Hybrid Genetic Algorithms for the Open-Shop Scheduling Problem

Zbigniew Kokosiński and Łukasz Studzienny,
zk@pk.edu.pl
Department of Automatic Control, Cracow University of Technology,
ul.Warszawska 24, 31-155 Kraków, Poland

Summary
In this paper novel hybrid genetic algorithms for Open--Shop Scheduling Problem (OSSP) are presented. Two greedy heuristics LPT-Task and LPT-Machine are proposed for decoding chromosomes represented by permutations with repetitions. For comparison the standard permutation representation of OSSP instances is used. The algorithms apply also efficient crossover operator LOX and mutation operators SWAP and INVERT with constant and variable mutation probabilities. We compare conventional GA to parallel genetic algorithm (PGA) in a migration model. The performance of the algorithms with various settings is verified by computer experiments on a set of large random OSSP instances.

Key words:
Hybrid metaheuristic, parallel genetic algorithm, open-shop scheduling, LPT heuristic

1. Introduction

Given *n* jobs, every job composed of *m* operations to be processed by *m* dedicated machines, and the processing times required for all operations, the Open-Shop Scheduling Problem (OSSP) is defined as a problem of determining order of execution of a given set of non-splitable job operations on dedicated machines providing that each job is processed by at most one machine at any given time, operations of the same job do not overlap and the makespan of the obtained schedule is minimal.

In opposition to many specific scheduling problems (Job-Shop, Flow-Shop), where some additional assumptions are applied in the OSSP problem no specific assumption on operation order for the given job are made.

OSSP belongs to the class of NP-hard combinatorial optimizations problems. Collections of hard OSSP instances were proposed by Taillard [23], Guéret and Prins [25], Brücker [26].

Summary of best results obtained by researchers applying various metaheuristics and hybrid techniques are available at [5,17]. Intensive research conducted in this area resulted in a large number of exact and approximate algorithms, heuristics and metaheuristics [5,6,8,10].

Genetic algorithms (GA) are metaheuristics often used for solving combinatorial problems [12-15] and scheduling [6,16-21]. This approach is based on co-evolution of a number of populations that exchange genetic information during the evolution process according to a communication pattern [2,3,4]. Recently a number of parallel versions of GA for OSSP were studied [22].

In this paper we present results of our research concerning both sequential and parallel hybrid genetic algorithms for the OSSP. Two greedy heuristics for chromosome decoding are proposed: LPT-Task and LPT-Machine, that are developed on the basis of the well known Longest Processing Time (LPT) scheduling heuristic [7]. A set of randomly generated problem instances is used.

The obtained results can be helpful construction of new hybrid GAs combining many techniques and providing a better performance in solving OSSP.

In the next section basic OSSP representations and their decoding schemes are presented. Then, in section 3, genetic operations are characterized. Hybridization of GA by means of the two greedy heuristics used for decoding chromosomes is introduced in section 4. In section 5 parallel models of GA are characterized. The main focus is on the migration model of PGA. The experimental results are presented and analyzed in section 6. The last section contains conclusions resulting from the research.

2. Basic representations and decoding schemes of OSSP instances

OSSP schedules can be encoded in chromosomes representing permutations or permutations with repetitions.

The standard benchmark of the OSSP problem with *n* jobs and *m* machines is specified by a integer table T[n,m], $n=m$, where each T_{ij} denotes *j*th operation of the *i*th job.

In permutation representation all operations of all jobs of the problem T are assigned ranks. No specific order is assumed. A feasible schedule S is built out of elementary operations in order of their ranks as they appear in *n*-element input permutation vector X (see Fig.1).

Example 1

Let $n=m=3$. An OSSP instance is given in the table
$T=\{O_{1,1}, O_{1,2}, O_{1,3}, O_{2,1}, O_{2,2}, O_{2,3}, O_{3,1}, O_{3,2}, O_{3,3}\} =$
$=\{2, 3, 5, 1, 2, 4, 3, 5, 2\}$. A chomosome X1 is composed of operations $X1=\{O_{2,1}, O_{3,2}, O_{2,3}, O_{1,1}, O_{3,3}, O_{3,1}, O_{1,2}, O_{1,3}, O_{2,2}\}$ and does correspond to the input permutation
$X=<4, 8, 6, 1, 9, 7, 2, 3, 5>$.

Manuscript received September 5, 2007
Manuscript revised September 20, 2007

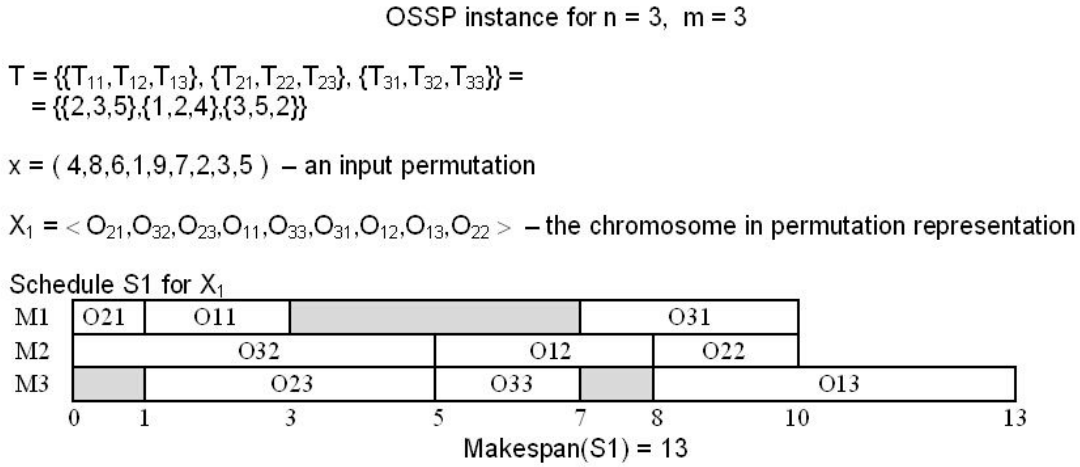

Fig. 1 An OSSP instance in the permutation representation.

The resulting schedule S1 is obtained from the sequence of operations <X1[1], X1[2], ... , X1[9]>: S1={M1, M2, M3}, where: M1=<$O_{2,1}$, $O_{1,1}$, iddle=4, $O_{3,1}$>, M2=<$O_{3,2}$, $O_{1,1}$, $O_{2,2}$> and M3=<iddle=1, $O_{2,3}$, $O_{3,3}$, idle=1, $O_{1,3}$>. Thus, MS(M1)=10, MS(M2)=10 and MS(M3)=13.
Finally, MS(S1)=max(MS(M1), MS(M2), MS(M3))=13.

Another useful encoding scheme is permutation with repetitions. In this case the *nm*-element vector contains *n* job numbers, each number repeated exactly *m* times. Two feasible schedules are built out of a sequence of all elementary job operations, according to the sequence of job (machine) numbers as they appear in the related permutation vector x_i (x_j) - see Fig.2.

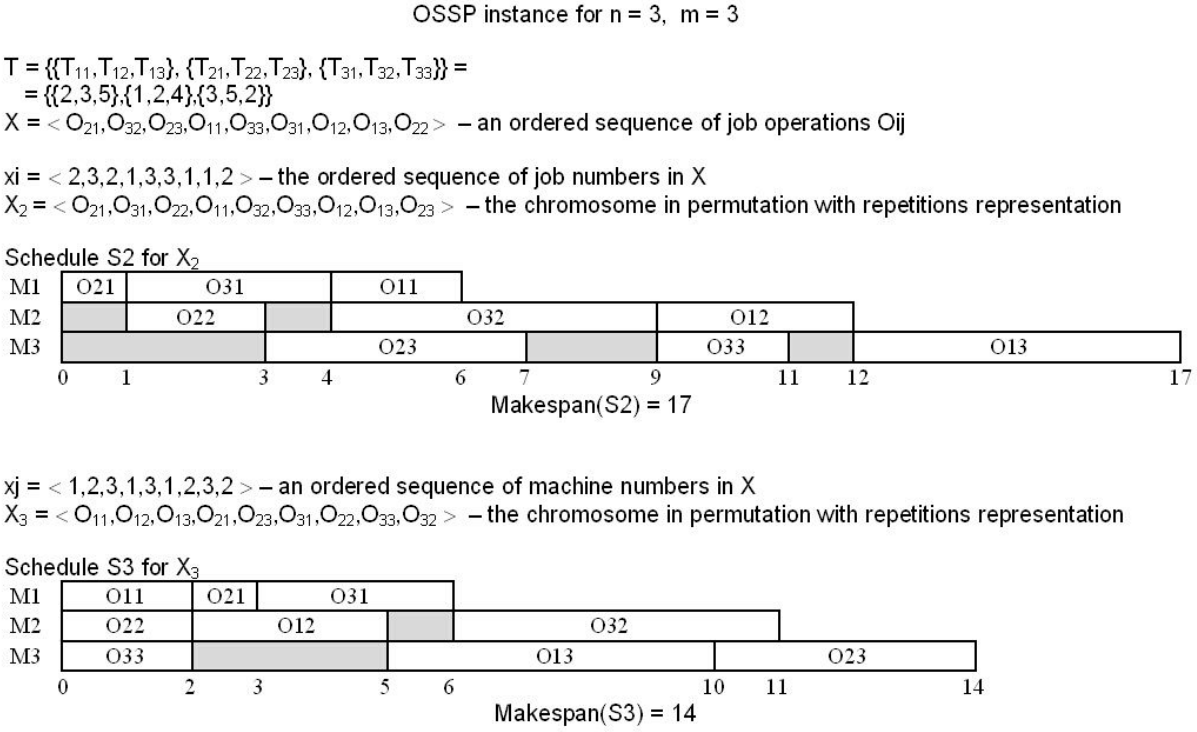

Fig. 2 An OSSP instance in the permutation with repetitions representation.

Example 2

Let n=m=3. An OSSP instance is given in the table

T={{$O_{1,1}$, $O_{1,2}$, $O_{1,3}$ },{$O_{2,1}$, $O_{2,2}$, $O_{2,3}$ },{$O_{3,1}$, $O_{3,2}$, $O_{3,3}$}} = {{2, 3, 5}, {1, 2, 4}, {3, 5, }}.

Chomosomes X2 and X3 in permutation with repetitions representation are derived from a sequence of all job operations, i.e. X={$O_{2,1}$, $O_{3,2}$, $O_{2,3}$, $O_{1,1}$, $O_{3,3}$, $O_{3,1}$, $O_{1,2}$, $O_{1,3}$, $O_{2,2}$} by taking into account only one of the indices {i, j} of $O_{i,j}$, i.e. x2=<2, 3, 2, 1, 3, 3, 1, 1, 2>, where indices of x1 are job numbers in X.

X2=<$O_{2,1}$, $O_{3,1}$, $O_{2,2}$, $O_{1,1}$, $O_{3,2}$, $O_{3,3}$, $O_{1,2}$, $O_{1,3}$, $O_{2,3}$>. x3=<1, 2, 3, 1, 3, 1, 2, 3, 2>, where indices of x2 are machine numbers in X.

X3=<$O_{1,1}$, $O_{1,2}$, $O_{1,3}$, $O_{2,1}$, $O_{2,3}$, $O_{3,1}$, $O_{2,2}$, $O_{3,3}$, $O_{3,2}$}>.

The resulting schedules S2 and S3 are obtained from the sequence of operations <X2[1], X2[2], ... , X2[9]> and <X3[1], X3[2], ... , X3[9]>, respectively.

Hence, S2={M1, M2, M3}, where: M1=<$O_{2,1}$, $O_{3,1}$, $O_{1,1}$>; M2=<iddle=1, $O_{2,2}$, iddle=1, $O_{3,2}$, $O_{1,2}$ > and M3=<iddle=3 , $O_{2,3}$, iddle=2, $O_{3,3}$, iddle=1, $O_{1,3}$>.

Thus, MS(M1)=6, MS(M2)=12 and MS(M3)=17.

Finally, MS(S2)=max(MS(M1), MS(M2), MS(M3))=17.

Similarly, S3={M1, M2, M3}, where: M1=<$O_{1,1}$, $O_{2,1}$, $O_{3,1}$>; M2=<$O_{2,2}$, $O_{1,2}$, iddle=1, $O_{3,2}$> and M3=<O_{3,3}, iddle=3, O_{1,3}, O_{2,3}>.

Thus, MS(M1)=6, MS(M2)=11, MS(M3)=14.

Finally, MS(S3)=max(MS(M1), MS(M2), MS(M3))=14.

3. Genetic Operators for OSSP

In this section we introduce a collection of genetic crossover, mutation and selection operators that are used in our GA.

3.1 LOX Crossover

Linear Order Crossover (LOX) can be used for both OSSP representations presented is section 2 (see Fig.3).

```
procedure: LOX (X1, X2, X3)
begin
  select at random two points dividing X1
  and X2 into 3 subsequences;
  copy the middle subsequence of operations
  from X1 to X3 and delete these operations
  from X2;
  copy the remaining operations of X2 to
  free positions of X3 starting from the
  left and preserving their order in X2;
  output X3;
end
```

Fig.3. The LOX operator for OSSP.

3.2 Mutation operators

Two basic mutation operators are used : Swap (transposition) and Invertion. In Swap mutation positions of the two randomly selected job operations are mutually exchanged.

Inversion mutation is a classical type of mutation that reverse order of the operation subsequence between two randomly selected job operations in the vector representation. The mutation appears in two modes : with constant mutation probability and with variable mutation probability. The last one is defined by selection of three points *p1*, *p2*, and *p3*, where *p1* - start probability related to the first iteration, *p3* - end probability related to the last iteration, and *p2* - middle probability related to a selected iteration between *p1* and *p2* ([*p1,p2*] subrange is given as a fraction of the range [*p1,p3*] in %). Within subranges [*p1,p2*] and [*p2,p3*] the mutation probabilities are changing linearly.

3.3 Cost Function and Selection Operator

In our GAs the classical 2-element Tournament Selection scheme is applied together with eliticist policy. The quality of a solution is measured by the cost function equal to the makespan of the resulting schedule S. The makespan can be determined according to the chosen decoding method as described in section 4.

4. Hybridization techniques

A hybrid genetic algorithm can be obtained when GA metaheuristic is combined with another heuristic method. Two novel greedy heuristics are proposed in this paper for decoding the input chromosome X[*nm*] in the permutation with repetitions representation.

Both heuristics LPT-Task and LPT-Machine are developed on the basis of the Longest Processing Time (LPT) scheduling heuristic proposed by Graham [7]. LPT heuristic was intended for off-line scheduling of tasks in multi-processor systems.

The LPT procedure can be sketched as follows: at any time a processor becomes available for processing schedule an available task with the longest processing time. The method was then generalized for on-line scheduling. LPT was also used for solving some instances of shop scheduling problems [11].

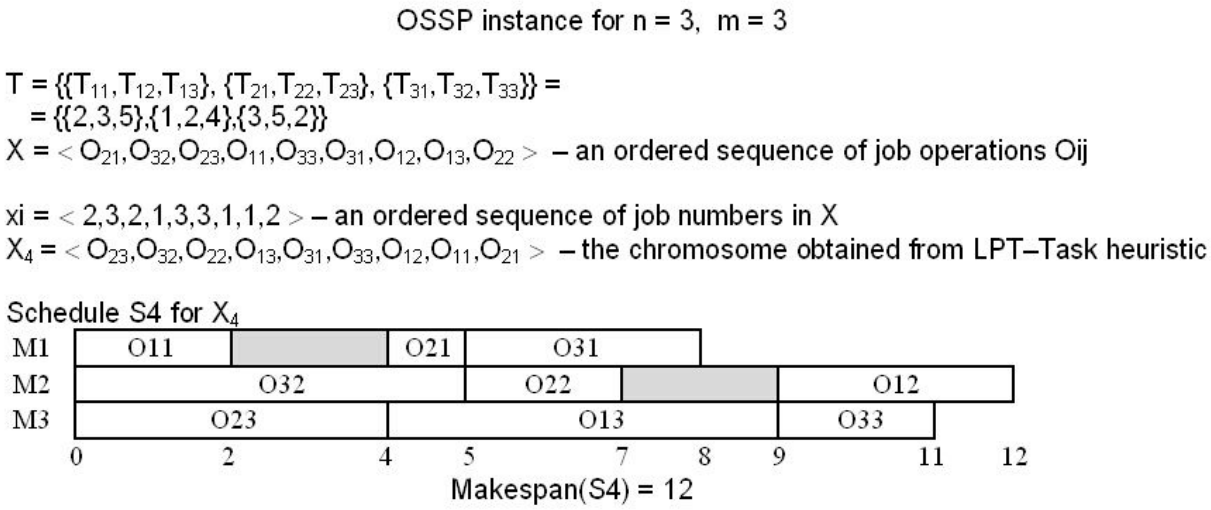

Fig. 5 LPT--Task decoding scheme for the OSSP chromosome

The LPT-Task heuristic is applied in an instance of OSSP problem to operation times for the given job providing that the sequence of job numbers is taken into account (see Fig.4).

```
procedure: LPT-Task (T, X, S)
begin
  scan the chromosome X from the position
  ind:=1 to nm;
  read job index i of the operation Oi,j
  on the position ind;
  choose the next unscheduled operation of
  the job i with maximum processing time
  Ti,k;
  scan the schedule S for the machine k and
  put the operation Oi,k in the first
  feasible place from the left;
  mark the operation Oi,k as scheduled;
end
```

Fig. 4 LPT--Task heuristic.

An application of the LPT-Task heuristic is shown in Fig.5.

Example 3

Let n=m=3. An OSSP instance is given in the table
T={{$O_{1,1}$, $O_{1,2}$, $O_{1,3}$},{$O_{2,1}$, $O_{2,2}$, $O_{2,3}$}, {$O_{3,1}$, $O_{3,2}$, $O_{3,3}$}}=
={{2, 3, 5}, {1, 2, 4}, {3, 5, 2}}.
The chomosome X4 in permutation with repetitions representation is derived from a sequence of all job operations, i.e. X=<$O_{2,1}$, $O_{3,2}$, $O_{2,3}$, $O_{1,1}$, $O_{3,3}$, $O_{3,1}$, $O_{1,2}$, $O_{1,3}$, $O_{2,2}$>
by taking into account only one of the indices of $O_{i,j}$, i.e.
xi=<2, 3, 2, 1, 3, 3, 1, 1, 2>, where elements of xi are job numbers in X, as well as the LPT order of operations in each job.
Thus, X4=<$O_{2,3}$, $O_{3,2}$, $O_{2,2}$, $O_{1,3}$, $O_{3,1}$, $O_{3,3}$, $O_{1,2}$, $O_{1,1}$, $O_{2,1}$>.
The resulting schedule S4 is obtained from the sequence of operations <X4[1], X4[2], ... , X4[9]> :
S4={M1, M2, M3}, where: M1=<$O_{1,1}$, 2, $O_{2,1}$, $O_{3,1}$}>, M2=<$O_{3,2}$, $O_{2,2}$, 2, $O_{1,2}$> and M3=<$O_{2,3}$, $O_{1,3}$, $O_{3,3}$}>.
Thus, MS(M1)=8, MS(M2)=12, MS(M3)=11.
Finally, MS(S)= max (MS(M1), MS(M2), MS(M3))=12.

The LPT-Machine heuristic is applied to operation times for the given machine providing that the sequence of machine numbers is considered (see Fig.6).

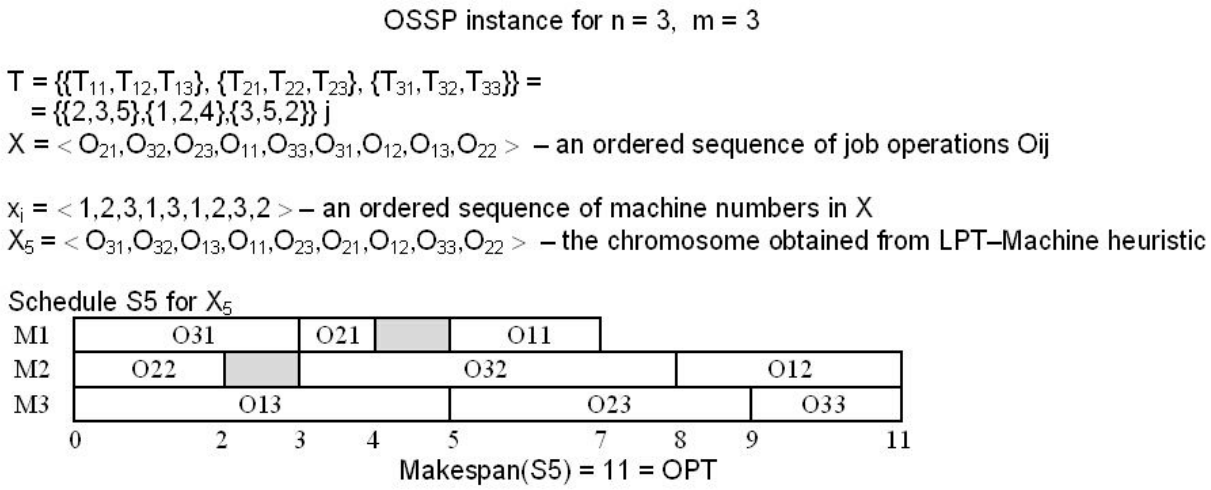

Fig. 7 LPT-Machine decoding scheme for the OSSP chromosome.

```
procedure: LPT-Machine (T, X, S)
begin
  scan the chromosome X from the position
  ind:=1 to nm;
  read machine index j of the operation
  Oi,j on the position ind;
  choose the next unscheduled operation of
  the job k with maximum processing time
  Tk,j;
  scan the schedule for the machine j and
  put the operation Ok,j in the first
  feasible place from the left;
  mark the operation Ok,j as scheduled;
end
```

Fig. 6 LPT--Machine heuristic.

An application of the LPT-Machine heuristic is shown in Fig.7.

Example 4

Let n=m=3. An OSSP instance is given in the table

$T=\{\{O_{1,1}, O_{1,2}, O_{1,3}\}, \{O_{2,1}, O_{2,2}, O_{2,3}\}, \{O_{3,1}, O_{3,2}, O_{3,3}\}\} = \{\{2, 3, 5\}, \{1, 2, 4\}, \{3, 5, 2\}\}$.

The chomosomes X4 in permutation with repetitions representation is derived from a sequence of all job operations, $X=<O_{2,1}, O_{3,2}, O_{2,3}, O_{1,1}, O_{3,3}, O_{3,1}, O_{1,2}, O_{1,3}, O_{2,2}>$ by taking into account only one index of $O_{i,j}$, i.e.

xj=<1, 2, 3, 1, 3, 1, 2, 3, 2>, where indices of xj are machine numbers in X, as well as the LPT order of ith operations on each machine.

Thus, $X5=<O_{3,1}, O_{3,2}, O_{1,3}, O_{1,1}, O_{2,3}, O_{2,1}, O_{1,2}, O_{3,3}, O_{2,2}>$.

The resulting schedule S5 is obtained from the sequence of operations <X5[1], X5[2], ... , X5[9]> : S5={M1, M2, M3}, where: $M1=<O_{3,1}, O_{2,1}, 1, O_{1,1}>$, $M2=<O_{2,2}, 1, O_{3,2}, O_{1,2}>$ and $M3=<O_{1,3}, O_{2,3}, O_{3,3}>$. Thus, MS(M1)=7, MS(M2)=11 and MS(M3)=11. Finally, MS(S5)= max (MS(M1), MS(M2), MS(M3))=11. The obtained makespan is optimal.

5. Models of Parallel Genetic Algorithms

There are many models of parallelism in evolutionary algorithms: master-slave PGA, migration based PGA, diffusion based PGA, PGA with overlapping subpopulations, population learning algorithm, hybrid models etc.

The above models are characterized by the following criteria:

- number of populations : one, many;
- population types : disjoint, overlapping;
- population topologies : various graph models;
- interaction model : isolation, migration, diffusion;
- recombination, evaluation of individuals, selection : distributed/local, centralized/global;
- synchronization on iteration level: synchronous/asynchronous algorithm.

The most common models of PGA are:

- master-slave : one global population, global genetic operations, fitness functions computed by slave processors);
- massively parallel (cellular): static overlapping subpopulations with a local structure, local genetic operations and evaluation;
- migration (with island as a submodel): static disjoint subpopulations/islands, local genetic operations and migration;
- hybrid : combination of one model on the upper level and other model on the lower level (the speedup achieved in hybrid models is equal to product of level speedups).

5.1 Migration Model of Parallel Genetic Algorithm

Migration models of PGAs consist of a finite number of disjoint subpopulations that evolve in parallel on their "islands" and only occasionally exchange genetic information under control of a migration operator. Migration models of PGAs consist of a finite number of subpopulations that evolve in parallel on their "islands" and exchange the genetic information under the control of a migration operator. Co-evolving subpopulations are built of individuals of the same type and are ruled by one adaptation function. The selection process is decentralized.

In our model the migration is performed on a regular basis. During the migration phase every island sends its representatives (emigrants) to all other islands and receives the representatives (immigrants) from all co--evolving subpopulations. This topology of migration reflects so called "pure" island model. The migration process is fully characterized by migration size, distance between populations and migration scheme. Migration size determines the emigrant fraction of each population. This parameter is limited by capacity of islands to accept immigrants. The distance between migrations determines how often the migration phase of the algorithm occurs. Migration of best individuals is applied.

```
procedure:
genetic algorithm for subpopulation
begin
  iteration counter t = 0;
  initialization of subpopulation Pt;
  evaluation of Pt;
  while (not termination condition) do
    begin
      parental population Tt = selection
      from Pt;
      offspring population Ot = crossover
      and mutation on Tt;
      evaluation of {Pt or Ot};
      Pt+1 = selection from {P t or Ot};
      if (migration condition) then
        migration of representatives of
        Pt+1 to all other subpopulations
      t = t + 1;
    end;
end
```

Fig. 8 Genetic algorithm for a subpopulation in the migration model.

Genetic algorithm performed in parallel for each subpopulation is shown in Fig.8.

In our algorithm a specific model of migration is applied in which islands use two copies of genetic information: migrating individuals still remain members of their original subpopulation. In other words they receive new "membership" without losing the former one. Incoming individuals replace the chromosomes of host subpopulation at random. Then, a selection process is performed. The rationale behind such a model is as follows. Even if the best chromosomes of host subpopulation are eliminated they shall survive on other islands where their copies were sent. On the other hand any eliticist scheme or preselection applied to the replacement phase leads to premature elimination of worse individuals and lowers the overall diversity of subpopulation.

6. Experimental Verification

In our computer program *GA_for_OSSP* two basic models of the genetic algorithm are implemented: conventional GA and PGA in migration model. It is possible to set up most parameters of evolution, monitor evolution process and measure makespan, the number of generations and time of computations. The program generates detailed reports and basic statistics.

Computations were performed with the following parameters of GA determined in an initial series of experiments: *population_size* = 300 , *crossover* = LOX, *crossover_probability* = 0.75, *mutation* = Swap with *constant_mutation_probability* = 0.3 or Swap/Inverse (in equal proportions) with *variable mutation_probability* defined by selection of three points *p1*=0.4, *p2*=0.2, *p3*=0.1 and position of *p2* at exactly 500 iterations.
In migration based PGA the following settings were made: *number_of_islands* = 3, with *population_size* = 100 on every island, migration of best individual with *migration_rate* = 25. All experiments were repeated 10 or 30 times with the *iteration_number* = 500 or 1000. The initial experiments confirmed also that combination of LOX and SWAP operators is superior with respect to makespan [22].

For computer experiments we used four large instances of OSSP with n=m= 25, 30, 40, 50 and random integer operation times selected at random from the integer range [1, …, 100] or [1, …, 500] with uniform probability (available from [27]). The instation sizes were determined experimentally. They significantly excede the maximum size of benchmarks available for OSSP problem. Therefore, we do not report comparison of our hybrid algorithms to other methods known from the literature [5,17].

In the first experiment the efficiency of 5 genetic algorithm configurations was compared for the problem size *n*=*m*=20 (25), operation times from the range [1, … , 100], population size 200 (300), 1000 iterations per experiment, 30-20 runs. The obtained results are presented in Table 1.

Table 1: Hybrid GA for random Open-Shop Scheduling Problem, n=m=20,25.

OSSP size	heuristic	mutation	makespan				no. iter.	time [s]
			min	max	avg	stddev	avg	avg
20	none	constant	1307	1333	1320.3	6.73	688.3	103.9
	LPT-Task	constant	1249	1271	1255.7	5.33	475.0	109.3
	LPT-Task	variable S-I	1249	1273	1256.7	5.17	373.8	82.1
	LPT-Machine	constant	**1248** (3/30)	1261	1253.1	3.42	325.6	75.2
	LPT-Machine	variable S-I	**1248** (1/30)	1261	1254.0	3.53	456.7	99.4
25	none	constant	1610	1641	1623.5	9.08	709.7	426.9
	LPT-Task	constant	1541	1554	1545.8	3.55	371.2	329.0
	LPT-Task	variable S-I	1543	1554	1546.9	2.73	497.7	422.0
	LPT-Machine	constant	**1540** (6/20)	1551	1542.7	3.15	314.8	268.1
	LPT-Machine	variable S-I	**1540** (8/20)	1548	1541.5	2.42	297.0	247.1

Table 2: Hybrid parallel GA for random Open-Shop Scheduling Problem, n=m=20,25.

OSSP size	heuristic	mutation	makespan				no. iter.	time [s]
			min	max	avg	stddev	avg	avg
20	none	constant	1295	1333	1320.9	14.71	668.3	105.5
	LPT-Task	constant	1249	1267	1256.5	5.09	335.5	77.1
	LPT-Task	variable S-I	1250	1267	1257.7	4.80	262.8	58.9
	LPT-Machine	constant	1249	1266	1256.0	3.81	368.3	82.1
	LPT-Machine	variable S-I	**1248** (2/30)	1267	1254.9	4.87	423.5	94.0
25	none	constant	1625	1659	1642.7	10.10	697.3	267.9
	LPT-Task	constant	1543	1554	1548.5	3.14	403.4	232.7
	LPT-Task	variable S-I	1541	1556	1546.4	3.79	404.4	338.1
	LPT-Machine	constant	**1540** (11/20)	1547	1541.5	2.48	412.0	348.9
	LPT-Machine	variable S-I	**1540** (7/20)	1548	1541.7	2.43	244.3	200.9

In the second experiment the efficiency of 5 parallel genetic algorithm configurations was compared for the problem size n=m=20 (25), operation times from the range [1, … , 100], population size 3×67 (3×100), 1000 iterations per experiment, 30-20 runs. The obtained results are presented in Table 2.

In the third experiment the efficiency of algorithm configurations was compared for the problem size $n = m$ = 30, 40, 50 and operation times from the range [1, … , 500], population size = 300 or 3×100, 1000 iterations per experiment, 10 runs. The obtained results are presented in Table 3.

All computer experiments were performed on a HP Pavilion computer with Pentium 4 processor (3.06 GHz) and 1 GB RAM.

Analysis of the obtained results justify detailed several conclusions.

At first PGA is better then GA when no hybridization is provided. Two low level hybridization techniques LPT-Task and LPT-Machine improve efficiency of the results for both GA and PGA.

LPT-Machine heuristic outperforms LPT--Task in terms of minimal makespan, average makespan and standard deviation. In general LPT-Machine heuristic works better with conventional GA then with PGA. The parallelization of GA does not reveal any advantages in our experiments although PGA is very efficient for solving other hard problems like Graph Coloring Problem (GCP).

Table 3: Hybrid GA for random Open-Shop Scheduling Problem, n=m=30,40,50.

OSSP size	heuristic	mutation	makespan				no. iter.	time [s]
			min	max	avg	stddev	avg	avg
30	LPT-Machine	constant	8802	8918	572.2	36.6	572.2	983.0
		variable S-I	8813	8871	691.8	18.8	691.8	1171.4
	LPT-Task	constant	8861	8923	752.8	23.1	752.8	1335.8
		variable S-I	8842	8911	633.8	22.6	633.8	1102.9
40	LPT-Machine	constant	**12445**(10/10)	12445	11.0	0.0	11.0	65.5
		variable S-I	**12445**(10/10)	12445	11.2	0.0	11.2	65.6
	LPT-Task	constant	**12445** (6/10)	12451	72.4	2.1	72.4	450.0
		variable S-I	**12445** (6/10)	12455	71.4	4.1	71.4	435.8
50	LPT-Machine	constant	**15280**(10/10)	15345	22.2	0.0	22.2	427.1
		variable S-I	**15280**(10/10)	15364	20.4	0.0	20.4	325.0
	LPT-Task	constant	15328	15370	71.2	13.5	71.2	1188.3
		variable S-I	15328	15318	73.2	13.9	73.2	1184.6

Table 4: Hybrid parallel GA for random Open-Shop Scheduling Problem, n=m=30,40,50.

OSSP size	heuristic	mutation	makespan				no. iter.	time [s]
			min	max	avg	stddev	avg	avg
30	LPT-Machine	constant	**8800** (1/10)	8862	8830.7	18.9	676.1	1197.5
		variable S-I	**8800** (2/10)	8867	8821.1	22.0	432.0	755.0
	LPT-Task	constant	8851	8914	8883.8	22.3	674.6	1249.6
		variable S-I	8851	8909	8872.7	19.1	634.3	1145.6
40	LPT-Machine	constant	**12445**(10/10)	**12445**	**12445**	0.0	11.2	66.9
		variable S-I	**12445**(10/10)	**12445**	**12445**	0.0	11.3	78.6
	LPT-Task	constant	**12445** (1/10)	12463	12451.4	5.9	73.8	456.4
		variable S-I	**12445** (5/10)	12454	12447.6	3.3	89.7	546.4
50	LPT-Machine	constant	**15280**(10/10)	**15280**	**15280**	0.0	20.2	393.2
		variable S-I	**15280**(10/10)	**15280**	**15280**	0.0	23.2	369.7
	LPT-Task	constant	15312	15345	15329.6	11.6	77.1	1282.9
		variable S-I	15314	15364	15341.6	18.0	79.1	1328.3

In the last experiment the efficiency of 4 best algorithm configurations was compared for the problem size n=m=30, operation times from the range [1, …,500], population size = 300 or 3×100, 500 iterations per experiment, 10 runs. The obtained results are presented in Table 4.

7. Conclusions

In the paper we proved by computer simulation that hybrid genetic algorithms with LPT-Machine heuristic can be efficiently used for the class of large OSSP problems. The results presented in this paper encourage further research in this area. One interesting possibility would be combination of the LPT-Machine heuristic with other efficient techniques proposed recently in hybrid algorithms for solving smaller but harder OSSP benchmarks by by Taillard [24], Guéret and Prins [25], Brücker [26] et al. The search for new representations, their decoding techniques as well as new hybrid metaheuristics for solving large OSSP instances still remains an open question.

Acknowledgments

This work was supported in part by the research grant No. E-3/112/BW/07 from Cracow University Technology.

References

[1] Alba E. (Ed.): Parallel Metaheuristics. A new class of algorithms, John Wiley & Sons, New York (2005)

[2] [Alba E., Tomasini M.: Parallelism and evolutionary algorithms, IEEE Trans. Evol. Comput. Vol.6 (2002) No.5, 443-462

[3] Bäck T.: Evolutionary algorithms in theory and practice, Oxford U. Press (1996)

[4] Cantú-Paz, E.: Efficient and accurate parallel genetic algorithms, Kluwer (2000)

[5] Colak S., Agarwal A.: Non-gready heuristics and augmented neural networks for the open shop scheduling problem, Naval Research Logistics 52(2005) 631-644

[6] Feng H-L., Ross P., Corne D.: A promising genetic algorithm approach to job-shop scheduling, rescheduling and open shop scheduling problems, Proc. Fifth Int. Conf. on Genetic Algorithms, Morgan Kaufmann (1993) 375-382

[7] Graham R.L.: Bounds on multiprocessing timing anomalies, SIAM Journal on Applied Mathematics 17 (1969) 416-429

[8] Guéret C., Jussien N.: Combining AI/OR techniques for solving Open Shop problems, CP--AI--OR'99 (1999) 25-26

[9] Guéret C., Prins C.: A new lower bound for the Open--Shop problem, Annals of Operations Research 92 (1999) 165-183

[10] Guéret C., Prins C.: Forbidden intervals for open--shop problems, 15th IFORS'99 (1999)

[11] Hong T.-P., Huang P.-Y., Horng G.: Using the LPT and Palmer approaches to solve group flexible flow-shop problems, Int. J. Computer Science and Network Security 6 (2006) 98-104

[12] Kokosiński Z., Kołodziej M., Kwarciany K.: Parallel genetic algorithm for graph coloring problem, Proc. ICCS'2004, LNCS 3036 (2004) 215-222

[13] Kokosiński, Z., Kwarciany, K., Kołodziej, M.: Efficient graph coloring with parallel genetic algorithms, Computing and Informatics 24 (2005) 109-121

[14] Kokosiński, Z.: Effects of versatile crossover and mutation operators on evolutionary search in partition and permutation problems, Proc. IIS:IIPWM'05, [in:] Advances in Soft Computing , Springer (2005) 299-308.

[15] Kokosiński Z., Kwarciany K.: On sum coloring of graphs with parallel genetic algorithms, Proc. ICANNGA'2007, LNCS 4431 (2007) 211-219

[16] Khuri S., Miryala S.R.: Genetic algorithms for solving open shop scheduling problem, Proc. EPIA'1999, LNCS 1695 (1999) 357-368

[17] Liaw C-F.: A hybrid genetic algorithm for the open shop scheduling problem, European Journal of Operational Research 124 (2000) 28-42

[18] Louis S.J., Xu Z.: Genetic algorithms for open shop scheduling and re--scheduling, 11th ISCA'96 (1996) 99-102

[19] van Otterloo S.: Evolutionary algorithms and scheduling problems, M.S, thesis, University of Utrecht (2002)

[20] Prins C.: Competitive genetic algorithm for the open--shop scheduling problem, Mathematical Methods of Operations Research 52 (2000) 389-411

[21] Puente J., Diez H.R., Varela R., Vela C.R., Hidalgo L.P.: Heuristic rules and genetic algorithms for open shop scheduling problem, Proc. CAEPIA'2003, LNCS 3040 (2004) 394-403

[22] Studzienny Ł.: Parallel evolutionary algorithm for solving open-shop scheduling problem, M.S. Thesis, Faculty of Electrical and Computer Eng., Cracow University of Technology (2005)

[23] Taillard E.: Benchmarks for basic scheduling problems, European Journal of Operational Research 64 (1993) 278-285

[24]

http://ina2.eivd.ch/Collaborateurs/etd/default.htm

[25]

http://www.emn.fr/x-uto/gueret/OpenShop/OpenShop.html

[26]

http://mathematic.uni-osnabueck.de/research/OR/software.shtml

[27]

http://www.pk.edu.pl/~zk/OSSP/ossp-random-instances.zip

Zbigniew Kokosiński received his M.S. degree in 1982 from Cracow University of Technology, Kraków, Poland. In 1992 he received Ph.D. degree with distinction in Computer Science from the Gdańsk University of Technology, Gdańsk, Poland. In 1994-1997 he was employed as an Assistant Professor at the Department of Computer Software, University of Aizu, Aizu-Wakamatsu, Japan. Currently, dr. Kokosiński is an Assistant Professor at the Dept. of Automatic Control, Faculty of Electrical and Computer Engineering, Cracow University of Technology, Kraków. His research is focused on combinatorial optimization and parallel metaheuristics, generation of combinatorial objects in parallel, associative processors and algorithms, programmable devices and systems. The publications include over 30 refered papers in international scientific journals and conference proceedings. In the past years he was a member of ACM, IEEE Computer Society and IASTED.

Łukasz Studzienny graduated from Cracow University of Technology (Politechnika Krakowska), Kraków, Poland in 2005, where he received M.Sc. degree in electrical engineering. Currently he works as a software engineer in IBM Poland, Kraków. His professional interests include computer programming, combinatorial optimization, artifficial intelligence and automatic control.