An Application of Hierarchical Genetic Strategy in sequential scheduling of permutated independent jobs

Joanna Kołodziej and Marcin Rybarski

University of Bielsko-Biała, Department of Mathematics and Computer Science,
ul. Willowa 2, 43-309 Bielsko-Biała, Poland
email: jkolodziej@ath.bielsko.pl; email: mrybarski@ath.bielsko.pl

Abstract. The aim of this paper is to present an implementation of Hierarchic Genetic Strategy (HGS) in solving the Permutation Flowshop Scheduling Problem (PFSP). We defined a hierarchic scheduler based on HGS structure for the exploration of the wide and complicated optimization landscape studied by Reeves [6]. The objective of our work is to examine several variations of HGS operators in order to identify a configuration of operators and parameters that works best for the problem. From the experimental study we observed that HGS implementation outperforms existing schedulers in many of considered instances of a static benchmark for the problem.

1 Introduction

The *Permutation Flowshop Scheduling Problem (PFSP)* can be defined as a problem of processing of a sequence $\{J_s\}_{1 \geq s \geq n}$ of n jobs on a set of m machines $\{M_r\}_{1 \geq r \geq m}$. It is not only NP-hard , but it is one of the worst members in the class. An indication of this is given by the fact that problem for 10 jobs and 10 machines remained unsolved for over 20 years.

Besides exhaustive search algorithms based on branch and bound methods (see [9] for example), several heuristic algorithms have been developed. The performance of these heuristics has been measured on a set of 120 benchmark instances of the PFSP proposed in [10] by Taillard. The steady-state genetic algorithms hybridized with some specialized local search procedures (see [11], [7]) has been recently shown to be very successful in solving PFSP. High efficiency of such methods comes from the abilities of the accurate exploration of the optimization landscape by the neighborhood operators. Reeves showed in [6] that the landscape for Taillard's benchmarks has a "big valley" structure, where the local optima occur relatively close to each other, which can be the main reason of problems in the detection of accurate solutions of PFSP.

The objective of this work is to design and implement batch schedulers based on a family of dependent genetic processes enabling a concurrent local search in the optimization domain, which can reduce significantly the complexity of the local procedures implemented in Reeves's and Yamada's work [7]. We defined a *Hierarchic Genetic Scheduler (HGS-Sched)* which is based on the Hierarchical Genetic Strategy (HGS) defined by [4]. HGS has been successfully applied in solving many ill-posed optimization problems in the continuous domain.

The rest of the paper is organized as follows. We define PFS Problem in Section 2. Definitions of HGS-Sched procedures and a short description of applied genetic operators are given in Section 3. Section 4 contains an experimental analysis of the performance of HGS-Sched and specialized local search genetic algorithm for the benchmark of static scheduling defined by Taillard [10]. The paper ends with some final remarks.

2 Statement of the problem

The processing of job J_s on machine M_r in PFSP is called the *operation* $O_{sr}, 1 \geq s \geq n$, $1 \geq r \geq m$. It requires the exclusive use of the machine for an uninterrupted duration called the *processing time*. A *schedule* is the solution of the problem and it can be represented by the set of permutations of jobs on each machine.

An example of the definition of 3×3 PFSP is given in Table 1 (see also [7]).

Table 1. An example of 3×3 PFSP.

Job	Operations routing (processing time)		
1	1(3)	2(3)	3(3)
2	1(2)	3(3)	2(4)
3	2(3)	1(2)	3(1)

The data includes the routing of each job through each machine and the processing time for each operation (in parentheses). An example schedule for that problem is defined as *a job sequence matrix* presented in Figure 1.

M_1	1, 2, 3
M_2	3, 1, 2
M_3	2, 1, 3

Figure 1. An example schedule for 3×3 problem

Let us denote the processing times $p(s;r)$ for job J_s on machine M_r, and a job permutation $\{\pi_1, \pi_2, \ldots \pi_n\}$. We can calculate the completion times $C(\pi_s;r)$ of jobs on machines as follows:

$$\begin{array}{rll} C(\pi_1;1) = & p(\pi_1;1) & \\ C(\pi_s;1) = & C(\pi_{s-1};1) + p(\pi_s;1); & s = 2 \ldots n \\ C(\pi_1;r) = & C(\pi_1;r-1) + p(\pi_1;r); & r = 2 \ldots m \\ C(\pi_s;r) = & max\{C(\pi_{s-1};j); C(\pi_s;r-1)\} + p(\pi_s;r); & s = 2 \ldots n; r = 2 \ldots m \end{array} \tag{1}$$

The time required to complete all the jobs is called the *makespan* L, which can be defined in the following way:

$$L = C_{max}(\pi) = C(\pi_n;m). \tag{2}$$

The objective when solving or optimizing this problem is to determine the schedule which minimizes L. Formally, we have to find a permutation π^* in the set of all permutations of jobs Π, such that $C_{max}(\pi^*) \leq C_{max}(\pi); \forall \pi \in \Pi$.

3 Hierarchic Genetic Scheduler

Differently from classical GA algorithms, which maintain only an unstructured population of individuals, Hierarchical Genetic Strategy (HGS) enables a concurrent search in the optimization domain by many small populations. The creation of these populations is governed by the dependent genetic processes with low complexity. The processes of low order represent chaotic search. They detect the promising region on the optimization landscape in which more accurate processes are activated.

The dependency relation among processes in HGS has a tree structure. For each branch in this tree we define a *degree* parameter, denoted by $j \in \mathbb{N}$, the value of which corresponds to the accuracy of the search. There is a unique branch of the lowest degree 0 called *root*. A new branch could be created after running a metaepoch in the parental one.

An example structure of HGS-Sched, which is similar to the basic structure of HGS [8] is shown in Figure 2, where $P^e_{i,j}$ denotes populations evolving in branches of the different degrees with $e \in \mathbb{N}$ as the global metaepoch counter and i as the unambiguous branch identifier, which describes the "history of creation" of the given branch [4].

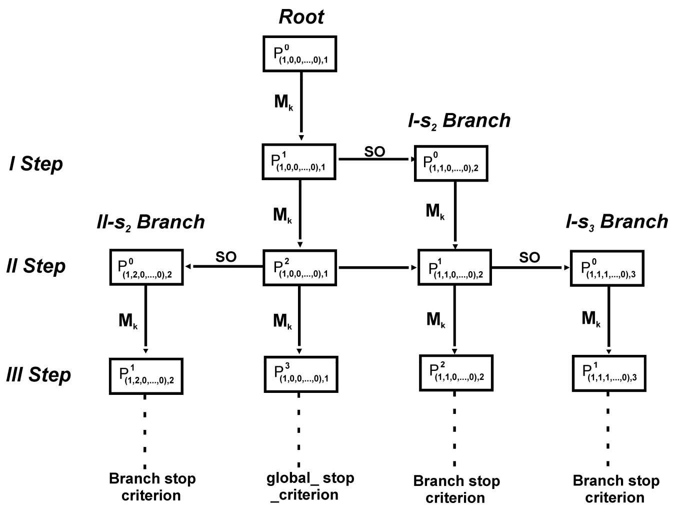

Figure 2. An example structure of HGS-Sched after execution of 3 metaepochs

A k –periodic metaepoch $M_k, (k \in \mathbb{N})$ is a discrete evolution process executed in the branch of a given degree, which is terminated after k generations by the selection of the best adapted individual $\widehat{x}$. A procedure of activation of a new process of higher degree is called a *Sprouting Operation* and it is defined in the rest of this section.

3.1 Genetic Engine in HGS-Sched Branches

We used in our implementation a genetic algorithm template, presented in Figure 3 as a genetic engine in HGS-Sched. This template is similar to classical $(\mu + \lambda)$ Evolutionary Strategy framework [2].

The individuals (schedules) in populations are encoded into sequences of jobs permutations $\{\pi_1, \pi_2, \ldots \pi_n\}$. Each permutation π_s is defined as a n-length sequence of integers of the range $\{1, \ldots, n\}$. A chromosome for the schedule presented in Figure 1 is defined by the following sequence: $((1, 2, 3); (3, 1, 2); (2, 1, 3))$.

An initial population for the root of the HGS-Sched structure is selected randomly in this approach. The processes of higher degrees start from the populations obtained as the outcomes

```
Generate the initial population $P^0$ of size $\mu$;
Evaluate $P^0$;
while not termination-condition do
    Select the parental pool $T^t$ of size $\lambda$; $T^t := Select(P^t)$;
    Perform a subsequence crossover procedure on pairs of schedules in $T^t$ with probability $p_c$; $P_c^t := Cross(T^t)$;
    Perform a subsequence mutation procedure on schedules in $P_c^t$ with probability $p_m$; $P_m^t := Mutate(P_c^t)$;
    Evaluate $P_m^t$ ;
    Create a new population $P^{t+1}$ of size $\mu$ from schedules in $P^t \cup P_m^t$ ; $P^{t+1} := Replace(P^t; P_m^t)$
    $t := t + 1$;
end while
return Best found schedule as solution;
```

Figure 3. Genetic engine template

of the *Sprouting Operator* defined in the next section.

The crossover and mutation procedures used in a main loop of algorithm in Figure 3 are the simple extensions of similar procedures defined for problems in permutation spaces.

The main idea of *Subsequence crossover* is to perform a crossover operation independently on corresponding permutation sequences in both parental chromosomes. Let us denote by $\pi^1(p) = \{\pi^1_1(p), \ldots, \pi^1_n(p)\}$ and $\pi^2(p) = \{\pi^2_1(p), \ldots, \pi^2_n(p)\}$ a pair of parental individuals. The outcome of the subsequence crossover is a pair of offsprings denoted by $\pi^1(ch) = \{\pi^1_1(ch), \ldots, \pi^1_n(ch)\}$ and $\pi^2(ch) = \{\pi^2_1(ch), \ldots, \pi^2_n(ch)\}$, where $\{\pi^1_s(ch), \pi^2_s(ch)\} = Cross(\pi^1_s(p), \pi^2_s(p))$. We proposed *Cycle Crossover (CX)* as the basic crossover procedure executed on the pairs of job permutations. Each job sequence in the given chromosome is equivalent to the path representation in the TSP.

The *Sequential mutation* is defined in the similar way: a simple mutation operation is performed for each job permutation in a given chromosome. Two definitions are considered for the basic mutation operator in our implementation: *Move* mutation, where a job from the randomly selected position is moved to the end of the jobs sequence, and *Swap* mutation, where two randomly selected positions are chosen and their corresponding jobs are swapped. We also used *Linear Ranking* as the selection mechanism and *Elitist Replacement* procedure in our HGS implementation.

3.2 HGS-Sched Procedures

The tree structure of HGS-Sched is created by a *HGS-Sched Sprouting Operator (SO)*. It defines the new branches of higher degree "sprouted" from the current (parental) branch. The *SO* operator is given by the following formula:

$$SO(P^e_{i,j}) = (P^e_{i,j}, P^0_{i',j+1}), \tag{3}$$

where $\widehat{x}$ is the best adapted individual found in the parental branch $P^e_{i,j}$ after execution of e-th metaepoch, $P^0_{i',j+1}$ is an initial population for a new branch of degree $j+1$ and $i' = (i_1, \ldots, i_{j-1}, 1, 0, \ldots, 0)$.

Let A_{s_j} be the operator which "cuts out" a s_j-length prefix from each sequence of jobs $\pi_s; s = 1, \ldots, n$ (permutation of jobs) in a given chromosome x. It is defined as follows:

$$A_{s_j}(x) = (\tilde{\pi_1}, \ldots, \tilde{\pi_n}); |\tilde{\pi_s}| = s_j, |x| \geq m \cdot s_j, \tag{4}$$

where $|x| = n \cdot m$ denotes the length of x.

The s_j-neighborhood of solution x contains all individuals, in which the sequences of jobs can differ from the corresponding components of x by the $(n - s_j)$-length suffixes. It can be achieved by the permutation of jobs in the suffix. The individuals in population $P^0_{i',j+1}$ are selected from the s_j-neighborhood ($1 \leq s_j \leq n$) of the solution $\widehat{x}$.

The values of s_j are calculated in the following way:

$$s_j = S^j \cdot n, \tag{5}$$

where $S \in [0, 1]$ is a global strategy parameter called *neighborhood parameter*, j- the branch degree, n- the number of jobs.

The code of *HGS-Sched SO* procedure is similar to the code of *SO* procedure for HGS, which can be found in [4].

4 Experimental study

We have performed a simple experimental evaluation of the HGS implementation for Taillard's benchmark of static instances for PFSP [10]. They are labeled by $n \times m$, where n is the number of jobs and m - the number of machines. We wanted to verify the effectiveness of HGS-Sched in the exploration of the optimization landscape with a "big valley" structure and compare it with the performance of GA-based scheduler with a special local search procedure defined by Yamada and Reeves [12].

Table 2. HGS-Sched global parameters values

Parameter	**Value**
degrees of branches (j)	0 and 1
$period\_of\_metaepoch$ - (k)	100
$nb\_of\_metaepochs$	10
neighborhood parameter - (s)	0.5
$mut\_prob(0)$	0.4
$mut\_prob(1)$	0.2
$cross\_prob$	0.8

The values of HGS-Sched parameters for all tests are given in Table 2. The parameter $nb\_of\_metaepochs$ denotes the maximal number of metaepochs executed in the root. It defines a global stop criterion for the strategy. The parameter $mut\_prob(j)$ denotes the probability of mutation in the branch of degree j and $cross\_prob$ is the probability of crossover, which is identical in the branches of all degrees. The initial population of the root was created by a uniform random generator and linear ranking method was used as selection procedure.

In the first step of our simple experimental study we tried to find an appropriate mutation operator for HGS-Sched, which is crucial for the variation of the search accuracy in the branches of the strategy. In the second step we made a short comparison analysis of the performances of our algorithm and GA-based scheduler defined in [12].

4.1 Experimental calibration of mutation operator for HGS-Sched

A simple tuning process of mutation operator for HGS-Sched was performed for one of the Taillard's problems formulated for 50 jobs and 20 machines. The sizes of populations in root and sprouted branches of degree 1 were 50 and 18, respectively. The intermediated populations consisted of 48 and 16 individuals. Each experiment was repeated 30 times under the same configuration of operators and parameters and the average makespan was computed.

Table 3. Comparison of mutation operators for makespan and flowtime values.

Mut. Operator	Average Makespan	Mut. Operator	Average Makespan
Move	**3870**	Swap	3876
	3715		3722
	3660		3678
	3745		3750
	3608		3640
	3708		3749
	3710		3729
	3715		3746
	3762		3780
	3777		3797

The results of the comparison analysis of the performance of two mutation operators are reported in Table 3. We combined them with *CX* procedure. *Move* mutation outperforms significantly *Swap* method.

4.2 Comparison Analysis of Genetic-Based Schedulers

We applied HGS-Sched with the operators and parameters selected as optimal combination in the previous section for a comparison analysis of our strategy with the method defined by Yamada and Reeves in [12].

The main idea of Yamada's and Reeve's method, known as the MSXF-GA scheduler, is based on a generalization of Genetic Local Search algorithm (GLS), where an offspring obtained by a recombination operator is not included in the next generation directly but it is used as an initial solution for the subsequent local search. Yamada and Nakano [11] have defined first the Multi-Step Crossover Fusion operator (MSXF), where one of the parents itself is a new starting point for a local search procedure, then Yamada and Reeves [12] applied this procedure as the recombination operator in the steady-state GA framework.

We select three of large scale instances of Taillard's benchmark to our experimental study: "50 $\times$ 20", "100 $\times$ 20" and "200 $\times$ 20". Each algorithm was stopped after 700 iterations (calculated for root in the case of HGS implementation). It means that the number of executed metaepochs was reduced to 7 in the comparison with the value of this parameter given in Table 2. Each experiment was repeated 30 times under the same configuration of parameters. For both algorithms each run took about 12, 21 and 47 minutes of CPU time respectively for each "50 $\times$ 20", "100 $\times$ 20" and "200 $\times$ 20" problem.

We reported in Table 4 the makespan values obtained by the two algorithms under study together with theoretical lower bounds (lb) and upper bounds (ub) of the makespan taken from OR-library originally published in [1] and frequently updated on the following website: (http://people.brunel.ac.uk/$^–"sim˜$mastjb/jeb/info.html). These upper bounds are the currently best-known makespans for Taillard's benchmarks.

From the results in Table 4 we can observe that HGS-Sched outperforms MSXF-GA for all but one Taillard's benchmarks. HGS-Sched gives better results than those reported in the OR-library in 20% of considered instances defined for "50 $\times$ 20" problem and in 10% of considered instances defined for "100 $\times$ 20". It was not so effective for "200 $\times$ 20" problem. However,

Table 4. Comparison of best makespan values for the selected Taillard's benchmark problems. Best results are shown in bold

Problem	MSXF-GA	HGS-Sched	lb-ub	Problem	MSXF-GA	HGS-Sched	lb-ub
50×20	3861	3851	3771- 3850	100×20	6242	6223	6106 - 6202
	3709	3708	3668- 3704		6217	6192	6183
	3651	3641	3591- 3640		6299	**6270**	6252- 6271
	3726	3721	3635- 3720		6288	6282	6252- 6271
	3614	3613	3553- 3610		6329	6298	6254- 6269
	3690	3682	3667- 3681		6380	6319	6262- 6314
	3711	**3703**	3672- 3704		6302	6302	6302- 6364
	3699	3698	3627- 3691		6433	6301	6184- 6268
	3760	**3742**	3645- 3743		6297	6280	6204- 6275
	3767	3760	3696- 3756		6448	6421	6315- 6401

Problem	MSXF-GA	HGS-Sched	lb-ub
200×20	11272	11265	11152-11195
	11299	11276	11152-11195
	11410	11324	11281
	11347	11319	11275
	11290	11285	11259
	11250	11212	11176
	11438	11392	11337-11360
	11395	11383	11301-11334
	11263	11209	11145-11192
	11335	11331	11284-11288

the differences between obtained makespan values and the best values from OR-library are not significant, so we can conclude that HGS-based scheduler seems to be the promising method in the exploration of the ”big valley” structure in the optimization landscape for some Taillard's problems. The main advantage of HGS application is no need of implementation a specialized complicated local search mechanism.

5 Conclusions and future work

In this work we have presented the implementation of Hierarchic Genetic Strategies (HGS) for Permutation Flowshop Scheduling Problem. We have examined several variations of HGS operators in order to identify a configuration of operators and parameters that works best for the problem aiming at the design of efficient batch schedulers. The HGS implementation was experimentally studied using a known benchmark of static instances for the problem and the obtained results were compared with the effective local search-based scheduler and the best obtained results known from the literature. From the experimental study we observed that HGS-Sched can be an effective tool for exploration of the ”big valley” structure in the optimization landscape for the PFSP.

In our future work we plan to study the performance of the HGS scheduler in the heterogeneous environment like computational grid systems. The results of simple experiments reported in [5] confirm high efficiency of our methods in solving large scale static problems defined in [3]. We would also like to design an interface for the HGS scheduler for its application to real Grid environments.

Bibliography

[1] J. E. Beasley. Or-library: Distributing test problems by electronic mail. *European J.Operational Research,*, (41):1069–1072, 1990.

[2] H. G. Beyer. *The Theory of Evolution Strategies*. Natural Computation. Springer Vlg., Berlin-Heidelberg, 2001.

[3] T. D. Braun, H. J. Siegel, N. Beck, L. L. Boloni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen, and R. F. Freund. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *Journal of Parallel and Distributed Computing*, 61(6):810ℓ–837, 2001.

[4] R. Gwizdała J. Kołodziej and J. Wojtusiak. Hierarchical genetic strategy as a method of improving search efficiency,. *Advances in Multi-Agent Systems, R. Schaefer and S. Sędziwy (Eds.), UJ Press, Cracow*, pages 149–161, 2001.

[5] J. Kołodziej, F. Xhafa, and Ł. Kolanko. Hierarchic genetic scheduler of independent jobs in computational grid environment. In To Appear, editor, *Proc. of ECMS09*, 2009.

[6] C. R. Reeves. Landscapes, operators and heuristic search. *Annals of Operations Research*, 86:473–490, 1999.

[7] C. R. Reeves and T. Yamada. Genetic algorithms, path relinking and the flowshop sequencing problem. *Evolutionary Computation*, 6(1):230–244, 1998.

[8] R. Schaefer and J. Kołodziej. Genetic search reinforced by the population hierarchy. *FOGA VII,Morgan Kaufmann*, pages 383–401, 2003.

[9] T. Śliwiński and E. Toczyłowski. Algorytm harmonogramowania zadań podzielnych na maszynach równoległych przy uwzględnieniu przezbrojeń i ograniczeń zasobowych. In *Proc. of XV National Conference of Automation 2005, Warszawa, 27-30.06.05*, 2005.

[10] E. Taillard. Benchmarks for basic scheduling problems. *E. J. of Oper. Res*, (64):278ℓ–285, 1993.

[11] T. Yamada and R. Nakano. Scheduling by genetic local search with multi-step crossover. In *4th PPSN*, pages 960ℓ–969, 1996.

[12] T. Yamada and C. R. Reeves. Permutation flowshop scheduling by genetic local search. In *Pro- ceedings of the 2nd IEE/IEEE International Conference on Genetic ALgorithms in Engineering Systems (GALESIA '97)*, pages 232–238, 1997.