Escalada do Monte Aplicada ao Problema Flowshop de Permutação

Miguel Ângelo Teixeira
Lic. Informática, Univ. Aberta, Portugal
etmat@sapo.pt

José Silva Coelho
DCET, Univ. Aberta, Portugal
jcoelho@univ-ab.pt

Resumo

Este artigo apresenta uma meta-heurística baseada na escalada do monte para o problema "flowshop" de permutação (PFSP). O algoritmo é descrito e são propostas cinco afinações, que foram testadas e comparadas entre si e com o algoritmo proposto. É feito um conjunto de testes, onde se mostra que sem essas afinações os resultados pioram, e que a qualidade dos resultados desta abordagem está ao nível dos algoritmos de referência.

palavras-chave: meta-heurística, *flowshop*, escalada do monte

Abstract

This paper presents a meta-heuristic to the permutation flowshop problem (PFSP) based on hill climbing. The algorithm is described and five adjustments proposed, which were tested and compared with each other and with the proposed algorithm. A set of tests is made, which shows that without these adjustments the results get worse, and that the quality of the results of this approach is at the level of the algorithms of reference.

keywords: meta-heuristic, flowshop, hill-climbing

1. Introdução

Os problemas de calendarização são actualmente uma área muito ativa de investigação e uma atividade de grande relevância devido à sua aplicabilidade na produção industrial, gestão de projectos, gestão de recursos, planeamento de horários. Têm sido desenvolvidas técnicas modernas e ferramentas computorizadas para o uso nestes problemas. Apesar dos progressos, os problemas continuam a ser complexos e ainda é possível melhorar os algoritmos que actualmente estão disponíveis.

O problema flowshop de permutação (PFSP) é um dos vários tipos de problemas de calendarização. Este problema é considerado NP-hard [Rinnooy kan,1976] e para um conjunto de n tarefas existem $n!$ soluções possíveis. Para lidar com espaços de soluções desta dimensão é necessário recorrer a heurísticas ou meta-heurísticas para obter aproximações da solução óptima.

A utilização da escalada do monte neste tipo de problema parece ter sido pouco explorada na literatura, no entanto, a sua simplicidade serviu de motivação para desenvolver um algoritmo que seja baseado na escalada do monte. O algoritmo é descrito e são propostas cinco afinações, que foram testadas e comparadas entre si e com o algoritmo proposto. É feito um conjunto de testes, onde se mostra que sem essas afinações os resultados pioram, e que a qualidade dos resultados desta abordagem está ao nível dos algoritmos de referência.

2. Descrição do problema

O problema flowshop de permutação consiste em n tarefas que têm que ser processadas por uma sequência fixa de m máquinas. O tempo do processamento da tarefa i na máquina j é dado por t_{ij} onde $i \in \{1,\ldots,n\}$ e $j \in \{1,\ldots,m\}$, esses tempos são fixos, não negativos e se zero significa que a tarefa não é processada nessa máquina.

Uma tarefa não pode ser processada em simultâneo por mais que uma máquina, nem uma máquina pode processar mais que uma tarefa em simultâneo, no entanto, diferentes máquinas podem processar diferentes tarefas deste que todas as máquinas processem as tarefas pela mesma ordem.

O objetivo é minimizar o tempo total do processamento das tarefas (o tempo gasto entre o início da primeira tarefa na primeira máquina até ao fim da última tarefa na última máquina).

Tarefas:

$$T = \{\ 1, 2, \ldots, n\ \}$$

Máquinas:

$$M = \{\ 1, 2, \ldots, m\}$$

Tempo de processamento da tarefa i, na máquina j:

$$t_{ij} : T \times M \ \rightarrow \ Z_0^+$$

Solução:

$S = \{ (s_1, \ldots, s_n) \in T^n : \{s_1, \ldots, s_n\} = T \}$

Tempo parcial de processamento (acumulado) da tarefa k, na máquina j:

$C_{kj} : T \times M \rightarrow Z_0^+$

$$C_{kj} = \begin{cases} t_{s_1 1} & , k = 1; j = 1 \\ C_{1(j-1)} + t_{s_1 j} & , k = 1; 2 \leq j \leq m \\ C_{(k-1)1} + t_{s_k 1} & , 2 \leq k \leq n;\ j = 1 \\ \max\{C_{(k-1)j},\ C_{k(j-1)}\} + t_{s_k j} & , 2 \leq k \leq n; 2 \leq j \leq m \end{cases}$$

O objectivo é encontrar uma solução s^* tal que $C_{nm}^* \leq C_{nm}\ \forall_{s \in S}$, onde C_{nm} é o tempo total de processamento da solução (*makespan*).

3. Escalada do monte

A escalada do monte é uma meta-heurística que, quando aplicada aos problemas, procura gananciosamente no espaço de soluções, executando apenas movimentos que melhoram a solução, sem considerar as consequências futuras desses movimentos, pois não verifica mais à frente se a opção tomada foi boa.

Algoritmo 1: Pseudo-código do Algoritmo Proposto.

```
procedimento EscaladaMonte
    melhorSoluçãoGlobal = +∞
    repetir
        S0 = GerarSoluçãoInicial()
        melhorSolução = S0
        S = PesquisaLocal(S0)
        repetir
            se Duração(S) inferior Duração(melhorSolução) então
                melhorSolução = S
                AdiarReínicio()
            se Duração(melhorSolução) inferior Duração(melhorSoluçãoGlobal) então
                melhorSoluçãoGlobal = melhorSolução
            se Duração(S) igual Duração(melhorSolução) então
                melhorSolução = S
            S' = Perturbar(melhorSolução)
            S = PesquisaLocal(S')
        até CondiçãoReinício Ou CondiçãoFim
    até CondiçãoFim
fim
```

A solução inicial é gerada ordenando as tarefas aleatoriamente. O objetivo da pesquisa local é para melhorar a solução inicial ou a melhor solução depois ter sido perturbada. A perturbação aplicada é mínima e é feita através de uma permutação entre duas tarefas ou de um deslocamento de uma tarefa para outra posição. Se a solução gerada pela pesquisa for melhor ou igual do que a melhor solução anterior, então esta é substituída pela nova. Este processo repete-se até que a condição de reinício seja atingida.

Em princípio, existem vários algoritmos que podem ser utilizados na pesquisa local, no entanto, o desempenho e a qualidade das soluções são condicionados pelo algoritmo escolhido.

A condição de reinício é a não existência de computação disponível para evoluir a melhor solução encontrada, a medida utilizada para a computação é a quantidade de soluções geradas e testadas (todas as soluções geradas são testadas). A condição de fim é atingida quando o limite máximo de computação reservado para a pesquisa for atingido, em alternativa este critério pode substituído por um limite de tempo.

O progresso ou a sua falta determina quando a pesquisa é reiniciada, pois se progride (consegue melhorar a solução) antes de consumir a computação disponível, então, o reinício é adiado em AdiarReínico(), disponibilizando mais computação. Caso contrário, acaba por reiniciar, descartando a melhor solução encontrada nesse ciclo.

3.1. Pesquisa local

A pesquisa local deve ser capaz de obter soluções de qualidade com um baixo custo computacional. O seu propósito é melhorar a solução que lhe seja fornecida.

Algoritmo 2: Pseudo-código da pesquisa local.

```
função PesquisaLocal  (solução S) retorna uma solução
    BaralharListaPertubações()
    melhorSoluçãoLocal = S
    Repetir
        se Duração(S) inferior Duração(melhorSoluçãoLocal) então
            melhorSoluçãoLocal = S
            AdiarFimLocal()
        se Duração(S) igual Duração(melhorSoluçãoLocal) então
            melhorSoluçãoLocal = S
        ActualizarMelhorSoluçãoLocal(S)
        S = Perturbar(melhorSoluçãoLocal)
    até CondiçãoFimLocal
    retornar melhorSoluçãoLocal
fim
```

Um algoritmo guloso é eficiente, mas facilmente fica preso em óptimos locais de pouca qualidade. Para melhorar a qualidade é necessário diversificar a procura, aceitando soluções com o mesmo tempo de processamento e alterando frequentemente a ordem das perturbações a que estas são sujeitas. BaralharListaPertubações() assume que as perturbações estão colocadas numa lista, sendo essa lista baralhada.

Verificou-se que aceitar soluções com o mesmo tempo de processamento do que a melhor solução encontrada diversifica substancialmente a pesquisa e não trava o *momentum* guloso da procura.

Se a ordem das perturbações for alterada aleatoriamente, então, uma determinada solução que seja encontrada mais do que uma vez, provavelmente, vai evoluir para uma solução distinta das outras. Como o algoritmo aceita o primeiro movimento

(perturbação) que melhore a solução ou com tempo total de processamento igual, se existir mais que um movimento aceitável na vizinhança mas com uma ordem diferente da anterior, então, a procura vai seguir por outro caminho, diversificando a geração de soluções.

Quando o tempo de processamento da nova solução for igual ou menor à melhor solução local, então, esta é actualizada, se for menor, a pesquisa é premiada com computação.

AdiarFimLocal() disponibiliza computação suficiente para voltar a testar novamente todas as perturbações. Contudo, se for igual, não recebe mais computação. No entanto, se toda a computação for consumida (CondiçãoFimLocal) sem que a última melhor solução local tenha sido melhorada, a pesquisa termina.

3.2. Qualidade da solução

Sem outra informação é razoável assumir que quanto menor for a duração (tempo total de processamento), melhor é a solução. É também mais provável obter ou aproximar-se da solução óptima explorando a vizinhança a partir de uma solução melhor ao invés de uma inferior, e é por esta razão que a melhor solução encontrada é sucessivamente reutilizada para encontrar novas soluções.

Existem outros critérios para avaliar a qualidade de uma solução, contudo, o tempo total de processamento foi o único critério utilizado.

3.3. Óptimo local

Quando, a partir da vizinhança, já não é possível melhorar a solução, significa que estamos diante de um óptimo local.

Revelou-se computacionalmente eficiente assumir que a falta de progresso é um óptimo local, pois a sua confirmação tem um custo computacional elevado e um benefício baixo. Este critério é utilizado na pesquisa local, ao aceitar soluções com o mesmo tempo de processamento, mas sem premiar com mais computação, e também para decidir quando o algoritmo deve reiniciar.

No entanto, deve-se tentar escapar ao óptimo local antes de tomar a decisão de reiniciar, pois o esforço computacional gasto a evoluir a solução não deve ser descartado antes de ter alguma certeza que já não é possível evoluir mais.

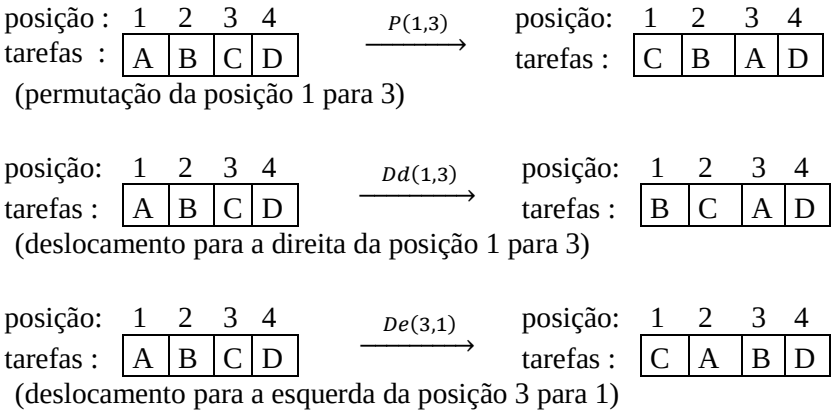
O óptimo local (alegado) que é considerado é sempre a melhor solução encontrada a partir do último reinício, ao invés da solução encontrada pela última pesquisa local.

3.4. Reinício

Reiniciar a pesquisa serve para escapar dos óptimos locais em que fica presa. Sempre que a solução é melhorada, a pesquisa é premiada com mais computação disponível. A quota de computação a ser atribuída é igual à computação gasta desde o último reinício, mais um valor fixo parametrizável. A otimalidade do valor a ser utilizado depende da instância. Quando toda a computação disponibilizada é consumida, a pesquisa é reiniciada (condição de reinício).

3.5. Vizinhança

A exploração de vizinhança de uma determinada solução é realizada por meio de perturbações a que esta é sujeita. A perturbação pode ser uma permutação de duas tarefas (troca de uma tarefa numa determinada posição por outra noutra posição) ou um deslocamento de uma tarefa para outra posição (a tarefa é removida da sua posição e é inserida noutra posição). As perturbações são exemplificadas no exemplo 1.



Exemplo 1: Exemplos de perturbações.

Existem C_2^n permutações de duas tarefas, C_2^n deslocamentos para a direita e C_2^n deslocamentos para a esquerda. A união do conjunto das permutações com os conjuntos dos deslocamentos totalizam $3C_2^n - 2(n-1)$ perturbações possíveis (existe contagem triplicada na troca das tarefas adjacentes, é por isso que é necessário subtrair $2(n-1)$).

No algoritmo 2, por defeito a CondiçãoFimLocal são $3C_2^n - 2(n-1)$, soluções geradas até que estas sejam todas testadas (sempre que uma solução é testada a CondiçãoFimLocal é decrementada), e se a melhorSoluçãoLocal for melhorada, então, AdiarFimLocal() coloca novamente a CondiçãoFimLocal em $3C_2^n - 2(n-1)$.

Todas as perturbações são colocadas numa lista para depois serem sequencialmente aplicadas à melhor solução (Perturbar(), foi utilizado nos algoritmos 1 e 2, aplica uma perturbação), no entanto, a ordem da lista é sempre alterada aleatoriamente no início da pesquisa local (BaralharListaPertubações(), no algoritmo 2).

3.6 Pontos-chave do algoritmo

O algoritmo proposto, resume-se a cinco afinações aplicadas à escalada do monte:

1. A vizinhança é expressa por permutações de duas tarefas e deslocamentos (pela direita e esquerda) de uma tarefa.
2. Reutilizar a melhor solução.
3. Aceitar soluções com o mesmo tempo total de processamento, relativamente à melhor solução encontrada.
4. Baralhar frequentemente a lista de perturbações
5. Quando a procura estiver estagnada, reiniciar.

Estes pontos-chave não são exclusivos do flowshop, e, em princípio, podem ser aplicados a outros problemas.

4. Testes Computacionais

Foram realizados sete testes utilizando as instâncias de [Taillard 2012], seis dos quais estão relacionados com os pontos-chave do algoritmo, existem dois testes sobre o ponto 1, (um com permutações mas sem deslocamentos e vice-versa), e um para cada um dos restantes pontos, em cada desses testes o algoritmo proposto é executado sem uma parte essencial.

A média do desempenho dos resultados de cada grupo de instâncias (50x20, 100x20 e 200x20) está em percentagem, a medida de desempenho é dada pela fórmula $100 \times \frac{(x-MSP)}{MSP}$, onde MSP é o tempo total de processamento da melhor solução publicada.

Os testes foram realizados no Windows 7, no processador Intel Dual-Core à frequência de 3.16 GHz, cada corrida de cada uma das instâncias 50x20, 100x20, 200x20, sendo a condição de paragem (fim) 120 milhões de soluções geradas (aproximadamente 11, 21, 35 minutos), e foram executadas 5 corridas por cada instância. A computação acrescentada para adiar o reinício (AdiarReínicio(), algoritmo 1) é igual ao número de soluções geradas desde o último reinício mais um milhão. Este critério foi utilizado para todas as instâncias. Os resultados de cada instância são a média dos tempos totais de processamento das 5 corridas.

A tabela 1 também inclui os resultados de [Reeves, Yamada 1998] um artigo de referência nesta área, e na última coluna estão os melhores tempos das melhores soluções publicadas em [Taillard 2012] para cada instância.

5. Conclusão

De acordo com a tabela 1, os resultados foram piores quando a melhor solução não foi reutilizada, o que sugere que reutilizar a melhor solução é a afinação mais importante. A vizinhança constituída só com deslocamentos obteve melhores resultados do que só com permutações, no entanto, com a utilização de ambas os resultados são superiores. Não reiniciar também obteve bons resultados, mas por vezes a procura fica presa num óptimo local de pouca qualidade, resultando em soluções menos boas.

Se a escalada do monte for aplicada sem afinações, as soluções encontradas provavelmente ficaram aquém, no entanto, o algoritmo proposto encontra soluções muito boas, inclusive são na generalidade melhores do que as de Reeves & Yamada [1998], e estão em média a menos de 1% de distância dos tempos das melhores soluções publicadas [Taillard 2012] para todos os grupos de instâncias.

Apesar da simplicidade desta abordagem, os resultados experimentais indicam que é possível obter soluções de qualidade semelhante à dos algoritmos de referência nesta área.

50 x 20	Só permu-tações	Só desloca-mentos	Melhor solução não reutilizada	Solução de duração igual não aceite	Lista de perturbações não baralhada	Sem reinicio	Algoritmo proposto (completo)	Colin Reeves & Taleshi Yamada	Melhores soluções publicadas
1	3888	3877	3906	3886	3882	3882	**3872**	3880	3850
2	3723	3716	3755	3717	3718	3715	**3711**	3716	3704
3	3683	3665	3711	3674	3671	3669	**3661**	3668	3640
4	3756	**3744**	3776	3752	3751	3745	3746	**3744**	3723
5	3642	**3626**	3674	3638	3639	3635	3628	3636	3611
6	3711	3703	3739	3714	3707	**3698**	3700	3701	3681
7	3734	3726	3768	3730	3735	3729	3725	**3723**	3704
8	3732	3721	3763	3726	3730	3721	**3716**	3721	3691
9	3783	**3764**	3809	3768	3773	3779	**3764**	3769	3743
10	3785	3774	3807	3776	3775	3779	**3766**	3772	3756
%	**0,90%**	**0,57%**	**1,63%**	**0,75%**	**0,75%**	**0,67%**	**0,50%**	**0,61%**	
100 x 20									
1	6301	6260	6332	6289	6300	6260	6261	**6259**	6202
2	6257	6237	6299	6264	6266	6232	**6230**	6234	6183
3	6332	6313	6380	6334	6342	6315	**6310**	6312	6271
4	6317	6309	6364	6327	6317	6323	6307	**6303**	6269
5	6388	6358	6422	6390	6389	6363	6360	**6354**	6314
6	6453	**6414**	6485	6453	6436	**6414**	6428	6417	6364
7	6364	6328	6394	6338	6345	6318	**6309**	6319	6268
8	6489	**6459**	6528	6474	6465	6464	6466	6466	6401
9	6355	6336	6386	6349	6326	6320	**6312**	6323	6275
10	6510	6474	6533	6514	6504	6482	**6471**	**6471**	6434
%	**1,25%**	**0,81%**	**1,81%**	**1,19%**	**1,13%**	**0,81%**	**0,75%**	**0,76%**	
200 x 20									
1	11297	11301	11347	11345	11350	**11279**	11292	11316	11195
2	11346	11319	11403	11367	11377	**11305**	11307	11346	11203
3	11452	11408	11501	11482	11492	11420	**11403**	11458	11281
4	11381	11362	11466	11421	11455	**11349**	11356	11400	11275
5	11334	**11317**	11384	11357	11350	**11317**	11319	11320	11259
6	11304	**11267**	11347	11360	11345	11289	11284	11288	11176
7	11450	11440	11498	11476	11487	11447	**11438**	11455	11360
8	11444	11418	11500	11493	11495	**11411**	11415	11426	11334
9	11353	11295	11394	11376	11358	11315	**11277**	11306	11192
10	11401	11368	11474	11445	11460	11369	**11363**	11409	11288
%	**1,07%**	**0,83%**	**1,56%**	**1,39%**	**1,43%**	**0,83%**	**0,79%**	**1,03%**	

Tabela 1: Média do tempo total de processamento das soluções encontradas pelos testes, nas instâncias de Taillard. Em percentagem estão as médias do desempenho dos resultados de cada grupo de instâncias comparativamente às melhores soluções publicadas.

REFERÊNCIAS

Colin R. Reeves, Takeshi Yamada, 1998: "Genetic Algorithms, Path Relinking, and the Flowshop Sequencing Problem." Evolutionary Computation 6(1): pp. 45-60.

Taillard's Scheduling Instances (on-line novembro 2012), http://mistic.heig-vd.ch/taillard/problemes.dir/ordonnancement.dir/ordonnancement.html

Rinnooy kan, A., 1976: Machine Scheduling Problems: Classification, Complexity and Computation. The Hague: Martinus Njhoff.

Apêndice

Encontrada uma nova solução para ta051 melhor do que a publicada [Taillard 2012] (50x20 - 1).
19 30 38 26 42 14 43 10 7 44 34 36 5 16 33 27 6 13 41 32 39 23 4 28 9 1 17 46 47 20 45 0 15 48 11 22 21 35 31 37 18 8 25 24 12 40 29 3 49 2
Duração: 3846

Miguel Teixeira foi técnico de informática (1999-2003) e programador de aplicações para a web (2003-2007) nos sistemas de informação da Segurança Social Regional da Madeira e no site da Câmara Municipal do Funchal. Licenciatura em Informática pela Universidade Aberta (2008-2013).

José Coelho é Doutor pelo Instituto Superior Técnico ("Modelo Genérico para Gestão de Projetos: SATPSP ", 2004), Mestre em Investigação Operacional e Engenharia de Sistemas, pelo Instituto Superior Técnico ("Análise de Problemas e Heurísticas para o "Resource Constrained Project Scheduling Problem " (RCPSP) 2001), Licenciado em Engenharia Informática e de Computadores, pelo Instituto Superior Técnico ("Optimização de trajetórias de pistas em placas de circuitos impressos", 1995). Os principais programas de software desenvolvidos são o RiskNet de análise de riscos e otimização para gestão de projetos (1999), o MacModel de apoio à decisão multicritério (2001), o DistributionView que permite gerar/simular qualquer distribuição (2003), e o EngineTester para efetuar testes empíricos (2006).