Transportation Safety Research Alliance
2100 Wharton Street, Pittsburgh PA 15203

Optimizing Train Performance through Advanced Planning and Integrated Data Recording Systems

(FRA Grant Number: DTFRDV-99-G-60017)

FINAL REPORT

May, 2000

Period Covered in this project report:
June 1, 1999 – March 1, 2000

Transportation Safety Research Alliance
2100 Wharton Street, Pittsburgh PA 15203

Optimizing Train Performance through Advanced Planning and Integrated Data Recording Systems

(FRA Grant Number: DTFRDV-99-G-60017)

FINAL REPORT

May, 2000

Period Covered in this project report:
June 1, 1999 – March 1, 2000

Table of Contents

OPTIMIZING TRAFFIC PLANNER PROJECT
FRA Grant: DTFRDV-99-G-60017
FINAL REPORT

EXECUTIVE SUMMARY

This Final Report to the Federal Railroad Administration (FRA) presents an summary of the work conducted under FRA Grant, DTFRDV-99-G-60017, by Union Switch & Signal (US&S), a subcontractor of the Transportation Safety Research Alliance (TSRA), to develop an Optimizing Traffic Planner (OTP) for optimizing the movements of trains across railroad networks. The report summarizes the work done between June 1, 1999 and March 1, 2000 (time period covered by the Grant), describing the core planning engine component of the OTP system that was developed during that time period and prior to June 1, 1999.

The work supported by FRA Grant, DTFRDV-99-G-60017, constitutes the *first* phase of a three phase project to develop and install a real-time OTP for optimizing the flow of rail traffic across a railroad track network. There are currently no optimizing traffic planners in service on any freight railroads. The OTP is a planning system that will be interfaced to a Central Traffic Control (CTC) system to automate the optimized movement planning of trains. Initially the OTP will be interfaced to US&S's CTC system, but it is being designed so that it can be interfaced to any CTC system. A unique feature of the OTP is its state-of-the-art, scalable problem-solving architecture, which enables it to optimize the movements of trains regardless of how many it plans or the size of the rail network over which they travel. It includes advanced auto-routing features for generating multiple candidate solutions (movement plans), and then recommends the most promising solution based on optimization criteria related to on-time performance, increased capacity utilization and operating cost. The specific criteria chosen depend on the business objectives of the particular railroad. The OTP will provide substantially improved operating efficiencies, typically in the form of increased capacity utilization coupled with better on-time performance, and, because it works to avoid congestion, will increase the level of safety by avoiding unsafe train configurations.

The outcome for this first phase of the project – the Alpha Demonstration Phase – is a core planning engine that generates optimized meet/pass plans given a set of train schedules, a track configuration, train characteristics, an objective function and train movement constraints. This core planning engine has been tested on a number of track configurations and train schedules obtained from two Class I American railroads. The results of testing demonstrate the capability of the planning engine to optimize train movements in a short period of time.

The planning engine is the core of the OTP, and will continue to evolve as it is fine tuned for different track configurations and traffic profiles. Phase 2 of the project will see this

first version plan generator interfaced to US&S's CTC system so that it can dynamically plan in a changing environment. This second phase will be followed by an installation phase, Phase 3, the goal of which will be to install it on a Class I railroad property and field test it. The potential benefits of optimized traffic planning for the railroads are significant and will be realized in the form of increased capacity utilization, increases in car revenue, better on-time performance and increases in average train velocity.

1. INTRODUCTION

The Optimizing Traffic Planner (OTP) is a software system that plans the movements of trains across a railroad track network. This report describes the core planning engine (Alpha Demonstration Release) of the OTP, which was the deliverable for Phase 1 of a 3-phase project. Phase 1 was completed at the end of February, 2000. This report describes the results of work done during the period between June 1, 1999 and March 1, 2000 (the Grant period) as well as work carried out on the core planning engine development prior to June 1, 1999. The outputs of the core planning engine are static plans in the form of time-distance graphs of train movements.

Phase 2 of the project, which began March 1st following the end of Phase 1, involves interfacing the core planning engine to Union Switch & Signal Inc.'s Central Traffic Control (CTC) system so that it can plan and then dynamically re-plan using live data received from the field. This will require modifying the core planning engine so that it generates *robust* schedules: those capable of absorbing some amount of delay. Phase 3 of the project will involve implementing the Phase 2 optimizing traffic planning system on a railroad, testing and fine tuning it, and placing it into revenue service.

The scope of this report covers the core planning engine of the OTP. Given a train schedule, track layout, train characteristics, operational and physical constraints, and an objective function, the core planning engine generates static meet/pass plans for a specified time period over the region it was set up to plan over. How the planner generates optimized train plans is described in more detail in the sections that follow. The report discusses the testing of the system as well as its performance and the data it outputs.

This report is organized as follows: Section 2 describes the core problem-solving architecture of the OTP. This unique organization of multiple problem-solving software *agents* acting on a "pool" of potential solutions is what makes the OTP scalable and capable of optimizing to multiple objectives. Section 3 describes the scope of Phase 1 in terms of the components of the final product at the end of Phase 3. Section 4 goes into a more detailed discussion of the core planner engine and the components of which it is composed. Section 5 very briefly describes the software implementation particulars. Section 6 and 7 discuss testing and test results. Finally, Section 8 provides a summary of the project results.

2. BACKGROUND

2.1 A_Teams Problem-solving Architecture

The core of the OTP problem-solving engine consists of multiple problem-solving algorithms, in the form of software agents, and a pool of potential train movement plans that the agents modify over time. This is referred to as the A_Teams problem solving system. Agents may consist of any problem-solving algorithm whatsoever, from simple

heuristics to more complex branch-and-bound inference engines. Typically, agents make changes to train movement plans by identifying plan characteristics that when modified optimize the combined movements of all trains in the plan.

A software agent is an algorithm or operator encapsulated by a number of control mechanisms that enable it to act *autonomously*. That is, there are no control-flow restrictions or supervisory control structures, which complicate many software programs, making them difficult to modify. Agents interact *asynchronously* (i.e., there are no data dependencies between them) via shared memories, searching through spaces of feasible solutions (in this case, train movement plans) until one is found that satisfies a given optimization criterion (or multiple criteria).

Figure 1 below is a schematic of an A_Team. Although this illustration depicts a single solution memory, three construction agents, four modification agents and one destroyer agent, in general the number of agents and memories is arbitrary.

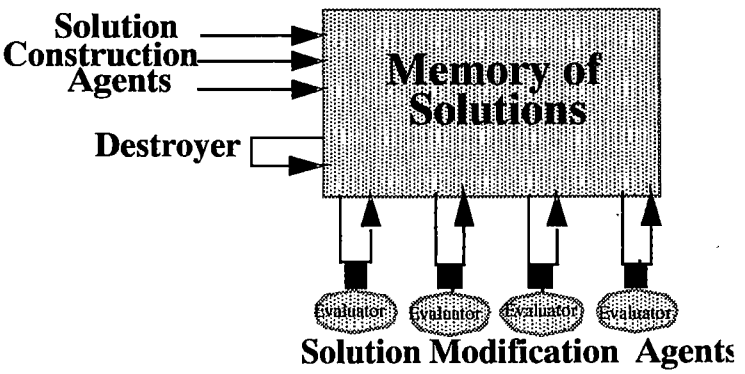

Figure 1. Schematic of an A_Team Problem-solving System

Such a system works as follows: initially, a number of construction, or seeder, agents create a pool, or memory, of candidate solutions (tactical traffic plans in the case of the OTP) that are, in general, not optimized. Modification agents, which embody algorithms that lead to better solutions with respect to specific optimization criteria, then make incremental changes to the solutions in memory. The modification agents request solutions from the memory and operate on them. Whenever a change is made, an evaluator computes the value of the solution with respect to specific optimization criteria, such as total lateness of all trains. Destroyer agents maintain a constant or nearly constant number of solutions in memory by eliminating the worst solutions or solutions that have particular (usually undesirable) characteristics. At any point in time, an external request can be made to the problem-solving system to output the best solution in its solution memory.

The construction and modification agents can be encapsulations of any problem-solving algorithm whatsoever, cooperating to produce a synergistic effect with respect to problem-solving performance, including heuristics in the form of rules, branch and bound

programs and genetic algorithms. The main idea behind the A_Teams architecture is that it enables different algorithms to work together cooperatively (known as *cooperative computation*), producing a synergistic effect with respect to problem-solving performance, very much like a group of experts who collaborate to solve a problem.

2.2 OTP Operation

Given this description of A_Teams problem solving, the following is a brief outline of how the OTP core planning engine operates.

Initially, the track database for the trackage to be planned over is read into the OTP's memory. This is followed by inputting train schedules and train characteristics. Once this data is loaded, the construction agents generate an initial set of train movement plans, which are stored in the solution pool. As noted above, these initial plans are not necessarily optimized, but are nevertheless feasible in that they do not violate any physical or operational constraints. The plans are then improved through the application of modification agents, as described above, until the system is queried for the best solution. Each solution in the pool consists of a *priority matrix* and a movement plan. Agents typically make changes to the priority matrix, which specifies the relative priorities of all trains for moving into each section of track. The priority matrix must be converted into a movement plan each time a change is made to it. Most of the agents, though, use the movement plan part of the solution to determine the changes that should be made to the priority matrix. Whenever such a change is made, a "dispatcher" uses the new priority matrix to generate a new movement plan. Hence, the OTP's problem solving can be characterized by an agent changing the priority matrix and the dispatcher converting that changed representation into a new plan, which is repeated again and again.

Using the relative priorities of trains, the dispatcher moves the trains, as it were, across the territory, producing the required time-distance plan representation through simulation. To do this, it employs a number of software modules that determine train movement (movement model), avoid deadlock (deadlock avoidance algorithm), connect routes (connectivity model) and evaluate plans (plan scorer). Most of the work during the course of this project involved development of the dispatcher and these latter modules.

Before June 1st, the framework of the A_Teams problem-solving engine, which is the core of the OTP, along with an initial set of construction, modification and destroyer agents, was implemented as well as the code to compute the value of the objective function. In addition, the interface to the US&S track database and most of the deadlock avoidance algorithm were developed. Development continued after June 1st, adding agents to the system, improving the movement model, performance testing and so on.

When the project began, a set of requirements for the OTP were laid out and a Software Requirements Specification (SRS) written to formally document them. In order to track the requirements throughout the development process, a Requirements Traceability Matrix (RTM) was created. An analysis and high-level design model were developed with the aid of Rational corporation's CASE tool, *Rose*. This was followed by

development of a detailed design model. By June 1st, a fair amount of this latter model was completed and parts of the software implemented. Development progressed according to an iterative process: design part of the system, then implement that part, repeating this two-step cycle until implementation is complete (the highest risk items are designed and implemented first).

3. SCOPE AND FUNCTIONALITY

3.1 Major Components of the OTP

This section illustrates the scope of Phase 1 by placing it in the context of the remaining components proposed for the final version of the OTP (at the end of Phase 3). A schematic of the final version of the OTP is depicted in Figure 1 below.

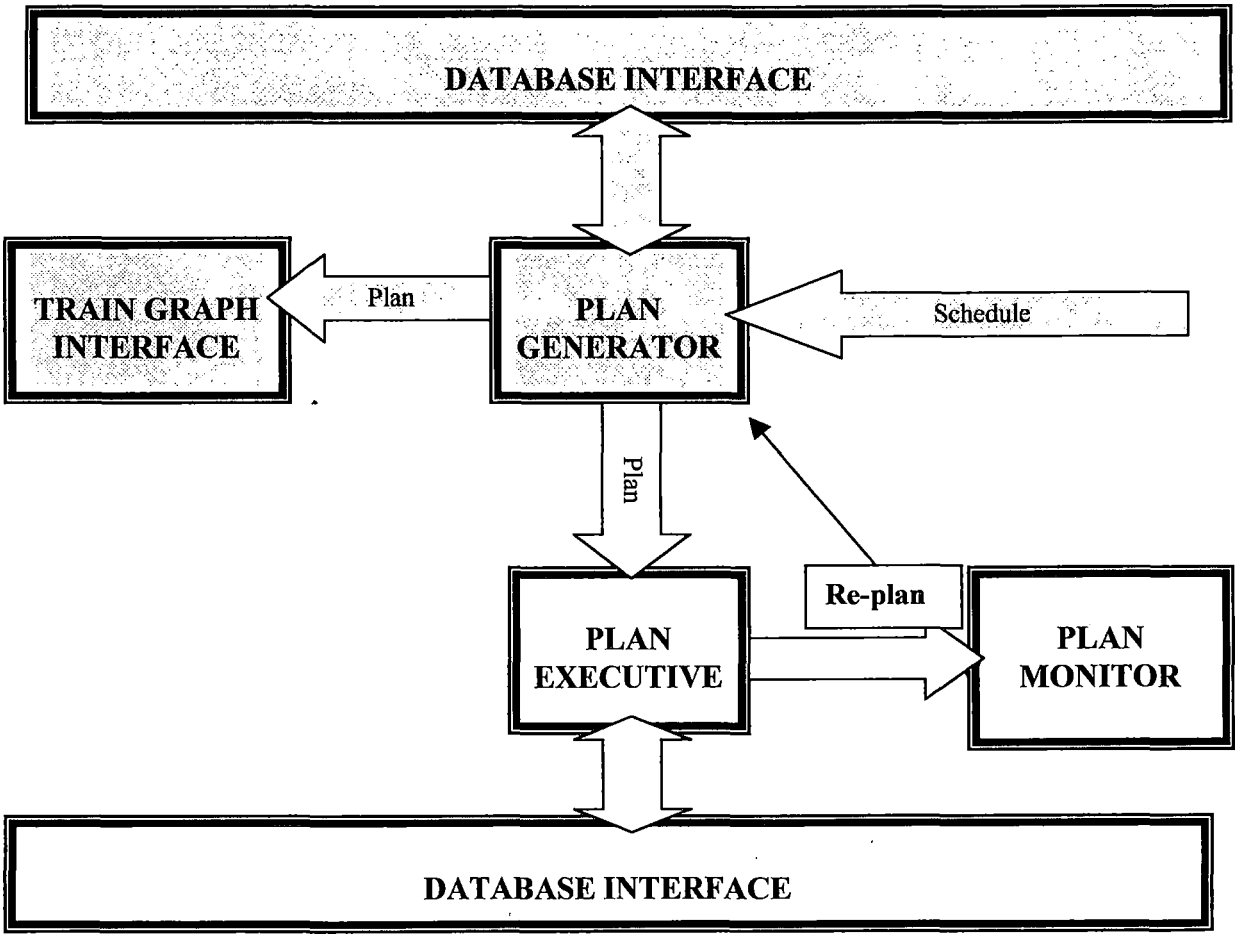

Figure 1. Optimizing Traffic Planner Components

The shaded parts in the figure are those components developed in Phase 1 (i.e., part of the Alpha Demonstration Release). The functionality for each of the five major components is briefly described:

- *Plan Generator*: the core of the OTP, and that part of the system which is the focus of this report, receives inputs about trains, schedules, track layout and so forth, and generates optimized movement plans for the region it plans over.
- *Plan Monitor*: (for dynamic planning) continually compares the state of the railroad against the movement plan that is executing in order to determine if re-planning is necessary. Re-planning may be triggered by train delays or faults in the field.
- *Plan Executive*: converts a movement plan into signal clears and other control commands so they can be executed by the CTC system.
- *Database Interface*: the plan generator employs various representations of the railroad planning problem that help expedite the production of optimized plans. In order to be compatible with the CAD system, these representations must interface to the CTC system database, which is where the OTP gets information about the state of the railroad.
- *Train Graph Interface*: the OTP will output plans to a human-machine interface called a train graph, which is a time-distance representation of the planned movements of scheduled trains.

3.2 Scope of Phase 1

The core planning engine developed in this phase includes a first version of the plan generator component and an interface to the CTC system database. There is also a string-line graphics output, but it is only a temporary version. The final system will replace this version with the TrainGraph process running on US&S's CTC system. The main components of plan generation include the railroad infrastructure to be planned over (stored in the database), a schedule, movement plan, plan scorer, A_Teams engine, agents, dispatcher, and movement model. Each of these plan generation components will be described in the next section.

The primary output of the planner is the string-line diagram or time-distance/location graph. This is just a Gantt Chart that represents the movements and planned movements of trains over a given territory as sloped lines (the values of the slopes are the velocities of the trains). Such diagrams also show where trains are stopped and the points where two trains meet and pass.

4. ARCHITECTURE AND DESIGN

4.1 Primary Components of the Plan Generator

This section describes the components that constitute the core plan generator. It accepts as inputs train schedules, train properties and track descriptions for a region of a rail network and then constructs optimized, detailed movement, or meet/pass plans for trains

over a specified time interval (e.g., 24 hours). It generates *feasible* plans (feasible plans do not violate any contraints) based on *static* data for the trackage planned over. Constructor agents, a destroyer agent and modification agents provide its problem-solving capabilities. Given a feasible schedule, an operationally feasible movement plan is generated. Currently, the plans are scored by adding the total minutes late for all trains, but this can be changed. When queried, it will output the best plan in the form of a string line diagram.

Figure 2 below is a diagram of the components and their interactions. In the figure, a loop structure (solid arrows) represents the primary problem-solving cycle that the Plan Generator executes in order to generate optimized plans. Before this problem-solving cycle begins, a number of Construction Agents generate an initial set of solutions in the form of priority matrices containing priorities for each train moving into each of the sections of track being planned over. These *internal* solutions are sent to the Dispatcher, which converts them into plans, after which they are scored and then placed in the solution pool (the priority matrix and the movement plan together constitute a solution, as noted earlier). This initial solution set is then improved by Modification Agents. Each time a modification agent selects and modifies a solution from the solution pool, it only modifies the priority matrix, but it does so based on information in the corresponding plan. The modified priority matrix is then submitted to the Dispatcher, which converts it to a new plan using constraint information, such as the Schedule and Connectivity Model, and simulation tools, such as the train Movement Model. The plan (along with its priority matrix) is then sent to the Scoring Mechanism so that it can be scored, after which it is placed back in the solution pool. To improve the population of solutions in the solution pool and to maintain a constant number of solutions, Destroyer Agents remove the solutions with the lowest scores (i.e., the least fit members of the population). Thus, plan generation continues until the best plan is requested from the system.

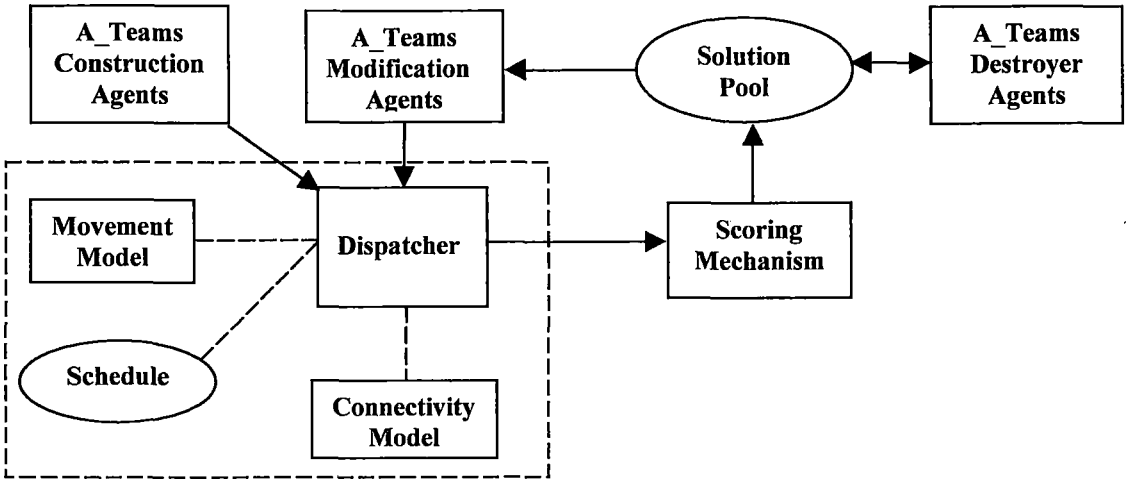

Figure 2. Plan Generation

Descriptions of the components shown in Figure 2 are described in the next several sections.

4.2 A_Teams Agents

The Plan Generator consists of 19 Modification Agents. They include a crossover agent (genetic algorithm), four bottleneck improver agents, three useless siding agents, four early arrival remover agents, a route divider agent, four critical path improver agents, a prioritize-by-schedule agent, a pass constructor agentand several others (some of the agents, particularly the critical path/bottleneck improver agents are actually implemented as different functions in a multi-functional algorithm). In addition to these Modification Agents, there is a Construction Agent and a Destroyer Agent. All the agents are described in Appendix 1. How they work is discussed in Section 2.1. The frequency of application of the Modification Agents depends on a probability table that can be modified according to the relative frequencies of applications for the different agents.

4.3 Connectivity Model

The basis for the rail connectivity model of the OTP is Union Switch & Signal's *Support Database*. The physical devices that make up the Rail Network are defined in this database. The OTP connectivity model inherits from this basic framework to build its own model of the railroad. From the connection of tracks, switches and signals in the Support Database, the connectivity model builds a track infrastructure configuration that depends on authority units (the smallest portion of the rail network that may be dispatched), interchanges (interlockings where alternate routing options may be employed) and segments (areas delimited by the same 'from' and 'to' interchanges, which contains a list of authority units that originate at the 'from' interchange and terminate at the 'to' interchange).

The Rail Network database provides a model of the physical network required by the planner. It extracts the needed static and dynamic information from the rest of the CTC system. The network, which consists of representations of all the objects found in the field (tracks, signals, switches, etc.), is stored in a data file that is read by the OTP when it starts up. The objects represented in the file are converted to software objects, the states of which are changed to reflect changes in the field (state changes, of course, will only be utilized by the OTP when it is interfaced to the CTC system in Phase 2 of the overall project).

The rail network representation is partitioned into three levels: the connectivity level, the control level, and the feedback level.

a. *The connectivity level*: represents the physical track sections and their interconnections. It is used to determine *where* trains can move. Track sections may contain geometric information such as length
b. *The control level*: is used to determine *how* trains will move across a section of track. It depends on the particular means of controlling train movements. There are two

main control mechanisms: Central Traffic Control (CTC) and dark territory control (train orders, warrant territory, etc).

1. In CTC (signaled) territory, the means for giving permission to a train to enter a track section is by a signal lamp at the entrance to that section. The lamp may be represented by an icon on a human dispatcher's computer screen, which indicates its particular state (i.e., red, yellow or green). Lamps have a face and hence they give permission to move only from one direction. The directed nature implies directed permissions (i.e. it is okay to proceed beyond this lamp). Lamps are typically red and turn green or "amber" only to allow a movement. Conceptually they give permission to move from one lamp to the next lamp facing the oncoming train. The directed path or part of a track section over which permission is given is called a route. Routes are known to failsafe equipment in the field that control the signal lamps and interlockings. Entering a route that conflicts with (fouls) an already lined route will be rejected by the field. The rules that determine when the route will be prevented from being lined are called the interlocking rules.
2. In dark territory, trains are sent a piece of paper or given verbal instructions that allow them to proceed through a sequence of track sections between two landmarks (e.g., mileposts). The landmarks are associated with the track section.

Each control method has its own preferred position feedback system. These are track circuit feedback in CTC territory and voice exchange in dark territory. In CTC territory, a train's position is indicated by a completed track circuit signal that is communicated first to a local collection site (called a control point) and then to a dispatch office of the railroad. Sections of the physical steel are electrically isolated from one another and, hence, *some degree* of local isolation of the trains is achieved. Thus, the degree to which a train's position is known depends on the length of track that makes up a single track circuit. All the steel electrically connected is called a track circuit or track for short. Associating isolators with a length along a track circuit allows all the track circuits to be determined. Track circuits interface with one another at these isolator locations.

In dark territory, the train typically has its position communicated back to the railroad's dispatching office via radio. It is usually sent as a voice message that uses local landmarks to identify train position.

In addition to train position, the CTC system is also sent states of devices in the field. The most common devices are switches and signals. Switches have three distinct states: normal, reverse, or not detected either normal or reverse (also called free). Occasionally, two switches are tied together and must be controlled as a group. Signals indicate the state of a lamp. There can be many states including green, go slow and red. Other equipment such as temperature gauges, flood detectors and so forth may also have their states indicated to the CTC system.

For the OTP, the most important level is the connectivity level, from which the other levels can be accessed as needed. The relevant database objects constituent of this level are as follows.

1. *PlanningRegion* is a collection of elements of a railroad network that are considered together for the purpose of constructing movement plans. A planning region may be disconnected; that is, there may be two or more groupings of elements where there is no path from elements of one group to elements of the other group.
2. *Authority Unit* is the smallest controllable entity in the railroad network for which it is possible to grant permission for a train to enter. The amount of track this corresponds to depends on the traffic control system. While in an authority unit, the train may have to follow some movement speed restrictions. Authority units are directed. Thus a single piece of track typically has at least two authority units on it; one for traffic in each direction.
3. *Interchange* is a section of a railroad network where trains can switch from line to line or track to track. They are delimited by a set of *InterchangeLimits* (see below) according to the following rule: Given a signal, P, and some route, R, if R's destination signal is P's partner, then R's source signal is in the same interchange as P. Another connectivity-level object, *Segment*, which is defined as a collection of *AuthorityUnits* that connect adjacent *Interchanges,* is also used by the Plan Generator.
4. *InterchangeLimit* (IL) models the elements that bound a switching area on a railroad (e.g. signals). An IL's partner is the first signal encountered on the line in the reverse direction of the IL, and granting access in that direction; i.e., the partner of a left facing signal is the first right facing signal on the line to the right of the first signal. This places partners *between* interchanges, *not* across, as one might initially guess.

Theory Of Operation
The space domain used by the OTP is provided by *PlanningRegion*, which represents a region of the track network. *PlanningRegion* has pointers to neighboring regions for continuity. Within a planning region, *AuthorityUnit* is the primary interface to the track network and its current state. *AuthorityUnits* can be grouped to form a *Segment* or a *Route*. Three or more *Segments* are connected by an *Interchange*. An *Interchange* is delimited by *InterchangeLimits*.

For the purposes of planning, other objects besides those listed above are employed, including *Location*, which specifies a place where a train can be stopped, *Via,* which specifies a *Location* where a train stops to perform an activity (e.g., setout or pickup), and *ByWay*, which is a location where a *Via* can be defined.

4.4 Schedule

The schedule component represents the traffic demands being placed on the railroad network. It is composed of a set of requirements on the travel of individual trains. Each train in a schedule (known as a *Service*) is required to enter the railroad at a certain point

and time, travel across the rail network according to its trip plan, possibly making stops, and then leave the railroad at another specific place and time.

Theory Of Operation
A *Schedule* is composed of an unordered set of *Services*. Each *Service* represents the travel of a single train from the time the train enters the planning region until the train leaves the planning region.

Each *Service* contains an ordered sequence of *WorkSets* to be accomplished in the order given. Each *WorkSet* is a collection of *WorkUnits* to be accomplished at a given *Location*. *WorkUnit* types are things like car setout or pickup, crew-change, add a helper engine, remove a helper engine, etc. Each *WorkUnit* also specifies *TimeConstraints*, which indicate when the work should be performed. The first *WorkSet* represents the start of the service, while the last *WorkSet* represents the end of the service.

Each *Service* contains an ordered sequence of *Legs*. A *Leg* represents the travel required between a pair of *WorkSets* that are adjacent in the sequence of *WorkSets* in the *Service*. Each *Leg* contains a collection of *ByWays* known as a *ByWayStore*. The store is a subset of all possible *ByWays* that might appear on any of the possible paths from the starting *Location* to the ending *Location* of a given *Leg*.

Schedule elements are served to the Plan Generator as they are known. The Plan Generator will be notified (through an interface to the CTC system, which will be completed in Phase 2 of the project) as trains are added to, removed from, or modified in the schedule. At any instant in time, there will be a number of trains on the railroad that can be identified as the trains in the schedule. There may also be unscheduled trains on the railroad. The Plan Generator must also plan those trains from their current *Locations* to their final *Locations*.

4.5 Movement Model

The Dispatcher must estimate the duration of travel for the trains as they move. It is the Movement Model component of the Plan Generator that determines the duration and speed profile of a train's movement between two points on the railroad network. Such a calculation depends on restrictions that apply at any point in time, like grade and curvature information, as well as train characteristics. The static information needed, such as civil speed limits and terrain information, exists in the track database. Other dynamic information, such as certain train characteristics and train ahead identification, is provided by other components of the CTC system through the Support Database. Additionally, device behaviors such as switch position change and signal state change are also accounted for.

All of this information is used to calculate a maximum speed profile that the train needs to satisfy. Terrain and train characteristics affect acceleration and deceleration rates. The tractive and resistive forces are computed based on available information about locomotives, load and length of the train, terrain information, and so on. Additionally,

operating style for manual locomotives and control parameters for automated cabs may affect the movement dynamics. Because the required information is not always readily available, the Movement Model should gracefully deal with incomplete data. For example, when the terrain information is incomplete, the acceleration and deceleration are approximated. Finally, it should be mentioned that safe braking distance models play an important role in the determination of simulated speed and deceleration rates.

A simulated speed profile over a particular segment of track results after applying acceleration and deceleration rates to the previously computed maximum speed profile. This *discrete* profile is used to determine the duration of the movement that is returned to the Dispatcher, so that it can be accounted for in the movement plan.

4.6 Dispatcher

The Dispatcher is responsible for creating the reservations that consititute a given plan. The plan contains information about the trains and their desired destinations. Trains are dispatched one *Leg* (one or more sections of track) at a time based on the priority of *Leg* destinations. The priorities of a train's destinations are set by the A_Teams problem-solving agents. The agents set the priorities in the Priority Matrix in order to influence the dispatching process.

During the dispatching of a train, the dispatcher reserves the routes that will allow the train to move to its desired destination along a preferred path. If there is a *Route* that cannot be reserved, the Dispatcher attempts to find another route that will allow the train to continue towards its destination.

When there are no other routes, the dispatcher determines if the train could wait at its last reservation for the next route to clear. If waiting is possible, the Dispatcher will extend the last reservation and continue reserving the path with the consequent delay. If waiting is not possible, the last reservation is deleted and the train is "virtually" backed up one route and the process is repeated by looking for an alternative and waiting. This will accomplish a meet/pass for two opposing trains as required. The process will continue until the train either reaches its destination or the start of the leg that is being dispatched.

If backing up continues to the beginning of the path, the Dispatcher will try an alternative if one exists. If no other paths exist for that train, it is placed in a holding pool for later consideration. Dispatching then continues with the next train. After successfully dispatching the next train, any trains that were put on hold are reintroduced so that dispatching of those trains can begin again. This process will continue until all trains have been dispatched or the remaining trains have been put on hold. Trains remaining in the holding pool when the Dispatcher has completed will contribute a very large penalty to the score of the plan, indicating that it is not feasible, since one or more trains could not be dispatched without deadlocking the system.

This method of dispatching has an advantage over static branch and bound methods for movement plan generation in that it is able to account for time-dependent constraints like temporary slow orders, which the latter methods cannot do without a lot of difficulty.

Theory Of Operation

The Dispatcher uses simulation methods to construct detailed movement plans for a schedule. It constructs the reservations for a given Movement Plan based on the hints contained therein, along with current railroad conditions and committed reservations for trains. The Dispatcher considers physical constraints such as bridge loading limits and tunnel dimensions when selecting routes. Reservation times are computed based on the Movement Model.

The Dispatcher's goal is to route all of the *Services* past the *Vias* without creating any deadlock situations in the railroad. A train is considered to be in a deadlock situation if the only available move for the train requires that it back up. A train is also considered to be deadlocked if there remain some forward moves, but those moves will still not allow the train to reach its destination. For example two long trains may be facing each other 400 miles apart on single track, but there are no intervening sidings long enough to allow them to meet and pass.

Note that there may be situations that appear to be deadlocks to the Dispatcher, which were caused by a human dispatcher. For example, the human dispatcher may know that a siding is longer than is recorded in the track database, so that a meet/pass may occur there, while the Dispatcher determines that it cannot occur. The human dispatcher may also know that a train is required to back up into a siding for unloading purposes, rather than pulling in engine first. The OTP will never back up trains, since that is a costly and risky operation.

When the Dispatcher begins a dispatching session, the simulated railroad is initialized to the state of the real railroad. Trains are placed on the network in their most currently known locations. Any *Routes* already cleared for a given train are said to be committed and cannot be changed, since that might result in the train loosing its green light with no opportunity to stop (i.e., knocking down a signal in front of a train). Any future reservations are also recorded.

The Dispatcher then organizes the *DispatchTable*. The head of each queue in the *DispatchTable* represents the next *Via* that a given train should be routed to, with what priority, and how long the train should occupy that location (including traversal time). The Dispatcher selects the service whose next via has the highest priority for being dispatched.

The Dispatcher defers the selection of routes from a train's current location to the target *Via* to *CourseSelectionBySegment*. The Dispatcher will ask the *MovementModel* for traversal times for each of the routes on the course and attempt to acquire reservations at the indicated times. If one or more of the reservations are not available, the Dispatcher will attempt to delay the train for a short interval (determined by the size of reservation overlap in time). If the Dispatcher is unable to delay the train, another course will be requested.

The interplay between *Segments* and *AuthorityUnits* is important to understand. *Courses* are sequences of *Segments*, and *Segments* are bundles of nearly equivalent *AuthorityUnits*. Using *Segments* in *Courses* dramatically reduces the state space that needs to be searched, since nearly equivalent *AuthorityUnits* are handled as a group. The dispatcher must choose one of the *AuthorityUnits* from a *Segment* when acquiring actual reservations.

The Dispatcher makes only a few attempts to route a train before giving up. If it is unable to route a train, the train is moved to a holding pool (referred to as the "hold-out-pool"), and another train is dispatched. Whenever a any train is dispatched to another *Via*, all the trains in the *hold-out-pool* are moved back to the *DispatchTable*, and the process of dispatching those trains resumes. If all remaining trains have been moved to the *hold-out-pool*, then the Dispatcher terminates. This condition should only result from a *human*-induced deadlock.

4.6.1 Deadlock Avoidance

Deadlock avoidance by the Dispatcher has been designed to work on *general* network configurations of track, and so should be applicable to any railroad's track layout. The algorithm is used by the Dispatcher to avoid deadlocking trains as it dispatches them over the tracks. Such a mechanism is required to prevent machine induced deadlocks. A particular method for predicting deadlocks, called the Modular Switch Array Method, predicts whether the movement of a train onto a segment of track (a schedulable operation) will create an eventual deadlock. Without deadlock avoidance, it was found that most solutions generated by the OTP contained deadlocks, which is computationally prohibitive and operationally unacceptable.

Deadlock avoidance is based on a model of the connectivity and capacity of the railroad network derived from the routing and track interconnections, along with a few heuristic statements about the ability of trains to move. The model and heuristics determine whether the movement of a train into the next route will result in deadlock of the railroad network. It avoids NP-completeness by limiting the search scope when evaluating the statements to a reasonable space. The limits will result in accurate predictions for all but the most congested networks without undue computational cost.

4.7 Movement Plan

The Movement Plan contains a representation of the detailed train movement plan that satisfies the associated *Schedule*. It consists of two main parts: a set of hints to the Dispatcher (the Proto-plan), and a complete reservation list (the Plan). The latter is the *ResourceUse* structure that the Dispatcher constructs. The leaf elements of the Proto-plan are *Vias*.

The Proto-plan consists of the *ServicePlans*, *LegPlans*, and *Vias*. It is this set of objects that are created to guide the work of the Dispatcher. In particular, the *Vias* provide hints to the Dispatcher to choose a path that goes over particular parts of the railroad network.

The *Via* may also suggest that the service be delayed at that point for a short interval. The *Vias* also provide guidance to the Dispatcher on the order in which the services should be assigned resources. One way to view the Proto-plan for a particular *Service* (i.e. Train) is as a list of *Waypoints*: suggestions to the Dispatcher that the trains be routed so that they pass close to certain places on the railroad. The *Vias* contain that information, along with a time interval and a priority number.

The Plan consists of the *ResourceUse* and *Reservations*. It is the result of the efforts of the Dispatcher. It is this developed plan that contains the information that is delivered to a plan execution task (the Plan Executive in Figure 1), which translates the plan into controls to be sent to the field. It contains nearly all of the clues that an agent would use to construct plan improvements.

Reservations in the *ResourceUse* fall into 3 categories: committed, occupancy, and fouling. Committed reservations are those that have been issued to the railroad, i.e. signals have been cleared and trains are approaching the control points. The planner is not permitted to change these reservations since it is unlikely that the train operators would have time to react to the change. Occupancy reservations are issued for the *AuthorityUnits* that a train will use in a given plan. Fouling reservations occur on opposing, or overlapping *AuthorityUnits* to an *AuthorityUnit* with an occupancy reservation, since those *AuthorityUnits* will be unavailable for a period of time related to the type of overlap and the speed of the train. Fouling reservations will occur for committed reservations as well as planned occupancy reservations.

Theory Of Operation
An agent creates a Proto-plan based on an examination of any or all of the *Schedule*, the current railroad conditions, or an existing Plan, presumably with some modifications that the agent "thinks" will improve it. When the new Plan is scored, it is placed back into the holding pool.

4.8 Scoring Mechanism

Since the OTP maintains a set of Movement Plans (solutions to the problem of moving trains to a schedule) that are optimized according to specific optimization criteria, it needs a method for ranking the overall goodness of each plan. Plans with the lowest scores will be destroyed, while the better plans will be retained for modification and, if selected, execution.

The Scoring mechanism used in the OTP allows multiple evaluation metrics to be used for judging each plan. Each evaluation metric represents an aspect of a plan's goodness or badness. For example, there is a Schedule Deviation Metric, which measures the number of minutes a train has deviated from its schedule. For that particular metric, the smaller the number, the better a plan is. The scores generated by the multiple evaluation metrics are then combined to form a single score that represents a plan's overall goodness.

5. IMPLEMENTATION

The OTP is a distributed, object-oriented software system, implemented using the C++ programming language and the Standard Template Library (STL). It is compiled with Cygnus compilers on Linux, SunOS, and OSF1-Alpha platforms, making it cross-platform ready. The system is configured in a 3-tier architecture: database (Support Database), server (OTP) and client workstation (for inputs and Train Graph display output). The code is partitioned into two main class libraries along with the application code itself.

6. TEST PROCEDURE AND ENVIRONMENT

In order to validate that the plan generator was working correctly, a series of tests were set up and executed. Initially, testing was performed for debugging purposes and later to assess system performance. This section describes the test procedure, the input data and various measures and criteria for evaluation.

6.1 Test Procedure Overview

The test procedure that was used to evaluate the OTP consisted of a series of executions of a test program, *OPG,* with a variety of inputs. The inputs presented *OPG* with track descriptions from two Class I railroads, and a schedule of trains that were to run over that railroad during some time interval. These pairings of a railroad and schedule are called scenarios for test purposes. In addition to the track and the train schedules, there are train characteristics and operational constraints.

Most of the schedules used for testing covered an interval of time in the neighborhood of 24 hours, but ranging from a low of 11 hours to a high of 84 hours. The schedules contained varying numbers of trains ranging from 2 trains to 45 trains.

The track descriptions from which the data were gathered for this phase of the project consisted of a 250 mile CSX line running through Orlando, Florida, and a 240 mile section of a BNSF coal line in Nebraska.

The test program was set to run a fixed number of agent trials, after which execution was halted. For this report, and for the regular regression tests, 1010 agent executions were allowed to complete. Because some agents produce more than one plan, the total number of generated plans was usually more than 1010, typically around 1100.

6.2 Evaluating OTP Performance: The Objective Function

A score for comparing plans was computed for each generated plan by applying an *objective function,* which is a measure of the fitness of the plan. The objective function used by the Plan Generator in this phase of the project was related to on-time performance. Specifically, it allocated a penalty of 2 units for each minute of lateness and 1 unit for each minute that a train was early, 10,000 units for each missed destination, and

100,000 units for each deadlocked train. Thus, smaller scores indicate better plans. A Destroyer Agent removed plans with lower scores. Clearly, plans with missed destinations or deadlocks will most likely be removed. On average the pool of solutions improves over time, so that the overall effect of the objective function was to drive the OTP to produce the best on-time performance.

A feature of the objective function is that specific limits on its upper and lower bounds are not defined. The lower bound would be the *optimal* plan (lowest possible score), but cannot be discovered without an exhaustive search of the space of possible plans, which in most cases is computationally prohibitive. Thus, the goodness of a plan and, hence, the value of the objective function, are relative measures not absolute. In this case, the comparison would be to the performance of an automated routing system that does not try to optimize or to a human dispatcher. The upper bound, on the other hand, is arbitrarily large by inspection.

6.3 Input Formats of Test Data

There are a number of data formats for the data used by the Plan Generator. These are as follows:

A. Railroad Description Files: The railroads are described using the standard file formats employed by the US&S Computer Aided Dispatching product.
B. Schedule File: For purposes of testing, the schedules were provided in a file with a simple grammatical structure. The structure is nearly self-explanatory. Comments provided in the example schedule file in Appendix 3 describe the schedule elements.
C. Agent Configuration: The agent configuration file informs *OPG* to construct agents. Entries in the file specify what the agent type to construct and possibly some parameters to configure on the agent instance. The parameters and types of agents have been held constant for trials to date. Tuning of the agents will be considered at a later date.

6.4 Test Program Execution

OPG reads a traffic scenario and begins to generate detailed movement plans to satisfy the schedule. Each time a plan is constructed that is better than any previous plan, a string-line is constructed representing that plan. At the end of the run, a detailed movement plan is written along with a string-line diagram. While *OPG* executes, it writes updates to a log file reporting simple statistics on the operation of the current agent. This file is used to collect statistics of agent performance.

6.5 Test Results Processing

Each run of *OPG* produces a number of output files containing various kinds of information, from plan validation results to string line plot data to detailed movement plans. The files of interest are processed by a series of scripts that create the various

tables and graphs used in the reports. Clearly, in a production environment, most of the output would not be generated. The graph images were created using the *GnuPlot* utility.

7. Discussion of Results

The outputs from testing consist of two primary graphs as well as various tables that contain the values of a number of parameter calculations, which summarize a particular test run. The graphs consist of a Solution Pool Score Trend Curve and a String Line Graph. These, along with the data in the tables are described below. Appendix 2 contains a set of graphs and tables for a specific scenario (40 trains) on the BNSF coal line.

7.1 Best Plan Score Trend Graph

The Best Plan Score Trend graph shows the relationship between the current best objective function value and the number of agent applications. In other words, it is a graph of the best plan score over time, where time is measured by the number of plans generated so far. It is interesting to note that the resulting curve has a generally asymptotic character, as can be readily seen from the Best Plan Score Trend graph in Appendix 2.

The Best Plan Score Trend graph indicates that over time the goodness of the solutions in the solution pool improves. At this point, however, it is unknown what the optimal score for this BNSF scenario is for the *same* capacity and movement models used in generating the plans. Likewise, there are no *relative* measures for evaluating the goodness of the solution (comparing to the optimal score is an *absolute* measure) because there is no data about how the 40 trains with those schedules actually run. However, reviewing the detailed data, it appears that all but about five trains were on time.

The graphs also indicate that trains exiting the line at the west end (bottom of the graph) have to wait because there is only a single track, whereas the exit at the east end (top of graph) is double track. This is why there are horizontal lines at the ends of the string lines for west-bound trains (at the bottom) and not for east-bound trains.

7.2 String-Line Graph

The string-line graph is inspired by the string line displays used by railroads to depict schedules graphically, but with some notable differences. The y-axis indicates the position of the train, while the x-axis indicates the calendar time in universal time coordinates. Horizontal line segments indicate the time spent by the locomotive on a particular section of the rail network. Vertical segments indicate the entry of the locomotive onto the next segment of the rail network. Consequently, these graphs resemble Gantt-charts.

The railroad network model used by the OTP contains the notion of a place where a locomotive can be controlled to a stop by the signalling system in either direction. These places are called *ByWays*, as noted in Section 4. We further label the *ByWays* using a

topological sort on distance from the left side of the network, breaking loops to favor early placement. The position of each *ByWay* in the resulting ordering is called the *ByWay* index, and this index is used to order *Byways* on the y-axis of the string line graph.

For simple to moderately complex lines, this method works well. For some more complex regions with multiple traffic loops and a mixture of directions mapped on to right and left, this method does not work as well. In actual operation, a train graph of the planning region would be much better for human dispatchers to view. A supervisor would prefer to examine many such train-graphs taken as a summary for the entire line or region.

The set of string-line graphs in Appendix 2 correspond to those points on the Best Plan Score Trend graph where an improvement in the best plan score occurred; that is, points where there is a step down. The number of the agent application is indicated at the top of each graph. Thus, the first drop on the pool score trend graph occurs on the 31st agent application, the next one occurs on the 39th, and so on.

7.3 Agent Performance Table

This table reports on the performance of the A_Teams software problem-solving agents. The first column is a one character identifier for the agent type. The second column indicates the number of times that agent was selected to run. The third column indicates the number of times that agent's actions resulted in a new best plan. The fourth column reports the number of times the resulting score was better than the score of the Plan (or Plans, in the case of the Genetic Algorithm Agent) that the agent used to generate the new Plan. The fifth column reports the number of times the new Plan was worse than the Plan(s) it was generated from. Agents sometimes produce new Plans that have the same score as the Plan(s) from which they were generated. Thus, the number of runs (second column) may be greater than the sum of the better and worse columns (the fourth and fifth columns).

It is interesting to note that in all the trials to date, the rate of improvement is quite low for *individual* agents, but the overall performance of the system appears to be quite good. This would indicate that the core idea of cooperative computation embodied by the A_Teams is sound.

7.4 Test Results Table

This table contains collected regression test results. The first column indicates the number of trains appearing in each schedule. The second column specifies the time interval covered by the schedule, extending from the appearance of the first train to the departure of the last train out of the planning region. The third column contains the actual planned time interval. The fourth column specifies the number of agent applications. The elapsed run time in the fifth column includes only agent execution times. The sixth column reports the rate at which the agents executed.

The column labeled "Final Minutes Late" contains the sum of the minutes late arriving at the *final* destination for every train. The scores in the next two columns are the best and median scores of all scores in the pool at the end of the run. The score is computed using the formula: 100,000*(# of deadlocked trains) + 10,000*(# of missed locations) + 1*(# of minutes deviated from schedule) + 1*(# of minutes late).

The remaining columns in the table report the number of various types of agents and the schedule file used in that trial.

7.5 General Discussion

Without some base-line data obtained from the actual running of trains planned by the OTP, it is difficult to assess the performance of the plan generator. Clearly, the OTP optimizes, as indicated by the Best Plan Score Trend graph in Appendix 2, but more testing can be done with different sets of agents on different configurations of track to determine which sets work best and where. Because the set of agents is open-ended, this implies extensive testing of the OTP to duly assess its capabilities. Part of the work in Phase 2 will be to continue such testing. New combinations of agents will be tried on different configurations of track with different train mixes and schedules. As more data are gathered, the relative performances of the different agent combinations can be compared. With the implementation of Phase 2, and the capability to inject faults, a comparison with actual train runs can be made, providing quantitative data on the improvements afforded by the OTP.

Currently, with only the capability for static planning, a comparison with actual railroad performance is confounded by the fact that the state of the railroad changes over the period planned. One comparison that could be made is with results from ART, US&S's non-optimizing planner, applied to the same train scenarios, or with results from the OTP using an objective function that is modified so that higher priority trains will always go first. Still another possibility is to get an absolute measure by comparing with the results obtained from running a branch-and-bound program that is guaranteed to find the optimal solution, even if it takes a very long time to compute. This is a program that is currently being developed for such comparisons and will be utilized during Phase 2.

8. PROJECT SUMMARY AND CONCLUSIONS

The outputs at the end of Phase 1 of this Project are a designed and implemented core Plan Generator for the Optimizing Traffic Planner, with accompanying documentation. Given a set of inputs, the plan generator produces optimized movement plans for trains that are displayed as string-line diagrams. The performance of the plan generator has not been base-lined at this point because of a lack of comparison data as noted above; e.g., human dispatcher performance statistics for the lines planned over. However, given the plan generator's performance on the two test lines, the system appears to be generating good plans, with all but a few trains arriving on time.

Testing of the plan generator will continue following the end of this project. Because it is a multi-agent distributed problem-solving architecture, there is an arbitrarily large number of system configurations – combinations of different agents, probabilities of agents making changes, number of iterations, etc. – that can be tried in order to fine tune the system. Moreover, different parts of a railroad will likely demand different configurations for best performance. Thus, testing of the OTP is open-ended in this regard. However, because of the modularity of the problem-solving architecture, there is almost no effort involved in reconfiguring the system making testing different configurations straight forward.

Consequently, testing of the plan generator will continue into the second phase of the project. Phase 2 will also see additional testing of agents and agent configurations capable of generating robust schedules and re-planning because the OTP will be planning in a dynamic environment.

This final report describes the state of the OTP at the end of Phase 1. The specific tasks that were accomplished during the time period covered by the FRA grant are summarized in the attached interim reports (Appendix 4).

APPENDIX 1

Agent Descriptions

In the following table, the agents are either modifier, constructor, or destroyer agents. Initializers create new plans from schedules. Modifier agents function by examining one or more existing plans for potential improvements, possibly combining one or more plans, and creating a new plan based on the discovered opportunities. Destroyer agents remove the lower quality plans from the pool. When plans are selected, the normal mechanism is to select the plans randomly, with the selection weighted by fitness.

Agent Descriptions

<table>
<tr><th>Code(s)</th><th>Agent Name, description</th></tr>
<tr><td>BbCc<>^%</td><td>CPMImprover</td></tr>
<tr><td></td><td>This agent will analyze the Critical Path Method resource allocation graph with the events in the graph representing the start of reservations. There are two types of reservations in the graph; those representing the presence of a train, or those representing the unavailability of one or more routes due to the presence of a train on some near by route. The activities represent the transit of a locomotive from reservation starting location to reservation starting location, or the corresponding start and end of unavailability of a route due to the presence of a train on a near by route.
The agent looks for two types of features in the graph, and makes one of 4 types of repairs, depending on the configuration. The two features are the longest critical path sequence in the graph, and the event with the most items depending on its completion (a bottle neck). The types of repairs are to lower or raise the priority of a suitable event in the feature, or to choose another routing that does not use the same resource as the event with either lower or higher priority.
<table>
<tr><td></td><td>Critical Path</td><td>Bottleneck</td></tr>
<tr><td>Lower</td><td>c</td><td>b</td></tr>
<tr><td>Raise</td><td>C</td><td>B</td></tr>
<tr><td>Relocate and Lower</td><td><</td><td>^</td></tr>
<tr><td>Relocate and Raise</td><td>></td><td>%</td></tr>
</table>
</td></tr>
<tr><td>x</td><td>CrossoverAgent</td></tr>
<tr><td></td><td>The Crossover agent carries out one step of a genetic algorithm, specifically selecting two existing plans, choosing two places to combine them and combining the plans. The resulting new plan is then scored and placed in the pool.</td></tr>
<tr><td>d</td><td>DividingAgent</td></tr>
<tr><td></td><td>The dividing agent selects a plan, selects a spot in the plan and inserts a randomly selected via at that point in the new plan. The via is selected so that the plan is still plausible</td></tr>
</table>

!	*IPCByViaExitAgent*
	This agent creates new plans and chooses priorities for the destinations according to the exit time from each destination.
$	*IPCOverkill*
	This agent creates plans by inserting a via at every reasonable routing point along the preferred path. *This agent is obsolete*
s	*IPCSortedAgent*
	This agent constructs new plans by sorting the destinations in the schedule by desired arrival time and assigns priority accordingly.
u	*IPCUniformAgent*
	This agent constructs new plans by assigning the same priority to every destination.
$Vv	*IPCViaCreationBySegmentAgent*
	This agent creates new plans by considering the preferred course between each pair of destinations. If the selection strategy is all (indicated with $), all elements of the course are selected. Other selection strategies are; select one (marked by V), or select a configurable number (marked by v) Priorities are assigned to the course elements according to one of several strategies, depending on the configuration of the agent instance. The priorities are chosen uniformly, randomly, or by exit time at each destination.
i	*InitialPlanConstructorAgent*
	This agent constructs new plans with randomly assigned priority for each destination in the schedule.
FfLl	*NotTooEarlyAgent*
	This agent looks for planned arrival at a destination early, and inserts delays to correct for that. Depending on the configuration of the agent instance, the first (Ff) or last (Ll) early destination for each train will be examined, and delays will be inserted either at the start (FL) or end (fl) of the leg. The amount of earliness that will trigger action is configurable.
Zz	*NotTooLateAgent*
	This agent looks for planned late arrival at a destination, and increases routing priorities of that service before the late point. The priorities are increased according to a calculated (Z) or random (z) factor.
P	*PassConstructorAgent*
	This agent looks for fast trains that are following slower trains, and creates a pass by stopping the slower train in a siding.
p	*PrioritizeAgent*
	This agent raises the routing priorities of trains that performed poorly.
r	*PrioritizeByViaExitAgent*
	This agent adjusts priorities so that earlier destinations are processed with a higher priority than later destinations.
Yye	*UselessSidingRemover*

	This agent looks for routing into a siding where there is no apparent reason for the move, and removes the siding move. This situation may occur when a meet or pass was constructed by one agent, another agent then re-routed, delayed or otherwise changed the timing of one of the trains, leaving one train in a siding for no apparent reason. The agent will repair all such sidings (Y), sidings for late trains only (y), or randomly selected sidings (e).

APPENDIX 2

Plan Generation

Figure 3 shows the best plan score trend for a train schedule scenario on the BNSF 240 mile territory, which was one of the data sets used to test the OTP. The diagram shows the size and frequency of the changes, which together resemble an exponential decay, assymptoting toward some final value. The number of agent applications to reach the best score is approximately 1300. As can be seen from the graph, almost 85% of the total improvement is obtained by the 315th agent application, where it reaches a value of approximately 1525.

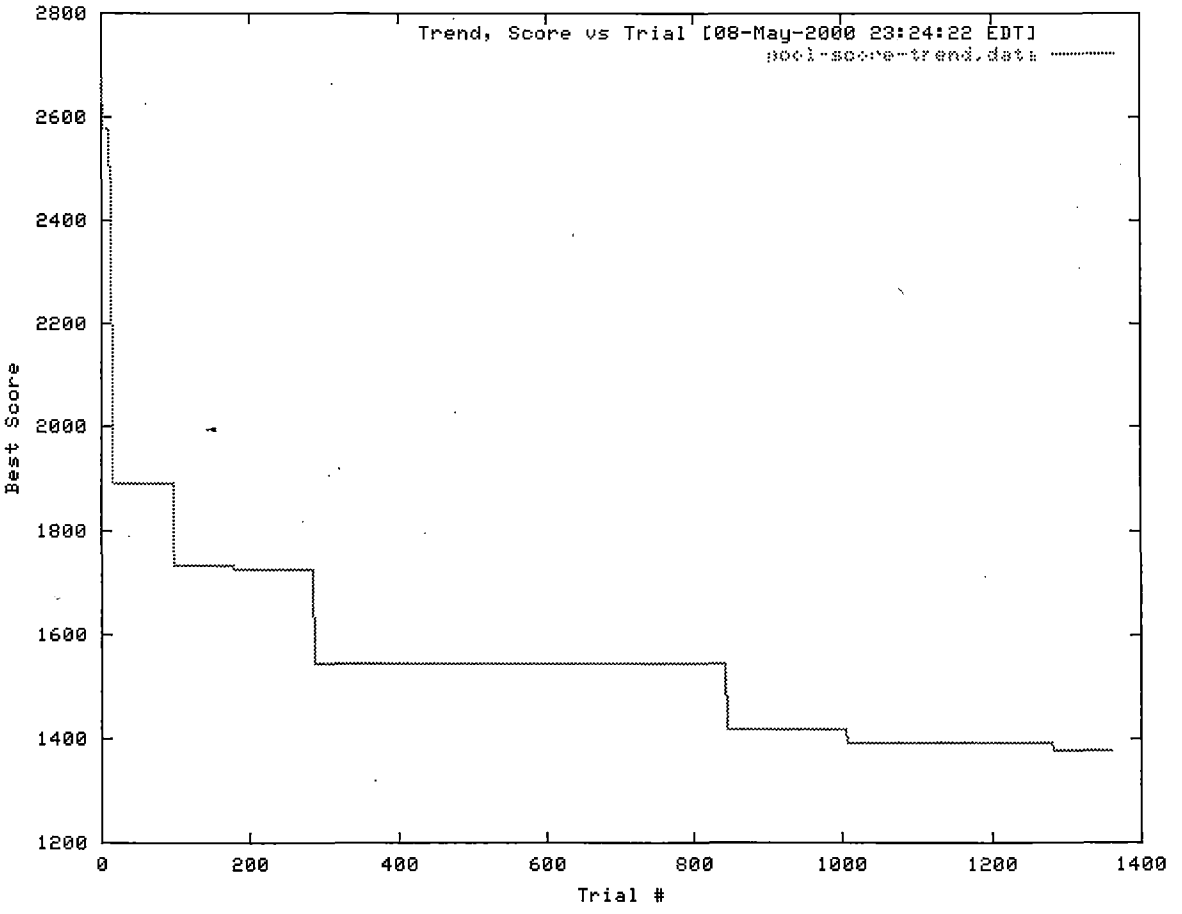

Figure 3. Best Plan Score Trend

The best score is the sum of early and late arrival times at every scheduled stop for all trains: 1 point for each minute early and 2 points for each minute late. Obviously, the best score has the lowest number of points. From the values of the scores shown, it is clear that there are no deadlocks or missed locations.

The following graphs are string-line diagrams depicting train plans that correspond to the best scores in Figure 3 above. For each direction of travel, there is a different color representing each train. String-lines from the upper left corner toward the lower right represent trains traveling in one direction and string lines from the lower left to the upper right represent trains traveling in the other direction. Each graph shows the number of the agent application at the top, which corresponds to an improvement step in Figure 3.

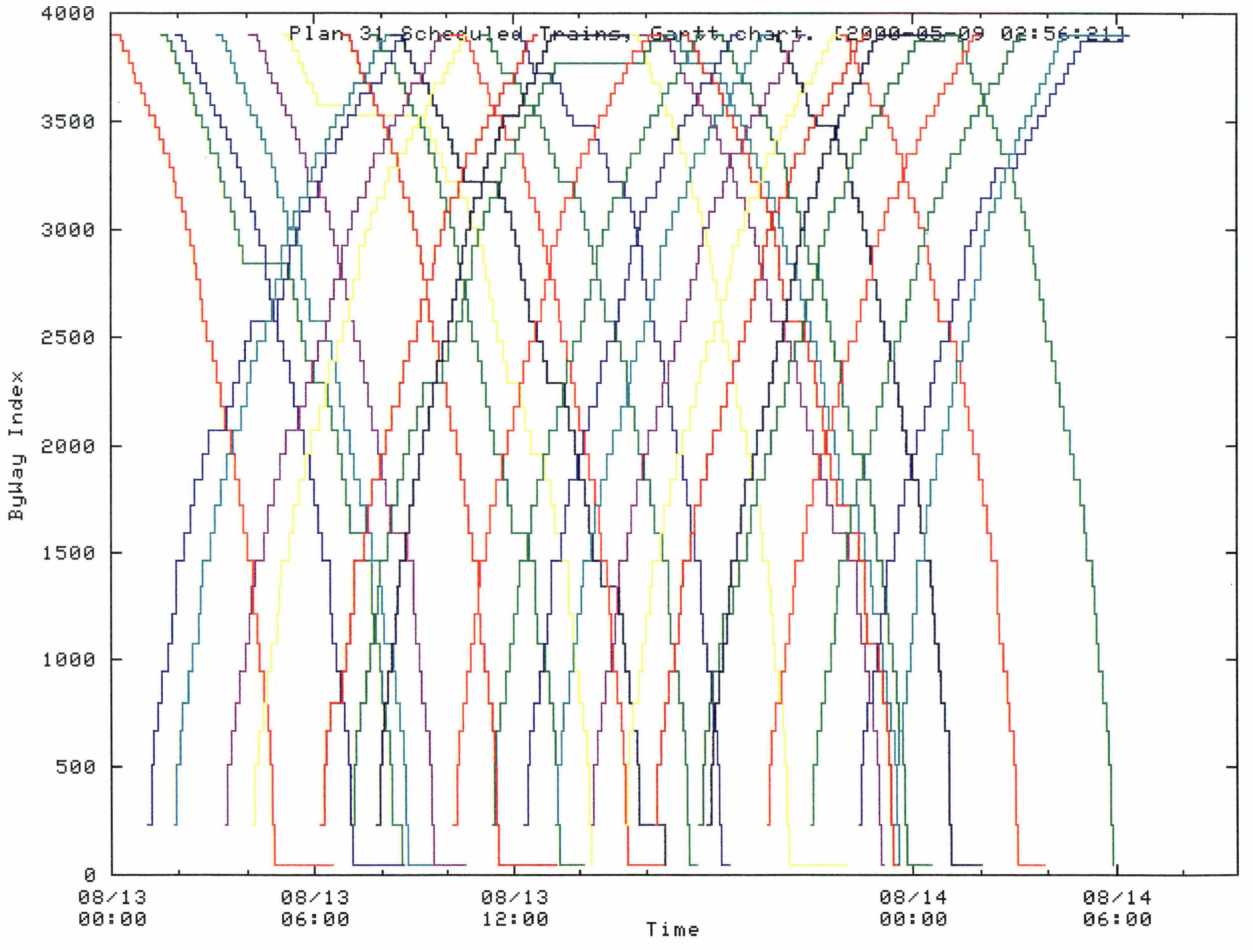

Figure 4: Plan Produced by Agent Application #31

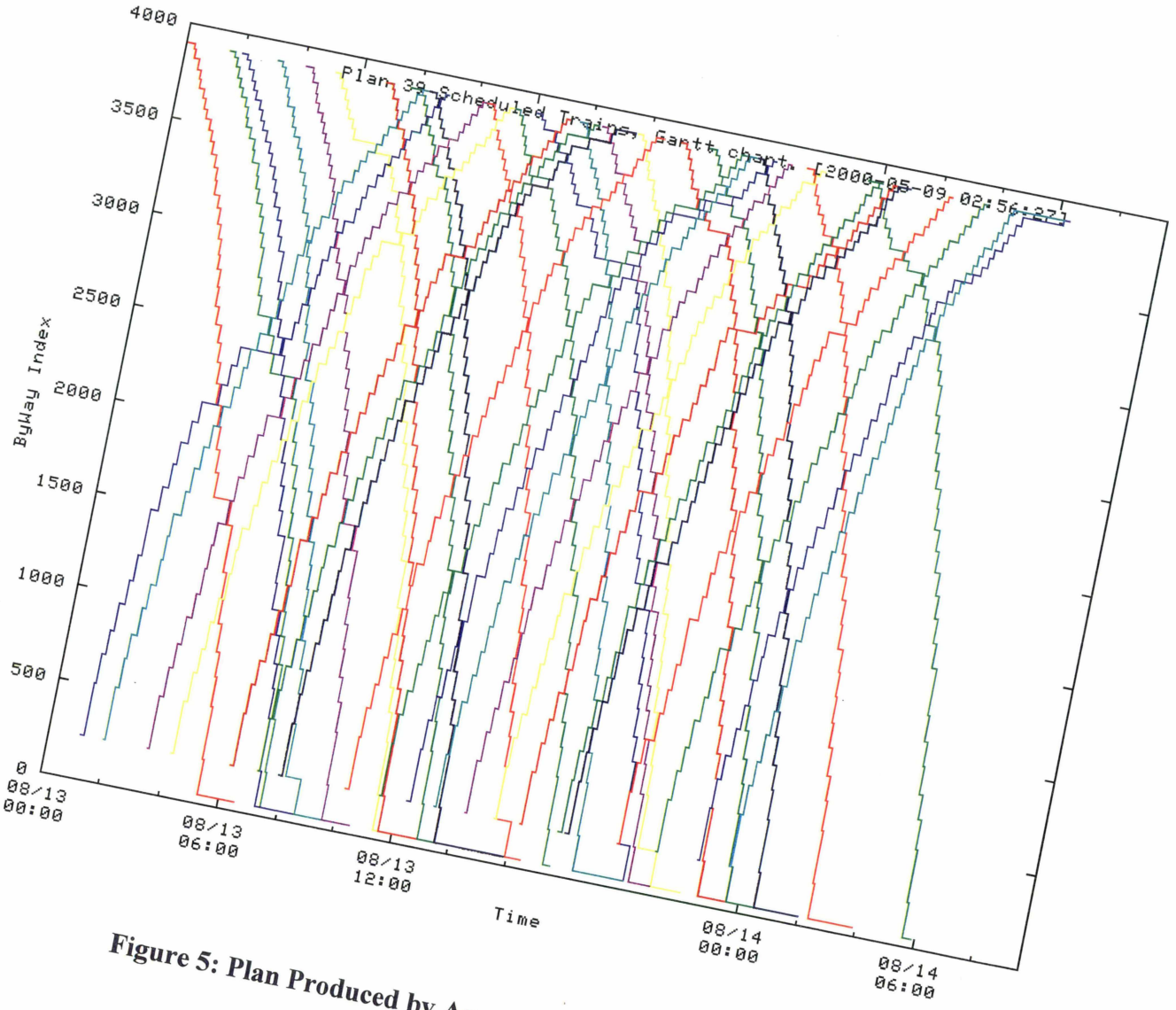

Figure 5: Plan Produced by Agent Application #39

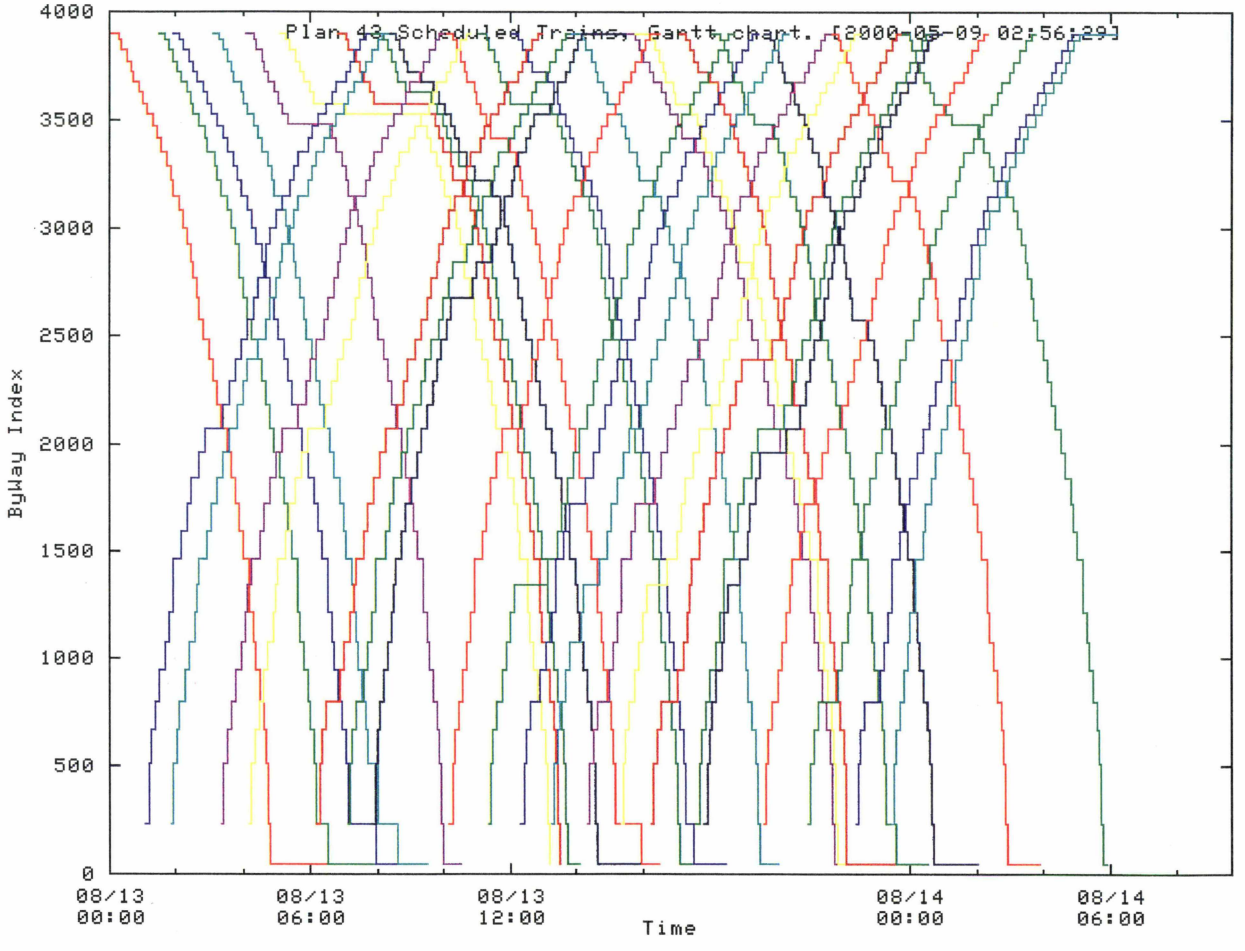

Figure 6: Plan Produced by Agent Application #43

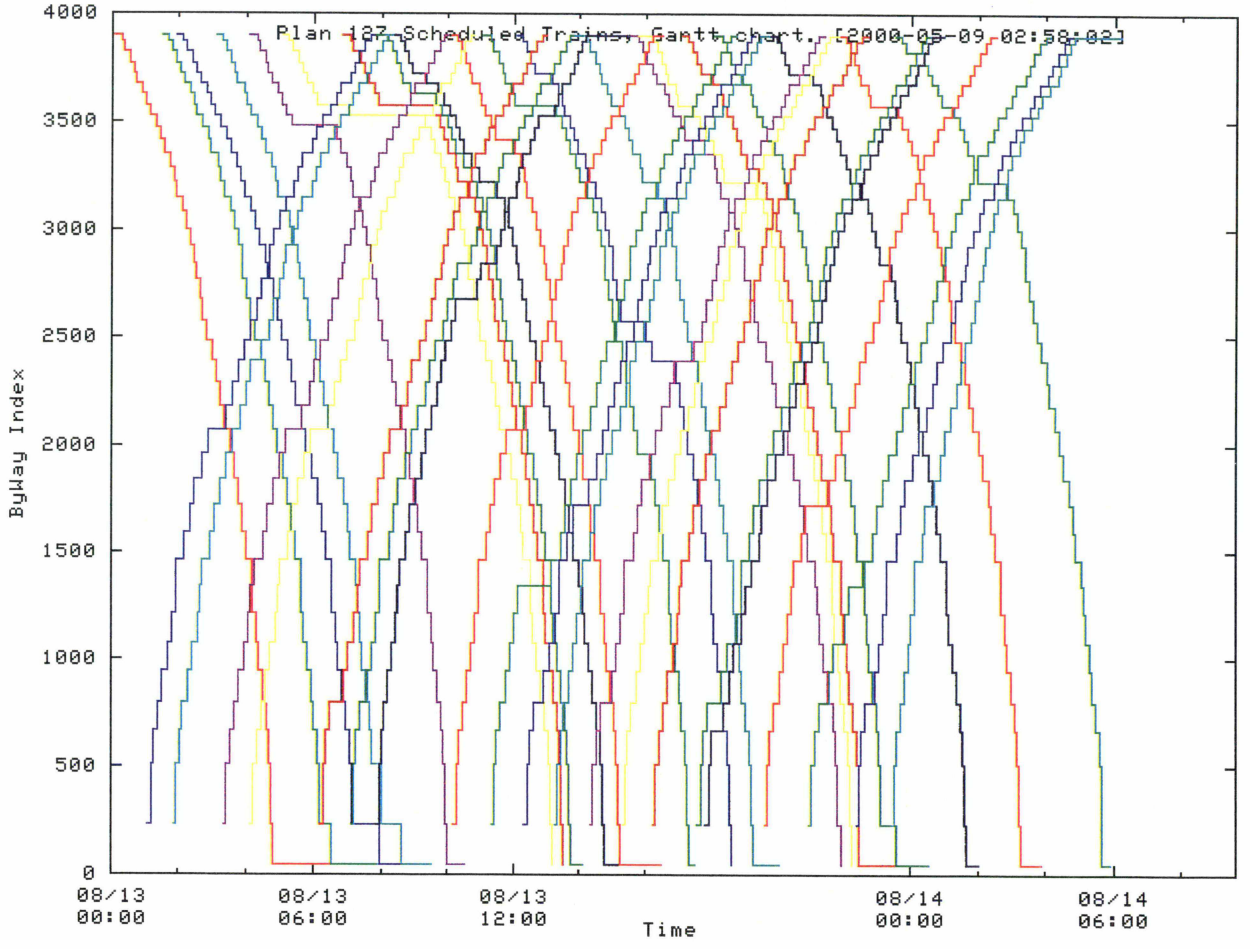

Figure 7: Plan Produced by Agent Application #127

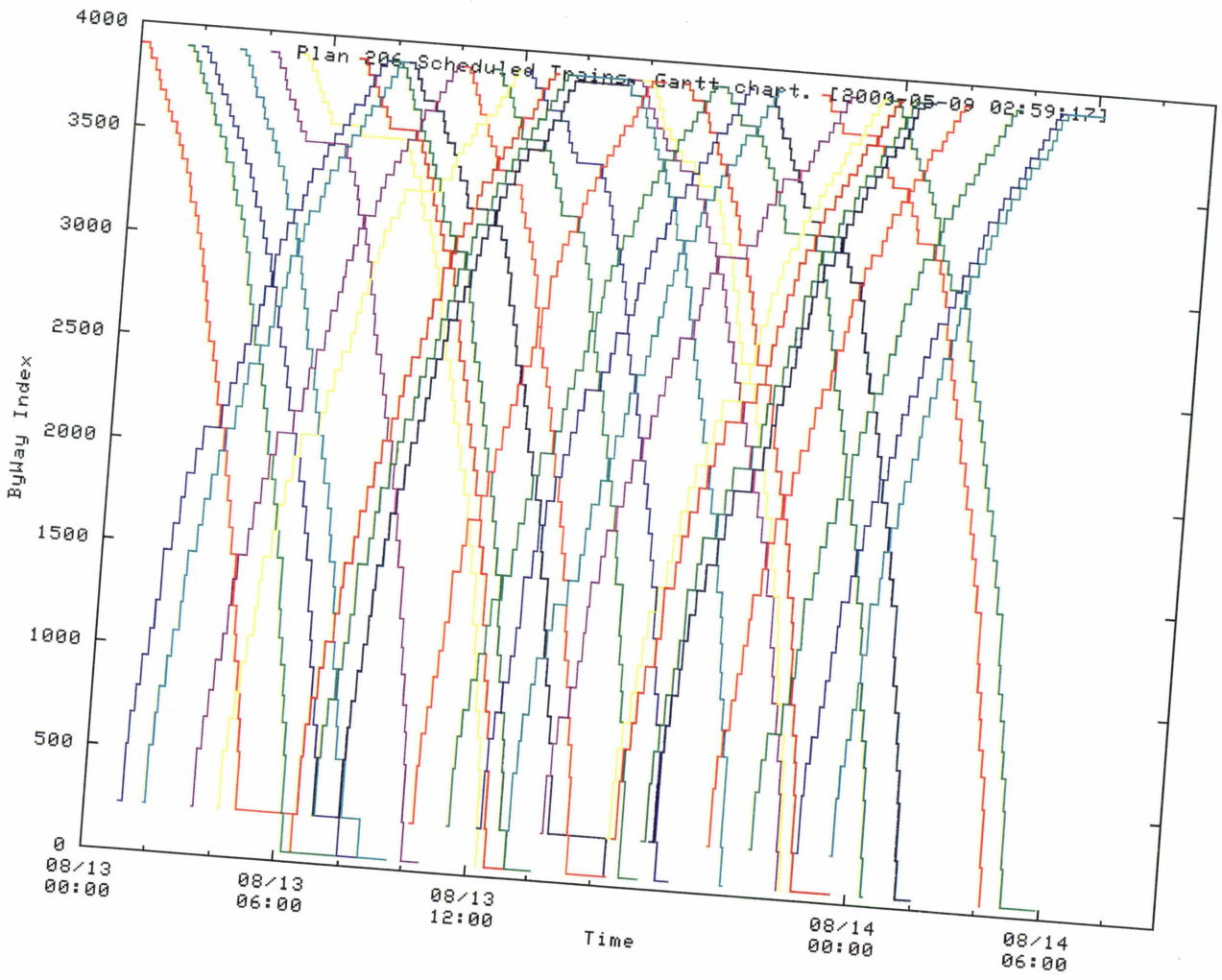

Figure 8: Plan Produced by Agent Application #206

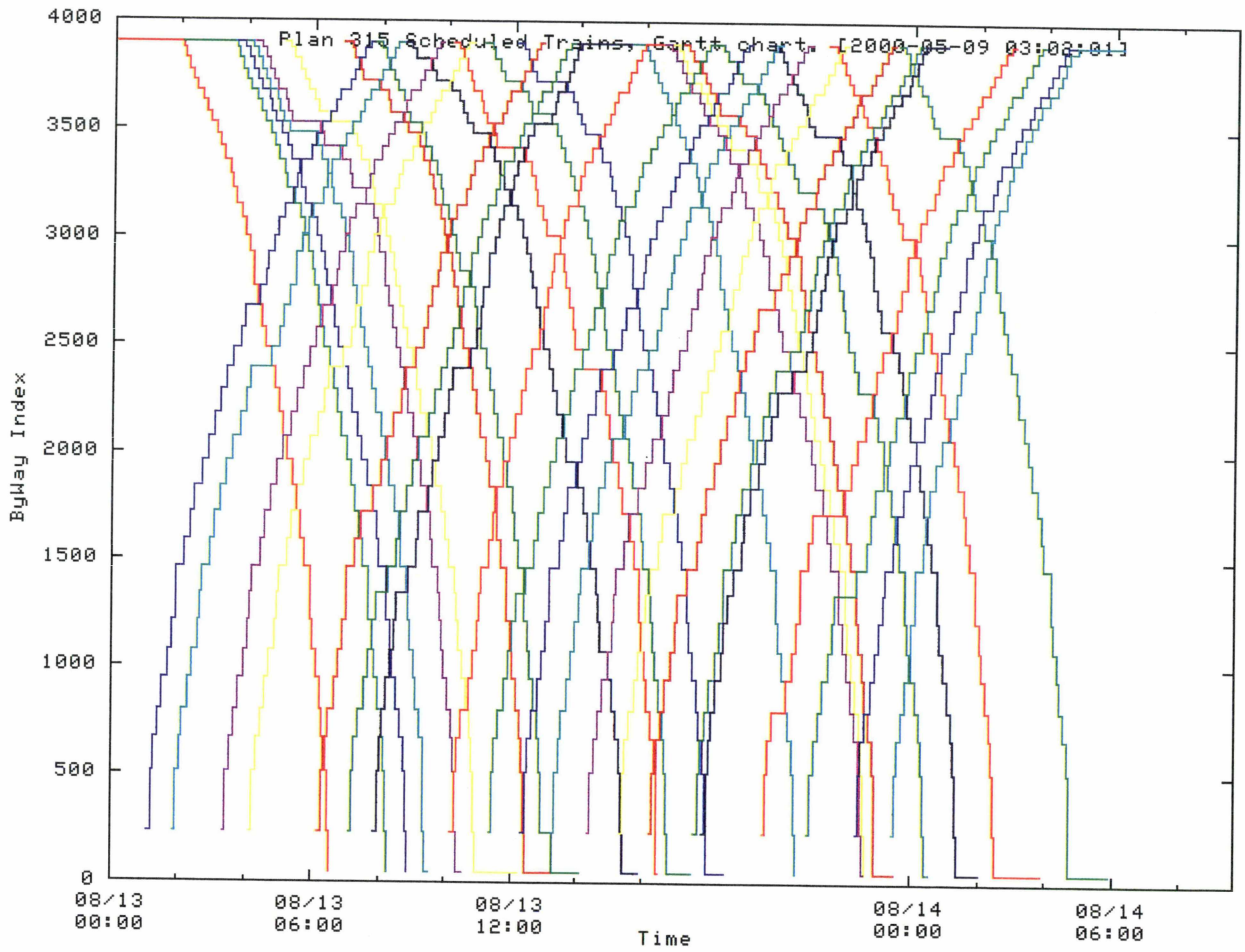

Figure 9: Plan Produced by Agent Application #315

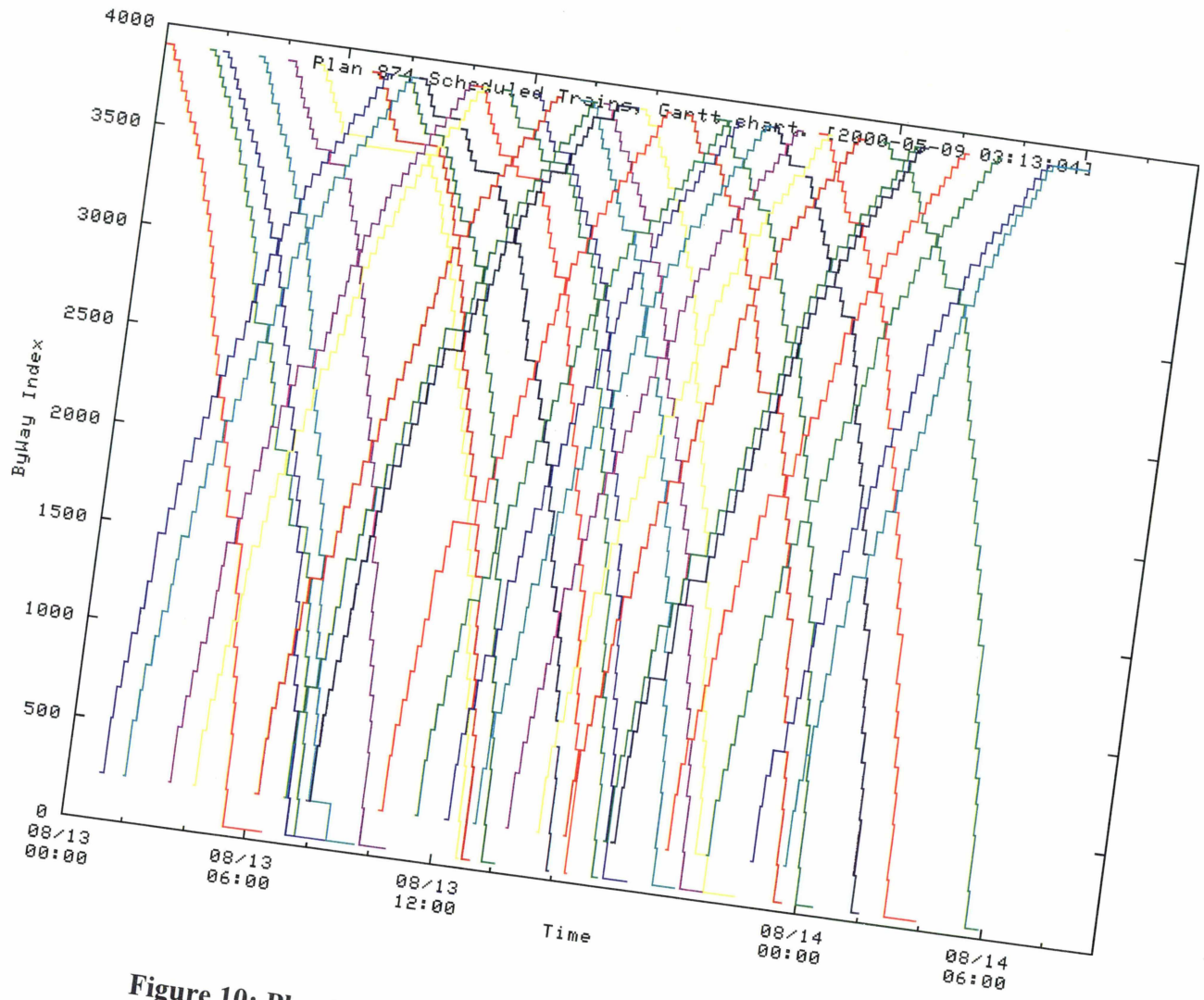

Figure 10: Plan Produced by Agent Application #874

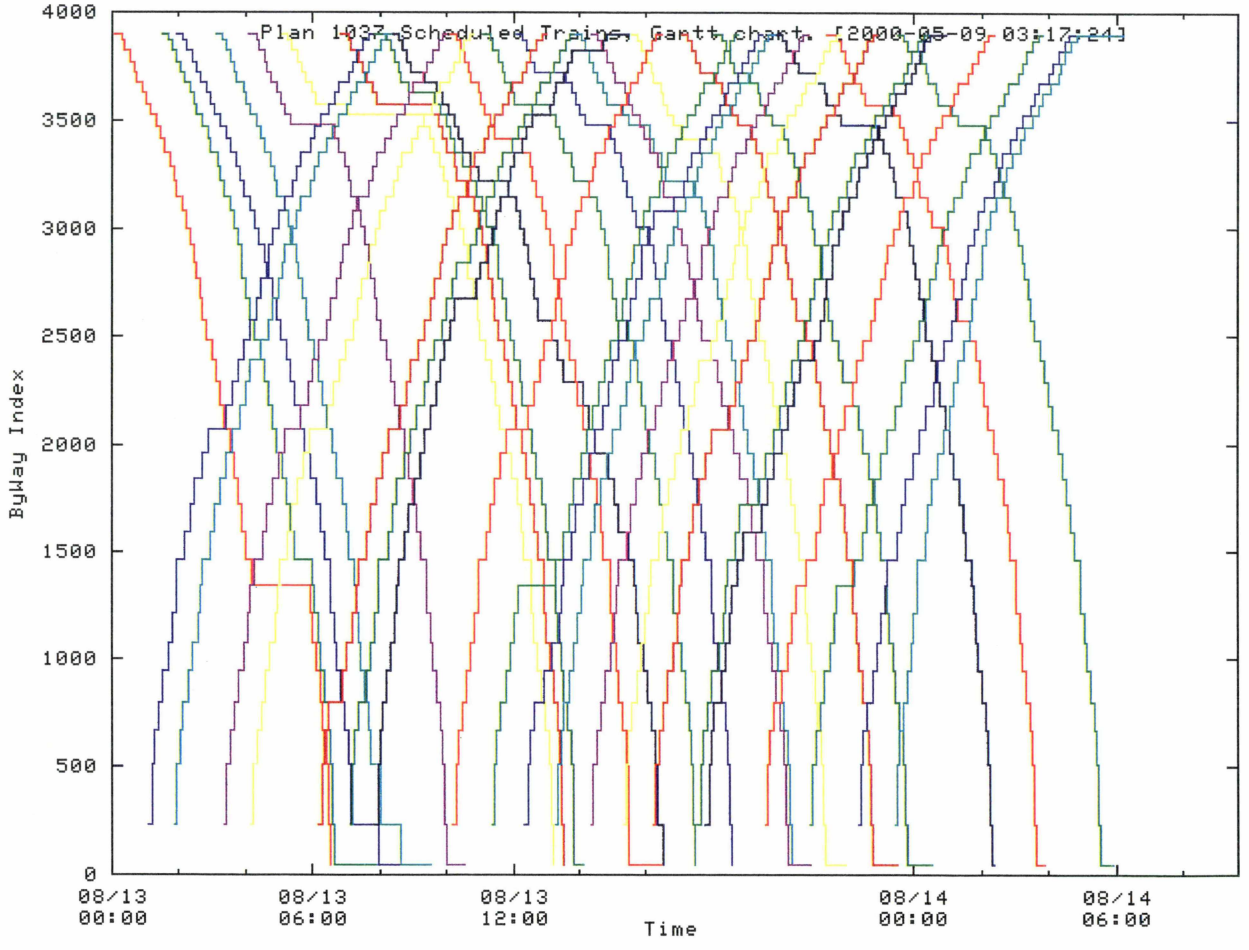

Figure 11: Plan Produced by Agent Application #1037

Agent Performance

In the following Agent Performance table, the agents are specified by their single character ids (refer to Appendix 1 for the names and descriptions of the different agents). The count column reports the number of applications of an agent. The *best* column reports the number of times an agent's action resulted in a new best score. The *better* column reports the number of times that an agent improved a plan, and the *worse* column reports the number of times an agent made a plan worse.

What is interesting to note about the data is that individual agents consistently generate more plans that are worse than the plan they changed rather than better. However, from the Best Plan Score Trend graph above, the group as a whole does very well. The trick in fine tuning the plan generator is to improve the ratio of better to worse plans in order to improve the performance of the system over time.

Agent Performance Statistics Table

Agent	count	best	better	worse
P	112	1	2	67
B	27	0	2	3
b	27	0	1	26
r	151	1	8	128
C	58	0	0	34
c	58	0	1	57
s	7	0	0	0
d	138	0	13	113
%	27	0	2	3
e	48	0	3	29
f	38	0	6	32
F	43	4	10	33
v	1	0	0	0
x	163	5	71	98
Y	48	0	2	32
y	74	0	5	35
i	5	1	0	0
l	45	0	0	45
<	58	0	1	57
L	33	1	17	16
>	58	0	0	34
^	27	0	1	26

Regression Test Results

The table below contains collected regression testing results for the planning scenario described above. The first column specifies the number of trains appearing in each schedule. The second column contains the time interval covered by the schedule from the appearance of the first train to the departure of the final train. The third column lists the actual planned time interval. The fourth column contains the number of agent applications. The elapsed run time in the fifth column includes only agent execution time. The next column shows the number of missed destinations, which is zero in this case (if it was not zero, then the scores would have been greater than 10,000).

The column labeled "Final Minutes Late" contains the sum of the minutes late arriving at the *final* destination for all services (trains). The Scores in the next two columns are the best and median scores of all plans in the pool at the end of the run. Each plan score is computed from: 100000*(*deadlocked trains*) +10000*(*missed locations*) +1*(*minutes deviated from schedule*) +1*(*minutes late at each destination*). Finally, the last four columns are the numbers of the types of agents that were applied (21 modification agents, 1 destroyer agent and 6 constructor agents).

Regression Test Results Table

Number of Trains	Scheduled Span Time, hours	Plan Span Time, hours	Agent Trials	Elapsed Run Time, seconds	Agents / sec	Missed Dest	Final Minutes Late	Score		Agent Types			
								best	median	Mod	Mig	Des	Cons
40	30.0833	29.2005	1010	1022	0.988258	0	298.43	1416	1503	21	0	1	6

APPENDIX 3

Schedule Data

This Appendix shows the schedule data used for generating the plans in Appendix 2. Each service is a train, and each station is a scheduled stop for the particular train. Trains have speed classes and their lengths are in feet. The sizes of the time windows between arriving at a station and departing are zero seconds, though this could be changed. A train id#, the type of train – coal (empty or loaded), freight, etc. – are listed in the first line, along with the train's length in feet, scheduled time of departure onto the line, scheduled arrival time at the end and a few other parameters. (Note: only west-bound trains are listed for brevity. There are 20 trains traveling west and 20 traveling east in 24 hours)

```
// 1 CE001 5 COALEMTY 6222 1 1 30128 W RAVENNA 0 00:05 33 30364 EALLIANCE 0 06:30
0 0 287 0 0 0 08:30 50000 287 0 0 9 99:99

service
  train 0xCE001
  location 0x101 // Station ID: W RAVENNA
  DESTTYPE=0

  dwell=120          //seconds, for the moment.
  earliest_arrival = 1999/8/13 00:05
  desired_arrival  = 1999/8/13 00:05
  latest_arrival   = 1999/8/13 00:05
  earliest_departure= 1999/8/13 00:05
  latest_departure = 1999/8/13 00:05
  depart_speedclass = 5
  delta_length = 6222

    location 0xC0001 // Station ID: EALLIANCE
      leg=0
    // more work properties
      desttype=1
    dwell=0 //seconds, for the moment.
    earliest_arrival  = 1999/8/13 06:30
      desired_arrival   = 1999/8/13 06:30
      latest_arrival    = 1999/8/13 06:30
      earliest_departure = 1999/8/13 06:30
    latest_departure  = 1999/8/13 06:30

// 2 CE002 5 COALEMTY 6222 1 1 30128 W RAVENNA 0 01:30 33 30364 EALLIANCE 0 07:55
0 0 287 0 0 0 09:55 50000 287 0 0 9 99:99

service
  train 0xCE002
  location 0x101 // Station ID: W RAVENNA
  DESTTYPE=0

  dwell=120          //seconds, for the moment.
  earliest_arrival = 1999/8/13 01:30
  desired_arrival  = 1999/8/13 01:30
  latest_arrival   = 1999/8/13 01:30
```

```
  earliest_departure= 1999/8/13 01:30
  latest_departure  = 1999/8/13 01:30
  depart_speedclass =  5
  delta_length =  6222

     location 0xC0001 // Station ID: EALLIANCE
         leg=0
     // more work properties
         desttype=1
     dwell=0 //seconds, for the moment.
     earliest_arrival  = 1999/8/13 07:55
         desired_arrival   = 1999/8/13 07:55
         latest_arrival    = 1999/8/13 07:55
         earliest_departure = 1999/8/13 07:55
     latest_departure  = 1999/8/13 07:55

// 3  121   2 FREIGH45  6730   1   1 30128 W RAVENNA  0 01:55  33 30364 EALLIANCE  0 08:35  0
0   0   3  0 0 10:35   50000   0   3  0 9 99:99

service
   train 121
   location 0x101 // Station ID: W RAVENNA
   DESTTYPE=0

  dwell=120          //seconds, for the moment.
  earliest_arrival  = 1999/8/13 01:55
  desired_arrival   = 1999/8/13 01:55
  latest_arrival    = 1999/8/13 01:55
  earliest_departure= 1999/8/13 01:55
  latest_departure  = 1999/8/13 01:55
  depart_speedclass =  2
  delta_length =  6730

     location 0xC0001 // Station ID: EALLIANCE
         leg=0
     // more work properties
         desttype=1
     dwell=0 //seconds, for the moment.
     earliest_arrival  = 1999/8/13 08:35
         desired_arrival   = 1999/8/13 08:35
         latest_arrival    = 1999/8/13 08:35
         earliest_departure = 1999/8/13 08:35
     latest_departure  = 1999/8/13 08:35

// 4  CE003  5 COALEMTY  6222   1   1 30128 W RAVENNA  0 03:05  33 30364 EALLIANCE  0 09:30
0    0  287   0  0 0 11:30   50000 287   0  0 9 99:99

service
   train 0xCE003
   location 0x101 // Station ID: W RAVENNA
   DESTTYPE=0

  dwell=120          //seconds, for the moment.
  earliest_arrival  = 1999/8/13 03:05
  desired_arrival   = 1999/8/13 03:05
  latest_arrival    = 1999/8/13 03:05
```

```
  earliest_departure= 1999/8/13 03:05
  latest_departure  = 1999/8/13 03:05
  depart_speedclass =  5
  delta_length =  6222

      location 0xC0001 // Station ID: EALLIANCE
          leg=0
      // more work properties
          desttype=1
      dwell=0 //seconds, for the moment.
      earliest_arrival   = 1999/8/13 09:30
          desired_arrival    = 1999/8/13 09:30
          latest_arrival     = 1999/8/13 09:30
          earliest_departure = 1999/8/13 09:30
      latest_departure   = 1999/8/13 09:30

// 5  CE004  5 COALEMTY  6222   1   1 30128 W RAVENNA  0 04:05  33 30364 EALLIANCE  0 10:30
0    0  287   0  0 0 12:30    50000 287   0  0 9 99:99

service
   train 0xCE004
   location 0x101 // Station ID: W RAVENNA
   DESTTYPE=0

  dwell=120            //seconds, for the moment.
  earliest_arrival  = 1999/8/13 04:05
  desired_arrival   = 1999/8/13 04:05
  latest_arrival    = 1999/8/13 04:05
  earliest_departure= 1999/8/13 04:05
  latest_departure  = 1999/8/13 04:05
  depart_speedclass =  5
  delta_length =  6222

      location 0xC0001 // Station ID: EALLIANCE
          leg=0
      // more work properties
          desttype=1
      dwell=0 //seconds, for the moment.
      earliest_arrival   = 1999/8/13 10:30
          desired_arrival    = 1999/8/13 10:30
          latest_arrival     = 1999/8/13 10:30
          earliest_departure = 1999/8/13 10:30
      latest_departure   = 1999/8/13 10:30

//* 6  223    2 FREIGH45  5000   1   1 30128 W RAVENNA  0 05:05  33 30364 EALLIANCE  0 12:15  0
0    0   3  0 0 14:15    50000   0   3  0 9 99:99

service
   train 223
   location 0x101 // Station ID: W RAVENNA
   DESTTYPE=0

  dwell=120            //seconds, for the moment.
  earliest_arrival  = 1999/8/13 05:05
  desired_arrival   = 1999/8/13 05:05
  latest_arrival    = 1999/8/13 05:05
```

```
  earliest_departure= 1999/8/13 05:05
  latest_departure  = 1999/8/13 05:05
  depart_speedclass =  2
  delta_length =  5000

     location 0x1C0BD704 // Station ID: 30183
          leg=0
     // more work properties
          desttype=1
     dwell=1800 //seconds, for the moment.
     earliest_arrival   = 1999/8/13 05:05
          desired_arrival    = 1999/8/13 05:05
          latest_arrival     = 1999/8/13 12:15
          earliest_departure = 1999/8/13 05:05
     latest_departure   = 1999/8/13 12:15

     location 0xC0001 // Station ID: EALLIANCE
          leg=0
     // more work properties
          desttype=1
     dwell=0 //seconds, for the moment.
     earliest_arrival   = 1999/8/13 12:15
          desired_arrival    = 1999/8/13 12:15
          latest_arrival     = 1999/8/13 12:15
          earliest_departure = 1999/8/13 12:15
     latest_departure   = 1999/8/13 12:15

// 7 CE005 5 COALEMTY 6222 1 1 30128 W RAVENNA 0 06:50 33 30364 EALLIANCE 0 13:15
0   0 287 0 0 0 15:15  50000 287 0 0 9 99:99

service
   train 0xCE005
   location 0x101 // Station ID: W RAVENNA
   DESTTYPE=0

  dwell=120          //seconds, for the moment.
  earliest_arrival  = 1999/8/13 06:50
  desired_arrival   = 1999/8/13 06:50
  latest_arrival    = 1999/8/13 06:50
  earliest_departure= 1999/8/13 06:50
  latest_departure  = 1999/8/13 06:50
  depart_speedclass =  5
  delta_length =  6222

     location 0xC0001 // Station ID: EALLIANCE
          leg=0
     // more work properties
          desttype=1
     dwell=0 //seconds, for the moment.
     earliest_arrival   = 1999/8/13 13:15
          desired_arrival    = 1999/8/13 13:15
          latest_arrival     = 1999/8/13 13:15
          earliest_departure = 1999/8/13 13:15
     latest_departure   = 1999/8/13 13:15
```

```
// 8 CE006 5 COALEMTY 6222 1 1 30128 W RAVENNA 0 07:40 33 30364 EALLIANCE 0 14:05
0 0 287 0 0 0 16:05 50000 287 0 0 9 99:99

service
  train 0xCE006
  location 0x101 // Station ID: W RAVENNA
  DESTTYPE=0

  dwell=120          //seconds, for the moment.
  earliest_arrival = 1999/8/13 07:40
  desired_arrival  = 1999/8/13 07:40
  latest_arrival   = 1999/8/13 07:40
  earliest_departure= 1999/8/13 07:40
  latest_departure = 1999/8/13 07:40
  depart_speedclass = 5
  delta_length = 6222

    location 0xC0001 // Station ID: EALLIANCE
      leg=0
    // more work properties
      desttype=1
    dwell=0 //seconds, for the moment.
    earliest_arrival  = 1999/8/13 14:05
      desired_arrival   = 1999/8/13 14:05
      latest_arrival    = 1999/8/13 14:05
      earliest_departure = 1999/8/13 14:05
    latest_departure  = 1999/8/13 14:05

// 9 125 2 FREIGH45 6730 1 1 30128 W RAVENNA 0 08:20 33 30364 EALLIANCE 0 15:55 0
0 0 3 0 0 17:55 50000 0 3 0 9 99:99

service
  train 125
  location 0x101 // Station ID: W RAVENNA
  DESTTYPE=0

  dwell=120          //seconds, for the moment.
  earliest_arrival = 1999/8/13 08:20
  desired_arrival  = 1999/8/13 08:20
  latest_arrival   = 1999/8/13 08:20
  earliest_departure= 1999/8/13 08:20
  latest_departure = 1999/8/13 08:20
  depart_speedclass = 2
  delta_length = 6730

    location 0xC0001 // Station ID: EALLIANCE
      leg=0
    // more work properties
      desttype=1
    dwell=0 //seconds, for the moment.
    earliest_arrival  = 1999/8/13 15:55
      desired_arrival   = 1999/8/13 15:55
      latest_arrival    = 1999/8/13 15:55
      earliest_departure = 1999/8/13 15:55
    latest_departure  = 1999/8/13 15:55
```

```
// 10  CE007  5 COALEMTY  6222   1   1 30128 W RAVENNA  0 10:05  33 30364 EALLIANCE  0 16:30
0    0  287   0  0 0 18:30    50000 287   0  0 9 99:99

service
   train 0xCE007
   location 0x101 // Station ID: W RAVENNA
   DESTTYPE=0

   dwell=120            //seconds, for the moment.
   earliest_arrival  = 1999/8/13 10:05
   desired_arrival   = 1999/8/13 10:05
   latest_arrival    = 1999/8/13 10:05
   earliest_departure= 1999/8/13 10:05
   latest_departure  = 1999/8/13 10:05
   depart_speedclass =  5
   delta_length =  6222

      location 0xC0001 // Station ID: EALLIANCE
          leg=0
      // more work properties
          desttype=1
      dwell=0 //seconds, for the moment.
      earliest_arrival  = 1999/8/13 16:30
          desired_arrival   = 1999/8/13 16:30
          latest_arrival    = 1999/8/13 16:30
          earliest_departure = 1999/8/13 16:30
      latest_departure  = 1999/8/13 16:30

// 11  CE008  5 COALEMTY  6222   1   1 30128 W RAVENNA  0 11:05  33 30364 EALLIANCE  0 17:30
0    0  287   0  0 0 19:30    50000 287   0  0 9 99:99

service
   train 0xCE008
   location 0x101 // Station ID: W RAVENNA
   DESTTYPE=0

   dwell=120            //seconds, for the moment.
   earliest_arrival  = 1999/8/13 11:05
   desired_arrival   = 1999/8/13 11:05
   latest_arrival    = 1999/8/13 11:05
   earliest_departure= 1999/8/13 11:05
   latest_departure  = 1999/8/13 11:05
   depart_speedclass =  5
   delta_length =  6222

      location 0xC0001 // Station ID: EALLIANCE
          leg=0
      // more work properties
          desttype=1
      dwell=0 //seconds, for the moment.
      earliest_arrival  = 1999/8/13 17:30
          desired_arrival   = 1999/8/13 17:30
          latest_arrival    = 1999/8/13 17:30
          earliest_departure = 1999/8/13 17:30
      latest_departure  = 1999/8/13 17:30
```

```
// 12  CE009  5 COALEMTY  6222   1   1 30128 W RAVENNA  0 12:05  33 30364 EALLIANCE  0 18:30
0    0  287   0  0 0 20:30    50000 287   0  0 9 99:99

service
   train 0xCE009
   location 0x101 // Station ID: W RAVENNA
   DESTTYPE=0

  dwell=120            //seconds, for the moment.
  earliest_arrival  = 1999/8/13 12:05
  desired_arrival   = 1999/8/13 12:05
  latest_arrival    = 1999/8/13 12:05
  earliest_departure= 1999/8/13 12:05
  latest_departure  = 1999/8/13 12:05
  depart_speedclass =  5
  delta_length =  6222

      location 0xC0001 // Station ID: EALLIANCE
          leg=0
      // more work properties
          desttype=1
      dwell=0 //seconds, for the moment.
      earliest_arrival   = 1999/8/13 18:30
          desired_arrival    = 1999/8/13 18:30
          latest_arrival     = 1999/8/13 18:30
          earliest_departure = 1999/8/13 18:30
      latest_departure   = 1999/8/13 18:30

// 13  CE010  5 COALEMTY  6222   1   1 30128 W RAVENNA  0 13:35  33 30364 EALLIANCE  0 20:00
0    0  287   0  0 0 22:00    50000 287   0  0 9 99:99

service
   train 0xCE010
   location 0x101 // Station ID: W RAVENNA
   DESTTYPE=0

  dwell=120            //seconds, for the moment.
  earliest_arrival  = 1999/8/13 13:35
  desired_arrival   = 1999/8/13 13:35
  latest_arrival    = 1999/8/13 13:35
  earliest_departure= 1999/8/13 13:35
  latest_departure  = 1999/8/13 13:35
  depart_speedclass =  5
  delta_length =  6222

      location 0xC0001 // Station ID: EALLIANCE
          leg=0
      // more work properties
          desttype=1
      dwell=0 //seconds, for the moment.
      earliest_arrival   = 1999/8/13 20:00
          desired_arrival    = 1999/8/13 20:00
          latest_arrival     = 1999/8/13 20:00
          earliest_departure = 1999/8/13 20:00
      latest_departure   = 1999/8/13 20:00
```

```
// 14  021    1 INTERMOD  5081   1    1 30128 W RAVENNA  0 14:20  33 30364 EALLIANCE  0 20:55  0
0  443   0  0 0 0 22:55    50000 443   0  0 9 99:99

service
   train 021
   location 0x101 // Station ID: W RAVENNA
   DESTTYPE=0

  dwell=120            //seconds, for the moment.
  earliest_arrival  = 1999/8/13 14:20
  desired_arrival   = 1999/8/13 14:20
  latest_arrival    = 1999/8/13 14:20
  earliest_departure= 1999/8/13 14:20
  latest_departure  = 1999/8/13 14:20
  depart_speedclass =  1
  delta_length =  5081

      location 0xC0001 // Station ID: EALLIANCE
          leg=0
      // more work properties
          desttype=1
      dwell=0 //seconds, for the moment.
      earliest_arrival   = 1999/8/13 20:55
          desired_arrival    = 1999/8/13 20:55
          latest_arrival     = 1999/8/13 20:55
          earliest_departure = 1999/8/13 20:55
      latest_departure   = 1999/8/13 20:55

// 15  CE011  5 COALEMTY  6222   1    1 30128 W RAVENNA  0 15:35  33 30364 EALLIANCE  0 22:00
0    0  287   0  0 1 00:00    50000 287   0  0 9 99:99

service
   train 0xCE011
   location 0x101 // Station ID: W RAVENNA
   DESTTYPE=0

  dwell=120            //seconds, for the moment.
  earliest_arrival  = 1999/8/13 15:35
  desired_arrival   = 1999/8/13 15:35
  latest_arrival    = 1999/8/13 15:35
  earliest_departure= 1999/8/13 15:35
  latest_departure  = 1999/8/13 15:35
  depart_speedclass =  5
  delta_length =  6222

      location 0xC0001 // Station ID: EALLIANCE
          leg=0
      // more work properties
          desttype=1
      dwell=0 //seconds, for the moment.
      earliest_arrival   = 1999/8/13 22:00
          desired_arrival    = 1999/8/13 22:00
          latest_arrival     = 1999/8/13 22:00
          earliest_departure = 1999/8/13 22:00
      latest_departure   = 1999/8/13 22:00
```

```
// 16  CE012  5 COALEMTY  6222   1   1 30128 W RAVENNA  0 17:05  33 30364 EALLIANCE  0 23:30
0    0  287   0  0 1 01:30   50000 287   0  0 9 99:99

service
   train 0xCE012
   location 0x101 // Station ID: W RAVENNA
   DESTTYPE=0

  dwell=120            //seconds, for the moment.
  earliest_arrival  = 1999/8/13 17:05
  desired_arrival   = 1999/8/13 17:05
  latest_arrival    = 1999/8/13 17:05
  earliest_departure= 1999/8/13 17:05
  latest_departure  = 1999/8/13 17:05
  depart_speedclass =  5
  delta_length =  6222

      location 0xC0001 // Station ID: EALLIANCE
          leg=0
      // more work properties
          desttype=1
      dwell=0 //seconds, for the moment.
      earliest_arrival   = 1999/8/13 23:30
          desired_arrival    = 1999/8/13 23:30
          latest_arrival     = 1999/8/13 23:30
          earliest_departure = 1999/8/13 23:30
      latest_departure   = 1999/8/13 23:30

// 17  CE013  5 COALEMTY  6222   1   1 30128 W RAVENNA  0 18:05  33 30364 EALLIANCE  1 00:30
0    0  287   0  0 1 02:30   50000 287   0  0 9 99:99

service
   train 0xCE013
   location 0x101 // Station ID: W RAVENNA
   DESTTYPE=0

  dwell=120            //seconds, for the moment.
  earliest_arrival  = 1999/8/13 18:05
  desired_arrival   = 1999/8/13 18:05
  latest_arrival    = 1999/8/13 18:05
  earliest_departure= 1999/8/13 18:05
  latest_departure  = 1999/8/13 18:05
  depart_speedclass =  5
  delta_length =  6222

      location 0xC0001 // Station ID: EALLIANCE
          leg=0
      // more work properties
          desttype=1
      dwell=0 //seconds, for the moment.
      earliest_arrival   = 1999/8/14 00:30
          desired_arrival    = 1999/8/14 00:30
          latest_arrival     = 1999/8/14 00:30
          earliest_departure = 1999/8/14 00:30
      latest_departure   = 1999/8/14 00:30
```

```
// 18  CE014  5 COALEMTY  6222   1   1 30128 W RAVENNA  0 19:35  33 30364 EALLIANCE  1 02:00
0    0  287   0  0 1 04:00    50000 287   0  0 9 99:99

service
   train 0xCE014
   location 0x101 // Station ID: W RAVENNA
   DESTTYPE=0

  dwell=120            //seconds, for the moment.
  earliest_arrival  = 1999/8/13 19:35
  desired_arrival   = 1999/8/13 19:35
  latest_arrival    = 1999/8/13 19:35
  earliest_departure= 1999/8/13 19:35
  latest_departure  = 1999/8/13 19:35
  depart_speedclass =  5
  delta_length =  6222

      location 0xC0001 // Station ID: EALLIANCE
          leg=0
      // more work properties
          desttype=1
      dwell=0 //seconds, for the moment.
      earliest_arrival   = 1999/8/14 02:00
          desired_arrival    = 1999/8/14 02:00
          latest_arrival     = 1999/8/14 02:00
          earliest_departure = 1999/8/14 02:00
      latest_departure   = 1999/8/14 02:00

// 19  CE015  5 COALEMTY  6222   1   1 30128 W RAVENNA  0 21:25  33 30364 EALLIANCE  1 03:50
0    0  287   0  0 1 05:50    50000 287   0  0 9 99:99

service
   train 0xCE015
   location 0x101 // Station ID: W RAVENNA
   DESTTYPE=0

  dwell=120            //seconds, for the moment.
  earliest_arrival  = 1999/8/13 21:25
  desired_arrival   = 1999/8/13 21:25
  latest_arrival    = 1999/8/13 21:25
  earliest_departure= 1999/8/13 21:25
  latest_departure  = 1999/8/13 21:25
  depart_speedclass =  5
  delta_length =  6222

      location 0xC0001 // Station ID: EALLIANCE
          leg=0
      // more work properties
          desttype=1
      dwell=0 //seconds, for the moment.
      earliest_arrival   = 1999/8/14 03:50
          desired_arrival    = 1999/8/14 03:50
          latest_arrival     = 1999/8/14 03:50
          earliest_departure = 1999/8/14 03:50
      latest_departure   = 1999/8/14 03:50
```

```
// 20  CE016  5 COALEMTY  6222   1   1 30128 W RAVENNA  0 23:30  33 30364 EALLIANCE  1 05:55
0    0  287   0  0 1 07:55    50000 287   0  0 9 99:99

service
   train 0xCE016
   location 0x101 // Station ID: W RAVENNA
   DESTTYPE=0

  dwell=120            //seconds, for the moment.
  earliest_arrival  = 1999/8/13 23:30
  desired_arrival   = 1999/8/13 23:30
  latest_arrival    = 1999/8/13 23:30
  earliest_departure= 1999/8/13 23:30
  latest_departure  = 1999/8/13 23:30
  depart_speedclass =  5
  delta_length =  6222

      location 0xC0001 // Station ID: EALLIANCE
          leg=0
      // more work properties
          desttype=1
      dwell=0 //seconds, for the moment.
      earliest_arrival   = 1999/8/14 05:55
          desired_arrival    = 1999/8/14 05:55
          latest_arrival     = 1999/8/14 05:55
          earliest_departure = 1999/8/14 05:55
      latest_departure   = 1999/8/14 05:55
```

APPENDIX 4

This appendix contains the activity sections from each of the project progress reports that were submitted between November 1, 1999 and February 29, 2000.

PROGRESS REPORT #1:

Progress from June 1, 1999 to October 31, 1999

The following is a list of system components and concomitant work activities that have been progressed since June 1, 1999. In order to provide the reader with a better understanding of the work involved and how it relates to the overall system, some of these are accompanied by extended explanations. Generally speaking, a working version of the OTP has been implemented that accepts as inputs train schedules, train properties and track descriptions for a region of a rail network and then constructs optimized, detailed movement, or meet/pass, plans for trains over a specified time interval (e.g., 24 hours). Currently this software generates feasible plans based on static data for the track plan being used for testing. Constructor agents, a destroyer agent and modification agents have been implemented. Given a feasible schedule, an operationally feasible movement plan will be generated. Currently the plans are scored by adding the minutes late based on the schedule that is being optimized to. When queried, it will output the best plan in the form of a string line diagram.

- *Train Dispatcher. The dispatcher has been fully designed and implemented, and is currently undergoing integration and performance testing.*

  The Train Dispatcher is responsible for creating the reservations that accomplish a given plan. The plan contains information about the trains and their desired destinations. Trains are dispatched one *leg* (one or more sections of track) at a time based on the priority of leg destinations. The priorities of the train's destinations are set by the agents. The agents set the priorities in the priority matrix in order to influence the dispatching process.

  During the dispatching process for a train, the dispatcher reserves the routes that will allow the train to move to its desired destination along a preferred course. If there is a route that can not be reserved, the dispatcher tries to find another route that will allow the train to continue towards its destination.

  When there are no other routes the dispatcher determines if the train could wait at its last reservation for the next route to clear. If waiting is possible, the dispatcher will extend the last reservation and continue reserving the course with the consequent delay. If waiting is not possible, the last reservation is deleted and the train is "virtually" backed up one route and the process is repeated by looking for an alternative and waiting. This will accomplish a meet/pass for two opposing trains as

required. The process will continue until the train either reaches its destination or the start of the leg that is being dispatched.

If backing up continues to the beginning of the course, the dispatcher will try an alternative course if one exists. If no other courses exist for that train, it is put in a holding pool for later consideration. The dispatching then continues with the next train. After successfully dispatching the next train, any trains that were put on hold are reintroduced in an attempt to dispatched those trains again. This process will continue until either all trains have been dispatched or the remaining trains have been put on hold. Trains remaining in the holding pool when the dispatcher has completed will contribute a penalty to the score of the plan.

This method of dispatching has an advantage over static branch and bound methods for movement plan generation in that it is able to account for time-dependent constraints like temporary slow orders, which the latter methods annot do without a lot of difficulty.

- *Deadlock predictor algorithm. The deadlock predictor has been designed and implemented to work on general network configurations of track, and so should be applicable to any railroad's trackage. It is used by the dispatcher component.*

  Though started before June 1st, this critical component was finished during the summer. A deadlock avoidance mechanism is required to prevent machine induced deadlocks from occurring. A particular method for predicting deadlocks that is general across different kinds of interlockings was chosen for this task. Called the Modular Switch Array Method (MSAM), it predicts whether the movement of a train onto a segment of track (a schedulable operation) will create an eventual deadlock. Without deadlock avoidance, it was found that most solutions generated by the OTP contained deadlocks, which is a computationally prohibitive and operationally unacceptable.

  The deadlock predictor is based on a model of the connectivity and capacity of the railroad network derived from the routing and trackwork interconnects, along with a few heuristic statements about the ability of trains to move. The model and the statements predict whether the movement of a train into the next route will result in deadlock of the railroad network. The predictor avoids NP-completeness by limiting the search scope when evaluating the statements to a reasonable space. The limits will result in accurate predictions for all but the most congested networks, without undue computational cost.

- *Development of the Movement model: The movement model has been designed. Implementation of several of the factors affecting train movement has begun.*

  The OTP must estimate the duration of travel for the trains as they move. It is the Movement Model component that determines the duration and speed profile of a train's movement between two points on the railroad network. Such a calculation

depends on restrictions that apply at any point in time, like grade and curvature information, as well as train characteristics. The static information needed, such as civil speed limits and terrain information, exists in the track database. Other dynamic information, such as certain train characteristics and train ahead identification, is provided by other components of the CTC system through the OTP's database. Additionally, device behaviors such as switch position change and signal state change are also accounted for.

All of this information is used to calculate a maximum speed profile that the train needs to satisfy. Terrain and train characteristics affect acceleration and deceleration rates. The tractive and resistive forces are computed based on available information about locomotives, load and length of the train, terrain information, and so on. In addition, operating style for manual locomotives and control parameters for automated cabs may affect the movement dynamics (though this will not be part of the movement model for this release of the OTP). Because the required information is not always readily available, the Movement Model should gracefully deal with incomplete data. For example, when the terrain information is incomplete, the acceleration and deceleration are approximated. Finally, it should be mentioned that safe braking distance models play an important role in the determination of simulated speed and deceleration rate.

A simulated speed profile over the segment chosen results after applying acceleration and deceleration rates to the previously computed maximum speed profile. This discrete profile is used to determine the duration of the movement that is returned to the planner, so that it can add it to the movement plan.

- *Development of software problem-solving agents. Development of software agents is an ongoing task, since better agents along with different combinations of agents will solve the problem more effectively.*

  Agents developed after June 1st include a crossover agent, which is a genetic algorithm that mimics genetic crossover, a critical path agent that alters which trains are on the critical path, a meet/pass improver agent, a late-time improver agent and a wait-time improver agent. The latter three agents are heuristics that look for specific features of a train's movement profile, like waiting too long at some point. If such features are identified, the agent attempts to change the train's priority in order to remove or reduce the effects of the feature, after which the dispatcher component generates a new movement plan.

- *Development of the Connectivity Model: The connectivity model has been designed and implemented.*

  The basis for the rail connectivity model of the OTP is US&S's Support Database. The physical devices that make up a rail network are defined in this database. The OTP connectivity model inherits from this basic framework to build its own model of the railroad. From the connection of tracks, switches and signals in the US&S

support database, the OTP model builds a configuration based on authority units (the smallest portion of the rail network that may be dispatched), interchanges (interlockings where alternate routing options may be employed) and segments (areas delimited by the same 'from' and 'to' interchanges, which contains a list of authority units that originate at the 'from' interchange and terminate at the 'to' interchange).

- *Scoring mechanism*: *A simple scoring mechanism has been designed and implemented. In the coming months, this component will be enhanced.*

  Since the OTP maintains a collection of movement plans (solutions to the problem of moving trains to a schedule) that are optimized according to specific optimization criteria, it needs a method of ranking the overall goodness of each plan. The lowest scoring plans will be destroyed, while the better plans will be retained for modification and, if selected, execution.

  The Scoring mechanism used in the OTP allows for multiple evaluation metrics to be used for judging each plan. Each evaluation metric represents an aspect of a plan's goodness or badness. For example, there is a Schedule Deviation metric, which measures the number of minutes a train has deviated from its schedule. For that particular metric, the smaller the number, the better a plan is. The scores generated by the multiple evaluation metrics are then combined to form a single score that represents a plan's overall goodness.

- *Integration testing*: integration testing of the OTP began in September and continues today. The A_Teams problem-solving subsystem and the above described modules currently make up the OTP. A number of bugs in the dispatcher have been found and corrected because of integration testing. As new components are completed, they will be integration tested.

- *Performance testing*: One of the things that is imperative for the OTP is that it be able to generate optimized plans in under 30 seconds and preferably in less than 20 seconds. This is necessary so that the *human* dispatcher is able to quickly see the movement plans after a change has occurred, such as a track block or slow order. The dispatcher component of the OTP represents the most time consuming of the processes and so has been the focus of performance improvements. In the course of this testing, many algorithms have been streamlined to enhance performance. In addition, since better agents produce better solutions faster, agent development is another area that will likely enhance performance.

PROGRESS REPORT #2:

Progress for November, 1999

The work in November consisted of the following tasks:

- Further refinement of the movement model. This work consisted primarily of debugging, followed by testing, which took longer than expected. The movement model now appears to work correctly. It is used by the dispatcher component of the OTP to determine how long it will take a train to traverse a particular section of track.

- Design document for the movement model. This document will be completed in December as part of the Alpha Demonstration release of the Optimizing Traffic Planner (OTP).

- New software agent development. Several new agents were added to the OTP. These include:
    a. *Pass Constructor Agent*. This software agent determines if a pass is needed, and if so, will change the priorities of the trains involved for entering the section of track where the next pass is possible.
    b. *Useless Siding Agent*: This agent eliminates trains from entering sidings for no reason. The reason a plan would contain such a maneuver is that another train, which was the original reason for the first train entering the siding, had its route changed in the meantime.
    c. *Early Arrival Remover*: This agent attempts to get rid of trains arriving at their destinations too early.

  The system currently has 24 agents, including the three above. Besides the latter three, they consist of a crossover agent (genetic algorithm), five bottleneck improver agents, two additional useless siding agents, three additional early arrival remover agents, a route divider agent, three critical path improver agents, a prioritize-by-schedule agent, and five plan initialization agents.

- Found and corrected several bugs in the statistical analysis software. This software is responsible for keeping statistics on the number of times agents are applied to the solutions in the solution pool, the improvements of movement plans over time, and so forth. This software is used to analyze the performance of the OTP and helps determine where improvements can be made.

- Ran the planner on a second railroad database. Previous to this, the OTP was being run on a CSX territory that was primarily single track. In November, the testing territories were switched over to those of BNSF. Switching databases brought to the fore a number of issues and a couple of bugs. Since the BNSF database was much larger than the CSX database, changes were made to the OTP software to shorten the initialization phase (when the track database is read into the system). Also, there were a couple of bugs in the database itself, and a bug was detected in the OTP and corrected.

- Software was added so that agents can be configured during runtime rather than compile time. Doing this allows agents to be added to (and deleted from) the active set so that new agents can be analyzed in terms of how well they improve the performance of the OTP. Different combinations of agents will probably work better depending on the track configuration they are planning over.

- Modified OTP software to reduce memory consumption. Because there are approximately 50 solutions in memory simultaneously, the system uses a large amount of memory. This usage was streamlined so that less swapping would occur and the OTP would run faster.

- Incorporating the constraint propagation solution generator with the A_Teams planning architecture. One of the efforts of the OTP project has been to develop an alternative problem solver – that is, one that could solve the planning problem on its own, without benefit of the A_Teams organization. This alternative solver applies constraint satisfaction in the context of a branch and bound search algorithm (though such algorithms exhibit certain limitations when applied to problems the size of rail traffic planning). Because this alternative problem solver generates fairly good initial solutions quickly (the other initialization agents alluded to above are fast, but the solutions they generate are not that good), it was decided to integrate it with the A_Teams problem solver as a construction agent. There is still work remaining to get this integration completed.

During November, having overhauled the movement model and debugged other parts of the system, the development team was able to reduce the time to generate solutions, as well as improve the quality of solutions. Following the next section, there is a data section showing output from the OTP planning over a BNSF coal line in Nebraska (between Ravenna and Alliance). The train data for this was received from BN in 1994 – 95.

PROGRESS REPORT #3:

Progress for December, 1999

Work on the Optimizing Traffic Planner (OTP) in December consisted of the following:

- Fixing problems with several agents. This work consisted primarily of debugging, followed by testing the changes.

- Design document for the movement model. This document was completed during December as part of the Alpha Demonstration release high-level design

documentation of the OTP. It will have added to it the detailed system design during January and February.

- New software agent development. Several new agents were added to the OTP. These include:
    d. *Overtake Agent*. This software agent looks for situations where passes between trains could occur.
    e. *Late Arrival Time Improver Agent*: This agent changes the plans of trains with late arrival times.
    f. *Early Arrival Time Improver Agent*: This agent changes the plans of trains with early arrival times.

  The system now has 27 agents, including the three above.

- Continued testing on the BNSF data. Testing consisted of running different combinations of agents and with different frequencies of application to generate potential solutions. The goal is to both improve the planning capability of the system and to study the computational behaviors of different combinations of agents.

- Added end of train exit time to the movement model. Makes it a more realistic movement model. Also fixed the initial and final velocity calculation of the movement model.

Progress for January, 2000

The work in January consisted of the following:

- CSX line data for analysis. We obtained data from CSX for one of their lines, which we will run the OTP on. Their track database is configured differently than the BNSF database, which necessitated changes to the program reading it in.

- Detailed Design Document. The detailed design document for the Alpha Demonstration release of the OTP was started this month and is scheduled for completion in February (being added to the high-level design document completed in December).

- Preparation for working with George Washington University: We will be subcontracting software agent algorithm development work to George Washington University (GWU) during most of this year (kickoff is scheduled for February 10th and 11th), which required putting together materials and a working system so that the GWU team can get started.

  - Continued testing on the BNSF data. Same goals of tuning the system and studying the performance of the A_Teams problem solving agents. A composite of the results will be included in the final report, which is due next month (March).

  - General debugging of the OTP software.

  - Re-writing particular parts of the code to make it more efficient with respect to memory usage and some speed of processing.

- Started work on System Architecture Document.

Work Planned for February, 2000

The work planned for February is as follows:

- Work with GWU. The team at GWU will become familiar with the OTP in its current state and begin work about mid-February. US&S project personnel will interface with the team on a regular basis, primarily through email and telephone, but will meet face-to-face approximately once a month.

- Continue to run the OTP on the BNSF track scenarios to fine tune and debug the system.

- Apply the OTP to the CSX track database using the train schedules provided by CSX. The plan is to demo the results to CSX sometime in March.

- Complete the detailed design document.

- Modify the Dispatcher Component algorithm to make it execute faster. Presently is has an n2 time complexity, which should be reduced to nlog(n) or n.

APPENDIX 5

University Participation:

It was originally intended to subcontract with two universities for this phase of the project: George Washington University (GWU) and the University of Alabama (UA). Due to the relatively short duration of the project and the time required to put together agreements and, subsequently, set up a development environment with the universities, GWU was on board and working only for the latter part of January and the month of February (through the end of this phase – Phase 1 – of the project). UA did not participate. GWU continues to work in Phase 2, and it is planned that UA will have an agreement in place toward the end of Phase 2 and continue into Phase 3.

The following is a summary of GWU's work during Phase 1. The time was used primarily for training, to work out a plan or statement of work and to set the GWU lab up with the requisite software and development environment. They were asked to complete the following three related tasks:

1. Develop new agents for improving the performance of the OTP;
2. Develop a process agent that will decide which agents to apply to a solution and when (to improve performance);
3. Perform an extensive analysis of the application of problem-solving agents to different problems and summarize the results.

All these tasks, though begun at the very end of Phase 1, are currently ongoing (post Phase 1).

Optimizing Train Performace through Advanced Planning and Integrated Data Recording Systems (Final Report), Transportation Safety Research Alliance, 2000 -06-Signals, Control & Communications