Trabalho Prático II: Problema da Mochila 0/1 com Conflitos

Bruno Dias
brunolagedias@gmail.com
Universidade Federal de Ouro Preto
Ouro Preto, Brasil

RESUMO

O presente relatório descreve os resultados práticos advindos da implementação de um algoritmo para a resolução do Problema da Mochila 0/1 com Conflitos. Foram descritos as estratégias adotadas na concepção e no desenvolvimento do método *Branch and Bound* utilizado para a busca e o melhoramento de soluções. Além disso, são apresentados os resultados da execução de experimentos empíricos em 20 instâncias fornecidas para avaliar a corretude do projeto.

KEYWORDS

Analysis of Algorithms, Knapsack Problem, Empiric Study, Branch And Bound

ACM Reference Format:
Bruno Dias. 2020. Trabalho Prático II: Problema da Mochila 0/1 com Conflitos. In *Projeto e Análise de Algoritmos (TPI)*. PAA, Ouro Preto, MG, Brasil, 4 pages. fi

1 INTRODUÇÃO

O Problema da Mochila 0/1 é pertencente à classe NP-completo, ou seja, ele um problema que, entre outras características, possui um tempo de resolução exponencial. Em sua versão de otimização, busca-se maximar o lucro obtido ao preencher uma mochila com um conjunto de itens, que por sua vez possuem seus respectivos pesos e valores. Por se tratar de problema 0/1, um item só pode estar ou não estar na solução, sendo que a sua capacidade máxima da mochila deve ser respeitada.

O presente relatório descreve uma implementação de um algoritmo para a resolução do Problema da Mochila 0/1 com Conflitos, onde há a adição da restrição dos conflitos entre itens, ou seja, soluções válidas não podem conter itens conflitantes entre si. O método utilizado em tal algoritmo foi o *Branch and Bound*, cuja codificação se deu na linguagem C++ utilizando a Programação Orientada à Objetos. Também foram relatados os resultados dos projeto para 20 entradas fornecidas.

Este documento possui a seguinte organização. A Seção 2 aborda o algoritmo de *Branch and Bound* concebido e as suas principais características. A Seção 3 descreve a metodologia adotada para analisar a corretude do código desenvolvido. A Seção 4 apresenta os resultado obtidos com a execução de instâncias fornecidas para o Problema da Mochila 0/1 com Conflitos. A Seção 5 apresenta as conclusões finais deste projeto.

2 BRANCH AND BOUND

O *Branch and Bound* (B&B) é um algoritmo utilizado para encontrar a solução ótima em problemas de otimização. Ele se baseia na execução de duas operações em uma árvore de soluções, sendo elas o *Branch* (ou expansão) e o *Bound* (ou poda). A seguir são descritos especificações e detalhes do algoritmo implementado neste projeto.

2.1 Arvore de solução

Na concepção do B&B é visto comumente a adoção de dois modelos de árvore de soluções. Considerando o Problema da Mochila 0/1 com Conflitos no primeiro modelo, cada nível representa a adoção ou não adoção de uma variável na resposta final. Conforme mostra a Figura 1, essa árvore tende a permanecer balanceada caso podas não sejam realizadas. Além disso, a sua altura é igual ao número de variáveis consideradas.

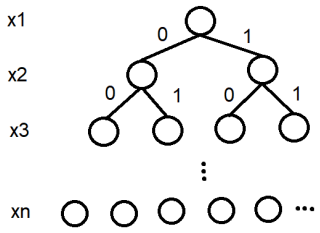

Figura 1: Modelo de arvore de soluções balanceada.

No segundo modelo de implementação da árvore de soluções, cada nó representa a adoção de uma variável e ao expandi-los, seus filhos serão as variáveis que ainda não foram incluídas na resposta até então. Vale ressaltar nesse modelo que a cada nível, o nó mais à direita representa uma resposta completa visto que ao se observar as respostas obtidas em um mesmo nível, a medida que se vai da esquerda para a direita, mais variáveis são consideradas ao mesmo tempo. Portanto, diferente da primeira abordagem sem considerar as podas, sempre será gerada uma árvore desbalanceada que cresce mais à esquerda como visto na Figura 2.

Neste projeto foi adotado o segundo modelo, sendo que cara nó, além de armazenar a solução atual, também possui um *Upper Bound* e um *Lower Bound* para auxiliar nas operações de expansão e poda.

2.2 Lower Bound

O Lower Bound (LB) de um nó refere-se a uma solução do Problema da Mochila 0/1 com Conflitos que limita inferiormente a solução que pode ser encontrada no caminho onde o nó está presente. Em nós intermediários ele pode ser visto como uma solução parcial, já em nós folhas ele será uma solução completa.

Diversas heurísticas podem ser adotadas para realizar o calculo do LB, sendo que a adotada neste projeto foi uma heurística gulosa.

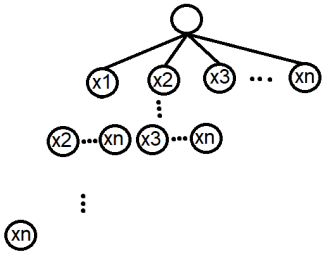

Figura 2: Modelo de arvore de soluções desbalanceada.

Nela, o Problema da Mochila 0/1 com Conflitos é resolvido a partir da escolha primária dos itens disponíveis que possuem a maior densidade, ou seja, a maior razão entre o valor do seu lucro pelo valor de seu peso.

2.3 Upper Bound

O Upper Bound (UB) de um nó no Problema da Mochila 0/1 com Conflitos refere-se a um valor que limita superiormente a solução, ou seja, é o potencial máximo pode se encontrar ao percorrer o caminho onde o nó está presente.

Para calcular o UB, houve a adoção de uma técnica conhecida como relaxação. Em problemas de otimização, essa técnica gera um problema semelhante ao original, só que com menos restrições. Tal retirada de restrições torna problemas muito complexos triviais, sendo o calculo de uma solução a partir deles uma operação rápida. Para o Problema da Mochila 0/1 com Conflitos, a restrição retirada foi a de integralidade das variáveis, transformando ele no Problema da Mochila 0/1 Fracionário com Conflitos.

2.4 Critérios para o Branch

O operação de *Branch* se refere a expansão dos nós da árvore de solução. Nela, um problema é dividido em subproblemas mais fáceis a partir da fixação da variável representada pelo nó como presente no conjunto de soluções.

Na presente implementação do B&B, o critério utilizado para realizar uma *Branch* foi o do maior UB. Portanto, dentre os nós folhas presentes na árvore que estão aptos a serem expandidos, será expandido aquele que possuir o maior UB, ou seja, o maior potencial de lucro.

2.5 Condições para o Bound e para uma solução

Sendo o *Branch* uma operação que naturalmente leva a uma árvore de tamanho exponencial, o *Bound*. Nele, as restrições e os limites são observados junto a solução incumbente, levando a ponderações que permitem determinar a aptidão de um nó.

No Problema da Mochila 0/1 com Conflitos há duas restrições que levam à poda de um nó. A primeira condição se dá quando uma solução ultrapassa a capacidade total da mochila. Já a segunda, ocorre quando variáveis conflitantes estão presentes numa mesma solução. Em ambos os casos as solução se encontra inválida e o nó não precisa ser expandido. A última condição de *Bound* advém da observação do UB e da solução incumbente. Caso um nó possua um UB inferior à solução incumbente ele não precisará ser explorado, dado que o melhor valor a ser encontrado no caminho em que ele está é inferior à melhor solução encontrada até o momento.

Um nó também pode deixar de ser expandido caso o mesmo possua uma solução completa. Isso pode ocorrer em duas ocasiões. A primeira é quando não há mais itens a serem considerados dentro da solução. Já a segunda é quando o UB se iguala ao LB. Nesse último caso, não há mais refinamento possível para a resposta advinda do caminho em que o nó está presente.

2.6 Tempo Limite

Conforme árvore de soluções do B&B é percorrida, soluções viáveis são encontradas e armazenadas como soluções incumbentes e caso o algoritmo termine de executar a solução ótima é encontrada e armazenada. Todavia, achar a solução ótima pode ser muito custoso, o que torna as soluções viáveis uma boa opção para situações com limite de tempo.

O limite do tempo foi uma restrição adicional para a algoritmo implementado neste projeto. Através de uma ferramenta de *benchmark* fornecida, foi definido um tempo limite para a execução, sendo quando tal limite era alcançado, as operações eram encerradas e a solução incumbente era retornada.

2.7 Classes

O código em questão foi implementado na linguagem C++ devido o alto desempenho da mesma. Além disso, foi utilizado o padrão de Programação Orientada à Objetos para organizar as funcionalidades e suas execuções.

A seguir estão descritas as classes codificadas junto a seus respectivos atributos e métodos.

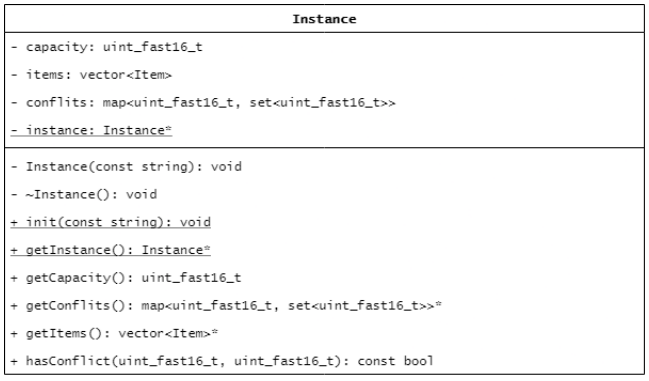

Figura 3: Classe `Instance`.

2.7.1 Instance. A classe `Instance` (Figura 3) representa os dados lidos das instâncias do Problema da Mochila 0/1 com Conflitos. Por ser uma classe com dados que serão utilizados ao longo de toda a execução, foi utilizado o padrão de projeto *Singleton* para possibilitar que somente uma instância da classe fosse criada e utilizada. Quanto aos seus demais atributos, eles representam as seguintes características:

- `capacity`: capacidade máxima da mochila;
- `items`: lista de itens que possuem seu identificador, seu peso e seu valor;

- `conflicts`: lista de adjacência que representa os conflitos entre itens.

Quanto aos métodos, além dos que constroem e retornam a instância do *Singleton* e os *getters* para o retorno os atributos, há o método `hasConflict`. Nele é possível verificar se há conflitos entre dois itens da mochila.

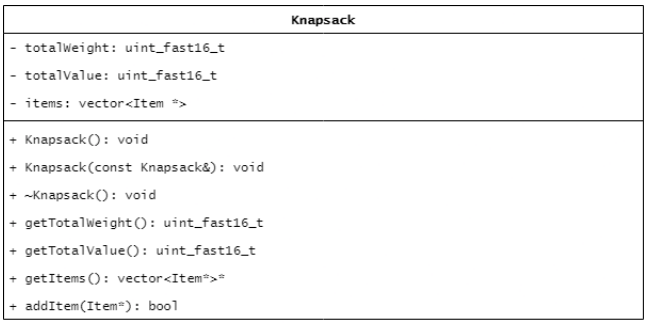

Figura 4: Classe `Knapsack`.

2.7.2 Knapsack. A classe `Knapsack` (Figura 4) representa uma mochila, que por sua vez serve para armazenar respostas para o Problema da Mochila 0/1 com Conflitos. Seus atributos representam respectivamente:

- `totalWeight`: peso total presente na mochila;
- `totalValue`: lucro total presente na mochila;
- `items`: lista de itens presentes na mochila.

Além dos *getters*, há o método `addItem` para adicionar um novo item na mochila. Ele retorna verdadeiro e atualiza o atributos da mochila caso o item seja adicionado com sucesso e retorna falso, caso haja um conflito com os itens já presentes ou ultrapasse a capacidade máxima da mochila.

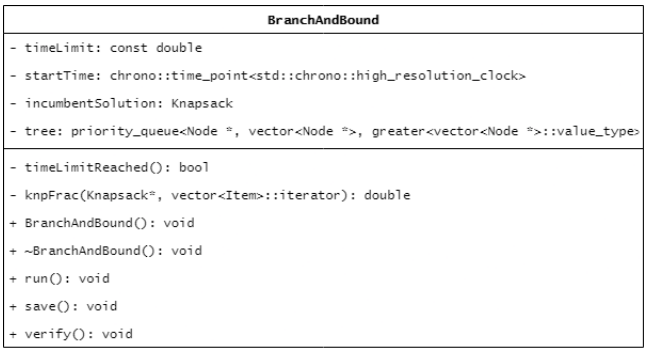

Figura 5: Classe `BranchAndBound`.

2.7.3 BranchAndBound. A classe `BranchAndBound` (Figura 5) representa o algoritmo B&B, incluindo seus métodos de resolução e a sua restrição de tempo. Os atributos dessa classe representam:

- `timeLimit`: tempo limite de execução do algoritmo em segundos;
- `startTime`: tempo em que começa o algoritmo;
- `incumbentSolution`: mochila com a solução atual;
- `tree`: lista de prioridade que representa a árvore de soluções.

Referente ao controle do tempo, o método `timeLimitReached` realiza a checagem se o tempo limite foi ultrapassado, para que se possa controlar a execução do algoritmo. Observando o UB, o método `knpFrac` executa o Problema da Mochila 0/1 Fracionário com Conflitos em um conjunto de itens e uma determinada mochila, retornando o valor de UB obtido.

O método `run` executa o algoritmo B&B, onde após atualizar o `startTime`, ele cria um nó raiz vazio para a árvore e o adiciona à lista de prioridade. Enquanto a `tree` não for vazia e ou o `timeLimit` não for ultrapassado, ele retira o nó presente do topo da lista para realizar a sua expansão, caso o mesmo possua um UB superior ao valor da mochila na `incumbentSolution`. Como `tree` é uma lista cuja prioridade é o maior UB, sempre que se retirar um item de sem topo, ele será o melhor candidato à expansão. Caso ele não seja expandido, ele só é removido.

Durante a criação dos nós filhos do nó expandido, é verificado primeiramente através de `timeLimitReached` se o processo pode continuar. Se sim, é verificado se o mesmo possui uma solução válida dadas as já mencionadas condições de *Bound*, ou se ele já possui uma solução completa. Caso ele seja cortado, o nó criado é simplesmente deletado. Caso seja uma solução final, a `incumbentSolution` é atualizada se o valor da solução for superior e o nó também é deletado. Caso não seja uma solução final e nem um nó cortado, ele é adicionado na `tree` para futuras expansões.

Além dos métodos de para a resolução do problema, havia o método `save` para a gravação dos resultados em um arquivo de texto e o método `verify` para verificar se uma resposta é válida.

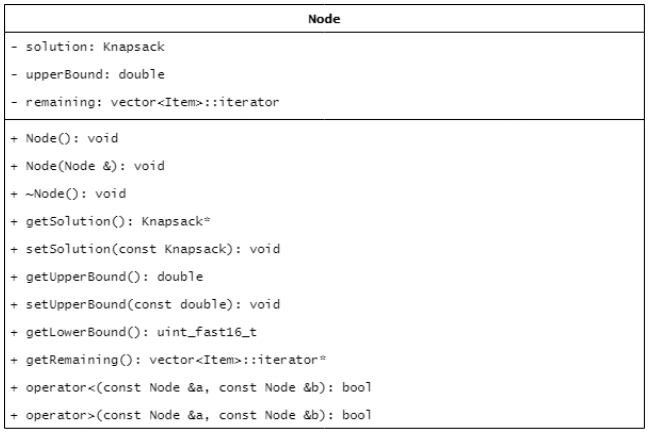

Figura 6: Classe `Node`.

2.7.4 Node. A classe `Node` (Figura 6) representa um nó da árvore de soluções. Ela armazena os seguintes atributos:

- `solution`: uma mochila com a solução atual do nó;
- `upperBound`: o UB do nó;
- `remaining`: um `iterator` para os valores os itens que ainda ondem entrar na solução, da lista da classe `Instance`.

Quanto aos métodos, além dos *getters* e *setters* dos atributos, e as sobrecargas de operadores, há o método `getLowerBound` que

retorna o LB do nó. Vale ressaltar que o calculo de LB já é realizado quando se adiciona um nó em um mochila, já que é considerado Problema da Mochila 0/1 com Conflitos, portanto esse método retorna o lucro da `solution`.

3 METODOLOGIA

Com o desenvolvimento finalizado foi necessário avaliar os resultados da implementação. Para isso foi realizada a execução de 20 instâncias previamente fornecidas do Problema da Mochila 0/1 com Conflitos. Elas foram executadas com e sem a restrição de tempo, sendo então os resultados gerados comparados com a solução ótima fornecida para cada uma delas.

3.1 Ambiente de Testes

Para a execução dos testes foi utilizado um notebook Samsung Expert x51 NP500R5L-YD2BR[1]. Essa máquina possui as seguintes especificações:

- **SO**: Windows 10 Home Single Language (x64);
- **Processador**: Intel® Core™ i7 6500U (2.5 GHz até 3.1 GHz 4 MB L3 Cache);
- **GPU**: NVIDIA® GeForce® 940MX Graphics com 2 GB de memória dedicada;
- **RAM**: 8 GB LPDDR3 1866 MHz;
- **Armazenamento**:
  - **Principal**: SSD WD Green M.2 2280 480GB SATA III 545 Mb/s WDS480G2G0B;
  - **Secundário**: HD 1 TB (5400 RPM).

O compilador utilizado foi o MinGW x64 (gcc version 9.3.0) e o limite de tempo fornecido pela ferramenta de *benchmark* foi de 771 segundos.

4 RESULTADOS E DISCUSSÕES

Com a execução das 20 instâncias fornecidas para o Problema da Mochila 0/1 com Conflitos, foi possível obter os resultados presentes na Tabela 1. Observando os dados obtidos pelas instâncias n150c50dc0.36, n450c20dc0.69, n600c19dc0.58, n650c18dc0.56, n700c49dc0.63, n750c46dc0.38 e n800c27dc0.56, é possível notar que a solução encontrada pelo implementação foi superior à solução ótima fornecida, abrindo margens para que uma das duas implementações estejam incorretas. Embora não se tenha a coleção de itens que resultou na solução fornecida, foi possível através do método `verify` constatar que as soluções encontradas foram válidas, ou seja, não excederam a capacidade da mochila e não incluíram itens conflitantes.

Quanto a restrição de tempo adotada, somente as instâncias n850c42dc0.32, n900c25dc0.40 e n950c50dc0.51 foram longas o suficiente para acioná-las. Embora isso tenha ocorrido, todas foram capazes de alcançar soluções iguais ou superiores à solução ótima fornecida e não em soluções menores, como já é de se esperar ao se considera um resultado incumbente. Vale ressaltar que a instância n1000c11dc0.62, apesar de possuir a maior quantidade de itens, foi resolvida antes do tempo limite adotado alcançando a sua solução ótima.

[1]Mais detalhes em: https://www.samsung.com/br/support/model/NP500R5L-YD2BR/

Tabela 1: Resultado da execução das 20 instâncias fornecidas.

instância	Nº de itens	Capacidade	Sem restrição de tempo	Com restrição de tempo	Ótima fornecida
n50c36dc0.64	50	36	21	21	21
n100c29dc0.60	100	29	33	33	33
n150c50dc0.36	150	50	55	55	54
n200c27dc0.42	200	27	23	23	23
n250c49dc0.63	250	49	36	36	36
n300c37dc0.30	300	37	34	34	34
n350c33dc0.39	350	33	39	39	39
n400c31dc0.68	400	31	20	20	20
n450c20dc0.69	450	20	35	35	33
n500c22dc0.46	500	22	26	26	26
n550c14dc0.39	550	14	22	22	22
n600c19dc0.58	600	19	31	31	29
n650c18dc0.56	650	18	24	24	23
n700c49dc0.63	700	49	55	55	47
n750c46dc0.38	750	46	52	52	48
n800c27dc0.56	800	27	35	35	30
n850c42dc0.32	850	42	70	70	66
n900c25dc0.40	900	25	42	42	42
n950c50dc0.51	950	50	38	38	35
n1000c11dc0.62	1000	11	14	14	14

5 CONCLUSÃO

O presente documento apresentou os detalhes de implementação do algoritmo *Branch and Bound* para a resolução do Problema da Mochila 0/1 com Conflitos, um problema NP-completo de resolução em tempo exponencial.

Através da codificação na linguagem C++ utilizando a Orientação à Objetos foi possível chegar em um resultado satisfatório que além de gerar soluções exatas para 20 instâncias do problemas, foi capaz de se aproximar das mesmas quando houve uma limitação em seu tempo de execução.