SOME IMPROVEMENTS OF THE GENETIC ALGORITHM FOR QUEUEING THEORY

[a]PAVOL ORŠANSKÝ, [b]VLADIMÍR GULDAN
Department of Applied Mathematics, The Faculty of Mechanical Engineering at the University of Zilina, Univerzitná 8215/1, 010 26 Žilina, Slovakia
email: [a]pavol.orsansky@fstroj.uniza.sk,
[b]vladimir.guldan@fstroj.uniza.sk

This work was supported by KEGA under the Grant No. 029ŽU-4/2022 "Implementation of the principles of blended learning into the teaching of the subject Numerical Methods and Statistics".

Abstract: In this work, we deal with the use of optimization methods of the genetic algorithm queueing theory. Concrete for the optimization of production process plans in serial production. Specifically, we deal with the creation of an optimal sequence of production processes with regard to time and economic savings. In addition to the classically used genetic algorithms, we will focus on their improvements, such as the Tabu search, the penalty algorithm and simulated annealing. The simulation modeling took place in the Matlab environment.

Keywords: Optimization, Genetic algorithm, Simulated annealing.

1 Introduction to optimization flow-shop scheduling

Optimization is the search for the best solution from a set of possible solutions. In this case, we do not understand the term "solution" only in a strictly mathematical sense, such as solving an equation. Optimization deals with the search for the global minimum (or maximum) of functions of many variables with respect to possible limiting conditions. Many tasks from both engineering practice and natural sciences correspond to this general definition.

Flow-shop scheduling is an optimization problem in computer science and operations research. It is a variant of optimal job scheduling. In a general job-scheduling problem, we are given n jobs $J_1, J_2, \ldots, J_n$ of varying processing times, which need to be scheduled on m machines with varying processing power, while trying to minimize the makespan – the total length of the schedule (that is, when all the jobs have finished processing). In the specific variant known as flow-shop scheduling, each job contains exactly m operations. The i-th operation of the job must be executed on the i-th machine. No machine can perform more than one operation simultaneously. For each operation of each job, execution time is specified. The flow shop problem is a special case of the job shop problem.

There are m machines and n jobs. Each job contains exactly m operations. The i-th operation of the job must be executed on the i-th machine. No machine can perform more than one operation simultaneously. For each operation of each job, execution time is specified.

Operations within one job must be performed in the specified order. The first operation gets executed on the first machine, then (as the first operation is finished) the second operation on the second machine, and so on until the m-th operation. Jobs can be executed in any order, however. Problem definition implies that this job order is the same for each machine. The problem is to determine the optimal such arrangement, i.e., the one with the shortest possible total job execution makespan. [1]

The planning problem generally belongs to NP-complete problems (nondeterministic polynomial-time complete). This means that the time required to solve an NP-complete problem grows asymptotically faster than polynomially (usually exponentially) with the size of the problem input (instance). The consequence is that the time required to solve even moderately large instances of NP-complete problems easily reaches billions or trillions of years using any amount of computing power available today. This is also why the question of whether it is possible to solve NP-complete problems efficiently is one of the central questions of computer science today. [2]

More extensive problems must be solved by heuristic methods, which do not guarantee finding an optimal solution, nor can they determine how close to the optimum a certain admissible solution is but are able to provide a "satisfactory" solution in a reasonable time. Evolutionary algorithms based on biological knowledge are a special chapter of heuristic algorithms. The disadvantages of these algorithms include the probabilistic nature of the results. This means that a different result may occur after each run. Among the basic and best-known evolutionary algorithms at the present time is so called genetic algorithm and its improvements such as the Tabu search, simulated annealing and the penalty algorithm.

2 Genetic algorithm

The genetic algorithm (GA) is a method for solving optimization problems that is based on natural selection, the process that drives biological evolution. It was pioneered by John Hol-land (1975) and his students at the University of Michigan. [3] The GA repeatedly modifies a population of individual solutions. At each step, the genetic algorithm selects individuals from the current population to be parents and uses them to produce the children for the next generation. Over successive generations, the population "evolves" toward an optimal solution. The genetic algorithm is a non-deterministic method of problem solving based on the principles of Darwin's theory of evolution. Each solution to the problem is called a chromosome and is made up of a binary string of a given length, which is the same for all chromosomes of the given population. A population is a finite set of chromosomes.

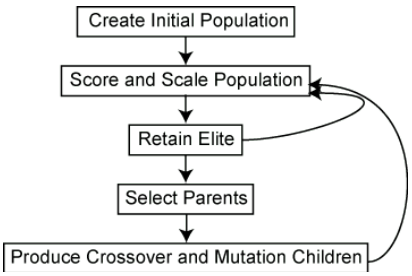


Figure 1: This flow chart outlines the main algorithmic steps.

The GA uses three main types of rules at each step to create the next generation from the cur-rent population:

Selection rules select the individuals, called parents, that contribute to the population at the next generation. The selection is generally stochastic and can depend on the individuals' scores.

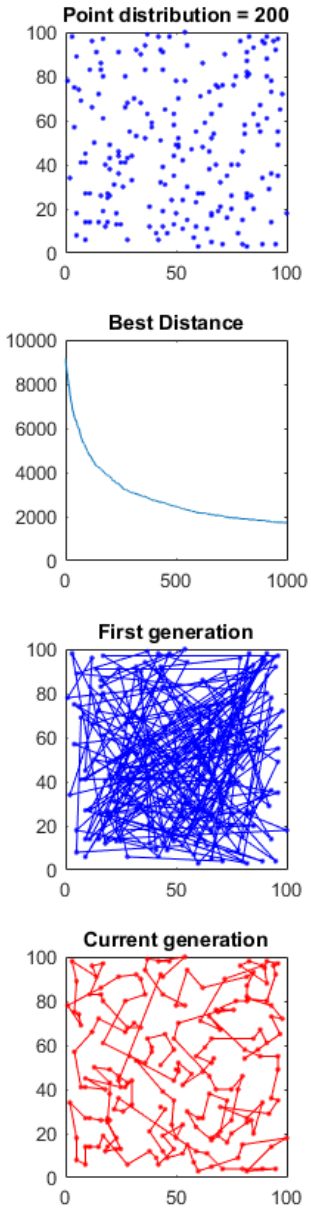
Crossover rules combine two parents to form children for the next generation.

Mutation rules apply random changes to individual parents to form children.

The selection is based on the fitness function. The fitness function is the function you want to optimize. For standard optimization algorithms, this is known as the objective function. The fitness value of an individual is the value of the fitness function for that individual. To create the next generation, the genetic algorithm selects certain individuals in the current population, called parents, and uses them to create individuals in the next generation, called children. Typically, the algorithm is more likely to select parents that have better fitness values.

The algorithm creates crossover children by combining pairs of parents in the current population. At each coordinate of the child vector, the default crossover function randomly selects an entry, or gene, at the same coordinate from one of the two parents and assigns it to the child. For problems with linear constraints, the default crossover function creates the child as a random weighted average of the parents.

The algorithm creates mutation children by randomly changing the genes of individual parents. The GA differs from a classical, derivative-based, optimization algorithm in two main ways. GA generates a population of points at each iteration. The best point in the population approaches an optimal solution, unlike classical algorithm, which generates a single point at each iteration. The sequence of points approaches an optimal solution. GA selects the next population by computation which uses random number generators, unlike classical algorithm, which selects the next point in the sequence by a deterministic computation. [4, 5]



Graph 1: Results of a simple example of optimizing the task of a business traveler using a genetic algorithm for 1000 generations. Finding the shortest path between 200 points is the task of finding the shortest distance between $200! = 8.8 \cdot 10^{377}$ permutations. The found path is not the shortest but it is found in real time.

2.1 Tabu search

Tabu search (TS) is a metaheuristic search method employing local search methods used for mathematical optimization. The method was created by Fred W. Glover in 1986 and formalized in 1989. The basic idea of TS is to penalize moves that take the solution into previously visited search spaces (also known as tabu). TS does deterministically accept non-improving solutions in order to prevent getting stuck in local minimums. During the algorithm, the best solution is recorded, which we consider to be the resulting optimal solution. The disadvantage of this algorithm is that after a certain given number of interaction steps it returns to the local solution that has already occurred in the previous steps. This deficiency was solved by introducing the so-called *short-term* and *long-term memory*.

Short-term memory (tabu list) contains inverse transformations to the transformations used in previous interactions. If the transformation is contained in a tabu sheet, then it cannot be used to construct the neighborhood of the current solution. When the algorithm is initialized, the tabu sheet is empty, after each iteration a transformation is added to it, which provided a locally optimal solution. After the tabu list is filled, it is updated in each iteration (the length of the ban for all moves is reduced by one). An important parameter is the length of the tabu sheet. If the size is too small, then looping of the algorithm may occur. If the length is too large, then with a high probability we will miss local minima, which could be global minima. The search process can be significantly improved by using the so-called aspirational criteria. An aspirational criterion is a condition that allows ignoring a tabu constraint under certain circumstances (e.g., a forbidden move leads to a solution that is better than all solutions achieved so far).

Long-term memory works by disadvantaging (penalizing) those transformations which are not contained in the tabu list, but often occurred in the previous history of the algorithm. We distinguish two processes, intensification and diversification. Intensification strategies will focus on supporting "good" attributes in the search for solutions. Diversification strategies instead generate solutions involving attributes significantly different from those encountered in the previous search process. [6, 7, 8]

2.2 Penalty algorithm

The idea of using penalty functions in calculations was for the first time presented in [9]. Sometimes they have been used in optimization techniques. Several early approaches of evolutionary computations using the penalty functions were born in 1993–1995 [10]. Since genetic algorithms are generic search methods, most applications of genetic algorithms to constraint optimization problems have used the penalty function approach of handling constraints. The most frequently used and the simplest approach is penalization, in which the original fitness function $f(x)$ is supplemented with a penalty function. The objective function $F(x)$ expanded in this way has the form

$$F(x) = f(x) + \text{penalty}(x).$$

The penalty function approach involves a number of penalty parameters which must be set right in any problem to obtain feasible solutions. This dependency of genetic algorithms performance on the penalty parameters has led researchers to devise sophisticated penalty function approaches, such as multilevel penalty functions, dynamic penalty functions, and penalty functions involving temperature-based evolution of penalty parameters with repair operators.

We decided to make the optimization algorithm more efficient by introducing a dynamic penalty. This means that the amount of the penalty increases during the calculation depending on the number of generations. We have therefore introduced a variable into the algorithm which grows during the calculation, always after a certain number of generations (is multiplied to the second). The weight of such an increasing penalty increases with the increasing number of generations. [11, 12]

2.3 Simulated annealing

The basics of this method were first published in 1953 in an algorithm that simulated the cooling of the material in a hot bath. In the early 1980s, Kirkpatrick, Gelatt and Vecchi (1983) [13] and independently Vladimír Černý (1985, MFF UK in Bratisla-

va) [14] proposed that this type of simulation could be used to find admissible solutions of optimization problem in order to ensure convergence to an optimal solution. They also proposed its current name, simulated annealing (SA).

Physical Annealing is the process of heating up a material until it reaches an annealing temperature and then it is cooled down slowly in order to change the material to a desired structure. When the material is hot, the molecular structure is weaker and is more susceptible to change. When the material cools down, the molecular structure is harder and is less susceptible to change.

The acceptance criterion determines whether a new solution is accepted or rejected. The acceptance depends on the energy difference between the new solution and the current solution, as well as the current temperature. The classic acceptance criterion of SA comes from statistical mechanics, and it is based on the Boltzmann probability distribution. A system in thermal equilibrium at temperature t can be found in a state with energy E with a probability proportional to

$$P(\Delta E) = e^{\frac{-\Delta E}{k \cdot t}},$$

where k is the Boltzmann constant. Hence, at low temperatures, there is a small chance that the system is in a high-energy state. This plays a crucial role in SA because an increase in energy allows escape from local minima and find the global minimum.

Based on the Boltzmann distribution, the following algorithm defines the criterion for accepting an energy variation ΔE at temperature t. If we $t, \Delta E$ are the temperature and energy variation between new and current one candidate, than the pseudocode for SA could be written in the next form.

if ($\Delta E < 0$) **then**

    true;

else

    r = random number $\in [0,1)$;

    if $r < \exp(-\Delta E/(k \cdot t))$ **then**

        true;

    else

        false;

    end

end

A candidate solution with lower energy is always accepted. Conversely, a candidate solution with higher energy is accepted randomly with probability $P(\Delta E) = \exp(-\Delta E/k \cdot t)$. The latter case can be implemented by comparing the probability with a random value generated in the range $[0, 1)$. The temperature schedule determines how the temperature of the system changes over time. In the beginning, the temperature is high so that the algorithm can explore a wide range of solutions, even if they are worse than the current solution. As the iterations increase, the temperature gradually decreases, so the algorithm becomes more selective and accepts better solutions with higher probability. A simple scheduling can be obtained by dividing the current temperature by a factor $0 < \alpha < 1$.

And so, as a final improvement, we supplemented the algorithm with a decision to accept a new individual into the next generation based on the principle of simulated annealing. The mutation operation is of more fundamental importance to the calculation process, we decided to apply this procedure to it. And so that the quality (determined by the objective function) of each mutated individual is compared with the quality of its predecessor. If the quality of the mutated is better than the quality of the predecessor, the mutated advances to the next generation. If not, it advances to the next generation with a probability given by the principle of simulated annealing. [15, 16]

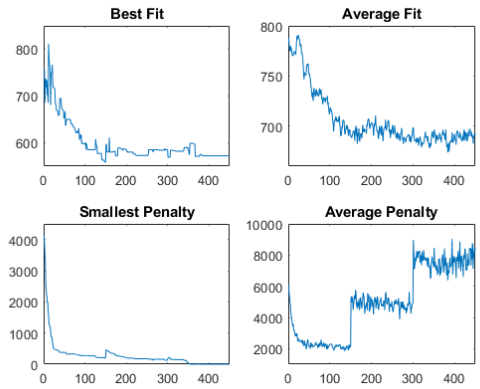
3 Results

We simulated the algorithm to run a production schedule optimization with a dynamic penalty and a decision operation using simulated annealing after mutation for 50 entities with predefined move times between entities. The termination of the simulation occurs if the smallest dynamic penalty is sufficiently close to zero, which means that the found approximate solution is not an approximation of the local solution. The entire simulation was programmed in the Matlab environment.

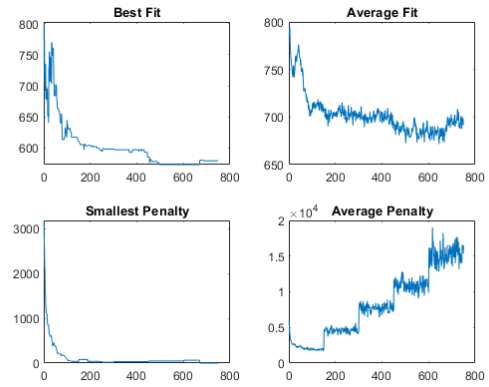
Of course, we could explore countless cases with different input conditions or different borderline situations, but for reasons of capacity we will limit ourselves to a minimalistic abridgement for one case of input conditions. The results are in the following graphs (Graphs 2-4).

4 Discussion

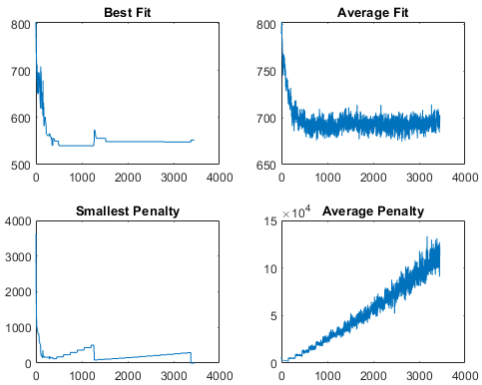
The first and second attempts decrease quite clearly, although the second attempt shows a strong influence of the increasing penalty caused by the penalty of simulated annealing. This also caused a higher number of generations of the genetic algorithm simulation. But the third case is somewhat borderline, as the penalty grew exponentially. However, this was apparently caused by a sharp drop in the fit value at the beginning of the simulation, i.e. the "annealing temperature", which subsequently caused a sharp increase in the penalty. Even this did not cause the divergence of the algorithm, which stopped after a higher number of generations, but with an acceptable value of the fit function.



Graph 2: Simulation (first attempt) results for 50 randomly selected flow-shop operations with predefined times between operations, with a population size of 200.



Graph 3: Simulation (second attempt) results for 50 randomly selected flow-shop operations with predefined times between operations, with a population size of 200.



Graph 4: Simulation (third attempt) results for 50 randomly selected flow-shop operations with predefined times between operations, with a population size of 200.

5 Conclusion

Genetic algorithms are widely used in practice, not only technical and economic, but also, as we can see, in production processes. As a method for solving both constrained and unconstrained optimization problems that is based on natural selection, the process that drives bio-logical evolution can be the genetic algorithm apply the genetic algorithm to solve a variety of optimization problems that are not well suited for standard optimization algorithms, including problems in which the objective function is discontinuous, nondifferentiable, stochastic, or highly nonlinear. All three mentioned ways of improving this algorithm not only speed up the algorithm but also increase its reliability.

Literature:

1. GAREY M. R., JOHNSON D. S., RAVI SETHI: *The Complexity of Flow-shop and Job-shop Scheduling*, Mathematics of Operations Research, Volume 1 (Issue 2), 1976. 117-129 p.
2. COBHAM A.: *The intrinsic computational difficulty of functions. Logic, methodology and philosophy of science*, Proceedings of the 1964 International Congress, edited by Yehoshua Bar-Hillel, Studies in logic and the foundations of mathematics, North-Holland Publishing Company, Amsterdam 1965. 24–30 p.
3. HOLLAND J. H.: *Adaptation in natural and artificial systems*, University of Michigan Press, Ann Arbor, MI, 1975.
4. FOGEL D. B.: *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*, 3rd Edition, Wiley-IEEE Press Series on Computational Intelligence, 2005. 296, ISBN 978-0-471-66951-7.
5. MATHWORKS: *MATLAB Documentation*, The MathWorks, Inc., 2023.
6. GLOVER F.: *Future Paths for Integer Programming and Links to Artificial Intelligence*, Computers and Operations Research, Volume 13 (Issue 5), 1986. 533–549 p.
7. GLOVER F.: *Tabu Search - Part I.*, ORSA Journal on Computing, Volume 1 (Issue 3), 1989. 190-206 p. ISSN 0899-1499.
8. GLOVER F.: *Tabu Search—Part II.*, ORSA Journal on Computing, Volume 2 (Issue 1), 1990. 4-32 p. ISSN 0899-1499.
9. COURANT R.: *Variational methods for the solution of problems of equilibrium and vibrations*, Bulletin of the American Mathematical Society, Volume 49 (Issue 1), 1943, 1-23 p., ISSN 02730979.
10. GEN M., CHENG R.: *A survey of penalty techniques in genetic algorithms*, Proceedings of IEEE International Conference on Evolutionary Computation, Nagoya, Japan, 1996. 804–809 p. ISBN 0-7803-2902-3.
11. DEB K.: *Optimization for Engineering Design: Algorithms and Examples*, Prentice-Hall, New Delhi, 1995. 421 p. ISBN 978-81-203-4678-9.
12. OTTEN R. H. J. M., GINNEKEN L. P. P. P.*: Annealing Algorithm*. Kluwer, Boston: 1989. 201 p. ISBN 0-7923-9022-9.
13. KIRKPATRICK S., GELLAT Jr. C. D., VECCHI M. P.: *Optimization by Simulated Annealing*, Science 220 (Issue 4598) 1983. 671-680 p.
14. ČERNÝ V.: *Thermodynamical approach to the traveling salesman problem*. An efficient simulation algorithm, Journal of Optimization Theory and Applications 45, 1985. 41–51 p.
15. LAARHOVEN P. J. M., AARTS E. H. L.: *Simulated Annealing. Theory and Applications*, Reidel, Dordrecht 1987. 157-187 p. ISBN 978-94-015-7744-1.
16. REKLAITIS G. V., RAVINDRAN, A., RAGSDELL, K. M.: *Engineering optimization: Methods and applications*. Wiley, 1981. 973 p.

Primary Paper Section: B

Secondary Paper Section: BB, BC