International Journal of Computer and Information Technology (ISSN: 2279 – 0764)
Volume 05 – Issue 01, January 2016

Job shop Scheduling under Answer Set Programming

Omar EL-Khatib
Computer Science Department
Taif University
Taif, SA
Email: omer.khatib [AT] tu.edu.sa

***Abstract*—Answer set programming (ASP) is a new programming language paradigm combining the declarative aspect with non-monotonic reasoning. In this paper, we will show an application of job shop scheduling in Answer Set Programming. The problem is a highly combinatorial and is generally solved by specific programs written in procedural languages. We present an approach to solve the job shop scheduling problem in Answer Set Programming and compare them with the standard ones. It turns out that, although Answer Set Programming greatly simplifies the problem statement. It is comparable in efficiency to specialized programs. (Abstract)**

Keywords-component; job shop scheduling, answer set programming, logic programming.

I. Introduction

Many real-life problems belong to the class of NP-complete problems [19]. Logic Programming under answer set semantics provides a powerful language for a logical formulation of these problems. Its nondeterministic computation liberates the user from the tree-search programming. Answer set programming (ASP) is a form of declarative programming that is emerged from logic programming with negation and reasoning formalism that is based on the answer set semantics [1, 2]. ASP is considered in the late 1990s as a new programming paradigm [3]. Answer set programming languages has been used to solve many real life application problems, among them, production configuration [4], decision support for NASA shuttle controllers [5], synthesis of multiprocessor systems [6], reasoning tools in biology [7, 8], team building [9], composition of Renaissance music [10], and many more. A number of solvers have been proposed, such as: smodels [11, 12, 13], dlv [14], cmodels [17], assat [15, 16], and clasp [18].

In this paper, we show an application of Answer Set Programming to real life job shop scheduling problem occurring in a factory. The job shop scheduling problem (JSSP) is a very important practical problem. Efficient methods of solving it can have major effects on profitability and product quality. However, the JSSP is considered a member among the worst class of NP complete problems [7]. In general, the difficulty of the general JSSP makes it very hard for conventional search based methods to find near optima in reasonable time.

The JSSP is to schedule jobs on different machines minimizing the total time spent to complete all jobs. Given an n-jobs and m-machines, each job comprises a set of operations which must each be done on a different machine for different specified processing time, in a given job-dependent order. Each job must be processed in an uninterrupted fashion or a non-preemptive scheduling environment. It is not necessarily for a job to visit all the machines. The job can visit a subset of the existing set of machines. Each job has a release time and a due time to complete. The release time of a job is the arrival time for that job. The due time is the time that the job must be completed.

Example 1: assume we have 3-jobs j_1, j_2 and j_3 and three machines m_1, m_2, and m_3. Table 1, shows the pre-specified order of operation for each job on the machines. The pair (m, t) specifies the processing time for a particular operation on each machine. For example, to complete job J1, it must completes three operations in the following order: visit machine M_1 for 7 unit of time, then machine M_3 for 8 unit of time, then machine M_2 for 10 unit of time. The table also, shows the due time and release time for each job.

Table 1 Job shop scheduling example, where (m, t) is the machine name and processing time pair for the operation

Jobs	Operations			Release Time	Due Time
	(m, t)	(m, t)	(m, T)		
J1	1, 7	3, 8	2, 10	2	25
J2	2, 6	1, 4	3, 12	4	30
J3	1, 8	2, 8	3, 7	0	35

A standard 6x6 benchmark problem (i.e. j=6 and m=6) from [20]. A legal schedule is a schedule of job sequences on each machine such that each job's operation order is preserved, a machine can process at most one operation at one time, and different operations of the same job are not simultaneously processed on different machines. The problem is to minimize the total time elapsed between the beginning of the first operation and the completion of the last operation (this is called the makespan). Other measures of schedule quality exist, but shortest makespan is the simplest and most widely used criteria. For example 2, the minimum makespan is known to be 55.

Several methods used to solve the JSSP using B&B [22], simulated annealing [23], tabu search [24, 25, 26] and genetic algorithms [27, 28, 29, 30, 32], practical swarm optimization [31]. However, up to our knowledge, there is no answer set program implementation for the JSSP.

In this paper, we first give a brief overview of Answer Set Programming and its semantics. We then, present the job shop scheduling problem formally. After that, we present the solution to the job shop scheduling problem under Answer Set Programming. Finally, we present experimental results and conclusion.

II. BRIEF OVERVIEW OF ANSWER SET PROGRAMMING

We briefly recall the basics about ASP. An ASP-program is a collection of rules of the form

$$a_0 \leftarrow a_1, \ldots, a_m, not\ a_{m+1}, \ldots, a_n \qquad (1)$$

Where, each a_i is an atom. The head of the rule is a positive atom which is the left hand side of the clause in (1). The body of the rule is composed of literals (a literal is an atom or its negation, denoted by *not a*) which is on the right side of the clause in (1). A rule without body is a fact. A rule without head is a constraint. Also, the rules can be positive (m>0); negative (n>0) or both (m>0 and n>0). The symbol *not* stands for default negation, also known as negation as failure.

If P is a ground, positive program (no negation as failure used), a unique answer set is defined as the smallest set of literals constructed from the atoms occurring in program P (minimal model). The last definition can be extended to any ground program P containing negation by considering the reduct of P with respect to a set of atoms X obtained by the Gelfond-Lifshitz's operator [1]. The reduct, P^X, of P relative to X is the set of rules:

$$a_0 \leftarrow a_1, \ldots, a_m$$

for all rules (1) in P such that $a_{m+1}, \ldots, a_n \notin X$. Then P^X is a program without the negation *not*. Then X is an answer set for P if X is an answer set for P^X.

Once a program is described as an ASP-program P, its solutions, if any, are represented by the answer set of P. One important difference between ASP semantics and other semantics is that a logic program may have several answer sets or may have no answer set at all.

Answer Set Programming is a totally declarative language. ASP programs are not algorithms describing how to solve the problem; the program is just a formal description of the problem. The solution is completely found by the solver. An ASP solver requires grounded programs as input, and that is why before searching the answer set or solutions, the program is grounded by a preprocessor. Actually there are many ASP's solvers such as: smodels, dlv, assat, cmodels, and clasp. The computation of answer sets is done in two phases: (i) grounding of the logic program (P): which is eliminating variables to obtain a propositional program ground(P). (ii) Computation of answer sets on the propositional program ground(P).

III. JOB SHOP SCHEDULING PROBLEM

Job shop scheduling is an optimization problem in which jobs are assigned to resources at particular times. There are several problem formulation for the JSSP, we have adopted the one presented by [32] as follows:

Given a set of n-jobs $J=\{j_1, j_2, \ldots, j_n\}$, and m-machines $M = \{M_1, \ldots, M_m\}$. Let n_j be the number of operations of job j. Denote O_{jkq} the operation k of job j to be processed on machine q, T_{jkq} and P_{jkq} be the start time and processing time of operation O_{ikq} respectively. Denote r_j and d_j the release time (earliest start time) and due time (latest ending time) of job j. Let S_j denote the set of operation pairs (O_{jkp}, O_{jlq}) of job j, where O_{jkp} must be processed before O_{jlq}. Let R_q be the set of operations O_{jkq} to be processed on machine q. Our goal is to schedule all jobs on m-machines, while trying to minimize the completion time or the makespan. The makespan is the total time of the schedule (that is, when all the jobs have finished processing). Given a schedule U, the completion time for a job j is $C^U_j=\max(T_{jkq}+P_{jkq})$ for all $k\in\{1, \ldots, n_j\}$, $j \in J$ and $q \in M$. the makespan of a schedule S is the maximum completion time over all jobs in schedule U: $C^U_{max}=\max_{j\in J} (T_{jkq}+P_{jkq})$, $k\in\{1, \ldots, n_j\}$ and $q \in M$. The job shop scheduling problem is represented as follows:

Minimize C^U_{max}, where, $C^U_{max} = \max_{j\in J}(T_{jkq} + P_{jkq})$
Subject to:

1) $T_{jwq} - T_{jkq} >= P_{jkq}$, where $(O_{jkp}, O_{jwp}) \in S_j$, $k, w \in \{1, \ldots, n_j\}$, $j\in J$ and $q\in M$.

2) $T_{jwq} - T_{ikq} >= P_{ikq}$ or $T_{ikq} - T_{jwq} >= P_{jwq}$, where $O_{ikq}, O_{jwq} \in R_q$, $i, j \in J$, $q \in M$, $k \in \{1, \ldots, n_i\}$ and $w \in \{1, \ldots, n_j\}$.

3) $r_j <= T_{jkq} <= d_j - P_{jkq}$, $j \in J$, $k \in \{1, \ldots, n_j\}$, and $q \in M$.

Equation (1) represents the sequence constraint; Equation (2) represents resource constraints in a disjunctive format; and equation (3) represents the release and deadline time constraints.

IV. Problem Description and Resolution in ASP

In this section, we describe job shop scheduling problem in the language of gringo which is the grounder for the answer set programming solver. Initially, all operations of each job are in a waiting state. If an operation is selected for processing then the operation waiting state is removed. Once the operation processing is completed the operation state is in the complete state. Similarly, initially all machines are in the empty state. If a machine is selected for processing an operation, then the machine empty state is removed. If the machine completes processing the operation, then the machine will be in the empty state again. Six conditions need to be satisfied:

N1: When a machine selects an operation for a job, the machine must not be busy.
N2: Each operation of a job selected for processing must be in non complete state.
N3: each operation of a job selected for processing must be completed before the due time.
N4: Each operation of a job selected for processing must be after the release time for that job.
N5: The operation processing order must be preserved.

A. *Constructing the data module D_1 of ASP:*

This module defines an instance of the JSSP. This module consists of a list of jobs, operations and machine. The jobs list is defined as a fact of the following form:

job(jobName, releaseTime, dueTime).

The operations list for each job is defined as a fact of the following form:

operation(Job, operation, processTime, machine).

To define the order of processing of operations in each job, the following operation dependency fact is defined:

dep(jobName, operation1, operation2)

This facts means that 'operation1' depends on 'operation2' in a particular job.

The machines list available is defined as a fact of the following form:

machine(machineName).

B. *The job shop schduling preparation module D_2:*

This module defines new predicate that will simplify and speeding up finding the answer set models of the JSSP. In this module we assume that the total schedule time 'n' is specified by the user. It consists of the following rules:

- The first group consists of determining operation dependency. It is suffice to write the following rules:

  dependent(J, O) :- dep(J, O, O1).

  nonDependent(J, O) :- operation(J, O, P, M),
  not dependent(O).

  The first rule defines all operations that are dependent on some other operations, i.e. operations that cannot be executed until some operation completes its execution. The rule defines that operation 'O' of job 'J' is dependent on some other operation. The second rule defines the operations that are not dependent on other operations, i.e. operations that are at the beginning of each job. Therefore, we have two kinds of operations; the non-dependent operations and the dependent operations.
- The second group consists of one rule that finds the total time to execute all operations for each job. This is done in ASP as:

  totalTimeJob(J, T) :- T = #sum { P, O:
  operation(J, O, P, M) }, job(J, R, D).

  This rule uses the aggregates 'sum' to find the total time needed to execute all operations of each job 'J' available.
- The third group consists of finding the possible start time of each operation in each job. The start time is a range of possible times. Assume we have a job with the following operations-processing time pair: (O_1, t_1), …, (O_s, t_s). In addition, assume that the operations are ordered in the same order listed, i.e. O_2 depends on O_1, O_3 depends on O_2, …, O_s depends on $O_{(s-1)}$. Assume also that the total time of all operations is $T = \sum_{i=1}^{s} t_i$. This time T is the minimum time needed to finish all operations of that job. Let the total schedule time specified by the user is n. The first operation O1 may start from 0 to n-T. The n-T is because operation O_1 cannot starts at n-T+1, since it needs a minimum of T-time unit to complete. Similarly, operation O_2 may start from t_1 (which is after completing O_1) to n-T+t_1. The following set of rules will do that:

  op(J, O, P, M, R, E) :- operation(J, O, P, M),
  notDependent(J, O), totalTimeJob(J, T),
  E=n-T, job(J, R, D), E<D.
  op(J, O, P, M, R, D) :- operation(J, O, P, M),
  notDependent(J, O), totalTimeJob(J, T),
  E=n-T, job(J, R, D), E>=D.

  op(J, O, P, M, S, E) :-
  operation(J, O, P, M), dep(J, O, O1),
  op(J, O1, P1, M1, S1, E1),
  totalTimeJob(J, T), S=T1+P1,
  E = n-T+S, job(J, R, D), E<D.

op(J, O, P, M, S, D) :-
operation(J, O, P, M), dep(J, O, O1),
op(J, O1, P1, M1, S1, E1),
totalTimeJob(J, T), S=T1+P1,
E = n-T+S, job(J, R, D), E>=D.

The rules determines the possible range of time [S, E] to execute an operation 'O' of job 'J; with processing time 'P' on machine 'M'. The first two rules are for non-dependent operations. The rules compare the due time for the job with possible time range and select the earlier time. The second two rules determine the possible time range [S, E] to execute dependent operations. It computes the possible time range for the operation from the dependent operation and compares the times with the due time of the job. These four rules handle the constraint N3 and N4 that are listed in section IV.

- The fourth set of rules consists of finding the time range for each machine, it is suffice to write:

Initial(M) :- operation(J, O, P, M),
notDependent(J, O), job(J, R, D),
R=0, D>0.
minPr(M, Pr) :- Pr= #min { P : task(J, O, P, M),
notDependent(J, O) }, machine(M), initial(M).
minPr(M, 0) :- machine(M), not initial(M).

time(M, 0) :- machine(M).
time(M, P..n) :- minPr(M, P).

This group of rules determines the lower bound of the time-steps from [0, n] for each machine – n is determined by the user and represents the total time for the schedule.

The first rule determines the availability of non-dependent operations for each machine. The second rule computes the least time to start a machine 'M'. It computes the least processing time among all non-dependent operations for a particular machine. This time will be considered the lower bound for the time of each machine. The third rule says that the lower bound for a machine that do not have non-dependent operations is zero. The fourth rule specifies that all machines must start at zero. The fifth rule specifies that the time step for all machines is the lower bound computed to the total time 'n' of the schedule that is determined by the user. These rules have great effects of the performance of ASP.

Note, the rules defined in this module are all facts and it accelerates the search for a solution significantly.

C. The job shop schduling solver module D_3:

This module describes solving the job shop scheduling problem. We are mainly interested in finding a schedule of processing all operations of all jobs in the shortest makespan.

The module D_3 will contain fluent *busy(M, T)* – "machine M is in a busy state at time T", fluent complete(J, O, T) – "operation O of job name J is in a complete state at time T". One type of action – 'select', will be used.

The transition diagram of D_2 will be described by group of axioms:

- The first group defines the executability conditions for actions. We have one action "select(J, O, P, M, T)" which means selecting an operation O with a processing time P of job J to be executed on machine M at time T. The rules are as follows:

0 { select(J, O, P, M, T) : op(J, O, P, M, S, E),
T>=S, T<=E } 1 :- machine(M), time(M, T),
not busy(M, T), avail(M, T).

The action "select" selects one operation among the operations available to execute on a machine. The rule is a choice rule that is bounded by 0 and 1. This means selecting an action at any time for a machine is arbitrary. It means further that an action is either selected or not selected. These rules are the generate rules that will generate all possible schedules. The rule will execute when the machine 'M' is not busy at time 'T' ('not busy(M, T)') and there is available jobs to select from 'avail(M, T)'. The rules 'busy(M, T)' and 'avail(M, T)' will be explained later. The 'not busy(M, T)' in the body of the rule handles the N1 constraint listed in section IV.

The rule states that all machines are in an empty state at time zero.

- The second group contains causal laws describing direct effect of actions. For example it is suffice to have the rules:

complete(J, O, T+P) :- select(J, O, P, M, T),
op(J, O, P, M, S, E), time(M, T),
T>=S, T<=E.
busy(M, T) :- select(J, O, P, M, T1), time(M, T1),
op(J, O, P, M, S, E),T1>=S, T1<=E,
time(M, T), T>T1, T1<T+P.

The first rule says that if an action "select" selects an operation for processing in a machine; then the operation will be in a complete state after passing the operation's processing time. The second rule says that a machine 'M' will be busy all the time step when it is selected for processing an operation 'O' of job 'J' with processing time 'P' at time 'T'.

- Two auxiliary rules are needed as follows:

```
selectedMachine(M, T) :- select(J, O, P, M, T),
            op(J, O, P, M, S, E),
            time(T), T>=S, T<=E.
avail(J, O, T) :- op(J, O, P, M, S, E), time(M, T),
            T>=S, T<=E.
```

This first rule is trivial. It states that a machine is selected when it has been selected for processing some operation of a job at some time T. The second rule checks the availability of the rules at each time T.

- The third group of rules are constraints that eliminate unwanted answer set models, which are defined as follows:

```
:- select(J, O, P, M, T), op(J, O, P, M, S, E),
   complete(J, O, T), time(M, T), T>=S, T<=E.

:- select(J, O, P, M, T), operation(J, O, P, M),
   time(M, T),  job(J, R, D), T+P>D.
:- select(J, O, P, M, T), operation(J, O, P, M),
   time(M, T),  job(J, R, D), T < R.

:- select(J, O, P, M, T), operation(J, O, P, M),
  dep(J, O, O1), not complete(J, O1, T),
  time(M, T).
:- select(J, O, P, M, T), op(J, O, P, M, S, E),
   time(M, T), T>=S, T<=E,
   op(J1, O1, P1, M, S1, E1), J!=J1,
   not complete(J1, O1, T), T+P>E1.
```

The first constraint states that an operation should not be selected if it is already in a complete state. This is constraint N2 of the section IV. The second constraint states that an operation should not pass the due time of the job (this is constraint N3 of the section IV). The third constraint states that an operation should not be selected for processing before its arrival time or release time (this constraint N4 of section IV). The fourth constraint states that an operation should not be selected if its pre-operation is not in a complete state (i.e. order of operations is preserved). This is constraint N5 of the section IV. The fifth constraint rejects the selection of an operation 'O' of a job 'J' that will cause another operation 'O1' of job 'J1' (J is different than J1) on the same machine 'M' out of its time range to execute. However, the second and third constraint can be eliminated since they are handled by the 'op/6' rules.

- The fourth group consists of rules that make sure that all operations of all jobs are completed. This can be written as:

```
finish(J, O) :- operation(J, O, P, M, S, E),
            complete(J, O, T), time(M, T).
:- not finish(J, O), operation(J, O, P, M).
```

The first rule finds all operations of a job that are complete. The second rule is a constraint that rejects answer set models that includes non complete operations.

- To find the schedule of minimum makespan , the following optimization rules are added:

```
makespan(X) :- X =
         #max {T+P:select(J,O,P,M,T) }.
#minimize {X: makespan(X)}.
```

The first rule returns the makespan for the schedule produced. It uses the aggregate max of clingo to find the maximum time to complete the schedule. The second rule finds the minimum makespan among all schedules produced by the module M_2.It uses the optimization statement "minimize" of gringo to find the schedule with the minimum makespan.

To complete the definition of the transition diagram of the domain, we need to specify what fluents do not change as the results of actions. This is a famous Frame Problem from (McCarthy and Hayes, 1969) where the authors suggested to solve it by formalizing the Inertia Axiom which says that "things tend to stay as they are". This is a typical default which can be easily represented in answer set programs. In our particular case, it will have the form:

One special fluent is the complete state fluent, which once an operation of a job is complete, then it will stay in that state until the end of the schedule. This is represented as:

```
complete(J, O, T+1) :- complete(J, O, T),
      op(J, O, P,M, S, E), time(T), T<n, T>=S.
```

V. Experimental Results

Our experiments were designed to assess the performance of each of the ASP on job shop scheduling problems. We used the ten scheduling problems produced by Taillard with 7 jobs and 7 machines. Each of these problems consists of forty-nine operations to be scheduled subject to sequencing restrictions and resource capacity constraints. The operations are grouped into seven jobs of seven operations each. Operations within each job must be performed in order. Further, each job requires one of seven resources and each resource can be used by at most one job at a time.

Table (3) shows running the answer set program on several instances of the job shop scheduling problem. The ASP was run on An Intel core 2 due laptop with 1.2 GHz processor and 4GB RAM is used.

TABLE III. Experimental results of ASP

Problem Instance	***CPU time in seconds***	***Shortest makespan found by ASP***	***Known shortest makespan***
3x3 Example 1	***0.094***	***42***	***42***
4x4 Instance	***2.29***	***272***	***272***
5x5 Instance	***4.96***	***333***	***333***
Ft6: 6x6	***0.45***	***55***	***55***
7x7 instance1	***47.00***	***590***	***590***
7x7 instance2	***36.44***	***558***	***558***
7x7 instance3	***54.77***	***605***	***605***
7x7 instance4	***59.51***	***671***	***671***

Note that, the problems with large value of time steps can have big influence on the program's performance when employing Answer Set programming as solution method, since the number of answer set candidates that need to be checked is heavily dependent on the number of time steps. For larger problems such as 10x10 and 20x5 the cpu time is large (more than 10 hours).

VI. CONCLUSION

In this paper, we present an approach that uses ASP to represent the job shop scheduling problem to produce optimal plans. Job shop scheduling is known to be a hard problem. We have proposed to investigate and evaluate the capabilities of ASP to job shop scheduling problem. Furthermore, we investigated writing heuristic methods in answer set programs to solve the job shop scheduling problem. ASP is expressive enough to represents the constraint of the job shop scheduling problem. The paper also shows the expressive use of the aggregates and optimization sentences defined in the 'clingo' solver. Job shop scheduling problem can be a killer application of ASP when the time step increases and solving other job shop scheduling problems is an interesting future work.

Although we cannot solve the general case of the job shop scheduling problem satisfactorily at the moment, we note that the solution methodology proposed in our program could be very useful for further development in future work. In conclusion, ASP as a declarative programming language has been shown to be an elegant and highly maintainable approach for solving the job shop scheduling problem, but we have to admit that there is still work to do in order to obtain a competitive and robust solver.

REFERENCES

[1] M. Gelfond and V. Lifschitz, "the Stable Model Semantics for Logic Programming," ICLP/SLP, pp. 1070-1080, 1988.

[2] C. Baral. "Knowledge Representation, Reasoning and Declarative Problem Solving," Cambridge University Press, 2003.

[3] V. Marek and M. Truszczyński, "Stable models and an alternative logic programming paradigm," In Apt, Krzysztof R. The Logic programming paradigm: a 25-year perspective, pp. 169-181, Springer. 1991.

[4] T. Soininen and I. Niemela, "Developing a declarative rule language for applications in product configuration, " In Gupta, G., ed.: Proceedings of the First International Workshop on Practical Aspects of Declarative Languages (PADL'99), pp. 305–319, Springer 1999.

[5] M. Nogueira, M. Balduccini, M. Gelfond, R. Watson, and M. Barry, "An A-prolog decision support system for the space shuttle," *Proceedings of the Third International Symposium on Practical Aspects of Declarative Languages (PADL'01)*, pp 169-183. Springer-Verlag, 2001.

[6] H. Shebabi, P. Mahr, C. Bobda, M. Gebser, and T. Schaub, "Answer set vs integer linear programming for automatic synthesis of multiprocessor systems from real-time parallel programs," Journal of Reconfigurable Computing, 2009.

[7] E. Erdem, and F. Ture, "Efficient haplotype inference with answer set programming," Proceedings of the Twenty-third National Conference on Artificial Intelligence (AAAI'08), pp. 436–441, 2008.

[8] M. Gebser, T. Schaub, S. Thiele, and P. Veber, "Detecting inconsistencies in large biological networks with answer set programming," Theory and Practice of Logic Programming 11 (2), pp1–38, 2011.

[9] G. Grasso, S. Iiritano, N. Leone, V. Lio, F. Ricca, and F. Scalise, "An ASP-based system for team-building in the Gioia-Tauro seaport". In Proceedings of the Twelfth International Symposium on Practical Aspects of Declarative Languages (PADL'10), Volume 5937 of Lecture Notes in Computer Science., Springer-Verlag, pp. 40–42, 2010.

[10] G. oenn, M. Brain, M. de Vos, and J. Fitch, "Automatic composition of melodic and harmonic music by answer set programming". Proceedings of the Twenty-fourth International Conference on Logic Programming (ICLP'08). Volume 5366 of Lecture Notes in Computer Science., Springer-Verlag, pp. 160–174, 2008.

[11] P. Simons. "Efficient implementation of the stable model semantics for normal logic programs," Research Report 35, Helsinki University of Technology, September 1995.

[12] P. Simons, I. Niemels, and T. Soininen, "Extending and implementing the stable model semantics". Artificial Intelligence 138 (1-2) pp. 181–234, 2002.

[13] I. Niemelä and P. Simons. Smodels - an implementation of the stable model and well-founded semantics for normal logic programs. In *Proceedings of the 4th International Conference on Logic Programming and Nonmonotonic Reasoning*, volume 1265 of *Lecture Notes in Artificial Intelligence*, pages 420-429, Dagstuhl, Germany, July 1997.

[14] N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello. "The DLV system for knowledge representation and reasoning," ACM Transactions on Computational Logic, 7(3):499–562, July 2006.

[15] Yu. Lierler and M. Maratea, "Cmodels-2: SAT-based answer set solver enhanced to non-tight programs," In Proc. of LPNMR-7, 2004.

[16] F. Lin and Yu. Zhao, ASSAT: "Computing answer sets of a logic program by SAT solvers," Artificial Intelligemce 157(1-2), pp. 115-137, 2004.

[17] V. Lifschitz and A. Razborov. Why are there so many loop formulas? ACM Transactions on Computational Logic, pp 261-268, 2006.

[18] M. Gebser, B. Kaufmann, A. Neumann and T. Schaub, "clasp: A Conflict-Driven Answer Set Solver," LPNMR'07, 2007.

[19] M. R. Gary and D. S. Johnson, "Computers and Intractability: a Guide to the Theory of NP Completeness," Freeman 1979.

[20] .J. F. Muth and G. L. Thompson. Industrial Scheduling. Prentice Hall, Englewood Cliffs, New Jersey, 1963.

[21] H. Shebabi, P. Mahr, C. Bobda, M. Gebser, and T. Schaub, "Answer set vs integer linear programming for automatic synthesis of multiprocessor systems from real-time parallel programs," Journal of Reconfigurable Computing, 2009.

[22] P. Bucker, B.Jurisch, and B. Sievers. A branch and bound algorithm for job-shop scheduling problem. Discrete Applied Math, vol 49, pp. 105-127, 1994.

[23] H.R. Loureco. Local Optimization and the job shop scheduling problem. European Journal of Operational Research 83, pp. 347-364, 1995.

[24] E. Nowicki and C. Smutnicki. A Fast Tabu search Algorithm for the Job-Shop problem. Management Science, 42(6), pp. 797-813, 1996.

[25] E. Nowicki and C. Smutnicki. An advanced Tabu search Algorithm for the Job-Shop problem. Journal of Scheduling, 8(2), pp. 145-813, 2005.

[26] C. Y. Zhang, P. Li and Z. Guan. A very fast TS/SA algorithm for the job-shop scheduling problem. Computers and Operations Research, 35, pp. 282-294, 2008.

[27] S. M. K. Hasan, R. Saarker and D. Cornforth. GA with Priority Rules for Solving Job-Shop Scheduling Problems. Proceeding of the IEEE Congress on Evolutionary Computation, CEC 2008, June 1-6, Hong Kong, China, pp. 3804-3811, 2008.

[28] R. Qing-doa-er-ji and Y. Wang. A new hybrid genetic algorithm for job-shop Scheduling Problem. Computers and Operations Research, 39(10), pp. 2291-2299, 2012.

[29] D. Y. Sha and C. Hsu. A hybrid practical swarm Optimization for job-shop schedulingptoblem. Computers and Industrial Engineering, 51(4), pp. 791-808, 2006.

[30] L. Wang and D. Zheng. An effective hybrid optimization strategy for job-shop scheduling problem. Computers and Operations Research, 35, pp. 282-294, 2008.

[31] D. Y. Sha and C. Hsu. A hybrid practical swarm Optimization for job-shop schedulingptoblem. Computers and Industrial Engineering, 51(4), pp. 791-808, 2006.

[32] L. Wang and D. Zheng. An effective hybrid optimization strategy for job-shop scheduling problem. Computers and Operations Research, 35, pp. 282-294, 2008.

[33] S. Yang. An imporved Adaptive Neural Network for Job Shop Scheduling. 2005.