Trabalho Final de Otimização

Eduardo Fantini[1], **João Pedro S. Silva**[2]

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brazil

eduardo.fantini@inf.ufrgs.br[1], joao.silva@inf.ufrgs.br[2]

Resumo. *Este trabalho apresenta a formulação matemática de um problema de otimização e compara o desempenho de um solver à uma heurística utilizando Late Acceptance Hill Climbing (LAHC). O objetivo é aplicar os conhecimentos aprendidos na disciplina de Otimização Combinatória (INF05010).*

1. Problema

Dado um grafo $G = (V, E)$ representando um mapa com um vértice inicial v_0, desejamos colocar caixas em posições dadas por vértices v percentence a V. Isso deve ser feito de modo que toda caixa (digamos, em um vértice v) seja acessível a partir de v_0. Isto é, deve existir um caminho de v_0 a v que não usa nenhum vértice ocupado por outra caixa. Em particular, v_0 não deve ser ocupado por nenhuma caixa. Encontre uma maneira de maximizar o número de caixas colocadas respeitando as restrições acima.

2. Definição do problema de otimização

A primeira informação extraída é a necessidade de garantir que haja um caminho entre cada vértice com caixa até a origem.

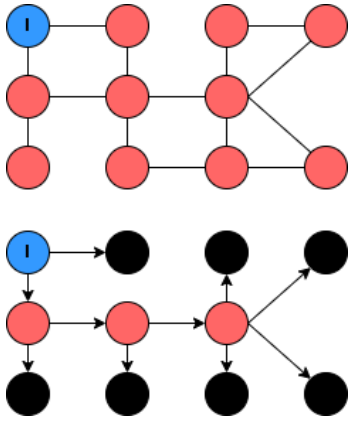

Figura 1. Exemplo de grafo com possível solução. Vértices com caixa são representados em preto, enquanto vértices livres estão em vermelho.

Na Figura 1 temos um grafo de entrada que possibilita múltiplas soluções, sendo que toda aresta entre dois vértices garante um possível caminho. Logo abaixo do grafo temos um exemplo de solução com os vértices com caixa destacados em preto, onde podemos notar que as arestas de caminho foram definidas, além de serem direcionais dado que vértices com caixa não podem ser utilizados para acessar outros vértices.

Nossa primeira abordagem será garantir que exista um caminho para cada caixa a partir da origem. Para isto primeiro definimos uma matriz tridimensional X, onde a

primeira dimensão representa um vértice qualquer em V, tirando a origem, e a segunda e a terceira representam cada possível combinação de vértices. O valor de cada índice de X será 1 caso exista um caminho entre o par de vértices e 0 caso não exista.

$$X_{n-1,n,n}$$

Também vamos definir uma lista C - que definirá se cada vértice em V possui ou não uma caixa - e uma restrição que impeça que o vérticie inicial v_0 possua uma caixa.

$$C \in \mathbb{Z}^n$$

$$C_0 = 0$$

Dado uma matriz bidimensional A representando as ligações entre vértices no grafo G, onde cada índice indica se há uma ligação, valor 1, ou não, valor 0. Com isto podemos definir nossa primeira restrição, dado um vértice v_d, sendo uma posição de caixa possível, e outros dois vértices, v_x e v_y, quaisquer do grafo, em X o valor do índice d_xy será 1 se, e somente se, houver uma ligação no grafo original de v_x para v_y e v_x não possui caixa.

$$X_{dxy} <= A_{xy} * (1 - C_x) \quad \forall d \in V/\{0\} \quad \forall x, y \in V$$

É necessário definir que o vértice origem deve conectar apenas um vértice e não deve ser conectado por nenhum outro vértice. Também iremos definir M, como uma constante grande o suficiente para ser maior qualquer resultado possível do lado esquerdo das inequações as quais pertence. A constante M será utilizada para ignorar a restrição para todo vértice d que não possua uma caixa, visto que o problema não define que estes devem ser acessíveis a partir vértice inicial.

$$\sum_{v \in V} X_{div} - \sum_{v \in V} X_{dvi} >= 1 - M * (1 - C_d) \quad \forall d \in V/\{0\}$$

$$\sum_{v \in V} X_{div} - \sum_{v \in V} X_{dvi} <= 1 + M * (1 - C_d) \quad \forall d \in V/\{0\}$$

Note que a soma das duas restrições acima, quando $C_d = 1$, significa que a subtração da esquerda deve ser igual a um.

De forma semelhante vamos definir que para cada possível vértice destino é necessário que ele seja conectado por apenas um vértice e não conecte ninguém.

$$\sum_{v \in V} X_{ddv} - \sum_{v \in V} X_{dvd} >= -1 - M * (1 - C_d) \quad \forall d \in V/\{0\}$$

$$\sum_{v \in V} X_{ddv} - \sum_{v \in V} X_{dvd} <= -1 + M * (1 - C_d) \quad \forall d \in V/\{0\}$$

Por fim teremos uma restrição para cada vértice intermediário de cada caminho entre o vértice inicial e cada possível vértice com caixa. Esta restrição de fluxo irá definir

que para vértice que se conecte a eles, estes terão que se conectar a outro, tornando o fluxo constante.

$$\sum_{v \in V} X_{duv} - \sum_{v \in V} X_{duv} >= 0 - M * (1 - C_d) \quad \forall d \in V/\{0\} \quad \forall u \in V/\{0, d\}$$

$$\sum_{v \in V} X_{duv} - \sum_{v \in V} X_{duv} <= 0 + M * (1 - C_d) \quad \forall d \in V/\{0\} \quad \forall u \in V/\{0, d\}$$

Nosso objetivo será maximizar o número de caixas possíveis no nosso grafo, e podemos definir isto usando a nossa lista C dado que a mesma possuirá o valor um caso haja uma caixa sobre um vértice em determinado estado e zero caso não haja.

$$max \sum_{v \in V} C_v$$

2.1. Definição completa do problema de otimização

$$max \sum_{v \in V} C_v$$

$$s.a \quad C_0 = 0$$

$$X_{dxy} <= A_{xy} * (1 - C_x) \quad \forall d \in V/\{0\} \quad \forall x, y \in V$$

$$\sum_{v \in V} X_{div} - \sum_{v \in V} X_{dvi} >= 1 - M * (1 - C_d) \quad \forall d \in V/\{0\}$$

$$\sum_{v \in V} X_{div} - \sum_{v \in V} X_{dvi} <= 1 + M * (1 - C_d) \quad \forall d \in V/\{0\}$$

$$\sum_{v \in V} X_{ddv} - \sum_{v \in V} X_{dvd} >= -1 - M * (1 - C_d) \quad \forall d \in V/\{0\}$$

$$\sum_{v \in V} X_{ddv} - \sum_{v \in V} X_{dvd} <= -1 + M * (1 - C_d) \quad \forall d \in V/\{0\}$$

$$\sum_{v \in V} X_{duv} - \sum_{v \in V} X_{duv} >= 0 - M * (1 - C_d) \quad \forall d \in V/\{0\} \quad \forall u \in V/\{0, d\}$$

$$\sum_{v \in V} X_{duv} - \sum_{v \in V} X_{duv} <= 0 + M * (1 - C_d) \quad \forall d \in V/\{0\} \quad \forall u \in V/\{0, d\}$$

3. Implementação em Julia

Além da definição do problema utilizando os cinco subconjuntos de restrições apresentados na seção anterior, fizemos o uso de duas bibliotecas que facilitaram o desenvolvimento.

3.1. Gurobi

Optamos por utilizar o solver Gurobi através de uma licença acadêmica por trazer algumas vantagens como uma melhor análise durante a execução, fato importante para encontrar problemas de implementação, e também uma melhor performance.

Comparamos ambos solver com a primeira instância do problema, abaixo temos a média de tempo de execução de ambos. Dados os resultados optamos por utilizar o Gurobi.

Solver	Tamanho do Grafo	Tempo	Entrou solução ótima
GLPK	12x17	293 ms	Sim
Gurobi	12x17	285 us	Sim

3.2. Utilização do LightGraphs

O LightGraphs é uma das principais bibliotecas para manipulação de grafos em Julia. Ela aumentou a legibilidade do nosso código quanto a manipulação dos grafos e notamos uma melhora na definição das restrições do problema, que realizam várias operações de leitura.

4. Late Acceptance Hill Climbing

O Late Acceptance Hill Climbing (LAHC) é um algoritmo de busca local baseado no Hill Climbing (HC). A principal diferença entre o HC e o LAHC é o fato do segundo aceitar soluções piores que a solução atual durante sua execução, assim como o Simulated Annealing, porém sem necessitar de um paramêtro que regula o quão inferiores as novas soluções aceitas poderão ser.

Como não possui uma dependência de performance por paramêtros que precisam ser regulados a cada novo problema, o LAHC se torna uma solução simples e rápida para encontrar boas soluções sobre problemas pouco conhecidos.

Durante sua execução, cada nova solução gerada é comparada com um histórico de soluções possíveis, sendo que ela será adicionada a este histórico caso seja melhor que alguma outra solução salva. Para cada nova solução adicionada ao histórico removemos a pior solução salva.

O critério de parada utilizado é o mesmo do HC, quando o algoritmo não é mais capaz de encontrar nenhuma solução melhor dentro de um certo número de iterações. Também é importante definir um número mínimo de iterações para que o algoritmo não termine de forma prematura.

4.1. Geração de vizinhança

A geração de soluções vizinhas atual é feita a partir de uma alocação aleatória de caixas (à exceção da origem), que é aplicada como máscara na solução atual. Essa ideia veio

de um desejo de não permitir que os vizinhos divirjam tanto do nodo atual. Tal máscara é aplicada por meio de operações lógicas *or* e *xor*, elemento a elemento, sendo que a operação é escolhida aleatóriamente.
Ao longo da implementação desse algoritmo, percebemos que o *xor* funcionava melhor para instâncias menores, enquanto o *or* funcionava melhor para instâncias maiores. Por isso buscamos uma forma de aproveitar o melhor de cada, e acabamos tendo bons resultados com uma escolha aleatória entre elas, mas não sabemos explicar o por quê.

4.2. Verificação de factibilidade

Como geramos a vizinhança de maneira aleatória, nada impede de gerarmos uma solução infactível. Por isso, para cada solução gerada, verificamos se todas caixas são acessíveis, aquelas que não forem são removidas.

5. Resultados

Devido à complexidade do problema e do conjunto de restrições, que aumenta de forma exponencial baseada no número de vértices, a utilização do solver para encontrar soluções ótimas se tornou inviável para grafos maiores. Porém para conjuntos menores foi possível encontrar uma solução ótima em um tempo pequeno.

Para soluções maiores a heurística de tornou mais eficiente, principalmente em questão de tempo, porém os resultados encontrados em sua maioria são ótimos locais.

Como será mostrado na Tabela 1 abaixo:

- para instâncias pequenas, mesmo que a Heurística alcance resultados piores que o Solver, a diferença é aceitável.
- para instâncias maiores, a Heurística consegue obter resultados próximos aos do Solver - dada a limitação de 5 minutos imposta a este - em menos tempo.

Considerando a simplicidade da Heurítica, para as duas maiores instâncias do problema o resultado encontrado não foi satisfatório. A geração de vizinhos de forma aleatória se mostrou não eficiente para resolver este problema, e o aumento do tamanho do histórico e da tolerância a falhas consecutivas não se mostraram efetivos para melhora do algoritmo.

Devido à limitações de memória e poder computacional, não utilizamos o Solver nas maiores instâncias do problema.

Referências

Burke, E. K. and Bykov, Y. (2017). The late acceptance hill-climbing heuristic. *European Journal of Operational Research*, 258(1):70–78.

Instância	Solver (Gurobi)		Heurística (LAHC)	
	BKV	**Tempo (minutos)**	**BKV**	**Tempo (minutos)**
n=12, m=18	7	$8.66e^{-4}$	6	$1.45e^{-4}$
n=50, m=60	27	$5.62e^{-2}$	18	$6.06e^{-4}$
n=50, m=85	25	$5.00e^{0}$	19	$5.74e^{-4}$
n=50, m=120	37	$5.00e^{0}$	28	$5.24e^{-4}$
n=50, m=300	44	$5.00e^{0}$	39	$5.03e^{-4}$
n=100, m=180	53	$5.00e^{0}$	35	$1.14e^{-3}$
n=100, m=245	74	$5.00e^{0}$	54	$1.00e^{-3}$
n=100, m=490	78	$5.00e^{0}$	72	$1.13e^{-3}$
n=100, m=1225	91	$5.00e^{0}$	87	$1.64e^{-3}$
n=150, m=555	46	$5.00e^{0}$	96	$1.74e^{-3}$
n=150, m=1110	126	$5.00e^{0}$	116	$2.00e^{-3}$
n=150, m=2775	130	$5.00e^{0}$	135	$3.34e^{-3}$
n=500, m=6225	NA	NA	422	$1.16e^{-2}$
n=500, m=12450	NA	NA	455	$1.85e^{-2}$
n=500, m=31125	NA	NA	481	$4.32e^{-2}$
n=1500, m=2920	NA	NA	116	$2.04e^{-2}$
n=10000, m=19800	NA	NA	134	$1.40e^{-1}$

Tabela 1. Comparação de resultados entre Solver e Heurística.
O tempo de execução de ambos algortitmos foi limitado a 5 minutos.