DISCRETE SELF-ORGANISING MIGRATING ALGORITHM FOR FLOW SHOP SCHEDULING WITH NO WAIT MAKESPAN

Donald Davendra, Ivan Zelinka, Roman Senkerik and Roman Jasek
Department of Applied Informatics, Tomas Bata University in Zlin, Nad Stranemi 4511, Zlin 76001, Czech Republic
{davendra,zelinka,senkerik,jasek}@fai.utb.cz

Abstract
This paper introduces a novel discrete Self Organising Migrating Algorithm for the task of flowshop schedul¬ing with no-wait makespan. The new algorithm is tested with the small and medium Taillard benchmark problems and the obtained results are competitive with the best performing heuristics in literature.

Key words: Self Organising Migrating Algorithm, Flow shop, Scheduling

1. Introduction

Scheduling is an integral part of advanced manufacturing systems. Flowshop scheduling problems have an extensive background in industrial applications, including manufacturing, mining, logistics etc. The general assumption of a majority of flowshop applications is that the sequencing of jobs relies on buffers, either wise refereed to as intermediate storage between machines.

However, a number of applications now exist where the jobs are not allowed to idle between machines, ie. the job proceeds continuously through all the machines. For instance, in the steel-making and continuous casting processes of iron and steel manufacturing enterprises, a no-wait scheduling can reduce the energy loss of high-temperature molten steel and plays an important role in realizing the advanced production style of HCR/DHCR [2].

Therefore, it becomes essential to devise heuristics, which are able to resolve this type of scheduling problem. The heuristic used in this research is the novel Discrete Self Organising Migrating Algorithm (SOMA). SOMA is a class of swarm heuristic, which has been used to solve real domain problems. The discrete variant introduced in the paper attempts to solve permutative variants of scheduling problems.

This paper is structured as follows. Section 2 introduces SOMA and Section 3 introduces Discrete SOMA. No-wait flowshop problem is described in Section 4, whereas the experimentation is given in Section 5. The paper is concluded in Section 6.

2. Self Organising Migrating Algorithm

SOMA [1] is a metaheuristic, which is based on the competitive-cooperative behavior of intelligent creatures solving a common problem.

In SOMA, individual solutions reside in the optimized model's hyperspace, looking for the best individual. It can be said, that this kind of behavior of intelligent individuals allows SOMA to realize very successful searches.

Because SOMA uses the philosophy of competition and cooperation, the variants of SOMA are called strategies. They differ in the way as to how the individuals affect all others. Taking *t* is the iterator for the migrations *M (t* = 1, 2,..., *M),* SOMA can be described as consisting of the following steps:

1. *Definition of parameters.* Before execution, the SOMA parameters (PathLength, Step, PRT, Migrations see Table 1) are defined.
2. *Creating of population.* The population $\left(P = \{X_i^t, X_,^t, \ldots, X_{SP}^t, \}\right)$ is generated consisting of a number of individuals *(SP),* where each individual $X_1^t = \{x_{i,1}^t, x_{i,2}^t, \ldots, x_{i,IS}^t\}$ contains a number of elements *(IS).*
3. *Migration loop.*
   (a) Each individual X_i^t is evaluated for the cost function:
   $$C_i^t = f\left(X_i^t\right) i = 1, \ldots SP$$
   (b) For each element $x_{i\ ;}^t$ in an individual X_i^t, the PRT Vector $A_{i,j}$ is created (1).
   (c) All individuals, perform their run towards the selected individual (Leader), which has the best fitness for that migration according to (2). Each individual is selected piecewise. The movement consists of jumps determined by the step parameter (s) until the individual reaches the final position given by the Path-Length parameter. For each step, the cost function for the actual position is evaluated and the best value is saved. Then, the individual returns to the position, where it found the best- cost value on its trajectory.

SOMA, like other evolutionary algorithms, is controlled by a number of parameters, which are predefined. They are presented in Table 1.

Table 1: SOMA Parameters

Name	Range	Type
PathLength	(1.1-3)	Control
StepSize	(0.11-PathLength)	Control
PRT	(0 - 1)	Control
Migrations	10+	Termination

2.1 Mutation

Mutation, the random perturbation of individuals, is applied differently in SOMA compared with other evolutionary strategies. SOMA uses a parameter called

Proceedings of the Fourth Global Conference on Power Control and Optimization
AIP Conf. Proc. 1337, 285-289 (2011); doi: 10.1063/1.3592479

PRT to achieve perturbation. It is defined in the range [0, 1] and is used to create a perturbation vector (PRT Vector (A)) (1):

$$A_{i,j} = \begin{cases} 1 & \text{if rand}() < \text{PRT} \\ 0 & \text{otherwise} \end{cases} \qquad (1)$$
$$i = 1,2,\ldots,SP;\ j = 1,2,..,IS$$

The novelty of this approach is that in its canonical form, the PRT Vector is created before an individual starts its journey over the search space. The PRT Vector defines the final movement of an active individual in search space.

The randomly generated binary perturbation vector controls the allowed dimensions for an individual. If an element of the perturbation vector is set to zero, then the individual is not allowed to change its position in the corresponding dimension.

2.2 Crossover

In standard evolutionary strategies, the crossover operator usually creates new individuals based on information from the previous generation. Geometrically speaking, new positions are selected from an *N* - dimensional hyper-plane. In SOMA, which is based on the simulation of cooperative behavior of intelligent beings, sequences of new positions in the *N*-dimensional hyperplane are generated. The movement of an individual is thus given as follows:

$$x_{i,j}^{t} = x_{i,j}^{t-1} + \left(x_{L,j}^{t-1} - x_{i,j}^{t-1}\right) s A_{i,j} \qquad (2)$$

where

- $x_{i,i}^{t}$: newcandidate solution
- $x_{i,i}^{t-1}$: original individual
- $x_{L,i}^{t-1}$: leader individual
- $s: \in [0, \text{pathlength}]$
- A: control vector for perturbation

It can be observed from (2) that the PRT vector causes an individual to move toward the leading individual (the one with the best fitness) in *N-k* dimensional space. If all *N* elements of the PRT vector are set to 1, then the search process is carried out in an *N* dimensional hyper-plane *(i.e.* on a *N+1* fitness landscape). If some elements of the PRT vector are set to 0, then the second terms on the right-hand side of (2) equals 0. This means those parameters of an individual that are related to 0 in the PRT vector are not changed during the search. The number of frozen parameters, *k,* is simply the number of dimensions that are not taking part in the actual search process. Therefore, the search process takes place in an *N-k* dimensional subspace.

3. Discrete Self Organising Migrating Algorithm

DSOMA is the discrete version of SOMA, developed to solve permutation based combinatorial optimization problem. The same ideology of the sampling of the space between two individuals is retained.

3.1 Initialization

The initial population is initialized as a permutative schedule representative of the size of the problem at hand. Each element within the individual is unique.

$$P \supseteq X_i^t \supseteq x_{i,j}^t = \begin{cases} 1 + \text{rand}() \bullet (IS - 1) \\ \text{if } x_{i,j}^t \notin \{x_{i,1}^t, \ldots, x_{i,j-1}^t\} \end{cases} \qquad (3)$$

Each individual is vetted for its fitness (4), and the best individual and its fitness its obtained as X_{best}^t and C_{best}^t. The migration counter *t* is set to 1; $t = 1; t = 1,..,M$ and the individual index *i = 1.*

$$C_i^t = f\left(X_i^t\right), i = 1,..,SP \qquad (4)$$

3.2 Jump Sequences

DSOMA operates by calculating the number of discrete jump steps that each individual has to circumnavigate. In DSOMA, the parameter minimum jumps (J_{min}) is used in lieu of PathLength which states the minimum number of individuals or sampling between the two individuals. Taking two individuals in the population, one as the incumbent (X_i^t) and the other as the leader (X_L^t), the possible number of jump solutions J_{max} is the mode of the difference between the adjacent values of the elements in the individual (5).

$$J = \left\{ \left| x_{i,j}^{t-1} - x_{L,j}^{t-1} \right| \right\} \qquad (5)$$
$$J_{max} = \text{mode}[J]$$

The step size *(s)* can now be calculated as the integer fraction between the required jumps and possible jumps (6).

$$s = \text{int}\left[\frac{J_{min}}{J_{max}} \right] \qquad (6)$$

The jump matrix **J**, which contains all the possible jump positions can be calculated as:

$$J_{j,l}^{P} = \begin{cases} x_{i,j}^{t-1} + sl & \text{if } x_{i,j}^{t-1} + sl < x_{L,j}^{t-1} \\ & \text{and } x_{i,j}^{t-1} < x_{L,j}^{t-1}; \\ x_{i,j}^{t-1} - sl & \text{if } x_{i,j}^{t-1} - sl < x_{L,j}^{t-1} \\ & \text{and } x_{i,j}^{t-1} > x_{L,j}^{t-1}; \\ 0 & \text{otherwise} \end{cases} \qquad (7)$$
$$j = 1,\ldots,IS;\ l = 1,..,J_{min}$$

3.3 New Jump Individual Selection

A total of J_{min} new individuals can now be constructed from the jump positions. Each new individual $Y_{w,j}^t$ where $w = 1,..,J_{min}$ is constructed piecewise from

the jump matrix **J**. Each element $y^t_{w,j}$ in the individual, indexes its values from the corresponding j^{th} array in the jump matrix $J^P_{j,l}$.Each l^{th} ($I = 1,.., J_{min}$) position for a specific $j^{th}(j = 1,2,..,IS)$ element is sequentially checked to ascertain if it already exists in the current individual $Y^t_{w,j}$.If this is a new element, it is then accepted in the individual, and the corresponding l^{th} value in the jump matrix is set to zero $J^P_{j,l} = 0$.This iterative procedure can be given as in (8).

$$Y^t_{w,j} = \begin{cases} J^P_{j,l} \begin{cases} \text{if } J^P_{j,l} \notin \{y^t_{w,1},\ldots,y^t_{w,j-1}\} \\ \quad \text{and} J^P_{j,l} \neq 0; l = 1,..,J_{min} \\ \text{then } J^P_{j,l} = 0 \end{cases} \\ 0 \qquad \text{otherwise} \end{cases} \quad (8)$$
$$w = 1,..,J_{min}$$

3.4 Population Update

After each individual is evaluated for its fitness value as in (9).

$$C^t_w = f(Y^t_w), w = 1,..,J_{min} \quad (9)$$

2-OPT local search is applied to the best individual Y^t_{best} obtained with the minimum fitness value ($\min[C^t_w]$). After the local search routine, the new individual is compared with the fitness of the incumbent individual C^{t-1}_i, and if it improves on the fitness, then the new individual is accepted in the population (10).

$$X^t_i = \begin{cases} Y^t_{best} & \text{if } f(Y^t_{best}) < C^{t-1}_i \\ X^{t-1}_i & \text{otherwise} \end{cases} \quad (10)$$

3.5 Iteration

Sequentially, each individual X^{t-1}_{i+1} is selected from the population, and it begins its own sampling towards the designated leader X^{t-1}_L. It should be noted that the This is generally considered the most constrained variant of flowshop scheduling.

The following notations are used to formulate the no-wait flowshop problems: n = number of jobs to be scheduled, m = number of machines in the no-wait flowshop, $t_{i,j}$ = processing time for the i^{th} job on the j^{th} machine, $d_{i,k}$ = minimum delay on the first machine between the start of job i and job k due to the no-wait constraint, *[i]* = the job processed in position i, $C_{[i]}$ = the completion time of the job processed in position. *TFT* = total flow time, i.e. the sum of flow times of all leader does not change during the evaluation of one migration.

4. No Wait Flowshop Scheduling Problem

A no-wait flowshop scheduling problem is one where once a job starts is cannot be buffered till it is completed. jobs.

The minimum delay time $d_{i,k}$ and completion time $C_{[i]}$ can be calculated as:

$$d_{i,k} = t_{i,1} + \max_{2 \le j \le m}\left(\sum_{p=2}^{j} t_{ip} - \sum_{p=1}^{j-1} t_{kp}\right)$$
$$\sum_{i=1}^{n}\sum_{j=1}^{m} t_{i,j}$$
$$C_{[i]} = \sum_{j=1}^{m} t_{[1],j}, \quad (11)$$
$$C_{[i]} = \sum_{k=2}^{i} d_{[k-1],[k]} + \sum_{j=1}^{m} t_{[1],j}, i = 2,3,..,n.$$

All jobs are assumed to be available at time zero, the total flow time can then be given as in (12).

$$TFT = \sum_{i=2}^{n}\left(\sum_{k=2}^{i} d_{[k-1],[k]} + \sum_{j=1}^{m} t_{[1],j}\right) + \sum_{j=1}^{m} t_{[1],j} =$$
$$\sum_{i=2}^{n}\sum_{k=2}^{i} d_{[k-1],[k]} + \sum_{i=1}^{n}\sum_{j=1}^{m} t_{[i],j} = \quad (12)$$
$$\sum_{i=2}^{n}(n+1-i) d_{[i-1],[i]} + \sum_{i=1}^{n}\sum_{j=1}^{m} t_{i,j}$$

where is the sum of the processing time of all jobs in all machines [2].

5. Experimentation

Experimentation was carried out on the benchmark flow-shop instances by DSOMA. The operating parameters of DSOMA is given in Table 2. All parameters were kept stagnant for all experiments.

The experimentation is done with the small and medium Taillard Benchmark Sets [3]. These benchmarks comprise of 12 different sets of problems ranging from

Table 2: Operating parameters for DSOMA

Parameter	Value
Solutions *(SP)*	100
Migrations	100
Sampling *(J_{min})*	20
Local Search	2 OPT

The small and medium sized problems can be designated from 20 jobs and 5 machines to 50 jobs and 20 machines; in a total of 6 data sets and 60 problem instances.

Each instance has 10 independent replications and the percentage relative difference *(PRD)* is computed as follows:

$$PRD = \frac{100 \times \left(C^{F\&V} - C^{DSOMA}\right)}{C^{F\&V}} \quad (13)$$

where $C^{F\&V}$ is the referenced makespan provided by [4], and C^{DSOMA} is the makespan found by the DSOMA algorithm. Furthermore, average percentage relative difference (APRD), maximum percentage relative difference (MaxPRD), minimum percentage relative difference (MinPRD) and the standard deviation (SD) of PRD are calculated. The average execution time for each set (T(s)) is also provided.

The results are given in Table 3. From the results, DSOMA is better performing than F&V algorithm, with an average APRD of 0.024. The paired *t*-test values at 95% confidence interval for the medium sized problems is given in Table 4.

From the *t*-test results, DSOMA is significantly better than F&V on the data set of 50 jobs 5 machines and identical on the data sets of 50 jobs 10 machines and 50 jobs 20 machines.

5.1 Comparison with Discrete Particle Swarm Algorithm

Comparison of DSOMA is done with the Discrete Particle Swarm Algorithm (DPSOVND) of [5]. From current literature it has been shown as the most promising algorithm. The results are given in Table 5.

From the results, DPSOVND is a better heuristic. For the first three data sets, the results are identical, as both algorithms obtain the optimal results. For the medium sized problems, DPSOVND is better performing with 0.168 average PRD against 0.04 of DSOMA.

The *t*-test values at 95% confidence interval is given in Table 6 for the medium sized problems. From the results of the *t*-test, DPSOVND is significantly better performing for the 50 jobs 5 machines and 50 jobs 10 machines problem whereas DSOMA is comparable to DPSOVND for the 50 jobs 20 machines problem.

6. Conclusion

DSOMA is introduced in this paper as a novel approach to solve no-wait flowshop scheduling problems. From the results, we can state that while not the best algorithm for this class of problem from literature, DSOMA is very competitive.

When comparing DSOMA with F&V algorithm, DSOMA is clearly a better performing heuristic, whereas it is competitive with the DPSOVND algorithm.

The fallibility of DSOMA can be attributed to the fact that it didn't use an seed solution like NEH, and it uses the rudimentary 2-OPT local search routine. A seed solution and a tailor made local search would, it is believed improve the algorithm.

A number of improvements are now envisioned for DSOMA, including extensive parameter testing and infusion of better local search routines.

Acknowledgement

This work was supported by grant No. MSM 7088352102 of the Ministry of Education of the Czech Republic and by grants of the Grant Agency of the Czech Republic GACR 102/09/1680.

REFERENCES

[1] I. Zelinka and J. Lampinen, "Soma - self-organizing migrating algorithm," in *Mendel, 6th International Conference on Soft Computing, Brno, Czech Republic,* (Brno, Cesch Republic), 2000.

[2] J. lin CHANG, D. wei GONG, and X. ping MA, "A heuristic genetic algorithm for no-wait flowshop scheduling problem," *Journal of China University of Mining and Technology,* vol. 17, no. 4, pp. 582 - 586, 2007.

[3] E. Taillard, "Benchmarks for basic scheduling problems," *European Journal of Operations Research,* vol. 64, pp. 278-285, 1993.

[4] A. Fink and S. Vo, "Solving the continuous flow-shop scheduling problem by metaheuristics," *European Journal of Operational Research,* vol. 151, no. 2, pp. 400-414, 2003.

[5] Q.-K. Pan, M. Fatih Tasgetiren, and Y.-C. Liang, "A discrete particle swarm optimization algorithm for the no-wait flowshop scheduling problem," *Comput. Oper. Res.,* vol. 35, no. 9, pp. 2807-2839, 2008.

Table 3: Computation comparison of DSOMA with F&V [4]

JxM	DSOMA				
	APRD	MinPRD	MaxPRD	SD	T(s)
20x5	0	0	0	0	0.18
20x10	0	0	0	0	0.32
20x20	0	0	0	0	0.51
50x5	**0.152**	0.0079	0.350	0.126	0.79
50x10	**0.028**	-0.129	0.164	0.099	1.53
50x20	-0.035	-0.252	0.055	0.085	2.25
Average	**0.024**	-0.062	0.095	0.103	0.93

Table 4: Paired t-test for DSOMA against F&V

Set	H_o	H_1	*t -value*	*p - value*	*p < 0.005*	H_o	H_1
50 x 5	DSOMA =F&V	DSOMA ≠F&V	-3.718 -	0.004	Yes	Reject	Accept
50 x 10	DSOMA =F&V	DSOMA ≠F&V	0.864	0.4098	No	Accept	Reject
50 x 20	DSOMA =F&V	DSOMA ≠F&V	1.301	0.2252	No	Accept	Reject

Table 5: Computation comparison of DSOMA with DPSOVND of [5]

JxM	DSOMA					DPSOVND			
	APRD	MinPRD	MaxPRD	SD	T(s)	PRD	MinPRD	MaxPRD	SD
20x5	0	0	0	0	0.18	**0**	0	0	0
20x10	0	0	0	0	0.32	**0**	0	0	0
20x20	0	0	0	0	0.51	**0**	0	0	0
50x5	0.152	0.0079	0.350	0.126	0.79	**0.333**	0.022	0.634	0.176
50x10	0.028	-0.129	0.164	0.099	1.53	**0.116**	-0.069	0.303	0.114
50x20	-0.035	-0.252	0.055	0.085	2.25	**0.057**	-0.082	0.181	0.097
Average	0.024	-0.062	0.095	0.103	0.93	**0.168**	-0.043	0.373	0.129

Table 6: Paired t-test for DSOMA against DPSOVND

Set	H_0	H_1	*t-value*	*p-value*	*p < 0.005*	H_0	H_1
50 x 5	DSOMA = DPSOVND	DSOMA ≠ DPSOVND	6.06521	0.000	Yes	Reject	Accept
50 x 10	DSOMA = DPSOVND	DSOMA ≠ DPSOVND	5.2861	0.000	Yes	Reject	Accept
50 x 20	DSOMA = DPSOVND	DSOMA ≠ DPSOVND	1.999	0.0766	No	Accept	Reject