A Very Efficient Algorithm to Representing Job Shop Scheduling as Satisfiability Problem

Juan Frausto-Solis[1], Felix Martinez-Rios[2] and Marco Antonio Cruz-Chavez[3]

[1] Department of Computer Science, ITESM, Campus Cuernavaca
Autopista del Sol km 104, Colonia Real del Puente, 62790,
Xochitepec, Morelos, Mexico
juan.frausto@itesm.mx
[2]Engineering School, Universidad Panamericana, Augusto Rodin 498,
Insurgentes Mixcoac, 03920, Mexico DF, Mexico
fmartin@up.edu.mx
[3] Faculty of Chemical Sciences and Engineering,
Autonomous University of Morelos State,
Av. Universidad 1001, Col. Chamilpa, 62270, Cuernavaca,
Morelos, Mexico
mcruz@buzon.uaem.mx

Abstract. The representation of Job Shop Scheduling Problem (JSSP) as the well known Satisfiabilty Problem (SAT) is useful because it helps to find feasible schedules in an efficient way. One of the most efficient SAT codifications of JSSP is RSF (Reduced Sat Codification) which transforms any JSSP instance to a 3-SAT problem. In this paper RSF is improved by codifying JSSP as 1-SAT problem by applying a heuristic tautology elimination and a random algorithm RandTaut. Starting with a random JSSP solution, RandTaut finds its 1-SAT representation and evaluates whether it is feasible or not. Experimental results presented in the paper confirm that the new codification is more efficient than the previous ones.

Keywords: JSSP, Satisfiability Problem, 1-SAT

1 Introduction

Nowadays, SAT researchers have given a great importance to practical problems, most of them are in scheduling area; for instance flights allocation in runways [1], scheduling in automobile factories [2] and Job Shop Scheduling Problem (JSSP) [3]. Among them, JSSP has many important industrial applications in industries as automotive, Aerospatiale, and so on. Crawford & Baker's codification for satisfiability (SAT) problem [4] is very advantageous because through it, feasible solutions can be easily obtained thanks to many SAT solvers available [5]. One of the most efficient SAT codifications of JSSP is RSF (Reduced Sat Codification) has been published previously which leads to a reduce number of clauses in its logic representation [6]. RSF leads any JSSP instance to a 3-SAT codification, known as 3-

M.A Cruz-Chávez, J.C. Zavala-Díaz (Eds): CICos 2008, ISBN:978-607-00-0165-9. pp. 52-62, 2008

SAT problem; that means 3 literals (each one representing a propositional formula) per clause. The format used to represent a SAT formula is the well known CNF formula [6]. Notice that 3-SAT is a NP Complete problem [7] while 1-SAT is a P problem; besides RSF leads JSSP to 3-SAT while Crawford and Baker leads a JSSP to a k-SAT (k greater or equal to 3). In this paper RSF is improved by codifying JSSP as 1-SAT problem by applying a heuristic tautology elimination and a random algorithm named RandTaut (it comes from "RANdom algorithm with TAUTologies elimination) is proposed. In order solve efficiently JSSP it is advisable to represent it with a short SAT formula; therefore, RandTaut searches and then eliminates tautology formula from the SAT representation of a JSSP problem.

2 Backgrounds

2.1 Job shop scheduling problem

JSSP is very complicated because every machine can execute many operations and different operations can be executed by different machines at different times. In JSSP we have a set of machines $M=\{M_1,M_2,..,M_m\}$ where a set of tasks $j=\{1,2,...,n\}$ should be scheduled. Each task requires a sequence of operation $O_{1,j}, O_{2,j},...,O_{n,j}$ (n,j is the number of operation made by the task j) [8]. The $O_{i,j}$ operation cannot be initiated before the $O_{i-1,j}$ operation has finished. The goal is to find the schedule that minimizes the Makespan that means the time to complete the last operation in the system [3]. JSSP is one of the hardest combinatorial optimization problems; as an example, JSSP instances with only 10 operations and 10 machines have lasted around 25 years to find an optimal solution [3].

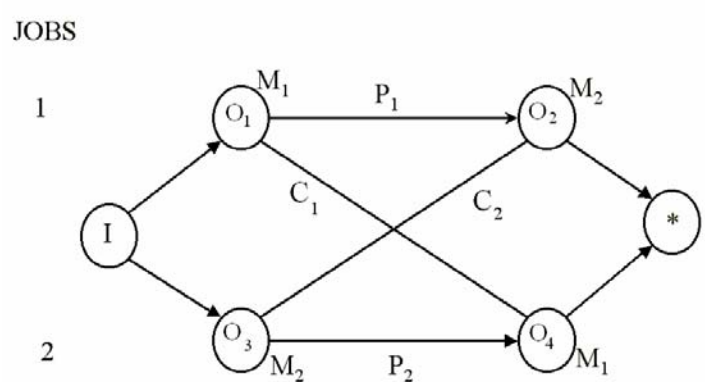

Fig. 1. Disjunctive Graph of the JSSP.

A common JSSP graphical representation is known as disjunctive graph [9, 10]. In Figure 1 a disjunctive graph $G = (N, A, E)$ shows a set of four nodes (N) symbolizing the operations O_1, O_2, O_3 and O_4. There are two jobs; job 1 has O_1 and O_2 while job 2

has O_3 and O_4 operations. Six conjunctive arcs *(A)* define precedence restrictions and two disjunctive arcs *(E)* represent resources capacity restrictions. There are two machines M_1 y $\underline{M_2}$ to realize the operations. The problem of Figure 1 has to satisfy two kinds of operation constraints: Precedence constraints (P_1, P_2) and capacity constraints (C_1, C_2). For instance P_1 specify that O_1 precedes O_2 $(O_1<O_2)$, while C_1 indicates that the precedence between O_1 and O_4 in M_1 machine can be is unknown; therefore constraint C_1 can be written as $(O_1<O_4)\vee(O_4<O_1)$.

JSSP has other important features. The ready (r_i) and deadline times (d_i) of each operation; r_i is the moment when an operation is available (calculated through its critical route). Each operation starts at s_i (so $s_i \geq r_i$) lasting p_i units of time (so $s_i+p_i \leq d_i$). Then JSSP is defined with the constraints shown in Table 1 [4].

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.

The Job shop that has been used to exemplify a codification in SAT form are simple and recirculation is not allowed, that can happen when tasks can visit one machine more than one time simplifying the problem because each work takes place in a single machine.

2.3 JSSP codified as 3-SAT

In order to establish JSSP as a Constraint Satisfaction Problem and codified as a SAT problem, Crawford and Baker introduce the clauses shown in Table 2 [4]. They are evaluated like true when all their respective constraints are fulfilled. The first precedence constraint indicates an execution order for *(i, j)* tasks pair, and express that *i* must have finished before *j* can be started. The second constraint is a capacity resource constraint (2) it establishes that there are not enough resources (machines) for the operations *(i, j)* involved; this constraint also establishes that for a pair of operations *(i, j)* it should be decided which of them is executed first. The third constraint means that for an operation *i* there is ready time (r_i) indicating when this operation can be started. Finally, the fourth constraint defines the duration of each operation in function of its deadline.

Crawford and Baker define the coherence conditions shown in Table 3 [4]. As it is explained in [6], if t is the Latest Start Time (LST) of every operation $(t=r_i)$, then the literals $sa_{i,t-1}$, $\sim eb_{i,t}$ and $\sim eb_{i,t+pi-1}$ are all trivially true. Then the clauses with these literals can be eliminated and then the last clause, $(\neg sa_{i,t} \vee \neg pr_{i,j} \vee sa_{j,t+p_i})$ is used by RSF codification as the only clause needed to define a feasibility JSSP solution [6]. In this form a feasible solution is obtained by a 3-SAT representation in RSF.

Table 2. JSSP restrictions for SAT codification.

Job shop scheduling restrictions		SAT codification	SAT meaning	Restriction/ Clause number
Precedence	$s_i + p_i \leq s_j$	$pr_{i,j} = T$	i operation precedes j operation	(1)
Resources capacity	$(s_i + p_i \leq s_j) \vee (s_j + p_j \leq s_i)$	$pr_{i,j} \vee pr_{j,i} = T$	i operation precedes j operation or j operation precedes to i	(2)
Operation beginning time	$s_i \geq r_i$	$sa_{i,r_i} = T$	i operation initiates at r_i time or later	(3)
Operation maximum duration time	$s_i + p_i \leq d_i$	$eb_{i,d_i} = T$	i operation maximum finishes at d_i time.	(4)

3. RandTout algorithm and 1-SAT representation

Figure 2 shows a RandTout Algorithm to obtain a SAT codification using RSF. As data for the JSSP codification we have: number of job, number of machines, processing time for each operation and precedence restrictions. First a set of random sequences of capacity constraints is evaluated. Then the LST is determined for every operation. Using RSF a set of clauses for each capacity constraint is obtained.

This SAT representation is written as 1-SAT problem by a tautology elimination. In order to do that notice that the only clause related to each operation is $\neg sa_{i,t} \vee \neg pr_{i,j} \vee sa_{j,t+p_i}$. The first proposition $sa_{i,t}$ means that operation i should start at time t, but this is always true for the LST previously calculated; therefore its negation is always false and can be eliminated. The second proposition $pr_{i,j}$ is trivially true for the sequence proposed and its negation is also false.

Table 3. Coherence transformation to propositional SAT.

Coherent Conditions for JSSP in all its relevant time intervals [r_i to d_i].	Meaning
$\neg sa_{i,t} \vee sa_{i,t-1}$	It does not initiate *i* operation at *t* time or after it or initiates at time *t-1*.
$\neg eb_{i,t} \vee eb_{i,t+1}$	The *i* operation does not finish at *t* time or finishes at time *t+1*.
$\neg sa_{i,t} \vee \neg eb_{i,t+p_{i-1}}$	The *i* operation does not initiate at *t* time or after it, or it does not finish before time $t + p_i$.
$\neg sa_{i,t} \vee \neg pr_{i,j} \vee sa_{j,t+p_i}$	The *i* operation does not initiate at *t* timer or after, or it does not precede to *j* or *j* initiates at time $t + p_i$ or after.

Therefore the only required proposition to determine if the clause is true or not is the last one. Proposition $sa_{j,t+p_i}$ indicates that the operation *j* can't never start after its LST of the operation *i* plus its processing time. The new codification uses the conjunction of all the $sa_{j,t+p_i}$ constraints for each resource capacity arc and this is a conjunctive normal form (CNF).

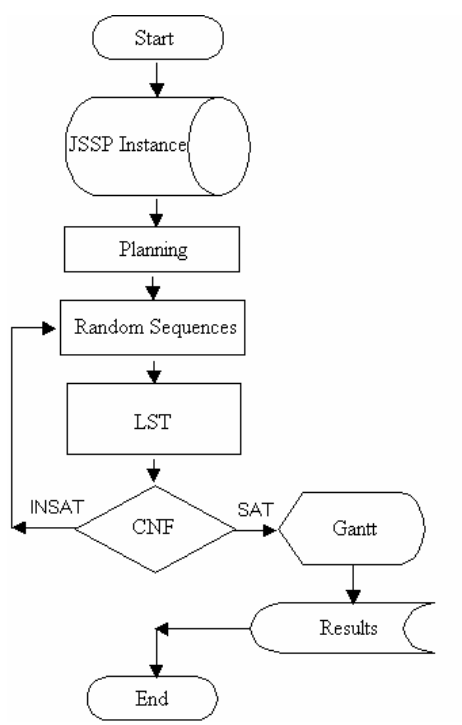

Fig. 2. RandTaut algorithm.

Once JSSP is represented with a CNF formula (named CNF codification), it is evaluated as true or false. If the CNF is false (INSAT branch in figure 2) other operation sequence is proposed, otherwise if a true solution is obtained the Makespan is determined and the Gantt graph is built. The optimal solution is obtained by using a simple Boltzman distribution (i.e. a Simulated Annealing Algorithm) [11].

4 Example of SAT codification for JSSP

This example consists of 2 machines *(M_1, M_2)* and 2 jobs each with 2 operations. The precedence restrictions and resources capacity are shown in Figure 3. The operation conditions are shown in Table 4.

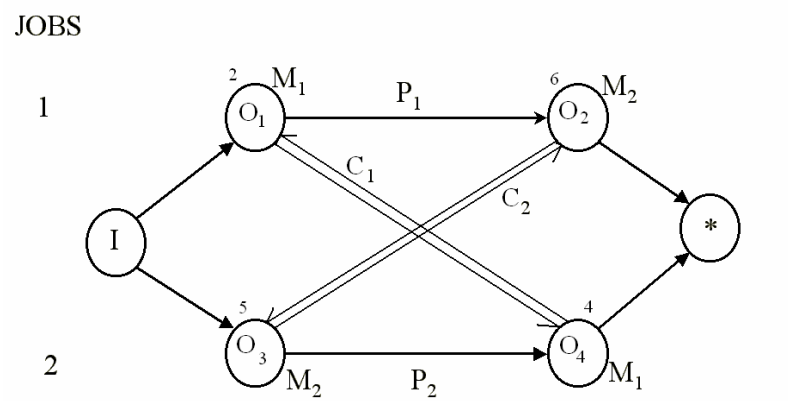

Fig. 3. JSSP with constraints of precedence P_1, P_2 and resource capacity C_1, C_2.

Table 4. Operation conditions for JSSP in Figure 3.

Operation	Processing time p_i
1	2
2	6
3	5
4	4

For this data a SAT codification for two proposed sequences that leads to a false or a true solution are presented in the next subsections.

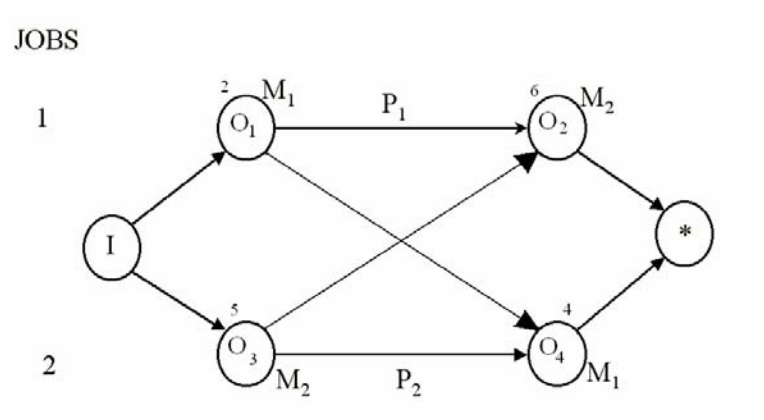

Fig. 4. JSSP instance with precedence restrictions.

4.1 Case 1: Sequence proposed leads to a true solution

The sequence proposed is shown in Figure 4 and Table 5. The ready times for this data are obtained in Table 6. Table 5 shows each operation to be executed and the respective precedence that must be completed.

Table 5. Activity Precedence for Figure 4.

Activity	Precedence
O_1	I
O_2	O_1, O_3
O_3	I
O_4	O_1, O_3
*	O_2, O_4

Table 6. Processing and ready times for Figure 4.

Operation	Processing Time p_i	Ready Time s_i
1	2	0
2	6	5
3	5	0
4	4	5

The SAT evaluation using Tables 5 and 6 is as follows:

$$\left(\neg sa_{3,t} \vee \neg pr_{3,2} \vee sa_{2,t+p_3}\right) = \left(\neg sa_{3,0} \vee \neg pr_{3,2} \vee sa_{2,0+5}\right) = F \vee F \vee T = T$$

$$\left(\neg sa_{1,t} \vee \neg pr_{1,4} \vee sa_{4,t+p_1}\right) = \left(\neg sa_{1,0} \vee \neg pr_{1,4} \vee sa_{4,0+2}\right) = F \vee F \vee T = T$$

LST of the operation 3 is zero (because this operation does not really have a precedence operation), then this operation can be started at time zero as is indicated by its ready time in Table 6; therefore $\neg sa_{3,t}$ is false. The second proposition $\neg pr_{3,2}$ is false because the sequence proposed defines that operation 3 precedes operation 2. The last proposition $sa_{2,t+p_3} = sa_{2,0+5}$ is true because the ready time of operation 2 (equal to 5) is greater or equal than the LST of operation 3 plus its processing time.

4.2 Case 2: Sequence proposed leads to a false solution

The sequence proposed is shown in Figure 5 and Table 7. The ready times for this data are shown in Table 8.

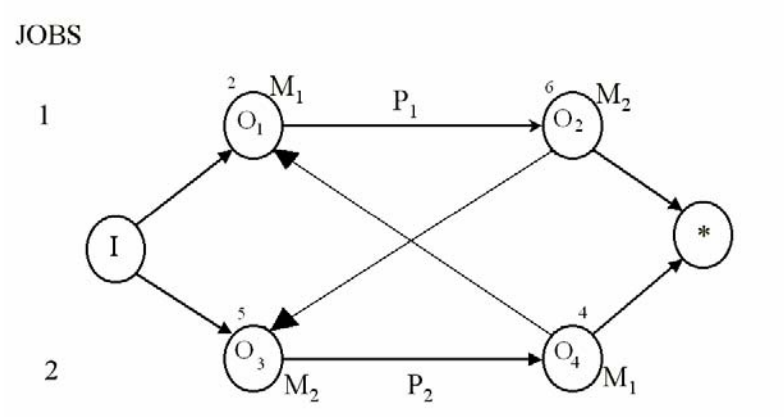

Fig. 5. JSSP instance with precedence restrictions.

Table 7 shows each operation to be executed and the respective precedence that must be completed.

Table 7. Activity Precedence for Figure 5.

Activity	Precedence
O_1	I, O_4
O_2	O_1
O_3	I, O_2
O_4	O_3
*	O_2, O_4

Table 8. Processing and ready times for Figure 5.

Operation	Processing Time p_i	Ready Time s_i
1	2	9
2	6	11
3	5	8
4	4	13

The SAT evaluation using Tables 7 and 8:

$$\left(\neg sa_{2,t} \vee \neg pr_{2,3} \vee sa_{3,t+p_2}\right) = \left(\neg sa_{2,11} \vee \neg pr_{2,3} \vee sa_{3,11+6}\right) = F \vee F \vee F = F$$
$$\left(\neg sa_{4,t} \vee \neg pr_{4,1} \vee sa_{1,t+p_4}\right) = \left(\neg sa_{4,13} \vee \neg pr_{4,1} \vee sa_{1,13+4}\right) = F \vee F \vee F = F$$

The first two propositions in both clauses are determined as explained previously. As we can observe, the third proposition in first clause is false because the ready time of operation 3 is smaller than LST of operation 2 plus its processing time. In the same way the third proposition in the other clause is also evaluated as false.

5 Experimental Results

Several tests were performed in order to verify the reduction approach presented in this paper. In Table 9, the comparison of the number of propositional formulas produced is shown. Two methods are examined, one of Crawford and Baker [4], and the other of the Reduced Sat Codification (RSF) [6]. The instances of JSSP were obtained from Beasley [12]: FT6 instance has 6 jobs and 6 machines; FT10 instance has 10 jobs and 10 machines, etc. The comparison of these results is shown in Figure 6. RandTaut algorithm finds a logical representation reduced 71.8% on the average of propositional formulas for all the nine instances. RSF published 75% of reduction [6] Notice that RSF and RandTaut has less number of propositional formulas than Crawford and Baker codification, but RandTaut always obtains 1-SAT instances that can be evaluated in polynomial time, therefore it is much efficient than the previous SAT codifications.

6 Conclusions

In this paper a very efficient algorithm RandTaut to represent JSSP in the well known SAT problem is presented. Very simple examples illustrate the algorithm. RandTaut uses a previous transformation known as RSF to transform any JSSP instance in a SAT problem. Even though RSF is a very efficient representation, each JSSP instance is transformed in a 3-SAT problem while RandTaut transforms the same instance to a 1-SAT problem. RandTaut uses a heuristic of tautology elimination in order to make

more compact the logical representation. RandTaut produces SAT formulas of JSSP instances with a significant reduction in the number of propositional formulas; the average reduction is 71.8% that is a much bigger reduction. This reduction approach can be used in classical optimization methods as Simulated Annealing to find the optimal. Because a 1-SAT representation of any job shop instance can be obtained with the proposed algorithm and it is known that 1-SAT belongs to the P class, it is trivially inferred that there are many polynomial algorithms to find feasible solutions for JSSP.

Table 9. Number of propositional formulas in a JSSP instance.

JSSP Instance	Number of propositional formulas				
	Crawford and Baker	RSF	% of reduction with RSF	RandTaut	% of reduction with RandTaut
FT6	270	60	77.8	90	66.7
FT10	770	180	76.6	450	41.6
LA21	3465	840	75.8	1050	69.7
LA24	4536	1106	75.6	1050	76.9
LA25	4925	1200	75.6	1050	78.7
LA27	5751	1404	75.6	1900	67.0
LA29	6641	1624	75.5	1900	71.4
LA38	11438	2812	75.4	1575	86.2
LA40	12680	3120	75.4	1575	87.6

References

1. Jonson, G.: Separating the Insolvable and Merely Difficult. New York Times, New York, UMI Publication , No. 05618128, Jul 13, (1999)
2. Brian, H.: Can't get no satisfaction. American Scientist, Research Triangle Park, 85, (1997), pp. 108-112
3. Schutten, M.J.: Practical job shop scheduling. Annals of Operations Research 83, (1988), pp. 161-177
4. Crawford, J.M., Baker, A.B.: Experimental Results on the Application of Satisfiability Algorithms to Scheduling Problems. Computational Intelligence Research Laboratory, (1994)

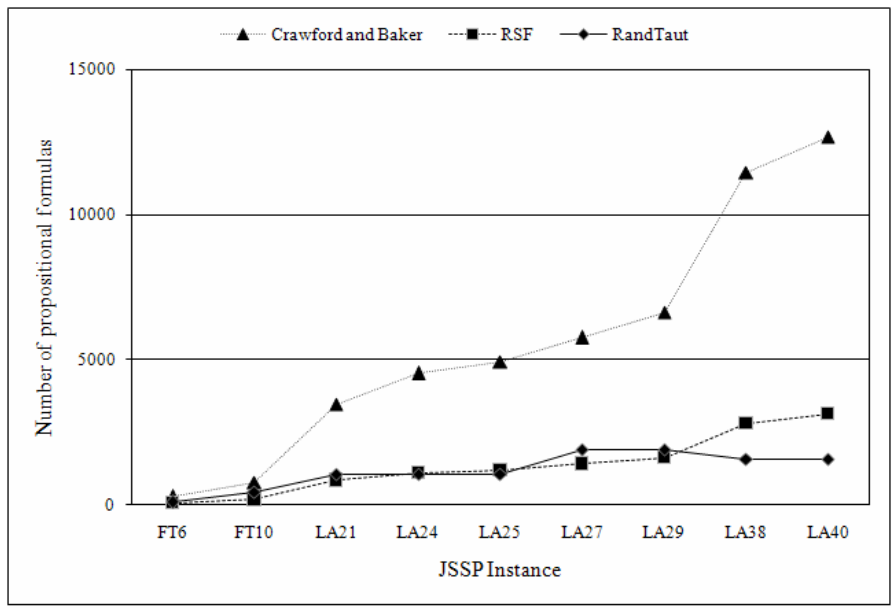

Fig. 6. The comparison between Crawford and Baker, RSF and RandTaut codification.

5. SATLIB: The Satisfiability Library, http://www.cs.ubc.ca/~hoos/SATLIB/index-ubc.html
6. Frausto-Solis, J., Cruz-Chavez, M.A.: A Reduced Codification for the Logical Representation of Job Shop Scheduling Problems: Lecture Notes in Computer Science, Springer-Verlag, ISSN: 0302-9743, Vol. 3046, pp. 553-562 , May 14-17, (2004)
7. Papadimitriou, C.H.: Computational Complexity. Addison Wesley Pub. Co., USA, ISBN 0-201-53082-1, (1994)
8. Adams, J., Balas, E., Zawack, D.: The Shifting Bottleneck Procedure for job shop scheduling. Management Science Vol. 34, No 3, (1988)
9. Conway, R.W., Maxwell, W.L. and Miller, L.W.: Theory of Scheduling. Addison-Wesley, Reading, Massachusett, (1967)
10. Balas, E: Machine Sequencing via Disjunctive Graphs: An Implicit Enumeration Algorithm, in Operations Research, Vol. 17, (1969), pp. 941-957
11. Golden Ratio Annealing for Satisfiability Problems using Dynamically Cooling Schemes. 17th International Symposium on Methodologies for Intelligent Systems (ISMIS'08), Toronto, Canada, May 20, (2008), Accepted as a full paper for publication in ISMIS 2008 proceedings will appear in Springer's Lecture Notes in Artificial Intelligence (LNAI)
12. Beasley, J.E.: OR Library, Imperial College, Management School, http://mscmga.ms.ic.ac.uk/info.html