responder a I y D2 a f(I)

4. Un problema P

V Corresponde a un problema de reconocimiento que puede ser respondido en tiempos polinomiales

F Corresponde a un problema de reconocimiento cuyas respuestas pueden ser verificadas en tiempos polinomiales

F Corresponde a un problema de reconocimiento que NO puede ser respondido en tiempos polinomiales

F Corresponde a un problema de reconocimiento cuyas respuestas NO pueden ser verificadas en tiempos polinomiales

3. Problemas de asignación

3.1. Asignación 1 a 1

- Encontrar la mejor asignación (máquina-trabajo, personal-cliente) para minimizar costos

Máquina	Tiempo (horas)			
	Tarea 1	Tarea 2	Tarea 3	Tarea 4
1	14	5	8	7
2	2	12	6	5
3	7	8	3	9
4	2	4	6	10

Modelo:

Variables: para $j \in \{1 \dots n\}$

$$X_{ij} = \begin{cases} 1, & \text{trabajo } i \text{ esta asignado a la maquina } j \\ 0, & \text{en el caso contrario} \end{cases}$$

Parámetros:

- C_{ij} - costo de asignar trabajo i a la maquina j

Función objetivo: minimizar costos de asignacion

$$\min \sum_{j=1}^n \sum_{i=1}^n x_{ij} b_{ij}$$

Restricciones:

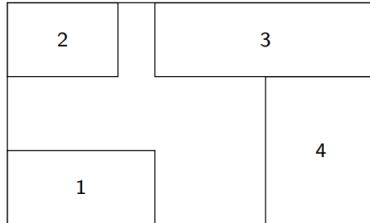
- Se puede asignar solo un trabajo a la máquina: $\sum_j x_{ij} = 1, \forall i$
- A cada trabajo le podemos asignar solo una máquina: $\sum_i x_{ji} = 1, \forall j$

3.2. Asignación cuadrática (Mall layout)

Modelos de Problema de Asignación Cuadrática

Ejemplo: Mall Layout

Se tienen 4 posibles ubicaciones para departamentos en un shopping mall. Se conocen las distancias (en metros) entre las ubicaciones. Se conoce además el número de clientes a la semana que desearán visitar los diferentes pares de departamentos. Por ejemplo, se proyecta que 500 clientes a la semana visitarán la tienda de ropa (Tienda 1) y computación (Tienda 2). El objetivo del problema es determinar la ubicación de las tiendas minimizando la molestia de los clientes.



Modelo:

Variables: i, j - índices de tiendas k, l - índices de lugares

$$X_{ik} = \begin{cases} 1, & \text{asignar tienda } i \text{ a lugar } k \\ 0, & \text{en el caso contrario} \end{cases}$$

Parámetros:

- d_{kl} - distancia entre lugar k y lugar l
- C_{ij} - cantidad de clientes entre tienda i y tienda j

Función objetivo: cuando se asigna tienda i a lugar k ($X_{ik} = 1$) y tienda j a lugar l ($X_{jl} = 1$) entonces queremos minimizar el producto de distancia por flujo de clientes con el fin de **minimizar la molestia de clientes**

$$\min \sum_l \sum_k \sum_j \sum_i d_{kl} C_{ij} X_{ik} X_{jl}$$

Como la FO tiene un producto entre dos variables: $x_{ik}x_{jl}$, el problema se conoce como asignación cuadrática.

Restricciones:

- Se puede asignar solo una tienda a una ubicación: $\sum_k x_{ik} = 1, \forall i$

- Cada tienda puede estar en solo una ubicación: $\sum_i x_{ik} = 1, \forall k$

Se puede reducir la FO a una FO lineal introduciendo una variable auxiliar. Esta variable no va a tomar la decisión sino que va a activarse bajo ciertas condiciones:

$$Y_{ijkl} = \begin{cases} 1, & \text{se asigna tienda } i \text{ a lugar } k, \text{ y tienda } j \text{ a lugar } l \\ 0, & \text{en el caso contrario} \end{cases}$$

Entonces FO lineal sería:

$$\min \sum_l \sum_k \sum_j \sum_i d_{kl} C_{ij} y_{ijkl}$$

Ya que esta variable no se va a activar por si sola, necesitamos agregar una restricción más:

- Restricción de **activación**. Cuando se asigna tienda i a lugar k, y tienda j a lugar l, y_{ijkl} tendrá que ser 1: $x_{ik} + x_{jl} - 1 \leq y_{ijkl}$

3.3. Job Shop

2. Modele las restricciones de precedencia del Job-Shop. Un problema tipo job-shop considera la planificación óptima para una colección dada de trabajos. Cada uno de dichos trabajos requiere una secuencia de procesadores, los cuales pueden realizar sólo un trabajo a la vez. Suponga tres trabajos a realizar: W_1 , W_2 y W_3 . Cada uno compuesto por las siguientes secuencias de uso de procesadores:

W_1	
p_2	3
p_3	3
p_4	3
...	...
p_8	2

W_2	
p_3	7
p_1	2
...	...
p_9	6

W_3	
p_7	5
p_6	9
p_3	2
p_5	1
...	...
p_4	5

Modelo:

Variables:

X_{ik} = instante de tiempo en que el trabajo i comienza a usar el procesador k

$$Y_{ijk} = \begin{cases} 1, & \text{trabajo } i \text{ se realiza antes de trabajo } j \text{ en procesador } k \\ 0, & \text{en el caso contrario} \end{cases}$$

Parámetros:

- t_{ik} - tiempo que toma el trabajo i en procesador k
- $a_{ikl} = \begin{cases} 1, & \text{trabajo } i \text{ usa el procesador } k \text{ antes que el procesador } l \\ 0, & \text{en el caso contrario} \end{cases}$

Función objetivo:

$$\min(\max\{x_{18} + 2, x_{29} + 6, x_{34} + 5\})$$

Restricciones:

- Restricción de precedencia - una procesador puede realizar solo una operación a la vez. Si el trabajo j se realiza después de trabajo i en el mismo procesador k ($y_{ijk} = 1$), entonces el momento de tiempo en que se empieza el trabajo j debe ser como mínimo después de que se termine el trabajo i .

$$x_{ik} + t_{ik} \leq x_{jk} + M(1 - y_{ijk})$$

- Restricción disyuntiva. Ninguna operación puede ser realizada en más de una máquina al mismo tiempo. Si la operación i necesita usar primero el procesador k y después l , primero tiene que terminar de usar el operador k . Es decir, el tiempo de inicio de uso de procesador l es como mínimo después de desocupar el procesador k .

$$x_{ik} + t_{ik} \leq x_{il} + M(1 - a_{ikl})$$

- M es un valor arbitrario y muy grande

3.4. VRP

El problema de ruteo de vehículos (Vehicle routing problem (VRP)) busca satisfacer la demanda de un conjunto de clientes utilizando una flota de vehículos. Los bienes son despachados desde un almacén central a todos los consumidores. El objetivo consiste en construir las rutas que deben realizar los vehículos de la flota de modo de minimizar los costos de entrega (tiempo, bencina).

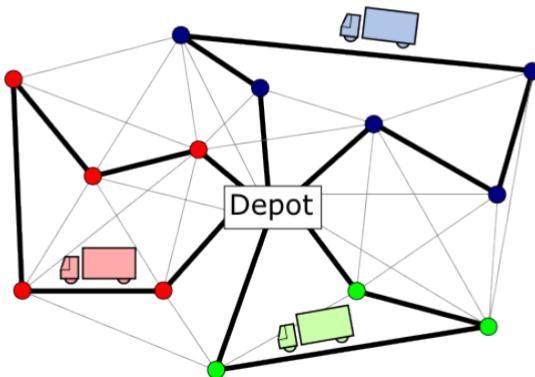


Figura: <http://www.liacs.nl/~ftakes/vrp/>

Modelo:

Variables:

$$X_{ij} = \begin{cases} 1, & \text{se usa el camino de } i \text{ a } j \\ 0, & \text{en el caso contrario} \end{cases}$$

u_i - carga de vehículo en el nodo i

Parámetros:

- c_{ij} - costo de viajar de i a j
- K - número de vehículos
- d_i - demanda del nodo i
- C - capacidad de cada vehículo

Función objetivo: minimizar los costos

$$\min \sum_i \sum_j c_{ij} x_{ij}$$

Restricciones:

- Se puede entrar y salir de cada nodo solo una vez: $\sum_i x_{ij} = 1$ y $\sum_j x_{ij} = 1$
- Restricción de distancia de orden: $u_j - u_i \leq d_j - C(1 - x_{ij})$
- Restricción de satisfacción de demanda: $d_i + d_j \leq C$
- Carga de vehículo: $0 \leq u_i \leq C - d_i$
- De deposito deben salir K vehículos: $\sum_i x_{i0} = K$
- Si salen K vehículos, los K deben volver al deposito: $\sum_j x_{0j} = K$

3.5. Preguntas de Examen/Control

-
1. Considere la siguiente definición de variables para el job-shop: X_{ik} : Instante de tiempo en que el trabajo i comienza a usar el procesador k $Y_{ijk} = 1$ si el trabajo i usa el procesador k antes que el trabajo j, 0 si no y las constantes: t_{ik} = tiempo que el trabajo i usa el procesador k La restricción: $X_{ik} + t_{ik} \leq X_{jk} + M \cdot (1 - Y_{ijk}), \forall i, j, k, i \neq j$
 - F Significa que el trabajo i debe terminar de usar el procesador k antes de usar el procesador j si es que requiere usarlos en ese orden
No. Es una restricción de precedencia de uso de mismo procesador que indica que un procesador puede hacer solamente un trabajo a la vez.
 - V Significa que el trabajo i debe terminar de usar el procesador k antes de usar el procesador j si es que requiere usarlos en ese orden
Sí, indica la precedencia para trabajos en mismo procesador.
 - F Significa que cada trabajo debe usar los procesadores en un orden establecido
No, solo indica que no deben ser dos trabajos a la vez.
 - V Significa que el uso de los procesadores es excluyente respecto a los trabajos
Sí, solo un trabajo puede ser realizado en un procesador al mismo tiempo
 2. Considere las siguientes definiciones de variables para el problema de ruteo de vehículos: $x_{ij} = 1$ si se usa el arco que va desde el nodo i al nodo j, 0 si no u_i = carga en el vehículo después de visitar el nodo i Si d_i corresponde a la demanda del nodo i y C a la capacidad de los vehículos. La restricción: $u_i - u_j \geq d_j - C \cdot (1 - x_{ij})$

- F Significa que si se visita el nodo j después del nodo i, debe迫使 el incremento de la carga en el vehículo respecto a la demanda del nodo j
No, no se fuerza el incremento en la carga si no se restringe que la carga debe ser suficiente para la demanda de nodo j
- F Significa que NO es posible visitar el nodo j después del nodo i si no queda capacidad en el vehículo
No. Solo se fuerza que haya suficiente carga para satisfacer la demanda de nodo j
- V Significa que si NO se visita el nodo j después del nodo i, las cargas de vehículos para i y j no necesitan estar relacionadas
Sí.
- V Permite evitar ciclos en las rutas de los vehículos
Sí, no se puede cumplir esta condición si haya tanto ruta de i a j como de j a i.

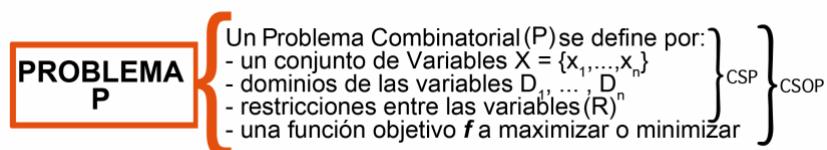
3. Respecto a los problemas de asignación y asignación cuadrática

- V Ambos permiten definir variables en común
Sí, permiten definir las variables de asignación de X a Y, por ejemplo
- V Ambos permiten definir restricciones en común
Sí, permiten definir restricción de una cosa X asignada a una cosa Y (y viceversa)
- F Ambos permiten definir función objetivo en común
No, función objetivo depende del problema
- F Solo asignación cuadrática requiere de asignaciones 2 a 1
No, es 1 a 1

4. CSP y CSOP

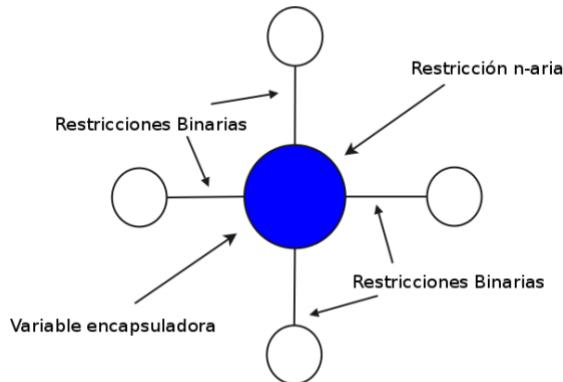
Los problemas CSP o Constraint Satisfaction Problem buscan encontrar un valor de las variables que satisface todas las restricciones, o bien, decir que no existe solución.

CSOP o Constraint Satisfaction Optimization Problem buscará optimizar el valor encontrado según el criterio definido.



4.1. Variable encapsuladora

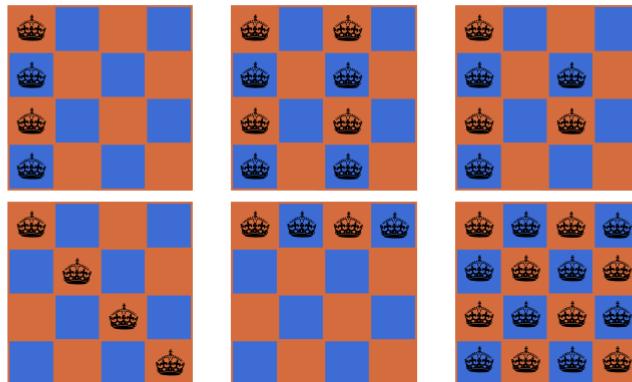
Es una variable que permite encapsular restricciones n-arias en una serie de restricciones binarias. La definición de restricciones depende de cómo se definen las variables.



4.2. N-Reinas

4.2.1. Modelo binario

Si se decide usar el modelo binario X_{ij} - si reina esta en casilla ij o no, entonces tenemos que manejar todas las restricciones de ataque por fila, columna y diagonal explícitamente.



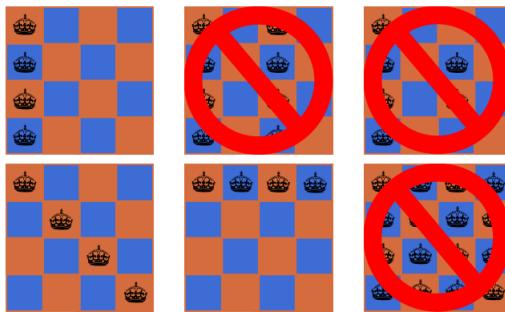
Ejemplo: N reinas

- X_{ij} - Binario
 - 1 : hay una reina en la casilla ij
 - 0 : caso contrario
- $\forall i, j \in 1 \dots 4$, si $N = 4$.

Cada variable X_{ij} posee dos posibles valores: $\{0, 1\}$. En total existen 4 filas y 4 columnas, lo cual nos da el total de $4 \cdot 4 = 16$ variables. Esto resulta en el espacio de búsqueda $2^{16} = 65536$. Para caso de N reinas este espacio es $2^{N \cdot N}$.

4.2.2. Modelo casilla

En el caso de modelo de X_k donde k indica la reina, con $k \in \{1, \dots, N\}$, esta definición de k ya no permite representar los tableros con más de k reinas. Es decir, ya estamos frente de encapsulamiento de la restricción : *la cantidad de reinas colocadas debe ser N .*



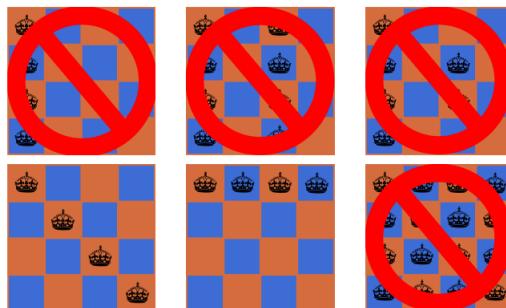
Ejemplo: N reinas

- X_k : casilla donde se encuentra la reina k
con $k \in [1, 4]$ y $X_k \in [1, 16]$

El espacio de búsqueda en este considera que cada variable puede tomar 16 valores, y existen a lo más 4 variables. Entonces el espacio es $16^4 = 65536$. En caso de N reinas este espacio es $(N \cdot N)^N$

4.2.3. Modelo fila

En el caso de modelo fila la respuesta tiene forma de un arreglo de largo N . Por ejemplo $(1, 2, 3, 4)$ representa la situación de las 4 reinas colocados a lo largo de la diagonal del tablero: reina de columna 1 esta en fila 1, reina de columna 2 en fila 2, etc.



Ejemplo: N reinas

- X_i : fila donde se encuentra la reina de la columna i
con $i \in [1, 4]$ y $X_i \in [1, 4]$

En este caso se encapsulan las restricciones de la cantidad de reinas permitidas, y además, la restricción de que en una columna puede haber a lo más una reina.

El espacio de búsqueda entonces considera que hay 4 variables, que pueden tomar a 4 valores distintos. Esto es $4^4 = 256$. En caso de N reinas el espacio sería N^N

Podemos **modelar** este problema:

Variables:

X_i - fila en que se encuentra la reina de la columna i

Constantes:

- N - cantidad de reinas

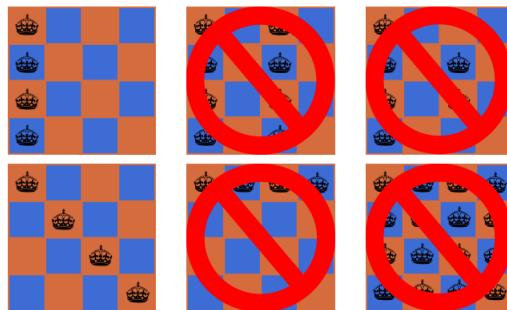
Restricciones:

- No puede haber más de una reina en la fila: $X_i \neq X_j$ para $i \neq j$
- Las reinas no deben atacarse en diagonal: $|X_i - X_j| \neq |i - j|$

El grafo de restricciones para este modelo es un grafo N-conexo, ya que son restricciones de todos con todos.

4.2.4. Modelo columna

En este caso se considera la situación parecida, pero el arreglo tendrá las columnas de cada reina. Por ejemplo (2,3,1,4) indica que la reina de la fila 1 está en columna 2, la reina de la fila 2 está en columna 3, etc.



Ejemplo: N reinas

- X_j : columna donde se encuentra la reina de la fila j
con $j \in [1, 4]$ y $X_j \in [1, 4]$

En este caso se encapsulan las restricciones de la cantidad de reinas permitidas, y además, la restricción de que en una fila puede haber a lo más una reina.

El espacio de búsqueda entonces considera que hay 4 variables, que pueden tomar a 4 valores distintos. Esto es $4^4 = 256$. En caso de N reinas el espacio sería N^N

4.3. Sudoku

- Problema: El objetivo es llenar una cuadrícula de $n^2 \times n^2$ celdas dividida en secciones de $n \times n$, con valores entre 1 a n^2 , considerando algunos números ya dispuestos en algunas de las celdas (pistas). No se debe repetir ningún valor en una misma fila, columna o bloque. Un sudoku está bien planteado si la solución es única.

5	3			7				
6				1	9	5		
	9	8					6	
8				6				3
4			8	3				1
7				2			6	
	6				2	8		
		4	1	9			5	
			8			7	9	

$$x_{ijk} = \begin{cases} 1 & \text{la casilla } (i,j) \text{ contiene en valor } k \\ 0 & \text{si no} \end{cases}$$

- Cada celda debe tener un valor

$$\sum_{k=1}^{n^2} x_{ijk} = 1, \forall i, j \in \{1..n^2\}$$

- Un valor en cada columna

$$\sum_{i=1}^{n^2} x_{ijk} = 1, \forall j, k \in \{1..n^2\}$$

- Un valor en cada fila

$$\sum_{j=1}^{n^2} x_{ijk} = 1, \forall i, k \in \{1..n^2\}$$

- Un valor en cada subcuadrícula

$$\sum_{i=i_0}^{i_0+n-1} \sum_{j=j_0}^{j_0+n-1} x_{ijk} = 1, \forall k \in \{1..n^2\}, \forall i_0, \forall j_0 \in \{n * l + 1\}, l \in \{0..n-1\}$$

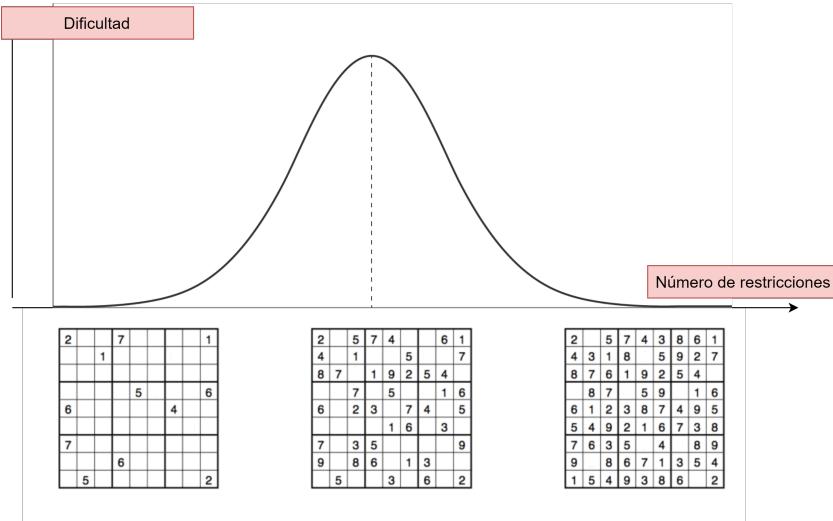
- Pistas

$$x_{ijk} = 1, \forall (i, j, k) \in Pistas$$

4.3.1. Fase de transición

Cuando existen muy pocas restricciones, la selección de valores es fácil ya que existen muchas soluciones posibles. Nuevamente, cuando el problema esta sobrestringido, la selección de valores también es fácil porque es guiada por las restricciones duras y existen pocas soluciones.

En el punto medio entre tener muchas y tener pocas soluciones, la dificultad de problema crece y el problema se encuentra en la **fase de transición** - peak de su dificultad.



4.4. Buscaminas

- Problema: Identificar las casillas que contienen una bomba a partir de la información desplegada por sus casillas adyacentes.

1			1	1	1	1	0	0
2						1	0	0
	3	2	1	1	1	1	1	1
			1	0	0	1		
1	1	1	0	1	3			
0	0	0	0	1				
0	0	0	0	1	2	4		
0	0	0	0	0	0	0	1	

■ 7 ■ 10 ■ 0 0:00

Modelo:

Variables:

$$X_i = \begin{cases} 1, & \text{hay bomba en la casilla oculta i} \\ 0, & \text{en el caso contrario} \end{cases}$$

Parámetros:

$$a_{ij} = \begin{cases} 1, & \text{casilla oculta i es adyacente a la casilla expuesta j} \\ 0, & \text{en el caso contrario} \end{cases}$$

- B - número de bombas
- V_j - valor de casilla expuesta j

Restricciones:

- Total de bombas: $\sum_i x_i = B$
- Valor de casilla expuesta es igual a la suma de bombas de casillas adyacentes y ocultas: $\sum_i x_i a_{ij} = V_j$

4.5. Car Sequencing Problem

- Secuencia de vehículos en una línea de ensamblaje
- Cada vehículo tiene una lista de opciones a ser instaladas (Radio, Techo plegable, Barras para llevar bicicletas, etc.).
- Cada opción es instalada por una estación diferente cuya capacidad de operación es limitada
- Por ejemplo, por cada q automóviles, sólo p_{barra} vehículos pueden requerir barras
- Cada combinación de opciones constituye una clase de vehículo
- Existe una demanda de vehículos de cada clase
- El objetivo es encontrar un orden en la secuencia de vehículos sin exceder la capacidad de cada estación y cumplir con la demanda



Modelo:

Variables:

$$X_{li} = \begin{cases} 1, & \text{en la posición } l \text{ hay un vehículo } i \\ 0, & \text{en el caso contrario} \end{cases}$$

Parámetros

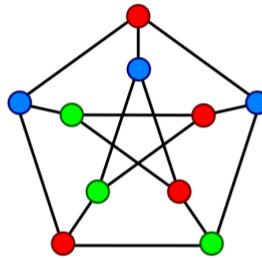
- C - número de clases
- nC_i - número de autos de clase C
- O - número de opciones
- p_j/q_j - la opción j alcanza a atender p_j de cada q_j autos seguidos
- $r_{ij} = \begin{cases} 1, & \text{clase } i \text{ requiere opción } j \\ 0, & \text{en el caso contrario} \end{cases}$

Restricciones:

- Se deben atender a todos los autos: $\sum_l^A x_{li} = nC_i, \forall i \in C$
- Restricción de atención. Cada opción j puede atender solo cada p_j de q_j autos: $\sum_{l=k}^{q_j+l} x_{li} r_{ij} \leq p_j, \forall j \in O, \forall k \leq A - q_j$

4.6. Coloreo de Grafos

- Asignar distintos colores a los vértices de un grafo de manera que ningún par de vértices adyacentes comparten el color.



- CSP
 - Considerando un conjunto de colores dado
- CSOP
 - Minimizando la cantidad de colores

Modelo:

Variables:

$$X_{ij} = \begin{cases} 1, & \text{a nodo } i \text{ se le asigna el color } j \\ 0, & \text{en el caso contrario} \end{cases}$$

Constantes:

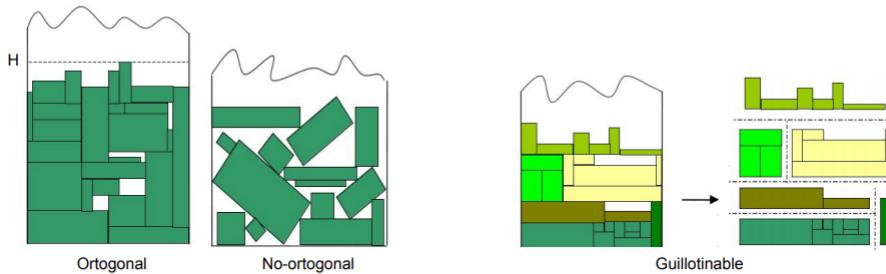
- $a_{ik} = \begin{cases} 1, & \text{nodo } i \text{ es adyacente al nodo } k \\ 0, & \text{en el caso contrario} \end{cases}$
- n - número de nodos
- m - número de colores

Restricciones:

- Cada nodo tiene asignado solo un color: $\sum_j x_{ij} = 1, \forall i$
- Si nodo i es adyacente al nodo k ($a_{ik} = 1$), entonces a lo más uno de estos puede ser asignado con el color j : $x_{ij} + x_{kj} \leq 1 + M(1 - a_{ik})$
- M es un valor arbitrario y muy grande

4.7. 2d space planning

- Un conjunto de rectángulos deben ser ubicados en un contenedor rectangular de mayor tamaño.



- CSP**
 - El contenedor tiene un ancho W y altura H
 - El objetivo es ubicar todos los rectángulos en el contenedor sin que se sobrepongan.
- CSOP**
 - El contenedor tiene un ancho W y altura infinita
 - El objetivo es minimizar la altura del contenedor luego de colocar todos los rectángulos.

Modelo:

Variables: (x_i, y_i) - coordenadas cartesianas de la esquina inferior del rectángulo i

$$l_{ij} = \begin{cases} 1, & \text{rectángulo } i \text{ está a la izquierda del rectángulo } j \\ 0, & \text{en el caso contrario} \end{cases}$$

$$b_{ij} = \begin{cases} 1, & \text{rectángulo } i \text{ está por debajo del rectángulo } j \\ 0, & \text{en el caso contrario} \end{cases}$$

Constantes:

- J - conjunto de rectángulos
- W - ancho de contenedor
- H - altura del contenedor
- w_i, h_i - ancho y altura del rectángulo i

Restricciones:

- Restricción de posición vertical de rectángulo: $0 \leq y_i \leq H - h_i$
- Restricción de posición horizontal de rectángulo: $0 \leq x_i \leq W - w_i$
- Posición relativa entre rectángulos: $l_{ij} + l_{ji} + b_{ij} + b_{ji} \geq 1$

- Si el rectángulo i esta a la izquierda de rectángulo j ($l_{ij} = 1$) entonces rectángulo j debe estar ubicado más a la derecha de este. $x_i + w_i \leq x_j + W(1 - l_{ij})$
- Si el rectángulo i esta a por debajo del rectángulo j ($b_{ij} = 1$) entonces rectángulo j debe estar ubicado más arriba de este. $y_i + h_i \leq y_j + H(1 - b_{ij})$

4.8. UCTP

Este problema trabaja con **restricciones duras** - aquellas que si o si deben ser cumplidas - y **restricciones blandas** (se desea que se cumplan, pero cuyo cumplimiento se mide dentro de la función objetivo).

Dado:

- Un conjunto de eventos
- Un conjunto de timeslots
- Un conjunto de salas
- Un conjunto de características
- Un conjunto de estudiantes que asisten a ciertos eventos en ciertas salas en ciertos timeslots

Se requiere:

- Cada evento debe ser asignado a una sala que posea las características y capacidad requerida
- Cada sala debe ser utilizada por a lo más un evento en cada timeslot
- Eventos con alumnos en común deben ser asignados a diferentes timeslots

Se desea:

- Los estudiantes no tengan eventos en el último timeslot de cada día
- Los estudiantes no tengan más de dos timeslots seguidos de eventos
- Los estudiantes no tengan un único evento en un día.



Aclaración 1: Hablaré de clases en vez de eventos (subíndice c para las clases) y reservar e para los estudiantes.

Aclaración 2: Se consideraron 9 timeslots diarios como en la USM, y sólo de lunes a viernes. Se asume que cada clase dura 1 timeslot (que en la USM serían dos bloques seguidos).

- Variables:

$$x_{cts} = \begin{cases} 1 & \text{si la clase } c \text{ se programa en el tiempo } t \text{ en la sala } s \\ 0 & \text{si no} \end{cases}$$

- Constantes:

$$a_{ec} = \begin{cases} 1 & \text{si el estudiante } e \text{ atiende la clase } c \\ 0 & \text{si no} \end{cases}$$

$$b_{sc} = \begin{cases} 1 & \text{si la sala } s \text{ posee las características requeridas por la clase } c \\ 0 & \text{si no} \end{cases}$$

■ Restricciones:

- Cada clase debe ser asignada a una sala que posea las características y capacidad requerida

$$\sum_s \sum_t b_{sc} \cdot x_{cts} = 1, \quad \forall c$$

- Cada sala debe ser utilizada por a lo más una clase en cada timeslot

$$\sum_c x_{cts} \leq 1, \quad \forall t, \forall s$$

- Clases con alumnos en común deben ser asignados a diferentes timeslots

$$\sum_c \sum_s a_{ec} \cdot x_{cts} \leq 1, \quad \forall e, \forall t$$

■ Restricciones blandas:

- Los estudiantes no tengan clases en el último timeslot de cada día.

Aclaración 3: Se usa el subíndice $d \in \{0, \dots, 4\}$ para contabilizar los días de la semana, así $9d + 8$ siempre corresponde al último timeslot de cada día.

$$S_1 = \sum_d \sum_s \sum_c \sum_e a_{ec} \cdot x_{c(9d+8)s}$$

- Los estudiantes no tengan más de dos timeslots seguidos de clases

$$S_2 = \sum_d \sum_e \sum_{k=9d}^{9d+6} h_{kde}$$

Aclaración 4: h_{kde} debería ser considerada otra variable del modelo.

$$h_{kde} = \begin{cases} 1 & \text{si } \sum_{n=0}^2 \sum_s \sum_c a_{ec} \cdot x_{c(k+n)s} = 3 \\ 0 & \text{si no} \end{cases}$$

- Los estudiantes no tengan una única clase en un día.

$$S_3 = \sum_d \sum_e q_{de}$$

Aclaración 5: q_{de} debería ser considerada otra variable del modelo.

$$q_{de} = \begin{cases} 1 & \text{si } \sum_{k=9d}^{9d+8} \sum_s \sum_c a_{ec} \cdot x_{cks} = 1 \\ 0 & \text{si no} \end{cases}$$

- Función objetivo: (Se minimiza la insatisfacción de las restricciones blandas)

$$\text{Mín } S_1 + S_2 + S_3$$

4.9. Preguntas de Examen/Control

- Respecto los tipos de restricciones:

V La insatisfacción de restricciones blandas debe ser considerada como función objetivo

Sí, restricciones blandas es algo que se desea tener y por eso la insatisfacción debe ser minimizada en la función objetivo

- F La insatisfacción de restricciones duras debe ser considerada como función objetivo

No, son las blandas

- V Las alternativas que cumplen restricciones duras se consideran factibles

Sí, si se cumplen las duras entonces es solución factible.

- F Las alternativas que cumplen restricciones blandas se consideran factibles

No, son las duras

2. La dificultad del problema de asignación cuadrática:

- V/F Aumenta al aumentar la cantidad de lugares

Puede aumentar ya que aumenta el espacio de búsqueda, pero depende de la fase de transición

- V/F Se considera fácil cuando hay muy pocas ubicaciones

Puede disminuir ya que el espacio de búsqueda decrece, pero también depende de la fase de transición

- V Aumenta si se consideran lugares pre-establecidos

Sí, si el problema entra en fase de transición puede ser más difícil asignar los lugares.

- F Puede aumentar si se consideran capacidades de ubicaciones

No, pero agrega restricciones.

3. La dificultad de un problema buscaminas:

- F Se considera difícil cuando hay una gran cantidad de casillas expuestas

No. Segundo la fase de transición se considera fácil ya que existen muchas restricciones para elegir la solución.

- V Se considera difícil cuando hay una gran cantidad de casillas expuestas

Sí, porque hay muchas restricciones y por ello, mucha información para elegir la solución

- F Aumenta a medida que aumenta la cantidad de casillas ocultas

No, depende. Si hay pocas expuestas (muchas ocultas), es fácil y si hay mucha expuestas (pocas ocultas), también lo es. El problema es difícil cuando se ubica en punto medio entre pocas y muchas casillas expuestas.

4. Considere la siguiente definición de variables para el problema de programación de cursos en Universidades:

$x_{cts} = 1$ si la clase c se programa en el tiempo t en la sala s, 0 si no. Y las constantes:

$a_{ec} = 1$ si la clase c se programa en el tiempo t en la sala s, 0 si no

La restricción: $\sum_s \sum_c a_{ec} \cdot X_{cts} \leq 1, \forall e, \forall t$

- V Significa que las clases que tienen alumnos en común deben asignarse a diferentes timeslots

Sí, es la definición de la restricción.

F Significa que las clases que tienen alumnos en común pueden asignarse a diferentes timeslots

No, no es que pueden, deben ser asignados a diferentes timeslots.

V Significa que para cada estudiante se debe programar a lo más una clase en cada timeslot

Sí porque para cada instante de tiempo se revisa que todas las clases a las que debe asistir el estudiante no sean más que 1

F Significa que para cada estudiante se debe programar a lo más una clase en cada timeslot

No, la restricción no verifica que la cantidad de clases del alumno sea igual a la cantidad de timeslots

5. Un CSOP:

F Puede tener solo restricciones unarias y binarias

No. También puede tener n-arias

■ Puede tener solo restricciones unarias y binarias

No, también puede tener unarias y binarias

V Define una función objetivo

Sí, define una función de optimización.

F Puede representarse como un grafo

No, no necesariamente por temas de optimización.

6. Considerando las siguientes dos versiones de definiciones de variables para el problema del vendedor viajero:

Definición 1: $x_{ij} = 1$, si se va de la ciudad i a la ciudad j, 0 si no

Definición 2: $x_{ijk} = 1$, si se va de la ciudad i a la ciudad j en el paso k, 0 si no. Es cierto que:

V La Definición 2 permite formular una restricción que la Definición 1 por sí sola no

Sí, las restricciones de primera definición son de tsp clásico, y de segunda trabajaran con dobles sumatorias, se definen otras restricciones para el doble paso.

F La Definición 2 encapsula una restricción que la Definición 1 no

V La Definición 1 requiere variables adicionales para formular restricciones de secuencia

Sí, se requiere de la variable u_i para indicar la posición de la ciudad i en el tour. La definición 2 usará la variable existente para definir las restricciones de orden.

7. Considerando las siguientes dos versiones de definiciones de variables para el Jobshop:

Definición 1: X_{ij} = Instante de tiempo en que el trabajo i comienza a usar el procesador j

Definición 2: $X_{ijk} = 1$, si el trabajo i comienza a usar el procesador j en el instante k, 0 si no

- V La Definición 2 requiere formular una restricción que la Definición 1 no requiere
Sí, habrá que definir la relación de instantes ya que el trabajo debe ser asignado en algún momento
- V La Definición 1 encapsula una restricción que la Definición 2 no
Sí, ya que habrá un único instante en que se asigna el trabajo i al procesador j
- V La Definición 1 requiere variables adicionales para formular restricciones de secuencia
Sí, se requiere de la variable a_{ikl} donde esta será 1 si el trabajo i usa procesador k antes del trabajo l
- F La Definición 2 encapsula una restricción que la Definición 1 no

8. Considerando las siguientes dos versiones de definiciones de variables para el problema de planificación de espacios en dos dimensiones:

Definición 1: X_i : coordenada horizontal de la esquina inferior izquierda del rectángulo i

Definición 2: $X_{ij} = 1$ si la coordenada horizontal de la esquina inferior izquierda del rectángulo i está en la coordenada j, 0 si no, es cierto que:

- La Definición 2 requiere formular una restricción que la Definición 1 no requiere
- La Definición 1 posee un espacio de búsqueda menor que la Definición 2
- La Definición 2 posee un espacio de búsqueda menor que la Definición 1
- La Definición 2 encapsula una restricción que la Definición 1 no

9. Considerando las siguientes dos versiones de definiciones de variables para el problema de las n-reinas:

Definición 1: X_i : casilla en que se ubica la reina i

Definición 2: X_{ij} : 1 si se ubica una reina en la casilla i,j, 0 si no, es cierto que:

- V La Definición 2 requiere formular una restricción que la Definición 1 no requiere
Sí, de cantidad de reinas.

- V La Definición 1 posee un espacio de búsqueda menor que la Definición 2

No. La definición 1 tiene el espacio de $(N^2)^N$ y la definición 2 tiene el espacio 2^{N^2} . Es el mismo espacio.

- F La Definición 2 encapsula una restricción que la Definición 1 no

No, al revés.

- V La Definición 1 requiere definir una restricción adicional que la Definición 2 no requiere

Sí, el valor de i está restringido por la cantidad de reinas a ubicar. No habrán más variables que reinas a posicionar.

10. En el problema de n-reinas

- V Se pueden proponer distintos modelos para el mismo problema
Sí, modelo binario, casilla, fila, columna
- V El mejor modelo es el que reduce el espacio de búsqueda
Sí, exacto
- F El mejor modelo es el que permite definir todas las restricciones
No, es el que reduce más el espacio de búsqueda
- V Se puede reducir el espacio al introducir variables encapsuladoras
Sí.
- V Se puede reducir el espacio al introducir tantas variables como filas *Sí, porque modelo fila encapsula restricciones y reduce el espacio.*
-
11. En el problema TSP
- V Se puede usar tanto variables binarias como enteras
Sí, binarias para tomar camino de i a j , enteras para posición en tour
- V Permite evitar ciclos introduciendo variables adicionales
Sí, pero también requiere usarlas en restricciones.
- F Reduce el espacio de búsqueda al evitar ciclos *No, solo descarta soluciones infactibles*
- La versión asimétrica tiene costo c_{ij} distinto a c_{ji}
Sí, por eso es asimétrica.
-
12. En el problema Job Shop
- F Se debe respetar la prioridad de uso de procesador
No, no existe este concepto
- V Se debe respetar el orden de uso de procesador por cada operación
Sí, debe existir orden de precedencia
- F Se debe optimizar el tiempo que se usa cada procesador
No, se quiere terminar todos los trabajos en mínimo tiempo
- V Se debe respetar el uso exclusivo de procesador por cada trabajo
Sí, debe existir restricción disyuntiva.

5. Técnicas de Filtro y Consistencia

Proceso de filtrado consiste en recibir el problema inicial P y transformarlo en el problema filtrado P' . Ambos problemas tienen la misma cantidad de soluciones.

El proceso de filtrado

- Elimina elementos que con seguridad no pueden ser parte de solución
- Espera simplificar el problema ya que reduce el espacio de búsqueda
- No existe pérdida de soluciones - problema P es equivalente a P'
- Se puede detectar ausencia de solución

Por **consistencia** se entiende el grado de compatibilidad entre valores de dominios $a \in D_i$ y las restricciones C .

1. Consistencia Local - Nodos, Arcos, Caminos, k-consistencia
2. Consistencia Global

Para cada problema se define:

- Conjunto de variables X
- Dominios de variables D - posibles valores de cada X_i
- Conjunto de restricciones C
- Conjunto de relaciones R - combinaciones de valores que satisfacen C_i

5.1. Nodo consistencia

Consistencia de nodos es el primer nivel de consistencia. Este algoritmo permite revisar el cumplimiento de las **restricciones unarias** dentro de cada nodo. Su complejidad es $O(n)$, lineal respecto a la cantidad de variables X_i .

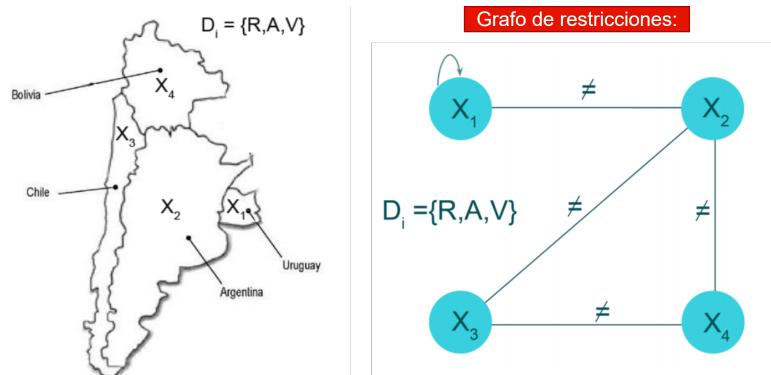
Algorithm 1: Nodo Consistencia(X,D,C)

```

for each  $X_i \in X$  do
  for each  $a \in D_i$ ;                                // por cada valor  $a$  de dominio de  $X_i$ 
    do
      if  $a \notin R$ ;                                // si valor  $a$  no satisface la relación
      then
         $D_i = D_i - \{a\}$ ;                         // se borra el valor de  $a$  del dominio
      end
    end

```

Para el problema de coloreo de mapas con el siguiente grafo de restricciones, podemos aplicar el algoritmo de Nodo Consistencia. Se sabe que en nodo X_1 existe la restricción unaria de que Uruguay debe ser Rojo : $X_1 = R$.



Corriendo el algoritmo se obtiene que:

- Para X_1 : Se eliminan los valores A, V del dominio por no cumplir con la relación $R_1 : X_1 = \{R\}$. El dominio final es: $D_1 = \{R\}$
- Para X_2, X_3, X_4 los dominios se mantienen iguales: $D_{2,3,4} = \{R, A, V\}$.

Esto transforma el problema inicial P en el siguiente problema equivalente P' :

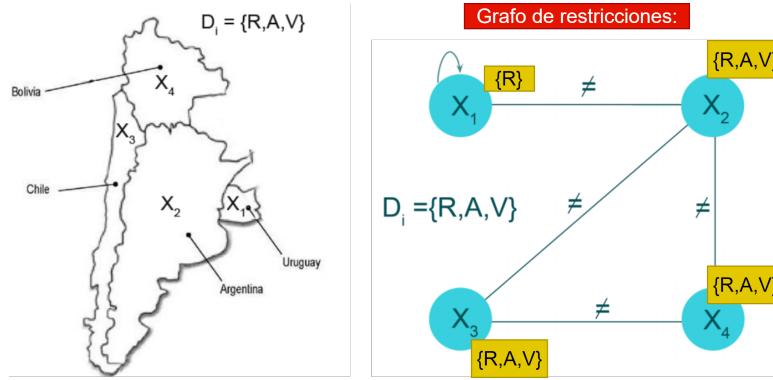


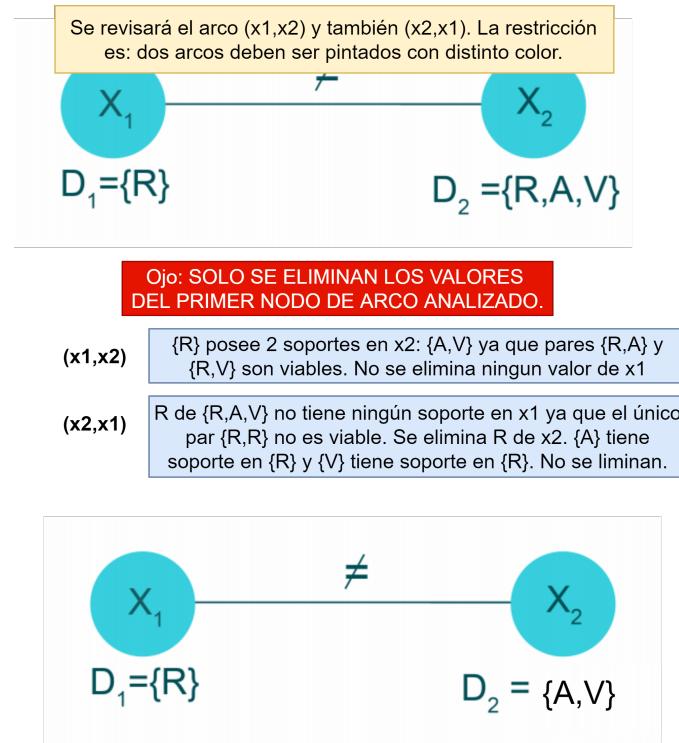
Figura 1: Coloreo de mapas con Nodo Consistencia

5.2. Arco consistencia

Arco consistencia se aplica **después de aplicar nodo consistencia**. El algoritmo consiste en revisar los arcos de ida y vuelta (x_1, x_2) y (x_2, x_1) .

Tener arco consistencia no implica que el problema tenga solución.

- Se considera que un valor a en x_1 es viable si posee **un soporte** en x_2 (y viceversa). Se considera que un soporte es un tal valor b que el par (a, b) cumpla con las restricciones del arco.



Cabe destacar que si un problema es arco-consistente, no necesariamente tiene solución. Sin embargo, si el problema tiene solución entonces es, como mínimo es arco consistente. Además, si el problema es arco-consistente y los dominios de las variables tienen solo un valor, entonces existe una **única solución**.

5.2.1. Procedimiento REVISE

El procedimiento REVISE recibe dos nodos que forman un arco, y se revisa si cada valor a del dominio de nodo X_i tenga un soporte en algún valor del dominio de nodo X_j . Si se encuentra tal a que no tenga soporte, este valor se borra del dominio D_i .

Algorithm 2: REVISE(X_i, X_j)

```

DELETE ← FALSE;           // flag para saber si hubo cambio en dominio  $X_i$ 
for each  $a \in D_i$ ;      // por cada valor  $a$  de dominio de  $X_i$ 
do
  if there is no such  $b$  in  $D_j$  such as  $(a, b)$  is consistent then
    |  $D_i = D_i - \{a\}$ ;          // se borra el valor de  $a$  del dominio
  end
return DELETE;
  
```

5.2.2. AC-1

El algoritmo AC-1 como primer paso crea una cola con todos los arcos posibles del grafo, en los dos sentidos (i,j) y (j,i) . Después en el ciclo se revisa que para un par de nodos (X_i, X_j) el procedimiento REVISE cambió el dominio de X_i . Si esto es cierto, **de nuevo se revisan todos los arcos**. Este procedimiento es **costoso** debido a la repetición de ciclo

para todos los arcos cada vez que se introduce algún cambio en algún dominio.

Algorithm 3: AC-1

```

Q ← {( $X_i, X_j \in arcs(G), i \neq j$ };  

repeat  

  CHANGE ← FALSE;  

  foreach ( $X_i, X_j$ ) ∈ Q do  

    | CHANGE ← (REVISE( $X_i, X_j$ ) or CHANGE)  

  end  

until not(CHANGE);
  
```

5.2.3. AC-3

El algoritmo AC-3 se propone para reducir el costo del algoritmo AC-1. AC-3 es una versión más eficiente en cuanto a la detección de los arcos que deben ser re-revisados.

Algorithm 4: AC-3

```

Q ← {( $X_i, X_j \in arcs(G), i \neq j$ };  

while Q not empty;  

do  

  select and delete any arc ( $X_k, X_m$ ) from Q;  

  if (REVISE( $X_k, X_m$ ));  

    then  

      | Q ← Q ∪ {( $X_i, X_k$ ) such that ( $X_i, X_k \in arcs(G), i \neq k, i \neq m$ } ;  

      |   agregan a la cola Q aquellos arcos donde  $X_i$  tiene soporte en  $X_k$   

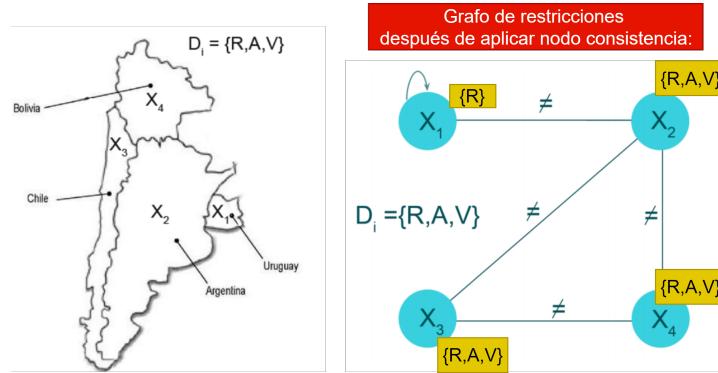
end
  
```

AC-3 vuelve a agregar a la cola Q los arcos de tipo (x_i, x_k) . No se agrega el arco inverso al actual analizado. Es decir, si se analiza (x_2, x_1) y produce un cambio, no debemos agregar (x_i, x_2) donde $i = 1$.

Volviendo al ejercicio de coloreo de mapa, después de aplicar NC podemos aplicar AC. Definamos el modelo completo antes de proceder:

- Variable: $X_i, i \in \{1, 2, 3, 4\}$ - un país
- Dominio de variables $D_i = \{R, A, V\}$
- Restricciones:
 - $X_1 = R$
 - $X_1 \neq X_2$
 - $X_2 \neq X_3$
 - $X_2 \neq X_4$
 - $X_3 \neq X_4$

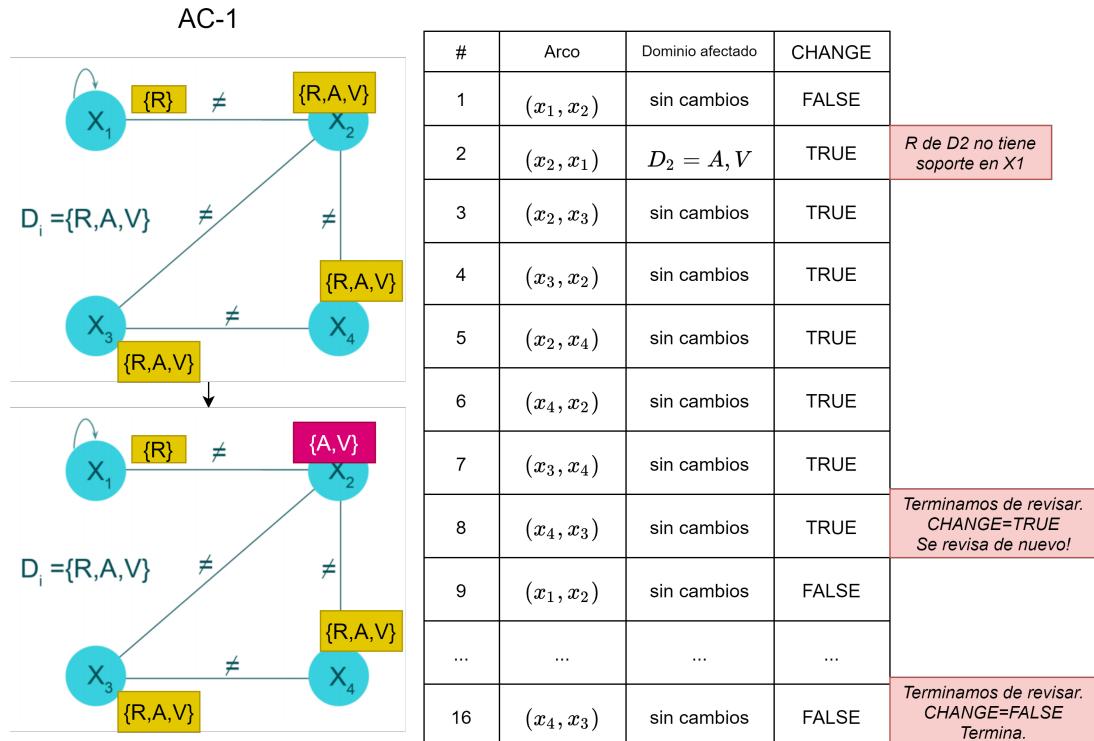
Después de aplicar NC obtuvimos los siguientes dominios (el único cambio en X_1):



Tenemos que formar una cola Q con todos los arcos. Esta sería:

$$Q = \{(x_1, x_2), (x_2, x_1), (x_2, x_3), (x_3, x_2), (x_2, x_4), (x_4, x_2), (x_3, x_4), (x_4, x_3)\}$$

Para AC-1 tenemos la siguiente situación:



$$Q = \{(x_1, x_2), (x_2, x_1), (x_2, x_3), (x_3, x_2), (x_2, x_4), (x_4, x_2), (x_3, x_4), (x_4, x_3)\}$$

Figura 2: Resultado de AC-1

Para AC-3 tenemos la siguiente situación:

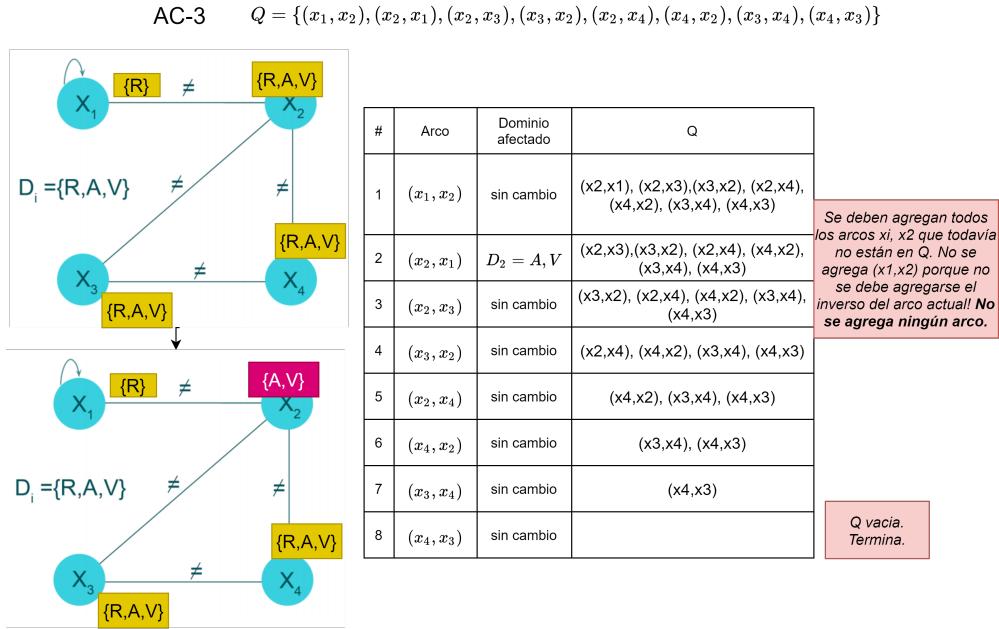


Figura 3: Resultado de AC-3

Se observa que AC-3 es menos caro que AC-1. Existe también el algoritmo AC-5 que es menor en complejidad pero requiere características de biyección y monotonidad.

5.3. Camino consistencia

Se dice que un grafo es camino consistente si para un par de variables (X_i, X_j) por cada par de valores (a, b) consistentes (viables) existe un c en todos los nodos X_k conectados con el par de nodos (X_i, X_j) tal que (a, c) cumple la restricción R_{ik} y (b, c) cumple la restricción R_{jk} .

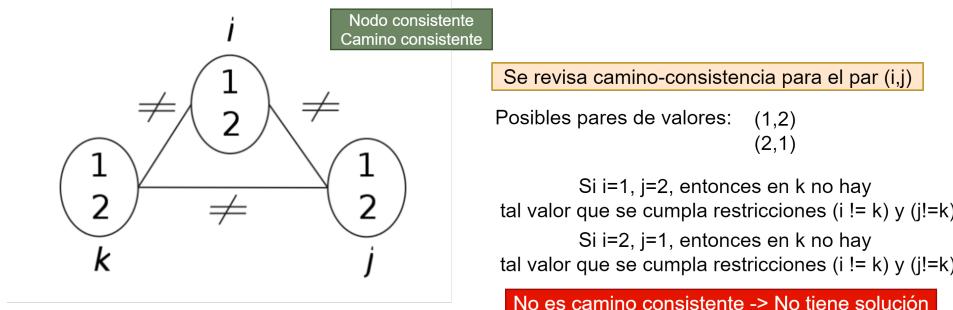


Figura 4: Ejemplo de aplicación de Camino Consistencia

- Un problema es camino consistente ssi: Todos los pares de variables son camino consistentes
- Es más complejo de implementar que AC
- Su complejidad es entre $O(n^3d^5)$ y $O(n^3d^3)$

- Su aplicación puede cambiar la topología de grafo agregando más restricciones
- Es poco usado

5.4. K consistencia

- 1-consistencia - consistencia de 1 nodo
- 2-consistencia - es arco consistencia
- 3-consistencia - es camino consistencia
- k-consistencia - consistencia entre k nodos

Una red es k-consistente ssi dada cualquier instanciación de $k - 1$ variables, que satisfagan todas las restricciones entre ellas, existe al menos una instanciación de una variable k , tal que se satisfacen las restricciones entre las k variables

Por ejemplo, 3-consistencia toma una instanciación de $k-1 = 3-1 = 2$ nodos y busca la instanciación de la 3ra variable que cumpla con restricciones.

Además, se supone que **antes del chequeo de k consistencia, se hace el chequeo de j-consistencia** con $\forall j < k$. Es decir, para 3-consistencia (camino) se debe hacer primero el NC y el AC (1 y 2 consistencia, respectivamente).

Si el problema es fuertemente k-consistente, entonces tiene solución.

5.5. Preguntas de Examen/Control

-
1. Respecto a las técnicas de filtro y consistencia es cierto que:

V **Nodo consistencia** revisa restricciones unarias

Sí. *Revisa restricciones de nodos (auto restricciones)*

V **Arco consistencia** revisa restricciones binarias

Sí. *Revisa pares de nodos.*

F **Trayectoria consistencia** revisa restricciones n-arias

No. *Revisa restricciones binarias entre pares de variables.*

F **K-consistencia** revisa restricciones n-arias

No. *Revisa restricciones binarias entre varias variables.*

2. Los algoritmos AC-1 y AC-3

V **Pueden revisar la misma cantidad de arcos en algunos problemas**

Sí. *Un ejemplo trivial es cuando el problema ya es arco consistente - ambos algoritmos revisarán todos los arcos existentes en el grafo de restricciones una vez, sin hacer ningún cambio a los dominios.*

V **Trabajan con problemas de satisfacción de restricciones binarios**

Sí. *AC-1 y AC-3 analizan los arcos entre dos nodos, donde el arco representa la relación de restricción binaria que existe entre estos.*

V Son capaces de simplificar el problemas

Sí, ya que permiten filtrar y reducir los espacios de búsqueda. Básicamente, un problema P se transforma en un problema P' equivalente pero más simple

F Eligen el conjunto de arcos que se vuelven a revisar después de un filtro

No. AC-1 vuelve a revisar todos los arcos si se produjo un cambio en alguno de los dominios, no tiene ningún filtro. AC-3 efectivamente añade solo aquellos arcos que estén directamente conectados con el dominio que fue modificado.

3. Si un problema tiene solución entonces:

V Existe una versión k -consistente para $k = n$

Sí. Si el problema tiene solución, entonces en cada dominio existe consistencia respecto a los otros dominios. El grafo de restricciones se vuelve fuertemente k -consistente.

V Las k consistencias revisarán sub-conjuntos de variables respecto a sus restricciones binarias

Sí, ya que k -consistencia no revisa restricciones n -arias.

V Existe una versión k -consistente para cada valor de k , $k < n$

Sí. Si se cumple que el problema se puede hacer k -consistente ya que tiene solución, entonces es consistente a los niveles más bajos (nodo, arco, camino, etc)

F Las versiones de k -consistencia para $k > 2$, requieren que el problema posea restricciones n -arias

No. Un contraejemplo trivial es camino-consistencia ($k=3$) en un grafo completo, donde las restricciones son solamente binarias. Además, k -consistencia revisa solamente las restricciones binarias.

F Aplicar arco consistencia no genera cambios

No. No se puede asegurar

F Los dominios de todas las variables tienen un solo valor

No. No necesariamente.

4. Si un problema tiene UNA SOLA solución entonces:

V Existe una versión arco consistente para el problema

Sí. Si el problema tiene solución, tiene que tener una versión arco-consistente. Si tiene solo una solución, entonces los dominios tendrán solo un valor.

V Existe una versión k -consistente (con $k <$ cantidad de variables) para el problema

Sí. Si el problema tiene solución debe ser k -consistente

V Posterior a k -consistencia (con $k =$ cantidad de variables) los dominios de todas las variables poseen un único valor

Sí. Si existe n -consistencia y una única solución, entonces en todos los dominios habrá solamente un valor posible que puede tomar la variable asociada a este dominio.

- F Posterior a k-consistencia (con $k <$ cantidad de variables) los dominios de todas las variables poseen un único valor

No. No se puede asegurar, pero es posible

5. Si un problema tiene NO TIENE solución entonces:

- F No existe una versión arco consistente para el problema

No. En clase hemos visto un problema de grafo completo con 3 nodos, donde existía arco-consistencia entre todos los nodos, pero no había solución debido a que el grafo no era camino consistente.

- V Puede existir una versión k-consistente del problema si $k < n$

Sí. El ejemplo anterior de grafo completo indica que existe versión consistente para $k=2 < 3$. Que no existe solución solamente significa que no se cumple la k-consistencia para $k=n$.

- F Puede existir una versión k-consistente del problema si $n = k$

No. El problema entonces tendría solución.

- F Existe una versión arco consistente del problema

No. No se puede asegurar que siempre exista arco consistencia - pero es posible.

6. Si un problema posee un grafo totalmente conexo:

- F Aplicar AC-3 será más eficiente que AC-1

No. AC-3 vuelve a agregar los arcos que implican la relación entre los dominios de las variables con el dominio recientemente filtrado. Como es un grafo conexo, se volverá a agregar toda la cola, lo cual es lo mismo que hacer AC-1 pero usa más espacio de memoria.

- F CBJ generará el mismo árbol de búsqueda que BT

No. CBJ usará los conjuntos de conflictos para no revisar tantas instanciaciones cuando se entra en conflicto.

- V GBJ generará el mismo árbol de búsqueda que BT

Sí. GBJ usa el salto inteligente hacia la variable más recientemente instanciada y unida por restricción. En grafo completo todo está unido con todo, por lo que GBJ saltará a la variable anterior. Eso es lo mismo que haría BT cronológico.

- V Puede considerarse fácil de resolver de acuerdo a la fase de transición

Sí. La fase de transición indica que si el problema está muy restringido (sobrerestringido, como en el grafo completo), entonces existen tan pocas configuraciones posibles que el problema se vuelve fácil de resolver

7. Los algoritmos AC-1 y AC-3

- V Son capaces de reducir el espacio de búsqueda

Sí. AC-1 y AC-3 se usan para filtrar los dominios y reducir el espacio de búsqueda

- V Usan el mismo procedimiento REVISE

Sí. Ambos algoritmos ocupan REVISE para saber si se produjo un filtrado de dominio

V Generan la misma cola de arcos

Sí. Ambos inicialmente generan la misma cola que contiene todos los arcos del grafo en ambos sentidos. AC-3 después modifica esta cola, y AC-1 no.

F Quitar arcos de la cola en cada paso No. Esto solo se aplica a AC-3

8. Dado X_1, X_2 con $D_1 = D_2 = \{1, 2\}$ y $X_1 + X_2 < 5$

F Aplicar arco consistencia reduce el dominio de solo X_2

F Aplicar arco consistencia reduce el dominio de solo X_1

F Aplicar arco consistencia reduce ambos dominios

V Aplicar arco consistencia no genera ningún cambio

Sí. No se genera ningún cambio porque cada valor de D_1 tiene soporte en D_2 tal que la suma es menor que 5. Lo mismo cumple al revés, por lo que este par de arcos es arco-consistente y no requiere modificar los dominios.

9. Cuando se aplica REVISE(x_k, x_m)

F Se puede reducir el dominio de x_m

V Se revisa cada valor del dominio x_k para verificar si existe un valor en dominio de x_m que permite satisfacer la restricción que los une

Sí. Se revisa si cada valor de dominio de x_k tenga un soporte o más en el dominio de x_m

F Se revisa cada valor del dominio x_m para verificar si existe un valor en dominio de x_k que permite satisfacer la restricción que los une

V Se puede reducir el dominio de x_k

Sí. Se puede reducir solamente el dominio de x_k

V Retorna TRUE si se filtro en dominio de x_k

Sí. Retorna TRUE si se produjo un cambio en dominio, FALSE en caso contrario

10. Respecto a las técnicas de filtrado y consistencia es cierto que

F k-consistencia revisa restricciones n-arias

V Permiten, bajo ciertas condiciones, detectar la ausencia de solución

Sí. Si los dominios quedan filtrados de tal modo que hayan dominios vacíos, entonces no hay solución.

F Permiten reducir el espacio de búsqueda reduciendo el número de variables

No. Nunca se reducen las variables, sino sus dominios.

V Se aplican a problemas de restricciones binarias

Sí. Trabajan con restricciones binarias.

F Permiten siempre detectar la ausencia de solución

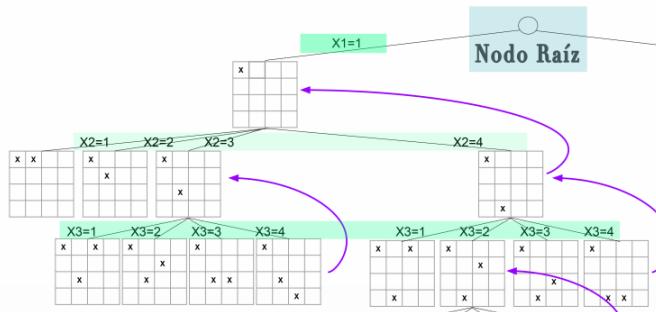
No. Depende - un AC-3 puede detectar que un grafo sea consistente, pero el otro filtrado puede decir que el grafo no es camino consistente y por ello, no tiene solución. Depende de nivel de consistencia que se chequea.

6. Técnicas Completas

Las técnicas de búsqueda completa permiten responder las siguientes preguntas: ¿tiene problema solución? ¿cuál es? ¿cuántas soluciones hay? ¿cuántos valores puede tomar una variable? ¿es X solución? ¿cuál es la mejor solución?

En búsqueda completa se utiliza la estructura de árbol de búsqueda.

- Nodo raíz es nodo de partida
- En cada nivel se hace una instanciación de variable
- Las hojas son las soluciones
- Hay tantos niveles de árbol como variables + 1
- Un nodo puede tener tantos descendientes como es el dominio de cada variable



Estructura

- Un árbol de búsqueda posee tantos niveles como variables tiene el problema + 1
- Un nodo puede tener tantos descendientes por nivel como valores tenga el dominio de cada variable

Figura 5: Árbol de búsqueda

A pesar de ser llamadas técnicas completas, **los algoritmos completos no necesariamente observan todo el espacio de búsqueda** ya que muchas veces son capaces de ver en qué punto se produce un conflicto y por ende, no vale la pena seguir instanciando.

6.1. Look-Back: Backtracking

En cada nivel busca un valor dentro del dominio que no genere conflicto con las variables ya instanciadas. La búsqueda es **cronológica** - si se encuentra un conflicto, se vuelve a la última variable instanciada.

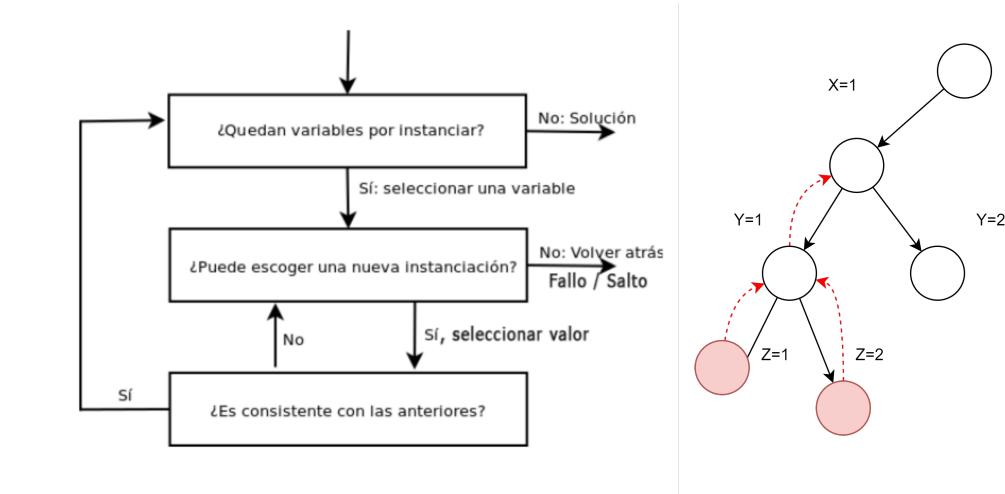
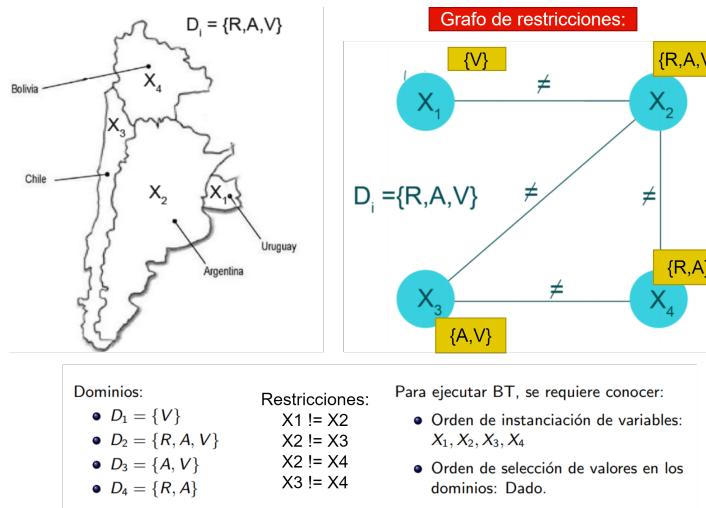


Figura 6: Algoritmo de Backtracking

Para ejecutar el BT se deben definir:

- El orden de selección de las variables
- El orden de selección de los valores en los dominios de cada variable

Para el problema de **coloreo de mapas** si se tiene la siguiente instancia:



Aplicando BT obtenemos el siguiente árbol:

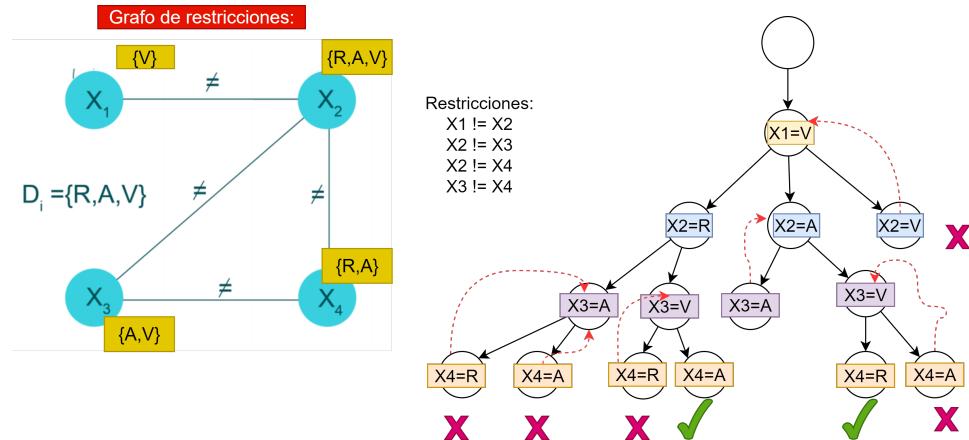
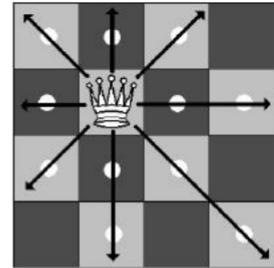


Figura 7: Resultado de aplicar BT en el colooreo de mapas

Otro problema en el cual es posible aplicar la búsqueda completa es N-Reinas:

- **Variables:**
 X_i : fila en que se ubica la reina de la columna i
- **Dominios:**
 $D_i: \{1, 2, 3, 4\}$
- **Restricciones:**

$$X_i \neq X_j, \quad \forall i \neq j$$



$$|X_i - X_j| \neq |i - j|, \quad \forall i \neq j$$

Requerimiento:

- Orden de instanciación de variables: X_1, X_2, X_3, X_4
- Orden de selección de valores en los dominios: $\{1, 2, 3, 4\}$

Figura 8: Ejemplo de N-Reinas para N=4

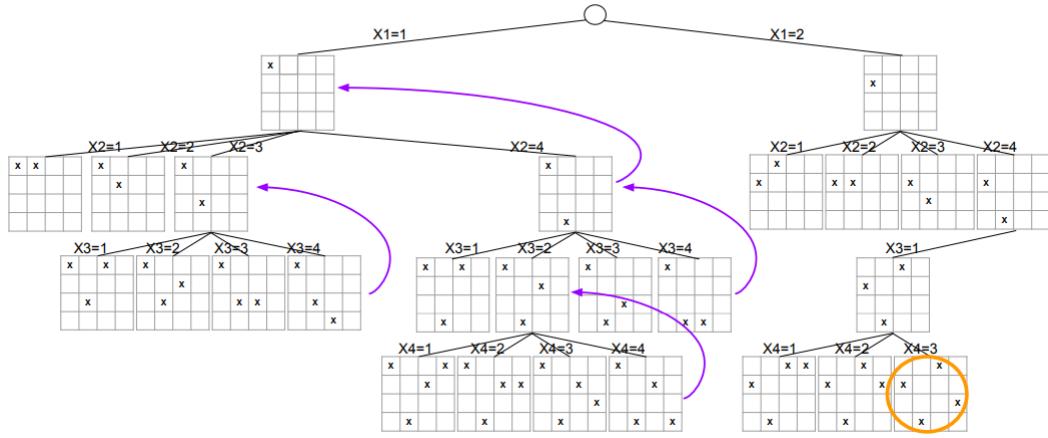


Figura 9: Árbol BT para 4-Reinas

6.2. Trashing

Es la situación cuando el algoritmo falla repetidamente por la misma razón. Entre razones de trashing se encuentran:

- Falta aplicar 1-consistencia (nodo consistencia)
- Falta aplicar 2-consistencia (arco consistencia)

BT puede sufrir de trashing, por lo que se introducen dos técnicas - GBJ y CBJ - para hacer un **salto inteligente**. Ambas técnicas reducen la cantidad de instanciaciones, sin embargo igualmente pueden entrar en trashing.

6.3. BT+GBJ

GBJ = Graph Based BackJumping o Retrono guiado el grafo de restricciones.

- En el caso de **fallo** (cuando ningún valor de una variable sirve para no generar conflictos) se regresa a la variable conectada por una restricciones **más recientemente instanciada en el árbol**. Básicamente se vuelve a la variable más cercana la cual tiene una posible relación con el fallo.
- Volver a una variable X_i implica instanciarla con otro valor.
- Es un buen método para grafos de restricciones poco densos

Cabe destacar que BT+GBJ hace saltos más grandes que BT normal. BT siempre retorna a la variable más recientemente instanciada, mientras que BT+GBJ usa más información y retorna a la variable más recientemente instanciada y que tenga alguna restricción con la variable actual.

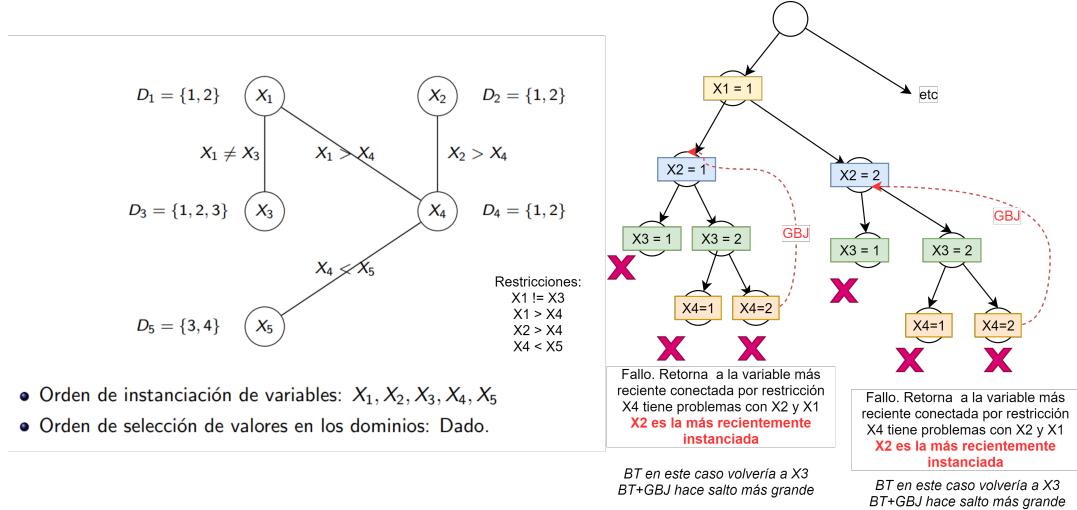


Figura 10: Ejemplo de saltos de GBJ

Como se observa, al aplicar GBJ **sigue existiendo trashing** - en el ejemplo analizado se repite el mismo fallo para cada uno de los valores en el dominio de X_2 . GBJ va a ser igual a BT en un grafo conexo, debido a que las restricciones de todo con todo harán que GBJ se comporta como un BT cronológico al volver a la reciente variable conectada por restricción que sería variable recién instanciada.

6.4. BT+CBJ

CBJ = Conflict Based BackJumping o Retorno guiado por conflictos.

- Se introduce más información - para cada una de las variables se guarda **conjunto de conflictos** $Conf(x_i)$
- Por cada valor inconsistente se registra en $Conf(x_i)$ la variable **más prematuramente instanciada y en conflicto** con la instanciación actual.
- La variable más prematura corresponde a aquella **más arriba en el árbol (más cercana a la raíz)**
- Cuando existe un fallo (no quedan valores para intentar a instanciar cierta variable), el conjunto entrega las razones de conflicto. El punto de retorno es la variable **más recientemente instanciada en** $Conf(x_i)$
- Al hacer salto se actualiza $Conf(x_i)$ hasta el punto donde llega
- Volver a una variable X_i implica instanciarla con otro valor.

Nota: GBJ es más eficiente que CBJ cuando los conflictos se producen en la variable más recientemente instanciada y conectada a la variable actual (ya que CBJ en este caso usa más recursos pero actúa como GBJ)

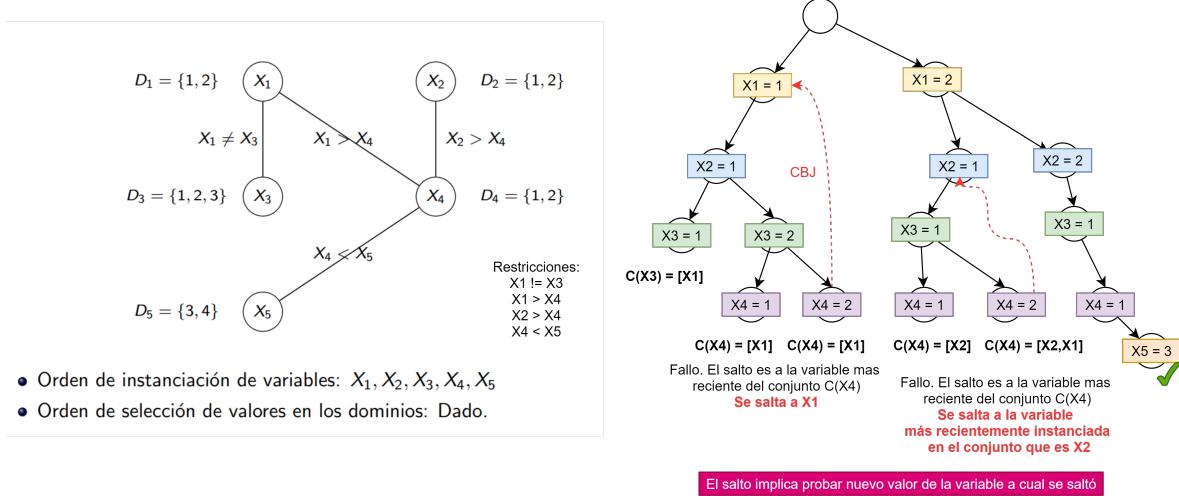


Figura 11: Ejemplo de saltos de CBJ

6.5. Look Ahead

La técnica de look-ahead consiste en aplicación de un filtro que permite filtrar el espacio de búsqueda que falta instanciar. En caso de que no existan más valores posibles para instanciar, el algoritmo termina su ejecución.

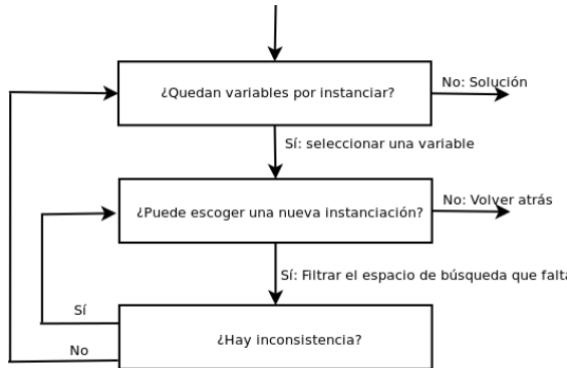


Figura 12: Idea de Look Ahead

6.6. Look-Ahead: FC

El algoritmo Forward Checking al instanciar una variable $X_i = a$, revisa todas las variables conectadas X_i a través de una restricción y filtra sus dominios de modo tal que se eliminen los valores incompatibles con $X_i = a$.

- Forward Checking

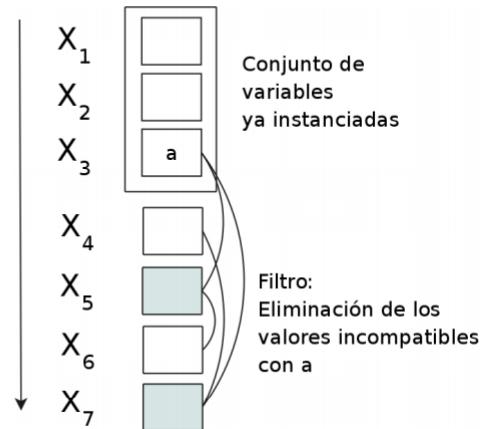
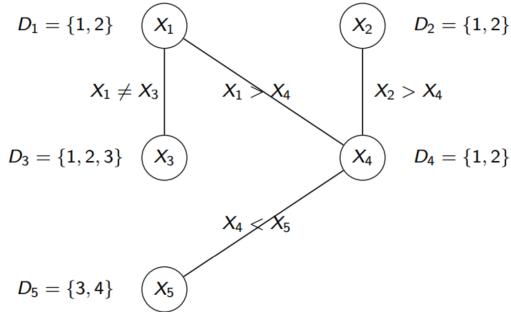


Figura 13: Idea de Forward Checking

Cabe destacar que los chequeos son **sólo hacia delante**. Si el orden de instanciación es X_1, X_2, X_3 entonces solamente se puede hacer chequeo tipo X_1 a X_3 pero no el X_3 a X_1 , asumiendo que estas dos variables están conectadas por una restricción.



- Orden de instanciación de variables: X_1, X_2, X_3, X_4, X_5
- Orden de selección de valores en los dominios: Dado.

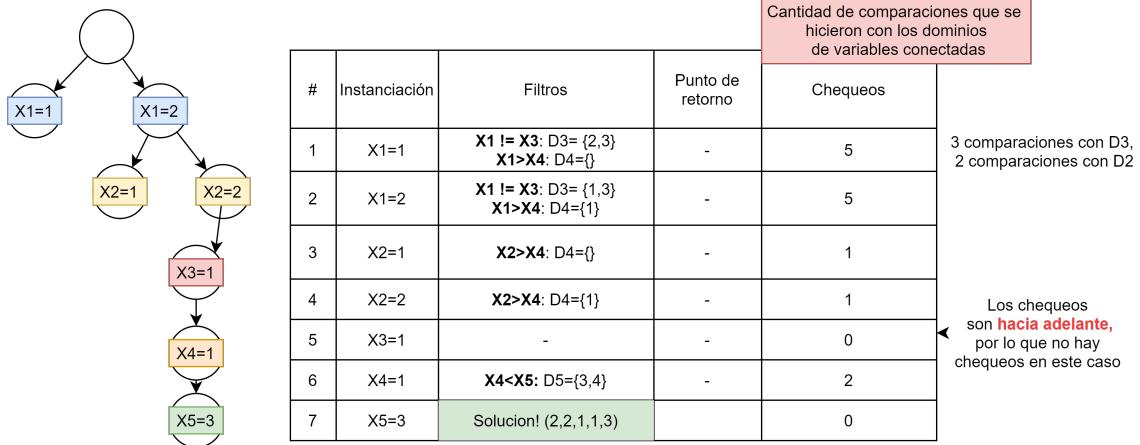


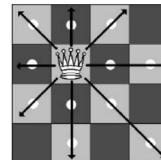
Figura 14: Ejemplo de FC

FC puede ser aplicado en el problema de N-Reinas:

Forward Checking 4 reinas

- Variables:
 X_i : fila en que se ubica la reina de la columna i
- Dominios:
 $D_i: \{1, 2, 3, 4\}$
- Restricciones:

$$X_i \neq X_j, \quad \forall i \neq j$$



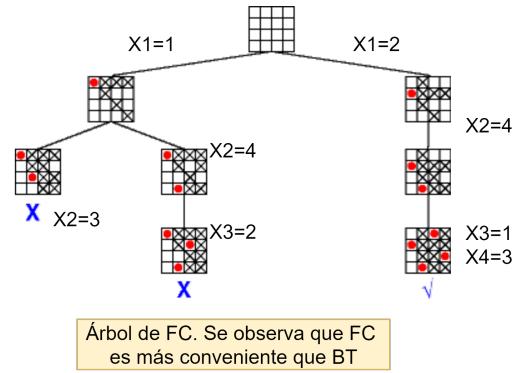
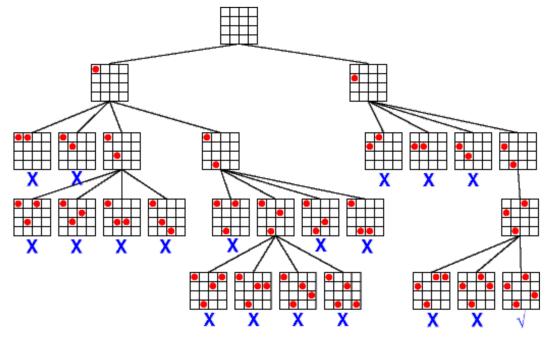
$$|X_i - X_j| \neq |i - j|, \quad \forall i \neq j$$

Requerimiento:

- Orden de instanciación de variables: X_1, X_2, X_3, X_4
- Orden de selección de valores en los dominios: $\{1, 2, 3, 4\}$

X

#	Instanciación	Filtros	Punto de retorno	Chequeos
1	$X_1 = 1$	$X_1 \neq X_2 : D_2 = \{2, 3, 4\}$ $X_1 \neq X_3 : D_3 = \{2, 3, 4\}$ $X_1 \neq X_4 : D_4 = \{2, 3, 4\}$ $ X_1-X_2 \neq 1-2 : D_2 = \{3, 4\}$ $ X_1-X_3 \neq 1-3 : D_3 = \{2, 4\}$ $ X_1-X_4 \neq 1-4 : D_4 = \{2, 3\}$	-	24
2	$X_2 = 3$	$X_2 \neq X_3 : D_3 = \{2, 4\}$ $X_2 \neq X_4 : D_4 = \{2\}$ $ X_2-X_3 \neq 2-3 : D_3 = \{2, 4\}$ $ X_2-X_4 \neq 2-4 : D_4 = \{\}$	-	8
3	$X_2 = 4$	$X_2 \neq X_3 : D_3 = \{2\}$ $X_2 \neq X_4 : D_4 = \{2, 3\}$ $ X_2-X_3 \neq 2-3 : D_3 = \{2\}$ $ X_2-X_4 \neq 2-4 : D_4 = \{3\}$	-	8
4	$X_3 = 2$	$X_3 \neq X_4 : D_4 = \{3\}$ $ X_3-X_4 \neq 3-4 : D_4 = \{\}$	Retorno a X_1	2
5	$X_1 = 2$	$X_1 \neq X_2 : D_2 = \{1, 3, 4\}$ $X_1 \neq X_3 : D_3 = \{1, 3, 4\}$ $X_1 \neq X_4 : D_4 = \{1, 3, 4\}$ $ X_1-X_2 \neq 1-2 : D_2 = \{4\}$ $ X_1-X_3 \neq 1-3 : D_3 = \{1, 3\}$ $ X_1-X_4 \neq 1-4 : D_4 = \{1, 3, 4\}$	-	24
6	$X_2 = 4$	$X_2 \neq X_3 : D_3 = \{1, 3\}$ $X_2 \neq X_4 : D_4 = \{1, 3\}$ $ X_2-X_3 \neq 2-3 : D_3 = \{1\}$ $ X_2-X_4 \neq 2-4 : D_4 = \{1, 3\}$	-	10
7	$X_3 = 1$	$X_3 \neq X_4 : D_4 = \{3\}$ $ X_3-X_4 \neq 3-4 : D_4 = \{3\}$	-	4
8	$X_4=3$	Solución! (2, 4, 1, 3)	-	0

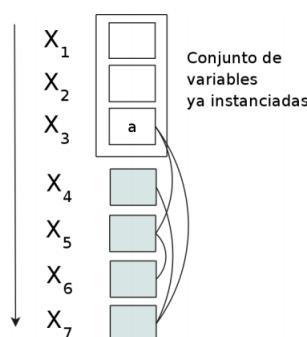
**X**

FC casi siempre es la mejor opción que BT.

6.7. Look-Ahead: RFLA

RFLA al instanciar realiza arco-consistencia en el espacio de búsqueda no instanciado. De esta manera en cada paso se filtran los dominios de las siguientes variables, de modo tal que en este paso se puede saber si no existe ninguna instancia factible y por ende, se debe probar otro valor de la variable actual X_i .

- Real Full Look-Ahead: a medida que se va instanciando, se realiza Arco Consistencia en el espacio de búsqueda no instanciado.



6.8. Heurísticas de selección de Variables/Valores

Heurística

- Conjunto de criterios que entre distintas posibilidades permite elegir la que promete ser más eficaz para resolver el problema
- Es un compromiso entre usar criterio simple y poder distinguir entre buenas y malas decisiones
- Un método heurístico puede ser usado para guiar acciones con el fin de encontrar soluciones aceptables
- Estos métodos no son difíciles de implementar y son eficientes computacionalmente, pero son estocásticos (no son predecibles) y no son precisos

Algunos tipos de heurísticas son:

- De orden de instanciación de las variables
- De orden de la elección de valores de las variables desde sus dominios

El orden puede ser estático (establecido antes de comenzar la búsqueda) o dinámico (que cambia durante la ejecución).

Algunos criterios de heurísticas de valores son:

- Búsqueda de una solución - elección de valores menos restringidos. Se conoce como Heurística de Minimización de Conflictos
- Búsqueda de solución óptima - se hace la mejor elección localmente según el criterio de optimización
- Búsqueda de todas soluciones - orden indiferente

Algunos criterios de heurísticas de variables son:

- La variable unida a mayor cantidad de restricciones ([variable más conectada](#))
- La variable unida a las restricciones más difíciles
- Orden del dominio más pequeño

Al aplicar las heurísticas a los algoritmos de búsqueda completa los posibles criterios de evaluación pueden ser:

- Tamaño de árbol(cantidad de instanciaciones) - tanto ancho como profundidad de árbol
- La cantidad de saltos
- La cantidad de chequeos de restricciones

6.9. Preguntas de Examen/Control

1. Para la siguiente aseveración: “Un árbol de búsqueda de BT será más ancho que uno para CBJ cuando se busca solo una solución”

V Falso, si el problema está poco restringido es probable que ambos árboles sean iguales

Sí. Si existen pocas restricciones CBJ puede quedar igual de ancho que BT ya que no habrán suficientes restricciones para hacer saltos.

V Verdadero cuando el problema está muy restringido y permite saltos inteligentes con CBJ

Sí. Si tenemos un grafo completo para las restricciones, CBJ va a tener árbol menos ancho. Por ejemplo, en N-Reinas CBJ termina menos ancho ya que no revisa tantas configuraciones de la misma variable.

F Verdadero, los árboles de CBJ siempre son más pequeños que los de BT

No. Si el grafo de restricciones es poco denso, entonces CBJ va a funcionar de manera muy parecida a BT.

F Verdadero, si el problema no tiene solución

No. Si el problema no tiene solución, es posible que en CBJ se obtenga el árbol igual de ancho que BT. Esto dependerá de la cantidad de restricciones que tenga el problema. Mientras más restricciones, más saltos puede hacer CBJ.

2. Al construir el árbol de búsqueda, es cierto que:

V Backtracking y Forward Checking pueden generar el mismo árbol de búsqueda

Sí. FC al instanciar una variable revisa si esta instancia genero algún conflicto. Se puede darse el caso que exista arco-consistencia y FC funcione igual a BT, es decir, sin filtrar.

V Forward Checking y Real Full Look Ahead pueden generar el mismo árbol de búsqueda

Sí. Nuavemente puede darse el caso de que FC y RFLA sean aplicados a un problema de modo tal que no ocurre ningún filtrado, y ambos terminan siendo un BT.

F El número de chequeos de restricciones en el primer paso de Forward Checking siempre es cero

No. Como los chequeos son hacia adelante, las primeras variables a instanciar de hecho tienden a hacer más chequeos que las variables posteriores.

V El número de chequeos de restricciones en el primer paso de Backtracking siempre es cero

Sí. BT no hace chequeos hacia adelante sino puede volver hacia atrás. Como es primera instancia, no hay chequeos.

3. Al construir el árbol de búsqueda, es cierto que:

F Los árboles son más grandes si los problemas tienen más restricciones

No. CBJ tiende a tener árboles más chicos para grafos de restricciones más conexos, por lo que la respuesta depende de la técnica.

- F Los árboles son más grandes si los problemas tienen más restricciones
No. Nuevamente, depende de la técnica. Es más grande para BT, pero más chico para CBJ.
- F Los árboles son más pequeños cuando se busca solo una solución
No. No necesariamente, para todas las técnicas el árbol puede quedar grande si la solución está en la última rama.
- V Los árboles son más pequeños con las técnicas look-ahead
Sí. Como look-ahead aplican un fuerte filtro en FC o RFLA, los árboles terminan siendo más chicos.

4. Si el conjunto de conflictos de X_1 , $C(X_1) = \{X_2, X_4\}$, y el de X_2 , $C(X_2) = \{X_3\}$, y el orden de instanciación es: X3, X4, X2 y X1, frente a un doble salto es cierto que:

- V El retorno de CBJ es a X2 y luego a X3
Sí. Se vuelve a la variable más recientemente instanciada de las que están en el conjunto de conflictos $C(X_1)$. Dado el orden de instanciación, X_1 fue instanciada más recientemente. Después el salto es a X_3 del conjunto de conflictos $C(X_2)$
- F El retorno de CBJ es a X2 y luego a X4
No. Si bien el primer salto es a X_2 , luego se salta usando $C(X_2)$. X_4 no es parte de este conjunto.
- F El retorno de BT es a X2 y luego a X3
No. BT tiene retorno cronológico que sería primero a X_2 y después a X_4 .
- F El retorno de GBJ es a X2 y luego a X3
No. GBJ salta a la más recientemente instanciada y unida por restricciones. La primera más recientemente instanciada unida por restricción con X_1 es la X_2 . Después, cuando el dominio de X_2 se vacíe se retorna a la variable más recientemente instanciada, que seria X_4 .

5. Respecto a la heurística de la variable más conectada, es cierto que:

- V Puede utilizarse de manera dinámica combinada con Forward Checking
Sí. Como FC usa filtrado de dominios, al combinar FC con esta heurística se puede filtrar de mejor manera y hacer menos instancias.
- F Puede ser útil combinada con Backtracking y con Forward Checking
No. Sirve con FC, pero BT no se ve beneficiado por esta técnica.
- F Instancia primero a las variables involucradas en menos restricciones
No. Es al revés.

6. Respecto a la heurística de minimización de conflictos, es cierto que:

- F Instancia primero el valor con menos probabilidad de pertenecer a una solución
No. Se instancia primero el valor menos restringido.
- F Es útil cuando se busca encontrar todas las soluciones
No necesariamente.

- V **Es útil cuando se busca encontrar una única solución**
Sí. Al usar valores menos restringidos es más probable encontrar una solución antes de caer en conflictos.
- F Instancia primero a los valores más altos de los dominios de las variables
No. Se instancia primero el valor menos restringido.
-
7. Respecto a la heurística del dominio más pequeño, es cierto que:
- F Instancia primero el valor con mayor probabilidad de pertenecer a una solución
No. Se instancia la variable con dominio más chico.
- V Instancia primero las variables de menor tamaño de dominio
Sí, es su definición.
- V Permite identificar tempranamente instanciaciones infactibles
Sí. Mientras más chico el dominio de la variable seleccionada, menos posibles soluciones con esta van a existir. Se puede filtrar los dominios y ver de antemano que no existe ninguna posible configuración para el problema.
- V Puede utilizarse de manera dinámica solo combinada con las técnicas look-ahead
Sí. Mientras más chico el dominio de la variable seleccionada, menos posibles soluciones con esta van a existir. Esto permite filtrar mejor los dominios de espacio por instanciar.
-
8. Acerca de Backtracking es cierto que:
- V Permite encontrar todas las soluciones del problema
- V Permite encontrar la solución óptima
- V En cada paso revisa la factibilidad de instanciación actual con las anteriores
- F Revisa todo el espacio de búsqueda
No. Las técnicas de búsqueda completa no siempre revisan todo el espacio.
- F Es más fácil de resolver en un grafo débilmente conectado que uno completo cuando se desea encontrar todas las soluciones del problema
No. Como se quieren todas las soluciones, igualmente se recorrerá todo el espacio.
-
9. Las técnicas de búsqueda completa son capaces de
- V Determinar que un valor para una variable no forma parte de solución
- V Determinar la solución óptima
- V Determinar la cantidad de soluciones para el problema
- V Determinar si el problema no tiene solución
- F Si un problema no tiene solución usando FC, puede ser que usando BT se encuentre la solución
No. Si cualquiera de las búsquedas completas no encontró solución es porque no hay solución y esta no puede ser encontrada por otra técnica.
-
10. Sobre las técnicas de búsqueda completa es cierto que

V GBJ puede experimentar trashing

Sí. GBJ en grafo completamente conectado es equivalente a un BT y puede sufrir de trashing.

F Espacio de búsqueda y árbol de búsqueda son términos equivalentes

No. Espacio de búsqueda es definido por variables y sus dominios. Árboles dependen de la técnica aplicada.

V Puede ser útil en grafos de restricciones con pocos arcos

Sí. Mientras menos restricciones hayan, menos parecido será GBJ de BT

V CBJ es más costoso que GBJ por eliminación y actualización de conjuntos de conflictos

Sí, exacto. Puede darse el caso que CBJ actue como GBJ en situaciones donde el conflicto se produce en variable más reciente unida por restricción. En este caso CBJ actúa como GBJ pero es más costoso.

11. Dados $D_1 = D_2 = D_3 = D_4 = 1, 2$ con orden de instanciación de X1,X2,X3,X4. Si se tienen las siguientes restricciones: $X1+X3 = 4$; $X2+X3 > 2$ y $X2+X4=3$

V El salto de CBJ sería de X3 a X1

Sí. En X3 se produce un fallo y la variable más reciente del conjunto de conflictos X3 es X1 (ya que es la más prematura entre X1 y X2), por ello se salta a esta.

V El salto de GBJ sería de X3 a X2 Sí, ya que X2 es la variable más reciente y unida por restricción a X3

12. La aseveración ".^C-3 realiza el mismo procedimiento que RFLA":

V Es falsa porque AC-3 no siempre encuentra soluciones

Sí. Es falso ya que AC-3 es una subrutina de RFLA. AC-3 por si solo genera arco-consistencia entre las variables una vez, mientras que RFLA lo hace por cada nueva instanciación.

F Es verdadera porque las técnicas look-ahead usan técnicas de filtro y consistencia

No. Es lo que hace RFLA.

F Es verdadera, en el primer paso de RFLA se realizan los mismos filtros que al aplicar AC-3

No. RFLA aplica AC-3 en cada instanciación.

F Es falsa porque RFLA realiza AC-3 en cada instanciación

No. RFLA aplica AC-3 en cada instanciación.

13. Respecto Forward Checking es cierto que

V Realiza retornos al detectar algún dominio de la variable futura vacío

Sí, porque significa que por esta rama no hay solución.

F Pierde soluciones al realizar filtro de dominios

No, no pierde soluciones

V Debe deshacer algunos filtros de dominio posterior al retorno

Sí, si se vuelve al punto anterior se debe devolver algunos dominios filtrados

F Realiza retorno al detectar el dominio de la variable actual vacío

No, nunca debe llegar a esta solución ya que no se filtra el dominio de la variable actual.

7. Técnicas incompletas

Si bien los algoritmos de búsqueda completa garantizan encontrar el óptimo global, para los problemas de mundo real (NP-completos) no logran encontrar la solución o se demoran demasiado. Es aquí donde se introduce el concepto de técnicas incompletas que proponen **sacrificar el óptimo global para obtener soluciones de buena calidad en tiempo reducido**.

Algunos conceptos importantes son:

1. Diversificación - el algoritmo visita otros espacios del vecindario y puede aceptar soluciones de peor calidad para permitir la exploración y para no estancarse en un óptimo local
2. Intensificación - el algoritmo se enfoca a encontrar la mejor solución posible (óptimo local) en la región actual

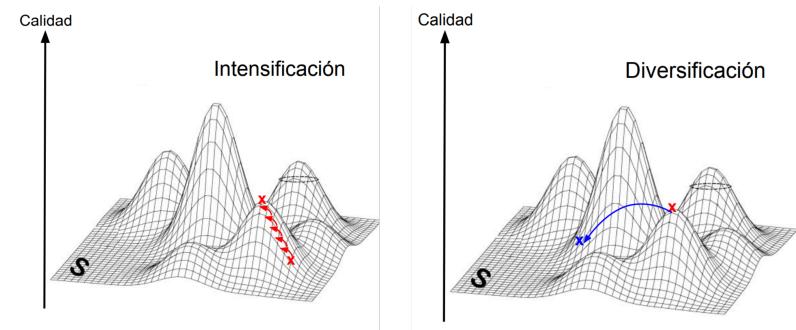


Figura 15: Intensificación / Diversificación

3. Función de evaluación - función informativa que representa los objetivos del problema
4. Solución candidata - a una instanciación completa (global) para el problema
5. Representación - la estructura que almacena las soluciones candidatas (listas simbólicas (coloreo), listas binarias (mochila), matriz simbólica (travelling tournament problem), etc)

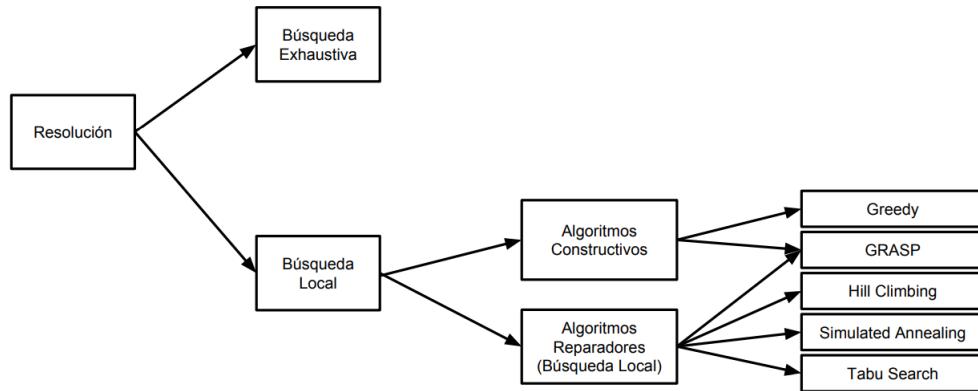


Figura 16: Taxonomía de Técnicas Incompletas

7.1. Algoritmos constructivos

Los algoritmos constructivos parten de una solución vacía y agregan componentes a esta hasta que se complete. La selección de qué agregar a la solución es localmente óptima. Se deben definir los siguientes 4 componentes:

- **Representación de la solución** - la estructura de la solución
- **Función miope** - función que toma decisiones localmente óptimas y se guía por la función de evaluación
- **Punto de Partida** - desde donde se empieza a construir la solución
- **Función de evaluación** - función que permite evaluar la solución actual usando los objetivos del problema

Por ejemplo, en TSP estos son:

- Representación de la solución - tour factible
- Función miope - agregar la siguiente ciudad más cercana y no visitada
- Punto de Partida - ciudad inicial A
- Función de evaluación - largo del tour

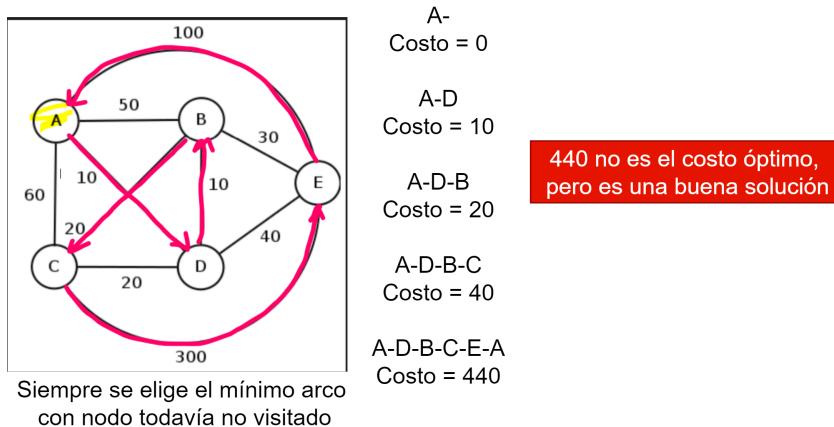


Figura 17: Solución de TSP Greedy partiendo desde el nodo A

Por ejemplo, en el Problema de Mochila podemos definir estas mismas componentes como:

- Representación de la solución - lista binaria
- Función miope - agregar un objeto que genera mayor ganancia y que sea factible (cumpla restricción de peso)
- Punto de Partida - Objeto 3
- Función de evaluación - ganancia total obtenida

$$\begin{aligned} & \max 18 \cdot x_1 + 25 \cdot x_2 + 11 \cdot x_3 + 14 \cdot x_4 \\ & \text{s.a. } 2 \cdot x_1 + 2 \cdot x_2 + x_3 + x_4 \leq 3 \\ & \quad x_1, x_2, x_3, x_4 \in \{0, 1\} \end{aligned}$$

Maximizar Sum ganancias

Restricción de peso maximo
peso*objetoVariable binaria - se agrega objeto
i (1) o no (0)

0	0	1	0
---	---	---	---

Agregamos objeto 3 (inicial)

x1	x2	x3	x4
0	1	1	0

El objeto que genera mayor ganancia es x2=25

$$2x_2 + x_3 \leq 3$$

Factible, se agrega a la mochila

Termina - no podemos agregar más objetos. Ganancia greedy = $0+25+11+0 = 36$
Se nota que si hubieramos partido desde objeto 4 la ganancia sería $25+14 = 39$

Figura 18: Solución de Mochila Greedy partiendo desde el objeto 3

7.2. Algoritmos reparadores

Son algoritmos que se mueven en el espacio de soluciones, partiendo de una solución generada y a través de las modificaciones, idealmente, la van mejorando. Generalmente, primero se **diversifica** y después se **intensifica**.

- Los algoritmos reparadores parten desde una solución inicial (x en la imagen)
- Para buscar otras soluciones, podemos aplicar un **movimiento** a x
- Todas las soluciones cercanas a x , respecto a un movimiento, forman el **vecindario** de x
- Si el algoritmo se enfoca en mejorar la calidad de las soluciones, podemos decir que está **intensificando** la búsqueda
- Si el algoritmo intenta visitar otras regiones del espacio de búsqueda, podemos decir que está **diversificando** la búsqueda

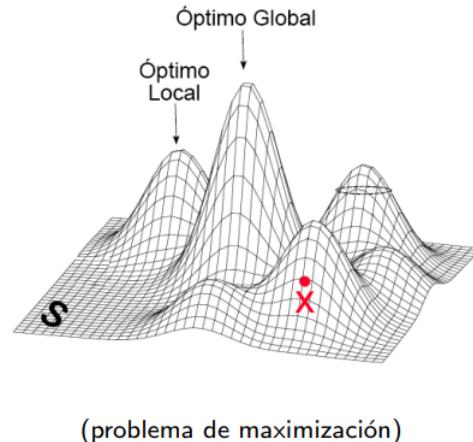


Figura 19: Movimiento de algoritmo reparador (intensificación y diversificación)

Algunos de los conceptos importantes son:

- **Movimiento** - transformación aplicada a la solución candidata, altera valores de algunas variables. Movimiento común es bit-flip para cadenas binarias: $1111 \rightarrow 0111$
- **Vecindario** - conjunto de soluciones generado al aplicar un movimiento a la solución actual (todas las soluciones que están a la distancia de 1 movimiento). Se anota con $\mathcal{N}(x)$.

$$x = 1111 \rightarrow \mathcal{N}(x) = [0111, 1011, 1101, 1110]$$

- **Óptimo local** - se dice que la solución \hat{x} es el óptimo local si para toda solución x en el vecindario $\mathcal{N}(\hat{x})$ su calidad es la mejor posible:

$$f(\hat{x}) \geq f(x), \forall x \in \mathcal{N}(\hat{x})$$

7.2.1. Preguntas del Examen/Control

1. Los algoritmos de búsqueda incompleta son mejores que los de búsqueda completa respecto a:

F Calidad de soluciones

No. Los algoritmos incompletos hacen un compromiso entre tiempo y calidad de solución. Algoritmos de búsqueda completa siempre entregan el óptimo global, sin embargo búsqueda incompleta entrega una solución suficientemente buena pero no el óptimo global.

V Tiempo de ejecución

Sí. Búsqueda completa no funciona para instancias del mundo real, por lo que se usa la búsqueda incompleta.

F Factibilidad de soluciones

No. Búsqueda incompleta puede recorrer espacio infactible.

F Predecibilidad del comportamiento

No. Algoritmos de búsqueda incompleta son estocásticos - no son deterministas.

2. Acerca de los algoritmos de búsqueda incompleta:

F Definen la representación de acuerdo al modelo del problema

No, no necesariamente. Por ejemplo, en TSP se habla de variable binaria que indica si se visitó un tour o no, pero el algoritmo real puede usar cualquier otra alternativa que más convenga.

V Pueden trabajar con soluciones factibles e infactibles

Sí. Cuando se diversifica e intensifica se pueden encontrar soluciones de ambos tipos.

F Trabajan siempre con soluciones factibles

No. Cuando se diversifica e intensifica se pueden encontrar soluciones de ambos tipos.

F No son capaces de resolver problemas de satisfacción de restricciones

No es cierto. En clases vimos que HC, TS y SA sirven para resolver el problema de la Mochila, que es CSOP.

7.3. Hill Climbing

Es un algoritmo que busca mejorar la solución generada de manera iterativa. Generalmente, se parte de una solución generada y en cada solución se genera el vecindario de esta solución. Si algún vecino mejora la solución actual, este reemplaza a la solución actual.

HC no define ninguna estrategia para escopar de óptimos locales. Cuando se encuentra la mejor solución candidata de la región, el algoritmo termina.

Existen dos tipos de HC - **Alguna Mejora (First-Improvement)** que selecciona el primer vecino mejor que la solución actual y **Mejor Mejora (Best-Improvement)** que selecciona el mejor vecino para sustituir la solución actual.

7.3.1. Hill Climbing Mejor Mejora

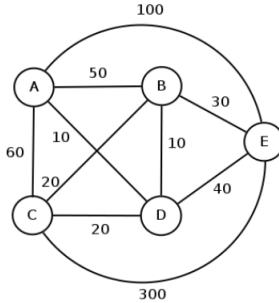
Mejor Mejora (Best-Improvement) selecciona el mejor vecino para sustituir la solución actual. Si no se encuentra un mejor vecino tal que se pueda reemplazar la solución actual, entonces el algoritmo termina.

Algorithm 5: Hill Climbing Best Improvement

```

local ← FALSE ;                                // flag de optimo local
repeat
     $s_n \leftarrow$  select from  $\mathcal{N}(s_c)$  the best quality point; //  $s_n$  es la mejor solución del
    // vecindario de  $s_c$ 
    if  $f(s_n)$  is better than  $f(s_c)$  then
        |  $s_c \leftarrow s_n$ ; // se reasigna la solución candidata si  $s_n$  tiene mejor calidad
    else
        | local ← TRUE
    end
until local;
  
```

Este algoritmo intensifica siempre y no tiene posibilidad de cambiar a otra región. Por ejemplo, si se aplica HC-MM en el problema TSP, se puede aplicar el operador de Swap (intercambio de ciudades) para generar la vecindad. Entre la vecindad generada se seleccionara el mejor tour (el de menor costo):



Tomando como solución inicial aleatoria A-E-D-B-C con costo 230 y utilizando Mejor Mejora, su vecindario usando un intercambio entre ciudades contiguas en el tour, sería:

- 1 E - A - D - B - C = 440
- 2 **A - D - E - B - C = 160**
- 3 A - E - B - D - C = 220
- 4 A - E - D - C - B = 210
- 5 C - E - D - B - A = 460

Si consideramos un algoritmo Alguna Mejora, para este caso, el primer vecino que mejora la solución es la número 2.

Figura 20: Ejemplo de cambio MM de HC aplicado en TSP

Si se aplica HC-MM en un el problema de la Mochila, podemos usar el operador de bit-flip para generar los vecindarios y, además, se descartan los vecinos que son infactibles debido a su peso. Se obtienen las siguientes iteraciones:

$$\text{Máx } 18 \cdot x_1 + 25 \cdot x_2 + 11 \cdot x_3 + 14 \cdot x_4$$

$$\text{s.a. } 2 \cdot x_1 + 2 \cdot x_2 + x_3 + x_4 \leq 3$$

$$x_1, x_2, x_3, x_4 \in \{0, 1\}$$

- Representación: Lista binaria de tamaño = número de objetos.
- Función de evaluación: Ganancia total.
 - Trabajare solo sobre soluciones factibles, descartaré las soluciones infactibles.
- Movimiento: Bit-flip.
- Solución inicial: Aleatoria factible.

Solución actual	Vecindario de solución actual generado con bityflip (soluciones infactibles se tachan)	La mejor de vecindario factible	Flag de óptimo local	
iteracion	s_c	$\mathcal{N}(s_c)$	s_n	local
1	1000 (18)			FALSE
		0000 (0) 1100(43) 1010(29) 1001(32)	1001(32)	
2	1001 (32)	0001 (14) 1101(57) 1011(43) 1000(18)	1000(18)	TRUE
		Mejor de vecindario no es mejor que la solución actual, entonces estoy en el óptimo local, termino.		

Figura 21: Ejemplo de solución de Knapsack usando HC-MM

7.3.2. Hill Climbing AM

Alguna Mejora (First-Improvement) selecciona el primer vecino mejor que la solución actual. Si se visitan todos los vecinos (max_neighbors) y ninguno mejora la solución

actual, se termina.

Algorithm 6: Hill Climbing First Improvement

```

local ← FALSE ;                                // flag de optimo local
 $s_c \leftarrow$  - select a random point ;          // solución inicial aleatoria
neighbor ← 0 ;                                 // contador de vecinos visitados
repeat
   $s'_n \leftarrow$  generate a neighbor point in  $\mathcal{N}(s_c)$ 
  neighbor++ ;                                // aumenta contador de visitados
  if  $f(s'_n)$  is better than  $f(s_c)$  then
    |  $s_c \leftarrow s'_n$  ;                      // se reasigna la solución candidata
    | neighbor ← 0;
  if neighbor == max_neighbor then
    | local ← TRUE ;                         // ya visitamos todos los vecinos
  until local;
  
```

7.3.3. Hill Climbing + Restart

Hill Climbing normal no tiene una manera de escapar de óptimos locales, por lo que introduce una estrategia Restart. La idea consiste en **re-comenzar el algoritmo con una solución nueva** y saltar a otra región. Se mantiene en memoria la mejor solución encontrada hasta el momento y se define la cantidad máxima MAX de Restarts permitidos.

```

Procedure hill-climbing
  t ← 0
  initialize  $s_{best}$ 
  Repeat
    local ← FALSE
     $s_c \leftarrow$  select a point at random
    Repeat
       $s_n \leftarrow$  select the best quality point in  $\mathcal{N}(s_c)$ 
      If  $f(s_n)$  is better than  $f(s_c)$  Then
         $s_c \leftarrow s_n$ 
      Else
        local ← TRUE
    Until local
    t ← t + 1
    if  $f(s_c)$  is better than  $f(s_{best})$  then
       $s_{best} \leftarrow s_c$ 
  Until t = MAX
  
```

Figura 22: HC-MM con Restart (puede ser AM también)

De esta manera HC+Restarts introduce la componente de **diversificación**.
Ventajas y Desventajas

- **Ventajas**
 - Permite escapar de óptimos locales
 - Incluye componente de la diversificación

■ Desventajas

- Se pierde información valiosa durante el proceso de búsqueda

Podemos intentar mitigar el estancamiento en óptimos locales aceptando peores soluciones, de esta manera en vez de saltar de la región potencialmente interesante aceptamos un vecino peor de esta región y nos quedamos en esta. Sin embargo, esto puede producir ciclos.

Por ejemplo, podemos aplicar HC-MM Restart en el problema de la Mochila. El procedimiento de HC-MM se repite tantas veces como lo indica el parámetro MAX de la cantidad de Restart permitidos.

$$\text{Máx } 18 \cdot x_1 + 25 \cdot x_2 + 11 \cdot x_3 + 14 \cdot x_4$$

$$\text{s.a. } 2 \cdot x_1 + 2 \cdot x_2 + x_3 + x_4 \leq 3$$

$$x_1, x_2, x_3, x_4 \in \{0, 1\}$$

- Representación: Lista binaria de tamaño = número de objetos.
- Función de evaluación: Ganancia total.
 - Trabajará solo sobre soluciones factibles, descartaré las soluciones infactibles.
- Movimiento: Bit-flip.
- Solución inicial: Aleatoria factible.

Contador de restarts	Solución actual	Vecindario de solución actual generado con bitflip (soluciones infactibles se tachan)	La mejor de vecindario factible	Flag de óptimo local	La mejor solución hasta el momento	
restart	it	s_c	$\mathcal{N}(s_c)$	s_n	local	s_{best}
1	1	1000 (18)	0000 (0) 1100(43) 1010(29) 1001(32)		FALSE	- - - - (0)
				1001(32)		
2	1	1001 (32)	0001 (14) 1101(57) 1011(43) 1000(18)			
				1000(18)	TRUE	1001 (32)
2	1	0000(0)	1000(18) 0100(25) 0010(11) 0001(14)			
				0100(25)	FALSE	
			Continuamos analizando de la misma manera hasta que se llegue a máxima cantidad de restarts			

Figura 23: Solución de Knapsack usando HC-MM y Restart

7.3.4. Preguntas del Examen/Control

1. Para la siguiente versión del problema de la mochila modificado, y respecto al algoritmo HC visto en clases para el problema de la mochila clásico, es cierto que:

$$\text{máx } 15x_1 + 25*x_2 + 12*x_3 + 20*x_4 + 30*x_5 + 40*x_6$$

$$x_1 + x_2 + x_3 + x_4 + x_5 + x_6 = 3$$

$$x_i \in \{0, 1\}$$

- F Es posible utilizar el mismo algoritmo si se desea buscar solo el espacio factible (descartando soluciones infactibles)

No, si no se introduce ningún cambio en el algoritmo entonces no tenemos como filtrar el espacio ya que los vecindarios que se generan no necesariamente contienen solo soluciones factibles. Debemos cambiar la función de evaluación.

- F Se debe modificar solo el movimiento si se desea buscar solo el espacio factible (sin descartar soluciones infactibles)

No, el movimiento (bit-flip) lo único que hace es generar el vecindario. No puede encargarse de generar vecindarios factibles, porque no es su función.

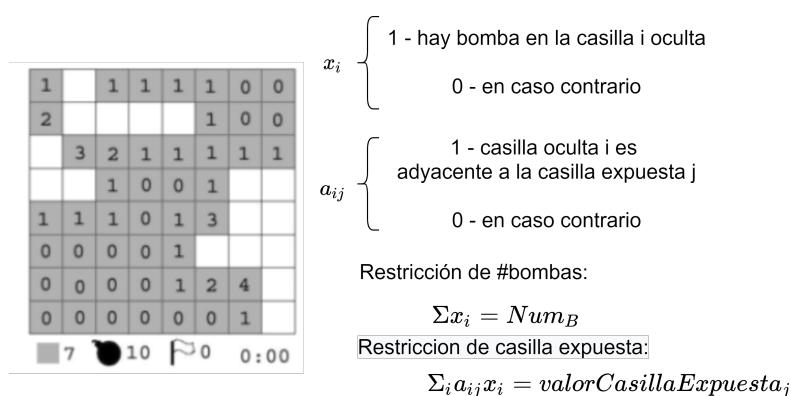
- V Se debe modificar solo la función de evaluación si se desea converger al espacio factible (sin descartar soluciones infactibles)

Sí, si queremos converger al espacio factible debemos cambiar la función de evaluación. La función de evaluación para Mochila es la Ganancia Total. Entonces se debe cambiar esta función para considere el peso exacto de 3 que deben dar los objetos.

- F Se debe modificar solo el proceso de inicialización si se desea buscar solo el espacio factible (sin descartar soluciones infactibles)

No. El proceso de inicialización inicializa solamente los flags algunas variables más para controlar el ciclo. No se puede cambiar solamente esto.

2. Para la siguiente versión del problema del buscaminas, y respecto al algoritmo HC visto en clases para el problema de la mochila, es cierto que:



- F Es posible utilizar el mismo algoritmo si se desea buscar solo el espacio factible (descartando soluciones infactibles)

No. Si no se introduce ningún cambio en el algoritmo entonces no tenemos como filtrar el espacio ya que los vecindarios que se generan no necesariamente contienen solo soluciones factibles. Debemos cambiar la función de evaluación.

- F Se debe modificar solo el movimiento si se desea buscar solo el espacio factible (sin descartar soluciones infactibles)

No. Bit-flip no se encarga de filtrar el vecindario, solo se encarga de generararlo.

- V Se debe modificar solo la función de evaluación si se desea buscar solo el espacio factible (sin descartar soluciones infactibles)

Sí. Se debe introducir la restricción de valor de casilla expuesta v_j : $\sum_i a_{ij} x_i = v_j$, donde a_{ij} - binaria de casilla i oculta adyacente a la expuesta j y x_i - hay bomba en i oculta.

- F Se debe modificar solo el proceso de inicialización si se desea buscar solo el espacio factible (sin descartar soluciones infactibles)

No. El proceso de inicialización inicializa solamente los flags algunas variables más para controlar el ciclo. Puede acotar el espacio pero no reducirlo a solo factible.

3. Respecto al algoritmo HC + Restart visto en clases para el problema de la mochila.

Si se quisiera extender su uso a la resolución del problema de ubicación de estaciones de bomberos considerando una cantidad de estaciones fija , es cierto que:

- F Es posible utilizar el mismo algoritmo si se desea buscar solo el espacio factible (descartando soluciones infactibles)

No. Se debe introducir un cambio en la función de evaluación, en caso contrario no se filtrará el espacio.

- F Se debe modificar el movimiento si se desea buscar solo el espacio factible (sin descartar soluciones infactibles)

No. El cambio de bit-flip no hace que la búsqueda sea solo en el espacio factible.

- V Se puede modificar solo la función de evaluación si se desea converger al espacio factible (sin descartar soluciones infactibles)

Sí. Se debe introducir la restricción de comunas $\sum_i a_{ij} x_i \geq 1$ que indique que en las comunas adyacentes ($a_{ij} = 1$ se debe instalar por lo menos 1 estación de bomberos).

- Se debe modificar el proceso de inicialización si se desea buscar solo el espacio factible (sin descartar soluciones infactibles)

No, no es suficiente.

4. Frente a la siguiente situación: Función de evaluación(Solucion actual): 25 y, sabiendo que, en el vecindario hay una solución vecina que tiene Función de evaluación (solución vecina): 20

- F En un problema de minimización HC+MM y SA siempre serán capaces de tomar esa solución vecina como solución actual de la siguiente iteración

No. HC+MM no va a tomar esta solución si es que existe una solución mejor que esta. Solo en caso de que no haya solución mejor que 20, se toma la solución 20. SA tiene selección probabilista del movimiento o vecino, por lo que tampoco podemos hablar de 'siempre'.

- F En un problema de maximización ni HC+MM ni SA, nunca consideraran esa solución vecina como solución actual de la siguiente iteración

No. Si bien HC+MM nunca la considerará como vecina debido a que empeora la calidad, SA sí puede considerar esta solución ya que la selección del vecino es probabilista.

- V En un problema de minimización HC+AM puede tomar esa solución vecina como solución actual de la siguiente iteración

Sí. Si esta solución aparece antes de alguna otra que mejore la calidad, se aceptara esta solución como la actual.

- V En un problema de maximización es posible que SA tome la solución vecina como solución actual

Sí. SA puede considerar esta solución ya que la selección del vecino es probabilista y permite tomar soluciones de peor calidad.

5. Suponga que $f(sc) = 25$ y su vecindario está compuesto por:

$$f(sv1) = 20$$

$$f(sv2) = 23$$

$$f(sv3) = 24$$

$$f(sv4) = 22$$

$$f(sv5) = 19$$

- V En un problema de minimización HC+MM y SA pueden tomar la misma solución como solución actual de la siguiente iteración

Sí. HC+MM sí o sí tomará la sv5 = 19 ya que es la mejor del vecindario. SA elige al vecino de manera probabilista, y es posible que se de el caso de que también se seleccione el sv5.

- F En un problema de minimización HC+MM y HC+AM siempre tomarán la misma solución como solución actual de la siguiente iteración

No. HC+MM tomará sv5 = 19 ya que es la mejor del vecindario, sin embargo SA elegirá de manera estocástica. Es decir, no necesariamente va a ser esta misma solución.

- F En un problema de maximización HC+MM y HC+AM siempre tomarán la misma solución como solución actual de la siguiente iteración

No. Misma razón - HC+MM tomará sv3 = 23 ya que es la que mejor maximiza. HC-AM tomará el primer vecino que mejore la calidad, que sería el sv1 = 20.

- V En un problema de maximización HC+MM y TS pueden tomar la misma solución como solución actual de la siguiente iteración

Sí. HC+MM tomará sv3 = 23. Tabu puede tomar este mismo vecino siempre que el movimiento a este no este en la lista tabu, ya que el criterio de tabu es 'tomar el mejor vecino que no este tabu'

- F En un problema de maximización HC+MM y TS tomarán la misma solución como solución actual de la siguiente iteración

No. HC+MM tomará la mejor solución sv3 = 23. TS puede no tomar esta solución si es Tabu, a pesar de que es la mejor.

- V En un problema de maximización HC+MM y SA pueden tomar la misma solución como solución actual de la siguiente iteración

Sí. HC+MM tomará sv3 = 23 y es posible que SA eliga esta misma solución, ya que elige al vecino de manera aleatoria.

6. A diferencia de HC+Restart, HC:

- F Define una estrategia para escapar de óptimos locales
No, es al revés. HC no escapa de los óptimos locales, pero HC+Restart sí intenta escapar.
- V Encuentra un único óptimo local
Sí. Una vez encontrado el óptimo local el algoritmo estanca en este.
- F Encuentra el óptimo global
No. Es una búsqueda incompleta que se estanca en el óptimo local.
- F Puede implementarse en sus versiones Best Improvement y First Improvement
No. Ambos pueden ser implementados en las dos versiones.
-

7. A diferencia de HC, HC+Restart:

- F Se estanca en óptimos locales
No. HC+Restart agrega la componente de Restart que permite escapar de los óptimos locales.
- V Diversifica
Sí. A diferencia de HC normal, HC+Restart reinicia el algoritmo con solución aleatoria y de esta manera diversifica.
- F Intensifica
No. Ambos algoritmos lo hacen.
- F Encuentra tantos óptimos locales como Restart
No. HC normal se estanca en el primer óptimo local, Restart puede encontrar varios.
-

7.4. Tabu Search

Es un algoritmo de búsqueda con **memoria a corto plazo** que permite aceptar las soluciones de peor calidad (permite la **diversificación**) para evitar estancarse en los óptimos locales. Debido a que pueden existir **ciclos** al aceptar las soluciones de más baja calidad, Tabu Search introduce el concepto de la **lista tabu**. Esta registra las últimas soluciones visitadas y impide volver a ellas.

- La lista se llena en FIFO - primero en entrar es primero en salir
- En la lista se guardan los atributos que permitieron llegar a la solución actual. **No se guardan las soluciones completas sino parte de sus atributos.**
- El largo de la lista tabu es un parámetro.
- El largo de la lista tabu indica que tanto se está memorizando y permite controlar la **diversificación y intensificación**. También se dice que el largo proporciona el balance estático.
 - Si la lista es larga (guarda muchas componentes), entonces aumenta **diversificación**. Obliga al algoritmo a explorar mayor espacio.
 - Si la lista es corta (guarda pocas componentes), entonces disminuye la diversificación y se revisan las soluciones cercanas (búsqueda local - **intensificación**). Aumenta la posibilidad de ciclos al visitar el mismo espacio.

- Tabu Search **diversifica** al aceptar soluciones de peor calidad e **intensifica** al hacer la búsqueda local.
- Reactive Tabu Search (Tabu con control de parámetros) - un algoritmo con la lista dinámica. Primero parte larga (para diversificar) y se acorta durante la ejecución (para intensificar).

Algorithm 7: Tabu Search

```

initialize  $s_c$  at random ;           // inicializa solución inicial candidata aleatoria
initialize  $tabulist$  as empty ;        // inicializa lista taboo vacía
initialize  $s_{best} \leftarrow s_c$  ;          // mejor solución
repeat
     $s_v \leftarrow$  select from  $\mathcal{N}(s_c)$  the best non tabu point;
     $s_c \leftarrow s_v$  ; // mejor solución del vecindario puede ser mejor o peor que anterior
    update  $tabulist$  ;      // agrega las condiciones que me llevaron a esta solución
    if  $f(s_c)$  is better than  $f(s_{best})$  then
        |  $s_{best} \leftarrow s_c$ ;
    end
until terminationCriterion();

```

El o los criterios de término `terminationCriterion()` son un criterio predefinido para terminar la búsqueda, como, por ejemplo, el tiempo de ejecución, cantidad de iteraciones, cuantas iteraciones han pasado desde que se actualizo s_{best} , etc.

Por ejemplo, podemos resolver el problema de la Mochila usando Tabu, con criterio de término igual a 5 iteraciones total. El largo de la lista Tabu será 2.

$$\begin{aligned}
 & \max 18 \cdot x_1 + 25 \cdot x_2 + 11 \cdot x_3 + 14 \cdot x_4 \\
 & \text{s.a. } 2 \cdot x_1 + 2 \cdot x_2 + x_3 + x_4 \leq 3 \\
 & \quad x_1, x_2, x_3, x_4 \in \{0, 1\}
 \end{aligned}$$

- Representación binaria de largo n , donde n corresponde a la cantidad de objetos.

1	0	0	0
---	---	---	---

- Movimiento: Cambio de una variable desde 0 a 1, ó desde 1 a 0
- Observación: Se trabaja sólo con soluciones factibles

Figura 24: Representación y Movimiento para Tabu Knapsack

						Vecindario generado con bitflip (se tachan los vecinos infactibles)	Movimiento tabu = bit que fue flipeado (j1,j2,j3 o j4)
it	s_c	s_{best}	$\mathcal{N}(s_c)$	$tabu_{list}$	mov.	tabú	
1	1000(18)	1000(18)		{}			
			0000(18) 1100(43) 1010(29) 1001(32) ←				
2	1001(32)	1001(32)		{j4}			
			0001(14) ← 1101(57) 1011(43) 1000(18)				
3	0001(14)	1001(32)		{j4, j1}			
			1001(14) 0101(39) ← 0011(25) 0000(0)				
4	0101(39)	0101(39)		{j1, j2}			
			1101(57) 0001(14) 0111(50) 0100(25) ←				
5	0100(25)	0101(39)		{j2, j4}			
			1100(43) 0000(0) 0110(36) ← 0101(39)				
6	0110(36)	0101(39)		{j4, j3}			
Criterio de termino: 5 iteraciones							

Figura 25: Solución de Knapsack con Tabu

En el proceso de búsqueda existe perdida de la información y buenas soluciones pueden ser excluidas del conjunto permitido. Para esto se introduce el **Criterio de Aspiración** que permite hacer el movimiento aún cuando la solución está en la lista tabu.

- Un ejemplo sencillo es si es que una solución candidata, a pesar de ser tabu, es mejor que la mejor encontrada, entonces se acepta.

Algorithm 8: Tabu Search with Aspiration Criteria

```

initialize  $s_c$  at random ;           // inicializa solución inicial candidata aleatoria
initialize  $tabulist$  as empty ;        // inicializa lista taboo vacía
initialize  $s_{best} \leftarrow s_c$  ;          // mejor solución
repeat
     $s_v \leftarrow$  select from  $\mathcal{N}(s_c)$  best point that satisfies aspiration criteria or the best
    non tabu point;
     $s_c \leftarrow s_v$ ;
    update  $tabulist$  ;
    if  $f(s_c)$  is better than  $f(s_{best})$  then
         $| s_{best} \leftarrow s_c$ 
    end
until terminationCriterion();

```

7.4.1. Preguntas del Examen/Control

1. ¿Cómo intensifica y cómo diversifica Tabu Search?

Tabu search puede controlar la diversificación y la intensificación usando el largo de la lista. Si la lista es larga, podemos explorar más espacio y mayor cantidad de vecindarios. Si la lista es corta, vamos a revisar un solo vecindario, intensificando la solución. Este último paso puede generar un ciclo.

2. Mencione dos elementos para almacenar en la lista Tabu para el problema TSP

El nodo y la posición en el tour. Por ejemplo (4,4) significa 4ta ciudad en 4ta posición del camino. De este modo si hago, por ejemplo, swaps, evitaré hacer swap que me deje ciudad 4 en la posición 4, teniendo así en la memoria los últimos movimientos.

3. Aparte de largo de la lista Tabu, ¿que más podría ayudar a diversificar la búsqueda?

El criterio de aspiración. Este permite aceptar aquellas soluciones que ya están en la lista tabu, por lo que, técnicamente, no se deben aceptar. El criterio de aspiración sirve para reparar la pérdida de información que ocurre durante la búsqueda, producto de la cual a veces se excluyen buenas soluciones del conjunto permitido.

4. Compare Tabu Search con Hill Climbing

Tabu Search y Hill Climbing, aparte de que uno usa memoria de lista tabú y otro no, se diferencian en cómo se actualiza la solución candidata s_c . Tabu actualiza la solución candidata por la de vecindario $\mathcal{N}(s_c)$, pero guarda la mejor solución vista hasta el momento aparte. De esta manera s_c puede ser peor que la mejor encontrada, pero esto permite a Tabu Search a diversificar. En cambio, Hill Climbing en cada iteración revisa si la solución s_n de vecindario es mejor si s_c y solo en este caso actualiza [5]. Es decir, HC mejora la solución y siempre intensifica, mientras que TS puede tanto intensificar como diversificar. Tabu puede revisar varios óptimos locales, en cambio HC termina cuando se estanca en el primer óptimo local encontrado.

1. Respecto al algoritmo Tabu Search visto en clases para el problema de la mochila. Si se quisiera extender su uso a la resolución del problema de ubicación de estaciones de bomberos, es cierto que:

V *Es posible utilizar el mismo algoritmo si se desea buscar solo el espacio factible (descartando soluciones infactibles)*

Sí, si se descartan las soluciones infactibles, TS debe ser capaz de buscar solo en el espacio factible. Es lo que se ha hecho con Knapsack Tabu es clases.

- F Se debe modificar solo el movimiento si se desea buscar solo el espacio factible (sin descartar soluciones infactibles)

No. El movimiento para el problema de bomberos es bit-flip (si se ubica una estación en comuna i o no). Cambio de este movimiento no va a permitir buscar solo en espacio factible. Usamos este mismo movimiento para Tabu Knapsack.

- V Se debe modificar solo la función de evaluación si se desea converger al espacio factible (sin descartar soluciones infactibles)

Sí. Para converger al espacio factible podemos cambiar la función de evaluación, para que se consideren las condiciones de estaciones y se minimice la cantidad de estas: $\min \sum x_i$. Además se pueden introducir las penalizaciones.

- F Se debe modificar solo el proceso de inicialización si se desea buscar solo el espacio factible (sin descartar soluciones infactibles)

No, el proceso de inicialización de Tabu consiste en crear una solución aleatoria e inicializar las listas. Al modificar esto no se puede asegurar moverse solo por el espacio factible.

2. Acerca de la Lista Tabu de Tabu Search:

- F Permite diversificar el proceso de búsqueda de soluciones

No. La lista de por sí sola no lo permite. Se diversifica con el largo de la lista. Mientras más largo, más se diversifica.

- F Almacena soluciones previamente visitadas

No. Por temas de optimización, no se almacenan las soluciones sino sus componentes - el paso que nos hizo llegar a la solución actual. Por ejemplo, en Knapsack esto puede ser un cambio de ciertos bits.

- V Sus prohibiciones pueden ser consideradas como solución actual de acuerdo al criterio de aspiración

Sí. El criterio de aspiración efectivamente puede permitir ignorar que la solución actual está dentro de la lista tabu, aceptándola a pesar de esto.

- F Se actualiza mediante protocolo LIFO

Se actualiza con el FIFO.

3. Respecto al algoritmo Tabu Search visto en clases para el problema de la mochila. Si se quisiera extender su uso a la resolución de un problema de set-partitioning, es cierto que

- V Es posible utilizar el mismo algoritmo si se desea buscar solo el espacio factible (descartando soluciones infactibles)

Sí. Es posible usar TS para buscar solo en espacio factible y es lo que se ha hecho con el problema de Tabu Knapsack.

- F Se debe modificar solo el movimiento si se desea buscar solo el espacio factible (sin descartar soluciones infactibles)

No. El movimiento visto en clases fue bitflip. Habría que introducir mayores cambios si no se descartan las soluciones infactibles.

- V Se debe modificar solo la función de evaluación si se desea converger al espacio factible (sin descartar soluciones infactibles)
- Sí. Podemos cambiar la función de evaluación para converger al espacio factible. Para knapsack la función de evaluación era max ganancia total, pero para set partitioning buscamos minimizar el costo $\sum c_j x_i$. Además se pueden introducir las penalizaciones.*
- F Se debe modificar solo el proceso de inicialización si se desea buscar solo el espacio factible (sin descartar soluciones infactibles) No. No es suficiente.
-
4. Frente a la siguiente situación: Función de evaluación(Solucion actual): 25 y sabiendo que, la solución del vecindario con menor función de evaluación tiene una Función de evaluación igual a 20
- F En un problema de minimización TS y SA siempre son capaces de tomar esa solución como solución actual de la siguiente iteración
- No. TS va a tomar esta solución siempre que el movimiento no esté tabu. Sin embargo SA va a elegir un vecino al azar, y solo si es que selecciona este vecino lo aceptará ya que mejora la solución.*
- F En un problema de maximización ni TS ni SA, tomaran esa solución vecina como solución actual de la siguiente iteración
- No. SA puede tomar esta solución si selecciona este vecino y su temperatura sea lo suficientemente alta para aceptar solución de peor calidad.*
- F En un problema de minimización TS con seguridad tomara esa solución vecina como solución actual de la siguiente iteración
- No. TS puede no tomar esta solución si el movimiento está tabu, sin embargo si no lo es TS sí seleccionará esta solución ya que es la mejor del vecindario.*
- V En un problema de maximización es posible que SA tome esa solución vecina como solución actual
- Sí. Si SA selecciona este vecino y la temperatura es suficientemente alta, SA puede elegir esta solución como actual.*
5. Frente a la siguiente situación: Función de evaluación(Solucion actual): 25 y función de evaluación de la solución seleccionada como solución nueva/vecina (Función de evaluación (Solucion nueva/vecina): 20)
- V En un problema de maximización solo con temperatura alta, TS y SA pueden tomar la misma decisión
- Sí. TS ya seleccionó este vecino así que este pasa a ser solución actual, ya que si fue seleccionado entonces fue el mejor no tabú. En caso de SA si SA ya seleccionó este vecino, lo aceptará siempre que la temperatura sea alta.*
- V En un problema de minimización TS y HC-MM pueden tomar la misma decisión
- Sí. TS ya seleccionó este vecino así que este pasa a ser solución actual, ya que si fue seleccionado entonces fue el mejor no tabú. Si HC-MM ya seleccionó este vecino entonces lo va a evaluar y lo va a tomar siempre que mejore la calidad de la solución actual.*

V En un problema de minimización TS y SA deben tomar la misma decisión

Sí. TS y SA deben tomar la misma decisión siempre que se asume que el vecino ya fue seleccionado. Esto significa que en TS este vecino es el mejor de vecindad y no es tabu, por lo que se guarda como solución actual. En SA significa que este vecino es el mejor de vecindad, y como mejora la calidad siempre se guarda.

6. Acerca del algoritmo Tabu Search:**F Se estancan el óptimos locales**

No. El algoritmo fue hecho de tal manera que para escapar de los ciclos se usa memoria a corto plazo. Además el algoritmo puede aceptar peores soluciones, por lo que no solo intensifica.

V Intensifica realizando búsqueda en vecindarios (local)

Sí. Usa búsqueda local para intensificar o mejorar la solución.

V Mientras más larga la lista, más diversificación realiza

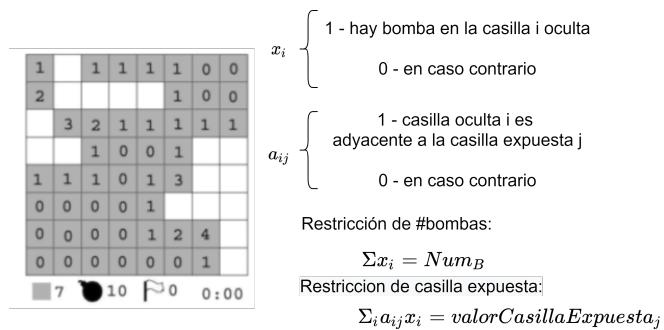
Sí. Mientras más larga la lista, más movimientos se restringen por lo que el algoritmo se ve obligado a explorar el espacio.

F No intensifica

No. Tabu search intensifica siempre que la lista permite mantenerse en el vecindario.

7. Respetto al algoritmo Tabu Search visto en clases para el problema de la mochila.

Si se quisiera extender su uso a la resolución de un problema de buscaminas:

**F Es posible utilizar el mismo algoritmo si se desea buscar solo el espacio factible (descartando soluciones infactibles)**

No. El problema de la Mochila es CSOP mientras que Buscaminas es CSP. Tenemos que introducir cambios al algoritmo ya que Buscaminas tiene un objetivo distinto.

V Se debe modificar solo la función de evaluación si se desea converger al espacio factible (sin descartar soluciones infactibles)

Sí. TS Knapsack usa función de evaluación de ganancia total, sin embargo Buscaminas es CSP y no tiene objetivo de optimización. La función de evaluación debe encargarse de revisar el cumplimiento de las restricciones.

V Se debe modificar el movimiento si se desea buscar solo el espacio factible (sin descartar soluciones infactibles)

Si. En buscaminas existe la restricción de número de bombas, en cambio en Mochila podríamos hacer una cantidad de bitflip no limitada.

- V Se debe modificar el proceso de inicialización si se desea buscar solo el espacio factible (sin descartar soluciones infactibles)

Sí.

7.5. Simulated Annealing

Simmulated Annealing o Recocido simulado se inspira en trabajo de termodinámica y pretende simular cambios energéticos en un sistema de partículas, hasta que este converge al estado estable.

- Para escapar de óptimos locales permite elegir soluciones que empeoran la calidad
- Existe una probabilidad de aceptar la solución de peor calidad y se asocia al temperatura del sistema. Mientras más alta la temperatura, más probable es aceptar la solución que empeora la calidad
- Al inicio la temperatura del sistema es alta
- Siempre se aceptan las soluciones que mejoran la calidad

Algorithm 9: Simmulated Annealing

```

t ← 0 ;                                     // contador de iteraciones
initialize  $T$  ;                           // temperatura del sistema
initialize  $s_c$  at random;
initilize  $s_{best} \leftarrow s_c$ ;
repeat
    repeat
         $s_n \leftarrow$  select new point in  $\mathcal{N}(s_c)$  ; // selecciona uno de vecinos usando número aleatorio [0,1]
        if  $f(s_n)$  is better than  $f(s_c)$  then
            |  $s_c \rightarrow s_n$  ;                      // siempre acepta la mejor solución
        else if  $random([0,1]) < e^{\frac{\Delta eval}{T}}$  then
            |  $s_c \rightarrow s_n$  ;                      // existe probabilidad de aceptar la solución si es peor
        if  $f(s_c)$  is better than  $f(s_{best})$  then
            |  $s_{best} \rightarrow s_c$ ;
    until haltingCriterion();
     $T \leftarrow g(T, t)$  ;           // actualizo la temperatura (decrece cada cierto tiempo)
     $t \leftarrow t + 1$ ;
until terminationCriterion();

```

La probabilidad de aceptación de una solución de peor calidad depende de la temperatura actual, solución nueva y solución actual:

$$P(\text{Aceptar}) = e^{\frac{\Delta eval}{T}}$$

donde $\Delta eval$ corresponde al valor negativo que representa un empeoramiento de la calidad de soluciones

- Si es problema de maximización: $\Delta eval = s_n - s_c$, ya que solución nueva s_n va a tener valor menor (peor) que la actual.
 - Si es problema de minimización: $\Delta eval = s_c - s_n$, ya que la solución nueva s_n va a tener valor mayor (peor) que la actual
- Temperatura versus Probabilidad de aceptación
 • Cada línea muestra un valor distinto de Δ_{eval}

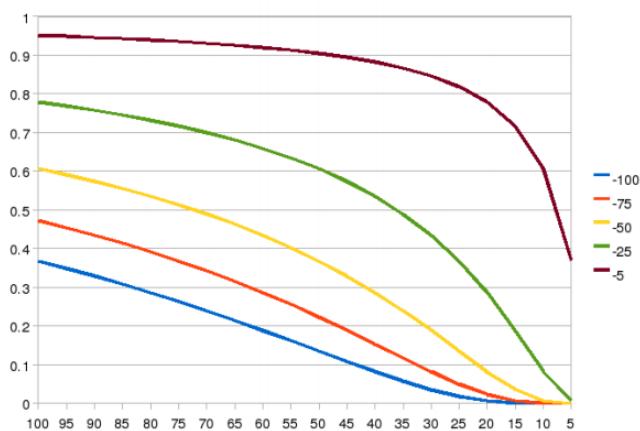


Figura 26: Mientras más alta la T, mayor es la probabilidad de aceptación

La temperatura del sistema es controlada con enfriamientos sucesivos y recalentamientos periódicos* (en algunos casos, para diversificar). Cada cierto número de iteraciones el sistema se enfría según:

$$T_{i+1} = \alpha T_i$$

donde $0 < \alpha < 1$ es un parámetro del problema, y se recomienda que su valor sea $\alpha \in [0.8, 0.99]$

- El algoritmo **intensifica** usando búsqueda local
- El algoritmo **diversifica** al escapar los óptimos locales aceptando soluciones de peor calidad
- El algoritmo parte con temperatura alta y **diversifica** más al inicio. Cuando la temperatura baja, el algoritmo **intensifica**
- Se puede implementar el recalentamiento que permite volver a **diversificar**

Se puede resolver el problema de la Mochila usando Simmulated Annealing:

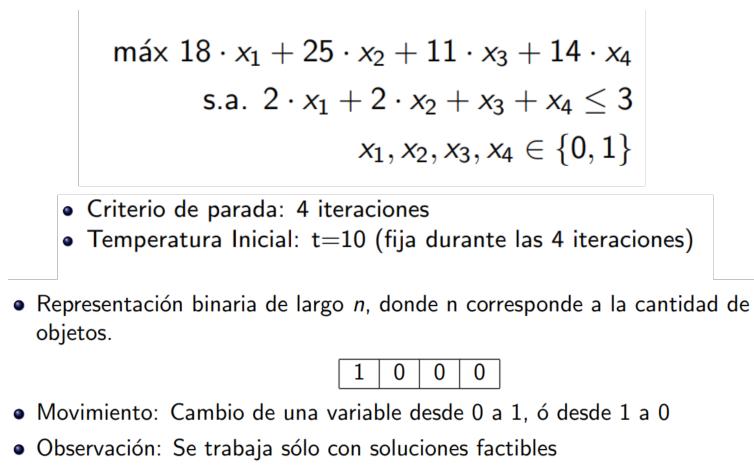


Figura 27: Representación y Movimiento para Knapsack con SA

Considere la siguiente secuencia de números aleatorios:

0.72; 0.33; 0.41; 0.83; 0.23; 0.64 ; ... $\stackrel{\text{=rand[0,1]}}{\dots}$

it	s_c	s_{best}	T	mov.	Δ_{eval}	$e^{\Delta_{eval}/T}$	condición
1	1000(18)	1000(18)	10	j1, j3 ó j4 j1, j3 ó j4	14	$e^{14/10} = 1.17$	se revisa que vecinos son factibles, y se eligen solo movim. factibles aceptado
2	1001(32)	1001(32)	10	j1 ó j4 j1 ó j4	-18	$e^{-18/10} = 0.165$	elige usando 0.33 y torta 50/50 0.42 no es menor que 0.165, rechazado rechazado
3	1001(32)	1001(32)	10	j1 ó j4 j1 ó j4	-14	$e^{-14/10} = 0.247$	aceptado
4	1000(18)	1001(32)	10	j1, j3 ó j4 j1, j3 ó j4	11	$e^{11/10} = 1.17$	aceptado
5	1010(25)	1001(32)	10				

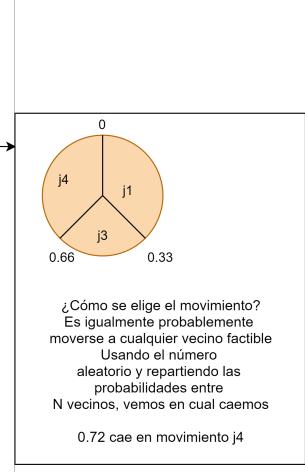


Figura 28: Solución de Knapsack SA sin bajar temperatura

Se puede resolver TSP usando SA, donde el movimiento que debemos hacer es un 2-opt, la función de evaluación corresponde al largo de tour y los arcos se seleccionan aleatoriamente.

7.5.1. Preguntas del Examen/Control

1. Acerca del algoritmo SA:

V Diversifica aceptando soluciones de peor calidad

Sí. Cuando la temperatura es alta, es más probable aceptar soluciones de peor calidad

F Solo puede ser aplicado a problemas de maximización

No. Se puede aplicar en ambos tipos del problema. Lo que cambiaría es como se calcula la diferencia entre calidad de soluciones Δ_{eval} . Esta debe ser negativa.

F Recalentar tiene el mismo efecto que el restart en HC+Restart

No. Si bien ambos algoritmos buscan volver a diversificar con Restart o Recalentamiento, Restart pierde toda la información y se mueve a una región aleatoria, SA simplemente puede volver a aceptar soluciones del vecindario que empeoren la calidad - a diferencia de Restart el salto no es tan brusco.

V Se puede estancar en óptimos locales cuando la temperatura es muy baja

Sí, ya que se vuelve difícil aceptar soluciones de peor calidad. SA terminará cuando se cumpla un cierto criterio de término.

2. En el algoritmo SA

V Es más probable aceptar una solución de peor calidad cuando la temperatura es alta

Sí. Mientras más alta es la temperatura, más probable es aceptar una solución de peor calidad.

F Es más probable aceptar una solución de peor calidad cuando la temperatura es baja

No. Mientras más baja la T, menor probable es aceptar una solución.

V El mismo movimiento puede ser aceptado o rechazado probabilísticamente

Sí. Siempre es en función de temperatura y números aleatorios.

V Intensifica al aceptar siempre soluciones de mejor calidad

Sí. SA siempre acepta la solución si esta mejora la calidad de la solución actual.

3. Respecto al algoritmo Simulated Annealing visto en clases para el problema de la mochila. Si se quisiera extender su uso a la resolución del problema de ubicación de estaciones de bomberos considerando una cantidad de estaciones fija , es cierto que:

F Es posible utilizar el mismo algoritmo si se desea buscar solo el espacio factible (descartando soluciones infactibles)

No.

F Se debe modificar solo el movimiento si se desea buscar solo el espacio factible (sin descartar soluciones infactibles)

No. Cambiar el bitflip no es suficiente.

V Se puede modificar solo la función de evaluación si se desea converger al espacio factible (sin descartar soluciones infactibles)

Sí. La SA Knapsack es un problema CSOP que utiliza como función de evaluación la ganancia total, sin embargo el problema de bomberos con número de estaciones fijas es CSP, por lo que se debe cambiar la función de evaluación.

7.6. Heurísticas K-Opt

Heurísticas K-Exchange o K-Opt reemplazan K arcos de un tour de TSP por K nuevos arcos, tal que el tour resultante es un tour factible del TSP.

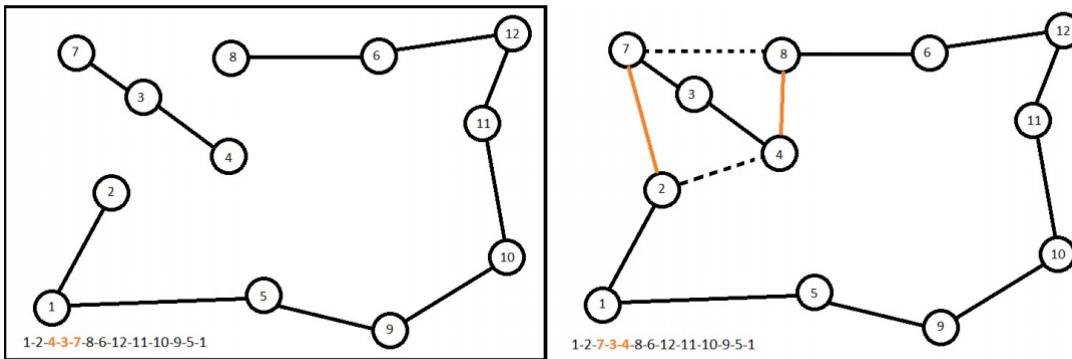


Figura 29: 2-opt

Cabe destacar que 2-opt es un movimiento **bastante más económico que un swap** entre 2 ciudades. Al hacer un swap, se intercambian 2 ciudades, y tenemos que recalcular todos los nuevos arcos que se generan entremedio y también en límites. En cambio, 2-opt **invierte el orden de visita**, por lo que se introducen solamente 2 nuevos arcos (en límites de subruta) que hay que recalcular.

- En el tour:

Tour1: A - B - C - D - E - F - G - H - I - J

El tour resultante después de swap es:

Tour2: A - B - **H** - D - E - F - G - **C** - I - J

En términos de costo, la nueva solución difiere en **cuatro** arcos respecto a la original:

$$\begin{aligned} f(\text{Tour2}) = f(\text{Tour1}) &- (d(B,C) + d(C,D) + d(G,H) + d(H,I)) \\ &+ (d(B,H) + d(H,D) + d(G,C) + d(C,I)) \end{aligned}$$

- En el tour:

Tour1: A - B - C - D - E - F - G - H - I - J

El tour resultante después de 2-opt es:

Tour3: A - B - **H** - **G** - **F** - **E** - **D** - **C** - I - J

En términos de costo, la nueva solución difiere en **dos** arcos respecto a la original: $f(\text{Tour3}) = f(\text{Tour1}) - (d(B,C) + d(H,I)) + (d(B,H) + d(C,I))$

Figura 30: 2-opt vs Swap

k-opt es más caro cuando aumenta el k, ya que aumenta la complejidad al tener que revisar más configuraciones, sin embargo es más eficiente en términos de intensificación.

7.6.1. Preguntas del Examen/Control

1. Acerca de las heurísticas K-exchange:

V **Seleccionan K arcos**, los quitan de la solución y vuelven a conectar los recorridos
Sí, por definición.

F 2-opt puede tener el mismo efecto que swap cuando se seleccionan dos arcos consecutivos

No. Porque para el ejemplo A-B-C-D en swap quedaría A-C-B-D y en 2-opt quedaría A-B-C-D

V **La cantidad de posibles re-conexiones crece exponencialmente respecto a K**
Sí, mientras más alto el k más configuraciones hay que revisar.

V **Swap** puede generar más cambios que 2-opt en los largos de los recorridos de TSP simétricos

Sí. El 2-opt invierte el camino entre 2 nodos, por lo que se debe recalcular solo los caminos de límites. En cambio, swap tiene que recalculiar todo el camino entre 2 nodos saneanados.

8. Algoritmos genéticos

Son algoritmos inspirados en el teorema de evolución de Darwin. La evolución se basa en el principio de supervivencia, donde aquellos individuos con mejor calidad son aquellos que viven más y tienen mayor probabilidad de producir descendencia.

Las características de los problemas incluyen:

- Una buena representación y función de evaluación (aptitud para supervivencia) ajustados al problema
- Una población - los individuos
- Mecanismos de herencia - **mutación** (operación unaria) y **cruzamiento** (operación n-aria)

El pseudocódigo general es (* indica el paso opcional):

Algorithm 10: Algoritmo Evolutivo

```

t ← 0 ;                                // inicia la cantidad de generaciones
iniciar población  $P(t)$ ;              // generalmente es aleatoria
evaluar  $P(t)$ ;
repeat
     $t \leftarrow t + 1$  ;                  // aumenta la generación
    seleccionar los individuos para tener descendencia*;
    aplicar operadores genéticos a individuos seleccionados*;
    evaluar descendencia;
    actualizar  $P(t) \leftarrow$  seleccionar individuos*
until terminationCriterion();
  
```

Representación:

Modelo que permite el adecuado procesamiento computación. **Existe una tendencia a la codificación de la representación** debido a la inspiración tomada desde ADN.

- **Genotipo** - la codificación de la solución. Es lo que se transmite. Características hereditarias definidas por un conjunto de genes. **Los operadores evolutivos se aplican sobre este.**
- **Fenotipo** - conjunto de propiedades observables y medibles, que pueden variar con el tiempo. Es la solución manejable por el usuario. No se transmite. **En base a este se calcula la calidad del individuo.**

Inicialización:

Se inicializa la población, intentando asegurar la variedad genotípica (para **diversificación**). Puede ser generada aleatoriamente o usando heurísticas. Generalmente su tamaño es fijo.

Evaluación:

Se hace uso de la función de aptitud para determinar la calidad de cada solución. Se evalúa al generar la población y, después, en cada iteración después de aplicar los operadores genéticos.

Selección:

Durante este proceso se eligen aquellos individuos que tendrán hijos. La selección se regula mediante **Presión de Selección** - que tanto importa la calidad a la hora de elegir el individuo. La selección puede ser:

- Elitista - sesgada a elegir los individuos de mejor calidad (más aptos). Es **extintiva** - no permite elegir los individuos de mala calidad. Presión alta.
- Uniforme - aleatoria, no se analiza la calidad de individuos. Presión baja.
- Determinista - se seleccionan μ mejores individuos de la población para ser padres. Existe riesgo de rápida convergencia y estancamiento.

Ejemplos de algoritmos:

1. **Roulette Wheel** - selección proporcional a la aptitud: $P_i = \frac{Aptitud_i}{\sum Aptitudes}$. La probabilidad de elegir individuo i depende de su aptitud versus la suma de aptitudes de la población. Tiene **presión alta cuando existe una gran variedad de aptitudes en el conjunto**.

Por ejemplo, tenemos que $\sum_{i=1}^n Aptitud_i = 15$ y $P(I1) = \frac{1}{15} = 7\%$, $P(I2) = \frac{2}{15} = 13\%$...

Individuo	Aptitud
I1	1.0
I2	2.0
I3	3.0
I4	4.0
I5	5.0

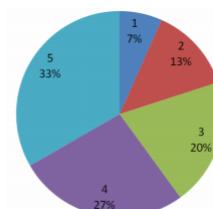


Figura 31: Ejemplo de ruleta

Este método tiene un **problema**: si una fracción de individuos tiene aptitud muy alta, siempre serán elegidos. Se perjudica la exploración, estancando en los óptimos locales.

2. Ranking - Los individuos se ordenan según su aptitud en un ranking. En base a su posición en el ranking se define una ponderación con la que se obtiene la probabilidad de que sea seleccionado. Para obtener la probabilidad de selección se divide el lugar del ranking por la sumatoria de los puestos.

Individuo	Desempeño	Rank	Probabilidad
I_2	12	6	$\frac{6}{23} = 26,1\%$
I_5	6	5	21,7%
I_1	2	4	17,4%
I_3	2	4	17,4%
I_4	1	2	8,7%
I_6	1	2	8,7%

Figura 32: Ejemplo de ranking

Su **ventaja** es que la aptitud (desempeño) influye menos, y, por ello, la elección es menos discriminatoria que en la ruleta.

3. k-tournament - Se selecciona un conjunto de soluciones al azar (**probabilista**) y entre ellas se elige la que tenga mejor calidad. K define el tamaño de torneo, es decir, la cantidad de soluciones que se tomarán cada vez para comparar. Si K es alto, implica mayor presión de selección, ya que existirán más candidatos y la posibilidad de elegir el mejor (elitista) será mayor.

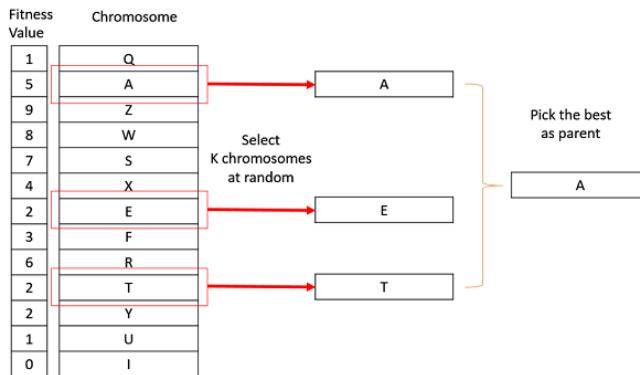


Figura 33: Ejemplo de 3-torneo

Cruzamiento:

Mezcla de las soluciones anteriores para obtener una nueva solución, generando una solución intermedia en el vecindario (**intensificación**). Para mezclar las soluciones se hace uso de un operador de cruzamiento, normalmente binario (dos soluciones) o n-ario. Existe **cierta probabilidad** de que se haga el cruce.

El cruzamiento puede producir **soluciones infactibles** con ciertas representaciones, por lo que cambio de representación o la codificación de soluciones permite arreglar la situación (GA Modificado [8.2])

- **Cruce de punto fijo** - Se elige un cierto punto, a partir del cual se mezclan los padres. Para 2 padres se tiene:

② Cruzamiento en un punto (Ejemplo - Problema de la Mochila)

P1	[1 - 1 1 - 1 - 1]
P2	[0 - 0 0 - 0 - 0]
<hr/>	
Hijo 1	[1 - 1 0 - 0 - 0]
Hijo 2	[0 - 0 1 - 1 - 1]

Figura 34: Cruce punto fijo

- **Cruzamiento uniforme** - En caso de representación de cadena de bits, por cada bit existe una probabilidad (selección estocástica) de elegirlo de Padre 1 o Padre 2 (bits marcados son los elegidos):

① Cruzamiento Uniforme (Ejemplo - Problema de la Mochila)

P1	[0 - 1 - 0 - 1 - 1]
P2	[1 - 0 - 1 - 0 - 0]
<hr/>	
Hijo	[1 - 1 - 1 - 0 - 1]

Figura 35: Cruce uniforme

Mutación:

La mutación suele considerarse un operador secundario, con una probabilidad de aplicación muy inferior a la del cruce. La probabilidad de mutación se asigna a cada gen. La mutación ayuda a **diversificar**, ya que genera nueva información (a diferencia de cruce que usa soluciones ya existentes).

- Cada uno de los hijos puede ser posteriormente mutado.

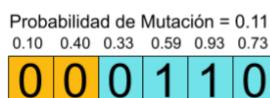


Figura 36: Ejemplo de mutación con bit flip

Término:

Existen varias criterios para el término del ciclo principal. Por ejemplo:

- Se llega a la cantidad de iteraciones límite
- Se llega a la cantidad de evaluaciones de individuos límite
- Se logra la aptitud esperada
- Se converge a individuos similares (se llega a nivel mínimo de diversidad)
- Se hace la cantidad fija de iteraciones sin mejoras (estancamiento)

- Combinación de los criterios anteriores.

La diversificación-intensificación se resume en la siguiente tabla:

Componente	Diversificación	Intensificación
Selección	Presión de selección baja	Presión de selección alta
Elitismo	-	Fomenta
Transformaciones	Mutación	Cruzamiento
Tamaño Población	Grande	Pequeño

Figura 37: Diversificación/Intensificación

8.1. Algoritmo Genético (GA)

Holland,1975. Algoritmo que surgió en el ámbito matemático y que, generalmente, usa representación de cadena de bits (mientras más bits, mayor precisión).

Para convertir una solución x_i que está definida en el dominio $[a, b]$ se utiliza la siguiente fórmula:

$$x_i = a + \text{decimal}(\text{cadena}) \cdot \frac{b - a}{2^k - 1}$$

donde $2^k - 1$ es el máximo número representable con k bits.

Ejemplo:

- Para convertir $x_1 \in [-3, 0; 12, 1]$
- Si el string de largo 18 que representa la variable x_1 es (010001001011010000)
- El valor de x_1 corresponde a:

$$x_1 = -3,0 + \text{decimal}(010001001011010000) \cdot \frac{(12,1 + 3,0)}{(2^{18} - 1)}$$

$$x_1 = -3,0 + 70352 \cdot \frac{(15,1)}{262143} = 1,0524$$

Figura 38: Ejemplo de conversión

Su pseudocódigo es:

Algorithm 11: Algoritmo Genético General

```

t ← 0 ;                                // inicia la cantidad de generaciones
iniciar población  $P(t)$ ;              // generalmente es aleatoria
evaluar  $P(t)$ ;
repeat
    t ← t + 1 ;                         // aumenta la generación
    seleccionar los individuos para tener descendencia;
    cruzar;
    mutar;
    evaluar descendencia;
    actualizar  $P(t)$ ;                // no hay selección entre los descendientes
until terminationCriterion();
```

Su flujo general es:

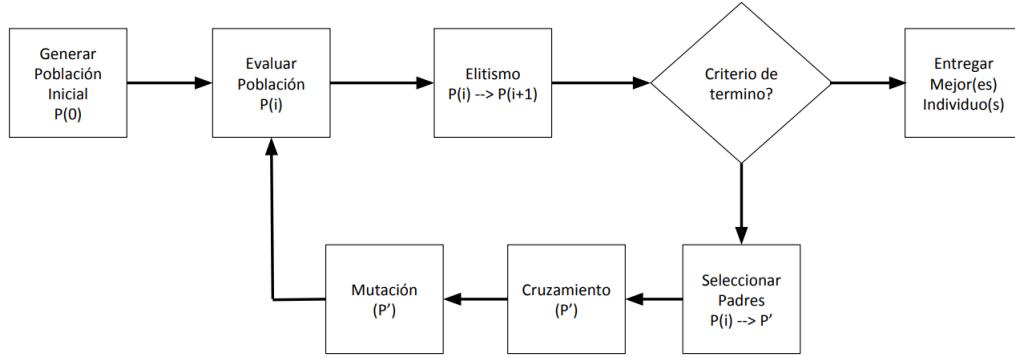


Figura 39: Flujo GA

Ojo: **no necesariamente siempre se aplica la mutación y/o cruce**. Como son probabilísticos, es posible que en alguna iteración los padres no cambien, o solo se crucen, o solo muten, o se crucen y se muten.

Resumen:

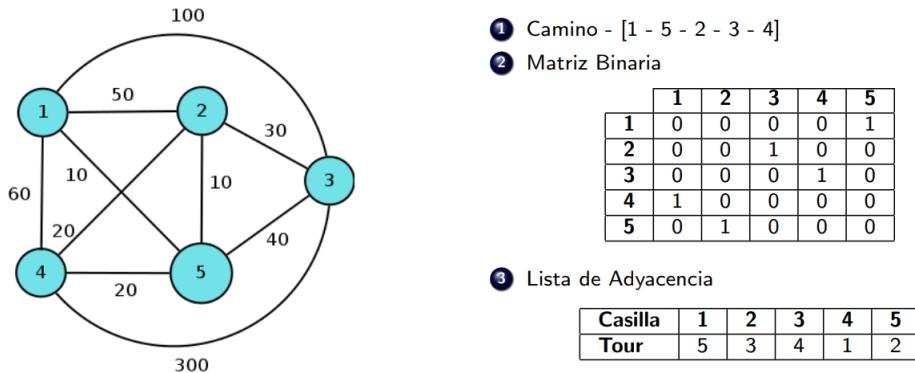
Algoritmos Genéticos	
Representación	Strings binarios
Cruzamiento	Un punto
Mutación	Bit-flip
Selección de padres	Proporcional a su aptitud
Reemplazo (Selección de Sobrevivientes)	Hijos reemplazan a padres
Especialidad	Énfasis en cruzamiento

Figura 40: Resumen GA

8.2. Algoritmo Genético Modificado

Consiste en modificar la representación de las soluciones para **evitar los casos donde el cruce entre dos padres da soluciones infactibles**.

Por ejemplo, las representaciones clásicas de TSP son el camino, la lista y la matriz de adyacencia:



Tour: [1 - 5 - 2 - 3 - 4]

Figura 41: Representaciones TSP

En el algoritmo genético modificado se agrega **Lista de Referencia**. Para el tour [1-5-2-3-4] se tiene la siguiente codificación:

Ciudad	1	2	3	4	5
Tour	1				

[1 - 5 - 2 - 3 - 4]
1 está en la posición 1 de la lista de ciudades
[1-2-3-4-5]

Ciudad	1	2	3	4	5
Tour	1	4			

[1 - 5 - 2 - 3 - 4]
5 está en la posición 4 de la lista de ciudades
[2-3-4-5]

Ciudad	1	2	3	4	5
Tour	1	4	1		

[1 - 5 - 2 - 3 - 4]
2 está en la posición 1 de la lista de ciudades
[2-3-4]

Ciudad	1	2	3	4	5
Tour	1	4	1	1	

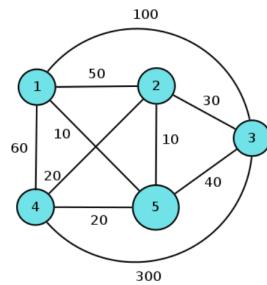
[1 - 5 - 2 - 3 - 4]
3 está en la posición 1 de la lista de ciudades
[1-4]

Ciudad	1	2	3	4	5
Tour	1	4	1	1	1

[1 - 5 - 2 - 3 - 4]
4 está en la posición 1 de la lista de ciudades
[4]

Figura 42: Generación de la lista de referencia

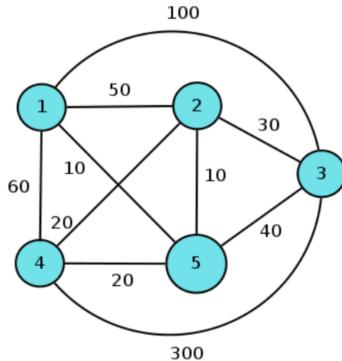
Podemos demostrar que el uso de la lista de referencia evita las soluciones infactibles. Para el siguiente TSP, sin cruzar, se obtienen dos soluciones infactibles (revisan misma ciudad 2 veces):



Cruzamiento en un punto
Tour 1 [1 - 4 | 3 - 2 - 5]
Tour 2 [5 - 1 | 4 - 3 - 2]
Hijo 1 [1 - **4** | **4** - 3 - 2]
Hijo 2 [**5** - 1 | 3 - 2 - **5**]

Figura 43: TSP: fallo de cruce

Si modificamos los padres y los codificamos usando la lista de referencia (codificación en negrita), se obtienen hijos que, al decodificar (última línea), son factibles:


Cruzamiento en un punto - Lista de Referencia

Lista	[1 - 2 - 3 - 4 - 5]
Padre 1	[1 - 4 - 3 - 2 - 5]
Padre 1 (c)	[1 - 3 2 - 1 - 1]
Padre 2	[5 - 1 - 4 - 3 - 2]
Padre 2 (c)	[5 - 1 3 - 2 - 1]
Hijo 1 (c)	[1 - 3 3 - 2 - 1]
Hijo 2 (c)	[5 - 1 2 - 1 - 1]
Hijo 1	[1 - 4 5 - 3 - 2]
Hijo 2	[5 - 1 3 - 2 - 4]

Padre 1 - Codificación:

Ciudad	1	2	3	4	5
Tour	1	3	2	1	1

[1-2-3-4-5] para 1 la posición es 1
 [2-3-4-5] para 4 la posición es 3
 [2-3-5] para 3 la posición es 2
 [2-5] para 2 la posición es 1
 [5] para 5 la posición es 1

Padre 2 - Codificación:

Ciudad	1	2	3	4	5
Tour	5	1	3	2	1

[1-2-3-4-5] para 5 la posición es 5
 [1-2-3-4] para 1 la posición es 1
 [2-3-4] para 4 la posición es 3
 [2-3] para 3 la posición es 2
 [2] para 5 la posición es 1

Hijo 1 - Decodificación:

Ciudad	1	2	3	4	5
Tour	1	3	3	2	1
Decod	1	4	5	3	2

[1-2-3-4-5] en la posición 1 esta el 1
 [2-3-4-5] en la posición 3 esta el 4
 [2-3-5] en la posición 3 esta el 5
 [2-3] en la posición 2 esta el 3
 [2] en la posición 1 esta el 2

Hijo 2 - Decodificación:

Ciudad	1	2	3	4	5
Tour	5	1	2	1	1
Decod	5	1	3	2	4

[1-2-3-4-5] en la posición 5 esta el 5
 [1-2-3-4] en la posición 1 esta el 1
 [2-3-4] en la posición 2 esta el 3
 [2-4] en la posición 1 esta el 2
 [4] en la posición 1 esta el 4

Figura 44: Codificación y decodificación en el cruce

Ventaja: Mantuvimos la factibilidad de solución

Desventaja: Costo de codificación, cruce no intensifica sino diversifica en este caso (genera nueva información).

GA modificado en TSP tiende a ocupar los algoritmos de transformación de Swap o de K-Opt, junto con selección con k-tournament.

- ➊ Intercambio (o Swap)
 - 1-2-7-3-4-8-6-12-11-10-9-5-1
 - 1-8-7-3-4-2-6-12-11-10-9-5-1
- ➋ K-opt (K-exchange): Es un procedimiento que reemplaza K arcos de un tour de TSP por K nuevos arcos, tal que el tour resultante es un tour factible del TSP.

• 2-opt:

1-2-4-3-7-8-6-12-11-10-9-5-1 //reemplazar 2-4 y 7-8 por 2-7 y 4-8
 1-2-7-3-4-8-6-12-11-10-9-5-1

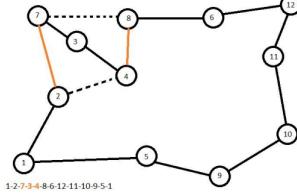


Figura 45: Transformaciones de GA modificado

Epistasis - consiste en la existencia de fuertes vinculaciones entre los genes, de modo tal que afectan uno a otro potenciando o inhibiendo ciertos genes. En algunos casos el resultado es tan difícil que debe ser analizado en conjunto con otros individuos y no individualmente ya que no tiene sentido ser analizado por si solo.

8.3. Programación Evolutiva (EP)

Fogel, 1965. Surgió para problemas de secuencia de máquinas finitas. Se enfatizan los nexos entre padres e hijos. El algoritmo no cruza, solo muta, y la nueva población se crea usando **mutación entre padres e hijos** $\mu + \mu$. EP es sesgado a los individuos con mayor calidad y tiene alta presión de selección.

La probabilidad de mutación cambia durante la búsqueda.

Algorithm 12: Programación Evolutiva

```

 $t \leftarrow 0;$                                 // inicia la cantidad de generaciones
iniciar población  $P(t);$                       // generalmente es aleatoria
evaluar  $P(t);$ 
repeat
   $t \leftarrow t + 1;$                           // aumenta la generación
  mutar;                                     // no hay selección de padres ni cruce, solo la mutación
  evaluar descendencia;
  actualizar  $P(t) \leftarrow$  seleccionar individuos;
until terminationCriterion();
  
```

Resumen:

Programación Evolutiva	
Representación	Máquinas de estado finito
Cruzamiento	No
Mutación	Agregar/Borrar estado Cambiar output/transition
Selección de padres	Determinista
Reemplazo (Selección de Sobrevivientes)	Probabilista ($\mu + \mu$)

Figura 46: Resumen PE

8.4. Estrategias Evolutivas (ES)

Rechenberg, 1973. Se selecciona un conjunto de individuos uniforme (aleatorio, presión de selección baja) para ser padres. Dos padres generan un hijo mediante cruzamiento, y este después puede mutar. Además, ES introduce el concepto de **adaptabilidad** - tiene un proceso evolutivo a nivel de parámetros (varianza de mutación o tipo de cruce). Además, en este algoritmo la mutación es más importante que el cruzamiento.

Algorithm 13: Estrategia Evolutiva

```

t ← 0 ;                                // inicia la cantidad de generaciones
iniciar población  $P(t)$ ;            // generalmente es aleatoria
evaluar  $P(t)$ ;
repeat
     $t \leftarrow t + 1$  ;                  // aumenta la generación
    cruzar;
    mutar;
    evaluar descendencia;
    actualizar  $P(t) \leftarrow$  seleccionar individuos;
until terminationCriterion();
  
```

Tiene 2 tipos de selección:

- Selección elitista - se eligen los mejores individuos entre los padres y los hijos
- Selección no-elitista - se eligen los mejores individuos entre solamente los hijos

Resumen:

Estrategias Evolutivas	
Representación	Vectores de valores reales
Cruzamiento	Discreto/Intermedio
Mutación	Perturbación gaussiana
Selección de padres	Aleatoriamente uniforme
Reemplazo (Selección de Sobrevivientes)	(μ, λ) o $(\mu + \lambda)$
Especialidad	Auto-adaptación de pasos de mutación

Figura 47: Resumen ES

8.5. Programación Genética (PG)

Koza, 1992. Algoritmo usa los áboles para representar los individuos (programas), donde los nodos internos son los operadores y los externos - nodos terminales con valores.

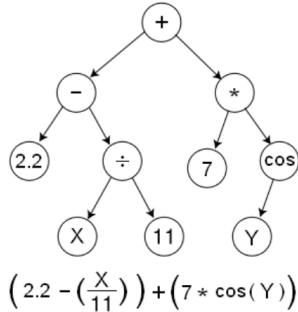


Figura 48: Árbol de individuo en PG

En PG los individuos son de **tamaño variable**. El cruzamiento es mucho más importante que la mutación.

Algorithm 14: Programación Genética

```

t ← 0 ;                                // inicia la cantidad de generaciones
iniciar población  $P(t)$ ;           // generalmente es aleatoria
evaluar  $P(t)$ ;
repeat
     $t \leftarrow t + 1$  ;                  // aumenta la generación
    seleccionar los individuos para tener descendencia;
    cruzar o mutar;
    evaluar descendencia;
    actualizar  $P(t)$ ;                // pasan todos los hijos cruzados o mutados
until terminationCriterion();
  
```

Resumen:

Programación Genética	
Representación	Árboles
Cruzamiento	Recombinación de sub-árboles
Mutación	Cambio aleatorio en árboles
Selección de padres	Proporcional a su aptitud
Reemplazo (Selección de Sobrevivientes)	Reemplazo generacional

Figura 49: Resumen PG

8.6. Preguntas del Examen/Control

1. ¿Cómo diversifican los algoritmos genéticos?

Primero, se crea una población tal que se intenta asegurar una variedad genotípica,

para poder diversificar a futuro. Segundo, los algoritmos usan el operador de mutación que permite diversificar. Se asigna una probabilidad de mutación a cada gen, y de este modo cuando ocurre la mutación de gen se genera nueva información (diversifica). Finalmente, se puede intentar mantener la diversificación al elegir un algoritmo de selección con menor presión de selección.

2. ¿Cuál es el objetivo de cruzamiento?

El objetivo de cruzamiento es intensificar, mezclando la información existente entre los padres y así obteniendo una nueva solución.

3. ¿Qué es el elitismo?

Elitismo consiste en elegir individuos con la mejor calidad, es decir, individuos más aptos, para que estos sean los que se propagan en las futuras generaciones.

4. ¿Cuáles son las ventajas de uso de la lista de referencia?

Evita soluciones infactibles. Se puede demostrar que si en TSP se utiliza la lista de referencia en vez de las representaciones clásicas como lista de adyacencia o matriz binaria, se evitan las soluciones infactibles. Cabe destacar que si bien se mantiene la factibilidad, existe el costo de codificación de la solución. Además el cruce en vez de intensificar termina diversificando.

5. ¿Cuáles son los parámetros del Algoritmo Genético?

Probabilidad de cruce (en un punto), probabilidad de mutación (con bit-flip), número de generaciones, tamaño de población

6. ¿Cómo se aplicaría el cruzamiento uniforme entre 20 padres?

En cruzamiento uniforme se utiliza una cadena de bits que representa al parent, donde por cada bit existe una probabilidad de su selección. Entonces, si tenemos 20 padres, por cada posición de la cadena tenemos que elegir la información entre los 20 padres en la columna que se forma, según la probabilidad que tiene cada parent en este bit específico.

7. ¿Qué es un k-torneo?

Es una de las técnicas de selección de individuos. Consiste en seleccionar k individuos de la población al azar y compararlos entre sí. Mientras más grande el K , más probable es que la selección será elitista (ya que entre más candidatos se seleccionará el mejor más global). La técnica es la misma que se usa, por ejemplo, en torneos de clasificación entre distintos equipos de deportes.

8. ¿Qué es el método de la ruleta, cómo se calcula la probabilidad de elegir un individuo con este método?

El método de la ruleta utiliza una asignación de probabilidad relativa a la suma de las aptitudes de los individuos. Es decir, la probabilidad de seleccionar un individuo X será igual a su aptitud partida entre la suma de aptitudes de la población. El método tiene presión alta cuando hay gran variedades de aptitudes en conjunto, y es sesgado a siempre elegir los mejores individuos.

9. Explicar la transformación en un algoritmo evolutivo.

Existen dos tipos de transformación - mutación (diversifica) y cruzamiento (intensifica). La primera cambia los genes de modo tal que se genera nueva información - en general se usan cadenas de bits y el operador es un bit-flip probabilista. El cruce consiste en mezclar información de los padres para obtener nueva solución

en base a la información conocida. Mutación es operación unaria y en general tiene menos importancia que el cruzamiento que es n-ario.

1. Son características de la Programación Evolutiva:

V **El reemplazo probabilista**

Sí. La selección de sobrevivientes es probabilista - se utiliza la mutación entre padres e hijos ($\mu + \mu$)

F No. La representación como vectores de valores reales

Esto corresponde a las Estrategias Evolutivas. Programación evolutiva utiliza Máquinas de Estado Finito

V **La selección determinista de padres**

Sí. Se usa selección determinista de individuos de mejor calidad.

F La representación como áboles

No. Esto corresponde a las Programación Genética. Programación evolutiva utiliza Máquinas de Estado Finito

2. Respecto a la representación basada en lista de referencia, es cierto que:

V **Es aplicable a la resolución de problemas de asignación máquina-trabajo**

Sí. TSP es un caso especial de Job Shop de 1 solo trabajo, donde existe una secuencia específica de asignación máquina-trabajo. Por ello sí podemos usar la lista en TSP por extensión se puede usarla en Job Shop. Básicamente, podemos utilizar TSP resuelto con algoritmo genético para resolver JSP, donde cada máquina será considerada como una ciudad de TSP y cada trabajo - un viajero.

V **Permite generar soluciones factibles para el TSP usando el operador de cruce en dos puntos y uniforme**

Sí. Si bien en clases vimos TSP con cruce en un punto, podemos generalizar el cruce de un punto a dos. Además se puede usar el cruce uniforme, ya que al final la lista de referencia permite mantener la factibilidad.

F Permite intensificar adecuadamente con el operador de cruce

No, el operador de cruce si bien generalmente intensifica, en caso de usar una lista de referencia, genera información nueva, por lo que diversifica en vez de intensificar.

V **Genera soluciones factibles combinado con el operador de cruce intermedio**

Sí. Por ejemplo, en clases vimos que en TSP al codificar la solución en una lista de referencia esta deja de generar soluciones infactibles, como pasa con representaciones más clásicas de tour - lista de adyacencia o matriz binaria.

3. Respecto a un Algoritmo Genético Estándar es cierto que:

V **Implementa elitismo**

Sí. Debido a que selección por ruleta es elitista - se basa en la aptitud.

V **Utiliza cruce en un punto**

Sí. Usa cruce en punto fijo

- V Utiliza mutación bit-flip**
Sí. Usa cadenas de bits con mutación probabilista de cada bit
- F Utiliza selección basada en k-tournament**
No. Se utiliza la selección de ruleta o también ranking - proporcional a la aptitud.
-
4. Un algoritmo genético:
- F Diversifica aplicando elitismo**
No. Elitismo intensifica.
- V Intensifica con altas tasas de cruzamiento**
Sí. Cruzamiento ayuda a intensificar ya que crea nuevos individuos en base a la información ya existente - genera una solución intermedia entre dos soluciones conocidas
- V Intensifica con alta presión de selección**
Sí. Alta presión implica elegir solo los individuos de mejor calidad, por lo que se intensifica la solución dentro de la vecindad
- F Intensifica con altas tasas de mutación**
No. Mutación diversifica.
-
5. En los algoritmos genéticos:
- V El genotipo corresponde a la estructura de la solución**
Sí. El genotipo es la codificación de la solución y es lo que se transmite.
- V Fenotipo y genotipo pueden ser iguales**
Sí. Pueden ser iguales. Tal es el caso de problema de Knapsack donde el fenotipo y el genotipo coinciden.
- F Los operadores de transformación se aplican sobre el fenotipo de las soluciones**
No. Se aplican sobre el genotipo.
- F La aptitud se calcula sobre el genotipo de las soluciones**
No. La aptitud se calcula en base a las propiedades del fenotipo.
-
6. Respecto a la representación basada en lista de referencia, es cierto que:
- V Es aplicable a la resolución de problemas de asignación cuadrática**
QAP y TSP tienen formulación parecida, donde en QAP se analiza si se debe ubicar cierta tienda i en cierto lugar k , y en TSP si se toma el camino entre ciudades i y j . Entonces podemos utilizar TSP para resolver el problema de QAP, y como TSP se puede resolver con lista de referencia, entonces es aplicable en QAP.
- V Puede utilizarse en un algoritmo Tabu Search para encontrar soluciones para el TSP**
Sí, lista de referencia se utiliza en solución de TSP. Si bien la lista de referencia genera solución factible, Tabu Search puede intentar mejorarla.
- F No permite diversificar adecuadamente con el operador de cruzamiento**
El operador de cruzamiento precisamente termina generando soluciones con nueva información, es decir, termina diversificando si se usa la lista de referencia.

- V Solo genera soluciones factibles combinado con el operador de cruzamiento en un punto

Sí, la ventaja de uso de lista de referencia es evitar generar las soluciones infactibles.

7. Respecto a la representación basada en lista de referencia, es cierto que:

- V Es aplicable a la resolución de problemas de asignación 1-1

Sí. Por ejemplo, se puede aplicar en TSP o en JSP o en QAP.

- V Permite generar soluciones factibles para el TSP usando el operador de cruzamiento en dos puntos y uniforme

Sí. Si bien en clases vimos TSP con cruce en un punto, podemos generalizar el cruce de un punto a dos. Además se puede usar el cruce uniforme, ya que al final la lista de referencia permite mantener la factibilidad.

- F Permite intensificar adecuadamente con el operador de cruzamiento

No, el cruce con lista de referencia diversifica.

8. El operador de selección k-tournament (por torneo):

- V Es un operador de selección extintivo si todos los individuos tienen diferente aptitud

Sí. Si el K de torneo es suficientemente grande y la población es muy diversa, el algoritmo es sesgado a elegir mejores individuos de la población. Es decir, es elitista y por ello es extintivo - no va a permitir que las peores soluciones pasen a la siguiente generación.

- V Su presión de selección crece a medida que crece el valor de k

Sí. A mayor k, más probable es elegir solamente los mejores individuos dentro de la población.

- F Se puede utilizar solo en problemas de minimización

No. No tiene un problema predeterminado

- F Es un operador de selección determinista

No. Es completamente probabilista.

9. El operador de selección k-tournament (por torneo):

- V Es un operador de selección extintivo si todos los individuos tienen diferente aptitud

Sí. Si el K de torneo es suficientemente grande y la población es muy diversa, el algoritmo es sesgado a elegir mejores individuos de la población. Es decir, es elitista y por ello es extintivo - no va a permitir que las peores soluciones pasen a la siguiente generación.

- F Su presión de selección disminuye a medida que crece el valor de k

No. La presión crece al aumentar el k ya que aumenta la probabilidad de elegir solamente a los mejores individuos

- F Se puede utilizar solo en problemas de maximización

No. No tiene un problema predeterminado

- V En el primer paso selecciona los k competidores con una probabilidad de selección uniforme
Sí. Selecciona aleatoriamente (uniformemente) a los k competidores para hacer la comparación.
-
10. El operador de mutación en un Algoritmo Genético:
- V Puede ser el mismo movimiento que usa un algoritmo reparador
Sí. El movimiento de algoritmo reparador altera los valores obtenidos, lo cual es lo mismo que hace la mutación. De hecho, en ambos casos se tiende a usar bit-flip. Incluso, los GA se consideran algoritmos reparadores.
- V Se aplica de acuerdo a una probabilidad de uso, por lo que puede generar más cambios en una solución que un movimiento en un algoritmo reparador
Sí, es posible que se den más cambios (como también es posible que se den menos). La cantidad de cambios en un algoritmo reparador depende del estado de búsqueda.
- F Siempre debe generar mejoras
No, mutación diversifica - de hecho, introduce peores soluciones para evitar estancarse en óptimos locales.
- F Se aplica siempre a todas las soluciones de la población
No, su aplicación depende de la probabilidad asignada, por lo que algunas soluciones pueden generarse sin aplicar nunca la mutación.
-
11. A diferencia de las Estrategias Evolutivas, las estrategias de Programación Genética:
- F Usan perturbación gaussiana como operador de mutación
No. Programación genética no usa la perturbación gaussiana - recombina los árboles. Son las estrategias evolutivas que ocupan la perturbación gaussiana.
- F Representan las soluciones como vectores de valores reales
No. Programación genética utiliza árboles.
- F No realizan cruzamiento en ninguna de sus versiones
No. Programación genética sí ocupa el cruzamiento. Las estrategias evolutivas también ocupan el cruzamiento (introducido por Schwefel, 1981)
- V Realizan reemplazo generacional
Sí. La programación genética usa el reemplazo generacional, es decir, la población es sustituida por completo por los descendientes. En las estrategias evolutivas existe selección de descendientes (determinista) que entran a la población.
-
12. Respecto a la representación basada en lista de referencia, es cierto que:
- V/N Es aplicable a la resolución del problema de las n-reinas cuando se consideran todas las restricciones (fila/columna/diagonal)
*Aquí la respuesta depende de qué quería decir la pregunta. Se puede aplicar en n-Reinas porque n-Reinas se resuelve con algoritmos GA. Sin embargo, la lista de referencia permite mantener solamente la factibilidad de no poner reinas en misma fila o en misma columna, pero **no logra mantener la factibilidad de la diagonal.***

V Es aplicable a la resolución del problema de las n-reinas siempre y cuando se consideren las restricciones de fila y columna

Sí. Para modelo fila y modelo columna la variable x_i nos indica en cual fila o columna esta la i -ésima reina. Por ejemplo, para modelo fila con configuración $(8, 6, 4, 1, 3, 5, 7, 2)$:

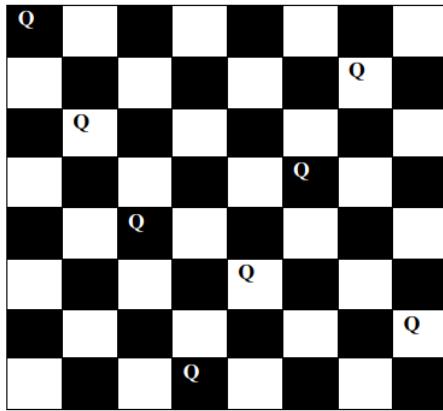


Fig.1: Arrangement signifying tuple: $(8, 6, 4, 1, 3, 5, 7, 2)$

Obtenemos una solución entonces, donde cada 8 indica la fila de la reina 1, 6 la fila de la reina 2, y así sucesivamente. Podemos ver que se puede resolver este problema aplicando Algoritmos Genéticos con cruzamiento. Sea Padre 1 = $(5\ 2\ 3\ 1\ 6\ 4\ 8\ 7)$ y Padre 2 = $(1\ 8\ 6\ 4\ 7\ 5\ 3\ 2)$, crucémoslos usando punto fijo:

Padre1: 5 2 3 1 6 4 8 7

P1(c)	5	2	2	1	2	1	2	1
-------	---	---	---	---	---	---	---	---

Padre2: 1 8 6 4 7 5 3 2

P2(c)	1	7	5	3	4	3	2	1
-------	---	---	---	---	---	---	---	---

Cruzando punto fijo:

Padre 1(c)

5	2	2	1	4	3	2	1
---	---	---	---	---	---	---	---

Padre 2(c)

Explicación de codificación de Padre 1 (Padre 2 se hace de la misma manera)

[1,2,3,4,5,6,7,8] para 5 la posición es 5

[1,2,3,4,6,7,8] para 2 la posición es 2

[1,3,4,6,7,8] para 3 la posición es 2

[1,4,6,7,8] para 1 la posición es 1

[4,6,7,8] para 6 la posición es 2

[4,7,8] para 4 la posición es 1

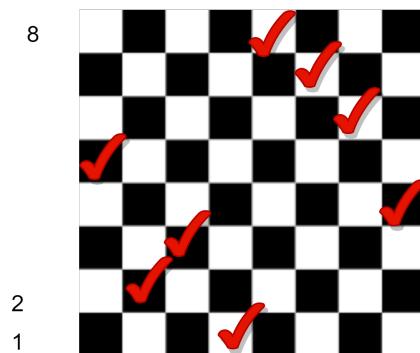
[7,8] para 8 la posición es 2

[7] para la posición es 1

Decodificando el resultado de cruce con el proceso inverso (arreglo de cruce contiene posiciones en la lista de referencia)

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

- [1,2,3,4,**5**,6,7,8]
- [1,2,3,4,6,7,8]
- [1,3,4,6,7,8]
- [1,4,6,7,8]
- [4,6,7,8]
- [4,6,7]
- [4,6]
- [4]



Solución visualizada
 NO SE MANTIENE
 RESTRICCIÓN DE LA DIAGONAL
 SÍ SE MANTIENE
 RESTRICCIÓN FILA Y COLUMNA

Es decir, la lista sirve para generar solución factible que mantiene restricción fila o restricción columna (dependiendo de qué representación se ocupa), pero no logra mantener la restricción de diagonal.

- F Es aplicable a la resolución de la versión CSOP del problema del colooreo de grafos

No. Coloreo de grafos CSP considera un conjunto dado de colores y dice si es que se puede colorear el grafo con estos. La lista de referencia no sirve en este caso porque no permite mantener la factibilidad de la única condición de 'nodos unidos por arco deben ser de distinto color' ya que una lista de referencia no permite tener la información de vecinos

- F Es aplicable a la resolución de la versión CSP del problema del colooreo de grafo
- No. Coloreo de grafos CSOP busca minimizar la cantidad de colores para colorear el grafo. Si no se puede resolver CSP, tampoco se puede resolver CSOP

con esta estructura.

13. Considerando la adaptación de un Algoritmo Genético Estándar para los problemas de la mochila y la mochila multi-dimensional:

V En ambos casos se puede mantener el operador de selección

Sí. El operador de selección proporcional a la aptitud sirve en ambas versiones del problema.

F En ambos casos se debe cambiar el operador de selección

No. El operador original sirve.

V En ambos casos se puede mantener la representación

Sí. La cadena de bits sirve para ambos casos, solo que en mochila multidimensional se trabajará con muchas dimensiones (muchas cadenas o matriz)

F Sólo en el problema de la mochila clásico se puede mantener la representación

No. La cadena de bits sirve para ambos casos.

9. Asignación de Parámetros

Problema de asignación de parámetros surge de la idea de que para un mismo problema existen muchas distintas *configuraciones* que contemplan un seteo de parámetros en cierto valor. Por ejemplo, para algoritmos genéticos tenemos tales parámetros como tipo de mutación, tipo de cruzamiento, estrategia de selección de individuos, probabilidad de mutación, tamaño de población, etc.

Como cada parámetro tiene su propio dominio de valores que puede tomar, el espacio de búsqueda para el problema crece. Nos interesa encontrar aquellas instanciaciones que mejoran la calidad de solución.

Entre algunas características del Problema de Seteo de Parámetros se encuentran:

- Es difícil asignar los parámetros debido a la naturaleza estocástica de las metaheurísticas - es complicado saber cómo se va a comportar la metaheurística
- Cada instancia puede tener características propias que desempeñan mejor con cierto conjunto de valores para los parámetros (configuración)
- Hay que tener en cuenta que existen parámetros interrelacionados y también aquellos que varían durante la ejecución (ej. tiempo T de Simulated Annealing)
- Es un problema de Optimización Combinatoria

Las técnicas de asignación de parámetros se clasifican según la siguiente taxonomía:

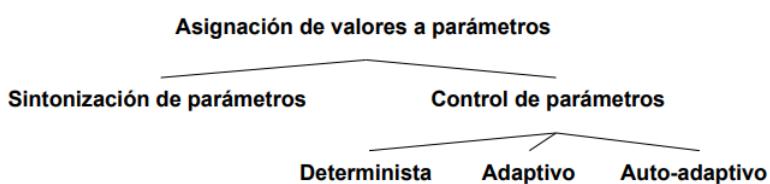


Figura 50: Taxonomía de técnicas de Asignación de Parámetros

9.1. Sintonización de Parámetros

Esta técnica consiste en la idea de no modificar los parámetros durante la ejecución del algoritmo.

1. Costoso en tiempo ya que requiere ejecutar el algoritmo muchas veces para probar distintas configuraciones.
2. Mantiene los valores de parámetros fijos durante la ejecución
3. Tres tipos de parámetros:
 - Numericos: probabilidad de cruzamiento, tamaño de población, etc. Permite definir métricas de distancia para analizar los valores.
 - Categórico: tipo de cruzamiento, tipo de mutación, etc.
 - Condicional: aquellas que se activan solo si otro parámetro esta activado. Por ejemplo, se activa el parámetro k solo si se activa la selección de población por k-torneo.

Existen varios tipos de sintonización:

- **Sintonización por analogía** - ocupa los resultados de los experimentos de los investigadores. Es decir, es una técnica que se basa en uso de valores de parámetros entregados en la literatura.
- **Sintonización Estadística** - técnica que prueba las configuraciones de manera sistemática obteniendo las métricas de desempeño de cada una. En base a estas métricas se seleccionan las mejores configuraciones.
- **Sintonización Mediante Meta-Algoritmos** - técnica que ocupa un meta-algoritmo que se comunica con la heurística principal. Este algoritmo va probando distintos valores de parámetros y retorna las mejores configuraciones para un set de instancias en particular.

El proceso de sintonización se resume con la siguiente esquema:

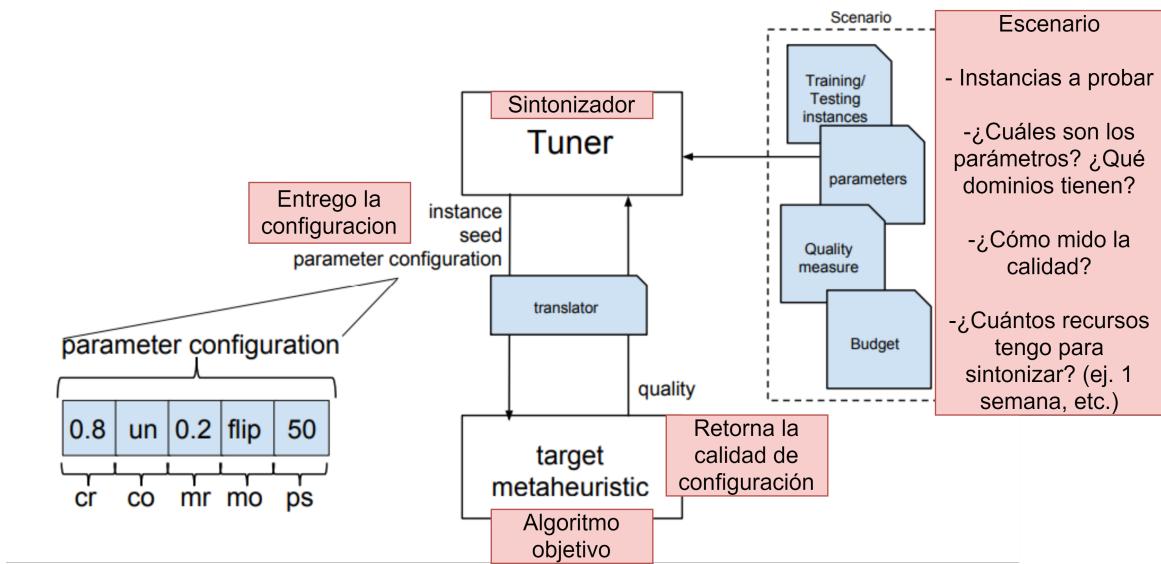


Figura 51: Esquema de proceso de sintonización

9.1.1. F-Race

Algoritmo *iterativo* basado en las carreras. En cada paso se evalua un conjunto de configuraciones para una nueva instancia del problema. Se eliminan aquellas configuraciones que desempeñan peor. La idea es **descartar las peores configuraciones y seguir evaluando solo las configuraciones más prometedoras**.

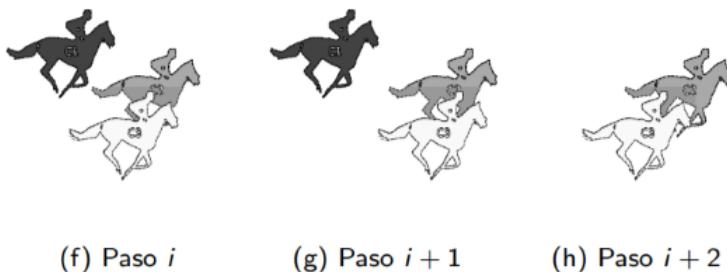


Figura 52: Se elimina al peor candidato de manera iterativa

Para comparar el desempeño del conjunto de configuraciones se utiliza el **test de Friedman**, comparando el conjunto con una instancia-semilla y viendo el desempeño. Se usa contraste de hipótesis, donde la hipótesis nula es: *todas las ranking de configuraciones están igualmente distribuidos*.

$$T = \frac{(n-1) \sum_{j=1}^n \left(R_j - \frac{m(n+1)}{2} \right)^2}{\sum_{l=1}^m \sum_{j=1}^n R_{lj}^2 - \frac{mn(n+1)^2}{4}}$$

- m : Paso de la carrera.
- n : Número de configuraciones restantes.
- R_{lj} : Ranking de la configuración c_j en el bloque l , $R_j = \sum_{l=1}^m R_{lj}$.
- T se distribuye χ^2 con $n-1$ grados de libertad.
- Si el T observado sobrepasa el cuartil $(1-\alpha)$ de la distribución, la hipótesis nula es rechazada con un nivel de aproximación α .
 - En este caso, la hipótesis alternativa que al menos una configuración candidata muestra mejor desempeño que al menos una de las otras es aceptada.

$$\frac{|R_j - R_h|}{\sqrt{\frac{2m\left(1 - \frac{T}{m(n-1)}\right)\left(\sum_{l=1}^m \sum_{j=1}^n R_{lj}^2 - \frac{mn(n+1)^2}{4}\right)}{(m-1)(n-1)}}} > t_{1-\frac{\alpha}{2}} \quad (2)$$

```

Procedure F-Race
  generar el conjunto de configuraciones candidatas  $C$ 
  carrera  $\leftarrow 1$ 
  while not se cumpla criterio de término do
    seleccionar aleatoriamente una instancia  $i$  desde el conjunto de instancias  $\mathcal{I}$ 
    foreach  $c_j \in C$  do
      agregar el costo de ejecutar  $c_j$  en  $i$  a  $Cost(c_j)$ 
    endfor
    foreach  $c_j \in C$  do
      foreach instancia  $l$  ejecutada do
         $R_{lj} \leftarrow$  ranking de la configuración  $c_j$  en el block  $l$ 
      endfor
       $R_j \leftarrow$  suma de rankings sobre todas las instancias de  $c_j$ 
    endfor
    if carrera  $> r$  then
      hacer el test de Friedman de acuerdo a la ecuación 11
      if la hipótesis nula asociada a  $T$  es rechazada then
         $c_{best} \leftarrow$  configuración en  $C$  con mínimo  $R$ 
        foreach  $c_j \in C \setminus c_{best}$  do
          hacer comparación de a pares de acuerdo a la ecuación 2
          if se rechaza la hipótesis nula then  $C \leftarrow C \setminus c_j$ 
        endfor
      endif
    end while
  return conjunto restante  $C$ 

```

Figura 53: Pseudocódigo de F-Race

Además, el algoritmo define meta-parámetros:

Meta-Parámetro	Descripción	Valor
r	carreras sin eliminación	7
α	nivel de confianza	0.05
max_execs	Número máximo de ejecuciones	1000

9.1.2. ParamILS

PARAMeter Iterated Local Search method. Se inicia con una configuración por defecto e iterativamente se busca su mejora, es decir, se busca el óptimo local más cercano y se avanza a este. El vecindario se define como variación de un solo parámetro.

Existen dos tipos de ParamILS, que difieren en cómo consideran una solución mejor que otra

- BasicILS - usa cantidad fija N de instancias para comparar dos configuraciones c y c'

- FocusedILS - introduce el concepto de **dominancia**. Se dice que c' domina a c si el desempeño promedio al usar la configuración c' para n' instancias es mejor que usar la configuración c para n instancias, con $n < n'$

```

Procedure ParamILS
   $c_0 \leftarrow$  configuración por defecto
  For  $i \leftarrow 1$  To  $R$  Do
     $c \leftarrow$  configuración aleatoria
    If mejor( $c, c_0$ ) then  $c_0 \leftarrow c$ 
  Endfor
   $c_{ils} \leftarrow \text{IterativeFirstImprovement}(c_0, \mathcal{N})$ 
  While not se cumple criterio de término Do
     $c \leftarrow c_{ils}$ 
    For  $i \leftarrow 1$  To  $s$  Do
       $c \leftarrow$  configuración aleatoria en  $\mathcal{N}$ 
       $c \leftarrow \text{IterativeFirstImprovement}(c, \mathcal{N})$ 
      If mejor( $c, c_{ils}$ ) Then  $c_{ils} \leftarrow c$ 
    If  $p_{restart}$  Then  $c_{ils} \leftarrow$  configuración aleatoria
  Endwhile
  Return mejor global  $c$ 
End

```

- La función $\text{IterativeFirstImprovement}(c, \mathcal{N})$ busca aleatoriamente en el vecindario de c por una configuración de mejor calidad.

```

Procedure IterativeFirstImprovement( $c, \mathcal{N}$ )
Begin
Repeat
   $c' \leftarrow c$ 
  Foreach  $c'' \in \mathcal{N}(c')$  en orden aleatorio Do
    If mejor( $c'', c'$ ) Then  $c \leftarrow c''$ 
    Break
  Until  $c' = c$ 
  Return  $c$ 
End

```

Además, el algoritmo define meta-parámetros:

Meta-parámetro	Descripción	Valor
R	soluciones aleatorias de la primera fase	10
s	soluciones aleatorias de cada iteración	3
$p_{restart}$	probabilidad de reinicio	0.01
max_execs	máximo número de ejecuciones	1000

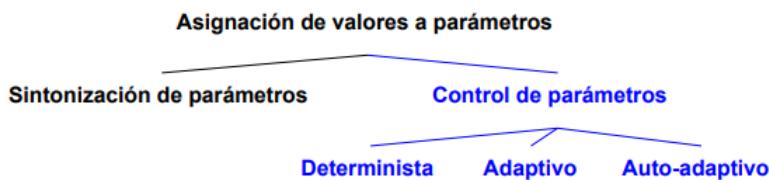
9.2. Control de Parámetros

Técnica que se basa en permitir que los valores de los parámetros varían durante la ejecución del algoritmo. Por ejemplo, para Tabu Search al inicio podemos partir con lista Tabu larga, que irá achicando con el tiempo para intensificar.

Algunos puntos importantes son:

- Es técnica difícil de diseñar - se debe mantener el balance entre, por ejemplo, diversificación e intensificación de las búsquedas
- Las soluciones son difíciles de generalizar, porque tienden a ser ajustadas a un problema en particular
- Se debe definir con cuidado cómo se inicializan los valores y cuando se debe gatillar la actualización de estos

Se resume la taxonomía de las técnicas de Control de Parámetros:



Las técnicas de control de parámetros se clasifican según los criterios:

- Parámetros del problema - ¿qué es lo que se cambia?
- ¿Cómo se cambian los parámetros?
 1. **Determinista** - regla definida por número de pasos o períodos de tiempo. Por ejemplo, Simmulated Annealing cambia su T después de cierta cantidad de iteraciones.
 2. **Adaptivo** - considera la retroalimentación global del algoritmo de búsqueda. Por ejemplo, se usan métricas globales como la diversidad o la convergencia del algoritmo.
 3. **Auto-adaptivo** - considera la retroalimentación individual del algoritmo de búsqueda. Por ejemplo, en algoritmos genéticos se puede asignar el valor propio de mutación a cada individuo.
- Evidencia de cambio ocurrido
 1. **Relativa** - por ejemplo, relativa a la población actual.
 2. **Absoluta** - por ejemplo, distancia al óptimo del problema.

En el Estado de Arte de Control de Parámetros existe una gran variedad de técnicas debido a la cantidad de parámetros y cambios que se puede seleccionar para un problema en particular. Destacan las siguientes técnicas:

- Control de Tasa de Uso de operadores
- Control de Tamaño de Población
- Control de Parámetros en Colonias de Hormigas (Ant Colony)
- Control de Parámetros en Búsqueda Local

9.2.1. Control en Estrategias Evolutivas

Se propone la representación de un individuo a partir de un arreglo:

$$\langle x_1, x_2, x_3, \dots, e_1, e_2, e_3, \dots \rangle$$

Donde x_i son variables del problema, y los e_i - parámetros estratégicos.

Como en Estrategias Evolutivas se trabaja con la mutación, se introduce la operación:

$$x'_i = x_i + N(0, \sigma_i)$$

que permite sumar (o restar) un valor aleatorio entre 0 y la varianza σ_i para mutar el valor x_i .

Considerando que la varianza ahora es un valor estratégico, la representación final queda como:

$$\langle x_1, x_2, x_3, \dots, \sigma_1, \sigma_2, \sigma_3, \dots \rangle$$

El control de varianza que se introduce se conoce como **Regla de 1/5 de Rechenberg**. Esta dice que se espera que 1 de cada 5 mutaciones generen una mejora. Se define la varianza de siguiente paso $t + 1$ de la siguiente manera:

$$\sigma^{t+1} = \begin{cases} i \cdot \sigma^t & \text{si } p^t > \frac{1}{5} \\ d \cdot \sigma^t & \text{si } p^t < \frac{1}{5} \\ \sigma^t & \text{si } p^t = \frac{1}{5} \end{cases}$$

Donde p^t - éxito en últimas h generaciones, i - valor de incremento de paso y d - valor que disminuye el salto.

- Si la solución se mejoraba en los últimos pasos (más mejoras que $\frac{1}{5}$), entonces si intenta dar pasos más grandes en mutación con $i \cdot \sigma^t$.
- Si se empeoraba, entonces se disminuye el paso con $d \cdot \sigma^t$.
- Si se cumplió que habían exactamente $\frac{1}{5}$ mejoras, entonces mantenemos la varianza.

Cuando se introducen más de una población, se empiezan a analizar las técnicas de cruceamiento para elegir con cual individuo quedar (a o b):

- Versiones ES posteriores usan una población de mayor tamaño.
 - Es posible implementar un operador de cruzamiento.

$$x'_i = \begin{cases} x_{a,i}, \chi \leq 1/2 \\ x_{b,i}, \chi > 1/2 \end{cases} \quad (5)$$

$$\sigma'_i = \begin{cases} \sigma_{a,i}, \chi \leq 1/2 \\ \sigma_{b,i}, \chi > 1/2 \end{cases} \quad (6)$$

con $a = (x_a, \sigma_a)$, $b = (x_b, \sigma_b)$.

- Los últimos acercamientos en ES incorporan un vector adicional.
 - Correlaciones entre los pasos de mutación.

Figura 54: Otras técnicas para más de una población

9.2.2. Control en Simmulated Annealing

En simmulated annealing existen varias maneras de control de la temperatura.

El control a través de **Recalentamiento como función de costos** usa la calidad de solución $C(X)$ y el parámetro P decidido como aumentar la temperatura con la intención de diversificar:

$$T_{new} = P \cdot C(X) + T_{max}$$

Algunas otras técnicas de control de temperatura de SA son:

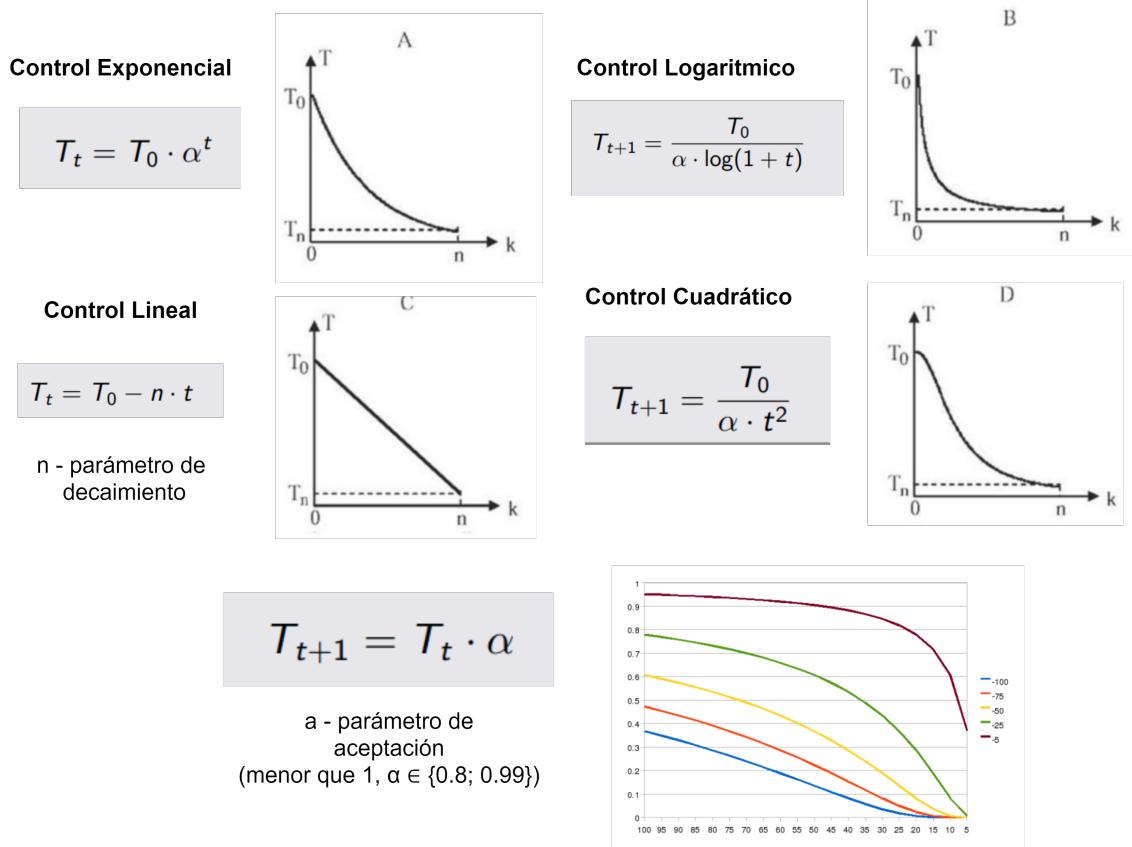


Figura 55: Distintas técnicas de control de T en Simmulated Annealing

9.2.3. Control en Tabu Search para Asignación Cuadrática (QAP)

El problema de QAP (Problema de Asignación Cuadrática (tráfico entre pares de tiendas)) considera la técnica de **Self Control Tabu Search** o **Tabu Tenure** con mecanismo para controlar el largo de la lista Tabu.

- Lista Tabu es una matriz (units x location) donde la celda tl_{ij} almacena la iteración cuando se movió la unidad i a la locación j
- Se usa un metaparámetro l_r^{ref} de la cantidad de iteraciones esperadas. El largo de la Lista (tabu tenure) es controlado considerando qué tan frecuentemente se repiten

soluciones: $e_{k+1} = l_r^{ref} - l_r(k)$, donde $l_r(k)$ es la cantidad de iteraciones que han pasado desde la última repetición de la solución actual

- El tamaño de la lista cambia: $ts_{k+1} = ts_k + K_i \cdot e_{k+1}$

La idea consiste en agrandar la lista si hay repeticiones de solución ya que estamos haciendo ciclos y, entonces, debemos *diversificar* para salir de estos. La lista se disminuye si hay menos repeticiones.

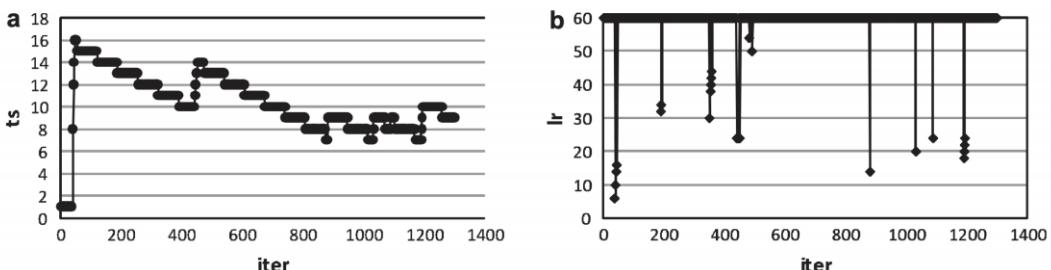


Fig. 3. Adaptive control behavior: (a) Tabu tenure vs. iterations. (b) Repetition length vs. iterations.

- ① Algoritmo inicia con tabu tenure en $ts=1$.
- ② Suceden las primeras repeticiones provocando el aumento de ts
- ③ Disminuyen las repeticiones (considerando una ventana de tiempo de 60 iteraciones), por ende disminuye ts

Figura 56: Ilustración de largo vs iteraciones, repeticiones vs iteraciones

9.3. Preguntas del Examen/Control

1. ¿Por qué se considera que el problema de Asignación de Parámetros es un problema de optimización combinatoria?

Porque existe una gran cantidad de parámetros que se pueden elegir, una gran cantidad de posibles configuraciones que se pueden crear, creando un espacio de búsqueda muy grande. Como meta se desea optimizar la configuración seleccionada de modo tal que permita obtener la mejor calidad de la solución, por lo que es un problema de optimización combinatoria.

2. Mencione 3 parámetros del problema Simmulated Annealing

Temperatura, parámetro α de enfriamiento (en caso geométrico), parámetro n de decaimiento (en caso lineal), parámetro P de control de costo (recalentamiento).

3. Mencione 3 parámetros del Algoritmo Genético

Probabilidad de mutación, probabilidad de cruzamiento, tamaño de población

4. ¿Cuál es la diferencia entre controlar y sintonizar?

Sintonización prueba distintas configuraciones antes de ejecutar el algoritmo de verdad. Es decir, los parámetros se mantienen fijos durante la ejecución. Control de parámetros modifica los parámetros durante la ejecución bajo cierto criterio definido.

5. Mencione los tipos de parámetros de la sintonización con ejemplos
Numéricos (α y T de SA), Categóricos (tipo de cruzamiento, tipo de mutación), Condicionales (si se elige selección por k-torneo, entonces se selecciona el valor de k)
6. Mencione 2 parámetros para el problema de Tabu Search
Largo de la lista Tabu (numérico), criterio de aspiración (categórico), número de iteraciones (numérico), etc.
7. ¿Qué se entiende por la sintonización por analogía?
Sintonización que se basa en los valores obtenidos en los experimentos de los investigadores. Básicamente, es técnica que se basa en las configuraciones encontradas en la literatura.

1. La variación de la temperatura en un algoritmo Simulated Annealing:

- V Puede clasificarse como una estrategia de control determinista
Sí, SA es determinista ya que control determinista consiste en cambiar el parámetro cada cierto tiempo o cantidad de iteraciones. SA cambia su temperatura T cada cierta cantidad de iteraciones
- F Puede clasificarse como una estrategia de control adaptivo
No, SA no necesita tener retroalimentación para cambiar la temperatura
- F Puede clasificarse como una estrategia de control auto-adaptivo
No, SA no necesita tener retroalimentación para cambiar la temperatura
- F No corresponde a una estrategia de control de parámetros
No, SA corresponde al control determinista (de su parámetro T)

2. Una estrategia de control determinista:

- V Permite saber de antemano los valores que tomarán los parámetros durante el proceso de búsqueda
Sí, ya que para el control determinista se define de antemano una regla que cambia los valores de parámetros, bajo el criterio de cierto tiempo o cantidad de iteraciones.
- F Sólo se puede implementar en algoritmos reparadores trayectoriales
No. Si bien el uso de control determinista es común en algoritmos reparadores (como SA), se puede aplicar el control determinista también en Estrategias Evolutivas. Por ejemplo, se puede proponer un esquema determinista para adaptar las mutaciones.
- F Codifica los valores de los parámetros en cada solución del proceso de búsqueda
No, los valores de parámetros se obtienen a partir de una regla predefinida, no se eligen durante la ejecución del algoritmo.
- F Realiza cambios en los valores basado en retroalimentación desde el proceso de búsqueda
No, el control determinista no usa retroalimentación. Esta se utiliza en control adaptivo.

3. La dificultad del problema de asignación de valores a parámetros se debe a:
- V **La interrelación entre los parámetros del algoritmo**
Sí, se producen relaciones entre los grupos de parámetros que dificultan el análisis de posibles efectos que puede producir algún cambio de algún parámetro.
 - V **El tiempo requerido para el proceso**
Sí, antes de ejecutar el algoritmo “de verdad” se hace una especie de proceso de aprendizaje. Durante este el algoritmo se ejecuta muchas veces para encontrar la mejor configuración de parámetros
 - F **La complejidad computacional de los algoritmos de sintonización**
No, no es la dificultad de la sintonización que aumenta la complejidad, sino más bien el problema en sí que se evalúa y su naturaleza.
 - F **La naturaleza determinista de los algoritmos**
No, no todos los algoritmos son deterministas. Si lo fuesen, el problema de asignación de hecho sería más fácil.
-
4. La dificultad del problema de asignación de valores a parámetros se debe a:
- V **La interrelación entre los parámetros del algoritmo**
Sí, se producen relaciones entre los grupos de parámetros que dificultan el análisis de posibles efectos que puede producir algún cambio de algún parámetro.
 - V **La naturaleza estadística del proceso**
Sí, antes de ejecutar el algoritmo “de verdad” se hace una especie de proceso de aprendizaje para adquirir las estadísticas de cómo funcionan los parámetros.
 - V **La necesidad de valores de parámetros particulares para cada instancia**
Sí, muchas veces las instancias tienen propiedades propias que deben ser analizadas. No toda instancia funciona bien con una configuración particular.
 - V **La naturaleza estocástica de los algoritmos de búsqueda local**
Sí, debido a que las metaheurísticas son probabilistas es difícil decidir qué parámetros se deben seleccionar, o si deben cambiar en ejecución, etc. Precisamente por esto existe el problema de Asignación de Parámetros.
-
5. El método de control de parámetros de las Estrategias Evolutivas
- V **Actualiza los valores de parámetros de acuerdo a la tasa de éxitos de las últimas iteraciones**
Sí. El parámetro de ES es la varianza y se actualiza usando la Regla de 1/5 de Recheberg.
 - V **Almacena en cada solución los parámetros controlados**
Sí. Cada individuo tiene forma $a = (x_a, \sigma_a)$, por lo que almacena el parámetro controlado (la varianza)
 - F **Puede clasificarse como una estrategia de control auto-adaptivo**
Sí, los parámetros controlados se guardan en los individuos. Además, se utiliza el feedback de éxito de las últimas h generaciones.

F Puede clasificarse como una estrategia de control adaptiva

No, es más bien autoadaptivo debido a que se asigna la mutación a cada individuo. Sin embargo se puede considerar que el éxito de últimas generaciones se puede considerar como el estadístico global.
