MC658 - Projeto e análise de algoritmos III - Unicamp

Laboratório 5 – s-t Caminhos com Coleta de Prêmios

Erik Perillo, RA135582

2 de julho de 2017

1 Problema

O problema consiste em, dado um grafo orientado $G = (N, A)$, encontrar o melhor caminho $P = (N_P, A_P)$ de s a t que maximize uma certa função de custo.

2 Heurística

De modo a obter um bom *lower bound* para o problema, foi desenvolvida uma heurística baseada na metaheurística *GRASP - Greedy Randomized Adaptive Search Procedure*. Os pseudo-códigos são descritos a seguir.

Algorithm 1

procedure GRASP-HEURISTICA(G, s, t, prize, cost)
    $S \leftarrow \emptyset$
    $S_+ \leftarrow \emptyset$
    while not *timeout* and not *limit_its* and not *stuck_in_local_opt* **do**
        $S \leftarrow$ RANDOM-GREEDY-SOL-DFS(S, s, t, prize, cost)
        $S \leftarrow$ LOCAL-SEARCH(S, G, prize, cost)
        if $cost(S) \geq cost(S_+)$ **then**
            $S_+ \leftarrow S$
        return S_+

Algorithm 2

procedure RANDOM-GREEDY-SOL-DFS(S, s, t, prize, cost)
    set s as visited
    if $s = t$ **then**
        return TRUE
    $A \leftarrow Adj(s)$
    while $A \neq \emptyset$ **do**
        select (s, v) with probability proportional to $prize(s) - cost((s, v))$
        $A \leftarrow A - \{(s, v)\}$
        if not visited(v) and RANDOM-GREEDY-SOL-DFS(S, v, t, prize, cost) **then**
            $S \leftarrow S \cup (s, v)$
            return TRUE
    return FALSE
=0

Algorithm 3

procedure LOCAL-SEARCH(S, G, prize, cost)
    while not *timeout* and not *limit_its* and not *stuck_in_local_opt* **do**
        for $(u, v) \in S$ **do**
            for $(u, w) \in A(G)$ and $(w, v) \in A(G)$ with $w \notin S$ **do**
                if $prize(w) - cost((u, w), (w, v)) \geq -cost(u, v)$ **then**
                    $S \leftarrow S \cup \{(u, w), (w, v)\} - \{(u, v)\}$
    return S

O algoritmo 1 é a metaheurística principal: enquanto o critério de parada não é verdade, ele busca uma solução gulosa-probabilística inicial e a refina com busca local. O algoritmo 2 explica a ideia da busca probabilística gulosa: Faz-se o caminho em DFS de s a t, escolhendo nós/arcos com probabilidades proporcionais a seus custos: quanto maior, maior a chance de escolher. O algoritmo 3 explica a ideia da busca local: para cada arco da solução aleatória-gulosa, ele procura outros dois arcos que formam um caminho entre os nós do arco original mas com custo melhor. Ele repete isso no máximo 3 vezes. A heurística obteve resultados sempre muito próximos do *upper bound*.

3 Solução exata com Programação Linear Inteira

Seja x_v a variável binária que é 1 se o nó v está na solução e 0 caso contrário. Seja x_a a variável binária que é 1 se o arco a está na solução e 0 caso contrário. Seja π_v os prêmios de cada nó v e c_a os custos de cada arco a.

Queremos encontrar o caminho de s a t $P = (N_P, A_P)$ que maximize:

$$\texttt{max} z = \sum_{v \in N} x_v \pi_v - \sum_{a \in A} x_a c_a$$

Sujeito a:

- Do nó s, só tem um arco de saída e nenhum de entrada:

$$\sum_{x_a \in \delta_+(s)} x_a = 1$$

$$\sum_{x_a \in \delta_-(s)} x_a = 0$$

- Do nó t, só tem um arco de entrada e nenhum de saída:

$$\sum_{x_a \in \delta_+(t)} x_a = 0$$

$$\sum_{x_a \in \delta_-(t)} x_a = 1$$

- Um nó que não seja s e t é selecionado se e somente se ele tem exatamente um arco de entrada e um de saída:

$$\sum_{x_a \in \delta_+(v)} x_a = x_v, \ \forall \ v \in N - \{s, t\}$$

$$\sum_{x_a \in \delta_-(v)} x_a = x_v, \ \forall \ v \in N - \{s, t\}$$

- Restrição de integralidade:

$$x_a \in \{0, 1\}, \ \forall \ x_a \in A$$

$$x_v \in \{0, 1\}, \ \forall \ x_v \in N$$

4 Implementação

A implementação foi feita com o uso do gurobi (versão `7.0.2`) e lemon.

5 Avaliação do modelo proposto

Na tabela a seguir, temos os valores para alguns arquivos de entrada. O algoritmo exato recebia o *lower bound* obtido pela heurística como *cutoff*. A heurística mostrou-se muito eficiente, obtendo *lower bounds* muito próximos do valor ótimo. O valor ótimo foi obtido em quase todos os casos. No caso em que o OPT não foi obtido, a heurística obteve um *lower bound* muito próximo do valor ótimo (3.00684e+09). O arquivo `100000_700000.in` não pôde ser executado na máquina dos testes pois ela não tinha memória suficiente.

Tabela 1: Resultados de execução com lower bounds (LB) e upper bounds (UB) obtidos.

entrada	LB	UB	OPT obtido?	tempo limite (s)
3_2.in	6	13	sim	1
30_600_1.in	9.33478×10^6	9.33510×10^6	sim	1
100_5000_1.in	2.86350×10^7	2.86361×10^7	sim	1
1000_20000_1.in	3.07347×10^8	3.07367×10^8	sim	1
10000_9000000_1.in	3.00666×10^9	3.00666×10^9	não	60