Use of an Artificial Immune System for Job Shop Scheduling

Carlos A. Coello Coello, Daniel Cortés Rivera and Nareli Cruz Cortés

CINVESTAV-IPN
Evolutionary Computation Group
Depto. de Ingeniería Eléctrica
Sección de Computación
Av. Instituto Politécnico Nacional No. 2508
Col. San Pedro Zacatenco
México, D. F. 07300
ccoello@cs.cinvestav.mx
{dcortes,nareli}@computacion.cs.cinvestav.mx

Abstract. In this paper, we propose an algorithm based on an artificial immune system to solve job shop scheduling problems. The approach uses clonal selection, hypermutations and a library of antibodies to construct solutions. It also uses a local selection mechanism that tries to eliminate gaps between jobs in order to improve solutions produced by the search mechanism of the algorithm. The proposed approach is compared with respect to GRASP (an enumerative approach) in several test problems taken from the specialized literature. Our results indicate that the proposed algorithm is highly competitive, being able to produce better solutions than GRASP in several cases, at a fraction of its computational cost.

1 Introduction

Scheduling problems arise in all areas. The purpose of scheduling is to allocate a set of (limited) resources to tasks over time [26]. Scheduling has been a very active research area during several years, both in the operations research and in the computer science literature [2, 1, 18]. Research on scheduling basically focuses on finding ways of assigning tasks (or jobs) to machines (i.e., the resources) such that certain criteria are met and certain objective (or objectives) function is optimized.

Several heuristics have been used for different types of scheduling problems (e.g., job shop, flowshop, production, etc.): evolutionary algorithms [8, 9], tabu search [4], and simulated annealing [7], among others. Note, however, that the use of artificial immune systems for the solution of scheduling problems of any type has been scarce (see for example [15, 14, 10]).

This paper introduces a new approach, which is based on an artificial immune system and the use of antibody libraries and is applied for optimizing job shop scheduling problems. The proposed approach is compared with respect to GRASP (Greedy Randomized Adaptive Search Procedure) in several test problems taken from the specialized literature. Our results indicate that the proposed approach is a viable alternative for solving efficiently job shop scheduling problems.

2 Statement of the Problem

In this paper, we will be dealing with the Job Shop Scheduling Problem (JSSP), in which the general objective is to minimize the time taken to finish the last job available (makespan). In other words, the goal is to find a schedule that has the minimum duration required to complete all the jobs [2]. More formally, we can say that in the JSSP, we have a set of n jobs $\{J_i\}_{1 \leq j \leq n}$, that have to be processed by a set of m machines $\{M_i\}_{1 \leq r \leq m}$. Each job has a sequence that depends on the existing precedence constraints. The processing of a job J_j in a machine M_r is called operation $O_j r$. The operation $O_j r$ requires the exclusive use of M_r for an uninterrupted period of time $p_j r$ (this is the processing time). A schedule is then a set of duration times for each operation $\{c_j r\}_{1 \leq j \leq n, 1 \leq r \leq m}$ that satisfies the previously indicated conditions. The total duration time required to complete all the jobs (makespan) will be called L. The goal is then to minimize L.

Garey and Johnson [19] showed that the JSSP is an **NP-hard** problem and within its class it is one of the least tractable problems [1]. To exemplify this statement is sufficient to mention that a 10×10 problem proposed in [21] remained without solution for over 20 years. Several enumerative algorithms based on *Branch & Bound* have been applied to JSSP. However, due to the high computation cost of these enumerative algorithms, some approximation approaches have also been developed. The most popular practical algorithm to date is the one based on *priority rules* and *active schedule generation* [17]. However, other algorithms, such as an approach called *shifting bottleneck* (SB) have been found to be very effective in practice [3]. Furthermore, a number of heuristics have also been used in the JSSP (e.g., genetic algorithms, tabu search, simulated annealing, etc.).

The only other attempt to solve the JSSP using an artificial immune system that we have found in the literature is the proposal of [14, 15]. In this case, the authors use an artificial immune system (adopting a traditional permutation representation) in which an antibody indirectly represents a schedule, and an antigen describes a set of expected arrival dates for each job in the shop. The schedules are considered to be dynamic in the sense that sudden changes in the environment require the generation of new schedules. The proposed approach compared favorably with respect to a genetic algorithm using problems taken from [20]. However, the authors do not provide enough information as to replicate their results (the problems and results obtained are not included in their papers).

3 Description of our Approach

Our approach is based on two artificial immune system mechanisms:

1. The way in which the molecules called antibodies are created. An antibody is encoded in multiple gene segments distributed along a chromosome of the genome. These segments must be placed together to make one antibody. In order to make such a molecule (i.e., an antibody), the gene segments are concatenated (see Figure 1). Note that other authors have used this sort of encoding of the antibodies in their corresponding computational models (e.g., [16, 25, 24]).

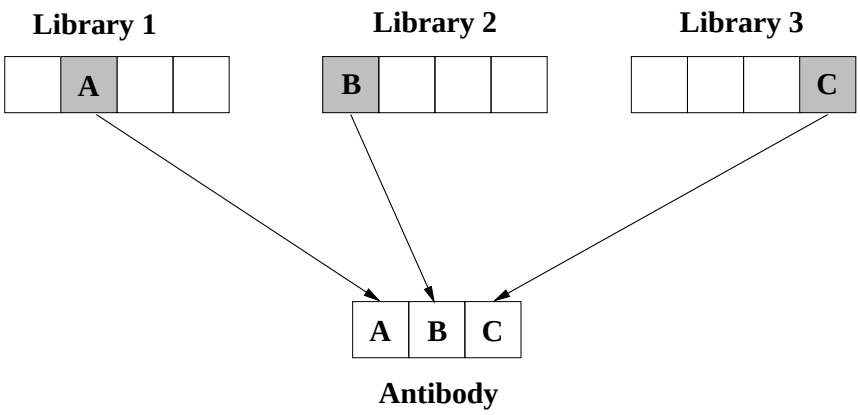

Fig. 1. Building antibody molecules from gene libraries (taken from [11]).

2. The clonal selection principle.

The approach proposed in this paper is a variation of CLONALG, which is an artificial immune system based on the clonal selection principle that has been used for optimization [23]. CLONALG uses two populations: one of antigens and another one of antibodies. When used for optimization, the main idea of CLONALG is to reproduce individuals with a high affinity, then apply mutation (or blind variation) and select the improved maturated progenies produced. Note that "affinity" in this case, is defined in terms of better objective function values rather than in terms of genotypic similarities (as, for example, in pattern recognition tasks), and the number of clones is the same for each antibody. This implies that CLONALG does not really use antigens when solving optimization problems, but, instead, the closeness of each antibody to the global optimum (measured in relative terms with respect to the set of solutions produced so far) defines the rate of hypermutation to be used. It is also worth noting that CLONALG does not use libraries to build antibodies as in our approach.

In order to apply an artificial immune system (or any other heuristic for that sake) to the JSSP, it is necessary to use a special representation. In our case, each element of the library represents the sequence of jobs processed by each of the machines. An antibody is then a chain with the job sequence processed by each of the machines (of length $m \times j$). An antigen is represented in the same way as an antibody. The representation adopted in this work is the so-called *permutations with repetitions* proposed in [27].

To illustrate this representation, we will consider the 6×4 problem (6 jobs and 4 machines) shown in Table 1.

Input data include the information regarding the machine in which each job must be processed and the duration of this job in each machine. Gantt diagrams are a convenient tool to visualize the solutions obtained for a JSSP. An example of a Gantt diagram representing a solution to the 6×4 problem previously indicated is shown in Step 1 of Figure 2. Figure 2 also requires some further explanation:

job	machine (time)			
1	1(2)	2(2)	3(2)	4(2)
2	4(2)	3(2)	2(2)	1(2)
3	2(2)	1(2)	4(2)	3(2)
4	3(2)	4(2)	1(2)	2(2)
5	1(2)	2(2)	3(4)	4(1)
6	4(3)	2(3)	1(1)	3(1)

Table 1. A problem of size 6×4

- The string at the bottom of Figure 2 corresponds to the solution that we are going to decode.
- **Step 1**: This shows the decoding before reaching the second operation of job 2.
- **Step 2**: This shows the way in which job 2 would be placed if a normal decoding was adopted. Note that job 2 (J_2) is shown to the extreme right of machine 3 (M_3).
- **Step 3**: Our approach performs a local search to try to find gaps in the currrent schedule. Such gaps should comply with the precedence constraints imposed by the problem. In this case, the figure shows job 2 placed on one of these gaps for machine 3.
- **Step 4**: In this case, we apply the same local search procedure (i.e., finding available gaps) for the other machines. This step shows the optimum solution for this scheduling problem.

Our approach extends the algorithm (based on clonal selection theory) proposed in [22] using a local search mechanism that consists of placing jobs in each of the machines using the available time slots. Obviously, this mechanism has to be careful of not violating the constraints of the problem.

Our approach is described in Algorithm 1. First, we randomly generate an antibody library. Such a library is really a set of strings that encode different job sequences for each machine. Then, we generate (also randomly) an antigen, which is a possible solution (i.e., a jobs sequence) to the problem. After that, we generate a single antibody by combining different segments taken from the library. The antibody is decoded and a local search algorithm is used to try to improve it by eliminating the larger gaps between jobs. At this point, the solution encoded by the antibody is compared to the solution encoded by the antigen. If the antibody encodes a better solution, then it replaces the antigen. So, the antigen will be keeping the best solution found along the process. In the following step, we generate N clones of the antibody (N is a parameter defined by the user) and we mutate each of them. From these mutated solutions, we select the best segments produced (i.e., the best jobs sequence for each machine) and we use them to update the library. Comparisons at this point are again made with respect to the antigen, but instead of comparing the entire solution, we only compare job sequences for each machine. Note that we do not select based on the complete solution which minimizes total makespan (such a solution is always located in the antigen), but we look for the best partial solutions to the problem to try to recombine them when building a new antibody. In order to define N (number of clones), we used an incremental approach: we started with 100,000 evaluations and we increased this value only if we considered that

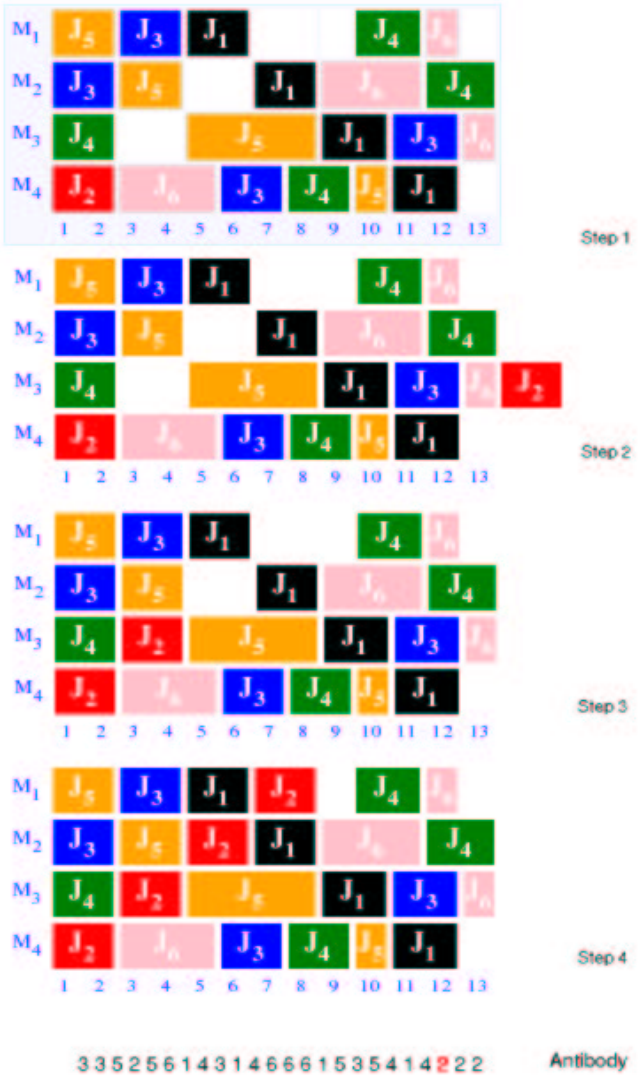

Fig. 2. The graphical representation of a solution to the 6×4 problem shown in Table 1 using a Gantt diagram. The string at the bottom of the figure indicates the antibody that we are going to decode. See text for an explanation of the different steps included.

Algorithm 1 Our AIS for job shop scheduling

Require: Input file (in the format adopted in [5]).
  Input parameters: #antigens, #libraries, mutation rate, random seed (optional)
  p - number of iterations
  i - counter
  Generate (randomly) an antibody library.
  Generate (randomly) an antigen (i.e., a sequence of jobs) and encode it.
  repeat
    Generate an antibody using components from the library.
    Decode the antibody and apply local search to improve it.
    if the antibody is better than the antigen **then**
      Make the antigen the same as the antibody
    end if
    Generate N clones of the antibody
    Mutate each of the clones generated
    Select the best segments produced to update the library
  until $i > p$
  Report the best solution found

the results obtained were not too good. In the number of evaluations reported below for our algorithm, we do not include the cost of fine-tuning the parameter N.

Note that no affinity measure is used. This is mainly due to the representation adopted which allows repetitions. This makes it difficult to define a measure of similary between two sequences of jobs and therefore our choice of not adopting an affinity measure. Thus, we use instead the values of the objective function (minimize makespan) as the affinity measure in order to determine what solutions should be adopted to produce new ones.

4 Comparison of Results

We compared our AIS with respect to the GRASP approach proposed in [6]. We chose this reference for two main reasons: (1) it provides enough information (e.g., numerical results) as to allow a comparison, (2) GRASP is an enumerative approach which has traditionally been found to be very powerful in combinatorial optimization problems such as the job shop scheduling problem studied in this paper [13].

Table 2 shows a comparison of results between our AIS and GRASP, using several test problems taken from the OR-Library [5]. We chose a set of problems that we found to be difficult both for GRASP and for our approach. Note however that better results than those presented here are available in the literature (see for example [27, 12]). However, we decided to compare our approach with respect to [6], because in this reference, we found a more exhaustive table of results (i.e., a larger set of problems was studied by the authors).

All our tests were performed on a PC with an AMD Duron processor running at 1 GHz with 128 MB of RAM and using Red Hat Linux 7.3. Our approach was implemented in C++ and was compiled using the GNU g++ compiler.

The parameters of our approach are the following:

- **Number of libraries**: We adopted between 4 and 8. Note that each library is of the same length as the antibodies and the antigens.
- **Number of antigens**: We used only one antigen in the experiments reported in this paper. However, we experimented with different values (up to 8) and no significant differences were detected in the performance of our approach.
- **Mutation rate**: This value is a function of the antibodies length and it is defined such that 3 mutations take place for each string (i.e., antibody). We used exchange mutation. Thus, we perform a $flip(P_m)$ (the function $flip(P)$ returns true P percent of the times that it is invoked) for each position along the string (P_m is the mutation rate) and if the result is true, then we exchange the current value with another one (randomly chosen) from the same string.
- **Number of clones**: We adopted values between 100 and 1000 depending on the complexity of the problem (these values were empirically found for each problem).

Results are summarized in Table 2. We report the following information:

- **problem:** Name of the problem (as given in the OR-Library [5]).
- **size**: Size of the problem in the format: $m \times j$ (m = number of machines, j = number of jobs).
- **BKS:** Best known solution for each problem.
- **AIS:** Best solution obtained by our AIS (we performed 20 runs per problem).
- **GRASP:** Best solution obtained using GRASP, as reported in [6]. Note that in [6] no statistical values are available for GRASP.
- **AIS err(%)**: Percentage of error (with respect to the best known solution) of our approach.
- **mean AIS**: Mean result of ALL the solutions produced by our approach on the 20 runs performed per problem.
- **sd AIS**: Standard deviation of ALL the solutions produced by our approach on the 20 runs performed per problem.
- **evaluations AIS**: Number of evaluations performed by our approach (expressed in millions).
- **iterations GRASP**: Number of iterations performed by GRASP, as reported in [6]. Note, however, that at each iteration, GRASP builds a valid solution, performs local search and updates the best current solution. This implies multiple evaluations for each iteration. In contrast, what we report for our approach is the total number of evaluations (rather than iterations) performed.

4.1 Discussion of Results

The first important aspect to discuss is the computational cost of the two approaches compared. In Table 2, we can clearly see that in all cases, our AIS performed less evaluations than GRASP. We attribute this lower computational cost of our AIS to the representation adopted, because the *permutations with repetitions* always generate feasible solutions, whereas GRASP uses an encoding based on graphs. In some cases, the differences in computational cost are remarkable. For example (keep in mind that an iteration of GRASP requires more than one evaluation as defined in our AIS):

problem	size	BKS	AIS	GRASP	AIS err(%)	GRASP err(%)	mean AIS	sd AIS	evaluations AIS	iterations GRASP
abz5	10×10	1234	1238	1238	0.3	0.3	1469.7	86.0	5.0	20.1
abz7	15×20	667	707	723	6.0	8.4	839.3	33.3	6.4	20.1
abz8	15×20	670	743	729	10.9	8.8	858.5	31.1	5.0	20.1
abz9	15×20	691	750	758	8.5	9.7	883.5	30.82	5.0	20.1
ft10	10×10	930	941	938	1.2	0.9	1141.4	80.6	20.0	90.1
la01	10×5	666	666	666	0.0	0.0	775.6	57.3	0.01	0.1
la02	10×5	655	655	655	0.0	0.0	775.1	58.0	0.01	0.1
la03	10×5	597	597	604	0.0	1.2	700.1	50.0	10.0	50.1
la04	10×5	590	590	590	0.0	0.0	705.9	62.2	0.01	0.1
la05	10×5	593	593	593	0.0	0.0	616.5	30.7	0.01	0.1
la06	15×5	926	926	926	0.0	0.0	961.3	35.1	0.01	0.1
la07	15×5	890	890	890	0.0	0.0	961.1	44.2	0.01	0.1
la08	15×5	863	863	863	0.0	0.0	964.9	49.5	0.01	0.1
la09	15×5	951	951	951	0.0	0.0	1018.7	43.7	0.01	0.1
la10	15×5	958	958	958	0.0	0.0	981.9	34.3	2.0	50.1
la16	10×10	945	945	946	0.0	0.1	1100.2	67.5	2.0	20.1
la17	10×10	784	785	784	0.1	0.0	911.8	60.0	2.0	20.1
la18	10×10	848	848	848	0.0	0.0	1013.3	68.9	2.0	10.1
la19	10×10	842	848	842	0.7	0.0	1030.8	64.6	10.0	50.1
la20	10×10	902	907	907	0.6	0.6	1072.1	63.3	5.0	10.1
la25	15×10	902	1022	1028	4.6	5.2	1234.9	71.5	5.0	10.1
la28	20×10	1216	1277	1293	5.0	6.3	1554.7	66.6	5.0	10.1
la29	20×10	1195	1248	1293	4.4	8.2	1463.0	55.0	6.4	10.1
la35	30×10	1888	1903	1888	0.8	0.0	2179.4	72.6	5.0	10.1
la36	15×15	1268	1323	1334	4.3	5.2	1560.5	71.22	6.4	11.2
la38	15×15	1217	1274	1267	4.7	4.1	1548.4	61.6	6.4	11.2
la39	15×15	1233	1270	1290	3.0	4.6	1548.3	73.4	6.4	11.2
la40	15×15	1222	1258	1259	2.9	3.0	1537.4	69.3	6.4	11.2
orb02	10×10	888	894	889	0.7	0.1	1069.5	72.75	5.0	40.1
orb03	10×10	1005	1042	1021	3.7	1.6	1275.5	91.6	5.0	40.1
orb04	10×10	1005	1028	1031	2.3	2.6	1220.82	80.1	5.0	40.1

Table 2. Comparison of Results between AIS and GRASP. Note that in all problems we are minimizing makespan. AIS = Artificial Immune System, GRASP = Greedy Randomized Adaptive Search Procedure. The number of evaluations of our AIS and the number of iterations of GRASP are expressed in millions.

– In problem **la03** (10×5), our AIS reaches the best known solution 75% of the time performing 5,000,000 evaluations of the objective function. In contrast, GRASP performs 50,000,000 iterations (ten times more than our AIS).
– In problem **la29** (20×10), the best known solution has a makespan of 1195. In this case, our AIS finds a solution with a makespan of 1248 performing 6.4 millions of evaluations, whereas GRASP finds a solution with a makespan of 1293 after performing 10.1 millions of iterations.
– In problem **abz7** (15×10), our AIS finds a better solution than GRASP (707 vs. 723) performing 6.4 millions of evaluations. GRASP required in this case 20.1 millions of iterations.

In general terms, we can see that our AIS was able to find the best known solution in 38.7% of the problems, whereas GRASP was able to converge to the best known solution only in 35.4% of the problems. It is also interesting to note that both approaches have a similar performance for problems in which 5 machines are used regardless of the number of jobs. However, for larger problems, AIS finds better solutions than GRASP with a lower number of evaluations. Despite the noticeable differences in computational costs of the two algorithms, their percentages of error are very similar. AIS has an average percentage of error of 2.2%, whereas GRASP presents a 2.3%.

All of the previous led us to conclude that our approach is a viable alternative for solving job shop scheduling problems. Our results are not only competitive in terms of the makespan, but were obtained at a fraction of the computational cost required by GRASP.

5 Conclusions and Future Work

We have introduced a new approach based on an artificial immune system to solve job shop scheduling problems. The approach uses concepts from clonal selection theory (extending ideas from CLONALG [23]), and adopts a permutation representation that allows repetitions. The approach also incorporates a library of antibodies that is used to build new solutions to the problem. The comparison of results indicated that the proposed approach is highly competitive with respect to GRASP, even improving on its results some times.

As part of our future work, we plan to improve our procedure to initialize the antibody library by using an additional heuristic (e.g., a genetic algorithm). We also intend to add a mechanism that avoids the generation of duplicates (something that we do not have in the current version of our algorithm). It is also desirable to find a set of parameters that can be fixed for a larger family of problems as to eliminate the empirical fine-tuning that we currently perform. Additionally, we plan to define an affinity measure that can work with our encoding.

Finally, we also plan to work on a multiobjective version of job shop scheduling in which 3 objectives would be considered [1]: 1) makespan, 2) mean flowtime and 3) mean tardiness. This would allow us to generate trade-offs that the user could evaluate in order to decide what solution to choose.

Acknowledgments

The first author acknowledges support from NSF-CONACyT project No. 32999-A. The second author acknowledges support from CINVESTAV-IPN to attend ICARIS'2003. The third author acknowledges support from CONACyT through a scholarship to pursue graduate studies at the Computer Science Section of the Electrical Engineering Department at CINVESTAV-IPN.

References

1. Tapan P. Bagchi. *MultiObjective Scheduling by Genetic Algorithms*. Kluwer Academic Publishers, New York, September 1999. ISBN 0-7923-8561-6.
2. Kenneth R. Baker. *Introduction to Sequencing and Scheduling*. John Wiley & Sons, New York, 1974.
3. J. Adams E. Balas and D. Zawack. The shifting bottleneck procedure for job shop scheduling. *Management science*, 34(3):391-401, 1988.
4. J.W. Barnes and J.B. Chambers. Solving the Job Shop Scheduling Problem using Tabu Search. *IIE Transactions*, 27(2):257–263, 1995.
5. J. E. Beasley. OR-Library: Distributing Test Problems by Electronic Mail. *Journal of the Operations Research Society*, 41(11):1069–1072, 1990.
6. S. Binato, W.J. Hery, D.M. Loewenstern, and M.G.C. Resende. A GRASP for Job Shop Scheduling. In Celso C. Ribeiro and Pierre Hansen, editors, *Essays and Surveys in Metaheuristics*, pages 59–80. Kluwer Academic Publishers, Boston, 2001.
7. Olivier Catoni. Solving Scheduling Problems by Simulated Annealing. *SIAM Journal on Control and Optimization*, 36(5):1539–1575, September 1998.
8. R. Cheng, M. Gen, and Y. Tsujimura. A tutorial survey of job-shop scheduling problems using genetic algorithms: I. Representation. *Computers and Industrial Engineering*, 30:983–997, 1996.
9. R. Cheng, M. Gen, and Y. Tsujimura. A tutorial survey of job-shop scheduling problems using genetic algorithms: II. Hybrid genetic search strategies. *Computers and Industrial Engineering*, 36(2):343–364, 1999.
10. Xunxue Cui, Miao Li, and Tingjian Fang. Study of Population Diversity of Multiobjective Evolutionary Algorithm Based on Immune and Entropy Principles. In *Proceedings of the Congress on Evolutionary Computation 2001 (CEC'2001)*, volume 2, pages 1316–1321, Piscataway, New Jersey, May 2001. IEEE Service Center.
11. Leandro Nunes de Castro and Jon Timmis. *An Introduction to Artificial Immune Systems: A New Computational Intelligence Paradigm*. Springer-Verlag, 2002.
12. U. Dorndorf and E. Pesch. Evolution based learning in a job shop scheduling environment. *Computers & Operations Research*, 22:25–40, 1995.
13. T.A. Feo and M.G.C. Resende. Greedy Randomized Adaptive Search Procedures. *Journal of Global Optimization*, 6:109–133, 1995.
14. Emma Hart and Peter Ross. The Evolution and Analysis of a Potential Antibody Library for Use in Job-Shop Scheduling. In David Corne, Marco Dorigo, and Fred Glover, editors, *New Ideas in Optimization*, pages 185–202. McGraw-Hill, London, 1999.
15. Emma Hart, Peter Ross, and J. Nelson. Producing robust schedules via an artificial immune system. In *Proceedings of the 1998 IEEE International Conference on Evolutionary Computation (ICEC'98)*, pages 464–469, Anchorage, Alaska, 1998. IEEE Press.

16. R. Hightower, S. Forrest, and A. S. Perelson. The evolution of emergent organization in immune system gene libraries. In L. J. Eshelman, editor, *Proceedings of the 6th. International Conference on Genetic Algorithms*, pages 344–350. Morgan Kaufmann, 1995.
17. Albert Jones and Luis C. Rabelo. Survey of Job Shop Scheduling Techniques. NISTIR, National Institute of Standards and Technology, 1998.
18. E.G. Coffman Jr. *Computer and Job Shop Scheduling Theory*. John Wiley and Sons, 1976.
19. David S. Johnson Michael R. Garey. *Computers and Intractability: A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences)*. W H Freeman & Co., June 1979. ISBN 0-7167-1045-5.
20. Thomas E. Morton and David W. Pentico. *Heuristic Scheduling Systems: With Applications to Production Systems and Project Management.* Wiley Series in Engineering & Technology Management. John Wiley & Sons, 1993.
21. J. F. Muth and G. L. Thompson, editors. *Industrial Scheduling*. Prentice Hall, Englewood Cliffs, New Jersey, 1963.
22. Leandro Nunes de Castro and Jonathan Timmis. *Artificial Immnue System: A New Computational Intelligence Approach*. Springer Verlang, Great Britain, September 2002. ISBN 1-8523-594-7.
23. Leandro Nunes de Castro and Fernando José Von Zuben. Learning and Optimization Using the Clonal Selection Principle. *IEEE Transactions on Evolutionary Computation*, 6(3):239–251, 2002.
24. Mihaela Oprea. *Antibody Repertories and Pathogen Recognition: The Role of Germline Diversity and Somatic Hypermutation*. PhD thesis, University of New Mexico, Albuquerque,NM, 1999.
25. A. Perelson, R. Hightower, and S. Forrest. Evolution and Somatic Learning in V-Region Genes. *Research in Immunology*, 147:202–208, 1996.
26. M. Pinedo. *Scheduling—Theory, Algorithms, and Systems*. Prentice Hall, Englewood Cliffs, 1995.
27. Takeshi Yamada and Ryohei Nakano. Job-shop scheduling. In A.M.S. Zalzala and P.J. Fleming, editors, *Genetic Algorithms in Engineering Systems*, IEE control engineering series, chapter 7, pages 134–160. The Institution of Electrical Engineers, 1997.