Incremental temporal reasoning in job shop scheduling repair

Incremental Temporal Reasoning in Job Shop Scheduling Repair

Y. Huang[1,2], L. Zheng[1], Brian C. Williams[2], L. Tang[1], H. Yang[1]
[1] Department of Industrial Engineering, Tsinghua University, Beijing, China
[2] CSAIL, MIT, Cambridge, MA, USA
huangyi00@mails.tsinghua.edu.cn, lzheng@tsinghua.edu.cn, williams@mit.edu

Abstract – **A working predictive schedule can be useless because of the various external or internal disruptions in a job shop. Total rescheduling may cause problems such as shop floor nervousness. Thus, the job shop scheduling repair (recovery) approach aims at generating a solution satisfying the updated constraints and making deviations minimized. We propose an incremental temporal reasoning approach in this paper to solve job shop scheduling repair problems. Specifically, such a problem is formulated as a disjunctive temporal problem (DTP), framed as an optimal constraint satisfaction problem (OCSP) formally, and finally solved by performing an algorithm integrating incremental temporal consistency and efficient candidate generation. Through involving human interactive mechanism, domain experts can make higher quality decisions by balancing makespan and deviations.**

Keywords – **Job shop scheduling repair (recovery), incremental temporal reasoning, disjunctive temporal problems, optimal constraint satisfaction problems, constraint–based scheduling**

I. INTRODUCTION

Deviations from predictive schedules occur when the job shop experiences both external and internal disruptions such as machines break down, workers do not perform as expected, urgent jobs arrives, target dates or order quantities changes on short notice, wrong job quantities are produced, materials or supplies do not arrive when expected, parts are misplaced, and tools defects [1-3]. These disruptions appear unexpectedly during manufacturing and cause the original predictive schedules useless. Distinctly, scheduling is an ongoing reactive process where evolving and changing circumstances continually force reconsideration and revision of pre-established plans [4].

A simple approach would be to totally reschedule the system when the deviation occurs [5]. However, this approach is not encouraged in industry as the new schedule can differ considerably from the old one and this is not desirable since many other decisions such as assignment of personnel, delivery of raw material and the subsequent processing of the jobs in other facilities, may be severely disrupted. This phenomenon is commonly referred to as shop floor nervousness [6].

Therefore, another approach is scheduling recovery and repair [7, 8] that modifies the original predictive schedule to accommodate sudden disruptions avoiding the total rescheduling of the system as far as possible. In this paper, we develop an incremental temporal reasoning approach for scheduling repair. Specifically, a job shop scheduling repair problem, whose objective is generally to minimize the deviations, is formulated as a disjunctive temporal problem (DTP), which is then framed as an optimal constraint satisfaction problem (OCSP). An algorithm developed from the conflict-directed A* algorithm (CDA*) [9] and the incremental temporal consistency algorithm (ITC) [10] is proposed to solve the OCSP.

II. REACTIVE SCHEDULE REPAIR

A. Related Work

A pragmatic reactive scheduling repair system in job shop faces several challenges to design and implement.

First, reactive schedule repair can be evaluated based on the original predictive schedule (or the previous revised schedule) and large amount of information indicating the status of the job shop. In traditional way, it is time-consuming to collect the information from the job shop. However, RFID and related sensor technologies provide the potential to change the way we control manufacturing processes in a fundamental manner [11], specifically gathering massive observable data in real time. In addition, tools such as reactive monitoring systems will manipulate these data and estimate the current status of the job shop, so as to support reactive schedule repairing.

The predictive schedules are usually based on optimization objectives such as minimizing tardiness, maximizing resource utilization, etc. Any recovery or repair applied will cause a deviation from the original optimized schedule. Thus, a better schedule strategy is one that leads to minimum deviation of the performance measures while incorporating the necessary modifications and repair objectives. The focus is to implement a repair method that can handle the repair of the schedules without compromising on the quality of the schedules [12]. In other words, the objective of reactive schedule repair could be to minimize the deviation in performance during the repair while satisfying the constraints [13] such as routing of jobs, availability of machines, and priority assignment for jobs and machines.

Reactive schedule repair responses to unexpected disruptions during the execution of original predictive schedules. Thus, the minimum time frame is required to react to the disruptions so that the schedule is still active and not out of date by the time the revised schedule is prepared.

Eliciting effective problem solving knowledge from human experts is a difficult task, and human schedulers

978-1-4244-8503-1/10/$26.00 ©2010 IEEE

typically lack the knowledge of solving large and complicated scheduling problems in the sophisticated manner [14]. Thus, human interactive mechanism may be involved in a reactive schedule recovery system allowing those experts to make difficult decisions efficiently armed with the system assistance.

Several various kinds of methods are applied in a reactive environment for schedule recovery, such as right-shift rescheduling, heuristic based approaches, affected operation rescheduling, multi-agent approaches, cased-based reasoning, constraint-based scheduling, fuzzy logic and neural network [2]. This paper will aim at constraint-based scheduling approaches because (1) the real-time status of job shop could be collected by RFID devices and manipulated by a reactive monitoring system, which is built on a constraint-based reactive language (reactive model-based programming language, RMPL [15]); (2) rules and relationships in schedule specifications can be presented by constraints formally; (3) there are a lot of efficient algorithms and engines to solve the constraint-based problem. Miyashita et al. [14] achieve the incremental accumulation and reuse of past experience through case-based reasoning, whereas constraint-based scheduling has been used for the propagation and resolution of the effect of repair. Spragg et al. [16] used constraint based reactive rescheduling, which involves periodic schedule repair, based upon reassignment heuristics, supported by partial order backtracking, to maintain line balance which may be disrupted by workers absence or machines breakdown. [17] provided a review of using constraint satisfaction processing (CSP) technology for scheduling in job shop environment.

B. Formal Definitions

A job shop contains a set of **machines** $M = \{m_1, m_2,\ldots, m_n\}$ and a set of **jobs** $J = \{j_1, j_2,\ldots, j_n\}$. A job, j_i, consists of a set of operations $O_i = \{o_1, o_2,\ldots, o_s\}$, where any two operations may be restricted by a sequence or parallel constraint. Formally, operations in O_i may be recurring, i.e. any two operations $o_x, o_y \in O_i$ may have $o_x = o_y$. Generally, some jobs may have earliest start time and latest end time to restrict the job temporal constraints.

A job shop contains a set of **operations** $O = \{o_1, o_2,\ldots, o_n\}$, where O_i (for the job j_i) $\subseteq O$. An operation, $o_i \in O$, is able to be handled on a set of candidate machines $M_i = \{m_1, m_2,\ldots, m_s\} \subseteq M$ (M denotes the set of all machines). Each operation also has a set of processing time, $T_i = \{t_{i,1}, t_{i,2},\ldots, t_{i,s}\}$, corresponding to M_i.

Rules in a job shop contains: 1) Each machine can handle at most one operation at a time; 2) Each operation needs to be processed during an uninterrupted period of a given length on a given machine; 3) A job can wait an arbitrary amount of time between two operations (i.e. wait in buffer of a machine).

A **job shop scheduling (JSS)** problem is often described by a three-field $\alpha \mid \beta \mid \gamma$. [18] The α field specifies the environment, including relationships, of m machines. The β field indicates a number of characteristics of n jobs, such as precedence relations and release dates. The γ field is characterized as the objective to minimize the makespan for all the jobs. A JSS problem is to find a solution satisfying the constraints in the β field and optimizing the γ field. Specifically, the constraints can be partitioned into three types: temporal constraints to restrict earliest start time and latest end time, temporal constraints to restrict operations sequence of a job, and temporal constraints to restrict operations sequence on a machine.

A **job shop scheduling repair (JSSR)** problem can be extended from the JSS problem. It contains the field α and the field β, as well as an ongoing schedule given by solving Job Shop Scheduling Problem. Generally, part of the ongoing schedule is finished but some disruptions happen, such as a new job is added, the latest end time of an existing job is changed. To avoid the shop floor nervousness, the objective of the JSSR problem is to minimize the deviations from the ongoing schedule.

III. METHODOLOGY

We use an incremental temporal reasoning algorithm to solve the JSSR problem. A candidate schedule can be represented as a simple temporal network (STN). The candidate schedule is a solution when the STN is temporal consistent. In a typical JSS problem, the constraints of earliest start time, latest end time, and the sequence of operations in a job are deterministic. A schedule will be made by arranging the sequence of operations on each machine. Formally, a JSSR problem can be formulated as a disjunctive temporal problem (DTP), which is able to be solved by employing temporal reasoning algorithms.

A. Disjunctive Temporal Problem (DTP)

Generally, a temporal reasoning problem is described by points and intervals, where constraints can be either qualitative or quantitative. Qualitative constraints specify the relative position of paired objects, and quantitative constraints place absolute bounds, or restrict the temporal distance between points [19].

A **simple temporal network (STN)** is defined as a collection of real-valued time point variables V corresponding to instantaneous events, admitting at most one interval constraint on any pair of time points [20], where each interval constraint $c_i = [t_1, t_2]$ restricts the temporal distance. A **disjunctive temporal problem (DTP)** has a similar definition as STN, but each constraint $c_i \in C$ is allowed to be disjunctive, in the form of $c_{i,1} \vee c_{i,2} \vee \ldots \vee c_{i,n}$, where n may be any number [20].

When formulating a JSSR problem into a DTP, each precedence relationship between two operations o_i, o_j on a machine can be formulated as a disjunctive temporal constraint, o_i either before or after o_j. By making choice for each disjunctive temporal constraint, either before or

after, a DTP can be reduced into a STN. Then a solution may be found when the STN is temporal consistent.

Fig. 1 shows a formulation example, where the JSSR problem has 3 jobs and 3 machines. Each 6 horizontal points denote the start and end time points of a job's operations. Fig. 1 (a) represents the disjunctive temporal constraints in a DTP, which describe the operation precedence relationships on a machine. Specifically, "a1" and "a2" consist of a disjunctive constraint that represents the precedence relationship between the operations "job1.op1" and "job3.op1". After making choice of "a1", "c2" and so on, the DTP could be reduced into a STN like Fig. 1 (b) that represents a candidate schedule. It worth noting that the dashed arcs represent the constraints restricting the earliest start time and the latest end time of the schedule.

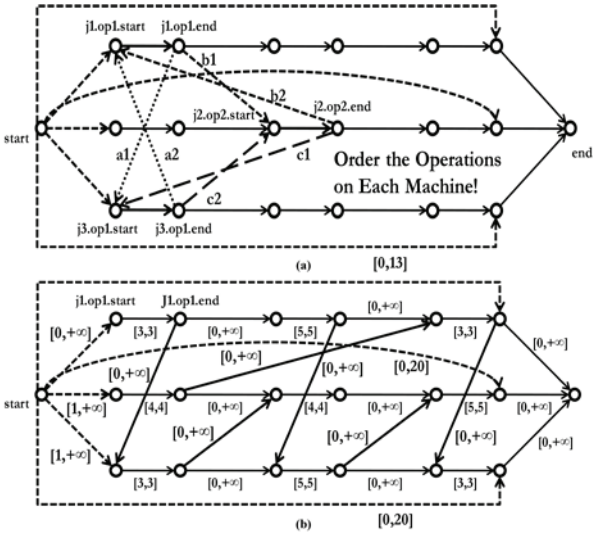

Fig. 1. (a) Disjunctive constraints that describe the operation precedence relationships on a machine; (b) A STN that presents a candidate schedule.

B. Optimal Constraint Satisfaction Problem (OCSP)

A **constraint satisfaction problem (CSP)**, $\langle X, D_X, C_X \rangle$, consists of a set of variables $x_j \in X$ that range over a finite domain $D_{Xi} \in D_X$, and a set of constraints $C_X : X \rightarrow \{\text{True, False}\}$. A solution is any assignment to X that satisfies C_X. While an **optimal constraint satisfaction problem (OCSP)**, $\langle \text{CSP}, Y, g \rangle$, consists of a CSP = $\langle X, D_X, C_X \rangle$, a set of decision variables $Y \subset X$, and a multi-attribute cost function $g : Y \rightarrow \text{R}$, that is mutually preferentially independent (MPI).

A DTP for a JSSR problem is easy to be framed as an OCSP, whose objective is to minimize the deviations from the ongoing schedule. Specifically, to frame a JSSR DTP into a OCSP, every pair of operations on the same machine is encoded as a set of decision variables Y. Each decision variable, $y_i \in Y$, has two values "before" and "after", either of which is respectively assigned a cost c_i rendering the similarity to the old schedule. If there are m operations on a specified machine, $m(m\text{-}1)/2$ pairs of decision variables will be constructed. For example, in a JSSR problem, "job1.op1", "job2.op2" and "job3.op1" are on the same machine. Three decision variables are then constructed: "job1.op1/job2.op2", "job1.op1/job3.op1" and "job2.op2/job3.op1". Constraints of earliest start time, latest end time and operations sequence of a job will be encoded as lower bound and upper bound form. Constraints of operations sequence on the same machine will be mapped to decision variables respectively.

C. Incremental Temporal Reasoning

In this paper, an incremental temporal reasoning algorithm, integrating the conflict-directed A* algorithm (CDA*) [9] and the incremental temporal consistency algorithm (ITC) [10], is used to solve the OCSP for a JSSR problem. This algorithm constrains the major steps as Fig. 2.

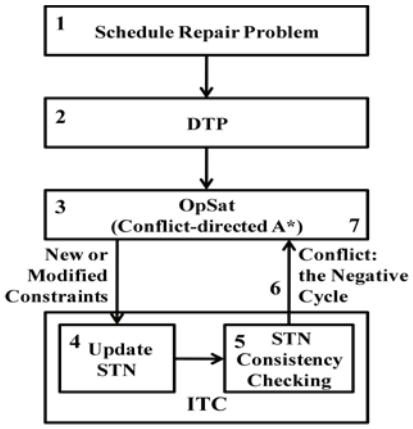

Fig. 2. The incremental temporal reasoning

Step 1: A JSSR problem is formulated as a DTP. The input contains the job shop model (n jobs, m machines, relationships of machines and operations, intervals of operations, etc.), the ongoing schedule, and the finished part of the ongoing schedule. The output is a DTP and related costs, which are generated by comparing disjunctive constraints and the ongoing schedule.

Step 2: A DTP and related costs are framed as an OCSP, where an OpSat form is used to represent and interpret it.

Step 3: If the first time, (a) there are not any known conflict in the OpSat. According to the decision variables and their costs, a candidate schedule will be selected, i.e. each decision variable will be assigned the value with the biggest cost so as to output a STN without any disjunctive constraints. Else, (b) there are some known conflicts. The CDA* algorithm is used to give a candidate schedule resolving all known conflicts. The output candidate schedule has the largest sum of costs of the decision variables after removing all candidates affected by known conflicts.

Step 4: If the first time, (a) just update the whole STN given by Step 3(a) to the ITC. Else, (b) there is an existing STN. Compare the new given STN (the candidate schedule) and the existing STN. The conflict of the existing STN (a negative cycle) is resolved and some

decision variables have changed values. Update these changes of the constraints to transform the existing STN to the new STN but maintaining the last search results, so that the following STN consistency checking can be continued incrementally based on the last search but not totally restarted.

Step 5: Check the updated STN consistency based on the last search. A STN can be easily to transformed as a distance graph. STN consistency can be checked by detecting negative cycles in the distance graph. Here FIFO label correcting algorithm is applied to detect negative cycles.

Step 6: If no negative cycles is found, the STN is temporal consistent and the repaired schedule is. The algorithm will be stopped here and output the consistent schedule as the final solution. If a negative cycle is found, it will be output as a conflict to update the sets of conflicts maintained in OpSat.

Step 7: Update the given conflict given by the step 6 into the OpSat conflict set. Repeat the Step 3(b) to generate another feasible candidate schedule.

Specifically, one of the most important issues to affect computing performance is the STN consistency checking algorithm, which is specifically a FIFO label correcting algorithm in the original ITC algorithm [10].

The original algorithm is inefficient because of the termination condition of detecting the negative cycles. It will take too much unnecessary time to traversal the nodes until one of them is less than $-nC$, where n denotes the number of the nodes and C denotes the largest absolute value of some arc length. For example, in a 10 jobs, 10 machines job shop problem, there are more than 200 nodes and the largest absolute value of some arc (i.e. the largest interval upper bound) may be thousands.

Therefore, we improve the algorithm by using a much faster termination condition: "Run the FIFO label correcting algorithm of label correcting, and stop if you have scanned any node at least n times". It can be easily proven by the theorem: "The FIFO label correcting algorithm finds the minimum length path from 1 to j for all j in N in $O(nm)$ steps, or else shows that there is a negative cost cycle." N is the set of nodes in the STN graph, n is the number of the nodes and m is the number of edges.

CDA* accelerates the search process of the traditional A* algorithm by eliminating subspaces around each inconsistent state. CDA* guides its search using conflicts, which are descriptions of states that are inconsistent with the OCSP. Specifically, CDA* generates a best valued candidate S that resolves all discovered conflicts. It tests S for consistency against the CSP using any suitable CSP algorithm. When S tests inconsistent, one or more conflicts that are inconsistent in a manner similar to S are extracted. When a new conflict is discovered, it is recorded and used to update the search queue of candidates to be explored, called kernels. CDA* then repeats to generate the next best candidate and test until the desired solutions are found or all candidates are eliminated.

IV. EXPERIMENTAL RESULTS

An experimental case is build to test the algorithm performance. We develop a test case based on a job shop scheduling benchmark from Lawrence, which is also known as LA19 [21]. The original problem is a JSS problem with 10 jobs and 10 machines, where the optimal solution is known to be 842 (see Fig. 3) computed within 300s [22, Baptiste, 1995 #586]. To development our test case, we use the optimal schedule as an ongoing schedule, and consider an 11th urgent job, which is specifically M0(32), M1(27), M2(50), M3(86), M4(67), M5(25), M6(53), M7(39), M8(44), M9(19). Then the schedule should be repaired to satisfy the resource constraints and the temporal constraints that may contain a new allowed makespan, at the same time has a minimum deviation from the original schedule.

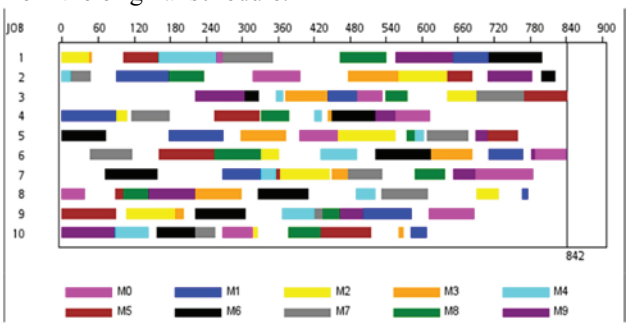

Fig. 3. The ongoing schedule (LA19)

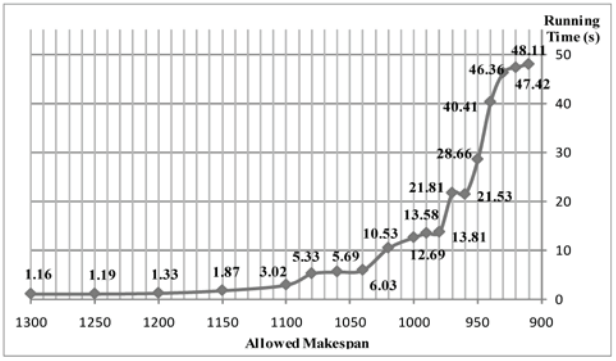

Fig. 4. The experimental result of repairing LA19 schedule

The experimental result of test case is given as Fig. 4. Obviously, when a smaller allowed makespan is given, there will be more adjustments performed on the original job operations, and it would take longer to repair. In our test case, an allowed makespan of 1300 means there is no need to adjust the original job operations, but place the new job after. While it may take almost longest (about 50s) to repair the schedule by giving an allowed makespan of 910, which is also the nearest to the optimal makespan of the new JSS problem. However, since we frequently repair a schedule when it is on execution, the shop floor nervousness caused by too many adjustments should be avoided.

V. CONCLUSION AND DISCUSSION

Using the increment temporal reasoning, the JSSR problem can be abstracted as a disjunctive temporal problem (DTP). The DTP is framed as an optimal constraint satisfaction problem (OCSP), which can be solved by the OpSat engine. The OpSat uses the conflict-directed A* algorithm (CDA*) and the incremental temporal consistency algorithm (ITC) to solve the temporal constraints network. An experiment is given to test the whole algorithm framework.

Actually, the temporal plan network (TPN) [23] is used to encode the job shop schedule repair problem. However, because of the properties of the TPN, the disjunctive constraints from different nodes are difficult to represent. Thus, the JSSR problem is interpreted as a lot of possible choices in TPN. Even in a small size of JSSR, the choices are still a large size $(n!)^m$. Thus finally, we specify the JSSR problem as a DTP and directly frame as an OCSP.

Certainly, there are some works to do in the future. First, the current STN consistency checking algorithm, FIFO label correcting algorithm terminated by "*n* times", is not efficient enough. More negative cycle detection algorithms will be tried to integrate in the incremental temporal reasoning mechanism to perform efficiently. Next, we will try to prune some semantic duplicate constraints when performing negative cycle detection. Finally, the current allowed makespan of the JSSR DTP is given by inputting. In the future, we will try to extend the current OpSat to represent the makespan issue using constraints or variables. Another kind of idea may be allow the OpSat to input a problem and some conflicts, which makes the OpSat also an incremental algorithm.

ACKNOWLEDGMENT

We thank Qing Xiang, Kaiquan Wei, Henri Badaro, Patrick Conrad and David Wang for their discussions and suggestions.

REFERENCES

[1] R. O'Donovan, *et al.*, "Predictable scheduling of a single machine with breakdowns and sensitive jobs," in *International Journal of Production Research* vol. 37, ed: Taylor & Francis Ltd, 1999, p. 4217.

[2] A. S. Raheja and V. Subramaniam, "Reactive Recovery of Job Shop Schedules – A Review," *The International Journal of Advanced Manufacturing Technology,* vol. 19, pp. 756-763, 2002.

[3] O. Gunther, *et al.*, *RFID in Manufacturing*: Springer, 2008.

[4] S. Smith, "Reactive Scheduling Systems," *Intelligent Scheduling Systems,* pp. 155-192, 1995.

[5] R. Shafaei and P. Brunn, "Workshop scheduling using practical (inaccurate) data Part 2: An investigation of the robustness of scheduling rules in a dynamic and stochastic environment," in *International Journal of Production Research* vol. 37, ed: Taylor & Francis Ltd, 1999, p. 4105.

[6] J. Dorn, "Case-based Reactive Scheduling," in *Artificial Intelligence in Reactive Scheduling*, ed: Shapman and Hall, 1994, pp. 32-50.

[7] J. Efstathiou, "Anytime heuristic schedule repair in manufacturing industry," *Control Theory and Applications, IEE Proceedings -,* vol. 143, pp. 114-124, 1996.

[8] J. Efstathiou, "Formalising the repair of schedules through knowledge acquisition," in *Advances in Knowledge Acquisition*, ed, 1996, pp. 306-320.

[9] B. C. Williams and R. J. Ragno, "Conflict-directed A* and Its Role in Model-based Embedded Systems," *Discrete Applied Mathematics,* vol. 155, pp. 1562-1595, 2006.

[10] I.-h. Shu, *et al.*, "Enabling Fast Flexible Planning Through Incremental Temporal Reasoning with Conflict Extraction," in *the 15th International Conference on Automated Planning and Scheduling (ICAPS 05)*, Monterey, CA, USA, 2005, pp. 252-261.

[11] J. Kletti, *Manufacturing Execution System - MES*: Springer, 2007.

[12] J. T. Lee, *et al.*, "Mininum adjustment for repairing an initial solution in reactive scheduling," *Journal of KISS (B) (Software and Applications),* vol. 25, pp. 923-930, 1998.

[13] V. J. Leon, *et al.*, "Robustness Measures and Robust Scheduleing for Job Shops," *IIE Transactions,* vol. 26, pp. 32 - 43, 1994.

[14] K. Miyashita, "Case-based knowledge acquisition for schedule optimization," *Artificial Intelligence in Engineering,* vol. 9, pp. 277-287, 1995.

[15] B. C. Williams, *et al.*, "Model-based programming of intelligent embedded systems and robotic space explorers," *Proceedings of the IEEE,* vol. 91, pp. 212-237, 2003.

[16] J. E. Spragg, *et al.*, "Constraint-Based Reactive Rescheduling in a Stochastic Environment," in *the 4th European Conference on Planning: Recent Advances in AI Planning*, 1997, pp. 403-413.

[17] C.-C. Cheng and S. Smith, "Applying constraint satisfaction techniques to job shop scheduling," *Annals of Operations Research,* vol. 70, pp. 327-357, 1997.

[18] J. K. Lenstra, *et al.*, "Optimization and approximation in deterministic sequencing and scheduling: A survey," *Annals of Discrete Mathematics,* vol. 5, pp. 287-326, 1979.

[19] R. Dechter, *Constraint Processing*: Morgan Kaufmann Publishers, 2003.

[20] R. Dechter, *et al.*, "Temporal constraint networks," *Artif. Intell.,* vol. 49, pp. 61-95, 1992.

[21] S. Lawrence, "Resource Constrained Project Scheduling: An Experimental Investigation of Heuristic Scheduling Techniques," Carnegie Mellon University, Pittsburgh, Pennsylvania1984.

[22] D. Applegate and W. Cook, "A Computational Study of the Job-Shop Scheduling Problem," *INFORMS JOURNAL ON COMPUTING,* vol. 3, pp. 149-156, January 1, 1991 1991.

[23] P. Kim, *et al.*, "Executing Reactive, Model-based Programs through Graph-based Temporal Planning," in *the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI 2001)*, Seattle, Washington, USA, 2001, pp. 487-493.