Proceedings of the 1996 IEEE
International Conference on Robotics and Automation
Minneapolis, Minnesota - April 1996

Cycle Detection in Repair-based Railway Scheduling System

Te-Wei Chiang and Hai-Yen Hau

Department of Electrical Engineering
National Taiwan University
Taipei, Taiwan 10617, R.O.C.

Abstract

In this paper, we propose an approach for railway scheduling based on iterative repair, a technique that starts with a complete but possibly flawed schedule and searches through the space of possible repairs. The search is guided by an earliest-conflict-first heuristic that attempts to repair the earliest constraint violation while minimizing the value of objective function. Since cycles may exist among a sequence of repairs during the repair process, a cycle detection and resolution scheme is proposed to prevent infinite loops. Experimental results show that the efficiency of the repair algorithm improves significantly when cycle detection is incorporated.

1 Introduction

Similar to conventional job-shop scheduling problem [1], the railway scheduling problem is the decision of the arrival/departure times and the assignment of resources (tracks) to all trains at every station while minimizing a particular objective function and satisfying some specified constraints. The resources to be allocated include the tracks within a station and the tracks between two stations. For most railway systems, there are two tracks between neighboring stations, one for southbound trains and the other for northbound trains, assuming the railway system is south-north rail. Thus, in our system, only the tracks within stations need to be assigned by the scheduler. On the other hand, the decision of the arrival and departure times of all trains at each station must satisfy the specified constraints, e.g., minimum stopover time constraints, minimum headway constraints, ..., etc.

An analogy can be made between the railway scheduling and the conventional job-shop scheduling. In railway scheduling problems, we can regard the tracks as machines in a job-shop and the assignment of trains to tracks as the dispatching of jobs to machines. The objective of railway scheduling is to minimize the average running time of each train, which corresponds to minimize the average flow time of each job in job-shop scheduling.

The main difference between the railway scheduling problem and the conventional job-shop scheduling problem lies in the constraints the final schedule is subject to. There are many conditional constraints that depend on the track assignments of each train. Furthermore, for a train that stops at station A, we must assign a track with platform. For a train passing through station A non-stop, we can assign any track with or without platform to the train. Therefore, the railway scheduling problem is similar to the conventional job-shop scheduling problem with alternative machines [8]. Such differences make the problem more difficult than conventional job-shop scheduling problem and make it impossible to be solved by conventional OR techniques, so we resort to AI techniques.

There is a long history of AI programs that use repair or debugging strategies to solve problems[9], [10] In repair-based approach, one starts with a complete but possibly flawed schedule and searches through the space of possible repairs. Several attempts have been made in the past to tackle the complexities involved in the automatic generation of timetables [4], [5], [7]. However, their railway systems usually are special cases of a general railway system. Chiang and Hau proposed an iterative repair approach for railway scheduling problems [2], [3]. The main problem of the repair-based scheduling system is that cycles may exist among a sequence of repairs during the repair process and hence circumvent the evolution of the repair process. To this end, we propose a cycle detection and resolution scheme for railway scheduling problems in this paper.

2 Problem description

The ultimate goal of railway scheduling is to provide good service for passengers (e.g., minimize the waiting time for each passenger) while minimizing the operation costs. Thus, we must generate a conflict-free schedule in accordance with a *master scheduling plan* while minimizing the average running time of each train. The master scheduling plan contains, for each train, the departure time of the starting station, the destination, and the stations along the route that the train must stop. The

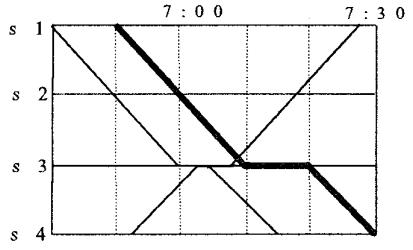

Fig. 1. A simple train diagram used for illustration.

railway scheduling process is carried out by drawing a train diagram. The train diagram indicates, for each train, the arrival time and departure time of each station along the route, and the stopover time at a station if the train makes a stop at that station. Fig. 1 shows a simple train diagram, where the vertical axis represents stations and the horizontal axis represents time. Slanted lines represent trains. The lines with positive slope represent northbound trains and that with negative slope represent southbound trains. The thick line in the diagram represents the southbound train that departs station S1 at 6:50 a.m., passes through S2, and arrives at S3 at 7:10, and stops at S3 for 10 minutes and departs S3 at 7:20. The goal of railway scheduling is to draw a train diagram for all trains and stations and determine the track assignments for each train at each station, and maximize (or minimize) certain performance criteria, subject to some physical constraints.

The physical constraints to the railway scheduling are :

1) *Running time constraints* : These constraints indicate that the shortest running time needed for trains to travel from one station to the next station is constant.
2) *Minimum stopover time constraints* : The minimum stopover time of each train at each station, which is prespecified according to the master scheduling plan.
3) *Minimum headway constraints* : Due to the safety concern, there must be a safety time separation between the consecutive arrival/departure of trains at each station.
4) *Level crossing constraints* : The time interval between two trains sharing the same crossover must satisfy a prespecified minimum time interval to prevent collision. Note that the crossovers corresponds to the fork points shown in Fig. 2.
5) *Track assignment constraints* : If two trains had been allocated to the same track in a station, then there is a minimum time interval limited to the two trains.

Corresponding to these constraints are five types (Type I - Type V) of conflicts that may arise during the scheduling process, each of which corresponds to one of the constraint type defined above. For instance, Type I conflict corresponds to the violation of running time constraint. Each conflict is associated with either one train (Type I and Type II) or two trains (Type III - Type V) and the station where the conflict happens. We can find that there are many conditional constraints (e.g. (4)-(5)). The minimum headway constraints, the level crossing constraints and the track assignment constraints are *route-dependent constraints*, i.e., whether these constraints exist or not depends on the track assignments. Such complex constraints make the problem impossible to be solved by conventional OR techniques, not to say the inherently large scale of the problem.

3 Approach

3.1 Problem formulation

Generally speaking, the goal of the railway scheduling is to minimize the average running time and the start time deviation of each train while satisfying all of the defined constraints. Mathematically, the problem can be formulated as a constrained optimization problem: the objective function is the minimization of the average total running time and the start time deviation of each train, and the constraints are that defined in section 2. We can transform the constrained optimization problem to unconstrained optimization problem via the incorporation of the constraint violations into the objective function.

Assume that $\underline{X}$ is a solution point (or a schedule) in the problem space. It can be expressed as

$$\underline{X} = \underline{x}_{ij},\ 1 \le i \le N \text{ and } 1 \le j \le M \tag{1}$$

where N is the number of trains and M is the number of stations. Each $\underline{x}_{ij}$ corresponds to an arrival time, departure time, track triplet $[a_{ij}, d_{ij}, t_{ij}]$. The objective function is

$$C(\underline{X}) = P(\underline{X}) + \lambda Q(\underline{X}) \tag{2}$$

where $P(\underline{X})$ represents the cost due to the delay of the stopover times of trains and the deviation of the departure time of the trains at their starting stations, which is our measure for schedule quality; $Q(\underline{X})$ is the cost due to the conflicts in schedule $\underline{X}$ and λ is the Lagrange multiplier used to relax the constraint violations. We usually refer the objective function $C(\underline{X})$ to the cost function of the problem. For simplicity, we call $C(\underline{X})$, $P(\underline{X})$ and $Q(\underline{X})$ the total cost, the conflict cost and the schedule cost of the schedule respectively.

The $P(\underline{X})$ can be defined as $\Sigma_{i=1}^{N} \Sigma_{j=1}^{M} p_{ij}$, where p_{ij} is the cost of operation $\underline{x}_{ij}$ and is defined as

$$p_{ij} = \alpha \left| d_{ij} - a_{ij} \right| + \beta \left| d_{ij} - d_{ij}^{0} \right| \Delta_{ij} \tag{3}$$

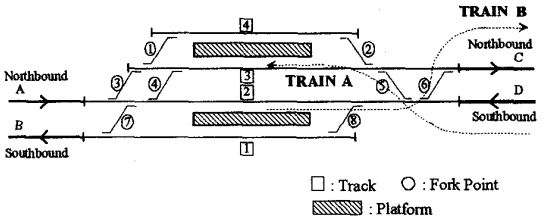

Fig. 2. A sample station configuration.

where d_{ij}^0 represents the ideal departure time of the train at its starting station, which is obtained from the master scheduling plan. Δ_{ij} is equal to 1 if x_{ij} corresponds to the operation of the associated train at its starting station and is equal to 0 otherwise. The α and β represent the weight of the delay time and the weight of the start time deviation. On the other hand, $Q(\underline{X})$ can be defined as $\Sigma_{k=1}^{K(\underline{X})} q_k$, where q_k is an positive integer representing the time interval the kth conflict is violated, assuming there are $K(\underline{X})$ conflicts in the schedule $\underline{X}$. The weight α , β and Lagrange multiplier λ are chosen to satisfy $\alpha<<\lambda$ and $\beta<<\lambda$ since a conflict-free schedule is our ultimate goal.

3.2 System architecture

The four basic components in the proposed system are : Initial Scheduler, Repair Scheduler, Local Scheduler and Conflict Management. First, an initial train diagram is established by Initial Scheduler according to the master scheduling plan. The track assignments of each train at each station are randomly assigned. The Repair Scheduler then determines the repairing sequence of conflicts after Conflict Management finds all conflicts in the initial schedule. The sequence of conflicts to be repaired in the proposed system is based on the *earliest-conflict-first* heuristic. Then the Local Scheduler will iteratively repair the conflict given by the Repair Scheduler until the overall train diagram is conflict-free.

4 Proposed algorithm

4.1 Repair methods

When a conflict arises between two trains at some station, one of the trains will be selected in an attempt to repair as much of the reduction of the *cost* function as possible. The system will try to shift the train left or right in time axis so that the resource is available, rather than exploring many possible alternatives.

There are five repair methods that can repair a conflict:

1) Change the track assignment of a train at the station.
2) Left-shift the stopover time of the violated train at the station in the train diagram such that the constraint is satisfied.
3) Right-shift the stopover time of the violated train at the station in the train diagram such that the constraint is satisfied.
4) Left-extend the stopover time of the train at the station (i.e. extend the stopover time while fixing the departure time of the train at this station).
5) Right-extend the stopover time of the train at the station (i.e. extend the stopover time while fixing the arrival time of the train at this station).

The first three repair methods are used to solve all types of conflicts except minimum stopover time violations. Among the three methods, change the track assignment has the highest priority since good track assignment can avoid some route-dependent constraint violations. Furthermore, the priority of a left-shift is higher than that of a right-shift because moving train right on time axis will cause the train to occupy the resource longer and hence affect the performance of the schedule. The last two repair methods are used only for resolving minimum stopover time violation. Similarly, the priority of the left-extend stopover time is higher than right-extend stopover time. Notice that the five repair methods only adjust the stopover time of each train, so the time required for each train running between any neighboring stations is unchanged.

4.2 Iterative repair

Initially, the initial train diagram is established according to the master scheduling plan and tracks are randomly assigned to trains. Since the initial schedule is not conflict-free, we must repair the conflicts in the initial schedule. During each iteration, we iteratively search a repair that resolves the conflict given by the *earliest-conflict-first* heuristic while minimizing the cost function. To select an appropriate repair method for a conflict, local search techniques can be used.

If a repair reduces the cost function, i.e. the new cost value, c_n , is smaller than c_o, then we accept the repair and assign c_n to c_o; otherwise we try next priority repair until all possible repairs to the conflict has been tried. If no repair method can reduce the cost function, the lowest priority repair method will be selected to repair the conflict. The main weakness of local search algorithms is that it has the tendency of stuck in a cycle. Unless the repair algorithm can identify the existence of cycles, it can run into infinite loops. Fig.3 illustrates how cycling could occur during the repair process. Fig. 3(a) shows a portion of train diagram in which there is a conflict C1(minimum time lag violation) occurred between train T1 and train T2 at station S1, as indicated by dashed circle. For the sake of simplicity, the repair methods

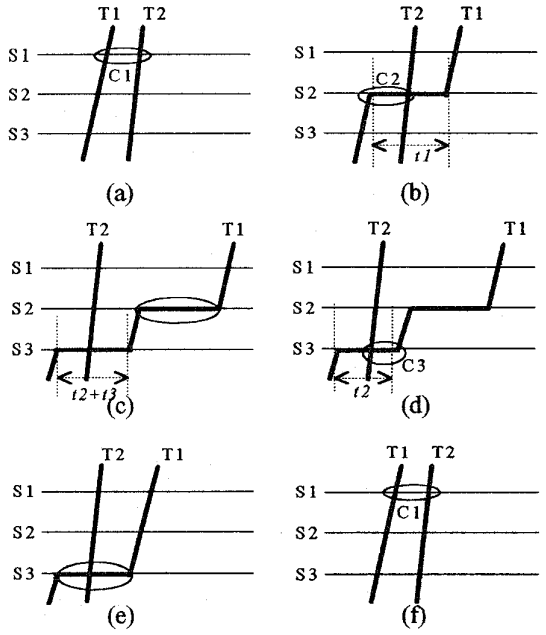

Fig. 3. Illustration of cycling

used in the example is somewhat different to aforementioned repair methods, but the concept is similar. In order to resolve the conflict, train T1 is selected to wait at station S2 (see Fig. 3(b)), but this repair introduces a new conflict C2(minimum time lag violation) (see Fig. 3(c)). So train T1 is further selected to wait at station S3, and results in conflict C3. To resolve C3, the stopover time of train T1 is extended at station S3. Although no new conflict is occurred at this time, the system finds that there is a redundant wait for train T1 at station S2 (this is because the train is originally planned to pass the station) (see Fig. 3(d)). So the system cancels the wait. At this time, the system finds another redundant wait for train T1 at station S3 (see Fig. 3(e)). To avoid the redundancy, the system further cancels the wait and hence result in a cycle (i.e., it comes back to the original schedule) as shown in Fig. 3(f).

4.3 Cycle detection

A *cycle* occurs when the schedule is unchanged after a sequence of repair operations. Since different values of objective(cost) function $C(\underline{X})$ always correspond to different schedules, we can easily verify that two schedules are not identical if they have different $C(\underline{X})$ values. On the other hand, two schedules having identical $C(\underline{X})$ values may not be identical. From above observations, we incorporated a *forbidden list* into the standard local search algorithm. We record the $C(\underline{X})$ value of each intermediate schedule as the element of forbidden list. We can conjecture that a repair may result in a cycle when the $C(\underline{X})$ value of the schedule resulting from this repair is equal to one of those stored in the forbidden list. If the $C(\underline{X})$ value of the current schedule is not found in the forbidden list, we can quickly ascertain that cycle does not exist; otherwise cycle may exist and further investigation is required. We employ another list called *action list* to memorize the information about each recent repair. From the action list, we can realize the transition of the intermediate schedule and find out whether the intermediate schedules having the same $C(\underline{X})$ values are actually identical.

In the following we define the train variables and the repair operations more precisely.

Definition 1: (*Train variables*) There are three variables associated with a train T_i and a station S_j the train stops at; $a(T_i, S_j)$, $d(T_i, S_j)$ and $t(T_i, S_j)$ represent the arrival time, departure time and track assignment of train T_i at station S_j, respectively. The total number of train variables equal to $3NM$, where N is the number of trains and M is the number of stations.

Definition 2: (*Repair operations, Characteristic pattern, Numeric pattern*) Each repair operation is represented by $F(T_i, S_j, O_k, \Delta)$. It is associated with four parameters: T_i (the train ID), S_j (the station ID), O_k (the operation type) and Δ (the amount changed), since a repair operation modifies the values of a set of train variables and the train variables depend on the train ID and the station ID (see Definition 1). The combination of the first three parameters is called the *characteristic pattern* of the operation, which identifies which variable(s) to be modified by the repair operation, and the last parameter is called the *numeric parameter* which specifies the degree of the value(s) of the variable(s) to be changed. Actually, each element of the action list records the four parameters of the associated repair.

Definition 3: (*Shift operation*) The shift operation $F(T_i, S_j, SHIFT, \Delta)$ shifts the line segment representing train T_i between station S_j and S_{j+1} in the train diagram (each line segment corresponds to the running time of the train between the two stations). The train variables to be modified are shown as follows:

$d(T_i, S_j) := d(T_i, S_j) + \Delta$ and $a(T_i, S_{j+1}) := a(T_i, S_{j+1}) + \Delta$, if the direction of train T_i is southbound.

$a(T_i, S_j) := a(T_i, S_j) + \Delta$ and $d(T_i, S_{j+1}) := d(T_i, S_{j+1}) + \Delta$, if the direction of train T_i is northbound.

Definition 4: (*Track reassignment operation*) The operation $F(T_i, S_j,$ TRACK_REASSIGN, (NewTrackId - OldTrackId)) modifies the following train variable:

$t(T_i, S_j) := t(T_i, S_j)$ + (NewTrackId-OldTrackId) (or $t(T_i, S_j)$:= NewTrackId), where OldTrackId and NewTrackId represent the index of the original assigned track

and the new assigned track, respectively.

Definition 5: *(Cycle) If the schedule is unchanged after a sequence of repairs, then we say that there is a cycle among the sequence of repairs.*

From the above definitions, we can derive the following properties to support our cycle detection method.

Property 1: *Using the two primitive repair operations defined in definition 3 and 4, different characteristic patterns refer to different variable(s) and all possible characteristic patterns cover all train variables.*

From train diagram's point of view, the schedule can be seen as a collection of slanted line segments. Each of the line segments corresponds to the running process of a train between two stations and is associated with two train variables, i.e., the departure time from the previous station and the arrival time at the next station. The difference between the two times corresponds to the running time between the two stations. Definition 3 implies that each of the line segments is characterized by a particular characteristic pattern and is controlled by the shift operations with the same characteristic patterns. The shift operation $F(T_i, S_j, SHIFT, \Delta)$ controls the position of the slanted line segment belonging to train T_i between station S_j and S_{j+1} in the train diagram. On the other hand, the track reassignment operations control all the variables referring to track assignments. Thus, the two primitive repair operations, shift operations and track reassignment operations, control all of the train variables. In conclusion, different characteristic patterns refer to different variable(s) and all possible characteristic patterns cover all train variables.

Property 2: *The schedule is unchanged, if and only if the sum of the numeric parameters of the repair operations with the same characteristic pattern is zero.*

If the schedule is unchanged then the value of each train variable must be unchanged, In other words, the operation applied to each variable has no effect. Since each variable is associated with a particular characteristic pattern, it is uniquely controlled by the repair operations with the same characteristic patterns. Thus, if the value of a variable remains unchanged then the combination of all changes to the variable must be zero, i.e., the sum of the numeric parameters of the repair operations with the characteristic pattern associated with the variable is zero.

From the above properties, we can devise an algorithm used for cycle detection. Let *ALIST* denotes the part of action list required to be examined. The *ALIST* records the parameters (see Definition 2) of the sequence of repairs between the two schedules having the same cost value. Since the above properties only hold for the two primitive repair operations defined in Definition 4 and 5, the aforementioned five repair methods have to be stored in the action list in their primitive forms. For example, the element $(S_k, T_k, Right\_Shift, t)$ can be decomposed into $(S_{k-1}, T_k, SHIFT, t)$ and $(S_k, T_k, SHIFT, t)$, where Right_Shift corresponds to the third of the five repair methods. The algorithm used for cycle detection is shown in Fig. 4. The algorithm *CD* returns TRUE if a cycle is detected and returns FALSE otherwise. If a repair method results in a cycle, we reject the repair and try another repair method. The proposed iterative repair algorithm is shown in Fig. 5, in which the following notations are used :

Algorithm *CD*
begin
  for all element (T_i, S_j, O_k, Δ) in *ALIST* **do**
    $SUM[T_i][S_j][O_k] := SUM[T_i][S_j][O_k] + \Delta$;
  end;
  for all element (T_i, S_j, O_k, Δ) in *ALIST* **do**
    if $SUM[T_i][S_j][O_k] \neq 0$
    then return FALSE;
  end;
  return TRUE;
end;

Fig. 4. The algorithm used for cycle detection.

- S_C: The set of conflicts corresponding to current schedule
- S_R : The set of repair methods
- c_o : The cost of the original schedule
- c_n : The cost of the new generated schedule
- *count* : the number of iterations
- *limit* : a prespecified number used to limit the number of iterations in the iterative repair process.

5 Experimental results

In our experiments, we set α , β and λ in eq. (2)-(3) to 1, 2 and 10 respectively. The station configuration used is the example station shown in Fig. 2. All experiments were run on a PC 80486-33. Each experiment ran until the resulting schedule was conflict-free. Since the repair functions are probabilistic, we calculated average results over five repeated trials for each experiment. In Fig.6 we graph the average cost as a function of iteration for a random generated 10-station 100-train problem solving by local search with cycle detection. The weight of conflict cost is set to 10 and the schedule range is 12 hours. The three curves in this figure represent the cost due to conflicts, the cost due to schedule performance and the total cost of the schedule (see eq. (2)) respectively. Each iteration corresponds to the process of repairing a conflict. As a result, the number of iterations

Step 1. Generate the initial schedule according to the master scheduling plan; Randomly assign tracks to train; Find all conflicts in the initial schedule and put these conflicts to S_C ; Evaluate c_n; *count* := 0.

Step 2. If S_C is empty or *count* > *limit* then stop, else select and delete the earliest conflict from S_C; Put all possible repair methods to S_R.

Step 3. Select and delete the highest priority repair method from S_R; Test to repair the selected conflict; Evaluate c_n.

Step 4. If S_R is empty, then goto step 7.

Step 5. If $c_n \geq c_o$, then goto step 3.

Step 6. If c_n is in *TLIST* and procedure ***CD*** returns TRUE, then goto step 3.

Step 7. Perform the selected repair; Update S_C; c_o := c_n; Increase *count* by one; Goto step 2.

Fig. 5. The proposed iterative repair algorithm.

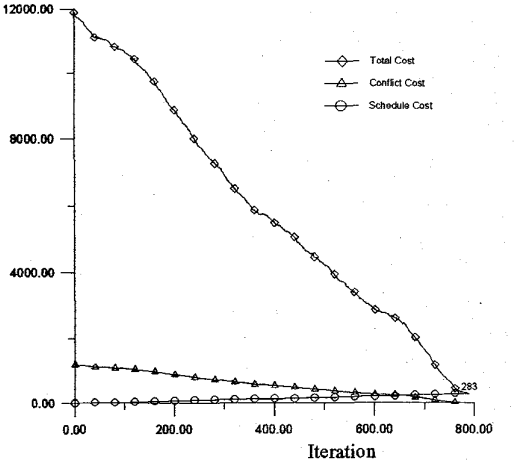

Fig.6. Results of scheduling a randomly generated 10-station 100-train problem using local search with cycle detection.

to achieve a conflict-free solution is 761.6. The average total cost is 283.

6 Conclusions

The railway scheduling problem is more difficult than conventional job-shop scheduling problems because it is inherently large-scale and there are conditional constraints, many alternative machines (tracks), and the processing time (stopover time) may be adjustable. In this paper, we demonstrated a railway scheduling system based on the *iterative repair* method. We introduced how to transform the railway scheduling problem into a repair-based search problem. Searching through the space of possible repairs, the system can quickly find a good feasible schedule without trying all possible alternatives. The proposed cycle detection and resolution scheme can help the repair process escape from infinite loops. Through the cooperation of the *earliest-conflict-first* heuristic and the local search with cycle detection, the system will result in a conflict-free schedule. In conclusion, the proposed repair-based system can resolve the large-scale railway scheduling problem in an efficient and effective manner. This approach can be applied to general scheduling problems to solve scheduling with alternative machines and dynamic scheduling problems.

References

[1] K. Baker, "Introduction to sequencing and scheduling," John Wiley & Sons, 1974.

[2] T. W. Chiang and H. Y. Hau, "Knowledge-based railway scheduling system," in *Proc. Int. Conf. on Industrial Fuzzy Control and Intelligent Systems*, Houston, Texas, pp. 42-46, 1993.

[3] T. W. Chiang and H. Y. Hau, "Railway scheduling system using repair-based approach," in *Proc. IEEE Int. Conf. on Tools with Artificial Intelligence*, Washington DC, pp. 71-78, 1995.

[4] J. E. Cury, F. A. C. Gomide, and M. J. Mendes, "A methodology for generation of optimal schedules for an underground railway system," *IEEE Trans. on Automatic Control,* vol. AC-25, no. 2, pp.217-222, April 1980.

[5] K. Fukumori, H. Sano, "Fundamental algorithm for train scheduling based on artificial intelligence," *Systems and Computers in Japan*, vol. 18, no. 3, pp. 52-63, 1987.

[6] J. Gu, "Local search for satisfiability (SAT) problem," *IEEE Trans. on Systems, Man, and Cybernetics*, vol. 23, no. 4, pp. 1108-1129, 1993.

[7] K. Komaya and T. Fukuda, "A Knowledge-based approach for railway scheduling," *in Proc. CAIA*, pp.405-411, 1991.

[8] A. Kusiak, "Intelligent manufacturing system," Prentice-Hall, 1990.

[9] S. Minton, M. D. Johnston, A. B. Philips, and P. Laird, "Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems," *Artificial Intelligence*, vol. 58, pp. 161-205, 1992.

[10] M. Zweben, E. Davis, B. Daun, and M. J. Deale, "Scheduling and rescheduling with iterative repair," *IEEE Trans. on Systems, Man, and Cybernetics*, vol. 23 , no. 6, pp. 1588-1596, 1993.