Extending Microsoft® Project for Real-World Job-Shop Scheduling with Genetic Algorithm

Peter Steininger
Steinbuch Center for Computing (SCC)
Karlsruhe Institute of Technology (KIT)
Kaiserstrasse 12
D-76128 Karlsruhe, Germany
steininger@KIT.edu

Abstract — **Although production scheduling has attracted the research interest of production economics communities for decades, there still remains a gap between academic research and real-world problems. Genetic Algorithms (GA) constitute a technique that has already been applied to a variety of combinatorial problems. We will explain the application of a GA approach to bridge this gap for job-shop scheduling problems, for example to minimize makespan of a production program or to increase the due-date reliability of jobs. The presented approach focuses on integrating a scheduling algorithm, based on GA, into a commercial software product, namely Microsoft Project 2003. We extended Microsoft Project in a range of aspects: A new graphical user interface is introduced to support users by a guided wizard describing the problem for which an optimal schedule is sought. The GA was developed to search for the solution with the maximum results for a given set of production logistical objectives. The developed GA algorithm and operators are tested on real-world data from a one-of-a-kind manufacturing department of a major company. It includes different aggregation operators for combining objectives. Furthermore, the efficiency of the algorithm was compared to benchmark tests available in literature.**

Keywords: Job-Shop Scheduling, Genetic Algorithms, Job-Shop Benchmarks, Real-World Scheduling Problems

I. PROBLEM STATEMENT

A. Characteristics of job-shop scheduling problems

Many jobs in industry and elsewhere require a collection of tasks or activities to be completing while satisfying temporal, resource and precedence constraints. Temporal constraints impose that some activities, or a set of them have to be started or finished before or only after a certain point in time. Resource constraints dictate that two tasks requiring the same resource cannot be carried out simultaneously. Precedence constraints refer to the technological winding-up of a job. The objective is to create a schedule specifying when each task is to begin (or finish) and what resources it shall use in order to satisfy all the constraints while pursuing an objective, e.g., taking as little overall time as possible, minimizing mean delay, minimizing maximum delay, minimizing the number of late jobs and so on. This is known as the job-shop scheduling problem (JSP).

The JSP is a very important and well-studied scheduling problem. It is a basic model, which may be extended by additional characteristics like buffers, transportation, setup time, time lags, etc., allowing practical scheduling problems in practice are to be modeled more precisely. In its general form, it is $\mathcal{NP}$-complete, meaning that there is probably no efficient procedure for exactly finding the shortest schedule for arbitrary instances of a problem.

Bagchi [1, p. 109] references the JSP as follows: "*Within the great variety of production scheduling problems that exist, the job shop scheduling problem (JSP) is one that has generated the largest number of studies. It has also earned a reputation for being notoriously difficult to solve. Nevertheless, the JSP illustrates at least some of the demands imposed by a wide array of real world scheduling problems… Attempts to tackle the multi objective job shop are still relatively few.*" A JSP is usually solved using a heuristic algorithm that takes advantage of special properties of a specific class of instances. This can be regarded as a backdoor to reducing the complexity of a given problem.

B. Formal problem description

An instance of the JSP consists of a set of NOJ jobs i and NOM machines j. Each job consists of a number of activities so that we can count the total number of activities NOA as follows:

$$NOA = \sum_{i=1}^{NOM} NOJ_i \tag{1}$$

Each job has fixed number and sequence of activities. Each activity has certain duration and requires a single machine for its entire duration. The activity following a preceding one within a job requires a different machine. An activity must be finished before each activity following it in its job. Two activities cannot be scheduled at the same time if they both require the same machine. The problem is to find a feasible schedule which minimizes some objective function, e.g., minimizing makespan, in other words the overall completion time of all activities, see Steininger [15, p. 26 f.]. These results in a complexity function for the JSP expressed as

$$O\left((NOJ!)^{NOM}\right) \tag{2}$$

which means in order to find the best schedule for a problem instance, we have to enumerate and evaluate all possible

 ISBN: 978-1-61208-172-4

schedules and the number of possible schedules to enumerate is the result of function (2).

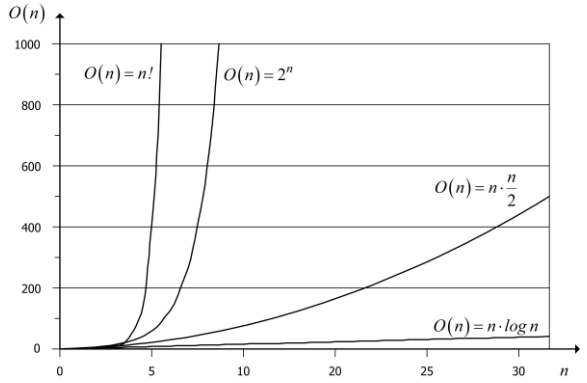

Figure 1. Selected complexity functions

Figure 1 illustrates the dimension of selected complexity functions, where n is the number of problem elements, here the number of activities. The graph illustrates that the complexity of JSP is much bigger than some other well-known problems, such as "Permutation problem" which is $O(n) = n!$, "Towers of Hanoi" which is $O(n) = 2^n$, "Quicksort" which is $O(n) = n \cdot \log n$, and so on.

C. 1.3 Classification of scheduling problems

Classes of scheduling problems can be specified in terms of the three-field classification approach initial introduced by Conway, Maxwell and Miller [5] and extended by Graham [12] and Brucker [3], which is under a continuous reconsideration.

The three-field classification is described as $\alpha \mid \beta \mid \gamma$, where α specifies the machine environment, β specifies the job characteristics and γ describes the objective function or a combination of objective functions. Using the three-field classification to specify the problem instance of the JSP we are examining, the following taxonomy is noted:

$$J, NOM \mid NOJ, \text{intree}, t_{ij} \mid C_{\max} \qquad (3)$$

Formula (3) describes a class of scheduling problems as JSP (J) with a fixed machine count of NOM and a predefined and fixed number of NOJ jobs. The order of activities in each job is predefined and fixed as a directed graph with operation times (t_{ij}), expresses as integer values, for each task.

The three-field classification notes γ as the objective function or a combination of objective functions. In formula (3) γ specifies the "traditional" objective function ($C_{\max}$), which de-scribes our goal as taking as little makespan as possible for the schedule of all NOJ jobs using NOM machines.

II. MODELLING OF JSP SCHEDULING PROBLEMS

A. Formal problem representation

Even slightly different job-shop problems require completely different encodings in order to find a good solution.

Thus, choosing a good representation is a very important component of solving a JSP. However, choosing a good representation for a scheduling problem is as difficult as choosing a good search algorithm for a search problem. Not all algorithms work equally efficient on a specific problem representation. To describe the representation technique developed for our solution we use a simple job-shop scheduling problem as shown in Table I.

Job j	**Machine** / S_j	t_{ij} [TU]		
		1$_i$	**2**	**3**
1	(3,2,1)	3	5	1
2	(1,2,3)	3	2	1
3	(2,1,3)	1	2	5

Table I. Example of a production schedule problem with 3 jobs, routing information S_j for jobs, 3 machines and task operation times.

The scheduling problem can be represented by a graph as shown in Table I. In addition to the activity nodes (j,i) , it contains a source node a with no duration (operation time), a sink node e , also with no duration (operation time), and two nodes called r_2 and r_3 which describe an imposed later start of job 2 and 3 relative to job 1.

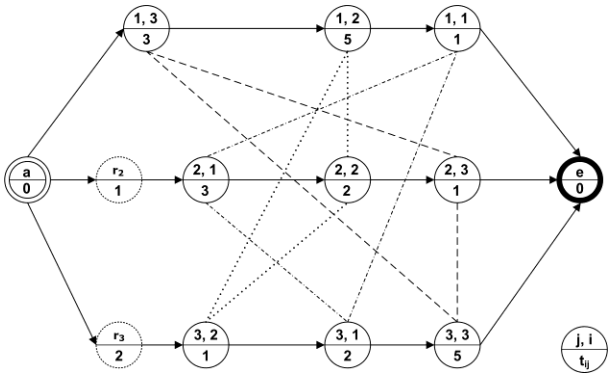

Figure 2. Network representation of a JSP based on Table I

The directed arcs running from the source node, through each activity node (j,i) to the sink node describes the technological sequence of activities based on S_j in Table I. Each node shows the job id, the machine needed and the operation time t_{ij} . There are also undirected arcs in the network, which references all possible sequences of an activity of a given job on a specific machine. Such a representation is called a disjunctive network.

B. Data representation and problem reduction

Care must be taken when adopting such a graphical representation into a data structure, especially for the JSP in an area with hundreds of jobs, thousands of sequences and millions of possible activity orders at specific machines.

A data structure which is very efficient in the use of storage (because of the size of a practical problem) as well as in

time has to be found to represent the introduced network. Gallo and Pallottino [10] introduced the so-called "Forward Star" data structure, which is the most efficient representation of all existing network data structures for representing a network [4]. The "Forward Star" data structure uses three arrays to describe a (directed) network. First we have an array called *from*, whose index represents all nodes of the network. The value of an index field references the index of the second array called *succ*, which is the index of a node to connect, referencing *from*. The third array is called *cost* and reports the cost of a specific arc connecting two nodes.

The "Forward Star" data structure allows for a perfect implementation of the activity order of any JSP, an efficient implementation in storage and time and a reduction of the initial problem described by the $\alpha \mid \beta \mid \gamma$ three-field classification. Using the "Forward Star" data structure our problem is reduced to the following taxonomy ():

$$J, NOM \mid NOJ, chains, t_{ij} \mid C_{\max} \quad (4)$$

As mentioned above, the selection of a good representation is very important for the solution of a JSP. Care must also be taken to adopt both representational schemes and the associated operators for an effective algorithm. When using the traditional way of solving a problem with GAs, the chromosomes are implemented as binary vectors. This simple representation is an excellent choice for problems in which a point naturally maps into a chromosome of zeros and ones. Unfortunately, this approach of zeros and ones cannot be used for real-word engineering problems such as JSP, because of the number of information needed to represent coding of the JSP. Therefore, we have to find a way to integrate the "Forward Star" data structure into a GA.

III. GENETIC ALGORITHMS

A. General principle

The term Genetic Algorithm describes a set of methods, which can be used to optimize complex problems. As the name suggests, the processes employed by GAs are inspired by natural selection and genetic variation. This GA uses a population of possible solutions to a problem and applies a cycle of processes to modify them. These processes mimic those in nature in such a way that subsequent populations are fitter and more adapted to their environment. As generations progress over time, they become better suited to their environment and provide an advantageous solution in a given time.

Since their development in the late 1980's GAs [11] have been used to find solutions too many types of problems. A unique characteristic of a GA is that the fundamental algorithm is unaware of the problem it is optimizing. All that is required is that the parameters entered into a model or system can be efficiently transformed to and from a suitable GA chromosome format. This means GA optimization can be applied to many types of complex problems. Detailed introductions are given by Goldberg [11] and Davis [7].

The flowchart for the GA is given in Figure 3. First, an initial population of randomly generated sequences of the tasks in the schedule is created. These individual schedules form chromosomes which are subject to a form of evolution. Once an initial population has been formed, "selection", "crossover" and "mutation" operations are performed repeatedly until the fittest member of the evolving population converges to an optimal fitness value. Alternatively, the GA may run for a user-defined number of iterations [11].

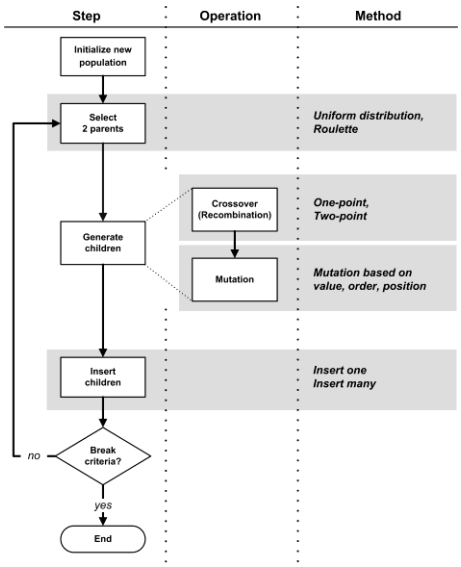

Figure 3. Principal flow of Genetic Algorithms.

The size of the population is user-defined and the fitness of each individual, in this case a schedule, is calculated according to a function, in our case the makespan or an additive combination of different goals. It is also possible to use a fitness function on other calculated values like mean or maximum delay, number of delayed jobs and so on. Also combinations of different functions are possible. The schedules are then ranked according to the value of their fitness function and, after that, selected for reproduction.

B. Schedule encoding and decoding

GAs were derived by examining biological systems. In biological systems evolution takes place on chromosomes which are organic devices for programming the structure of living beings. In this sense, a living being is a decoded structure of all chromosomes. Natural selection is the link between chromosomes and the performance of the decoded structures. When implementing the GA, the variables that characterize an individual are represented in arrays (by index ordered lists). Each variable corresponds to a gene and the array corresponds to a chromosome in biological systems.

Decision was made [15] to use the encoding schema introduced by Bean [2] to build the chromosomes. He calls his schema "Operation Based Representation". Encoding starts by enumerating jobs and corresponding activities in a list. Each activity in a job is encoded with the numerical id of the job in which it resides. All jobs and activities are encoded following that description in one potential schedule for the problem. The result is a chromosome which represents a potential schedule.

 ISBN: 978-1-61208-172-4

C. Crossover

The GA uses crossover, where mating chromosomes are cut once. Crossover is the most delicate operation of GA because of the problem that it can produce irregular activity sequences for a job. We use a corrected 2-point crossover to avoid non-regular activity sequences of orders, which Goldberg [11] refers to as a PMX-crossover operator. The following figures will illustrate a crossover operation of a sample JSP with 4 jobs, each with 3 activities, and 4 machines (see Figure 4 to Figure 8), based on an example in Steininger [15, p. 146 f.].

Step 1: Select two individuals randomly from the population (Figure 4).

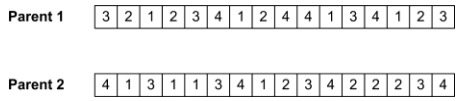

Figure 4. Crossover operator, step 1: Parent selection.

Step 2: Select a segment of the first chromosome which starts and ends with the same job number (Figure 5). Selected segment: 4124.

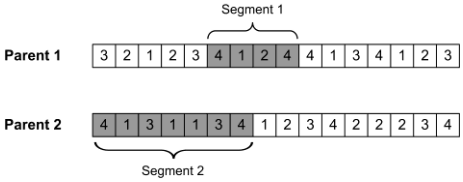

Figure 5. Crossover operator, step 2: Segment selection.

Step 3: Select a segment of the second chromosome which starts and ends with the same job number as the selected segment of the first chromosome (Figure 6). Selected segment: 4131134.

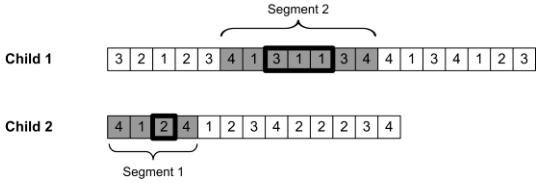

Figure 6. Crossover operator, step 4: Segment exchange.

Step 4: Exchange the selected segments between the two chromosomes to get the "child". The result is two chromosomes with non-regular activity sequences of jobs. Child 1 has too many activities and child 2 lacks some genes/activities (Figure 7). This result necessitates a correction step.

Step 5: The following process, called "normalization", initializes this correction step. It examines the original segment of a child with the exchanged segment of the same child (Figure 7). The result of that examination for child 1 is: 2 are missing; 3, 1, 1 and 3 are added. For child 2 we get: 3, 1, 1 and 3 are missing and 2 is added.

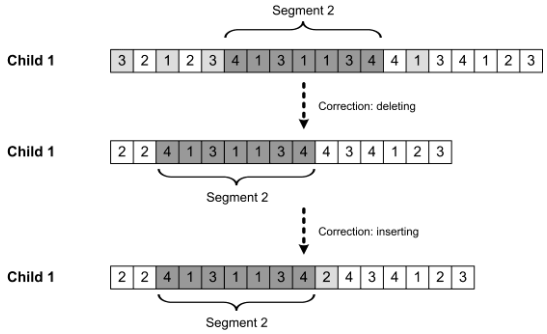

Figure 7. Crossover operator, step 5: Correction of child 1.

Step 6: Having detected the added and missed genes, we can start the correction step: For child 1 we delete 3, 1, 1 and 3 at first occurrence in child 1 without inspecting the exchanged segment; for child 1 we add 2 at the end of exchanged segment (Figure 8). The same operations are executed on child 2 and, at the end of all steps; we have two new and correct chromosomes.

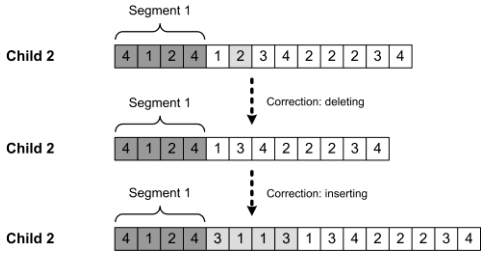

Figure 8. Crossover operator, step 6: Correction of child 2.

D. Mutation

Mutation is the process of random dissimilarity of the value of a gene with small probability. It is not a primary operator, but it ensures that the possibility of searching any section in the problem space is never zero and prevents complete loss of genetic material through reproduction and crossover. We execute the mutation operator as a permutation by first picking (and deleting) a gene before reinserting it at a randomly chosen position of the permutation (Figure 9).

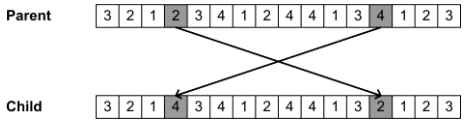

Figure 9. Mutation operator: Gene flipping.

E. Fitness

The fitness function is used to evaluate the fitness of each individual in the population and depends on the specific application. Since a GA proceeds toward more fit individuals and the fitness value is the only information available to the GA, the performance of the algorithm is highly sensitive

to the fitness function. In case of optimization routines, the fitness is the value of the objective function to be optimized.

F. Selection

To selectively reproduce the population and to determine the next generation we use a hit and miss selection procedure based on the fitness function. This could be implemented using a roulette wheel method. An imaginary roulette wheel is constructed with a segment for each individual in the population. An individual's section size is based on the fitness value of the particular individual. A fit individual will occupy a larger slice of the roulette wheel than a weaker one. Selection is made by rotating the roulette wheel a number of times equal to the population size. When the roulette wheel stops, the individual it points to is selected. This means that fitter individuals will have a propensity to be selected more frequently than weaker ones.

The GA needs a few additional parameters to work. These parameters specify the size of the population, use of operators and so on.

G. Population size

The population size depends on the nature of the problem [13]. Typically, it contains several hundreds or thousands of possible solutions. The population is generated randomly, so it is possible to cover the entire range of possible solutions. We use a population size of 500 individuals, which represents 500 possible schedules.

H. Probability of crossover

The parameter probability of crossover affects the rate at which the crossover operator is applied [11]. A higher crossover rate introduces new chromosomes more quickly into the population. If the crossover rate is too high, good individuals are eliminated faster than selection can produce improvements. A low crossover rate may cause stagnation due to the lower exploration rate. We use probability of crossover with a value of 0.6.

I. Probability of mutation

Probability of mutation is the likelihood with which each gene of each individual in the new population undergoes a random change after a selection process. A low mutation rate helps to prevent any gene positions from getting stuck to single values, whereas a high mutation rate results in essentially random search [11]. We use a value of 0.05 for mutation probability.

J. Final result

It is a characteristic of the GA that once fairly good solutions have been found their features will be carried forward into even better solutions, which will ultimately lead to a near-optimal solution. Therefore, GAs are particularly attractive for scheduling.

Compared with other optimization methods, GAs are suitable for traversing large search spaces since they can do this relatively rapidly and because the mutation operator diverts the method away from local minima. Being suitable for large search spaces is a useful advantage when dealing with schedules of increasing size since the solution space will grow very rapidly. It is important that these large search spaces are scanned as fast as possible to enable the practical and useful implementation of schedule optimization.

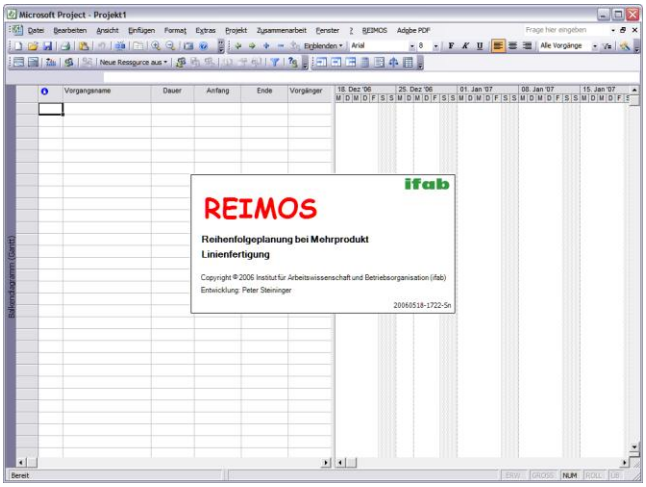

Figure 10. Microsoft Project 2003 with integrated REIMOS.

IV. APPLICATION EXAMPLES

A. Scheduling problem in a one-of-a-kind manufacture

We have tested our approach, *REIMOS* (*German abbreviation for "Sequence planning for multi-product manufacturing systems"*) [15], in a one-of-a-kind manufacture of a major German company. For confidentiality reasons, the model mix, job and operation data are under a non-disclosure agreement and we are not allowed to publish the data. But we have also tested our approach with a few publicly available benchmark sets for JSP, which can be acquired by any researcher from a public website (Taillard [16], e.g. Figure 12).

B. Benchmark tests

Early on, research of scheduling problems started a competition on the "best schedule" of a specific problem. For that reason some researchers designed (calculated) very hard to solve scheduling problems as so called "benchmark instances" [9]. Some modern benchmark instances are distributed by Taillard [16] and called JSP-15-15, JSP-20-15 and JSP-50-20. The name of the benchmark is derived from scheduling a problem; in the case of JSP-15-15 it stands for 15 jobs on 15 machines and so on (Figure 11). Taillard [16] also published the list of best results for each scheduling problem instance, allowing our own results to be compared with those of other researchers.

We used three benchmark instances to model Microsoft Project 2003 files as input for *REIMOS* and started a GA run with the parameter values mentioned above (3.8, Figure 11 & 12). Our algorithm for benchmarking was modified to write a so-called "debugging output", so we know specific values for the GA at every step. For example, these values are: number of iterations, best calculated schedule and so on. We compared the best schedule found thus far with the best known schedule of Taillard [16] and calculated a quality

 ISBN: 978-1-61208-172-4

level, which is the percentage reached of so far “best” known solution by Taillard [16]. This is shown in Figure 12 for JSP-15-15.

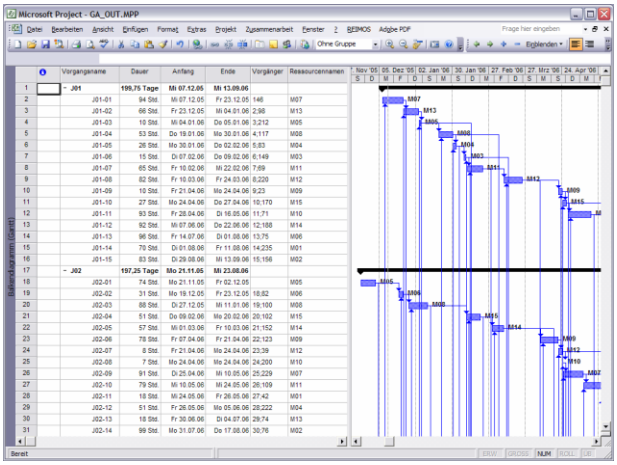

Figure 11. Benchmark instance JSP-15-15 in Microsoft Project 2003.

For all benchmark instances we attained a quality level around 95 % of the best known solution calculated over 1000 generations GA runtime. On a so-called standard PC suitable for the use of Microsoft Office, one calculation run around 10 to 20 minutes, which were our time goal for planning a schedule. We could have gone even further and let it run for hours or days, but the effect would not bring a realistic benefit. The problem with the best known solution by Taillard [16] is that we did not know its runtime, implementation of algorithm, computer etc. So it was hard to say "how good was best" from an economical point of view.

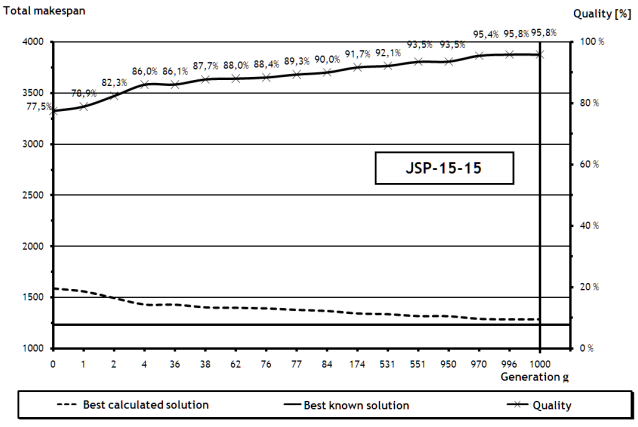

Figure 12. Results of benchmark instance JSP-15-15 with REIMOS.

V. CONCLUSION AND OUTLOOK

A computer algorithm being based on the evolution of living beings may be surprising, but the extensiveness with which this approach is applied in so many areas is even more surprising. Genetic algorithms have already proven their efficiency in many application areas, commercial, educational and scientific. Their usefulness in solving various kinds of problems have made them a preferable choice compared to traditional, mainly heuristic approaches.

The adaptation of a GA to schedule jobs in manufacturing shops with time, resource and precedence constraints has been demonstrated here [15]. The simplicity of the methods used supports the assumption that GA can provide a highly flexible and user-friendly solution to the JSP. The use of standard software and an implemented "add-in" for Microsoft Project 2003 to realize the GA has shown that this approach can be used for solving real industrial scheduling problems [15].

VI. REFERENCES

[1] Bagchi, Tapan P., 1999. Multiobjective Scheduling by Genetic Algorithms. Boston, Dordrecht, London: Kluwer Academic Publishers.

[2] Bean, James C., 1994. Genetic Algorithms and Random Keys for Sequencing and Optimizations. ORSA Journal of Computing. 6 (2), 154-160.

[3] Brucker, Peter, 2004. Scheduling Algorithms. Berlin, Heidelberg, New York et al.: Springer Verlag.

[4] Cherkassky, B. V., Goldberg, A. V., Radzik, T., 1993. Shortest Paths Algorithms: Theory and Experimental Evaluation. Technical Report 93-1480, Computer Science Department, Stanford University.

[5] Conway, Richard W., Maxwell, William L., Miller, Louis W., 2003. Theory of Scheduling. Mineola (NY): Dover Publications.

[6] Darwin, Charles, 1859. On the Origin of Species by Means of Natural Selection. London (UK): John Murry.

[7] Davis, Lawrence, 1996. Handbook of Genetic Algorithms. Florence (KY): International Thomson Computer Press.

[8] Domschke, Wolfgang, Scholl, Armin, Voß, Stefan, 1997. Produktionsplanung. Berlin, Heidelberg, New York et al.: Springer Verlag.

[9] Fisher, Howard, Thompson, Gerald L., 1963. Probabilistic Learning Combinations of Local Job-Shop Scheduling Rules. In: Muth, John F.; Thompson, Gerald L., (Eds.). Industrial Scheduling. Englewood Cliffs (NJ): Prentice-Hall, 225-251.

[10] Gallo, Giorgio, Pallottino, Stefano, 1982. A new Algorithm to find the Shortest Paths between all Pairs of Nodes. Discrete Applied Mathematics. Amsterdam, 3 (4), 23-25.

[11] Goldberg, David E., 1989. Genetic Algorithms in Search, Optimization and Machine Learning. München: Addison Wesley.

[12] Graham, Ronald L., Lawler, Eugene L., Lenstra, Jan Karel et al., 1979. Optimization and approximation in deterministic sequencing and scheduling. Annals of Discrete Mathematics, 16 (5), 287-326.

[13] Haupt, R. L., 2000. Optimum population size and mutation rate for a simple real genetic algorithm that optimizes array factors. IEEE Antennas and Propagation Society International Symposium, 2, 1034-1037.

[14] Roy, Bernard, Sussmann, B., 1964. Les problèmes d'ordonnancement avec contraintes disjonctive. Montrouge (F): SEMA Groupe.

[15] Steininger, Peter, 2007. Eine Methode zur Reihenfolgeplanung bei Mehrprodukt-Fertigungssystemen. Dissertation, Universität Karlsruhe. Aachen (D): Shaker Verlag.

[16] Taillard, Éric D., 2006. Scheduling instances. http://ina2.eivd.ch/Collaborateurs/etd/problemes.dir/ordonnancement.dir/ordonnancement.html.

 ISBN: 978-1-61208-172-4