A Rescheduling Algorithm of Train-group for Railway Emergencies and its Parallelization

Zifeng Wang, Li Ruan, Limin Xiao
School of Computer Science and Engineering
Beihang University
Beijing 100191, China
wangzifeng0324@163.com, ruanli@buaa.edu.cn, xiaolm@buaa.edu.cn

***Abstract*—In order to reduce the delays of train groups when suffering from railroad emergencies, we propose a rescheduling algorithm of train groups to reschedule trains, instead of previous just waiting. A heuristic algorithm is designed to search the proper path for the trains involved in the railroad emergencies. An evaluation criterion based on the extent of the disturbance towards the original timetable and limited time is designed to select the optimal path for each train. And a conflicts resolution strategy is designed to deal with the path and time conflicts among trains when rescheduling. In the end, due to the real-time needs of the railroad scheduling, we implement the parallel processing for the train-group rescheduling referring to the allocation of loads. Experimental results show that the rescheduling algorithm is efficient to reduce the delays of train groups and the parallel processing meets the real-time response of the train-group rescheduling well.**

Keywords-train groups; rescheduling; railroad emergencies; parallel processing

I. INTRODUCTION

With the rapid development of rail transportations, the speed of trains get further enhanced and the scale of rail net get further expanded, the challenges of safety issues of train operation and scheduling are growing. Take dealing with emergencies on rail net for example, if there is a fault in somewhere of the infrastructure, or suffering from snowstorms, landslides, mudslides or other emergencies that can't be solved in a short term, trains will be delayed, which will drop the efficiency of railway transportation and result into the waste of resources and great economic losses. Although, to some extent, the artificial rechanneling can reduce the detaining and delay of trains, the efficiency is relatively low, and the rechanneling may result into a consecutive influence on the operation of trains on other railroads. So it is essential to study on safety operation and rescheduling of train groups.

Designing a reasonable rescheduling strategy can effectively figure out the above-mentioned issues. When there is a railroad emergency, the trains that are involved do not necessarily have to wait until handling of the railroad emergencies is done, just like before, they can choose another proper path to the destination with the help of the rescheduling algorithm. And there may be several proper paths for the trains to steer clear of the delays, so it is necessary to design evaluation criteria to obtain the optimal path for each train. Taking into account greater data processing caused by larger scale of rail net and greater number of trains, and effectiveness required by trains' management, we decide to use a parallel method to improve the response speed.

The rest paper is organized as follows: In section 2, we introduce the related work on safety operation and rescheduling of train groups. Section 3 presents how to design the train-group rescheduling algorithm to avoid the delays caused by railroad emergencies, including the rescheduling strategy, evaluation criteria and the conflicts resolution strategy. Section 4 presents the parallel processing of the rescheduling algorithm. Section 5 demonstrates effectiveness of the algorithm to solve the trains' delays, and efficiency to meet the real-time response.

II. RELATED WORK

Rescheduling is a time reallocation of resources for task completion when the initial scheduling cannot meet demands in completing a task for deterministic or stochastic reasons, for example, the railroad emergencies. Rescheduling is directed towards achieving specific objectives while maintaining feasibility of safety and preferential constraints. Some common scheduling objectives are cost minimization and maximization of customer satisfaction [1]. The most common rescheduling objective is to recover the target initial schedule [1].

Train-group rescheduling is a common strategy in train operation control. Many researchers design different rescheduling strategy for different special situations. C.K. Chiu [2] formulates train rescheduling as constraint satisfaction problem and describes a constraint propagation approach to tackle it. S. T. [12] presents a coordinate technique based on AI system and uses it in a highly automated train rescheduling system. R. Acuna-Agost [7] designs a MIP-based local search method for the railway rescheduling problem. Chikara Hirai [8] uses rescheduling patterns to design a new algorithm for train rescheduling. K. kumazawa [9] presents a novel train rescheduling algorithm for correcting disrupted train operation in a dense urban environment. Keisuke Sato [10] discusses rescheduling of freight train locomotives when dealing with a disrupted situation in the daily operations in Japan.

Published by Atlantis Press, Paris, France.

C.K. Chui [2] defines several optimality criteria that correspond to minimizing passenger delay, mainly about how to reduce the delay time. Tomii Norio [11] proposes an algorithm for automatically train rescheduling, and uses passengers' dissatisfaction as a criterion of rescheduling plans and focus on how to minimize passengers' dissatisfaction. Keisuke Sato [10] uses one simple speeding-up technique named set-covering relaxation to the rescheduling problem, in order to implement the real-time freight locomotive rescheduling.

Although there are many rescheduling strategies on security operation and rescheduling of train groups, most of them aim at some special situation, for example, dense urban environment [9] or congestion control [10] and so on. We take the specific situation of Chinese rail net into account, and refer to the rescheduling of different types of train groups running on the railway, not only including the freight trains, but Express trains and so on. Additionally, we implement real-time train-group rescheduling for real-time control of train groups.

III. THE RESCHEDULING ALGORITHM OF TRAIN GROUPS

A. Rescheduling Strategy

The rescheduling strategy we design is a kind of heuristic algorithm referring to heuristic search. When there is an emergency on the railway, the involved trains cannot operate successfully on the railway corresponding to the initial timetable, so another proper path have to be available for each involved train, or they have to wait and delay. So how to get the "another proper path" is the biggest difference between heuristic algorithms and traditional search, for example, the DFS [4] and BFS [5] algorithms. Heuristic search tends to choose the next search node from the current node according to a heuristic function, which can get the proper result in a shorter time, instead of blindness search of DFS [4] and BFS [5].

A-Star algorithm [3] has been demonstrated to be one of the most efficient heuristic algorithms to get the shortest path in static road network, which has been widely used on solving shortest path and strategy-designing problems. We tag *s* as the start node, *t* as the destination node, *n* as current node between *s* and *t*. The core part of A-Star algorithm is the following formula:

$$f(n) = g(n) + h(n) \tag{1}$$

$f(n)$ is the estimated cost from *s* to *t* passing *n*, $g(n)$ is the actual cost from *s* to *n*, $h(n)$ is the estimated cost from *n* to *t*. So how to design the $h(n)$ determines whether the heuristic algorithm is the A-Star algorithm. And the sufficient conditions [3] are as follows:

1) There is the optimal path from *s* to *t* in the search tree.
2) Problem domain is finite.
3) The search costs of all the nodes' child nodes are above zero.
4) $h(n) <= h^*(n)$ ($h^*(n)$ is the actual cost from *n* to *t*).

The efficiency to search for the optimal shortest path is mostly determined by the function $h(n)$. When the $g(n)$ is definite, the less the $h(n)$ is, the less the $f(n)$ will be, which can guarantee the search direction towards the destination node as fast as possible. A good evaluation [3] of $h(n)$ is that $h(n)$ is below the lower bound of $h^*(n)$, and be close to $h^*(n)$ as much as possible.

Take the actual trains' operation on the railroad into account; the shortest path between nodes generally cannot meet the actual need to be an optimal "another path" for involved trains. So we have to choose the *kth* shortest path (we tag $k=1, 2, 3 \ldots k$) instead.

We set the $h(n)$ as the length of shortest path from *s* to *n*. Then the heuristic function can be embodied to the following formula:

$$g(x) = x.len; \quad h(x) = lsp(x.n);$$
$$f(x) = g(x) + h(x) = x.len + lsp(x.n), \tag{2}$$

x represents the path from *s* to some node, which we tag *x.n*. $lsp(x.n)$ represents the length of shortest path from the node *n* to *t*; and the length of the path notes *x.len*.

We can demonstrate the formula (2) is a kind of A-Star algorithms. According to the above-mentioned sufficient condition of A-Star algorithms, the demonstration is as follows:

1) The optimal path for each train exists in the actual rail net;
2) The rail net is surely finite;
3) The actual cost for each search path is surely above zero since the weight of line is above zero;
4) $h(x)$ is one of $h^*(x)$ in itself, so of course $h(x) <= h^*(x)$

Considering the complexity of implementing and coding, we choose the Dijkstra algorithm and priority queue to obtain the $lsp(n)$. And the pseudo code is as follows in Fig. 1.

```
Input:
        vector<line*> railLine;  // Store all the lines of the rail net, line(from, to, weight)
        map<int, double> lenSP; // Store the length of shortest path for each node to the destination
        map<int, set<node*>*> pathSP; // Store the shortest path for each node
        startNode, destNode, Kth; // the start node, destination and the kth shortest path
Output:
        map<int, double> lenKSP;  // The length of the kth shortest path for each node
        map<int, set<node*>*> pathKSP; // The kth shortest path of for each node
Key variable:
        map<int, int> countKth;  // Record the times for each node popped out of the queue
        struct node {
        double fx, gx;          // fx = gx + lenSPath[x.n]
        int currentNode;        // the current Node, "x.n" in the formula (2)
        bool operator<(const node& a) { return a.fx < fx; }
        }
        vector<node*> stationNode;  // Store all the nodes
         priority_queue<node> que;       // Node: operator <()
1) Initialization:
        x.n = startNode; gx = x.len = 0;
2) Get the lenSP and pathSP;
        priority_queue<Node> que;
        DijkstarSP();  // Compute the lenSP by Dijkstra algorithm
3)Get the lenKSP and pathKSP
        que.push(Node now(lenSPath[startNode], 0, startNode));
        Node next(0, 0, 0);
        while(!que.empty())
        {
           Node now = que.top();
           que.pop();
           countKth[now.currentNode]++;
           if(countKth[destNode] == kth)    return now.fx;
           if(countKth[destNode] > kth)     continue;
           while {
                update the fx of adjacent node;
                push the adjacent node to queue;
                record the search tree and store them into the pathKSP;
             }
        }
```

Figure 1. the rescheduling algorithm of train group based on heuristic algorithm.

The initial state, $x.n = s$; $x.len = 0$; every time, we pop the x whose has the minimal fx out of the priority queue, and then get the next state according to the edges whose start node is from $x.n$, push those edges into the queue; then when we meet the $x.n==t$ for the first time, the shortest path from s to t arises, whose length is fx; similarly, when it is the *kth* time to meet $x.n==t$, we get the *kth* shortest path from s to t.

B. Evaluation Criteria

When we are searching the proper paths for the trains that are involved in the railroad emergencies, there generally are several proper paths for the involved trains to choose. So how to select the optimal path for each involved train is mostly determined by what the evaluation criterion of optimal path is. It is generally known that some common scheduling objectives are cost minimization and maximization of customer satisfaction [1], and that the most common rescheduling objective is to recover the target initial schedule [1]. So the evaluation criteria of optimal paths we design are as follows:

1) The more matching with the original timetable. The matching factor is tagged as Q.
2) The less time cost of the *kth* shortest path for each train. The general time cost is tagged as T.

When we reschedule the involved trains, there may be the conflicts of trying to meet the two above-mentioned criterions. To avoid the situation that the changeable timetable affects too much other trains' operation, we take the first criterion into account prior to the second one.

The time cost of rescheduling T mainly is the time spent on the rescheduled railroads for the train. It refers to two problems:

1) Get the *kth* shortest path.
2) The time cost for the competition of trains on the same track.

The *kth* shortest path problem can be realized using above-mentioned heuristic algorithm; as to the second problem, we can compute it according to the following conflicts resolution strategy.

We tag T as the total time cost of rescheduling for each train, and ET as the time cost that the handling of the railroad emergencies spend, Q as the factor of matching with the original timetable, we can simply set the Q as the number of stations of *kth* shortest path for each train. The pseudo code is as follows in Fig. 3.

```
Input:
        map<int, set<station*>> timetable; // the timetable of each train
Output:
        map<int, set<station*>> timetable; // the renewed timetable;
Key variable:
        priors[];    // the priorities of each train
1) Initialization:
        priors[] = 0;
Conflicts resolution:
        while(the number of involved stations)
        {
            check the conflicts for each station;
            compute the priors[] for each conflicted trains;
            sort the priors big to small;
            compute the waiting time for each conflicted train based on priors;
            update the timetable;
        }
```

Figure 2. the conflicts resolution strategy. How to deal with the conflicts of station and lines resources when there are several trains tend to occupy the resources.

```
Input:
        map<int, set<path*>> pathKSP; // all the kth shortest path for each train
        map<int, set<station*>> timetable; // the timetable of each train
        int emergStart, emergEnd; // the nodes of involved lines
        int ET; // the time spent in waiting to handle railroad emergencies
Output:
        map<int, set<node*>> pathKSP; // the optimal path for each involved train
Key variable:
        int startStation, endStation; // the intermediate station for each train;
        vector<path*> optimalPath; // store the optimal path for each involved train
1)Initialization:
        startStation=emergStart, endStation=emergEnd;
2)Get the optimal path for each involved train:
        while(the number of involved trains)
        {
          while(whether both startStation and endStation are the stop-station)
          {
            KspAlg(startStation, endStation, kth); // get the pathKSP based on current
                                                   // startStation and endStation;
            compute all the Q and T for each kth path;
            while (T<ET)  {
                choose the maximal Q;
                store the path in optimalPath;
            }
            update the startStation and endStation
            according to the timetable of each involved train
            from the emergStart and emergEnd to the nearest stop-station;
          }
        }
```

Figure 3. the evaluation criteria. how to choose the optimal path for each train from the proper paths, it is based on two primary criteria.

C. Conflicts Resolution Strategy

When we get the rescheduled path for the trains involved in the railroad emergencies, they may be not the optimal path for some trains because of the time and line conflicts. Because we reschedule the trains on the lines of specific Chinese rail net, there may be only one-way road between some stations, especially in some remote regions, the lines are too busy and scheduled to let too much trains pass by, no matter whether these trains are rescheduled or not. So the conflicts resolution strategy is needed to deal with the above-mentioned issues.

The conflicts resolution strategy includes two parts, first is how to check the conflict, second one is how to tackle them. For the first part, we check the rescheduled timetable; for the second part, the simple but efficient policy is to let some trains wait according to their priorities until other conflicted trains pass, which does not waste too much time. We tag the *priors* as the priorities of all kinds of trains, and the more the speed of the type of the trains is, the more their *priors* is. And the pseudo code is as follows in Fig. 2.

IV. PARALLEL PROCESSING OF RESCHEDULING ALGORITHM

In the real rail net, there are thousands of trains operating on the lines among hundreds of stations. When several lines are involved in the emergencies, for instance, snow disaster, line fault and so on, there may be hundreds of trains that have to be rescheduled. Taking into account greater data processing caused by the above-mentioned problems, and in the last papers [6], the experiments have demonstrates that the serial processing is too slow to meet the real-time needs of train groups flow simulation; we decide to use a parallel method to improve the response speed.

In order to choose the better method to implement the parallel processing of rescheduling algorithm of train groups, in Data-Level Parallelism or in Task-Level Parallelism. We take an experiment to analyze the characteristic of the rescheduling algorithm for train groups. The experiment mainly analyze the range of data set involved in the course of rescheduling train groups to check the necessity of dividing the rail net to speed up the computation of rescheduling train groups parallelly. The experimental result is shown in the Fig. 4. We choose the rail net in the Fig. 6 as the graph of rescheduling algorithm of train groups. The horizontal axis represents the number of rescheduling train groups. The vertical axis represents the vertical distance between the furthest node involved in the rescheduling and the original route of the train. Here, the vertical distance will be increase by one unit when involved in another node in the vertical direction.

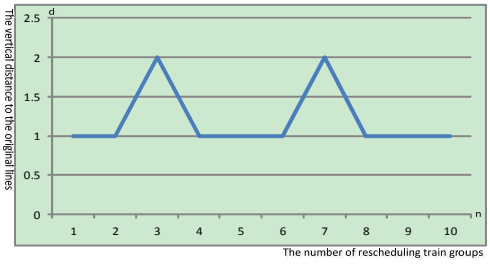

Figure 4. The furthest vertical distance to the original lines of trains every time rescheduling the involved train

According to the above experiment, we can conclude that searching the optimal routes for the trains commonly is involved in a very short range near the original lines of the trains; the furthest vertical distance is not beyond two. In other words, to some extent, it is meaningless to divide the rail net to speed up the searching the optimal routes for the involved trains.

Therefore, the parallel algorithm is mainly related to the task parallelism. The load balancing strategy is as the following processing: when a thread just completes the current computing task, and it will apply for a new task; if there is an unallocated task, the management node will allocate it to the thread. The pseudo code is as follows in Fig.4.

```
Key variable:
        list[]; // the processor lists
1) Inialization:
        list[] = 0;
While(the number of involved trains)
{
        i=0;
        request for the train rescheduling;
        while (!list[i])
        {
            allocate the rescheduling task to a free processor;
            send(data, tag); // the data needed for computing and thread tag
            i++;
        }
        when receive the computing result from the i processor;
        receive(data,tag);
        update the timetable;
        set the list[i]=0;
}
```

Figure 5. the parallel processing for rescheduling of train group. Mainly about how to allocate the load using MPI.

V. THE IMPLEMENTATION OF RESCHEDULING ALGORITHM OF TRAIN GROUPS AND ITS PARALLELIZATION

In the experiment, in order to show the process of rescheduling clearly, we choose the part of Beijing-Guangzhou lines as the experimental rail net. The rail net is shown in the following Fig.6. And the part rail net comes from the Wikipedia.org [13].

Figure 6. the experimental rail net [13]. Part of Beijing-Guangzhou lines.

And the part of timetable of all the trains is shown in the left side of Fig. 7. The horizontal axis is the time axis; the vertical axis represents the locations of all the stations.

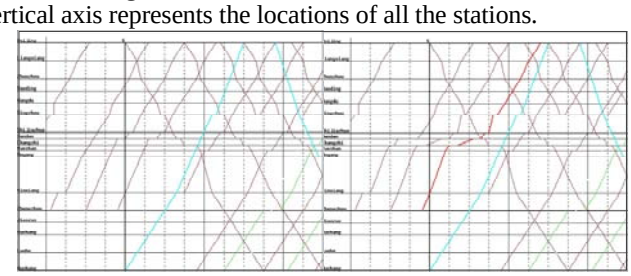

Figure 7. the timetable before and after the rescheduling. The left side is the previous timetable; the right side is the present timetable; the involved trains includes 5204\5206\5608

When there is a railroad emergency, whose time cost *ET* is two days, between Xinxiang and Handan, the trains that are involved in the line from Xinxiang to Handan either to wait or to choose another optimal path. We can find that there is a right path from Xinxiang to Handan passing the Changzhi and Yueshan, and the timetable have to be updated. The updated timetable is shown as follows in the right side of Fig. 7.

In order to show the operation of train groups clear and convenient, we simplify the timetable of the trains. Every line represents the operation of each train, and every color represents one type of train, which can help us recognize every type of train conveniently, we define 17 different types of trains according to the actual rail environment, different types of trains have different speed, which is necessary to be used in the rescheduling of train group. The types of trains are shown in the following Fig. 8.

```
[LINESTYLES]
LINES=0,64,68,72,76,208,212,216,220,224,228,232,108,
240,119,121,253
Train00=Local,Lc,80
Train01=Rapid,Rp,100
Train02=SpecialRapid,SpRp,150
Train03=NewRapid,NRp,200
Train04=CommuterRapid,CmRp,100
Train05=SemiExpress,SmEx,120
Train06=Express,Ex,180
Train07=RapidExpress,RpEx,240
Train08=LimitedExpress,LtEx,150
Train09=RapidLimitedExpress,RLEx,220
Train10=CommuterSemiRapid,CSRp,100
Train11=CommuterExpress,CmEx,100
Train12=PartialRapid,PaRp,90
Train13=PartialExpress,PaEx,120
Train14=OutofService,OtSv,80
Train15=Freight,Fr,80
Train16=ExpressFreight,ExFr,120
```

Figure 8. the different types of trains. Different types have the different colors and different speed to recognize the trains conveniently.

The efficiency of rescheduling of train group gets lower and lower as the rail net becomes larger and the number of involved trains gets more, so we take the parallel computation into the rescheduling of train group. In the experiment, we get three computers to constitute our hardware environment, every computer has one processor. We use the MPICH2-1.2 to support our software environment. The experimental result is shown in the following Fig. 9. The horizontal axis is the scale of the rail net; and the vertical axis represents the time cost of computation.

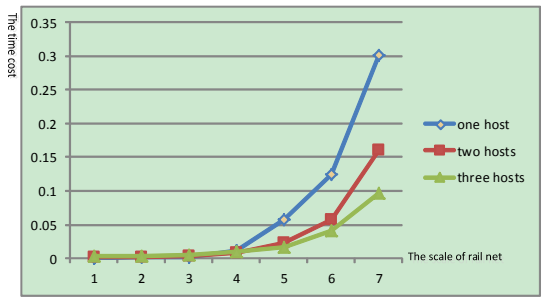

Figure 9. the graph of comparing the serial computation and parallel computation. We choose seven different groups of data to analyze the efficiency of rescheduling algorithm.

In the experiment, we collect seven different groups of data separately based on the different scales of rail net (the number of nodes/the number of edges): 6/8, 15/40, 51/100, 51/500, 51/1000, 80/2000, 80/5000, they have the same number of tasks among different scales. And we compare the time cost of serial computation to that of parallel computation. The time cost gets more and more as the scale of rail net becomes larger and larger in the figure; because of the great time cost of data transmission of MPI, it hardly change the time cost when the scale of the rail net is small; when the scale of rail net is beyond 51/500, the time cost begins to get lower as the increase of the number of hosts. Its speed-up ratio is about 3 comparing the time cost of three hosts to one host when the scale of rail net is beyond 80/2000, which is consistent with the theoretic speed-up ratio in general.

VI. Conclusion and Future Work

We proposed a new rescheduling algorithm of train group based on heuristic algorithm to deal with the time loss due to the rail road emergencies. The rescheduling strategy Based on the A-Star heuristic algorithm choose proper paths for the involved trains. An evaluation criterion based on the extent of the disturbance towards the original timetable and limited time is designed to select the optimal path for each train. And a conflict resolution strategy is designed to deal with the path and time conflicts among trains when rescheduling. In the end, Due to the real-time needs of the railroad scheduling, we implement the parallelization for the train-group rescheduling referring to the division of rail net.

The rescheduling strategy can be optimized to improve the efficiency of the rescheduling algorithm and in the process of rescheduling; we ignore some uncontrollable factors that may impact the time cost of rescheduling. In our future work, we will continue to optimize the rescheduling algorithm to get a better efficiency on the process of search proper paths. And it is also necessary to design a new model to take into the above-mentioned uncontrollable factors to improve the accuracy of rescheduling of train groups. We will focus a lot on them in the future.

Acknowledge

This paper is Supported by the Hi-tech Research and Development Program of China (863 Program) under Grant No. 2011AA01A205.

References

[1] Sundaravalli Narayanaswami (2012) Scheduling and Rescheduling of Railway Operations: A Review and Expository Analysis. Technol. Oper. Manag (July–December 2011) 2(2):102–122

[2] C.K. Chiu (2002) A Constraint-Based Interactive Train Rescheduling Tool. Kluwer Academic Publishers. Manufactured in The Netherlands

[3] P. E. Hart, N. J. Nilsson, and B. Raphael (1968) A formal basis for the heuristic determination of minimum cost paths in graphs. IEEE Trans. Syst. Sci. and Cybernetics, SSC-4(2):100-107

[4] http://zh.wikipedia.org/wiki/DFS

[5] http://zh.wikipedia.org/wiki/BFS

[6] Z. F. Wang (2011) Distributed train-group modeling and simulaiton based on cellular automaton and multi-agent. H. Deng et al. (Eds.): AICI 2011, Part I, LNAI 7002, pp. 146–154.

[7] R. Acuna-Agost (2011) A MIP-based Local Search Method for the Railway Rescheduling Problem. NETWORKS—2011—DOI 10.1002/net 2010 Wiley Periodicals, Inc.

[8] Chikaka Hirai (2006) A new algorithm for train rescheduling using rescheduling patterns. Railway Technology Avalanche No. 14, August 10, 2006

[9] K. Kumazawa (2008) A novel train rescheduling algorithm for correcting disrupted train operations in a dense urban environment. Wit press 199.

[10] Keisuke Sato (2012) Real-time freight locomotive rescheduling and uncovered train detection during disruption. European Journal of Operational Research 221 (2012) 636−648

[11] Tomii Norio (2005) Train Rescheduling Algorithm Which Minimizes Passengers' Dissatisfaction. M. Ali and F. Esposito (Eds.): IEA/AIE 2005, LNAI 3533, pp. 829-838, 2005

[12] Setsuo Tsuruta (1999) A Coordination Technique in A Highly Automated Train Rescheduling System. 0-7803-5731-0/99/ 1999 IEEE

[13] http://zh.wikipedia.org/wiki/ChinaRailwayNetwork