Flow-shop scheduling: uma abordagem com heurísticas e busca local

Aran Leite de Gusmão[1]*, Lucas Gomes Dantas[1]*

Resumo

Na disciplina de Análise e Projeto de Algorítmos, estudamos alguns métodos para solucionar problemas de otimização comuns, aplicáveis no mundo real. O projeto final foi desenvolver um algoritmo guloso, juntamente de uma busca local e uma meta-heurística, para resolver um problema derivado do Flow-shop scheduling.

Palavras-chave

Otimização, Algoritmos, Busca local, Heurística construtiva, Meta-heurística

[1] *Centro de Informática, Universidade Federal da Paraíba, João Pessoa, Brasil*

*__Contato__: hyoretsu, danta83.cc@gmail.com

Sumário

1. Introdução

O problema se resume a uma lista de produtos que devem ser produzidos uma única vez, dispostos em X linhas de produção. Cada produto tem um tempo Y de produção, e após um produto ser fabricado, há um tempo Z de limpeza para fabricar outro produto nessa mesma linha. A partir disso, pode ser construído um grafo com os produtos sendo as vértices e o tempo Y+Z como os pesos das arestas.

Os algoritmos gulosos que tratam de grafos ensinados em sala foram Prim e Dijkstra; o último serve para o problema do caminho mínimo, que é o que mais se assemelha ao nosso.

2. Métodos

Foram utilizados o algoritmo de Dijkstra com algumas modificações para dar a solução inicial do problema, o algoritmo Variable Neighborhood Descent (VND) para refinar a solução e o algoritmo Iterated Local Search (ILS) para escapar de soluções ótimas locais. Iremos explicar brevemente no que consiste cada um deles.

O algoritmo de Dijkstra original consiste em escolher, a cada passo, a aresta que tem o menor custo. Esse comportamento dele garante uma solução inicial bastante boa; porém, isso é apenas verdade quando temos uma única linha de produção. A partir do momento em que adicionamos linhas paralelas, ele vai perdendo eficiência. Outro ponto fraco desse algoritmo é que ele não leva em conta seu histórico, portanto uma solução pode ter um custo maior no início e um custo total menor no final.

Mas, diferente do algoritmo original de Dijkstra, o nosso não soma o custo atual ao total, mas sim o sobrescreve, e o custo é atualizado a cada iteração, não somente quando o custo novo é maior que o atual. Apesar de parecerem estranhas, essas modificações são necessárias para o problema dado e a nossa implementação das estruturas de dados e do algoritmo.

Por causa disso é aconselhável que usemos um algoritmo auxiliar de busca local, que vai basicamente modificar essa solução conscientemente atrás de uma solução melhor. Seu objetivo é pegar uma solução, aplicar movimentos de vizinhança a ela e checar se a nova solução é melhor do que atual. No nosso caso, a condição de parada é até termos tentativas falhas **consecutivas** iguais a 10 vezes o número de linhas de produção. O algoritmo de busca local usado foi o VND, e os movimentos de vizinhança foram swap aleatório e reverse para produtos de uma mesma linha e um swap entre produtos de linhas diferentes.

Porém, mesmo com um movimento de vizinhança inter-linha, essa abordagem pode ficar presa em um ótimo local e nunca alcançar o ótimo global. Para remediar isso, usa-se mais uma técnica, dessa vez uma meta-heurística. A meta-heurística escolhida para o projeto foi o ILS, que consiste em pegar uma solução inicial que foi gerada através de um algoritmo guloso e passou por uma busca local, aplicar uma perturbação a ela, e independentemente se houve benefício ou não, aplicar uma nova busca local nesse novo resultado.

3. Desenvolvimento

Estes foram os resultados obtidos para as instâncias de teste fornecidas. O tempo em todas as tabelas corresponde à média do tempo gasto nas 10 iterações. Para motivos de demonstração do potencial da implementação, foram incluídos dois grupos de tabelas: um para o melhor resultado encontrado, e outro para a média dos resultados de 10 iterações como pedido.

Instância	Solução referência	Solução encontrada	Tempo	GAP
n10m2_A	763	1186	112 μs	55,44%
n10m2_B	696	1117	109 μs	60,49%
n15m3_A	1006	1678	189 μs	66,80%
n15m3_B	1055	1714	193 μs	62,46%
n15m4_A	688	1128	219 μs	63,95%
n15m4_B	784	951	249 μs	21,30%
n29m4_A	1459	2235	591 μs	53,19%
n29m4_B	1527	2280	523 μs	49,31%
n29m6_A	1101	1787	735 μs	62,30%
n29m6_B	1043	1531	535 μs	46,79%
n40m5_A	447	760	665 μs	70,02%
n40m5_B	581	1026	508 μs	76,59%
n52m5_A	7070	12735	847 μs	80,13%
n52m5_B	6924	11962	805 μs	72,76%
n450m16_A	4750	12725	130 ms	167,90%
n500m10_A	5608	13937	184 ms	148,52%

Tabela 1. Resultados da melhor execução (algoritmo guloso)

Os resultados preliminares obtidos apenas do algoritmo guloso não são nem um pouco satisfatórias, mas esse não é o propósito dele. Ele serve apenas para obter uma solução inicial, e nisso ele foi bem-sucedido.

Instância	Solução referência	Solução encontrada	Tempo	GAP
n10m2_A	763	768	436 μs	0,66%
n10m2_B	696	718	363 μs	3,16%
n15m3_A	1006	1060	705 μs	5,37%
n15m3_B	1055	1129	829 μs	7,01%
n15m4_A	688	739	1014 μs	7,41%
n15m4_B	784	823	969 μs	4,97%
n29m4_A	1459	1727	1575 μs	18,37%
n29m4_B	1527	1759	1801 μs	15,19%
n29m6_A	1101	1051	3917 μs	-4,54%
n29m6_B	1043	1158	2536 μs	11,03%
n40m5_A	447	565	1766 μs	26,40%
n40m5_B	581	725	1175 μs	24,78%
n52m5_A	7070	8394	1901 μs	18,73%
n52m5_B	6924	9062	1209 μs	30,88%
n450m16_A	4750	5587	540 ms	17,62%
n500m10_A	5608	8215	550 ms	46,49%

Tabela 2. Resultados da melhor execução (busca local)

Já é possível notar uma melhora significativa nos resultados, com alguns até se aproximando da solução ótima ainda nesse segundo passo.

Instância	Solução referência	Solução encontrada	Tempo	GAP
n10m2_A	763	763	9 *ms*	0%
n10m2_B	696	696	10 *ms*	0%
n15m3_A	1006	1020	24 *ms*	1,39%
n15m3_B	1055	1059	28 *ms*	0,38%
n15m4_A	688	716	40 *ms*	4,07%
n15m4_B	784	784	34 *ms*	0%
n29m4_A	1459	1547	75 *ms*	6,03%
n29m4_B	1527	1670	92 *ms*	9,36%
n29m6_A	1101	1034	268 *ms*	-6,09%
n29m6_B	1043	1065	174 *ms*	2,11%
n40m5_A	447	459	146 *ms*	2,68%
n40m5_B	581	593	147 *ms*	2,07%
n52m5_A	7070	7485	207 *ms*	5,87%
n52m5_B	6924	7665	296 *ms*	10,70%
n450m16_A	4750	5579	18 *s*	17,45%
n500m10_A	5608	7872	54 *s*	40,37%

Tabela 3. Resultados da melhor execução (meta-heurística)

A meta-heurística serviu para refinar ainda mais os resultados. Foi possível notar uma melhora ainda maior em todas as soluções. Os únicos que não sofreram muita alteração foram os maiores, com 450 e 500 produtos.

Como o algoritmo guloso implementado é determinístico, omitiremos sua tabela de médias.

Instância	Solução referência	Solução encontrada	Tempo	GAP
n10m2_A	763	808	436 μs	5,90%
n10m2_B	696	753	363 μs	8,19%
n15m3_A	1006	1124	705 μs	11,73%
n15m3_B	1055	1168	829 μs	10,71%
n15m4_A	688	795	1014 μs	15,55%
n15m4_B	784	862	969 μs	9,95%
n29m4_A	1459	1862	1575 μs	27,62%
n29m4_B	1527	1865	1801 μs	22,13%
n29m6_A	1101	1187	3917 μs	7,81%
n29m6_B	1043	1288	2536 μs	23,49%
n40m5_A	447	626	1766 μs	40,04%
n40m5_B	581	760	1175 μs	30,81%
n52m5_A	7070	9874	1901 μs	39,66%
n52m5_B	6924	11001	1209 μs	58,88%
n450m16_A	4750	6284	540 *ms*	32,29%
n500m10_A	5608	9327	550 *ms*	66,32%

Tabela 4. Média de resultados em 10 iterações (busca local)

Aqui, o resultado apenas com o VND na prática, e não apenas no melhor caso, foi um pouco pior. Porém, mesmo assim está dentro do esperado, já que ele é um algoritmo de busca local e não necessariamente procura por ótimos globais.

Instância	Solução referência	Solução encontrada	Tempo	GAP
n10m2_A	763	770	9 *ms*	0,92%
n10m2_B	696	696	10 *ms*	0%
n15m3_A	1006	1045	24 *ms*	3,88%
n15m3_B	1055	1098	28 *ms*	4,08%
n15m4_A	688	741	40 *ms*	7,70%
n15m4_B	784	800	34 *ms*	2,04%
n29m4_A	1459	1614	75 *ms*	10,62%
n29m4_B	1527	1721	92 *ms*	12,70%
n29m6_A	1101	1058	268 *ms*	-3,91%
n29m6_B	1043	1097	174 *ms*	5,18%
n40m5_A	447	491	146 *ms*	9,84%
n40m5_B	581	627	147 *ms*	7,92%
n52m5_A	7070	7678	207 *ms*	8,60%
n52m5_B	6924	7852	296 *ms*	13,40%
n450m16_A	4750	5624	18 *s*	18,40%
n500m10_A	5608	7923	54 *s*	41,28%

Tabela 5. Média de resultados em 10 iterações (meta-heurística)

Agora, levando em consideração a tabela de médias, é possível notar uma melhora significativa em todos os resultados.

4. Conclusão

É notável a deficiência dessa combinação/implementação de algoritmos para entradas maiores. Uma possível forma de solucionar isso seria combinar uma maior diversidade de heurísticas, uma vez que já são feitas inúmeras iterações tanto na busca local quanto na meta-heurística escolhidas, então aumentar ainda mais só iria gastar mais tempo. Também houve uma dificuldade da implementação para achar resultados melhores nas instâncias em que a divisão entre produto e linhas é desigual.

5. Referências

[1] Slides do prof. Bruno Bruck