ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1998
Seria: AUTOMATYKA z. 125 Nr kol. 1391

Konrad WALA
Akademia Górniczo-Hutnicza, Kraków

EVOLUTIONARY ALGORITHM FOR THE CIM PRODUCTION SCHEDULING*

Summary. The paper presents the evolutionary algorithm, which realizes the original evolutionary search process, for optimization of the NP-hard permutational scheduling problems. The investigated algorithm is a hybrid of a modified genetic algorithm, simplified local optimization procedure, some 'tabu mechanism', as well as a simple self-adaptation algorithm parameters procedure. The computational results show, for the example of m-stages production flow line, that the proposed algorithm has a high potential as a optimization paradigm for the CIM production scheduling.

ALGORYTM EWOLUCYJNY HARMONOGRAMOWANIA PROCESÓW WYTWARZANIA W SYSTEMACH KOMPUTEROWO ZINTEGROWANYCH

Streszczenie. W pracy zaprezentowano oryginalny algorytm ewolucyjny przeznaczony do optymalizacji NP-trudnych zagadnień permutacyjnych. Badany algorytm jest hybrydą zmodyfikowanego algorytmu genetycznego, uproszczonej procedury optymalizacji lokalnej, pewnego 'mechanizmu tabu' oraz prostej procedury autoadaptacji niektórych parametrów algorytmu. Wyniki obliczeń komputerowych, wykonane dla m-stadialnej linii przepływowej wskazują, że proponowany algorytm z powodzeniem może być stosowany do optymalizacji harmonogramów wytwarzania w systemach komputerowo zintegrowanych.

1. Introduction

Computer Integrated Manufacturing (CIM) has been attracting many researchers for the past 20 years where particularly the use of AI techniques for problem solution search have been actively studied. The aim of the CIM is to make a computerized environment in which design, decision support for production management, manufacturing and marketing are integrated and this way to reduce the manufacturing costs and decrease the product design-to-market time in discrete parts manufacturing. In the field of CIM production, scheduling is intended to allocate the manufactory resources to the production orders to be processed. Due to the time complexity, exponentially increasing with the problem size, of the prevalent number of scheduling problems conventional optimization methods such as Branch & Bound

* Research supported by Polish Committee for Research, grant 11.120.227 (AGH, Kraków)

can accurately solve usually very small scheduling problems. Approximate methods include priority rules, Monte Carlo methods and neighborhood search. The drawback of these enumerative procedures is that each of the trial solution can only be examined as a whole in respect to their suitability. In case of scheduling problems the search space is so big that during the time period relevant to the practical application only a small part of the search space can be examined. Algorithms which combine elements of the neighborhood search and Monte Carlo method are the genetic (GA) and evolutionary (EA) algorithm.

GA and *EA* algorithms, as one of the promising tools, have been applied for solution of the flow shop scheduling, job shop scheduling, packing, routing, etc. These algorithms are powerful search techniques taking inspiration from genetics and natural selection [2]. They can effectively explore very large solutions spaces without being trapped by local optima, where particularly the EAs are the main representative of a class of algorithms based on the model of natural evolution. The one requirements for applying *G/EAs* to the problem are: solution representation of the problem to be solved, a set of pseudo-genetic operators called briefly genetic operators and a evaluation function of the problem solutions called fitness function.

Numerous class of scheduling problems, too the flow shop scheduling, can be formalized as permutation problems where the solution has form of the tasks set permutation. Permutation problems are an important problem class in the combinatorial optimization domain. Standard instances of permutation problems, besides scheduling, include travelling salesman problem, quadratic assignment problem, graph coloring, as well as variety of design problems. NP-hard permutation problems even with modest number of tasks overwhelm the capabilities of algorithms that guarantee solutions optimum in reasonable amount of time. That is the reason why effective approximate algorithms producing good suboptimal solutions proved important.

There is an agreement in the combinatorial optimization community that the binary representation of the permutation will not provide any advantage because binary representation would require special repair algorithms, since a change of a single bit may result in a illegal solution of the problem, i.e. a sequence which is not a permutation. Thus we apply in investigated *EA* permutation $\Pi = (\Pi(1), \Pi(2)...,\Pi(n))$ of the tasks set $J = \{1, 2,..., n\}$ as the natural representation of the scheduling problem as well as objective function $f(\Pi)$ as the fitness function.

Investigated evolutionary algorithm EVOL-1, adjusted for permutation problems, is a hybrid of an modified genetic algorithm, simplified local optimization procedure, some 'tabu

mechanism' as well as simple self-adaptation algorithm parameters procedure. The computation results have shown that the proposed evolutionary algorithm has a high potential for the CIM production scheduling.

2. The evolutionary search process

The knowledge, accumulated in the search process for the problem solution, is encoded as a set P of permutations called population, where $M = |P|$ is the size of the population P. The evolution process of the population P, modelled by *EA*, realizes the search process by use of genetic operators. Every genetic operator generates new permutation(s) called offspring on the basis of old permutation(s) called parent(s).

EVOL-1 algorithm, described in [4, 5] where also the 'philosophy' of the search process is widely presented, carries into effect the evolutionary search process organized like in the class of Steady State *GA*s [2], where in course of one iteration initially one crossover operator is randomly chosen and then two permutations-parents are selected from the population P and processed. We assume that the population set P is linearly ordered in accordance with the solution's fitness by means of objective function $f(\Pi)$: the best population permutation ($\Pi_{best} = \arg\min \{ f(\Pi): \Pi \in P\}$) has number 1, ..., the worst one ($\Pi_{worst} = \arg\max \{f(\Pi): \Pi \in P\}$) has number $M = |P|$. Besides, to simplify genetic search process and save the computation time, the parents are selected from the population P by means of random_uniform sampling mechanism and generated offspring are inserted between other population P permutations in accordance with its fitness $f(\Pi)$.

Let us notice that the permutation Π, as a solution of the problem, presents by itself the ordering information only. Thus we need to use genetic operators which permit the ordering processing. As a crossover operators we use three classical called PMX (partially matched crossover operator), OX (order crossover operator), CX (cycle crossover operator); (for details see, [2].) We use also three unary operators: two mutations (M1, M2) and simplified local optimization procedure LO which has the status of genetic operator used for local tuning of the best population permutation, i.e. permutation from items $[1, \mu]$ of the ordered set P. The mutation operators M1 and M2, as an insurance policy against the loss of genetic material diversity, are used for mutation of the permutations from items $[\mu + 1, M]$ of set P only. These operators do not change the values of the permutation but the order of elements only in this way.

Operator M1: Two elements of the permutation ($\Pi(1)$, ..., $\Pi(i-1)$, $\Pi(i)$, ..., $\Pi(j-1)$, $\Pi(j)$, ..., $\Pi(n)$) are picked at random along the permutation, say $\Pi(i)$ and $\Pi(j)$, and swapped resulting in another permutation ($\Pi(1)$, ..., $\Pi(i-1)$, $\Pi(j)$, ..., $\Pi(j-1)$, $\Pi(i)$, ..., $\Pi(n)$).

Operator M2: Two elements of the permutation ($\Pi(1)$, ..., $\Pi(i-1)$, $\Pi(i)$, $\Pi(i+1)$, ..., $\Pi(j-1)$, $\Pi(j)$, ..., $\Pi(n)$) are picked at random along the permutation, say $\Pi(i)$ and $\Pi(j)$, and element $\Pi(i)$ skips over to j-position of the permutation as well as $\Pi(i+1)$, ..., $\Pi(j-1)$, $\Pi(j)$ elements are shifted to i-position resulting in permutation ($\Pi(1)$, ..., $\Pi(i-1)$, $\Pi(i+1)$, ..., $\Pi(j-1)$, $\Pi(j)$, $\Pi(i)$, ..., $\Pi(n)$).

Simplified **local optimization operator LO** searches for better permutation than Π = ($\Pi(1)$, ..., $\Pi(i-1)$, $\Pi(i)$, ..., $\Pi(j-1)$, $\Pi(j)$, ..., $\Pi(n)$) in its neighborhood $S(\Pi)$ = { Π': Π'= ($\Pi(1)$, ..., $\Pi(i-1)$, $\Pi(j)$, ..., $\Pi(j-1)$, $\Pi(i)$, ..., $\Pi(n)$) } only.

Algorithm: EVOL-2

To determine the approximate solution Π_{approx} do the following.

Step 1. Initialization.

Generate randomly M permutations Π as well as compute objective function $f(\Pi)$ for each solution. Set up the initial population P ordering the generated permutations by function $f(\Pi)$ value so that the first permutation, permutation No.1, in population P is the best one (i.e., $\Pi_{best} = arg\ min\ \{f(\Pi): \Pi \in P\}$), ..., and the last permutation, permutation No.M, is the worst one (i.e., $\Pi_{worst} = arg\ max\ \{f(\Pi): \Pi \in P\}$). Set k:= 1, i:= 1, where k is the number of train iteration and i is the number of the train.

Step 2. Crossover

Choose randomly one crossover operator from the set $\{PMX, OX, CX\}$, where selection probabilities of the operators are: p_{PMX}, p_{OX}, $p_{CX} = 1 - p_{PMX} - p_{OX}$ as well as select randomly, with uniform distribution, and copy two permutation- parents from the population P . Using chosen crossover operator generate two offspring Π^1, Π^2 of the parents and insert them between other population P permutations in accordance with function $f(\Pi^j)$ value, (j = 1, 2). Remove two worst (last) permutations from the population P.

Step 3. Mutation and local optimization

Sample randomly number α, $\alpha \in [0, 1)$. If $\alpha < p_{M1}$ then chose one permutation from the items $[\mu + 1, M]$ of the population P, modify it by the use of mutation operator M1 and insert between other population solutions in accordance with the value of fitness function $f(\Pi.)$. If $\alpha < p_{M2}$ then chose one permutation from the items $[\mu + 1, M]$ of the population P, modify it by the use of mutation operator M2 and insert between other population solutions in accordance with the value of fitness function $f(\Pi)$.

If $\alpha < p_{LO}$ then chose one permutation from the items $[1, \mu]$ of the population P, improve it by the use of local optimization operator LO and insert between other population permutations in accordance with the value of function f(.).

Step 4.The end of train?

Set k:= k+1 and if $k \leq K$ then go to Step 2.

Step 5. The end of evolution search?

Set i:= i +1 and if one of termination criterion is not satisfied, i.e. $i \leq I$ and computation time < T, then go to Step 6, otherwise 'STOP": the approximate solution $\Pi_{approx} = \Pi_{best} = \arg\min \{f(\Pi): \Pi \in P\}$.

Step 6. Algorithm parameters modification

Determine new values of genetic operator selection probabilities: $p_{M1} := g_1(i)$, $p_{M2} := g_2(i)$, $p_{LO} := g_3(i)$, $p_l := p_l (1 + \beta e_l / d_l)/\gamma$ for $l \in \{PMX, OX, CX\}$, where $g_1(i)$, $g_2(i)$, $g_3(i)$ ($0 \leq g_1(i), g_2(i), g_3(i) \leq 1$) are the given functions of train numbers, β ($\beta \leq 1$) is the algorithm parameter determining the rate change of crossover operator selection probabilities, e_l is the number of train iteration where application of l-crossover operator gives better offspring then the better of parents, d_l is the number of train iteration of l-crossover operator application and $\gamma = \Sigma_{l=1/3} p_l (1 + \beta e_l / d_l)$ is the normalization coefficient of the crossover operator selection probabilities.

Let us notice that the population size M, population split number μ, maximum numbers of train iteration K and of train I as well as the maximum computation time T, initial genetic operator selection probabilities p_{PMX}, p_{OX}, p_{CX}, coefficient β are EVOL-1 algorithm parameters. It is assumed that functions g_l have form $g_l = a_l + b_l i$ ($l = 1, 2, 3$), thus the numbers $a_1, a_2, a_3, b_1, b_2, b_3$ are also EVOL-1 parameters.

3. Flow line scheduling problem

One of the approaches, called classical permutation flow shop, consists in considering the total flow shop as a complex single machine for which only the sequence in which the tasks are loaded into the first machine is scheduled. Let us take into consideration the following extension of the classical permutation flow shop problem to the case of flow line scheduling. There is a set $J = \{1,..., j, ...,n\}$ of production tasks (jobs, parts) and m-stages flow line where each of i-stage, i = 1,2, ..., m, has a set of m_i, $m_i \geq 1$, parallel identical machines (see, Fig.1). A task is associated with a sequence of at most m operations processed at successive stages, and all parts flow through the stages in the same order, where p_{ij} denotes processing time at i-stage for task j . We want to find a schedule that minimizes the makespan $Cmax = \max \{Cj: j \in J\}$ or the sum $\Sigma Cj = \Sigma_{j \in J} Cj$, where Cj is the completion time of task j . It is assumed that the system buffers have enough large capacity and in case $p_{ij} = 0$ the task j skip the stage i of production line. This kind of flow line scheduling problem appears in many CIM manufacturing processes. Exact and approximate algorithms as well as industry applications of the flow line scheduling are reviewed in [1]. An interested improvement algorithm of tabu search type for makespan minimization is presented in [3]. Below, we

present some computational results of EVOL-1 algorithm application for illustrative example of flow line scheduling problem where we assume that $f(\Pi) = Cmax$ or ΣCj.

In case we assume that the tasks permutation Π determines the tasks assignment order to the earliest free machines then the schedule and objective function values Cmax and ΣCj are calculated by the following SCHEDULE, of time complexity $O(n\ m)$, procedure.

Procedure SCHEDULE:

Input: tasks permutation $\Pi = (\Pi(1), \Pi(2), ..., \Pi(j), ..., \Pi(n))$;

STEP 1. Set $T(i, k) := 0$ for $i = 1$ to m and $k = 1$ to m_i, where $T(i, k)$ is the up-to-date completion time of tasks assigned to machine k of stage i.

STEP 2. For $j = 1$ to n do:

determine the earliest free machine of 1-stage

$k = \arg\min \{ T(1, v): v \in \{1, 2, ..., m_1\} \}$ and set

$T(1, k) := T(1, k) + p_{1\Pi(j)}$, $C(1, \Pi(j)) := T(1, k)$;

for $i = 2$ to m do:

determine the earliest free machine of i-stage

$k = \arg\min \{ \max\{T(i, v), C(i-1, \Pi(j)\}: v \in \{1, 2, ..., m_i\} \}$

and set $T(i, k) := \max\{T(i, k), C(i-1, \Pi(j)\} + p_{i\Pi(j)}$, $C(i, \Pi(j)) := T(i, k)$

as well as if $i = m$ then set $C_{\Pi(j)} := C(m, \Pi(j))$,

where $C(i, j)$, $C(m, j)$ are, respectively, the completion times of task j on i-stage and on the last stage m.

STEP 3. Set $Cmax = \max \{Cj: j \in J\}$ or $\Sigma Cj = \Sigma_{j \in J} Cj$.

Naturally, the objective function values calculated by the procedure SCHEDULE essentially depends on the permutation Π. Thus one can state the problem of permutation search which minimizes the selected objective function.

The quality of makespan Cmax of approximate solutions can be estimated by lower bound LB, i.e. $Cmax \geq LB$, determined on the ground of problem relaxation, calculated by the following expression:

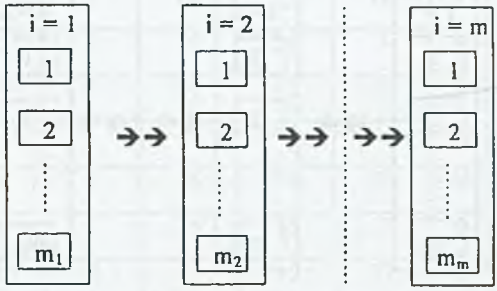
$$LB = \max \{ LB1, LB2 \}$$

where LB1 is the processing time of the longest task, i.e.

$$LB1 = \max \{ \sum_{i=1}^{m} p_{ij}: j \in J \}$$

and LB2 is the tasks processing time on the bottle-neck stage:

$$LB2 = \max \{ (\min_{j \in J} \sum_{k=1}^{i-1} p_{kj} + \sum_{j=1}^{n} p_{ij}/m_i + \min_{j \in J} \sum_{k=i+1}^{m} p_{kj}): i \in \{1, 2, ..., m\} \}$$



→→→ *(direction of production tasks flow)*

Fig.1. Schematic diagram of production flow line
Rys.1. Schemat przepływowej linii produkcyjnej

4. Case study

The case study deals with a small production flow line consisted of m = 5 stages and production program of n = 10 tasks. Table 1 presents the stage machine numbers m_i, and Table 2 summarizes data concerning tasks, i.e. processing times p_{ij}.

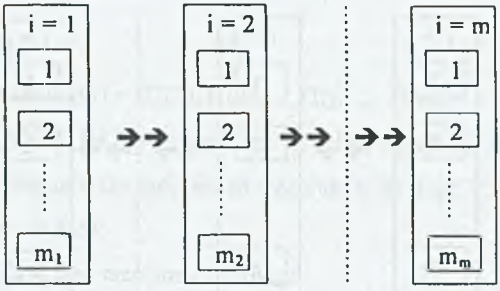
Table 1

Machine numbers of the flow line

stage i =	1	2	3	4	5
machine number m_i =	1	2	2	3	2

and LB2 is the tasks processing time on the bottle-neck stage:

$$LB2 = \max \{ (\min_{j \in J} \sum_{k=1}^{i-1} p_{kj} + \sum_{j=1}^{n} p_{ij}/m_i + \min_{j \in J} \sum_{k=i+1}^{m} p_{kj}): i \in \{1, 2, ..., m\} \}$$



→→→*(direction of production tasks flow)*

Fig.1. Schematic diagram of production flow line
Rys.1. Schemat przepływowej linii produkcyjnej

4. Case study

The case study deals with a small production flow line consisted of $m = 5$ stages and production program of $n = 10$ tasks. Table 1 presents the stage machine numbers m_i, and Table 2 summarizes data concerning tasks, i.e. processing times p_{ij}.

Table 1
Machine numbers of the flow line

stage i =	1	2	3	4	5
machine number m_i =	1	2	2	3	2

Table 2
Processing times of the task operations

j	p_{1j}	p_{2j}	p_{3j}	p_{4j}	p_{5j}
1	5	21	52	52	0
2	4	12	43	44	11
3	2	13	24	25	9
4	1	14	15	16	8
5	4	15	26	7	3
6	5	31	52	53	1
7	1	42	43	44	12
8	2	26	24	25	24
9	3	16	15	16	0
10	4	15	16	17	7

The lower bound of makespan Cmax for the investigated problem is equal LB = max{142, 180}= 180 and the best solution obtained by EVOL-1, after I = 10 trains, is equal 196

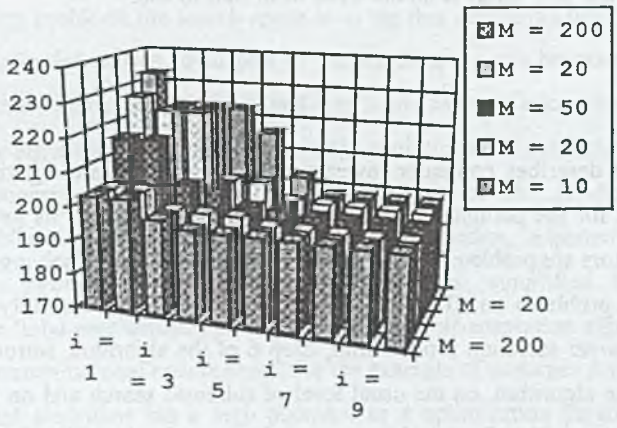


Fig.2. Graph of the function Cmax = Cmax(i) for population size M = 10, 20, 50, 100, 200
Rys.2. Wykres funkcji Cmax = Cmax(i) dla rozmiaru populacji M = 10, 20, 50, 100, 200

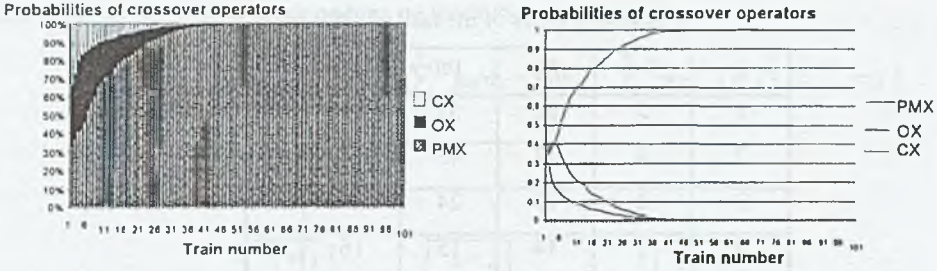


Fig.3. Illustration of self-adaptation of the of crossover operator's selection probabilities
Rys.3. Ilustracja autoadaptacji prawdopodobieństw wyboru operatorów krzyżowania

(see Fig.2). Figure 2 shows the influence of population size M (M = 10, 20, 50, 100, 200) on evolutionary process where others algorithm's parameters are constant.

In Figure 3 we display the variability of the crossover operators selection probabilities, i.e. the self-adaptation of these evolutionary algorithm parameters, during the evolution process, where start probabilities of crossover operators are: $p_{PMX} = p_{OX} = p_{CX} = 1/3$. Let us notice that at first probability p_{OX} grows lager and then, after some train numbers is on the wane as well as the p_{CX} value is on the wane from start to end.

5. Conclusion

The paper describes computer investigations of approximate algorithm of *EA* type specially adopted for the permutation optimization problems through its genetics operators. All genetic operators are problem-independent thus it can be used for solving all kind of NP-hard permutation problems on IBM PC compatible computers successfully. The adaptively changing of crossover selection probabilities, Step 6 of the algorithm, introduce learning on two levels into the algorithm, on the usual level of solutions search and on the level of *EA* algorithm parameters. This is promising way to overcome the difficulty of determining appropriate parameter settings of *EA* for each application task anew.

BIBLIOGRAPHY

1. Hunsucker J.L., Shah J.R.: Comparative performance analysis of priority rules in a constrain flow shop with multiple processors environment, European J. Oper. Res. 72, 1994, 102-114.
2. Michalewicz Z.: Genetic algorithms + data structures + evolution programs, Berlin, Springer-Verlag 1992.

3. Smutnicki Cz.: Scheduling of flexible flow lines and work centers, Zeszyty Naukowe Politechniki Śląskiej, Automatyka 118, 1996, 175-185.
4. Wala K., Chmiel W.: Evolution algorithm for quadratic assignment problem, Krakow, University of Mining and Metallurgy Press, Automatics, 1997, 1, 1, 409-414.
5. Wala K.: Evolutionary algorithm for optimization of the discrete problems (Polish), Prace z Automatyki, Wydawnictwa AGH, Krakow 1997, 69-82.

Recenzent: Dr hab.inż. Jan Kałuski, prof.Pol.Śl.

Abstract

In the field of CIM production, scheduling is intended to allocate the manufactory resources to the production orders to be processed. Due to the time complexity, exponentially increasing with the problem size, of the prevalent number of scheduling problems conventional optimization methods such as Branch & Bound can accurately solve usually very small scheduling problems. Approximate methods include priority rules, Monte Carlo methods and neighborhood search. The drawback of these enumerative procedures is that each of the trial solution can only be examined as a whole in respect to their suitability. In case of scheduling problems the search space is so big that during the time period relevant to the practical application only a small part of the search space can be examined. Algorithms which combine elements of the neighborhood search and Monte Carlo method are the genetic and evolutionary algorithm. The paper presents the evolutionary algorithm, which realizes the original evolutionary search process, for optimization of the NP-hard permutational scheduling problems. The investigated EVOL-1 algorithm, adjusted for permutation problems, is a hybrid of a modified genetic algorithm, simplified local optimization procedure, some 'tabu mechanism', as well as a simple self-adaptation algorithm parameters procedure. The computational results show, for the example of m-stages production flow line, that the proposed algorithm has a high potential as a optimization paradigm for the CIM production scheduling.