GRASP Aplicado ao Problema de Balanceamento de Linhas de Produção

Cristiano Medeiros Dalbem Fábio da Fontoura Beltrão
Lucas Fialho Zawacki

Otimização Combinatória (INF05010) - Prof. Marcus Ritt

Instituto de Informática
Universidade Federal do Rio Grande do Sul

6 de dezembro de 2010

Sumário

1 Introdução **2**

2 Definição do Problema **3**

3 Programação inteira **4**

3.1 Formulação inteira 4

3.2 Solução em GLPK 4

3.3 Resultados 5

4 Modelagem utilizando GRASP **7**

4.1 Primeira Abordagem 7

4.1.1 Construção Gulosa Randomizada 7

4.2 Busca Local 8

4.2.1 Resultados e Limitações 8

4.3 Segunda Abordagem 9

4.3.1 Resultados 9

5 Conclusão **11**

Capítulo 1

Introdução

O trabalho final da disciplina de Otimização Combinatória consiste na escolha de um problema de otimização e de uma meta-heurística, e o objetivo é implementar uma solução para aquele utilizando-se deste. O problema escolhido foi o Balanceamento de linhas de produção (5), e a meta-heurística a GRASP (4).

GRASP é um acrônimo para *Greedy Randomized Adaptive Search Procedure*, cujo nome já especifica muito bem a técnica utilizada. Tipicamente em problemas de otimização combinatória o número de soluções é exponencial e inviabiliza a busca exaustiva. Uma estratégia válida é montar soluções de maneira gulosa e esperar que elas sejam boas o suficiente, mas o é um método propício a ficar preso em mínimos locais. Já uma abordagem puramente aleatória explora rapidamente várias soluções possíveis, mas não é melhor do que uma busca cega no mar de soluções. Além disso dada uma solução inicial gerada à partir de algum método é possível explorar a sua vizinhança à procura de soluções melhores.

O GRASP é um compromisso entre a abordagem gulosa cega e a randômica bruta e visa combiná-las num passo de construção gulosa aleatória de uma solução inicial. Para cada solução criada é feita uma Busca Local (*best improvement* ou *first improvement*, dependendo da implementação), e o processo é repetido por até que se chegue à condição de parada. Esta pode ser dada em número de iterações, número de iterações sem melhora, tempo máximo, etc ...

Como objetivos adicionais do trabalho estão a formulação do problema como um problema de programação inteira/linear, e a execução das soluções desenvolvidas (GLPK e GRASP) sobre um conjunto de casos de teste providos pelo professor.

Capítulo 2

Definição do Problema

Tirada diretamente da definição do trabalho.

Uma linha de produção consiste em uma série de estações de trabalho. Dado um conjunto de tarefas com restrições de precedência, temos que atribuir as tarefas às estações, tal que a precedência é respeitada. Cada tarefa possui um tempo de execução. O tempo total das tarefas de uma estação define a sua *carga*, e a carga máxima entre todas as estações define o *tempo de ciclo*. Formalmente

Instância Um grafo G = (T,P) direcionado acíclico sobre um conjunto de tarefas T, o tempo de execução t_i de cada tarefa $i \in T$, e um número m de estações de trabalho.

Solução Uma atribuição $s : T \to [m]$ das tarefas às estações que satisfaz as restrições de precedência, i.e. para cada $tu \in P$, $s(t) \leq s(u)$.

Objetivo Minimizar o tempo de ciclo $max_{i \in [m]} \sum_{j \in T | s(t)=i} t_j$.

Capítulo 3

Programação inteira

3.1 Formulação inteira

Baseada em (1) e (2).

Seja T o conjunto de tarefas e M o conjunto de máquinas.x_{ij} é a variável que indica se a tarefa i foi atribuída à máquina j.

min	T	
s.a	$\sum_{i=1}^{n} t_i \cdot x_{ij} \leq T$	$\forall j \in M$
	$\sum_{j=1}^{m} x_{ij} = 1$	$\forall i \in T$
	$\sum_{j=1}^{m} j \cdot x_{uj} \leq \sum_{j=1}^{m} j \cdot x_{vj}$	$\forall (u,v) \mid PRECEDE[u,v] = 1$
	$x_{ij} \in B$	$\forall i \in T, \forall j \in M$

3.2 Solução em GLPK

A parte dos dados é gerada por uma função do programa principal. Chamando-o com uma flag específica, o programa não executará o GRASP, mas apenas se utilizará das funções de leitura dos casos de teste para escrever os programas para o solver.

Segue listagem do código em GNU MathProg que modela o programa, além de um exemplo de área de dados, gerado para o caso de teste MERTENS.IN2.

```
param M;
param N;
param costs{1..N} >= 0;
param prec{1..N, 1..N} >=0;

var cicletime, integer;
var delegate{1..N,1..M} >= 0, binary;
```

```
minimize minCicleTime:
  cicletime;

subject to cicleTimeLimit {j in 1..M}:
  (sum {i in 1..N} costs[i]*delegate[i,j]) <= cicletime;

subject to delegateAll {i in 1..N}:
  (sum {j in 1..M} delegate[i,j]) = 1;

subject to respectPrecedences {u in 1..N, v in 1..N}:
  if prec[u,v] == 1 then
    (sum {j in 1..M} j*delegate[u,j]) <= (sum {j in 1..M} j*delegate[v,j
        ]);\n\n");

data;
param M := 5;
param N := 7;

param costs :=
1 1
2 5
3 4
4 3
5 5
6 6
7 5;
param prec :
1 2 3 4 5 6 7 :=
1 0 0 0 0 0 0 0
2 1 0 0 0 0 0 0
3 0 1 0 0 0 0 0
4 1 0 0 0 0 0 0
5 0 1 0 0 0 0 0
6 0 0 0 0 1 0 0
7 0 0 0 1 0 0 0 ;
end;
```

3.3 Resultados

Instância	m	Tempo de execução	Solução Ótima	Solução Obtida
Arcus2	27	1800*	5689	52039
Arcus2	9	1800*	16711	32649
Bathol2	51	1800*	84	1527
Bathol2	27	1800*	157	1211
Warnecke	10	847	155	155
Warnecke	20	1800*	79	349
Scholl	50	1800*	1394	57552
Scholl	38	1800*	1834	0
Scholl	25	1800*	2787	32326
Wee-Mag	20	1800*	77	86
Wee-Mag	30	1800*	56	87

O testes marcados com * foram interrompidos por alcançarem o limite de

tempo imposto.

Capítulo 4

Modelagem utilizando GRASP

O grupo tentou duas abordagens para a resolver o problema. A primeira se mostrou muito "ingênua"e possui um tempo de execução e qualidade ruins.

4.1 Primeira Abordagem

4.1.1 Construção Gulosa Randomizada

Para cada tarefa do vetor de tarefas o algoritmo tenta colocá-la, se isto respeitar o grafo de precedência, em alguma máquina de maneira que o maior ciclo entre todas as máquinas seja o menor possível. Cada um desses resultados (o valor de um maior ciclo e a máquina que gera essa solução) é colocado de maneira ordenada em uma Restricted Candidate List (RCL) de tamanho n (parametrizado na execução do código) de tal sorte que se for adicionado um elemento quando ela estiver cheia o último seja descartado. Após construir a RCL retiramos um dos elementos de maneira aleatória e esta será a máquina na qual deveremos inserir a tarefa atual.

O processo é repetido para cada uma das tarefas e quando ele acaba teremos uma atribuição de tarefas i para cada uma das j máquinas.

4.2 Busca Local

Após gerada a solução incial, o algoritmo parte para a busca local. O objetivo da busca local é "caminhar" na vizinhança da solução, tentando sempre melhorá-la, até não conseguir mais. Nossa vizinhança é: sabendo que a máquina $m*$ é a máquina que determina o *tempo de ciclo*, tentamos retirar uma das tarefas $t*$ atribuida a essa máquina e, para cada outra máquina m, se for possível atribuir $t*$ à m, isto é feito. Utilizamos *first improvement* na nossa busca local, fazendo com que o primero vizinho que melhore a solução seja escolhido como próxima solução.

Há, porém, o problema de existir mais que uma máquina $m*$, definidora do *tempo de ciclo*. Tendo isto em vista, durante a busca local, é considerado que o vizinho melhorou a solução se o *tempo de ciclo* daquele vizinho for menor ou se o *tempo de ciclo* do vizinho for igual mas a quantidade de máquinas $m*$, que definem este tempo, for menor. Isto evita que a busca local entre em *loop*, mas não impede que eventualmente ela melhore as *cargas* de todas máquinas $m*$, melhorando o *tempo de ciclo* global.

4.2.1 Resultados e Limitações

Esta abordagem, principalmente a etapa de construção gulosa, não se mostrou muito efetiva e os resultados deixaram a desejar. A seguir uma lista com os resultados:

Todos resultados foram obtidos utilizando como limite 10000 iterações. Alguns deles demoraram muito tempo para executar e foram ignorados, sendo o tempo denotado com um símbolo -".

Instância	**m**	**Solução Ótima**	**Solução Obtida**	% optimalidade
Arcus2	27	5689	8023	70.90
Arcus2	9	16711	18171	91.96
Bathol2	51	84	87	96.55
Bathol2	27	157	168	93.45
Warnecke	10	155	163	95.09
Warnecke	20	79	111	71.17
Scholl	50	1394	8082	17.24
Scholl	38	1834	-	-
Scholl	25	2787	-	-
Wee-Mag	20	77	80	96.25
Wee-Mag	30	56	58	96.55

O grupo acredita que a ineficácia do algoritmo se deva a uma tendenciosidade do processo de construção de uma solução. Uma solução em que uma tarefa que denota muitas dependências fica designada para uma máquina i, tal

que i é muito próximo de n, acaba sendo bastante ruim. Nesses casos a busca local não conseguiu explorar o espaço de soluções e acabou com uma solução muito aquém do desejado.

4.3 Segunda Abordagem

Visando melhorar os resultados o grupo pensou em uma segunda abordagem para a etapa de construção gulosa e esta, talvez devido à natureza do problema, de longe se mostrou mais bem sucedida, em termos de eficiência computacional (neste caso medindo somente tempo de execução) e qualidade da resposta.

Primeiramente as tarefas são ordenadas topologicamente (3). Um problema aqui é que a ordenação é deterministica e gera sempre o mesmo resultado, e isso é indesejado. Por isso, sempre que temos mais de uma tarefa possível para ir na posição i do vetor ordenado, escolhemos randomicamente entre todas.

Tendo o vetor ordenado, calculamos um limite inferior para a resposta. Isto é feito escolhendo o valor máximo entre $\frac{\sum_{i \in T} t_i}{m}$ e $max_{i \in T} t_i$. Sendo o primeiro o valor da distribuição mais uniforme possível, ou seja, soma dos custos de todas tarefas e dividido pelo número de máquinas e o segundo o custo da tarefa mais custosa.

Sabendo esse limite inferior, procedemos: para cada máquina, coloca-se nela uma das tarefas que em ordem topológica pode ser atribuída. Isto é feito enquanto a *carga* da máquina for menor que o limite inferior calculado. Quando este valor for extrapolado, é preciso decidir se iremos adicionar esta tarefa a mais (extrapolando o limiar) ou se vamos manter como está (ficando abaixo do limiar). Sempre é tomada a decisão que mais aproxima a *carga* da máquina do limiar calculado.

Em resumo, o que ele faz é tentar aproximar a *carga* de cada máquina do limite inferior calculado. Como isso não garante a atribuição de todas tarefas, isso é forçado, no fim, colocando todas tarefas pendentes na última máquina.

A etapa de busca local permanece igual à anterior.

4.3.1 Resultados

Os resultados foram bastante satisfatórios, conseguindo para alguns casos um valor muito próximo da otimalidade, e para um o valor ótimo.

Todos resultados foram obtidos utilizando como limite 10000 iterações.

Instância	**m**	**Solução Ótima**	**Solução Obtida**	% optimalidade
Arcus2	27	5689	5835	97.49
Arcus2	9	16711	16724	99.92
Bathol2	51	84	89	94.38
Bathol2	27	157	160	98.12
Warnecke	10	155	159	97.48
Warnecke	20	79	81	97.53
Scholl	50	1394	1449	96.20
Scholl	38	1834	1883	97.39
Scholl	25	2787	2825	98.30
Wee-Mag	20	77	78	98.71
Wee-Mag	30	56	56	1

Capítulo 5

Conclusão

Os resultados obtidos usando a segunda abordagem se mostraram bastante satisfatórios, tendo em média uma porcentagem de optimalidade de 94.79. O GRASP se mostrou como uma heurística muito boa, não só pelos bons resultados, mas também pela facilidade em se trabalhar.

De fato, de um modo geral, é bastante fácil entender e implementar o GRASP: os conceitos envolvidos são bastante simples e ele é composto apenas de duas funções principais (construção gulosa e busca local) além de um parâmetro ajustável de randomicidade. A dificuldade que o grupo teve foi a parte dependente do problema sendo tratado, foi difícil formular uma vizinhança e pensar em um algoritmo guloso que gerasse uma solução inicial válida.

Referências Bibliográficas

1 Evaluating Optimization Models to Solve SALBP* Rafael Pastor; Laia Ferrer; Alberto García

2 O uso da programação inteira 0-1 para o balanceamento de linhas de montagem: modelagem, estudos de caso e avaliação Flávio César Faria Fernandes; Moacir Godinho Filho; Ricardo Augusto Cutigi; Aline Malerbo Guiguet

3 Steven S. Skiena and Miguel Revilla = Programming Challenges

4 `http://www.optimization-online.org/DB_FILE/2001/09/371.pdf` Greedy randomized adaptive search procedures RESENDE,Mauricio G.C. RIBEIRO,Celso C.

5 `http://www.assembly-line-balancing.de/index.php?content=classview&content2=classview&content3=classviewdlfree&content4=classview&classID=28&type=dl` *Homepage for Assembly Line Optimization Research (Assembly Line Balancing and Sequencing)*