PONTIFICIA UNIVERSIDAD CATÓLICA DE VALPARAÍSO
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA INFORMÁTICA

Desafío 1: algoritmo.doc

FRANCO ARDILES
RODRIGO GALLEGUILLOS
IGNACIO GARCÍA
BRAULIO LOBO

ESTRUCTURA DE DATOS AVANZADAS Y ALGORITMOS
OII443-1
INGENIERÍA CIVIL INFORMÁTICA

30 DE AGOSTO, 2020

El propósito de nuestro algoritmo es resolver el problema de bin packing de manera eficiente, rápida y confiable. El enfoque trata acerca de optimizar el dimensionado de tablas en el ámbito de construcción o albañilería. Con este algoritmo podemos obtener la cantidad mínima de tablas a utilizar para cotar una lista de dimensiones, así optimizar el material a utilizar.

Para este algoritmo definimos estados y acciones para así resolver el problema a través de grafos implícitos. Los estados encapsulan la información y representación del problema y las acciones representan la transición de un estado a otro.

Los estados y las acciones están definidos de la siguiente manera:

Definición de Estado y Acción

```
clase Estado:
    dims: Lista de dimensiones a cortar
    planks: Lista de dimensiones restantes en cada tabla
    cuts: Lista de cortes por tabla
    pl: Largo total de las tablas
    ub: Upper Bounde
    visited: Booleana que indica si este estado ha sido visitado
    bf: Heuristica para best-fit. Suma 1 cuando llega al 100% de una tabla

    constructor(dims, largo_tablas):
        visited = Falso
        dims = dims
        pl = largo_tablas
        ub = cantidad(dims)
        cuts = nueva lista
        bf = 0

        // Truncando cortes mas largo que las tablas
        cada dimension en dims:
            si dimension > pl:
                dimension = pl

        // Calculando cantidad minima de tablas
        total_corte = sumatoria(dims)
        total_tablas = redondear(total_corte/pl)
        planks = nueva lista de largo total_tablas

class Accion:
    cut: Dimensión del corte
    plank: Tabla a la que se le va a realizar el corte
```

La técnica utilizada para la resolución del grafo fue el recorrido $Best\_first$ junto con una heurística admisible que asegurara que el estado resultante sería un estado óptimo.

El comportamiento de esta técnica recorre el árbol de estados siguiendo siempre por el camino más prometedor. Nuestro algoritmo evalúa los estados según el porcentaje de llenado de las tablas, la cantidad de tablas y la cantidad de tablas con llenado perfecto.

Se utiliza la implementación de una cola con prioridad para siempre seleccionar el mejor estado. Por otro lado basta con que encuentre sólo una solución porque utiliza siempre el mejor estado posible. Esto significa que basta con comprobar que el largo del arreglo de dimensiones esté vacío para finalizar la búsqueda.

Para obtener todas las acciones posibles se prueba la primera dimensión del arreglo de dimensiones en todas las tablas. Si no se encuentran acciones posibles, se añade una nueva tabla y se repite la búsqueda de acciones. Luego, el algoritmo selecciona desde la cola con prioridad el mejor estado. En la siguiente iteración comienza una nueva búsqueda de acciones posibles a través del mejor estado de la iteración anterior.

La utilización de la búsqueda $Best\_first$ significa dejar de iterar sobre estados que no son prometedores. Por otro lado las dimensiones se pueden ordenar de manera conveniente para generar mejores estados desde un principio. Esto contempla probar primero los cortes mas grandes.

Búsqueda Best First

```
estado_inicial : Estado aleatorio
c_pioridad.push(estado_inicial)
mientras c_prioridad.largo > 0
   nodo : c_prioridad.pop()
   si estado_final(nodo)
      fin, retorna nodo
   acciones = obtener_acciones(nodo): lista de acciones para el nodo
   para accion en acciones
      nuevo_nodo = transicion(nodo,accion) : resultado de aplicar accion
      si nuevo_nodo no está c_prioridad
          c_prioridad.push(nuevo_nodo)
```

Los resultados obtenidos son bastantes buenos. Exceptuando algunas instancias realmente poco comunes, gran parte de las tablas se logran aprovechar en un valor mayor al 97 % dejando sólo la última tabla con un valor menor al mencionado. Esto ocurre principalmente porque no hay mas cortes para aprovechar mayor parte de la tabla.
Por motivos meramente experimentales realizamos algunas pruebas con distintos algoritmos de búsquedas y realmente no pudimos superar en forma notable los resultados finales en cuanto a cantidad de tablas utilizadas para solventar la cantidad de cortes. Por otro lado, algoritmos como búsqueda en anchura o algunos mas intensivos igualan u obtienen el mismo resultado pero a coste de un tiempo de procesamiento notablemente mayor.