A Reduced Codification for the Logical Representation of Job Shop Scheduling Problems

Juan Frausto-Solis[1] and Marco Antonio Cruz-Chavez[2]

[1] Department of Computer Science, ITESM, Campus Cuernavaca
Paseo de la Reforma 182-A, 62589, Temixco, Morelos, MÉXICO
juan.frausto@itesm.mx
[2] Faculty of Chemical Sciences and Engineering, Autonomous University of Morelos State
Av. Universidad 1001, Col. Chamilpa, 62270, Cuernavaca, Morelos, MÉXICO
mcruz@buzon.uaem.mx

Abstract. This paper presents the Job Shop Scheduling Problem (JSSP) represented as the well known Satisfiabilty Problem (SAT). Even though the representation of JSSP in SAT is not a new issue, presented here is a new codification that needs fewer clauses in the SAT formula for JSSP instances than those used in previous works. The codification proposed, which has been named the Reduced Sat Formula (RSF), uses the value of the latest starting time of each operation in a JSSP instance to evaluate RSF. The latest starting time is obtained using a procedure that finds the critical path in a graph. This work presents experimental results and analytical arguments showing that the new representation improves efficiency in finding a starting solution for JSSP instances.

Keywords: Job shop scheduling, the propositional satisfiability problem (SAT), Latest starting time, SAT formula.

1 Introduction

The Job Shop Scheduling Problem (JSSP) is one of the most relevant problems in manufacturing processes because the efficient resource management is a critical requirement. The JSSP is considered a very difficult problem, and, in computer sciences, is cataloged as an NP-hard optimization problem [1]. This indicates that there is not a deterministic algorithm to solve the problem, however, at the present time algorithms have been designed to solve certain instances of JSSP. Various approaches have been proposed for solving the JSSP using several models. Two of the most commonly used models are disjunctive graphs [2] and constraints satisfaction [3]. These models can be classified as search methods and optimization methods. The search methods can very quickly find a feasible solution to a JSSP instance, but unfortunately there are no guarantees that the solution found is the optimal one. However, search methods can provide a starting point. The methods based on satisfiability [4],

[5], and priority rules [6] are some examples of search methods. Shifting Bottleneck ([7], [8]) is a special type of search method that has a better performance that most others, but only in small instances. On the other hand, optimization methods attempt to find the best solution to a JSSP instance or at least one that is closer to a global optimum. Branch and Bound [9], Simulated Annealing [10], and Genetic Algorithms [11] are among the principal optimization methods. Even though JSSP optimization methods are outside the scope of this paper, it is important to mention that most of these methods require a starting feasible solution at the beginning of their process [12]. It is advantageous for all of these methods to find this starting solution as fast as possible. A very attractive possibility to the challenge of quickly finding a feasible solution is mapping a JSSP instance as a SAT problem [13] in such a way that the solution of the SAT problem is a feasible solution of the JSSP instance. Even though the codification of the problem of scheduling SAT is not a new issue [4], it is important to find alternative ways to codify JSSP as SAT for many reasons. First of all, because JSSP is NP hard, new SAT codification of JSSP is important in and of itself. Another reason is that for certain kinds of problems, a particular SAT codification can provide a feasible solution very quickly [5].

In this paper a SAT codification for JSSP is presented which is based on a previous one proposed by Crawford and Baker [5]. This codification is a reduced SAT formula in which the solution obtained is a feasible starting solution (a feasible schedule) of a JSSP instance, which can then be used in many JSSP optimization methods [12].

In order to get the satisfiability of the reduced logic formula one could use the well-known SAT solvers, such as GSAT [14], WalkSat [15], TABLEAU [16] and others. One could also think about using a solver in a testing plan in order to probe the efficiency of the two codifications. In this case, rather than proceed with a testing approach of the new codification, analytical arguments are presented, showing that the new Reduced Sat Formula has a smaller number of clauses, resulting in a more efficient performance than the methods which have been proposed previously.

2 Background

In order to develop the reduced codification, the following concepts were used: the JSSP, representation of the JSSP as a disjunctive graph, and SAT codification of JSSP. These concepts are explained here to give the reader background in order to be able to understand the reduced codification.

2.1 The Job Shop Scheduling Formulation

A JSSP consists of a set of jobs $J=\{j_1, j_2, \ldots, j_n\}$, a set of machines $M=\{m_1, m_2, \ldots, m_m\}$ and a set of operations $O=\{1, 2, 3, \ldots\}$. Each operation i is defined by six elements: (1) a machine m_j in which it will be processed, (2) a job j_k to which it belongs, (3) a time of processing p_i, (4) a ready time r_i, (5) a starting time s_i and (6) a deadline d_i. The ready time indicates the earliest time at which the operation can start and the deadline is the time by which the operation must be completed.

The operations of a JSSP have the following relations: For each pair *(i, j)* of operations that belong to the same job, a precedence relation exists. In addition, any single machine cannot execute simultaneously more than one operation.
These elements and relations allow the formulation of the JSSP as a constraint satisfaction problem (CSP) [3] that is known as Job Shop Deadline Scheduling. This formulation is shown in Table 1, where the subscripts *i* and *j* are associated with two distinct operations of the problem.

Table 1. The constraints of a JSSP as a constraint satisfaction problem

Constraint	Interpretation
$s_i \geq 0$	*Starting time constraint:* The starting time of the operation *i* must be non-negative.
$s_i + p_i \leq s_j$	*Precedence constraint:* The operation *i* must be complete before *j* can begin.
$s_i + p_i \leq s_j \vee s_j + p_j \leq s_i$	*Resource capacity constraint*: The operations *i* and *j* are in conflict. They require the same resource and they cannot be scheduled concurrently.
$r_i \leq s_i$	*Ready time constraint*: The operation *i* cannot begin before its ready time.
$s_i + p_i \leq d_i$	*Deadline constraint:* The operation *i* cannot finish after its deadline.

2.2 Disjunctive Graph of the JSSP

The JSSP can be represented using a disjunctive graph [2], [17]. This graph is a 3-tuple G = (N, A, E) where N is a set of nodes representing the operations of the problem. A and E are sets of arcs that symbolize precedence constraints and resource capacity constraints respectively. Precedence constraints are represented by conjunctive arcs, whereas resource constraints are represented by disjunctive arcs.

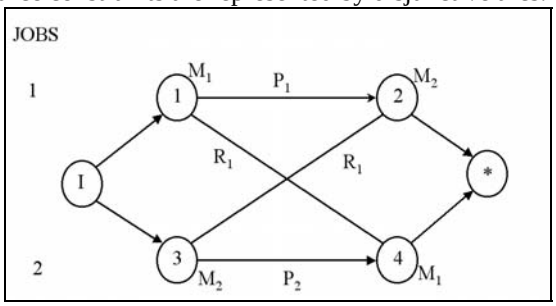

Fig. 1. Representation of a two job and two machine JSSP using a disjunctive graph

The operations connected by conjunctive arcs are those that belong to the same job, and the operations connected by disjunctive arcs are those which are executed by the same machine. Figure 1 represents a disjunctive graph where the operations 1 and 2 belong to job 1 and the operations 3 and 4 belong to job 2. In this figure, the prece-

dence constraints are represented by P1 and P2, and the resource capacity constraints are represented by R1 and R2.

2.3 SAT Codification of JSSP

The objective of the SAT problem is to confirm or deny the existence of an assignment of truth-values for the literals of a logic formula which make the formula true.

A SAT formula is usually written in its conjunctive normal form (CNF) that has the following three features: (1) a conjunction F of clauses F_i, i.e. $F = F1 \wedge F2,\ldots, \wedge Fn$, (2) each clause F_i is a disjunction of literals $X_i v \ldots . v X_k$, (3) each literal X_j is a Boolean variable–(negated or not). Crawford and Baker [5] propose a SAT codification for the JSSP based on the formulation of the JSSP as one CSP. This SAT codification is shown in Tables 2 and 3. In these tables, the subscripts i and j are associated with two distinct operations. The subscripts r_i, d_i and t are times, r_i representing the ready time and d_i representing the deadline. In this SAT codification, a JSSP instance is the set of clauses F, each CNF in Table 2 and 3 being a clause of F. In this way a JSSP instance is codified, or represented, by a SAT problem F, in which the solution is feasible for the JSSP instance.

Table 2. Logical clauses for the JSSP represented as a CSP

Constraint	CNF
$s_i + p_i \leq s_j$	$pr_{i,j}$
$s_i + p_i \leq s_j \vee s_j + p_j \leq s_i$	$pr_{i,j} \vee pr_{j,i}$
$s_i \geq r_i$	$sa_{i,ri}$
$s_i + p_i \leq d_i$	$eb_{i,di}$

Table 3. The coherence conditions for the SAT codification of a JSSP

Coherence condition	CNF	Interpretation
$sa_{i,t} \rightarrow sa_{i,t-1}$	$\sim sa_{i,t} \vee sa_{i,t-1}$	*Coherence of sa:* If i starts at or after time t, then it starts at or after the time $t-1$.
$eb_{i,t} \rightarrow eb_{i,t+1}$	$\sim eb_{i,t} \vee eb_{i,t+1}$	Coherence of *eb*: If i ends by time t, then it ends by time $t+1$.
$sa_{i,t} \rightarrow \sim eb_{i,t+pi-1}$	$\sim sa_{i,t} \vee \sim eb_{i,t+pi-1}$	*Coherence of* p_i: If i starts at or after time t, then it cannot end before time $t+p_i$.
$sa_{i,t} \wedge pr_{i,j} \rightarrow sa_{j,t+pi}$	$\sim sa_{i,t} \vee \sim pr_{i,j} \vee sa_{j,t+pi}$	*Coherence of* $pr_{i,j}$: If i starts at or after time t, and j follows i, then j cannot start until i is finished.

3 Reduced SAT Codification

The codification presented in the last section is complete because it represents all the constraints of the Job Shop Deadline Scheduling Problem with logical formulas; however for this codification, one should build and then evaluate many clauses. It would be advantageous to have a reduced codification of any JSSP instance, where the num-

ber of clauses is smaller, so in principle, the efficiency to build and evaluate the respective SAT formula is increased. The reduced codification of the JSSP is based on two concepts, the reduction of clauses and the determination of the latest starting time (LST). These two fundamental concepts are explained in the following sections.

3.1 Reduction of Clauses

It is possible to significantly reduce the number of clauses that compose the complete SAT codification, according to the following analysis.

First, all of the clauses in Table 2 can be eliminated for any JSSP instance because the respective clauses will automatically be true. They are true because any schedule proposed has a known sequence in which every pair of operations has a known order**.** In a schedule, the LST (latest starting time) of an operation is the latest time in which the operation can start. When the LST is calculated, the ready time is assigned to this time (r_i = LST) and the deadline can be determined ($d_i = r_i + p_i$). Table 4 describes the form in which the constraints can be eliminated. Given this treatment, all the clauses in Table 2 are true which means they can be eliminated.

Table 4. Conditions for eliminating the scheduling constraints in a reduced codification

Constraints which can be eliminated	**Rationale**
Precedence constraints $(pr_{i,j})$	For any JSSP, the precedence constraints are considered to define the problem and the equivalent clause $pr_{i,j}$ is always true. These clauses can be eliminated from the codification because their truth-values are always true.
Resource capacity constraints $(pr_{i,j} \vee pr_{j,i})$	Because the resource capacity constraints specify the use of the same machine by two operations, and because the schedule for the problem is defined, the resource capacity constraints can be exchanged for the precedence constraints in the logical representation. When this is done, it is possible to see that these clauses can be eliminated from the codification because their truth-values are always true.
Ready time constraints $(sa_{i,ri})$ and deadline constraints $(eb_{i,di})$	If the LST of each operation in the defined schedule is found, the ready times and the deadlines of each operation can be determined. If these times are known, the clauses in the codification that contain $sa_{i,ri}$ or $eb_{i,di}$ will be always true, so they can be eliminated from the codification.

For the clauses that represent the coherence conditions (Table 3), if t is the LST of the operation ($t=r_i$), then the literals $sa_{i,t-1}$, $\sim eb_{i,t}$ and $\sim eb_{i,t+pi-1}$ are all true. The clauses with these literals can be eliminated from the codification and the clauses that remain are the clauses that are used to evaluate the coherence of $pr_{i,j}$. Table 5 presents the justification for realizing this elimination process.

The key of this reduction is the determination of the LST for each operation in the defined schedule.

It should be noted that RSF is formed only by the coherences pr_{ij} presented in Table 3. A SAT solver like GSAT [14], WalkSat [15], TABLEAU [16] and many others can be used in order to find a solution to the SAT problem of RSF which represents a

feasible schedule. These solvers assign truth-values to the variables pr_{ij} in the RSF. Next, the LST (latest starting time) for each operation (i, j) is calculated. With the LST and truth-values of the variables involved in RSF obtained, the solver begins to prove the satisfiability of RSF.

Because all the eliminated clauses are always true, it is not useful to maintain them in the complete formula when a SAT solver is used. While the RSF presented here has fewer evaluations than the complete SAT codification, it requires that the LST be found. One can observe that the complete SAT codification requires the calculation of the t times using a procedure not described by Crawford and Baker [5].

Table 5. Justification for eliminating the coherence conditions in a reduced codification

Constraints which can be eliminated	Rationale
Coherence of *sa*	Because t is the LST, then $t=r_i$. For clauses with the form $\sim sa_{i,t} \vee sa_{i,t-1}$, the literal $\sim sa_{i,t}$ is false, and $sa_{i,ri}$ is true. The literal $sa_{i,t-1}$ is true because the operation i starts at r_i, which is equal to t and s_i is after r_{i-1}. Therefore, the clauses are always true and they can be eliminated.
Coherence of *eb*	Because t is the LST, then $t=r_i$. For clauses with the form $\sim eb_{i,t} \vee eb_{i,t+1}$, the literal $\sim eb_{i,t}$ is true and $eb_{i,ri}$ is false. Because the operation i cannot finish at its ready time or before, the literal $eb_{i,t+1}$ is false and $eb_{i,ri+1}$ is also false. Therefore the clauses are always true and they can be eliminated.
Coherence of p_i	Because t is the LST, $t=r_i$. For clauses in the form $\sim sa_{i,t} \vee \sim eb_{i,t+pi-1}$, the literal $\sim sa_{i,t}$ is false and $sa_{i,ri}$ is true. The literal $\sim eb_{i,t+pi-1}$ is true and $eb_{i,ri+pi-1}$ is false because the operation i cannot finish before of r_i+p_i. Therefore the clauses are always true and they can be eliminated.

3.2 Latest Starting Time

The LST of a given operation is equal to the critical (longest) path generated in a digraph between the given operation and the initial operation (I in Figure 1). The value of the critical path of an operation i is the sum of the processing times of all the operations in the sequence between I and i. The LST is used as the start time s_i of the operation i.

Figure 2 shows one example of two operations (i and j) processed by the same machine. In this figure, the ready time of the operation j (r_j) and the deadline of the operation i (d_i) are shown when the resource capacity constraint is transformed into one precedence constraint. These times are valid if the resource capacity constraint between i and j is not violated.

In this example, the LST of the operations i and j can be calculated (t_i and t_j, respectively), so the finish time of the operation i is $t_i + p_i$. If t_j is greater than or equal to $t_i + p_i$, the clause $\sim sa_{i,t} \vee \sim pr_{i,j} \vee sa_{j,t+pi}$ is true and the ready times will always be valid for each operation. If t_j and $t_i + p_i$ are equal to each other, r_j represents the finish time of the operation i ($r_j = t_i + p_i$) and is valid because the operation j will start at the time t_j (the operation j starts when the operation i finishes). The deadlines for these

operations are valid, as is shown for operation *i* in Figure 2. In this example, the operation could slow its finish (d_i) until the beginning of the operation *j* (t_j). In this case, (e.g., $d_i < r_j$), there would be idle time, a few moments when the machine would not be working, between the end of one operation and the beginning of another.

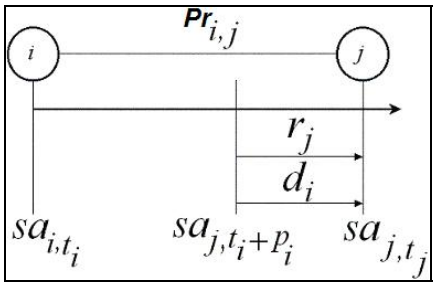

Fig. 2. Graphic representation of the resource capacity constraint between two operations (with their LST's) that have a defined sequence

3.3 Obtaining the Latest Starting Time

The general problem in obtaining the longest path in a graph is classified as NP-complete [18]. A certain relaxation is required to find the path in an efficient form. In this work, a method based on the approach proposed by Adams, et al. in [7] is used. In this method, Hamilton routes from each of the machines are taken for a possible schedule from the graph that represents the JSSP. This simplification of the directed graph generates a binary search tree for the schedule. The problem becomes finding the longest path between two nodes of the graph which can be solved in polynomial time [7], [18], with a complexity of *O(N)*, where *N* is the number of nodes generated in the binary search tree. In the search tree, the root of the tree is the operation that is needed to obtain the LST, and contains each possible route to arrive at the initial node (operation I). Each node of the resulting tree has a maximum of one successor and one predecessor.

This search tree is used to determine the LST of all the operations of the possible schedule. The LST of each operation is the critical path from this operation to the initial node. In this way, the determination of the LST is reduced to the determination of the critical path of each operation.

3.4 A Method of Generating the Reduced SAT Formula for Any JSSP

Figure 3 shows the algorithm that produces a reduced codification for the SAT representation of any JSSP. It is possible to check the satisfiability of the reduced codification obtained using SAT solvers.

In this algorithm, for each disjunctive arc (the resource capacity constraint), two clauses exist. One clause exists when operation *i* precedes *j*, and the other clause exists when the operation *j* precedes *i*. The generation of the RSF, is a function of the

number of disjunctive arcs on the JSSP graph (see Fig 1). The number of disjunctive arcs are defined as *arc = m(n-1),* where *m* is the number of machines and *n* is the number of jobs. For the purpose of comparing the complexity of the construction of RSF with the construction of the Crawford and Baker formula, the same number of jobs and machines are used so *m=n*. With this simplification, the RSF generates:

```
procedure reduced_codification (N,E,P)
{ N is a set of operations }
{ E is a set of disjunctive arcs without a sequence of use assigned}
{ P is a set of processing times of each operation }
{ The data represents one schedule as defined (feasible or not) }
{ C is the set of clauses in CNF, at the beginning C is empty }
begin
   for k=1 to number of arcs in E do
   begin
      { pr_{i,j} are arcs in E }
      C = C ∧ (~sa_{i,ri} ∨ ~pr_{i,j} ∨ sa_{j,di});
   end
   C is the reduced codification;
   return C;
end.
```

Fig. 3. The algorithm for constructing a reduced codification for the SAT representation of a JSSP

$$Clauses = 2n^2-2n \quad (1)$$

In addition, as each clause of the reduced SAT formula contains 3 literals, the evaluated literals are:

$$Literals=6n^2-6n \quad (2)$$

After taking into account the information in (1) and (2), the complexity in order to generate the RSF is $O(n^2)$.

In the case of the complete codification, when $m = n$, in addition to generating the RSF clauses, it is necessary to evaluate each $pr_{i,j}$ clause. For each pair of operations belonging to the same job, $n(n-1)$ clauses is needed. For each operation, it is also required to evaluate the following 5 types of clauses: $sa_{i,ri}$ (with one literal) that is equivalent to n^2 clauses for all operations, $eb_{i,di}$ (with one literal) that is equivalent to n^2 clauses for all operations, and three types of clauses $\sim sa_{i,t} \vee sa_{i,t-1}$, $\sim eb_{i,t} \vee eb_{i,t+1}$, $\sim sa_{i,t} \vee \sim eb_{i,t+pi-1}$ (each one with two literals) that are equivalent to $3n^2$ clauses for all operations. If all the clauses that need to be evaluated are added together, the Crawford and Baker formula generates a minimum of:

$$Clauses = 8n^2-3n \quad (3)$$

The literals that they will evaluate are:

$$Literals=15n^2-7n \quad (4)$$

It can be seen from (3) and (4) that the complexity in order to generate the Crawford and Baker formula is $O(n^2)$. Although the complexity of generating the SAT formula in the two methods is the same, it is clear that evaluating RSF is simpler. The simplification is demonstrated in 2 and 4 where it can be seen that when using RSF it is necessary to evaluate the truth-value of a smaller number of literals.

4 Experimental Results

Several tests were performed in order to verify the reduction approach presented in this paper. In Table 6, the comparison of the number of clauses produced is shown. Two methods are examined, that of Crawford and Baker, and that of the reduced codification. The problems were taken from Beasley [19]: the problem, FT6, has 6 jobs and 6 machines, the problem, FT10, has 10 jobs and 10 machines, etc. The comparison of these results is shown in Table 6. The reduced codification (RSF) reduced an average of 75.9% of clauses for the nine problems.

Table 6. Number of clauses in a JSSP

Problem	Number of clauses		% reduction
	complete SAT codification	RSF	
FT6	270	60	77.78
FT10	770	180	76.62
LA21	3,465	840	75.76
LA24	4,536	1,106	75.66
LA25	4,925	1,200	75.63
LA27	5,751	1,404	75.59
LA29	6,641	1,624	75.55
LA38	11,438	2,812	75.42
LA40	12.680	3,120	75.39

5 Conclusions

RSF produces a significant reduction in the number of clauses (75.9%). The key to this reduction approach is the determination of the *t* times as the latest starting times (with a complexity of $O(N)$) of each operation for a proposed schedule. The complete SAT codification of Crawford and Baker, requires an additional manipulation of the *t* times using the unit propagation as a polynomial time procedure. The complexity for this procedure could not be less than a complexity of a linear order. Although the complexity for the generation of the SAT formula in the two methods it is the same, it is clear that the evaluation of RSF is simpler because RSF requires the evaluation of the truth-values of a smaller number of literals. Due to the fact that RSF simplifies the number of clauses by such a significant percentage and the efficiency of linear complexity for obtaining the times *t*, it can be observed that it is more advantageous to use RSF than the complete SAT codification.

The algorithm that generates the RSF is applied only once in the beginning of the process of using a solver. RSF can be used in optimization methods that need initial solutions.

References

1 Garey, M. R., Johnson, D. S. and Sethi, R.: The Complexity of Flow Shop and Job Shop Scheduling, in Mathematics of Operation Research, Vol. 1, No. 2 (1976) 117-129

2 Conway, R.W., Maxwell, W.L. and Miller, L.W.: Theory of Scheduling. Addison-Wesley, Reading, Massachusetts (1967)

3 Smith, S.F. and Cheng, C.C.: Slack-Based Heuristics for Constraint Satisfaction Scheduling, in Proc. of the 11th National Conf. on Artificial Intelligence, Washington, D.C., (1993) 139-145

4 Ullman, J.D.: NP-complete scheduling problems, in Journal of Computer System Sciences, Vol. 10 (1975) 384-393

5 Crawford, J.M. and Baker, A.B.: Experimental Results on the Application of Satisfiability Algorithms to Scheduling Problems, in Proc. of the 12th National Conf. on Artificial Intelligence, Austin, TX, (1994) 1092-1098

6 Panwalker, S.S. and Iskander, W.: A survey of scheduling rules, in Operations Research, Vol. 25, No. 1, (1977) 45-61

7 Adams, E., Balas, E. and Zawack, D.: The Shifting Bottleneck Procedure for Job Shop Scheduling, in Management Science, Vol. 34, No. 3 (1988) 391-401

8 Schutten, M.J.: Practical job shop scheduling, in Annals of Operations Research, Vol. 83, (1988) 161-177

9 Carlier, J. and Pinson, E.: An algorithm for solving the job-shop problem, in Management Sciences, Vol. 35, No. 2 (1989) 164-176

10 Yamada, T. and Nakano, R.: Job-Shop Scheduling by Simulated Annealing Combined with Deterministic Local Search, in Metaheuristics Int. Conference, Colorado, USA, (1995) 344-349

11 Zalzala, P.J. and Flemming: Genetic algorithms in engineering systems, in A.M.S. Inst. of Electrical Engineers (1997)

12 Albert Jones and Luis C. Rabelo: "Survey of Job Shop Scheduling Techniques," NISTIR, National Institute of Standards and Technology, Gaithersburg, MD, 1998. In http://www.mel.nist.gov/msidlibrary/summary/authlist.htm.

13 Papadimitriou, C.H., Computational Complexity, Addison Wesley Pub. Co., USA, ISBN 0-201-53082-1, (1994).

14 Selman, B., Levesque, H. and Mitchell, D. A new method for solving hard satisfiability problems, In Proceeding of the Tenth National Conference on Artificial Intelligence, 139-144, 1992

15 Selman, B, and Kautz, II,: A. Local search strategies for satisfiability testing, In Procceding DIMACS Workshop on Maximum Clique, Graph Coloring and Satisfiability. 1993.

16 Davis, M., Logeman, G., and Loveland, D.: A machine Program for theorem proving. In CACM, (1962) 394-397.

17 Balas, E: Machine Sequencing via Disjunctive Graphs: An Implicit Enumeration Algorithm, in Operations Research, Vol. 17 (1969) 941-957

18 Garey, M.R. and Johnson, D.S.: Computers and Intractability: A Guide of the Theory of NP-Completeness, W.H. Freeman and Co, New York (1979)

19 Beasley, J.E.: OR Library, Imperial College, Management School, http://mscmga.ms.ic.ac.uk/info.html (1990)