TÉCNICO LISBOA

2º Projeto de ASA - 2º Semestre 2013/2014

GRUPO 187 - Carlos Correia nº76512 Gonçalo Mendes nº77044

Breve Introdução

A resolução de problemas complexos com base em soluções algorítmicas bem definidas tem tido uma expansão constante de forma acentuada com o aparecer de recursos que potenciem resultados significativos em espaços de tempo que definam um compromisso entre custos e performance. É esse o âmbito principal de ramos interdisciplinares como a Investigação Operacional em métodos de apoio à decisão. O método que mais se destaca quando entram em jogo redes de fluxos é o de Ford Fulkerson que generaliza uma forma de se obter fluxo máximo a partir de uma fonte até um determinado término. O projecto foi implementado com base no algoritmo que obedece ao teorema de corte minimo fluxo-máximo e pelo qual se encontra a solução. No modificar de algoritmos conhecidos genéricos reside a solução de inúmeros problemas de Engenharia nos dias que correm, e passa por nós estudantes, assimilar este tipo de conhecimento para soluções complexas que surjam no dia à dia.

Solução Implementada

Sendo conhecido o método de **Ford Fulkerson** foi implementado um algoritmo que tem por base este conhecimento dos caminhos de aumento, e que percorre o grafo de forma sistemática e ordenada, utilizando procura em largura, **uma BFS** para explorar o grafo residual da rede de fluxo. Deste modo quando o algoritmo acaba de correr temos o fluxo máximo calculado.
Esta abordagem genérica apenas permite obter o fluxo a partir de arcos com capacidade 1, dado não ter sido referida nem tendo relevância para este caso uma capacidade diferente entre os arcos do grafo. Ao haver arco de um aglomerado(vértice) para outro a capacidade é 1, caso contrário é 0.

Para obter o *ouput* desejado temos que recorrer ao teorema do corte-minimo/fluxo máximo, num algoritmo criado pelo grupo. Tendo uma fonte e um vértice destino aplica o algoritmo Ford Fulkerson à rede de fluxo com a procura em largura, sendo também chamado de Edmonds-Karp. Depois disto feito é realizada uma **DFS**, procura em profundidade no grafo residual para descobrir os vértices atingíveis a partir da origem, sendo assim possível descobrir os cortes. Se o vértice onde nos encontramos tiver sido marcado como atingível mas o seu adjacente não, e a capacidade for positiva, temos um corte. O procedimento é iterativo.

Tendo esta parte do programa feita, o restante é bastante “straightforward”. Basta para cada problema dado no input aplicar o algoritmo marcando como *source* o primeiro ponto critico e sink o segundo, mantendo o 1º e ir alterando a *sink* até ao n-ésimo ponto crítico, sendo a cada passo o algoritmo corrido; De seguida é marcado o 2º ponto crítico como *source* corrido com os n seguintes como *sink* e assim sucessivamente. O número minimo de cortes do conjunto de iterações resulta no output do dado problema. A figura ilustra este procedimento.

3-0	0-2	2-1
3-2	0-1	
3-1		
s-t	s-t	s-t

s - source
t - sink

Fig.1 Algoritmo corrido entre todos estes pares (exemplo)

Para armazenamento e manipulação de dados mantivemos uma FIFO para auxilio na BFS, utilizamos arrays alocados dinamicamente e para manter a estrutura do grafo foi utilizada uma lista de adjacências . Esta foi a representação considerada pondo de parte a matriz de adjacências por ocupar menos espaço, ser mais eficiente, e bastar ler a lista de adjacências de um vértice para encontrar os seus vértices adjacentes.

O nosso programa foi codificado em **linguagem C** visto ambos os membros do grupo terem maior experiência com esta, e por apresentar menor grau de abstração relativamente a outras, sendo possível realizar operações com maior liberdade sem restrições comparado a Java, segunda linguagem com qual temos alguma familiaridade.

Posto isto, a nossa lista de adjacências foi construída recorrendo a **duas estruturas e um array**. Existe uma estrutura vertex que guarda apenas um ponteiro para a estrutura edge, estando as suas “instâncias” guardadas no array correspondente ao número de vértices. A segunda estrutura, **edge** contem três campos, o índice do vértice no array, a capacidade do arco e o ponteiro para a mesma estrutura, criando assim uma lista ligada com as adjacências. Esta lista é duplicada no inicio do programa para manter o Grafo Residual.

Análise Teórica e Avaliação Experimental

O tempo de execução do Algoritmo Edmonds Karp que utiliza o método de Ford Fulkerson é de **O(V*E*^{2})**. Cada caminho de aumento é encontrado em O(E) tempo. A DFS utilizada tem tempo de O(E) de execução.