Síntesis arquitectónica y de alto nivel

Módulo 3: Asignación de Hardware

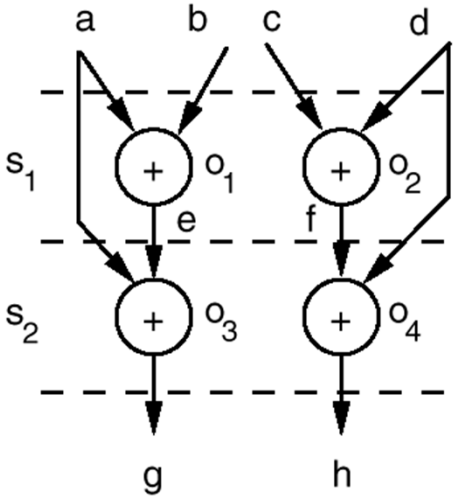
Módulo 3. Asignación de Hardware

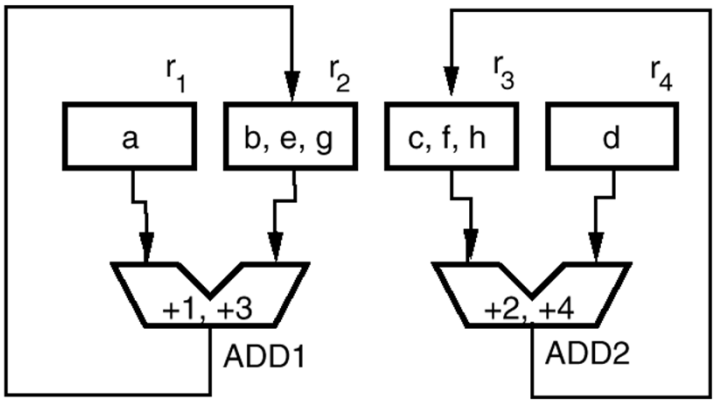
- Contenido
  - 1. Introducción y definición del problema
  - 2. Tareas de la asignación de hardware
  - 3. Enfoques al problema
  - 4. Descomposición basada en grafos
    - 4.1. Coloreado de grafos
    - 4.2. Particionamiento en cliques
    - 4.3. Aplicaciones a la síntesis
  - 5. Construcción avariciosa
  - 6. Construcción por “bifurcación y acotación”
    - 6.1. Método de asignación de FIDIAS: heurísticas

Introducción

- Definición del problema: Creación de una ruta de datos tal que exista:
  - una UF para cada operación
  - un elemento de almacenamiento para cada dato
  - un camino para cada transferencia
- Objetivo: cumplir los anteriores requisitos con el mínimo coste posible → compartir recursos
- Alternativas:
  - Después del scheduling (la más común)
  - Antes del scheduling

- Ejemplo: DFG planificado y ruta de datos



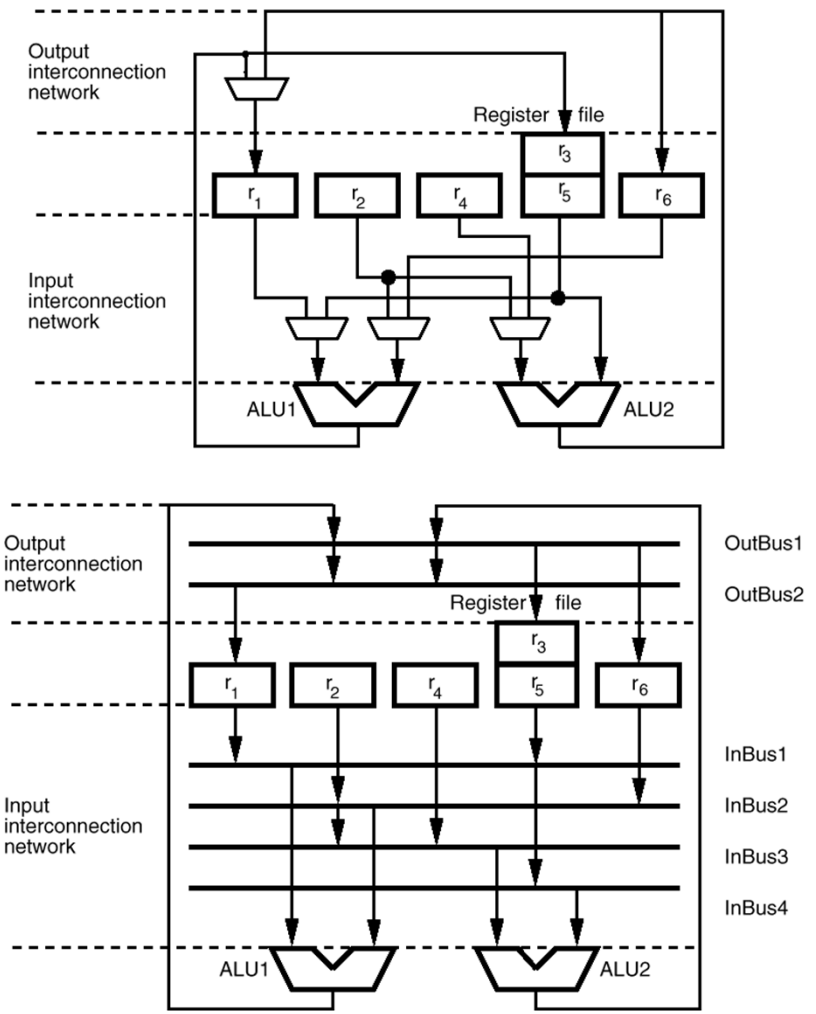


Introducción

- Arquitecturas de la ruta de datos
  - Punto a punto: mediante multiplexores
  - Basada en buses
  - Mixta: combina buses y multiplexores
- Ejemplo: Secuencia de operaciones en 3 pasos de control

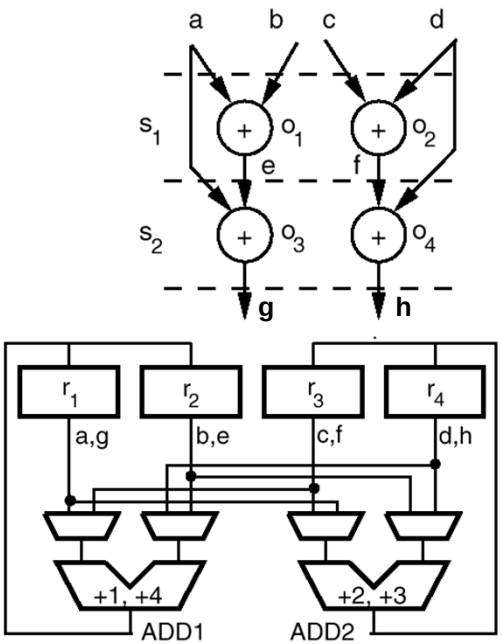
s_1: $r_3 \leftarrow$ ALU1 (r_1, r_2);
$r_1 \leftarrow$ ALU2 (r_4, r_3);
s_2: $r_1 \leftarrow$ ALU1 (r_5, r_6);
$r_6 \leftarrow$ ALU2 (r_2, r_5)
s_3: $r_3 \leftarrow$ ALU1 (r_1, r_6);

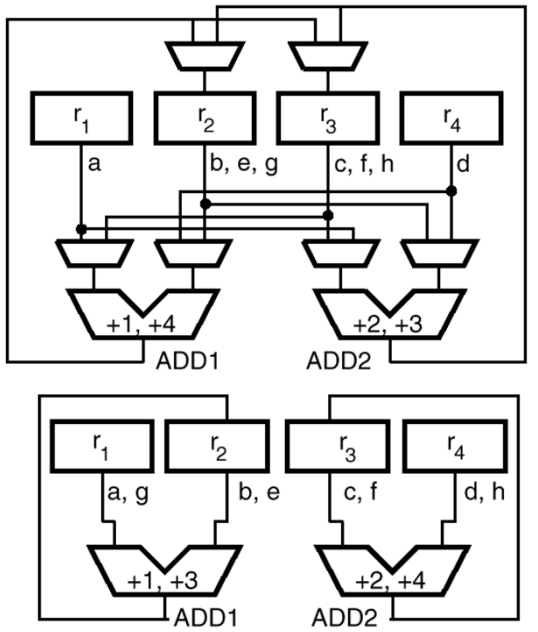
- Alternativas de implementación



Tareas de la asignación

- Selección de FUs
  - Pueden existir varias FUs diferentes capaces de ejecutar una cierta op.
    - Ejemplo: + puede ser implementado con distintos tipos de sumadores (CLA, RC)
  - Pueden existir FUs capaces de implementar varias operaciones
    - Ejemplo: ALU, Sumador/restador
- Vinculación de FUs con operaciones
- Asignación de almacenamiento
  - Compartir registros para almacenar datos cuyas intervalos de vida no se solapan
- Asignación de interconexiones
  - ***Cuidado!*** El hardware de las interconexiones también cuesta
- Interdependencias: Ejemplo





Enfoques para la asignación

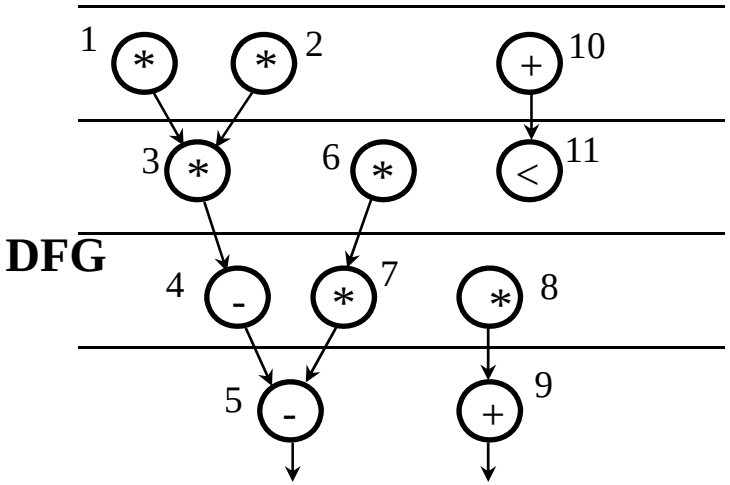
- Por descomposición:
  - Realizan las distintas tareas por separado
  - Comenzar por las más decisivas
  - Métodos basados en grafos
    - Soluciones exactas en casos particulares
    - Caso general: heurísticas

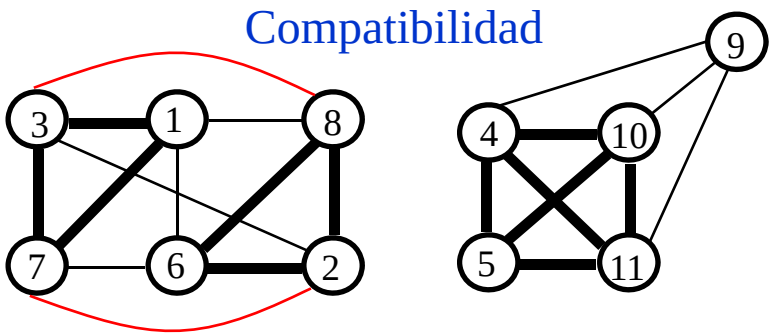
- Por construcción:
  - Procesar el DFG. Ir añadiendo nuevos elementos a la ruta de datos a medida que se procesa cada entidad del DFG.
  - Suelen abordar globalmente todos los subproblemas. (Al menos entremezclan las subtareas).
  - Métodos:
    - Avaricioso (greedy)
    - Branch-and-Bound (B&B): Heurística de búsqueda de alternativas de implementación y acotación.

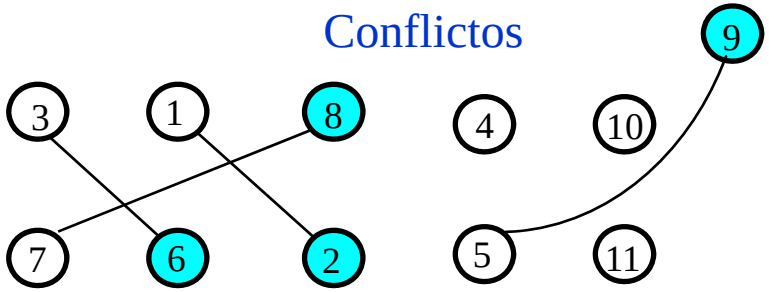
- Problema adicional: métricas de calidad del diseño que se está generando $\Rightarrow$ Estimar el coste.
  - Coste de UFs y registros: Puede derivarse aproximadamente de las bibliotecas de diseño
  - Coste de multiplexores: Idem
  - Coste de interconexiones: ???

Descomposición basada en grafos

- Definiciones:
  - *Grafo completo*: cada par de nodos están unidos por un arco
  - *Clique*: Dado un grafo G(V,E), un clique de G es un subgrafo de G que es completo.
- Modelado de problemas de compartición de recursos mediante grafos: traducir a problemas de ***particionamiento en cliques*** o de ***coloreado de nodos***
- Ejemplo: DFG, Gr. de compatibilidad y gr. de conflictos: (Tipos de FUs: * y UAL) para el DFG:







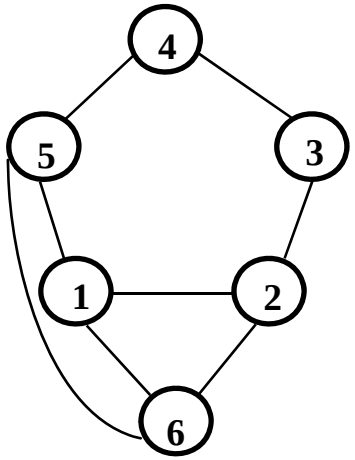
Coloreado de grafos

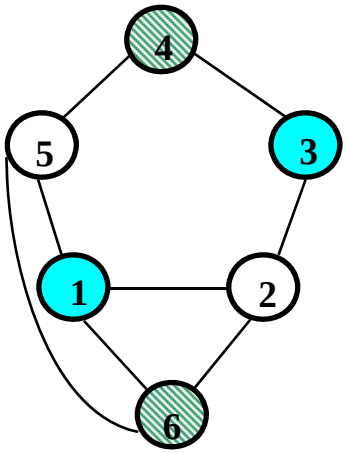
- Concepto: Asignar a cada nodo un color, de tal modo que si dos nodos son adyacentes, entonces deben tener distintos colores.
- Algoritmo heurístico

```
VERTEX_COLOR (G(V,E)) {
    for (i = 1 to |V|) {
        c = 1;
        while (∃ a vertex adjacent to v_i
            with color c) do {
            c = c+1;
        }
    }
}
```

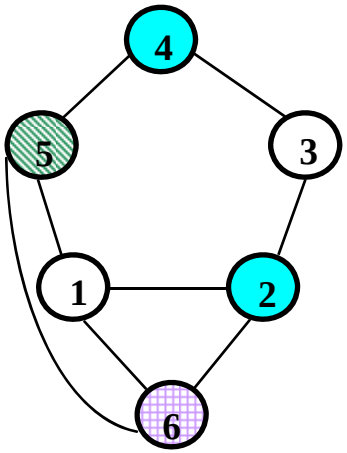
  - Problema: el nº de colores usado puede no ser mínimo

- Ejemplo:





Coloreado mínimo



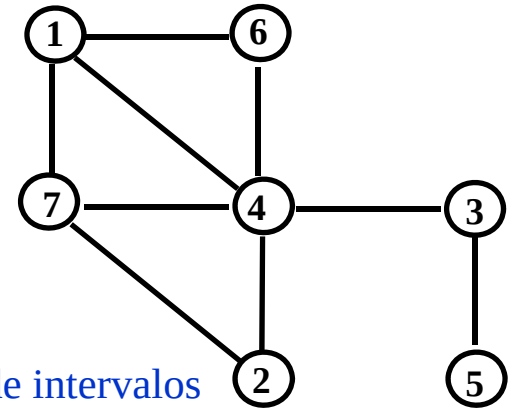
Resultado del algoritmo

Coloreado de grafos (2)

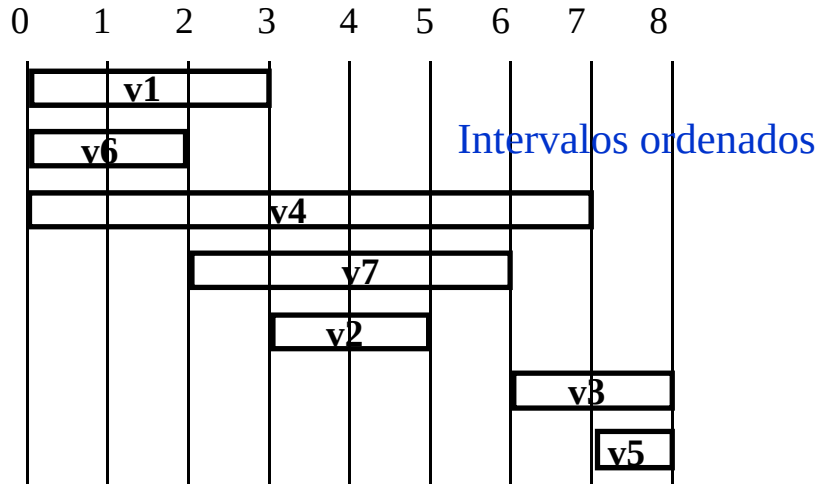
- Los grafos de intervalos (grafos generados por intervalos)
  - Modelan bien algunos problemas de asignación
  - Se pueden colorear en tiempo polinómico
- Algoritmo del lado izquierdo
  - Origen: trazado de pistas en CI

Ejemplo: conjunto de intervalos

v1 = [0,3] v2 = [3,5] v3 = [6,8] v4 = [0,7]
v5 = [7,8] v6 = [0,2] v7 = [2,6]

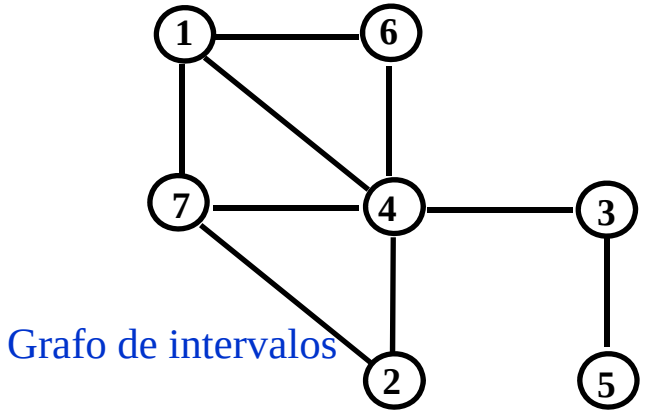


Grafo de intervalos

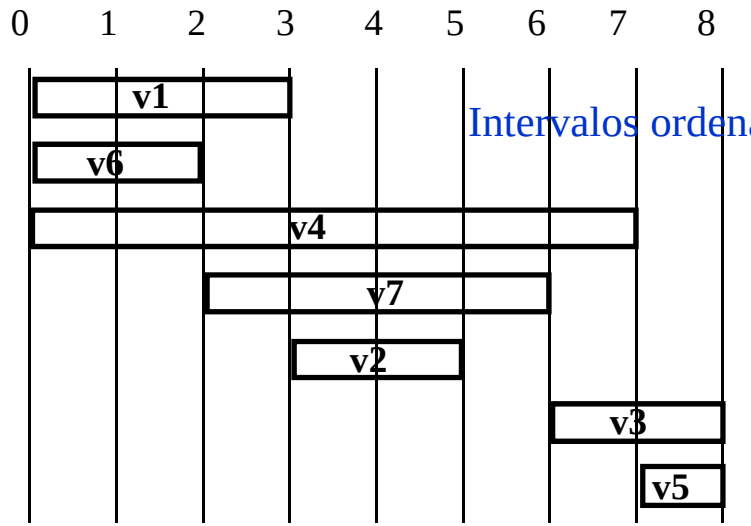


Intervalos ordenados

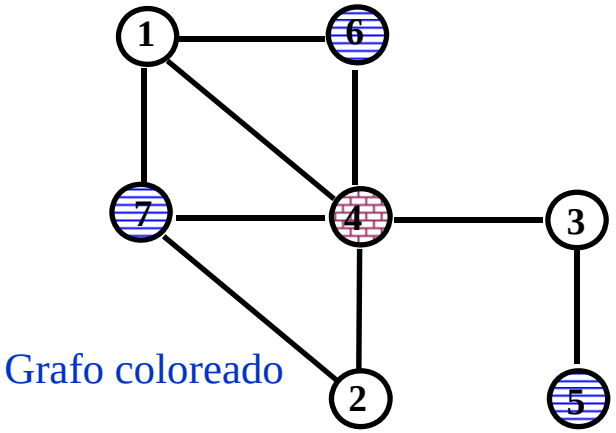
Coloreado de grafos (3)



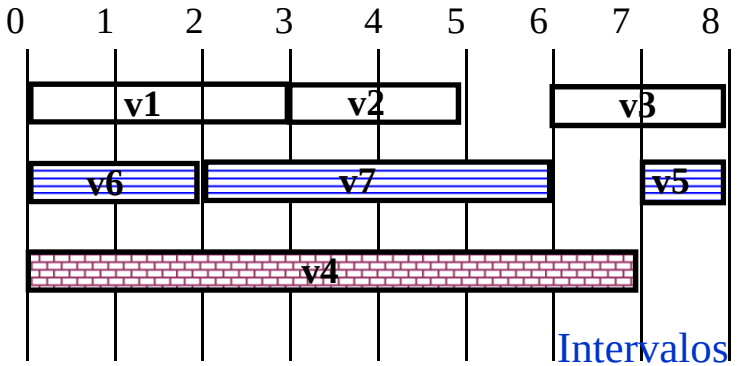
Grafo de intervalos



Intervalos ordenados



Grafo coloreado



Intervalos empaquetados

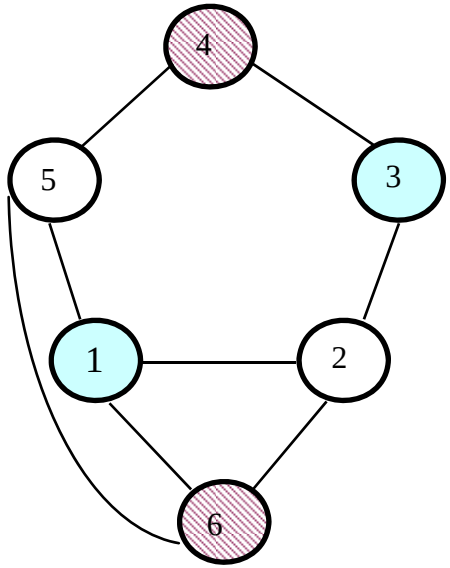
Coloreado de grafos (4)

- Algoritmo del lado izquierdo (código)

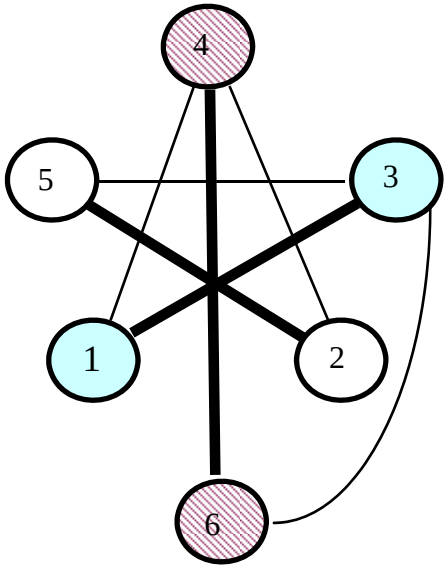
Conjunto de intervalos $I = \{ [l_j , r_j] ; j = 1, 2, \ldots, |I| \}$
LEFT_EDGE(I) {
    Sort elements of I in a list L in ascending order of l_i ;
    $c = 0$;
    while (some interval has not been colored) **do** {
        $S = \varnothing$;
        $r = 0$; /* initialize coord. of rightmost edge in S */
    while ($\exists$ an element in L whose left edge coord. is larger than r) **do** {
        s = first element in list L with $l_s > r$;
    $S = S \cup \{s\}$;
    $r = r_s$; /* update coord. of rightmost edge in S */
    Delete s from L ;
    }
        $c = c+1$;
        Label elements of S with color c ;
    }
}

Particionamiento en cliques

- Concepto: crear una partición del grafo de tal forma que cada partición es un clique.
- Problema equivalente: colorear el grafo complementario
- Ejemplo: coloreado de un grafo vs. particionamiento de su complementario



Coloreado
(Grafo de conflictos)



Particionamiento en cliques
(Grafo de compatibilidad)

Particionamiento en cliques (2)

- Algoritmo heurístico

```
CLIQUE_PARTITION (G(V,E)) {
   Π = ∅ ;
   while (G(V,E) not empty) do {
      C = MAX_CLIQUE(G(V,E));
      Π = Π ∪ C
      Delete C from G(V,E)
   }
}
```

```
MAX_CLIQUE (G(V,E)) {
   C = vertex with largest degree;
   repeat {
      repeat {
         U = { v∈V : v ∉C and adjacent to
         all vertices of C};
         if (U = ∅)
            return (C);
         else {
            Select vertex v∈U;
            C = C ∪ {v}
         }
      }
   }
}
```

Particionamiento en cliques (3)

- Algoritmo heurístico (Tseng & Siewiorek)
  - Inicio: Aglutinar el par de nodos vecinos que tienen el mayor nº de vecinos comunes ⇒ Formar supernodo. Eliminar arcos y nodos originales. Añadir arcos entre supernodo y vecinos comunes.
  - Progreso: repite paso anterior, pero considerando cada supernodo como un nodo más.
  - Algoritmo (iniciación):

/* create a supergraph $G'(S,E')$ */
$S = \varnothing$; $E' = \varnothing$;
for each $v_i \in V$ do $s_i = \{v_i\}$; $S = S \cup \{s_i\}$; **endfor**
for each $e_{i,j} \in E$ do $E' = E' \cup \{e'_{i,j}\}$; **endfor**

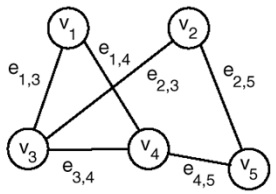
Particionamiento en cliques (4)

- Algoritmo (continuación)

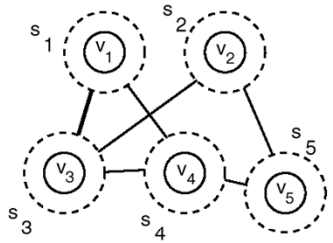
while E'$\neq \varnothing$ **do**
  /* find s_{index1}, s_{index2} having most common neighbors */
  MostCommons = -1;
  for each $e'_{i,j} \in E'$ **do**
    $c_{i,j}$ = |COMMON_NEIGHBOR(G', s_i , s_j)|;
    if $c_{i,j}$ > *MostCommons* **then**
      MostCommons = $c_{i,j}$;
      Index1 = *i*; *Index2* = *j*;
    endif
  endfor
  CommonSet = COMMON_NEIGHBOR (G',s_{index1} , s_{index2});
  /* delete all edges linking s_{index1} or s_{index2} */
  E' = DELETE_EDGE(E', s_{index1});
  E' = DELETE_EDGE(E', s_{index2});
  /* merge s_{index1} and s_{index2} into $s_{index1Index2}$ */
  $s_{index1Index2} = s_{index1} \cup s_{index2}$;
  $S = S - s_{index1} - s_{index2}$;
  $S = S \cup \{s_{index1Index2}\}$;
  /* add edge from $s_{index1Index2}$ to supernodes in *CommonSet* */
  for each $s_i \in$ *CommonSet* **do**
    $E' = E' \cup \{e'_{i,Index1Index2}\}$;
  endfor
endwhile

Particionamiento en cliques (5)

- Ejemplo: Algoritmo de Tseng & Siewiorek

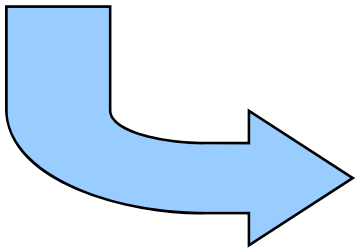


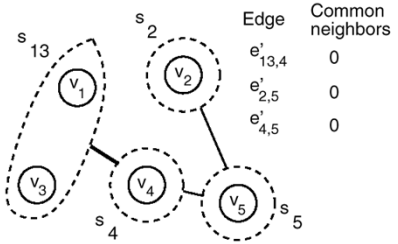
Graph G



Edge	Common neighbors
$e'_{1,3}$	1
$e'_{1,4}$	1
$e'_{2,3}$	0
$e'_{2,5}$	0
$e'_{3,4}$	1
$e'_{4,5}$	0

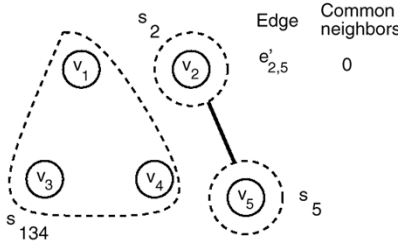
Common Neighbors





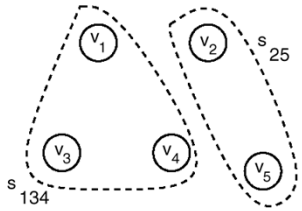
Edge	Common neighbors
$e'_{13,4}$	0
$e'_{2,5}$	0
$e'_{4,5}$	0

Supernode Creation 1



Edge	Common neighbors
$e'_{2,5}$	0

Supernode Creation 2



Supernode Creation 3

$s_{134} = \{v_1, v_3, v_4\}$

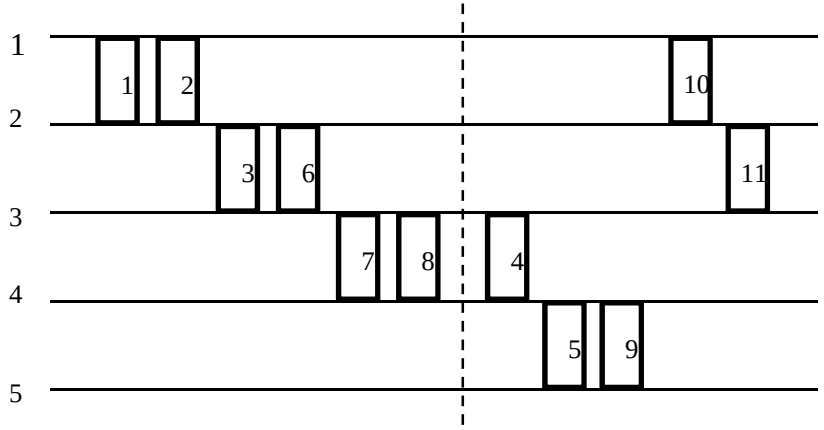
$s_{25} = \{v_2, v_5\}$

Cliques for Graph G

Aplicaciones

- Compartición de FUs
  - Crear gr. de compatibilidad. Si
    - Dos operaciones se ejecutan en instantes distintos,
    - son de mismo tipo entonces son compatibles
  - Partir el gr. de compatibilidad en cliques
    - Todas las operaciones de un clique se asignan a la misma FU
    - Se debe tratar de cubrir el grafo con el menor nº de cliques posible
  - Si se crea un grafo de intervalos, se puede aplicar el algoritmo del lado izquierdo

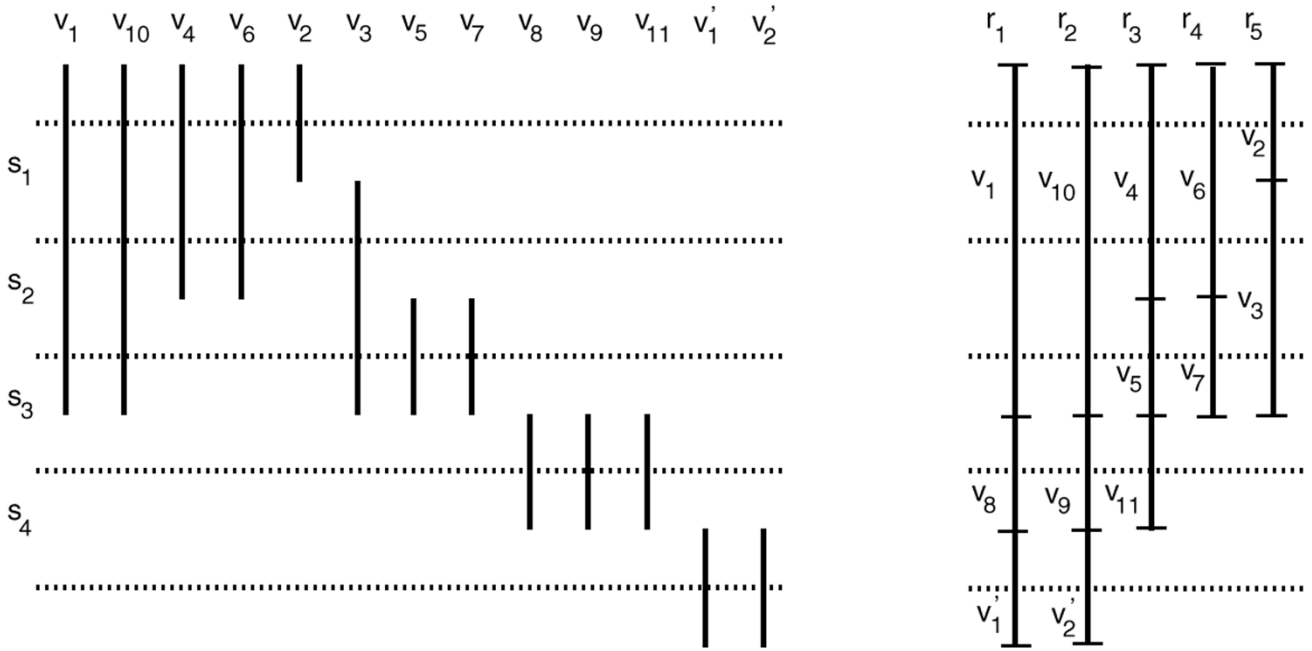
- Ejemplo: Grafo de intervalos derivado de DFG:



Multiplicador 1: Nodos 1,3,7
Multiplicador 2: Nodos 2,6,8
ALU1: Nodos 10,11,4,5
ALU2: Nodo 9

Aplicaciones (2)

- Compartición de registros
  - Crear grafo de conflictos: es claramente un grafo de intervalos
  - Aplicar el algoritmo del LI
  - Ejemplo:

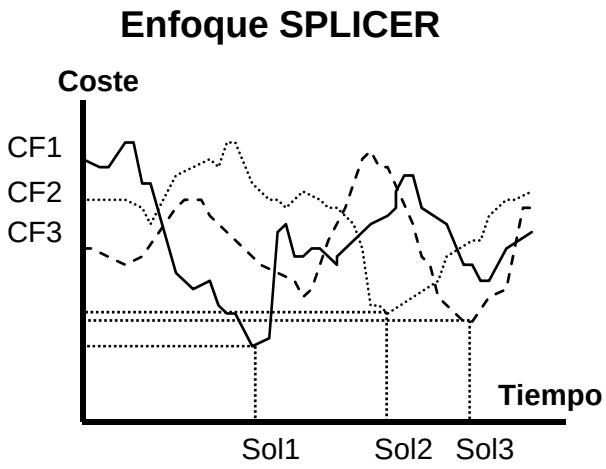


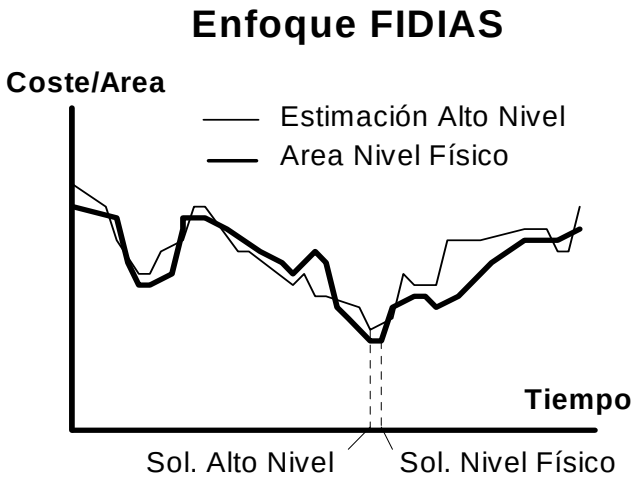
Construcción avariciosa

- Concepto: Comenzando con DP vacío, producir en cada instante el menor incremento de coste posible.
- Base racional
- Mecanismo
  - Examinar todos elementos del DFG que están sin asignar
  - Tomar el elemento que produce el menor incremento de coste de la ruta de datos
  - Asignar HW para dicho elemento. (Si existe HW válido y no usado: reusar; caso contrario: crear nuevo HW).
  - Iterar
- Requisito esencial: definir una función de coste que permita ejecutar el punto anterior.
- Problemas
  - Un gasto elevado al principio puede producir un ahorro al final del proceso.
  - No backtracking

Construcción por B&B

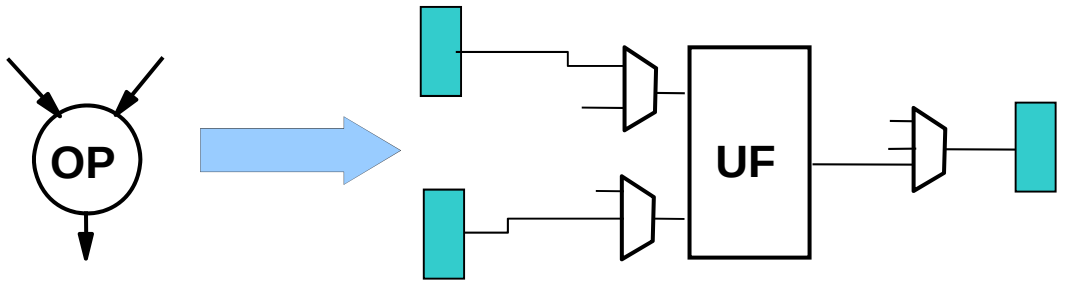
- Concepto: Explorar distintas alternativas de implementación en un *orden conveniente*, con posibilidad de deshacer decisiones ya tomadas.
- Aspectos clave:
  - ***Orden conveniente***. Las primeras soluciones exploradas deben ser aquellas que se espera produzcan buenos diseños. (Heurísticas de guía)
    - Ventajas:
      - Ahorro de tiempo
      - Menor probabilidad de hallar mínimos locales
    - Inconvenientes: hallar el mejor orden no es sencillo
  - Descartar pronto soluciones que conducirán a malos diseños. (Heurísticas de acotación)
- Es preciso un análisis global del DFG
- Dos relaciones de orden
  - Ordenación temporal: exploración de soluciones. Importancia: tiempo en hallar diseños de menor área.
  - Ordenación por costes. Importancia: Validez física de las soluciones halladas.





Método de asignación FIDIAS

- Algoritmo: Procedimiento recursivo. Una llamada => Dos acciones
- Acción de asignación de HW: A base de asignar elementos HW en torno a los arcos del GFD planificado.
- Búsqueda del elemento fuente: habrá sido asignado en una acción anterior
- Asignación de elemento destino.
  - ***Es donde se crea HW.***
  - ***Repercute tiempo de diseño***
  - Ensayar alternativas:
    - Elemento existente y conectado.
    - Elemento existente y no conectado
    - Nuevo elemento (acceso a biblioteca).
  - Anotar conexión en estructuras de información => generación microprograma abstracto.
- Acción de recorrido
  - Preparación de la siguiente llamada
  - Un paso de control tras otro.



Técnicas de acotación

- Objetivo: Detección precoz de diseños inferiores: ahorro tiempo de diseño.
  Cs = coste de la última solución hallada.
- Funciones de acotación: definen un coste de chequeo, Cc.
  Abandonar diseño si Cc < Coste parcial del diseño, Cp.
- Acotación global (por defecto): Cc = Cs
  - Prácticamente no acota: lentitud.
  - Patrón de comparación para ejemplos sencillos
- Acotación distribuida: Vector de costes de chequeo, Cc[i], a nivel de pasos de control.
  - Distribución de costes por etapas de control: Cc[i] = Cs[i]
  - Rápida
  - Problema: mínimos locales (pozos de coste)
- Acotación con margen
  - Se añade un cierto margen de tolerancia: escape de mínimos locales
    Cc[i] = Cs[i] + M[i]
  - Margen proporcional a la complejidad de la etapa:
    M[i] = (Cs[i] - Cs[i-1]) x P
  - Gradiente que remonta: depende de P

Técnicas de acotación (2)

- Efecto de las técnicas de acotación

Ejemplo FACET

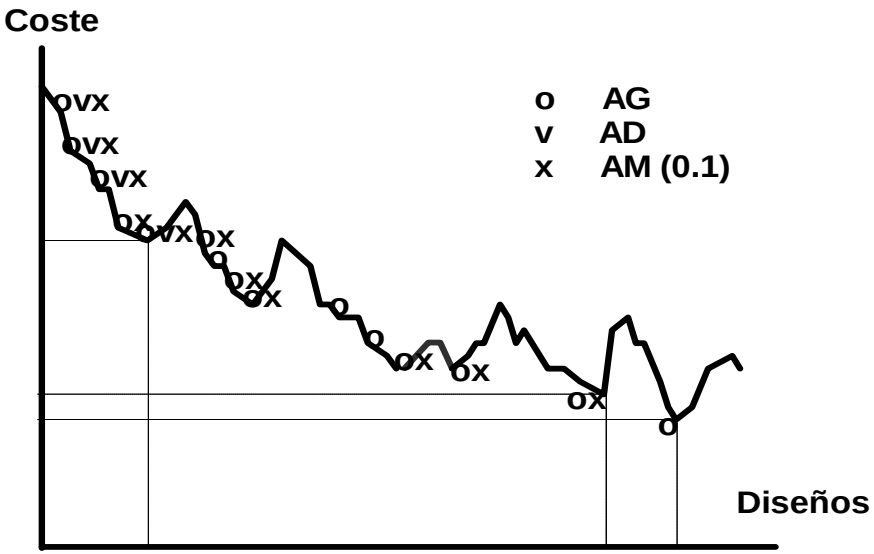


Tabla comparativa

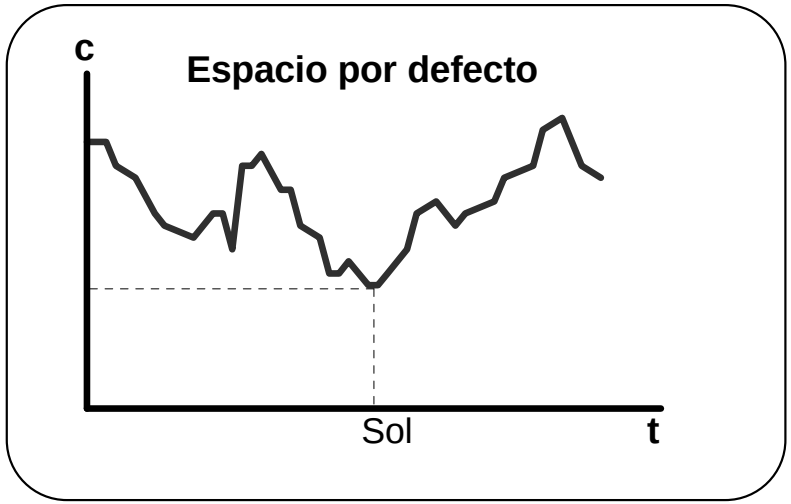
	Coste	Heurística	CPU tiem.	# Diseños	Exit
FACET	45530	AD	1	4	opt.
	36381	AM (.1)	9	11	opt.
	35898	AM (.2)	9	13	opt.
	35898	AG	40	15	opt.
ECUACION DIFEREN.	44875	AD	1	1	opt.
	43708	AM (.1)	1	3	opt.
	43916	AG	3900	7	t.o.
FILTRO	53544	AD	2	4	opt.
	49310	AM (.1)	98	12	opt.
	53216	AG	11500	8	t.o.

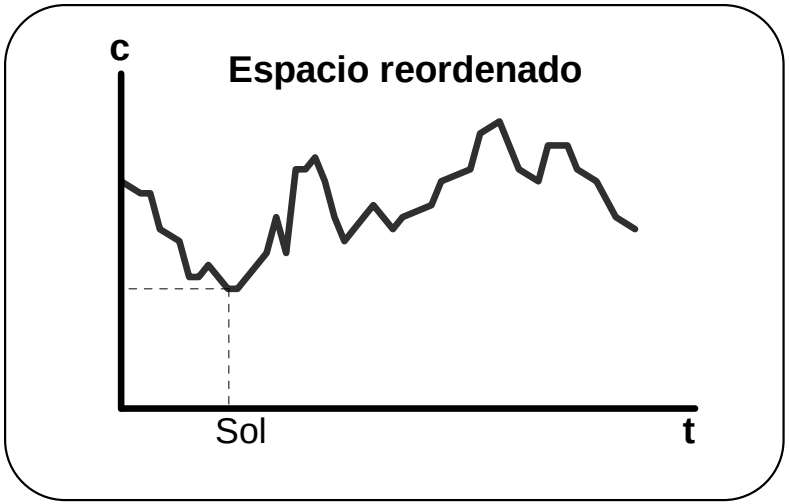
opt. = Óptimo en el espacio de diseño acotado

t.o. = Última solución hallada al detener la búsqueda por time-out

Estrategias de guía

- Objetivo: ensayar primero las alternativas de diseño que mejor se adaptan a GFD y biblioteca.
  - Diseño más rápido.
  - Menor probabilidad de quedar atrapado en pozos de coste.
- Idea:

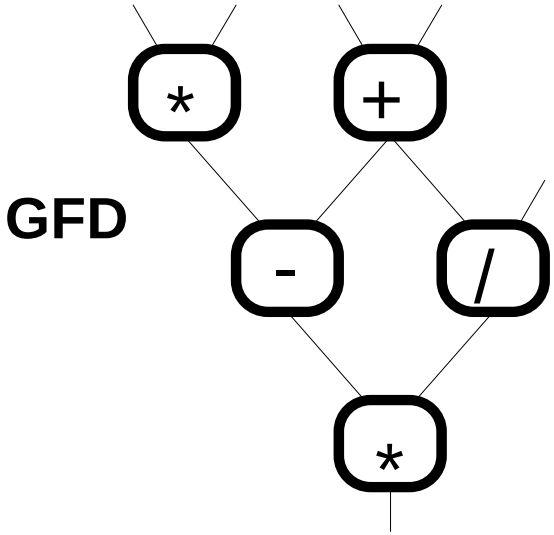


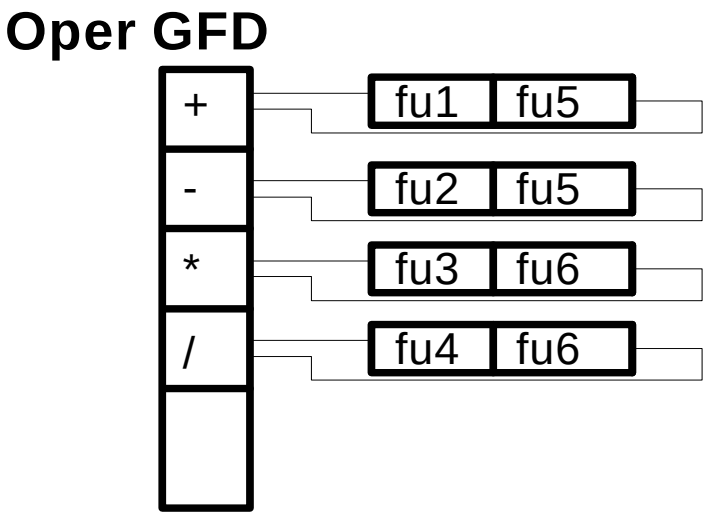


- Aspecto más crucial
  - En una biblioteca no restringida, selección de UFs más apropiadas para el diseño => orden de acceso a biblioteca

Estrategias de guía (2)

- Importancia de la ordenación temporal (ejemplo)



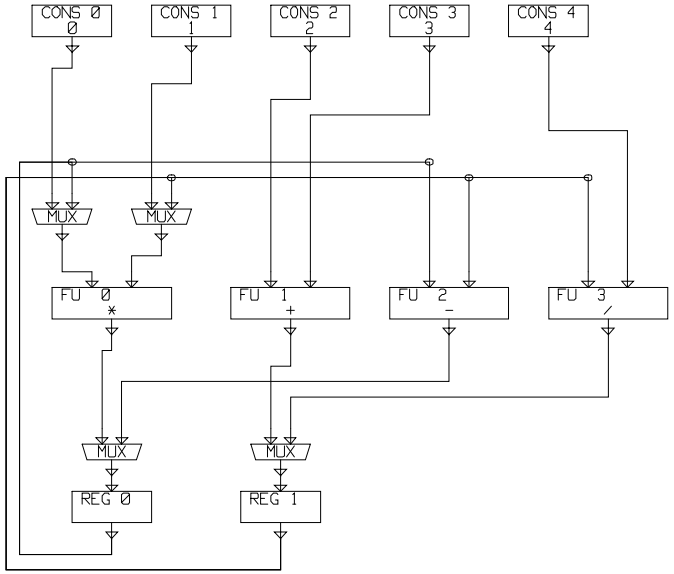


Lib FU

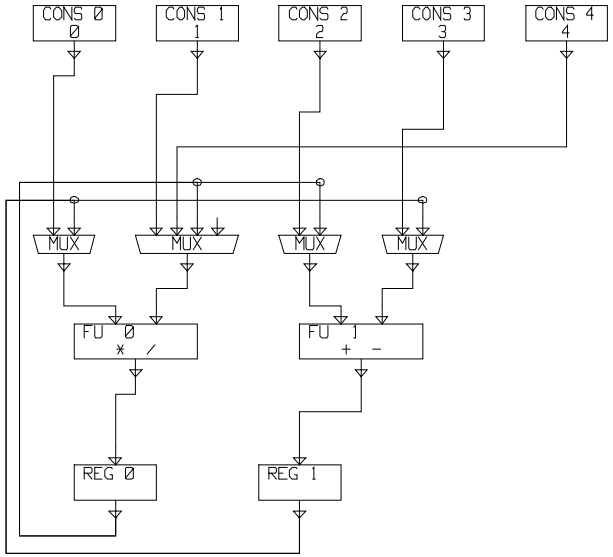
fu1	fu2	fu3	fu4	fu5	fu6	fu
+	-	*	/	+,-	*,/	

Estrategias de guía (3)

Influencia del sentido del recorrido de la biblioteca



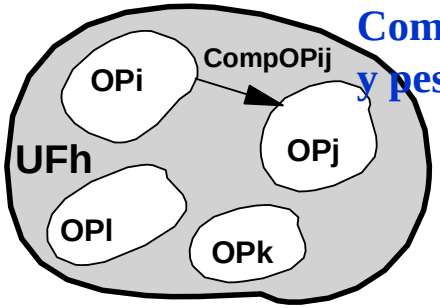
De izquierda a derecha



De derecha a izquierda

Estrategia de guía (4)

- Necesidad de una estrategia de acceso personalizada que favorezca ensayar primero
  - UFs multioperacionales que supongan ahorros notables de HW, siempre que
  - su funcionalidad pueda usarse eficientemente para diversas operaciones .



Compatibilidad y peso

Tabla comparativa

	Coste	Heurística	CPU tiem.	# Diseño	Exit
FACET	39339	AD	1	4	opt.
	36381	AM (.1)	1	7	opt.
	35898	AM (.2)	2	8	opt.
	35898	AG	6	8	opt.
ECUACION DIFERENC.	43367	AD	1	3	opt.
	43303	AM (.1)	5	3	t.o.
	43303	AG	2760	6	t.o.
FILTRO	50834	AD	116	18	opt.
	48420	AM (.1)	133	21	t.o.
	51516	AG	5340	23	t.o.

$Solap_{ij}$ = nº de veces que op_i aparece en un Cstep acompañado de op_j

$CompOP_{ij}\ (UF_h) = (1 - Solap_{ij} / Num\_op_i) \times ((Area_i + Area_j) / AreaUF_h)$

$Peso_i\ (UF_h) = max_j\ (CompOP_{ij}\ (UF_h))$

Ordenación: decreciente en peso