uc3m | Universidad Carlos III de Madrid

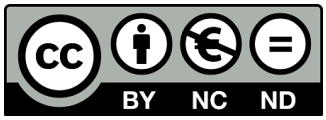
Grado en Ingeniería Informática

2020-2021

Apuntes

Heurística y Optimización

Jorge Rodríguez Fraile[1]



[1]Universidad: 100405951@alumnos.uc3m.es | Personal: jrf1616@gmail.com

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

1. INFORMACIÓN

1.1. Profesores

Magistral: Carlos Linares López

Reducido: Francisco Javier García Polo

2. TEMA 1 PROGRAMACIÓN LINEAL

2.1. Programación Lineal

2.1.1. Representación grafica

Un problema de Programación Lineal está en Forma Canónica si y solo si:

1. El objetivo es de la forma de maximización.
2. Si todas las restricciones son desigualdades son del tipo $\leq$.
3. Si todas las variables de decisión son no negativas.

$$
\begin{aligned}
\text{máx } z = c_1x_1 + c_2x_2 + \ldots + c_nx_n & \\
ba_{11}x_1 + a_{12}x_2 + \quad \ldots \quad + a_{1n}x_n &\leq b_1 \\
a_{21}x_1 + a_{22}x_2 + \quad \ldots \quad + a_{2n}x_n &\leq b_2 \\
\ldots & \\
a_{m1}x_1 + a_{m2}x_2 + \quad \ldots \quad + a_{mn}x_n &\leq b_m \\
\text{donde } x_i \geq 0, \forall i = 1, \ldots, n &
\end{aligned}
$$

- Nomenclatura:
  - z representa la función a maximizar.
  - Las restricciones son para un sujeto a.
  - Cada fila representa una restricción.
  - Las b's son términos escalares, racionales que deben ser menores. -
  - Las c's son coeficientes, números racionales.
- Algebraicamente:
  - OJO: Todas deben ser del mismo tipo < o >

$$
\begin{aligned}
\text{máx } z &= C^T x \\
Ax \leq b \quad , \quad & ejem : \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \leq \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \\
\text{donde } x_i &\geq 0, \forall i = 1, \ldots, n
\end{aligned}
$$

  - z: función objetivo.
  - C: coeficientes de la función objetivo. nx1
  - X: variables de decisión. nx1
  - A: matriz de coeficientes tecnológicos. mxn
  - b: recursos. mx1

Desarrollo

Representar todas las restricciones en un plano como rectas, también x, y>0.

- Después de trazar las rectas dar valor a las variables de decisión y ver que hiperplano es el que cumple cada restricción y el área que encierren todas es la Región Factible.

Por el teorema de Dantzig:

- La Región Factible es siempre un poliedro convexo. Por lo tanto, uno de los vértices es la solución óptima.

Solo evaluamos los puntos extremos, por lo que hay que hallar las intersecciones de las rectas si todavía no las conocemos.

- Para hallar una intersección:
  - Se hace un sistema con ambas ecuaciones de recta. Los valores obtenidos son las intersecciones.

$$\textit{Este es el método} \begin{cases} \begin{cases} -2x + y = -8 \\ -x + 6y = 18 \end{cases}, \begin{pmatrix} -2 & 1 \\ -1 & 6 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} -8 \\ 18 \end{pmatrix} \\ \begin{pmatrix} -2 & 1 \\ -1 & 6 \end{pmatrix}^{-1} \begin{pmatrix} -2 & 1 \\ -1 & 6 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} -2 & 1 \\ -1 & 6 \end{pmatrix}^{-1} \begin{pmatrix} -8 \\ 18 \end{pmatrix} \\ \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} -8 \\ 18 \end{pmatrix} \end{cases}$$

  - $x = A^{-1}b$; A es la matriz de coeficientes de las dos rectas y b los recursos de cada una.

Sustituimos los distintos puntos extremos (x, y, . . .) en la función objetivo.

Observamos todos los resultados y el máximo, será aquel de mayor valor.

- La solución óptima es la última vez que la curva de isobeneficio toca la región factible.

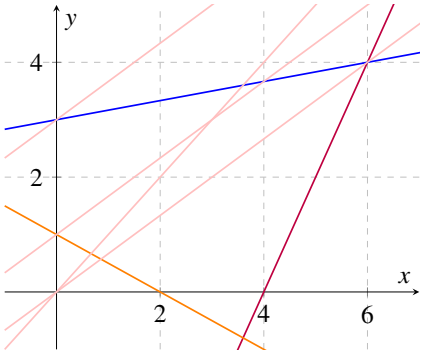


Fig. 2.1: Representación PL

Región factible

Es la intersección de las restricciones en forma de semiplanos. Son los infinitos puntos que cumplen las restricciones, cada uno es Solución factible.

Soluciones

Compatible Determinado Solución única.

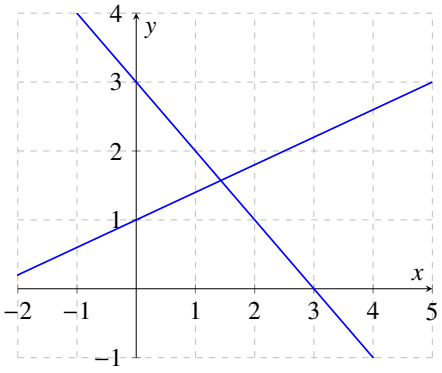


Fig. 2.2: Compatible Determinado

Compatible Indeterminado Soluciones infinitas, están superpuestas.

- Cuando la solución óptima no es única.
- Se detecta con el método de resolución gráfica si la curva de isobeneficio/isocoste es paralela o idéntica a una de las restricciones cuyos puntos extremos son soluciones óptimas.

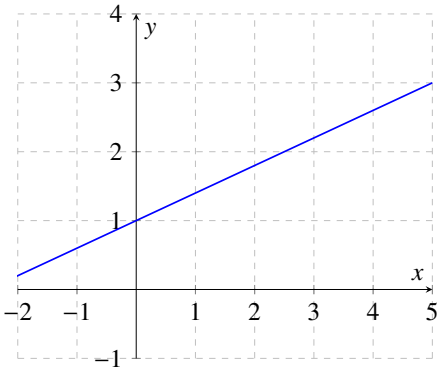


Fig. 2.3: Compatible Determinado

No acotado, faltan restricciones y hay soluciones infinitas

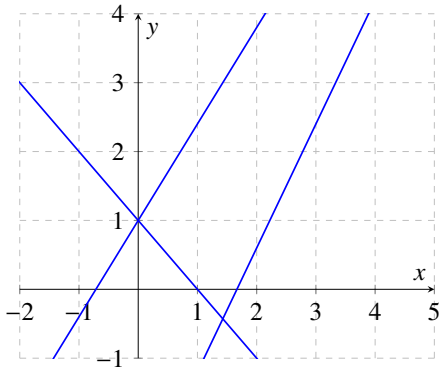


Fig. 2.4: Compatible Determinado

Incompatible:

- Infactible
    - Si y solo si la región de soluciones factibles es vacía: $F = \emptyset$, ya sea porque no cortan o porque cortan en zonas negativas.

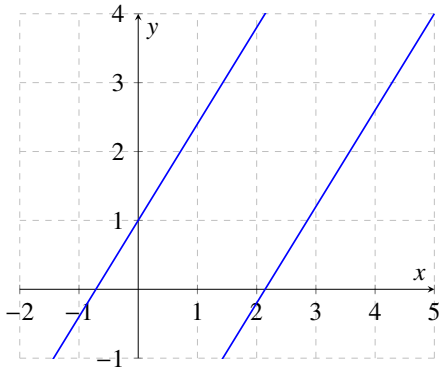


Fig. 2.5: Compatible Determinado

El método de resolución gráfica solo es posible para como mucho 3 variables de decisión.

2.1.2. Transformaciones

Pasar inecuaciones de un tipo a otro (para Forma Canónica)

$$\sum_{j=1}^{n} a_{ij}x_j \geqslant b_i \triangleq -\sum_{j=1}^{n} a_{ij}x_j \leq -b_i$$

Transformar maximización en minimización y viceversa:

$$\text{mín}\, z = C^T x \triangleq \text{máx}\, z = -C^T x$$

Quitar inecuación (para Forma Estándar)

$$\sum_{j=1}^{n} a_{ij}x_j \leqslant b_i \triangleq \sum_{j=1}^{n} a_{ij}x_j + s_i = b_i$$
$$\sum_{j=1}^{n} a_{ij}x_j \geqslant b_i \triangleq \sum_{j=1}^{n} a_{ij}x_j - s_i = b_i$$

s_i : *variables de holgura. Son variables de decisión cuando operamos.*

Poner inecuación a partir de igualdad:

$\sum_{j=1}^{n} a_{ij}x_j = b_i$ es $\sum_{j=1}^{n} a_{ij}x_j \leqslant b_i$ y $\sum_{j=1}^{n} a_{ij}x_j \geq b_i = -\sum_{j=1}^{n} a_{ij}x_j \leqslant b_i$

Si una variable de decisión x_i no está restringida se pone entonces como la diferencia de dos variables no negativas restringidas: $x_i = x_i' - x_i''; x_i', x_i'' \geq 0$

2.1.3. Método Simplex

Simplex: Poliedro convexo de n dimensiones.

Una tarea de Programación Lineal está en Forma Estándar (de maximización / minimización) si y solo si:

1. La función objetivo es de maximización (minimización, si se dice forma estándar se supone siempre maximización a menos que lo digan explícitamente).
2. Todas las restricciones son =.
3. Todas las variables de decisión son no negativas.
4. Todos los recursos son no negativos.

Algebraicamente:

$$\begin{aligned} &\text{máx } z = C^T x \\ &Ax = b \\ &x, y \geqslant 0 \end{aligned}$$

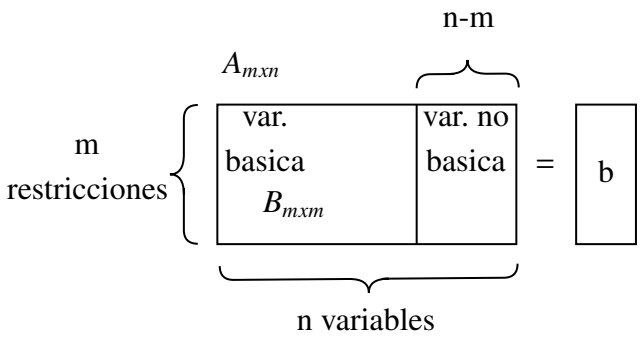


Fig. 2.6: Representación Simplex

Teorema George Dantzig: Dado una tarea de Programación lineal en forma estándar, el valor óptimo si lo hubiera, se alcanza en un punto extremo de la región factible.

Términos:

- Variable básica: Que no toma valor 0.
- Variable no básicas: Son las que toman valor 0.
- a_i: Vector columna, está formado por los coeficientes de x_i de las restricciones.
- c_i: Coeficientes de la x_i de la función objetivo.
- i: variables básicas.
- j: variables no básicas.

Tipos de soluciones:

- Definición 4

  Un vector $\mathbf{x_B} = \{x_{B1}, x_{B2}, \ldots, x_{Bm}\}$ se denomina solución básica si satisface $\mathbf{Bx_B} = \mathbf{b}$, donde $\mathbf{B}_{m\times m}$ es una submatriz de $\mathbf{A}_{m\times n}$ y todas las $(n - m)$ variables de decisión que no están en la base son nulas.

  Si además es factible, se denomina solución básica factible. Se dice que la base $\mathbf{B}_{m\times m}$, formada por las columnas a_i de $\mathbf{A}$ asociadas con las variables básicas x_{Bi} es una base factible sí $\mathbf{B}^{-1}\mathbf{b} \geq 0$.

  - Por lo tanto, $\mathbf{B}_{m\times m}$ debe ser no singular
  - Podría haber hasta $\left(\begin{array}{c} n \\ m \end{array} \right)$ soluciones básicas factibles
  - Típicamente se denotará $B = \{x_{B1}, x_{B2}, \ldots, x_{Bm}\}$

- Un vector x que satisface $Ax = b$ se llama Solución.
- Un vector x_B que satisface $Bx_B = b$ se llama Solución Básica.
- Un vector $x_B \geq 0$ que satisface $Bx_B = b$ se llama Solución Básica Factible.

Cálculo de la función objetivo: $z = C_B^T x_B$, solo se consideran las variables básicas.

Una solución factible óptima, x^*, si y solo si: $c^T x^* \geq c^T x; \forall \in xF$

Modelo iterativo:

1. Calcular una solución básica factible inicial.
2. Si existe un punto extremo adyacente que mejore z, transitamos a él.
3. En otro caso, detenerse.

Proceso normal:

- Cálculo de las variables básicas:

  1. Seleccionar una base mxm (m: número de restricciones), matriz cuadrada B, que tenga inversa y no haga los recursos negativos.
     - Tratamos de usar la matriz identidad.
     - Variables Artificiales: Para empezar con base matriz identidad.
       t_i se denomina Variable artificial que se añade a la función objetivo con coeficiente -M, para que no salga en la solución.
       Añadimos a una de las restricciones que tenga una variable de holgura que no nos interesa para hacer la matriz básica.
     - $B = \{x_i\}$, se recomienda escribir en orden de índice.
  2. Calculamos el valor de las variables básicas de la solución básica factible, $x_B = B^{-1}b$.
  3. Hallamos el valor de la función objetivo para esta solución básica factible, $z_B = C_B^T x_B$

- Selección de la variable de entrada: Buscamos un punto extremo adyacente que mejore la z, evaluamos las variables no básicas.

  1. Calculamos los costes reducidos para las variables no básicas, $z_j - c_j$

     $$z_j = C_B^T y_j$$
     $$y_j = B^{-1} a_j$$

  2. La variable de entrada será:
     - En max. se coge el más negativo y el proceso terminará cuando todos son positivos.
     - En min. se coge el más positivo y el proceso termina cuando todos los costes sean negativos.

- Regla de salida: Para que tenga dimensión m tenemos que sacar una variable básica de la base.

  - La variable que salga de la base, será: $min\{\frac{x_i}{y_{i'}}\}$, con $y_{i'} \geq 0$.

Casos de las distintas soluciones:

1. Dada la solución factible $x_B = B^{-1}b$ y $(z_j - c_j) > 0$.
   - Solución Óptima Única.
2. Dada la solución factible $x_B = B^{-1}b$ y $(z_j - c_j) > 0$ para todas las variables no básicas salvo una o más para las que $(z_j - c_j) = 0$.
   - Soluciones Óptimas Infinitas.
3. Dada la solución factible $x_B = B^{-1}b$ y $(z_j - c_j) < 0$ pero algún x_j no básica con $y_j \leq 0$.
   - Básicamente que al intentar sacar una variable todas las componentes no sean válidas, ya sea porque son divisiones entre 0 o entre negativos y dan negativos, ambas no válidas.
   - No Acotado.
4. Dada la solución factible $x_B = B^{-1}b$ y $(z_j - c_j) \geq 0$, pero alguna variable artificial toma valor positivo, que tenga valor en la solución. Variable artificial no es lo mismo que variable de holgura, las artificiales son para comenzar por la identidad y aportan negativamente a la función objetivo.
   - Infactible, región factible vacía.

2.1.4. Dualidad

Una tarea de Programación Lineal está en Forma Simétrica o Forma Canónica de maximización (o minimización si se dice explícitamente) si y solo si:

1. El objetivo es de la forma de maximización.
2. Si todas las restricciones son desigualdades son del tipo $\leq$.
3. Si todas las variables de decisión son no negativas.

El Problema Dual del Problema Primal (el canónico o simétrico):

$$\text{máx}\, z = C^T x A x \leq b x_i \geq 0 \qquad \text{es} \qquad \text{mín}\, w = b^T x' A^T x' \geq c x' \geq 0$$

Si la tarea de Programación Lineal en Forma Simétrica tiene una solución óptima correspondiente a una base B, entonces: $x'^* = c_B^T B^{-1}$

$$x'^* = c_B^T B^{-1} = \begin{pmatrix} -7 & 8 & 0 \end{pmatrix} \begin{pmatrix} 0 & \frac{1}{4} & 0 \\ -\frac{1}{2} & \frac{5}{8} & 0 \\ -\frac{1}{2} & \frac{23}{8} & 1 \end{pmatrix} = \begin{pmatrix} -4 & \frac{13}{4} & 0 \end{pmatrix}$$

- c y B, son las del problema resuelto, los que ya conocíamos. No las del problema simétrico.
- Teorema: la variable dual $x_i'^*$ indica la contribución al crecimiento de la función objetivo por unidad del recurso i-ésimo (de la primal, no la dual).
    - Nos permite saber cuándo aumenta la función objetivo si le sumamos 1 al recurso.

2.1.5. Interpretación de resultados

Interpretación de la solución.

1. Factible o Infactible. Se justifica con que no haya salido un valor positivo en las variables artificiales en la solución óptima.
2. Solución única o infinitas. Se justifica con que nos han salido costes reducidos estrictamente positivos en la última iteración.
3. Función objetivo acotable o no acotable. Se justifica con que el y_i de x_i no son negativos o nulos, por lo que se ha podido elegir una variable de salida.

Interpretación de los recursos.

1. Interpretación de las variables de holgura.
    - Si la variable de holgura suma, es que sobran recursos.
    - Si resta la variable de holgura es que falta recurso.
2. Dualidad: Contribución unitaria de cada recurso al crecimiento de la función objetivo.
    - Indicar de forma individual cuáles de las variables contribuyen, cuanto, y cuáles no.

2.1.6. Modelización

Problema de Transporte

Dados m orígenes y n destinos tal que:

a_i: capacidad del origen i, $1 \leq i \leq m$

b_i: demanda del destino j, $1 \leq j \leq m$

c_{ij}: coste unitario $i \rightarrow j$

Se modeliza con:

$x_{ij} \lesseqgtr$ *el número de ítems de i a j*

$$\text{mín } z = \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij}$$

$$\sum_{j=1}^{n} x_{ij} \leq a_i \quad \textit{No sobrepasar la capacidad}$$

$$\sum_{i=1}^{m} x_{ij} \geq b_j \quad \textit{Al menos superar demanda}$$

$x_{ij} \geq 0$ *A veces hay que restringir a que sean enteros*

Problema de Asignación

Dado m individuos a los que hay que asignar m tareas con c_{ij} es el coste unitario de coste asignación $i \rightarrow j$

Este ejemplo #tarea = #individuos

Se modeliza:

$$x_{ij} \begin{cases} 0 & si\ i \nrightarrow j \\ b & si\ i \rightarrow j \end{cases}$$

$$\text{mín } z = \sum_{i=1}^{m} \sum_{j=1}^{m} c_{ij} x_{ij}$$

$$\sum_{j=1}^{m} x_{ij} = 1 \quad \forall i = 1, ..., m$$

$$\sum_{i=1}^{m} x_{ij} = 1 \quad \forall i = 1, ..., m$$

$x_{ij} \in \{0, 1\}$

Condicionales

```
if a=1
  then b=1
  else b=0
```

Tecnica de la M-Grande: Consiste en añadir una variable binaria, y además en usar una constante M arbitrariamente grande. Acotar la variable y otra restricción que ajuste todo.

$a \leq b + My$

$a \geq b + 2y$

$b + y = 1$

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

Lo primero es acotar la variable, ≤ y ≥.

Después con una variable binaria y la arbitrariamente grande debemos obligar a la variable a tomar el valor que nosotros queramos.

Partiendo de la condición, miramos en que parte de la misma podemos meter la variable binaria y que obligue a que tome 1 o 0, idealmente ambos. Después con el valor arbitrariamente grande en la misma condición buscamos obligar a tomar el que nos falte por obligar.

2.2. Programación Lineal Entera

Una tarea de Programación Lineal es de Programación Lineal Entera si una o más variables de decisión tienen restricciones de integridad ($x_j \in N_0^+$), es NP-hard (Karp, 1972):

La colección de puntos, son enteros, los puntos rojos.

La Programación Lineal Entera es NP-hard (Karp, 1972)

3 tipos:

- Programación Entera Pura: $x_i \in N_0^+; \forall i$ Todas las variables están afectadas por una restricción de integridad.
    - Pueden no tener solución, si no hay puntos enteros en las subregiones factibles.
- Programación 0-1: $x_i \in \{0, 1\}$
- Programación Entera Mixta: Solo algunas variables de decisión tienen restricción de integridad.

2.2.1. Ramificación y Acotación en profundidad

Relajamos el problema, ignorando las restricciones de integridad, y vamos añadiendo las restricciones.

Ramificamos una de las variables de decisión, creando dos subregiones factibles.

La solución óptima de las subregiones será peor o igual que la óptima relajada, $z_S \leq z_F$.

Dado el problema de Programación Lineal Entera:

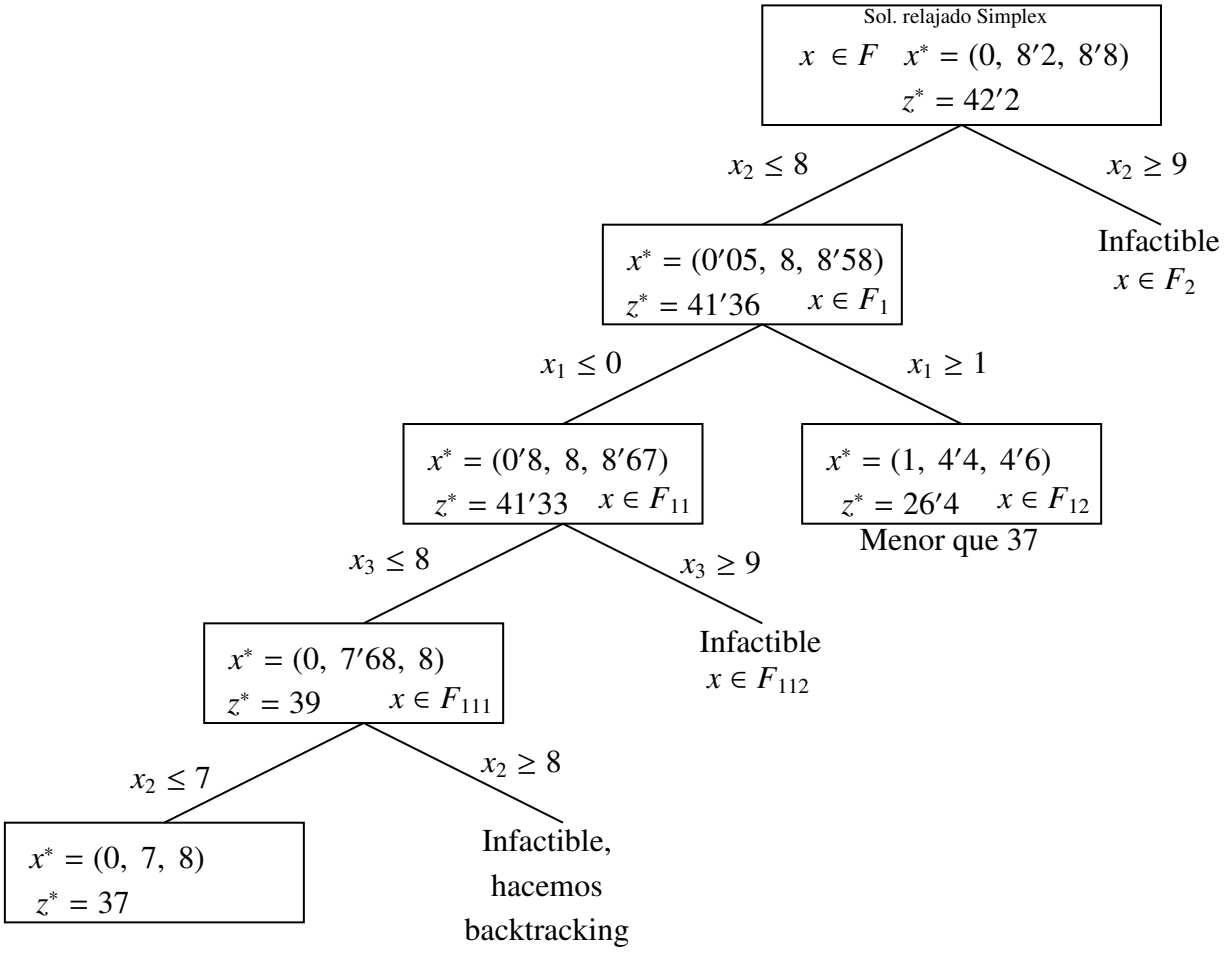
$$\begin{array}{c} \text{máx } z = z(x) \\ s.ax \in F; x_i \in N_0^+ \forall i \end{array}$$

Método:

1. $B_, -\infty$ o Valor negativo muy alto
2. Resolver el Problema Relajado.
   - Si $x* \in N_0^+$ entonces HALT
   - En otro caso, ir al 3.
3. Aplicamos alguna regla de ramificación sobre una variable de decisión no entera: F_1, F_2 (las llamaremos S, de subset)
4. Determinar el valor de z_s
5. Son nodos terminales:
   - $S = \emptyset$ Infactible esa subregión, hacemos backtracking.
   - $z_s \leq B$ Peor que alguno de los terminales, hacemos backtracking
   - $x* \in N_0^+, z_s > B$, entonces, $B_, z_s$
6. Si todos los nodos son terminales, HALT, en otro caso, ir a 2.

Ejemplo:

$$\begin{aligned} \text{máx } z &= 4x_1 + 3x_2 + 2x_3 \\ s.a. & \; 5x_1 - 2x_2 + 3x_3 <= 10 \\ & \; 3x_1 + 3x_2 - 2x_3 <= 7 \\ & -x_1 + 2x_2 - x_3 <= 9 \end{aligned}$$



3. TEMA 2 PROGRAMACIÓN DINÁMICA

Principio de Optimalidad: Una política óptima debe verificar que independientemente del estado inicial y decisiones iniciales, el resto de decisiones deben ser óptimas con respecto al estado que resulta de la primera decisión. (Bellman, 1957)

- Ejemplo que lo cumple: La ecuación de Bellman. $V(x) = max_{a \in A}\{f(x,a)+V(T(x,a))\}$
    - El coste de x, será el máximo evaluando todas las transiciones de: el coste de la acción a sobre más el coste del estado al que transiciona con la acción a sobre x.
- Ejemplo que no lo cumple: Longest Path Problem - LPP.

Los problemas que verifican el principio de optimalidad también verifican la propiedad de Subestructura Óptima (Coman, 2009).

- Si es el camino óptimo entre s y t, no habrá otro con menor coste. Además, para los nodos que se recorren continuar ese camino también será el camino óptimo hasta t.

La programación dinámica sugiere:

1. Descomponer el problema en subproblemas y caracterizar su estructura.
2. Definir una expresión de recurrencia para calcular la solución óptima de los problemas.
3. Derivar la solución óptima de cada problema. Calcular los valores de las soluciones óptimas de los subproblemas.
4. Calcular la solución óptima del problema global.

3.1. Single-Source Shortest-Path

Dado un grafo $G = (V, E)$ y una función de costes $c : eß\mathbb{Z}$, calcular el coste del camino óptimo desde $s \in V$ hasta todos los demás vértices.

3.1.1. Bellman-Ford-Moore (1958, 56, 57)

El camino óptimo entre dos puntos será: mín$(d[e.v], d[e.u] + e.c)$

- e.u = origen
- e.v = destino
- e.c = coste del arco entre u y v
- Lo que quiere decir, que escogemos el menor entre, el coste de ir al destino ya calculado o el coste de ir a otro punto y coger desde este un arco al destino.

```
def bellmanFordMoore (V,E,s):
        for v in V:
                if V==s: d[v]=0
                else: d[v]= +infinito
        for i in range(len(V)-1):
                for e in E:
                        d[e.v]= min(d[e.v], d[e.u]+e.c)
        for e in E:
                if d[e.u]+e.c<d[e.v]
                        raise(...) #Ciclos negativos
```

Complejidad:

- Spurse: $O(|V|^2)$
- Dense: $O(|V|^3)$

3.2. All Pairs Shortest-Path

Dado un grafo $G = (V, E)$ y una función de costes $c : eß\mathbb{Z}$, el coste del camino más corto entre cada par de vértices.

- Aplicando Bellman-Ford-Moore, la complejidad es $O(|V|^3) - O(|V|^4)$

3.2.1. Floyd-Warshall

En cada iteración vamos añadiendo un vértice más que podemos visitar.

- $D_{ij}^{(k} = \text{mín}\{D_{ij}^{(k-1}, D_{ik}^{(k-1} + D_{kj}^{(k-1}\}$ k son los vértices auxiliares que vamos añadiendo.
- $D_{ii}^{(0} = 0$
- $D_{ij}^{(0} = c(e(i, j))$ si hay arco entre i y j, si no es $D_{ij}^{(0} = +\infty$

Algoritmo:

- Partimos de una matriz con los costes a los vértices
- Ahora en cada iteración vamos añadiendo un vértice, k, que podemos usar como intermediario.
  - En cada una de esas iteraciones evaluamos todos los vértices con todos los vértices, ir de i a j.

    Para cada par evaluado nos quedamos con el menor entre, el coste de ir de uno al otro previo, dij, o el coste de ir del origen al vértice que hemos añadido más el cose de ir desde el vértice añadido al destino, dik+dkj.

```
def floydwarshall:
        for v in V:
                d[v][v]=0
        for e in E:
                d[e.u][e.v]=e.c
        for k in V:
                for i in V:
                        for j in V:
                                d[i][j]= min(d[i][j],d[i][k]+d[k][j])
```

- Complejidad: $O(|V|^3)$

4. TEMA 3 SATISFABILIDAD

1. Una fórmula está en Forma Normal Conjuntiva si y solo si: $F \equiv \bigwedge_{i=1}^{n} \zeta_i = \bigwedge_{i=1}^{n} (\bigvee_{j=1}^{|\zeta_i|} \ell_i)$

   - Ejemplo: $(x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee \overline{x_2} \vee x_3)$
     - 3 variables: x_1, x_2, x_3
     - 4 literales: $x_1, \overline{x_1}, \overline{x_2}, x_3$
     - 2 cláusulas: $(x_1 \vee \overline{x_2}), (\overline{x_1} \vee \overline{x_2} \vee x_3)$

2. Las variables x pueden estar afirmadas (x) o negadas ($\overline{x}$) , y se denomina literal a la asociación de una variable y su signo.

3. Un literal ℓ es puro si y solo si $\overline{\ell}$ no aparece en F ($\overline{\ell} \notin F$)

   - Ejemplo: x_1 y $\overline{x_1}$ no son literales puros

4. Un modelo M consiste en una asignación de las constantes proposicionales ($\bot$ $falso, \top verdadero$) a todos o algunas de las variables de F que le satisfagan, $M \models F$

- El problema de SAT (Satisfacibilidad) consiste en encontrar al menos un modelo $M \models F$, o probar que no existe ninguno. (K-SAT es NP-complete $K \geq 3$)

4.1. Método de Resolución

$$Res(F, x) \begin{cases} F & si \quad x \notin F \\ F - x & si \quad x \text{ es un literal puro en } F \\ (c_1 \vee c_2) & si \quad \begin{matrix} (x \vee c_1) \in F \\ (\overline{x} \vee c_1) \in F \end{matrix} \end{cases}$$

Tautología: Toma el valor cierto siempre.

$$F \equiv (x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2)$$

$$Res(F, x_1) = \emptyset SAT$$

Contradicción: Nunca será cierta.

$$F \equiv (x_1) \wedge (\overline{x_1})$$

$$Res(F, x_1) = \emptyset \vee \emptyset = \{\emptyset\} UNSAT$$

Si resulta la cláusula vacía, {∅}, UNSAT

Si resulta la conjunto vacía, $\emptyset$, SAT

El modelo de resolución reduce el número de variables, pero no necesariamente el número de cláusulas será $\frac{n^2}{4}$

El método de resolución no genera modelos, aunque preserva la identidad lógica.

Ejemplo UNSAT

$C_1 : (p \vee \bar{q} \vee \bar{r})$

$C_2 : (\bar{p} \vee s)$

$C_3 : (\bar{s} \vee q)$

$C_4 : \bar{q}$ igual que $(\bar{q})$

$C_5 : (p \vee \bar{r})$

$C_6 : (p \vee q)$

$C_7 : (q \vee s)$

$C_8 : (q \vee q) = (q)$

Paso 0: $G_0 = \{C_i\}_{i=1}^{6}$

$Res(G_0, \bar{r}) = \{C_2, C_3, C_4, C_6\}$

$G_0 \setminus Res(G_0, \bar{r}) = \{C_1, C_5\}$

Paso 1: $G_1 = \{C_2, C_3, C_4, C_6\}$

$Res(G_1, p) = \{C_3, C_4, C_7\}$

$G_1 \setminus Res(G_1, p) = \{C_2, C_6\}$

Paso 2: $G_2 = \{C_3, C_4, C_7\}$

$Res(G_2, s) = \{C_4, C_8\}$

$G_2 \setminus Res(G_2, s) = \{C_3, C_7\}$

Paso 3: $G_3 = \{C_4, C_8\}$

$Res(G_3, q) = \{C9 : \emptyset\} = \{\emptyset\}$ *UNSAT*

Insatisfacible, terminado

4.2. Algoritmos

4.2.1. Davis-Putnam (DP)

1. Selección de un literal $\ell \in F$ (empezando por los puros)
2. Aplicar $Res(F, \ell)$ y anotando la variable usada y las cláusulas involucradas.
    - Si creamos cláusulas que son Tautologías no las ponemos, se satisfacen directamente.
3. Si resulta $\{\emptyset\}$, F es UNSAT. Entonces HALT
    - Cuando hay que juntar cláusulas y la misma variable, pero cada una con un signo y se unen los vacíos.
4. Si resulta $\emptyset$, F es SAT. Ir a 6.
    - Cuando queda una sola cláusula con un único literal, por lo que se elimina en la siguiente.
    - Cuando al juntar literales no puros queda una tautología.
5. En otro caso, ir a 1.
6. Considerar las variables usables y las cláusulas involucradas en orden inverso, y asignar los valores $\bot$ y $\top$ a las variables usadas (y otras si hiciera falta) para satisfacer esas cláusulas.

Del 1-5 es la Fase I, es de progreso.

La 6 es la Fase II, de regresión.

Ejemplo

$C_1 : (p \vee r \vee \bar{s})$

$C_2 : (\bar{p} \vee \bar{s})$

$C_3 : (\bar{q} \vee \bar{r})$

$C_4 : (q \vee s)$

$C_5 : (r \vee \bar{s})$

$C_6 : (q \vee r)$

Paso 0: $G_0 = \{C_i\}_{i=1}^4$

$Res(G_0, p) = \{C_3, C_4, C_5\}$

$G_0 \setminus Res(G_0, p) = \{C_1, C_2\}$

Paso 1: $G_1 = \{C_3, C_4, C_5\}$

$Res(G_1, s) = \{C_3, C_6\}$

$G_1 \setminus Res(G_1, s) = \{C_4, C_5\}$

Paso 2: $G_2 = \{C_3, C_6\}$

$Res(G_2, q) = \{C_7 : (\bar{r} \vee r)\} = \emptyset$ *SAT*

$G_2 \setminus Res(G_2, q) = \{C_3, C_6\}$

Paso	Var	Clausulas	M
2	s	C_4, C_6	$q = \bot, r = \top$
1	s	C_4, C_5	$s = \top$
0	p	C_1, C_2	$p = \bot$

Buscamos en cada paso ir dando valor a las variables para que cumpla las cláusulas, primero a la var y si necesitamos otra a esa, pero no más

Los valores no son únicos

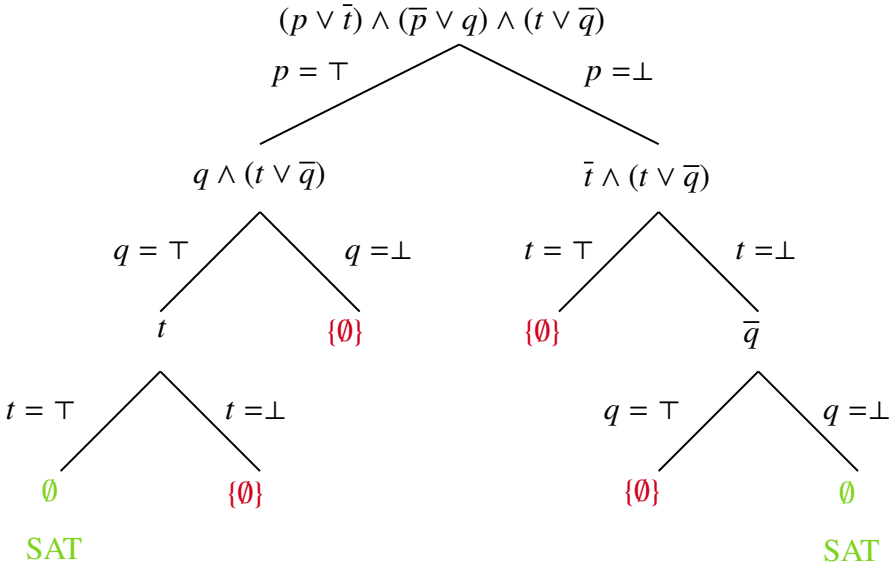
Si hay una variable que su valor no importa hay que indicar que puede ser $\top$ o $\bot$, «Se le da cualquier valor»

4.2.2. Davis-Putnam-Logemann-Loveland (DPLL)

1. Resolución unitaria: Resolución en la que una, al menos de las cláusulas padre es unitaria.
2. Sea F una CNF y $\ell \in F$. Si F es SAT entonces $F \cup \{\ell\}$ es SAT o $F \cup \{\bar{\ell}\}$ es SAT.

- Se denomina reducción de una fórmula F por un modelo parcial v a la fórmula resultante $F_v = Red(F, v)$ en la que se han propagado las asignaciones de v
    - Si el modelo es completo y resulta el conjunto vacío, entonces la formula F es satisfacible y el modelo v lo valida

Ejemplo



Hay 2 modelos:

$M_1 = \{p = \top, q = \top, t = \top\}$

$M_2 = \{p = \bot, q = \bot, t = \bot\}$

4.2.3. CSP - Procedimiento de Satisfacción de Restricciones

1. Una Red De Restricciones R consiste en un conjunto de variables $X = \{X_i\}_{i=1}^{n}$ definidas sobre un dominio $D = \{D_i\}_{j=1}^{h}$ que contienen los posibles valores de cada variable $D_i = \{V_1^{(i}, V_2^{(i}, ..., V_k^{(i}\}$ y un conjunto de restricciones $C = \{C_i\}_{n=1}^{m}$.

   - $R = (X, D, C)$

2. Unas restricciones C_i consiste en una relación (bidireccional típicamente) definida sobre un subconjunto de variables $S \subseteq X$, que denota todas las asignaciones simultáneamente legales.

3. Una instanciación de un subconjunto de variables $S \subseteq X$ consiste en una asignación de valores de los dominios de las variables en S que sea consistente con las restricciones.

   a) $S \subset X$: instanciación parcial.

   b) $S = X$: instanciación total.

   - No siempre es posible extender una instanciación parcial a otra total.
   - Objetivo: Dada una red de restricciones $R(X, D, C)$ encontrar una instanciación total que sea compatible con todas las restricciones en C, si existe alguna, en otro caso, salir con una instanciación vacía.

4. Una red de restricciones $R(C, D, C)$ puede representarse como un grafo donde los vértices representan X, y hay un arco entre los vértices x_i y x_j si $R_{ij} \neq \emptyset$

4.2.4. Arco-Consistencia

Una variable x_i es arco-consistente con otra variable x_j si y solo si para cada $a_i \in D_i$, existe otro valor $a_j \in D_j$,, $(a_i, a_j) \in R_{ij}$

AC-REVISE(x_i, x_j):

For $\sigma_i \in D_i$:

If $\nexists a_j \in D_j$, , $(\sigma_i, \sigma_j) \in R_{ij}$:

$D_i = D_i \setminus \{\sigma_i\}$

Arco-consistencia es Direccional.

AC(R):

REPEAT:

For (x_i, x_j),, $R_{ij} \neq \emptyset$

AC-REVISE(x_i, x_j)

AC-REVISE(x_j, x_i)

UNTIL no domain is changed

La arco-consistencia NO sirve para verificar la consistencia global. Ya que los arcos lo verifican para ese par solamente.

4.2.5. Camino-Consistencia

La variable x_i es camino-consistente con x_j con respecto de x_k sí y solo si para cada $(a_i, a_j) \in R_{ij}$ y $a_k \in D_k$,, $(a_i, a_k) \in R_{ik}$ y $(a_j, a_k) \in R_{jk}$.

- La camino-consistencia NO es direccional.
- La camino-consistencia tampoco sirve para verificar la consistencia global. (por ejemplo, si aumenta a 4)

4.2.6. Como lo calculamos realmente.

Se hace mediante un árbol de búsqueda, como todos los métodos que hemos visto hasta ahora.

Se escoge una variable y se crean tantas ramas como valores tenga el dominio de la misma, pero cuando vamos avanzamos por una rama calculamos la arco-consistencia con todas las variables anteriores y así reducimos el número de ramas que hay que considerar. Además, hay que calcular la camino consistencia de todas las variables que llevamos y la que estamos contemplando en conjunto.

Si llegamos a vacío en algún nodo, tenemos un fallo, y hacemos backtracking, si aun así todos salen vacíos no tenemos solución o por el contrario si encontramos una no vacía tenemos una sustanciación global, que cumpla todas las restricciones.

5. TEMA 4 BÚSQUEDA

5.1. Espacio De Búsqueda

1. Estados: (estructuras de datos) Contienen la información de tipo estático.
2. Operadores: (funciones) Dado un estado nos devuelve los que son inmediatamente accesibles.
    - if <precond>then <effects>
3. Un estado inicial, s (start o source).
4. Uno o más estados finales, t (target).
    - En satisfacibilidad conocemos las propiedades del estado final, pero no lo conocemos explícitamente. Lo que buscamos es saber cuál es.
    - En optimización se da explícitamente la meta. Lo que buscamos es saber cómo llegar a él.
    - Los grafos de búsqueda se recorren con Árboles De Búsqueda.
    - Si sabemos mucho de algoritmos de búsqueda, podríamos resolver cualquier tarea de optimización y decibilidad, incluso si es exponencialmente difícil.
5. Factor de ramificación (b): número medio de sucesores de cada nodo.
6. Profundidad: número mínimo de niveles hasta alguna solución.

5.2. Algoritmos de Fuerza Bruta (Búsqueda no informada)

5.2.1. Costes unitarios

1. Algoritmo general: (Paso 1. y 3. son operaciones atómicas, pero el 2. no)
    a) Generar el estado inicial (s).
    b) Expandir el primer nodo de la lista abierta.
    c) Si alguno de los sucesores es un nodo final, entonces halt
        - En otro caso, ir a 2.
2. Generar: Es el proceso de creación de un estado en memoria.

   Expandir: Es el proceso de generar todos los sucesores de un nodo.

3. Completitud: Si un algoritmo garantiza que encontrará una solución.

   Admisibilidad: Si un algoritmo garantiza que encontrará una solución óptima. No hay una solución más óptima que otra, depende de lo que se evalúe.

4. En general, se asume que no hay preferencia en la aplicación de operaciones (costes unitarios, si no son iguales son costes arbitrarios).

5.2.2. Primero en Amplitud

«Nunca expande un nodo a profundidad d si no ha expandido todos los nodos a profundidad (d-1)»

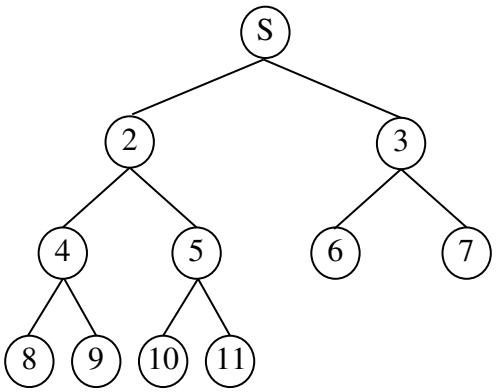


Fig. 5.1: Primero en Amplitud

Al ser un problema de satisfacibilidad solo nos interesa saber cuál es el nodo final, no como llegamos a el. En optimización devuelve el camino.

La lista abierta se implementa con una COLA.

Es COMPLETO y ADMISIBLE.

Tiempo: Depende del número de expansiones $O(b^d)$

Memoria: $O(b^d)$

El hecho que de que ambos sean exponenciales es bastante desafortunado.

5.2.3. Primero en Profundidad

«Se expande el primero de los nodos recién generados hasta que se encuentra una solución o se ha alcanzado un d_{max}»

Se usa una profundidad máxima para evitar caer en una rama infinita que nos aleja de la solución.

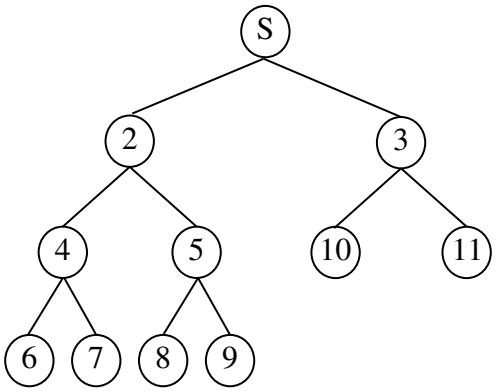


Fig. 5.2: Primero en Profundidad

Implementa Backtracking con una PILA.

Completo: No, puede entrar en un bucle.

Admisible: No, si no es completo no cumplirá el encontrar la óptima.

Tiempo: $O(b^d)$

Memoria: $O(d)$ Si solo se almacena 1 sucesor, que lo expandirás cada vez y si se hace backtracking ya se generará. Si se almacenan todos $O(bd)$

Se usa sobre el de amplitud, nos comprometemos hacia una dirección y esperamos encontrar la solución.

5.2.4. Profundidad Iterativa - Iterative Deepening

«Consiste en una serie de exploraciones en profundidad donde d_{max} se incrementa en k en cada iteración»

En cada iteración anterior a la solución ha expandido todos los nodos.

Has visto todos los nodos previos a la solución, por lo que habrá solución en la otra parte del árbol que todavía no se ha recorrido en una iteración.

Completo: Sí. Es como amplitud ve todo.

Admisible: Si y solo si $d_{max} = k = 1$ Para comprobarlo profundidad por profundidad, y no soltarse un posible nivel en el otro lado.

Memoria: $O(d)$ Va en profundidad.

Tiempo: $\frac{Tiempo(ID)}{Tiempo(BFS)} = \frac{b}{b-1}$

Es como hacer primero en amplitud, pero con memoria lineal, que compensa que tarde más (aunque no mucho más).

- $b^d >> \sum_{i=0}^{d-1} b^i$ (» Mucho más grande)

Expandir TODOS los nodos de TODOS los niveles precedentes no es nada comparado con lo que se tarda en expandir todos los nodos de una nueva profundidad d.

5.3. Costes arbitrarios

5.3.1. Ramificación y Acotación en Profundidad

Definimos el coste de un camino $\pi < s, n >$ como la suma de los costes de los arcos en π.

- $g(\pi) = \sum_{i=0}^{k-1} c(n_i, n_{i+1})$ donde $c(n_i, n_{i+1})$ es el coste del arco $< n_i, n_{i+1} >$ (modelo aditivo nosotros lo consideramos siempre, pero en la realidad no tiene por qué). Con frecuencia $g(\pi)$ se representa como $g(n)$.

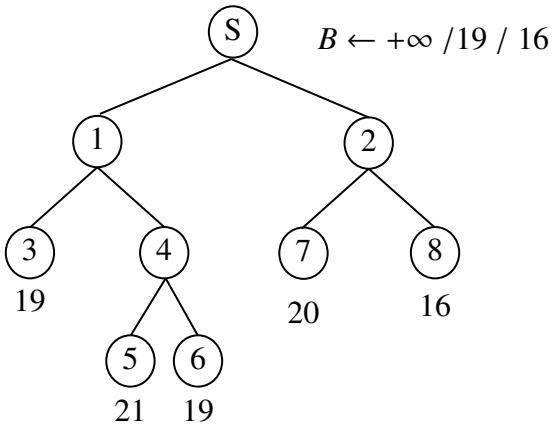


Fig. 5.3: Ramificación y Acotación

Pseudocódigo (B, valor arbitrariamente grande)

1. Generar el estado inicial, s.
2. Coger el primer nodo de Abierta (pila), n.
3. Si n=t entonces B ←$g(n)$. Salir (hacer backtracking)
4. Expandir $n : n_1, n_2, ..., n_k$
5. Si $g(n_i) < B$, insertar n_i en abierta, $\forall_i = 1, ..., k$

6. Volver a 2.

- Costes de $g(\pi)$: $g(\pi)$ es monótono creciente. Los costes de los arcos son positivos $c(n_i, n_{i+1}) \geq 0$.

5.4. Heurísticas

1. Si no tenemos conocimiento → Búsqueda no informada.

   Si tenemos información perfecta → no hay búsqueda.

2. $h(n, t)$: Devolver una estimación del coste del mejor camino para llegar desde n a t.

3. Sí $h(n, t) \leq h^*(n, t)$ entonces h es ADMISIBLE.

   El nodo terminal debe tener heurística 0.

5.4.1. Relajación de restricciones (Judea Pearl, 1983)

Las restricciones las encontramos en la definición de las precondiciones de las operaciones.

Se observan las precondiciones y vemos cuál podemos relajar, una vez relajada hallamos algunas soluciones con esta condición y nos damos cuenta de cuál es la función heurística en este caso.

Si $h_1(n) \geq h_2(n) \forall n$ y ambas son admisibles, entonces $h_1(n)$ está más informada que $h_2(n)$.

Preferimos las más informadas.

Si relajamos todas las restricciones se llama Heurística no informada.

Típicas heurísticas al relajar restricciones:

- Casillas mal dispuestas: Número de casillas que no ocupan la posición del estado final.
- Distancia de Manhattan: La suma del valor absoluto de las diferencias de las coordenadas.
- Distancia Euclídea: Hallar la hipotenusa que crea el estado actual, el final y un punto a la misma de ambos.

5.5. Algoritmos de Búsqueda Heurística

Se aplica tanto a costes unitarios como a costes arbitrarios, pero necesitamos emplear 1 heurística.

5.5.1. Hill-climbing (De escalada)

«Se escoge para su expansión el sucesor heurísticamente más prometedor descartando el resto de sucesores»

- Desestima nodos por tener menor coste heurístico.
- No hay backtracking, al no almacenarse.
- Aun así, se usa frecuentemente.
- Completo: no
- Admisible: no.
- Memoria: $O(d)$

5.5.2. Algoritmo de búsqueda en Haz (Beam search)

«Se expanden simultáneamente los k sucesores más prometedores heurísticamente»

k: ventana o amplitud del haz

Desechamos el resto que no son k. Si coinciden se elige arbitrariamente.

Completo: No, juzga por la heurística.

Admisibilidad: No, al no ser completo.

$BS(k = 1)$: Es hill-climbing.

$BS(k = \infty)$: Primero en amplitud.

El k permite decidir la memoria que consume.

Falta de monotonía de la búsqueda en haz, se obtiene una mejor solución con k+1 que k.

Tiempo: Exponencial.

Memoria: O(k) lineal, solo almacena los k más prometedores.

5.5.3. Algoritmo de «El mejor primero» (Raphael, Hart, Nilsson, 1968)

1. Considerar:

   a) Lista abierta: Contiene todos los nodos generados pendientes de ser expandidos, ordena=dos ascendentes por f(n)

   b) Lista cerrada: Contiene todos los nodos que ya han sido expandidos (duplicate detection, evita expandir un nodo expandido previamente)

   c) Terminación: Procedemos al expandir t, no al generarlo.

2. Miembros:

   - $f(n) = h(n)$ Algoritmo de búsqueda heurística pura.
   - $f(n) = g(n)$ Dijkstra (Realmente es fuerza bruta)
   - $f(n) = g(n) + h(n)$ A*

3. Si $h(n) \leq h^*(n)$ entonces A* es admisible.

4. Tiempo(A*): Exponencial.

   Memoria: Exponencial.

   Especialmente útil por la detección de nodos duplicados.

5.5.4. Iterative-deepening A* - IDA* (Korf, 1985)

«Consiste en una serie de recorridos del **primero en profundidad hasta que** $f(n) > \eta$ **o hemos encontrado la solución**, incrementando f(n) en cada iteración al menor exceso cometido»