balancing, assembly line, genetic algorithm

Jozef HNÁT, Milan GREGOR[*]

GENETIC ALGORITHM AS OPTIMIZING TOOL IN LINE BALANCING PROCESS

Abstract

This article deals with assembly line balancing problem and with possibility of using genetic algorithm as an optimizing tool in balancing process. There is briefly described line balancing problem and the main goals of balancing. Also it is shown here how genetic algorithm work and what are the advantages of this tool.

1. INTRODUCTION

Assembly line production is widely used and is one of the basic principles in production systems. It is the most commonly used method in a mass production environment. The main objective of assembly systems is to increase the efficiency of the line by maximizing the ratio between throughput and cost. Assembly lines are special flow-line production systems of a number of stations *(n)* arranged along a conveyor system. The assembled product takes its shape gradually starting with one part (the base part), with the remaining parts being attached at the various stations which are visited by the product.

Assembly line design involves the design of products, processes and plant layout before the construction of the line itself. These different modules interact at the different stages of assembly line design. The product analysis proposes a product design review based on the classical design for assembly rules and precedence constraints between tasks. The operating modes and techniques module proposes an assembly technique and the possible modes (manual, automated, robotic) for each task. The line layout module assigns tasks to a set of stations and decides on the position of stations and the resource on the plant floor (Figure 1).

The design of efficient assembly workshops is a problem of considerable industrial importance. In general, for simple products a single linear assembly line with possibility parallel station can do the job. For complex products, the assembly system is mostly decomposed into sub-systems with their own cycle time, reliability, and stations requirements.

[*] Ing. Jozef Hnát, Department of industrial engineering, Faculty of mechanical engineering, University of Žilina, Univerzitná 1, Žilina, jozef.hnat@fstroj.uniza.sk
Prof. Ing. Milan Gregor, PhD. Department of industrial engineering, Faculty of mechanical engineering, University of Žilina, Univerzitná 1, Žilina, milan.gregor@fstroj.uniza.sk

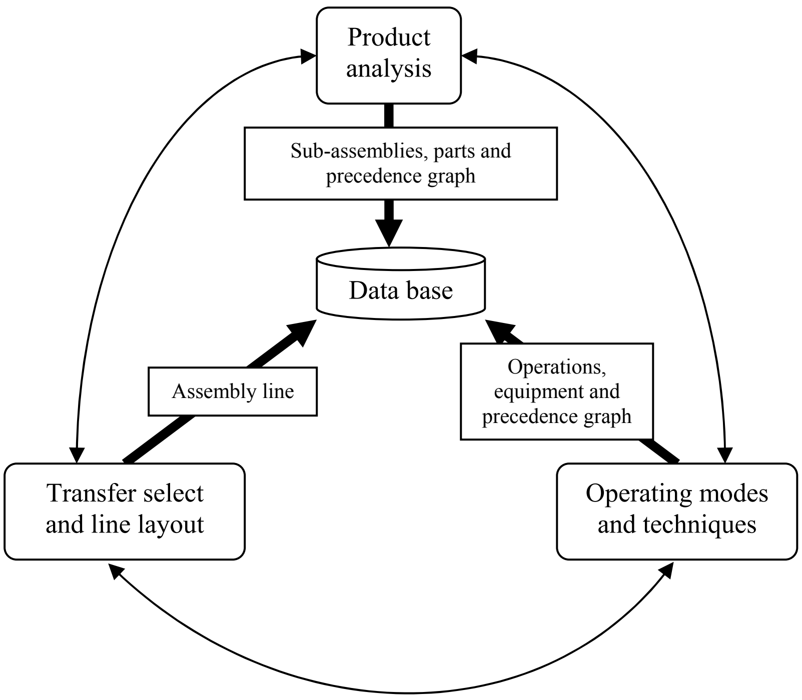

Product analysis
Sub-assemblies, parts and precedence graph
Data base
Assembly line
Operations, equipment and precedence graph
Transfer select and line layout
Operating modes and techniques

Figure 1 Methodology and information flow of the assembly line design

According to number of product variants produced by assembly line, we can divide them into two basic groups:

Single product assembly line - the single product line is used to produce only one product. If dynamic phenomena are neglected, the workload of all stations remains constant over time.
Mixed production assembly line - a family of products is a set of distinguished products (variants), whose main functions are preferably similar. A typical example is a family of cars with different options: some of them will have a sunroof, some will have ABS, *etc.*

2. LINE BALANCING PROBLEM

One of the greatest problems of assembly line design and optimization, is balancing problem. A line balancing problem is defined by a line along which products (vehicles) go through and are progressively assembled. The assembly operations are performed by workstations spread along the line. The objective is to assign operations to workstations in order to minimize, for instance the number of required workstations (workers), or we can say that objective is to balance the line (Figure 2). This is called a simple assembly line balancing problem. If the line is not well balanced the idle times on workstations rise. Sum of the workstations unbalances presents unbalance of whole assembly line *Z*. The basic constraints are cycle time and precedence constraints.

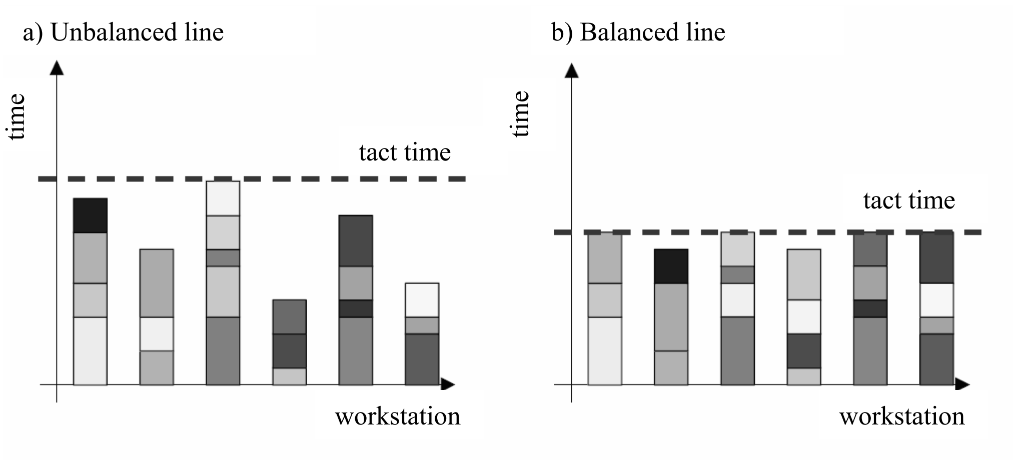


Figure 2 Line balance

If the sequence of operations and operation times are given, only number of workstations and tact time can be influenced. Goal of the solving line balancing problem can be:

a) minimize number of workstations *n* if tact time is given (cycle time is constatnt),
b) minimize length of tact time for given number of workstations, or generally,
c) minimize number of idle time units of whole line (maximize effectivity and minimize unbalance), when it is possible to select number of workstations and tact time.

Data inscription

Sequence of the operations and their technological relations can be described in several ways. First one is using table.

Tab. 1 Data table

Operation number *i*	**Operation time t_i**	**Numbers of directly precedent operations**
1	6	-
2	2	1
3	5	1
4	7	1
5	1	1
6	2	2
7	3	3,4,5
8	6	6
9	5	7
10	5	8
11	4	9,1
Sum	46	

This data can be written into precedence graph, where corners of the graph represent operations and flowlines represent relationship between them. Operation number is written within the circle and operation time is shown next to the circle. Data from table 1 are written into precedence graph (figure 3).

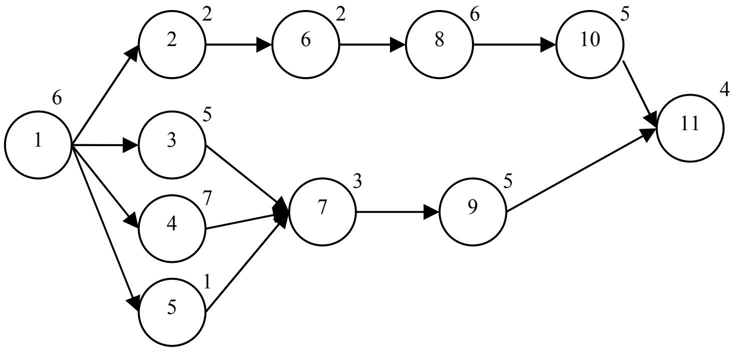


Figure 3 Precedence diagram

Technological relations can be expressed also by precedence matrices. In the matrices element *ij* equals 1, if operation *i* is direct precedent of operation *j*, in the other case it equals 0.

Tab. 2 Precedence matrices

i \ *j*	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
1		1	1	1	1						
2						1					
3							1				
4							1				
5							1				
6								1			
7									1		
8										1	
9											1
10											1
11											1

Basic relations

In general following marks and relations are used:

i	-	operation index, $i = 1, 2, 3, ..., n,$
n	-	number of operations,
$t_i(t_k)$	-	operating time of *i*-th (*k*-th) operation, i.e. number of time units needed for *i*-th (*k*-th) operation execution,

$T = \sum_{i=1}^{n} t_i$	-	sum of all operating times of all operations (product labor content),
j	-	workstation (workplace) index, $j = 1, 2, 3, ..., m$,
m	-	number of workstations,
A	-	set of all operations,
$A_j(A_l)$	-	set of operations assigned to j-th (l-th) workstation,
$t(A_j)$	-	sum of operating times of operations assigned to j-th workstation, $t(A_j) = \sum_{i \in A_j} t_i$
$c, c_1, c_2, ..., c_n$	-	manufacturing tact time, i.e. time spent by product on the each workstation, expressed in time units compatible with operation time units,
ipk	-	sequence relation, it means that i-th operation must precede k-th operation in the manufacturing process,
Z	-	objective function, i.e. number of idle time units on j-th workstation (unbalance),
Z_j	-	number of idle time units on j-th workstation,
E	-	line efficiency.

Formulations for conditions and goals:

$$\bigcup_{j=1}^{m} A_j = A, \tag{1}$$

$$A_j \cap A_1 = 0 \qquad (j = 1) \tag{2}$$

$$t(A_j) \leq c \qquad (j = 1,2,...,m) \tag{3}$$

If ipk and i $\in$ A_j and k $\in$ A_l, then j ≤ l (4)

minimize

$$Z = \sum_{j=1}^{m} \left(c - t(A_j)\right) \tag{5}$$

Condition (1) provides that all operations will be assigned to any workstation, and condition (2) provides that none operation will be assigned to more workstations at the same time. Both conditions provide clear and entire assignment of all operations on line workstations. Condition (3) provides, that sum of operation times of operations assigned to workstations, won`t exceed tact time and thanks to condition (4) technologic sequence will be obseved.

Each assignment which fulfils this four conditions is called feasible solution and solution satisfying also conditions (5) is called optimal solution. Minimizing condition (5) minimal sum of idle times is provided.

According to condition (3) sum of operating time of assigned operations to each workstation is less or equal to tact time and because of this objective function can be defined as final assignment effectivity.

$$E = \text{sum of operation times / real time spent on line} = T / mc \quad (6)$$

Objective function *Z,* express optimal solution by absolute value of difference between real time spent by product on the line and labor content.

$$Z = \sum_{j=1}^{m} Z_j = \sum_{j=1}^{m} \left(c - t\left(A_j\right)\right) \quad (7)$$

$$Z = mc - \sum_{j=1}^{m} t\left(A_j\right) = mc - T \quad (8)$$

First of all we will deal with number of workstations. Minimum number of workstations is one. But this solution is trivial and it is about piece production, not about line production. Maximal number of workstations can't be higher than number of operations executed on product ($m = n$). In this case effectivity of line would be dependent on operation time numbers, because minimal tact time value is limited by the longest operation time duration.

If the longest operation time duration is $t_{\max}$

$t_{max} = \max t_i \qquad (i = 1,2,...,n) \qquad (9)$

then relation for maximal number of workstations is:

$$m_{\max} = \left[\frac{T}{t_{\max}} \right] \quad (10)$$

Square bracket presents integral result.

Term (10) has only supply character, because some tasks exist where higher number of workstations than counted number, leads to higher effectivity of the line. For example line with four operations executed in series (figure 4) has $t_{\max}=5$, $T=15$, then $m_{\max}=15/5=3$.

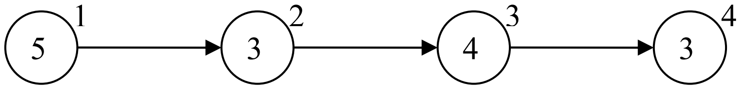


Figure 4 Precedence graph

For three workstations will be necessary to assign two operations on one workstation. Because of sequence and tact time restrictions the most suitable will be to combine second and third, or third and fourth operation. In the both cases tact time presents seven time units.

Effectivity is:

$$E = \frac{T \cdot 100}{mc} = \frac{15 \cdot 100}{3 \cdot 7} = 71{,}4\%$$

If there is four workstations $m = m_{\max} + 1$, tact time $c = 5$ is enough and final effectivity is:

$$E = \frac{15 \cdot 100}{4 \cdot 5} = 75\%$$

(This effectivity is higher.)

The best result could be reached if there are two workstations and two operations. Then $c = 8$ and effectivity is:

$$E = \frac{15 \cdot 100}{2 \cdot 8} = 93{,}8\%$$

Let integral number $m_{\max}$, designated by relation (10), gives maximal number of workstations. Real number of workstations must be natural number and must fulfill following constrain:

$$1 \le m \le m_{\max} \le n \tag{11}$$

When line balancing problem is solved, first of all tact time should be estimated for each number of workstations. It is needed to find optimal assignment with minimal tact time and to find equivavlent effectivity. Solution with the highest effectivity is considered as a final solution.

Generally we are not interested in all possible solutions, but mostly in solutions with higher number of workstations or in situation when the number of workstation is firmly given. Or if tact time is given, we are looking for corresponding minimal number of workstations. .

Effort to solve line balancing problem has resulted into using well known methods for line balancing problems, but also new very powerful optimizing tool genetic algorithm has appeared.

3. GENETIC ALGORITHMS (GA)

GAs are a stochastic search techniques based on the mechanism of natural selection and natural evolution. Biological inspired aspects of this algorithm can be seen in following basic steps [2]:

1. GA can operate on any data type (representation) which determines the bounds of the search space. It is desirable that the representation can only encode feasible solutions, so that the objective function (fitness) measures only optimality and not feasibility.

2. The initial population is created during an initialization phase and it is often generated at random. Generally, some knowledge is used by the GA to start the search from promising regions of the search space.
3. Every member of the population is then evaluated and a fitness value is given according to how well it fulfils the objectives. If there is no clear way to compare the quality of different solutions, then there can be no clear way for the GA to allocate more offspring in the fitter solutions.
4. The GA favours individuals with a higher overall fitness when picking "parents" from the population. The fitness function allows the evaluation of solutions. Then, these scores are used to determine which individuals will participate in creating the new population.
5. Based on the fitness values, the GA selects candidate solutions and combines (crossover) the best traits of the parents to produce superior children.
6. A small part of the population is mutated. Single existing individuals are modified to produce a single new one. It is more likely to produce harmful or even destructive changes than beneficial ones.
7. Natural selection ensures that the weakest creatures die, or at least do not reproduce as successfully as the stronger ones. In the same way, a population is maintained with the fittest solutions being favoured for reproduction. New generations are formed by selecting some parents and offspring and rejecting the less-fit ones.
8. A generation is a population at a particular iteration of the loop. This iterative process (selection, crossover. etc.) continues until the specified number of generations is passed, or an acceptable solution has emerged.

3.1 Standard GA

Standard GA written into pseudo-code is shown in the table:

Table 3 GA pseudo-code

t := 0	
Initialize G(0)	*choose initial population*
Evaluate G(0)	*evaluate the fitness of each individual in the population*
do while not Done	
t := t + 1	
Select G(t) from G(t-1)	*make natural selection*
Crossover G(t)	*apply crossover*
Mutate G(t)	*apply mutation*
Evaluate G(0)	*evaluate*
Replace G(t-1) with G(t)	*replace worst ranked part of population with offspring*
loop	*until termination*

In the block of initialization, generations counter is released *t=0* and initial population is generated *G(0)*, usually consisting of randomly created members. For each of them, in the block of evaluation, its feasibility is calculated. After testing fulfillment of fitness function, are in the block of selection choosed members from population *G(t)* dedicated for reproduction.

From them, in the block of offspring generation, new members are created using genetic operators (usually crossover and mutation. In the block of evaluation is fitness value given to each offspring. Block of replace realizes creating of new generation *G(t+1)* usually from offspring, but sometimes also from generation members *G(t)*. This cycle continues till the successful test in the block fulfillment of fitness function. Whole this algorithm is shown as flow process diagram on the figure 2.

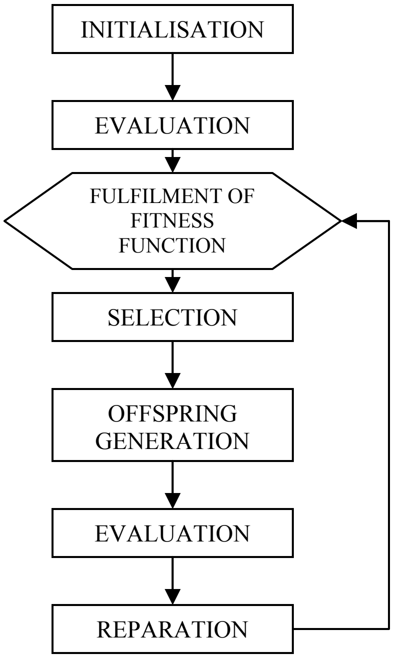


Fig.2 Flow process diagram of standard GA

3.2 Representation - encoding

The first step in designing a GA for a particular problem is to devise a suitable representation. For instance, it is quite natural to represent an *n*-dimensional vector as a string of *n* values (genes), while it is difficult to represent a graph without introducing extra information. Other problem we have to solve when we start to work with GA is encoding of chromosomes. Encoding very depends on the problem. So if form (parameters) of searched solution is encoded into chromosomes, in the world of computers it is usually represented as a string, whose attributes can be:

Binary encoding

Binary encoding is the most common, mainly because first works about GA used this type of encoding. In binary encoding every chromosome is a string of bits, 0 or 1.

Example of chromosomes with binary encoding:

Chromosome A	101100101100101011100101
Chromosome B	111111100000110000011111

Binary encoding gives many possible chromosomes even with a small number of alleles. On the other hand, this encoding is often not natural for many problems and sometimes corrections must be made after crossover and/or mutation.

Permutation encoding

Permutation encoding can be used in ordering problems, such as traveling salesman problem or task ordering problem. In permutation encoding, every chromosome is a string of numbers, which represents number in a sequence.
Example of chromosomes with permutation encoding:

Chromosome A	1 5 3 2 6 4 7 9 8
Chromosome B	8 5 6 7 2 3 1 4 9

Permutation encoding is only useful for ordering problems. Even for this problems for some types of crossover and mutation corrections must be made to leave the chromosome consistent.

Value encoding

Direct value encoding can be used in problems, where some complicated value, such as real numbers, are used. Use of binary encoding for this type of problems would be very difficult. In value encoding, every chromosome is a string of some values. Values can be anything connected to problem, form numbers, real numbers or chars to some complicated objects.
Example of chromosomes with value encoding:

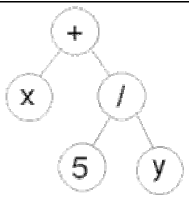
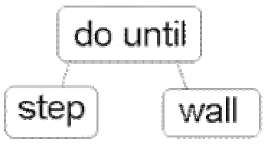
Chromosome A	1.2324 5.3243 0.4556 2.3293 2.4545
Chromosome B	ABDJEIFJDHDIERJFDLDFLFEGT
Chromosome C	(back), (back), (right), (forward), (left)

Value encoding is very good for some special problems. On the other hand, for this encoding is often necessary to develop some new crossover and mutation specific for the problem.

Tree encoding

Tree encoding is used mainly for evolving programs or expressions, for genetic programming.
In tree encoding every chromosome is a tree of some objects, such as functions or commands in programming language.

Example of chromosomes with tree encoding:

Chromosome A	Chromosome B
	
(+ x (/ 5 y))	(do_until step wall)

Tree encoding is good for evolving programs. Programming language LISP is often used to this, because programs in it are represented in this form and can be easily parsed as a tree, so the crossover and mutation can be done relatively easily.

Length of a string can be:

- fixed (typical for strings 1. – 3.)
- variable (comes from character of values type 4)

3.3 Feasibility

GAs may employ four basic strategies to deal with infeasible solutions: rejection, repair, modifying the genetic operator, and assigning penalties. The rejection strategy simply discards all infeasible individuals, while the repairing strategy attempts to create only feasible solutions. For some problems, genetic operators can be modified so that they create only feasible solutions. Finally, penalty functions can be used when infeasible solutions can be recombined to form feasible ones (figure 3)

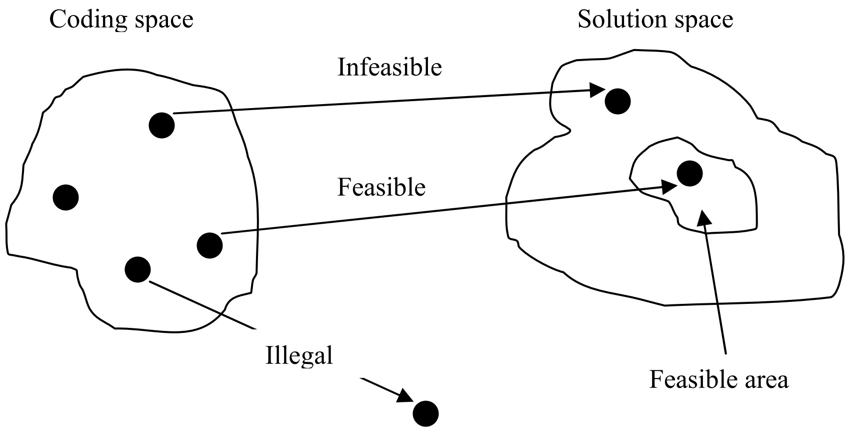


Figure 3 Feasibility of solutions

3.4 Selection

During each successive generation, a proportion of the existing population is selected to breed a new generation. Individual solutions are selected through a *fitness-based* process, where fitter solutions (as measured by a fitness function) are typically more likely to be selected. Certain selection methods rate the fitness of each solution and preferentially select the best solutions. Other methods rate only a random sample of the population, as this process may be very time-consuming.

Most functions are stochastic and designed so that a small proportion of less fit solutions are selected. This helps keep the diversity of the population large, preventing premature convergence on poor solutions. Popular and well-studied selection methods include roulette wheel selection and tournament selection.

In roulette wheel selection (fitness proportionate selection), as in all selection methods, the fitness function assigns a fitness to possible solutions or chromosomes. This fitness level is used to associate a probability of selection with each individual chromosome. If f_i is the fitness of individual i in the population, its probability of being selected is (12)

$$p_i = \frac{f_i}{\sum_{j=1}^{N} f_j} \qquad (12)$$

where N is the number of individuals in the population. While candidate solutions with a higher fitness will be less likely to be eliminated, there is still a chance that they may be. Contrast this with a less sophisticated selection algorithm, such as truncation selection, which will eliminate a fixed percentage of the weakest candidates. With fitness proportionate selection there is a chance some weaker solutions may survive the selection process; this is an advantage, as though a solution may be weak, it may include some component which could prove useful following the recombination process.

The analogy to a roulette wheel can be envisaged by imagining a roulette wheel in which each candidate solution represents a pocket on the wheel; the size of the pockets are proportionate to the probability of selection of the solution. Selecting N chromosomes from the population is equivalent to playing N games on the roulette wheel, as each candidate is drawn independently.

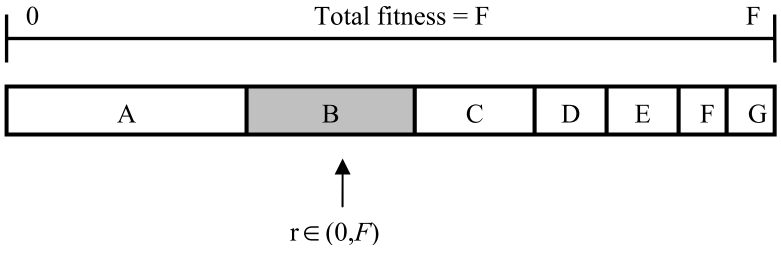


Figure 4 Example of the selection of a single individual

Tournament selection runs a "tournament" among a few individuals chosen at random from the population and selects the winner (the one with the best fitness) for crossover.

Selection pressure can be easily adjusted by changing the tournament size. If the tournament size is larger, weak individuals have a smaller chance to be selected.

Tournament selection pseudo code:

```
choose k (the tournament size) individuals from the population at random
choose the best individual from pool/tournament with probability p
choose the second best individual with probability p*(1-p)
choose the third best individual with probability p*((1-p)^2)
```

and so on...

Deterministic tournament selection selects the best individual (when p=1) in any tournament. A 1-way tournament (k=1) selection is equivalent to random selection. The chosen individual can be removed from the population that the selection is made from if desired, otherwise individuals can be selected more than once for the next generation.

Tournament selection has several benefits: it is efficient to code, works on parallel architectures and allows the selection pressure to be easily adjusted.

3.5 Genetic operators

Generally it exist three types of genetic operators:

1. asexual (mutation) – are applied on one parent, adding new genetic information.
2. sexual (crossover) – are applied on two parents, properly combining their genetic material.
3. panmictic (more than two parents) – are applied on more than two parents, also properly combining parents genetic material.

Each of this operators is applied with specific probability, typical values are 0,001 for mutation (p_m) and 0,6 for crossover (p_c). If operator hasn`t been applied then individual usually „survives“ without changes.

The change of chromosome caused by one of the operators depends on representation.

Mutation

The classic example of a mutation operator involves a probability that an arbitrary bit in a genetic sequence will be changed from its original state. A common method of implementing the mutation operator involves generating a random variable for each bit in a sequence. This random variable tells whether or not a particular bit will be modified.

The purpose of mutation in GAs is to allow the algorithm to avoid local minima by preventing the population of chromosomes from becoming too similar to each other, thus slowing or even stopping evolution. This reasoning also explains the fact that most GA systems avoid only taking the fittest of the population in generating the next but rather a random (or semi-random) selection with a weighting toward those that are fitter.

Each bit mutates with probability p_m and it can be also *1/l* , where *l* is length of string.

chromosome of individual	individual after mutation
01**1**1001010	01**0**1001010

Crossover

Essentially can be *n*-pointed (most frequently n = 1 or 2) or uniform.

- <u>onepoint crossover</u> (n = 1), point of crossover is randomly selected, i.e. number from interval <1, *l*-1> and in that strings of two individuals are crossed (in the case, that it is ruled out, the crossover is applied according given probability):

chromosomes of parents	chromosomes of offspring
011|1001010	**011**0011101
001|0011101	001**1001010**

- <u>doublepoint crossover</u> (n = 2),

chromosomes of parents	chromosomes of offspring
011|1001|010	**011**0011**010**
001|0011|101	001**1001**101

- <u>uniform</u> – mask is generated and according to it strings are crossed. Probability of generating 0 or 1 in the mask is usually 0,5, but on principle can be different. (Crossing is average in *l*/2 points.)

mask
0**11**0**1**00**11**0

chromosomes of parents	chromosomes of offspring
0 - 0111001010	**0**11**0111**0**11**
1 - 0010011101	0**011**001**10**0

Panmictic operators

This operators are used rarely. In principle it is crossover genetic material of several parents:

- <u>scanning crossover</u> - n parents, 1 offspring, i-th offsprings bit is defined by reading (scan) i-th bits of parents and by voting or by probability is defined value for offspring, $p_c = 1$
- <u>diagonal crossover</u> - n parents, n offspring, n crossover points. (If n = 2 then it is onepoint crossover.)

chromosomes of parents (n=3)	chromosomes of offspring
01|1100|1010	01**1001***1001*
00|1001|1101	**00***1101*1010
10|1101|1001	*10*1100**1101**

3.6 Forming new generation

Population size can be changed in general (usually is not changed). By forming new generation it has to be defined from which individuals it is going to be formed - P(t+1), and it can be selected from individuals in old population - P(t) and from population of offspring P"(t):

$$P(t+1) \subset P(t) \cup P''(t), \text{ and } |P(t)| = \mu(t), |P''(t)| = \lambda(t)$$

Two extremes and combination are possible here:

1) $P(t+1) \subset P(t)$, absurdity, it is not used.

2) $P(t+1) \subset P''(t)$, generation substitution (whole generation is changed),
   a) if $\mu(t) = \lambda(t)$, $P(t+1) = P''(t)$;
   b) if $\lambda(t) > \mu(t)$, it is needed to apply methods of selection therefore to reduce string.

3) $P(t+1) \subset P(t) \cup P''(t)$, two strategies are available here:
   a) plus strategy, signed $(\mu+\lambda)$ and by selection method is number of individuals reduced;
   b) selection by generation gap – proportion of population, which passes to the other generation without changes.

3.7 Ending condition

In general three possibilities of ending condition exist:

1. feasible solution has been achieved,
2. population has converged,
3. predetermined number of generations has been reached.

In the first case is on purpose said “feasible” and not “optimal” solution, because optimum is not usually known, but forecast of respectable solution for the problem exists. This approach is consistent with biological analogy EA, so far from also natural evolution finds feasible and not optimal solutions.

Gene has converged, when at least 90% individuals in population has the same value of this gene. Population has converged when all of the genes converge.

3.8 Main advantages and disadvantages of GA

Advantage of GAs is in their parallelism. GA is travelling in a search space with more individuals (and with genotype rather than phenotype) so they are less likely to get stuck in a local extreme like some other methods.
They are also easy to implement. Once you have some GA, you just have to write new chromosome (just one object) to solve another problem. With the same encoding you just

change the fitness function and it is all. On the other hand, choosing encoding and fitness function can be difficult.

Disadvantage of GAs is in their computational time. They can be slower than some other methods. But with todays computers it is not so big problem.

To get an idea about problems solved by GA, here is a short list of some applications:

- Nonlinear dynamical systems - predicting, data analysis
- Designing neural networks, both architecture and weights
- Robot trajectory
- Evolving LISP programs (genetic programming)
- Strategy planning
- Finding shape of protein molecules
- TSP and sequence scheduling
- Functions for creating images

Genetic algorithms has been used for difficult problems (such as NP-hard problems), for machine learning and also for evolving simple programs. They have been also used for some art, for evolving pictures and music.

4. CONCLUSION

Several approaches and algorithms in the line balancing problem exist. This approaches are usually suitable for simple line balancing problem but for more complicated balancing problem like mix-model production effective methods are missing. On this account it is necessary to handle with new methods and approaches to the line balancing process. One of very effective approaches is using of genetic algorithm. Optimization is the main field where genetic algorithm can be used, even though GA is not optimizer, doesn't guarantee finding optimal solution. However using GA like optimization tool is not so usual in solving of manufacturing problems and because of this more attention should be paid on this optimizing technique.

Literature

[1] UNČOVSKÝ, L.: *Operačná analýza v riadení podnikov*, Alfa Bratislava, 1985

[2] REKEIK,B.-DELCHAMBRE,A.: *Assembly line design*, Springer – Verlag London Limited, 2006

[3] ZANDIN, K.B.: *Maynard's Industrial Engineering Handbook*, 5th edition, McGraw Hill Standard handbooks, 2000

[4] http://alife.tuke.sk/