

An adaptive simulated annealing algorithm with variable
neighborhood search for optimization no-idle flow shop
scheduling problem

基于带变邻域搜索的自适应模拟退火算法解决零空闲
流水线调度优化问题

by Yunhe Wang





Introductions of the previous papers with different algorithms for the no-idle flow shop scheduling problem

P. Baptiste, K.H.A Lee. branch and bound algorithm for the $F|no-idle|C_{max}$, in: Proceedings of the International Conference on Industrial Engineering and Production Management, Lyon, 1997, pp. 429–438

Q.K. Pan, L. Wang, No-idle permutationflow shop scheduling based on a hybrid discrete particle swarm optimization algorithm, Int. J. Adv. Manuf.Technol. 39 (7 – 8) (2008) 796–807.The International Journal of Advanced Manufacturing Technology

Q.K. Pan, L. Wang, A novel differential evolution algorithm for no-idle permutationflow shop scheduling problems, Eur. J. Ind. Eng. 2 (3) (2008) 279–297.

D. Baraz, G. Mosheiov, A note on a greedy heuristic for the flowshop makespan minimization with no machine idle-time, Eur. J. Oper. Res. 184 (2) (2008) 810–813

W.J. Ren, Q.K. Pan, H.Y. Han, Tabu search algorithm for no-idle flow shop scheduling problems, Comput. Eng. Des. 31 (23) (2010) 5071–5074.



The basic idea of the simulated annealing algorithm

该算法可以分解为解空间、目标函数和初始解三部分。以目标最小化问题为例：

- (1) 初始化：初始温度 T (充分大)，初始解状态 S (是算法迭代的起始点)，每个 T 值的迭代次数 $time$ ；
- (2) 对 $k=1, \dots, time$ 做第(3)至第(6)步；
- (3) 产生新解 S^1 (局部搜索算法)；
- (4) 计算增量 $\Delta t = F(S^1) - F(S)$ ，其中 $F(S)$ 为目标函数的计算结果，即为适应度值；
- (5) 若 $\Delta t < 0$ ，表示新解较好，则接受 S^1 作为新的当前解，否则以一定概率接受 S^1 作为新的当前解；
- (6) 如果满足终止条件则输出当前解作为最优解，结束程序；
终止条件通常取为温度下降至某一较低温度或者连续若干个新解都没有被接受。
- (7) T 逐渐减少，且 $T^1(\text{下降后温度}) > \text{结束温度}$ ，则转向第2步。



Description of the no-idle flow shop scheduling problem

- 根据名字特点，这个调度问题可以描述为：

在一个生产系统中，有 n 个工作和 m 个机器，所有的工作在所有的机器上的处理顺序是相同的。假设每个工作的准备时间为0或者包含在处理时间中，要求每一个工作在某一时刻只能在一个机器上处理；一台机器某一刻只能处理一个工作；机器上的工作一旦开始，直到工作调度序列处理完毕，机器的运转就不可以停止。这就是所谓的零空闲，即每个机器上的操作处理工作阶段是连续的，不可以停止。

下面介绍算法中用到的一些重要变量的含义及表示方法：

- **变量1：一个完整工作调度序列** $\pi = \{\pi_1, \pi_2, \dots, \pi_n\} = \pi^{(p)}(n)$
- **变量2：一个部分工作调度序列** $\pi^{(p)}(i) = \{\pi_1, \pi_2, \dots, \pi_i\}$
- **变量3：受零空闲约束条件的限制，对于某一个工作调度序列 $\pi^{(p)}(i)$ ，它最后一个工作在机器 $J+1$ 和机器 J 上处理完成的最小时间差：** $E(\pi^{(p)}(i), j, j+1)$
- **变量4：工作 i 在机器 j 上的处理时间：** $t_{ij} (1 \leq i \leq n, 1 \leq j \leq m)$
- **变量5：可行解 π 的最大完成时间：** $C_{\max}(\pi)$
- **变量6：所有可行解即序列的集合：** Π
- **变量7：最优工作调度序列即有最小适应度的解：** π^*

了解这些变量后，使用它们通过相应公式可以计算出makespan即最大完成时间。其中 Π 为模拟退火算法中提到的解空间，目标函数为makespan，适应度值是makespan的大小。本问题是最小化最大完成时间，所以适应度值越小表示该解越优。



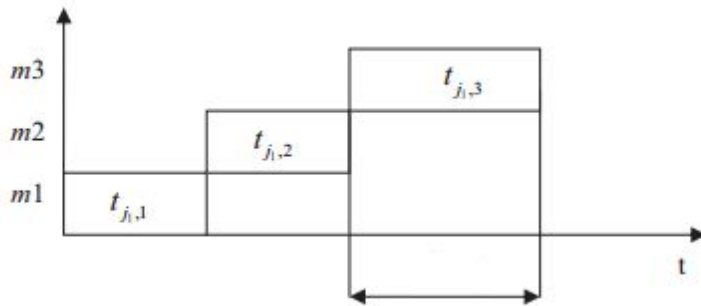
The way to calculate the makespan

- **公式(1)** : $E(\pi^{(p)}(1), j, j+1) = t_{j1, j+1} \quad j=1, 2, \dots, m$
- **公式(2)** : $E(\pi^{(p)}(2), j, j+1) = \max\{E(\pi^{(p)}(1), j, j+1) - t_{j2, j}, 0\} + t_{j2, j+1} \quad j=1, 2, \dots, m-1$
- **公式(3)** : $E(\pi^{(p)}(i), j, j+1) = \max\{E(\pi^{(p)}(i-1), j, j+1) - t_{ji, j}, 0\} + t_{ji, j+1} \quad i=2, 3, \dots, n; j=1, 2, \dots, m-1$.
- **公式(4)** : $C_{\max}(\pi) = \sum_{\bar{j}=1}^{m-1} E(\pi, \bar{j}, \bar{j}+1) + \sum_{\bar{i}=1}^n t_{\bar{j}, \bar{i}+1}$
- **公式(5)** : $\pi^* = \min(C_{\max}(\pi)) \quad , \pi \text{ is in } \Pi$

其中公式(4)表示零空闲流水线调度问题的目标函数，公式(5)表示最优解条件。其中n为工作的个数，m是机器的个数，其它变量为上一小节提出的变量。

- 下面用图1, 2, 3形象展示一下当 $n=2, m=3$, 工作调度序列为: $\pi = \{\pi_1, \pi_2\}$ 时, 如何计算出在零空闲流水线调度问题的不同条件下 $E(\pi^{(p)}(1), 2, 3)$, $E(\pi^{(p)}(2), 2, 3)$ 的大小。

图1



$E(\pi^{(p)}(1), 2, 3)$

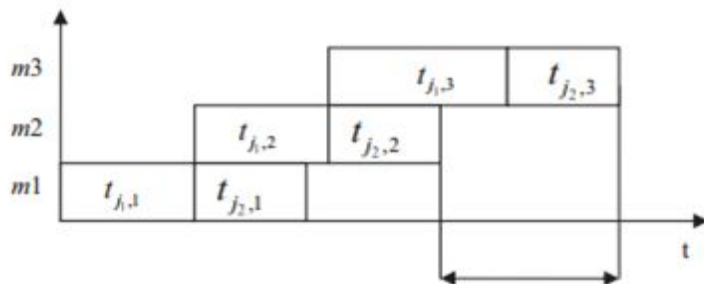
图3



$E(\pi^{(p)}(2), 2, 3)$

当 $E(\pi^{(p)}(1), 2, 3) < t_{j2,2}$ 时, 图3计算出 $E(\pi^{(p)}(2), 2, 3)$

图2



$E(\pi^{(p)}(2), 2, 3)$

最大完成时间makespan用公式求解为:

$$\text{makespan} = t_{j1,1} + t_{j2,1} + E(\pi^{(p)}(2), 1, 2) + E(\pi^{(p)}(2), 2, 3).$$

当 $E(\pi^{(p)}(1), 2, 3) > t_{j2,2}$ 时, 图2计算出 $E(\pi^{(p)}(2), 2, 3)$



ASAvns Algorithm

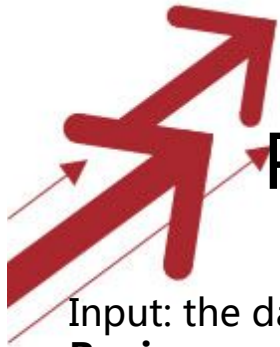
我们采用两阶段局部搜索方式。在算法的开始阶段，执行第一阶段的局部搜索，使用变邻域局部搜索算法获得一些可行解。当温度降低至0.005时，第二阶段局部搜索开始进行。在这阶段中，重新初始化参数是必须的，目的是快速收敛解空间中的解。同时，循环过程再一次被使用，为了收获在第一阶段获得解基础上的更好质量的解，即更优的工作调度序列。需要特别注意的是，在算法中需要设置一个标志参数，保证在第二阶段中只重新初始化参数一次。循环中，如果解比上一次循环中的解适应度小的话，表示新解更优，新解将会替换旧解；否则，当循环次数为N或者N的倍数的时候，循环中的解将会被强制接受。这样的做法是为防止算法陷入局部最优而得不到理想中的全局最优解。与此同时，如果算法完成完整的整个循环，需要耗费太长时间，所以，在循环中我们提出一个时间段上限(time limit)，若循环时间超过它，循环j将终止。这样就保证此算法的NFSP样例实验时间和其它算法耗费的时间相同或者相近。这个时间段上限是根据具体的NFSP样例和其它算法耗费时间大小规定的。

最后，算法结束条件是温度到达终止温度或者循环时间到达时间段上限。整个ASAvns算法可以循环10到20次，并将其中最好的解作为最终解。



参数设置

参数	意义	值
N	工作个数	样例中大小
M	机器个数	样例中大小
Initial_t	初始温度	1000.0
cool_t1	第一阶段温度冷却系数	0.99999
cool_t2	第二阶段温度冷却系数	0.9999
vns_t1	第一阶段变邻域搜索循环次数	$N/3$
vns_t2	第二阶段变邻域搜索循环次数	$3 \times N/2$
end_t1	结束温度	0.0025
end_t2	第二阶段变邻域搜索的开始温度	0.005
current_t	当前温度	$current_t = current_t \times cool_t$



Pseudo-code for the ASAvns Algorithm

Input: the data of the NFSP;

Begin

Initialize parameters including Initial_t, cool_t1, vns_t1;

Induce one solution s randomly;

count=0;

current_t=Initial_t;

While (current_t \geq end_t1 or the stopping criterion is not satisfied)

If (this statement is first executed and the current_t $<$ end_t2)

Reinitialize the parameters including Initial_t, cool_t2, vns_t2;

Endif

For times=1: vns_t1 or vns_t2

s' =s;

If(runtime>time limit)

break;

Endif

Carry out the variable neighborhood search for s' and obtain a new solution s' ' ;

If(fitness(s' ')<fitness(s'))

s' =s' ' ;

Endif

Endfor

If (fitness(s')<fitness(s))

s=s' ;

Else

{

count=count+1;

If(count%N=0)

s=s' ;

Endif

}

Endif

current_t =current_t \times cool_t;

Endwhile

s*=s;

Output: the best solution s* and the final result as the fitness(s*)

End



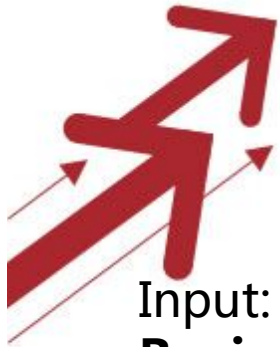
Neighborhood structures

The structures used in the variable neighborhood search:

- **Insert:** Remove the job from the i -th position of a job permutation and insert it into the j -th position in the same permutation. (i, j are rand $(1, N)$ and $i \neq j$)
- **Block insert:** Remove the job sequence between the i -th position and the j -th position and insert it into k -th position in the same permutation. (i, j, k are rand $(1, N)$ and $i \neq j \neq k$)

The neighborhood structure used in the initial part :

- **Swap:** Exchange the job from the i -th position of a job permutation and the j -th position job in the same job permutation. (i, j are rand $(1, N)$ and $i \neq j$)



Pseudo-code for the VNS Algorithm

Input: the permutation π

Begin

$s_0 = \pi$;

$i = \text{rand}(1, N); j = \text{rand}(1, N); i \neq j$;

$s_1 = \text{swap}(s_0, i, j)$;

For $ii = 1 : (N \times (N-1)/2)$

$k_count = 0$;

While ($k_count < 2$)

If ($k_count = 0$)

$i = \text{rand}(1, N); j = \text{rand}(1, N); i \neq j$;

$s_2 = \text{insert}(s_1, i, j)$;

Endif

If ($k_count = 1$)

$i = \text{rand}(1, N); j = \text{rand}(1, N); k = \text{rand}(1, N)$;

$i \neq j \neq k$;

$s_2 = \text{block insert}(s_1, i, j, k)$;

Endif

If ($\text{fitness}(s_2) < \text{fitness}(s_1)$)

$s_1 = s_2$;

$k_count = 0$;

Else

$k_count = k_count + 1$;

Endif

Endwhile

Endfor

If ($\text{fitness}(s_1) \leq \text{fitness}(s_0)$)

$s_0 = s_1$;

Endif

output: the permutation s_0 and
the fitness(s_0)

End



The benchmarks:

12组不同规模的NFSP测试样例将被采用。它们是由Taillard设计的。每一组测试样例包含10个相同规模的具体的NFSP样例。规模是从20个工作×5个机器到50个工作×20个机器变动,标号从Ta001到Ta120。

NEH算法：

[1]Nawaz M, Ensore EE Jr., Ham I (1983) A heuristic algorithm for the m-machine, n-job flow shop sequencing problem.

OMEGA 11:91–95.

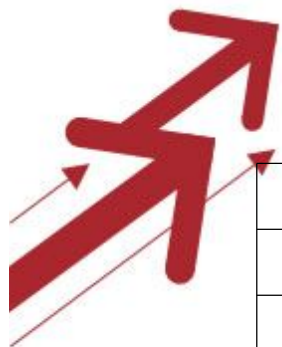
PSOvns 算法 和 HDPSO 算法：

在文献[2]中，一种专门为解决NFSP而改进的粒子群算法PSOvns提出，它的数据测试结果显示，该算法性能比单纯的变邻域局部搜索算法和基因算法高很多。

在文献[3]中，提出了一种混合离散粒子群算法（HDPSO）解决NFSP，这个算法的实验结果显示，它得到的解优于粒子群算法和其它元启发式算法。

[2]Su Shoubao, Wang Jiwen, Zhang Ling, et al., An invasive weed optimization algorithm for constrained engineering design problems, J. Univ. Sci. Technol. China 39 (8) (2009) 885–893.


[3]Q.K. Pan, L. Wang, No-idle permutation flow shop scheduling based on a hybrid discrete particle swarm optimization algorithm, Int. J. Adv. Manuf. Technol. 39 (7–8) (2008) 796–807.



Comparison of ASAvns and NEH

Problem	Scale	NEH	ASAvns	PIP
	$n \times m$	Makespan	Makespan	
Ta001	20×5	1 413	1 380	-2.335%
Ta011	20×10	2 266	2 188	-3.442%
Ta021	20×20	3 482	3 205	-7.955%
Ta031	50×5	3 028	3 014	-0.462%
Ta041	50×10	3 579	3 426	-4.275%
Ta051	50×20	5 813	5 408	-6.967%
Ta061	100×5	5 874	5 836	-0.647%
Ta071	100×10	6 893	6 756	-1.988%
Ta081	100×20	9 667	9 367	-3.103%
Ta091	200×10	11 926	11 684	-2.029%
Ta101	200×20	15 330	15 004	-2.127%
Ta111	500×20	30 600	30 049	-1.801%
Average	PIP:	-3.094%		

PIP(An improvement percent)= $|ASAvns-NEH|/NEH \times 100\%$, 计算结果越小表明算法性能相对NEH算法提高的越高。



Comparison of ASAvns with PSOVns and HDPSO

Scale	PSOVns			HDPSO			ASAvns		
$n \times m$	ARE	ASD	AT	ARE	ASD	AT	ARE	ASD	AT
20×5	-3.01	0.04	2.0S	-3.03	0.00	2.0S	-3.15	0.23	1.0S
20×10	-7.16	0.15	2.0S	-7.28	0.02	2.0S	-7.73	0.08	1.0S
20×20	-6.81	0.27	2.0S	-7.04	0.03	2.0S	-7.28	0.03	1.0S
50×5	-0.75	0.00	5.0S	-0.75	0.00	5.0S	-1.37	0.02	5.0S
50×10	-5.75	0.22	5.0S	-5.88	0.10	5.0S	-6.12	0.03	5.0S
50×20	-6.65	0.42	5.0S	-7.27	0.21	5.0S	-7.58	0.03	5.0S
100×5	-0.72	0.03	10.0S	-0.73	0.00	10.0S	-1.21	0.01	10.0S
100×10	-1.40	0.11	10.0S	-1.55	0.03	10.0S	-2.45	0.01	10.0S
100×20	-4.66	0.33	10.0S	-5.36	0.08	10.0S	-5.47	0.01	10.0S
200×10	-1.41	0.12	20.0S	-1.54	0.02	20.0S	-2.15	0.00	20.0S
200×20	-3.23	0.31	20.0S	-3.71	0.13	20.0S	-4.15	0.01	21.0S
500×20	-1.85	0.15	50.0S	-2.16	0.10	50.0S	-1.18	0.00	67.4S
Average	-3.62	0.18	11.75S	-3.86	0.06	11.75S	-4.15	0.04	13.07S

平均相对差错百分比ARE (the average relative percent error),平均标准差 (the average standard deviation) 和平均时间AT (the average time) 。当ARE越小时 , 表示算法的具体问题结果就越好。



ASAvns算法显著提高了算法性能，优化了NFSP具体问题计算结果。它比PSOvns算法和HDPSO算法有着更高的效率和更强的鲁棒性。

接下来，我们的工作思考更高效的局部搜索策略应用到该算法中，也可以尝试更换基本算法框架，从而进一步提高算法性能，并试验采用此算法解决更多调度问题。

谢谢大家