

Chapter 9

Extended Example: Distributed Train Scheduling

In this chapter I present a computational study of an auction-based method for decentralized train scheduling.¹ Auction methods are well suited to the natural information and control structure of modern railroads. I assume separate network territories, with an autonomous dispatch agent responsible for the flow of trains over each territory. Each train is represented by a self-interested agent that bids for the right to travel across the network from its source to destination, submitting bids to multiple dispatch agents along its route as necessary.

The natural separation of track control across multiple dispatch agents precludes a combinatorial auction for the entire problem. Instead each individual dispatcher runs an auction for the right to travel across its territory, and trains must solve a coordination problem to receive compatible entry and exit times across their complete route. Trains bid for the right to enter and exit a territory at particular times. The dispatcher agents selects bids to maximize revenue in each round. Feasibility requires that there is a safe schedule for trains over the dispatcher's region given the bid times. As such, the scheduling problem lies outside of the standard combinatorial allocation problem.

An additional difficulty in this domain is presented by the continuous time dimension to a bid. Instead of imposing a finite time discretization on the system I provide an expressive bidding language that allows trains to bid for the right to enter and exit within *ranges* of times (e.g. “arriving no later than 12 pm”). Prices are approximated with a finite grid of (entry, exit) pairs, and updated by dispatcher agents with *i*Bundle-style price update rules.

¹This chapter draws a lot of material from Parkes & Ungar [PU01].

Computational results on a simple network with straight-forward best-response bidding strategies demonstrate that the auction computes near-optimal system-wide schedules. In addition, the method appears to have useful scaling properties, both with the number of trains and with the number of dispatchers, and generates less extremal solutions than those obtained using traditional centralized optimization techniques.

9.1 Introduction

Auction-based scheduling methods are well-suited to the decentralized information and control structure of modern railroads. The flow of trains over a railroad network is not controlled by a single centralized scheduler, but rather by the joint decisions of a number of largely autonomous dispatcher agents, each responsible for a local track territory. In addition, trains are operated by competing companies, each of which would prefer for their trains to run on-schedule even if the trains of other companies must wait. Real train drivers receive bonuses for on-line arrivals, and have private information about repair schedules, etc.

Auction-based methods fill two important needs. First, they respect the natural autonomy and private information within such a distributed system. Secondly, they can provide incentives for trains to reveal truthful information (indirectly, via bids) about their values for different schedules. In a naive central implementation, a self-interested train with private information about its time constraints, value, and costs, cannot be expected to act truthfully, but rather to misrepresent this information if it will improve its own schedule in the system-wide solution.

The train scheduling problem that I address in this chapter falls within a hierarchy of interrelated train scheduling problems; see [KH95] for a recent survey. I assume that all *strategic planning*, i.e. deciding on train routes and assigning values, times, and costs, is already completed. The input is a set of trains, each with a defined routes over a track network, a value for completing its journey, and an optimal departure and arrival time and cost function for off-schedule performance. The system-wide problem is to compute a robust and safe schedule for the movement of trains over the network to maximize the total cost-adjusted value over all trains.

In constructing an optimal schedule I depart from earlier models for automatic train

scheduling that used fixed-priority scheduling rules.² These early models have been criticized for a “hurry up and wait” approach [KHC91], with high priority trains moved down lines as fast as possible, possibly causing problems and inefficiencies at yards further down the line. I build instead on the *spacing models* of Kraay *et al.* [KHC91], which control the speed of trains in finding optimal schedules.

The auction design has each dispatcher agent running a separate auction, for the right to enter and exit its territory at particular times. There are necessarily *multiple* auctions, to respect the autonomy of individual dispatchers to make local decisions. All auctions terminate simultaneously, when there is quiescence across the system.

A train agent must bid for pairs of entry and exit times across multiple dispatchers to complete its journey, which presents a coordination problem. The exit time from one dispatcher must be early enough to allow the train to enter the next dispatcher on its route at the required entry time. Iterative auctions (as opposed to sealed-bid auctions) allow trains to adjust towards a good solution, and should help to solve this coordination problem. In addition, trains can submit bids for *sets* of times, i.e. “I want to enter your territory at any time after 10 am, but leave no later than 1.30 pm, and my maximum travel speed is 100 km/hr.” This constraint-based bidding language is a concise way to handle the continuous time attribute of a bid without imposing an explicit discretization on time.

In each round the auctioneer computes the set of bids that maximize revenue, subject to the constraint that there must be a safe schedule for trains given the entry and exit times in accepted bids. The winner-determination problem is solved without a discretization on time, formulated as a mixed integer program.

Although there is no explicit discretization on time imposed on agents’ bids, the *prices* in the auction are maintained over a discrete price lattice. The lattice maintains prices on *pairs* of entry and exit times, and prices are computed on-the-fly for any pair of entry-exit times based on the closest lattice points. The discrete price lattice *does not* restrict the times that can receive bids, but rather provides an approximate method to maintain prices. Prices are increased across rounds with an *iBundle* style price-update, i.e. based on the bid prices from unsuccessful agents.

²Priority-based dispatching is still used in most North American railroads; high priority trains are generally not delayed even if they are running early, while low priority trains are delayed even if they are running late [Hal93].

Experimental results compare the quality of schedules computed in the auction-based method with schedules computed under a traditional centralized optimization approach. In order to make a fair computational comparison across the methods, the global scheduling problem and the winner-determination problems are both formulated as (closely related) mixed integer programs, and solved with CPLEX— a standard mixed integer programming software package. The experiments compare centralized solutions (with complete information about agents’ preferences) with auction-based solutions, on a set of stochastic problem instances.

In assessing the performance of the auction-based method I make a reasonable assumption about agent bidding strategies, i.e. that agents follow a *myopic best-response bidding strategy*, and submit bids to maximize value given the current ask prices.

The computational results demonstrate that the auction-based method can generate *better* schedules than the centralized method, and in less time. Moreover, the auction-based method appears to have good scaling properties with the number of agents and dispatchers, at least for the auction parameters selected in the tests (e.g. price update speed, time in each round to solve winner-determination, etc.)

Further experimentation is required to make a full assessment of the auction-based method’s computational properties. I expect that the performance of the simple myopic bidding strategy might begin to fall-off as the number of dispatchers continues to increase, as agents’ bid coordination problem becomes more difficult. The myopic bidding approach, in which agents bid across all dispatchers simultaneously in response to current prices can leave agents “exposed” to times that they cannot fit with times from other dispatchers. Agents would require alternative more sophisticated bidding strategies in these cases, to avoid this exposure problem.

The outline of the rest of this chapter is as follows. In Section 9.2 I define the *global train-scheduling problem*, and formulate a mixed-integer program model under assumptions of centralized information about the local values and cost functions of each train. In Section 9.3 I describe the key elements of the *auction-based solution*: the bidding language, price-updates, winner-determination, termination conditions, and bidding rules. Section 9.4 formulates the bidding problem for a train agent in the auction as a shortest-path problem, which is solved using dynamic-programming. Finally, Section 9.5 presents experimental results over a set of stochastic train scheduling problems.

9.2 The Train Scheduling Problem

Each train is assumed to have a *source* and *destination* node, a value to complete its journey, and a cost for off-time departure and arrival. The global objective is to find a safe schedule that maximizes the total *net* value, the total value minus cost of delay across all trains that run. I introduce a novel mixed-integer programming (MIP) formulation that allows trains to be dropped when necessary, i.e. to allow other high-valued trains to run on-time. A very similar formulation is adopted for the winner-determination problem in the auction (see the next section).

9.2.1 Track Network: Topology and Constraints

In modeling the train scheduling problem I make a number of simplifying assumptions, both about the network structure and about the types of interactions that are allowed between trains.

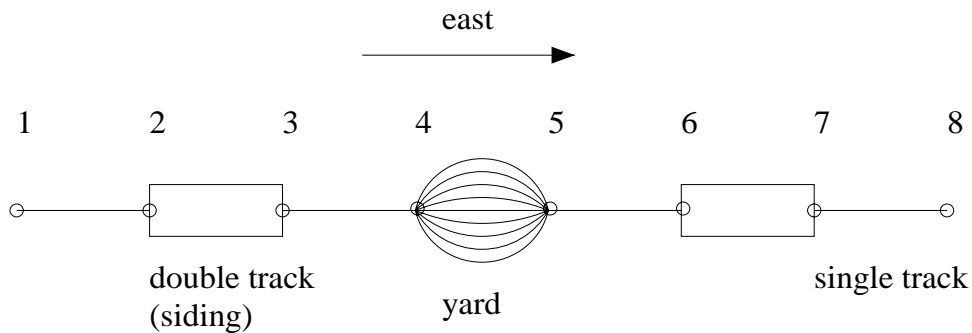


Figure 9.1: The train scheduling problem.

The key assumption is that of a *single line* operation— a sequence of *single-track*, *double-track*, or *yard* sections, separated by *nodes*. The structure is illustrated in Figure 9.1. The single-line operation simplifies the specification of the global train-scheduling problem and the winner-determination problem in the auction-based method. The single line assumption also allows train agents to restrict their attention to tradeoffs across multiple temporally different routes, ignoring alternate paths over the network. The same assumption is made across much of the train scheduling literature, for example in Kraay *et al.* [KHC91], Kraay & Harker [KH95], and Hallowell [Hal93]. Section 9.7 discusses a possible extension of this auction-based method to a multiple line network.

An interaction between a pair of trains may be a *meet* or a *pass*, and is associated with a network location and a time. A meet is when two trains traveling in *opposite directions* are at the same location at the same time. A pass is when two trains traveling in the *same direction* are at the same location at the same time.

The *feasibility* of a schedule for trains across a network is determined by the *safety* of meets and passes. This depends on the type of section:

- (S1) Any number of trains can meet and pass in yards.
- (S2) Any number of trains can meet on double-track sections, but no trains can pass.
- (S3) No trains can meet or pass on a single-track section.

In addition, a feasible schedule must maintain a *minimum separation distance*, Δ_{safety} , between trains on single and double track sections. This minimum separation distance requirement is waived for trains in yards (but not on exit into neighboring sections). Finally, no train can exceed either its maximum speed or the maximum safe speed on any section.

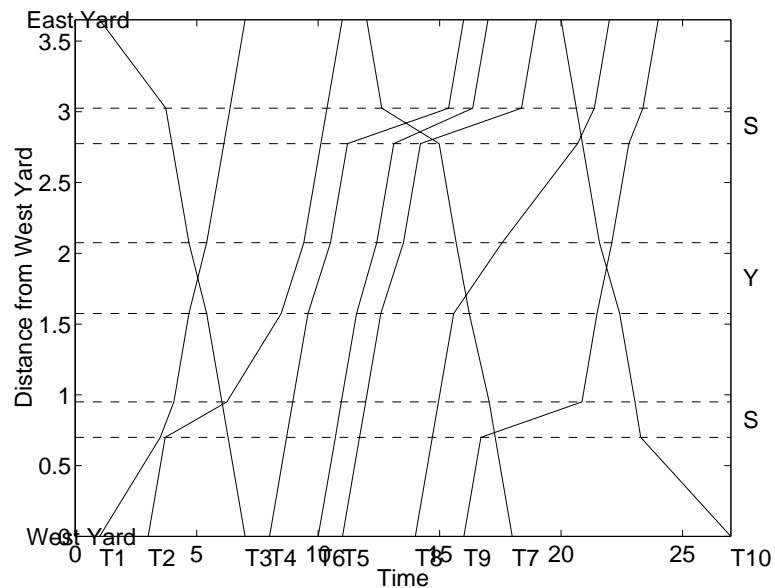


Figure 9.2: A safe train schedule.

Allowing trains to meet but not pass on double-track sections reduces problem-solving complexity because there are many ways for two trains to cross in the same direction but only a few ways for two trains to cross in opposite directions. Similarly, modeling infinite-capacity yards and double-track sections (sidings) is a simplifying assumption.

Figure 9.2 illustrates a safe train schedule for a network with the track topology illustrated in Figure 9.1. The exact parameters in this example are as described in the experimental results in Section 9.5; in this case with ten trains of which trains 1, 2, 4, 5, 6, 8, 9 run East. The sidings (or double-track sections) in this network are illustrated with an “S” on the right of the plot, the yard with a “Y”. Notice that trains meet at sidings and at the yard, but no trains meet or pass on the single track sections. In this example no trains pass in the yard, but this is not precluded in the model. Notice also that the trains remain the *safety* distance apart, and are expected to maintain constant speeds within each section.

9.2.2 Schedules

A schedule specifies the network position across time for each train in the system. It is sufficient to consider schedules in which trains travel at a constant speed with each section (the speed can vary from train to train and from section to section), by the following result:

LEMMA 9.1 *Any feasible schedule can be reduced to a feasible schedule where each train travels at a constant speed within each track section.*

The transformation that maintains feasibility is to hold times at nodes between track sections constant, and *smooth* the speed of each train between these points. The proof is quite straightforward— just show that the number of meets are the same for any speed profile consistent with the entry and exit points, and that the number of passes is (weakly) less when trains travel at a constant speed. I choose to ignore constraints on acceleration across sections.

This observation reduces the size of the search space in the scheduling problem, and simplifies the problem of finding optimal times for trains at the ends of each section.

9.2.3 A Mixed Integer Programming Formulation

Let \mathcal{I} denote the set of trains and \mathcal{N} denote the set of nodes between track sections. It is useful to view the network in a west–east orientation, with nodes ordered such that $j > k$ for $j \in \mathcal{N}$ further east than $k \in \mathcal{N}$. The trains are divided into a set $east \subseteq \mathcal{I}$ that travel west-to-east and $west$ that travel east-to-west. The nodes are labeled with

the type of section to the east, e.g. the section between node j and $j + 1$ is a yard if $j \in \text{yard}$, single-track if $j \in \text{single}$, and double-track otherwise. The minimum travel time for train i between node j and $j + 1$, its *free-running time* $r(i, j)$, is defined by the length of the section, the maximum speed of the train, and the maximum safe speed over the section. This is the time to run from west-to-east for a train $i \in \text{east}$, and from east-to-west otherwise. Later, when I formulate the MIP for winner-determination I will leave this information implicit in the bidding language to simplify the presentation.

Each train $i \in \mathcal{I}$ has a *source node* and optimal departure time, $(s(i), t_s^*(i))$, a *destination node* and optimal arrival time, $(d(i), t_d^*(i))$, a *value* $V_i \geq 0$ for completing its journey, and a *cost penalty*, $\text{cost}_i(t_s, t_d)$, for off-schedule performance. Following [Hal93] we assume a linear additive cost penalty for each train. Given actual source t_s and destination t_d times for train i , the cost for off-schedule performance is computed as:

$$\text{cost}_i(t_s, t_d) = C_i |t_s - t_s^*(i)| + C_i |t_d - t_d^*(i)|$$

where $C_i > 0$ is train i 's *unit cost for off-schedule performance*.

This cost function assumes that performance is measured only on the basis of a train's time at its source and destination nodes. This is reasonable for a freight train with a single shipment to make, but less appropriate for a train that must make intermediate scheduled stops.

Given that it is sufficient to consider only trains that travel at a constant speed across each section (Lemma 9.1), we can specify a schedule with the time, $t(i, j)$, of each train i at node j . A train can be dropped from the schedule. Let $y(i) \in \{0, 1\}$ equal 1 if train i is not dropped from the schedule, and 0 otherwise. Let $\Delta_{\text{source}}(i)$ and $\Delta_{\text{dest}}(i)$ denote the *absolute* error in departure and arrival time for train i at source node $s(i)$ and destination node $d(i)$. The system-wide objective is to maximize total value minus cost:

$$\max \sum_i V_i y(i) - \sum_i C_i \Delta_{\text{source}}(i) - \sum_i C_i \Delta_{\text{dest}}(i)$$

In congested networks with high cost penalties it can be better to drop low value trains in allow more trains to run on schedule and avoid cost penalties. Dropped trains neither achieve any value nor incur any cost penalties. This is achieved by a smart formulation of the method to compute arrival and destination errors, $\Delta_{\text{source}}(i)$ and $\Delta_{\text{dest}}(i)$.

The constraints make sure schedules are feasible, i.e. that a schedule is safe, trains are separated, and speed constraints are not violated. In the following (“the big M technique”),

M is a large positive number, used to make sure that dropped trains do not restrict schedules for other trains, to allow dropped trains to incur zero penalties, and to implement disjunctive logic constraints as a mixed-integer program.

Constraints (1a) and (1b) set the errors $\Delta_{\text{source}}(i)$ and $\Delta_{\text{dest}}(i)$ for train i :

$$\Delta_{\text{source}}(i) \geq |t(i, s(i)) - t_s^*(i)| - M(1 - y(i)) \quad \forall i \in \mathcal{I} \quad (1a)$$

$$\Delta_{\text{dest}}(i) \geq |t(i, d(i)) - t_d^*(i)| - M(1 - y(i)) \quad \forall i \in \mathcal{I} \quad (1b)$$

The absolute value constraint can be implemented by writing two greater than constraints, one for the positive term and one for the negative term.

Notice that if $y(i) = 0$ for train i then $\Delta(i) = 0$ is a solution, and we count no penalty for dropped trains. This avoids requiring non-linear terms, such as $C_i \Delta_{\text{source}}(i) y(i)$, in the objective function.

Constraints (2a) and (2b) ensure consistency of travel times for trains, given free running time $r(i, j)$ for train i between node j and $j + 1$. Again, neither constraint is binding for a dropped train by the “big M” formulation.

$$t(i, j + 1) \geq t(i, j) + r(i, j) - M(1 - y(i)) \quad \forall i \in \textit{east}, \forall j \in \mathcal{N} \quad (2a)$$

$$t(i, j + 1) \leq t(i, j) - r(i, j) + M(1 - y(i)) \quad \forall i \in \textit{west}, \forall j \in \mathcal{N} \quad (2b)$$

The zero-one variables $gap(i, i', j)$ make sure that trains are a safe distance apart at all times; $gap(i, i', j) = 1$ iff train i trails train i' by at least time *safety* at node j . The “big M” technique is used to constrain a train to be either more than *safety* ahead or more than *safety* behind another train. Note that constraint (3b) is true whenever at least one of the trains is dropped, so that the times on dropped trains are not constrained.

$$t(i, j) - t(i', j) + M gap(i, i', j) \geq \textit{safety} \quad , \forall j \in \mathcal{N}, \forall i, i' \in \mathcal{I} \quad (3a)$$

$$\begin{aligned} t(i', j) - t(i, j) + M(1 - gap(i, i', j)) + M(2 - y(i) - y(i')) \\ \geq \textit{safety}, \forall j \in \mathcal{N}, \forall i, i' \in \mathcal{I} \end{aligned} \quad (3b)$$

The zero-one variables $after(i, i', j)$ indicate whether train i arrives at node j after train i' ; $after(i, i', j) = 1$ if train i is after train i' at node j . This indicator variable is set by constraints (4a) and (4b) to be consistent with the times defined by variables $t(i, i', j)$. A dropped train can assume the same ordering with respect to all trains, allowing them

to trivially satisfy (4c) and (4d).

$$t(i, j) - t(i', j) \leq M \text{after}(i, i', j) \quad , \forall j \in \mathcal{N}, \forall i, i' \in \mathcal{I} \quad (4a)$$

$$t(i', j) - t(i, j) - M(2 - y(i) - y(i')) \leq \\ M(1 - \text{after}(i, i', j)) \quad , \forall j \in \mathcal{N}, \forall i, i' \in \mathcal{I} \quad (4b)$$

Constraint (4c) captures the restriction that trains traveling in the same direction cannot pass on sidings or single-track sections. East-bound train i must remain after east-bound train i' at node j if it is behind train i at node $j + 1$ and the section between j and $j + 1$ is not a yard. Similarly for west-bound trains.

$$\text{after}(i, i', j) = \text{after}(i, i', j + 1) \\ \forall j \notin \text{yard}, \forall i, i' \in \text{east}, \forall i, i' \in \text{west} \quad (4c)$$

Finally, constraint (4d) captures the restriction that trains traveling in opposite directions cannot meet on single-track sections. If east-bound train i is after west-bound train i' at node j it must also have followed west-bound train i' at node $j + 1$ for single-track sections between j and $j + 1$, otherwise the trains were on the same single-track section at the same time and traveling in opposite directions.

$$\text{after}(i, i', j) = \text{after}(i, i', j + 1) \\ \forall j \in \text{single}, \forall i \in \text{east}, \forall i' \in \text{west} \quad (4d)$$

Taken together with the objective function, the constraints specify a mixed-integer program to solve the centralized train scheduling problem. The optimal solution specifies which trains are dropped (with $y(i) = 0$) and the times $t(i, j)$ for other trains at each node j in the network.

9.3 An Auction-Based Solution

In introducing the auction-based solution, let us assume that the track network is divided across dispatcher territories, with each dispatcher responsible for the local flow of trains. A separate dispatcher agent auctions the right to travel across each territory. Each train is associated with a train agent that places bids for the right to travel across a territory, and coordinates times across dispatchers on its route to achieve a good schedule. The

dispatchers have information about the local network topology, i.e. the location of the double-track network sections and the location of the yards.

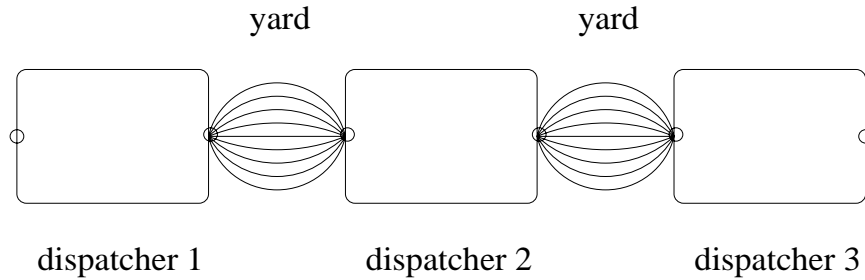


Figure 9.3: The dispatcher territory structure.

The decentralized problem structure is illustrated in Figure 9.3. I assume that dispatch territories are separated by neutral yards. This allows the safety constraints on meets and passes to be decoupled across dispatch territories because yards have infinite capacity and allow arbitrary meets and passes. The dispatcher on each side of a connecting yard must simply ensure that trains remain a *safety* distance apart as they enter and exit its territory.

9.3.1 Auction Innovations

The structure of the train scheduling problem requires a number of innovations in auction design:

- (1) A constraint-based bidding language to allow train agents to submit bids with continuous time attributes, and represent a choice set over different pairs of times.
- (2) An approximate representation of a continuous and non-linear price space. Prices are maintained over a discrete lattice with quotes computed on-the-fly for any pair of times.
- (3) An integer-programming method to compute feasible train schedules given entry and exit times in bids, and check conflicts across bids.

As noted earlier, there are multiple independent auctions, one for each dispatcher territory. This respects the decision autonomy of each dispatcher. Given that trains must receive compatible entry-exit times across multiple dispatchers, the auctions are necessarily iterative to allow train agents to coordinate their bids across multiple auctions. All auctions close simultaneously when bid quiescence is detected across the system.

9.3.2 Dispatcher Auction

Each dispatcher runs an ascending-price auction, maintaining *ask prices* for (departure, arrival) times at pairs of nodes, and a provisional schedule. The ask price is a *lower bound* on an acceptable bid price. A provisional schedule is computed in each round to maximize revenue, based on bids from train agents. Train agents can bid for *entry* and *exit* times in a territory, while the dispatcher agents have the flexibility to decide exactly how a train will run, so long as the schedule is consistent with those times. In the following I describe a single dispatcher auction in some detail.

Bidding Language

The bidding language is quite expressive. A train can bid to enter a territory at time t_{entry} and depart at time t_{exit} , and state whether each time is *fixed* or *flexible*. With a fixed time the train must enter (or exit) the territory at that exact time. With a flexible entry time, any time *after* t_{entry} is acceptable; with a flexible exit time, any time *before* t_{exit} is acceptable (subject to constraints on a train’s minimal travel time). Finally, a train agent can submit multiple bids to the same dispatcher, coupled with an “exclusive-or” constraint, to state that the dispatcher can accept any *one* pair of times.

Let \mathcal{K} denote the set of all bids received by a dispatcher, and $\beta(i) \subseteq \mathcal{K}$ the bids received from agent i . A set of bids from agent i in a particular round are all associated with a single entry node, $n_{\text{entry}}(i)$, a single exit node $n_{\text{exit}}(i)$, and true/false values $\text{fixed}_{\text{entry}}$ and $\text{fixed}_{\text{exit}}$ to state whether the times are fixed or flexible. Each individual bid $k \in \beta(i)$ specifies an entry time, $t_{\text{entry}}(k)$, an exit time $t_{\text{exit}}(k)$, and a bid price $p(k) \geq 0$.

Example:

Bid (5, 10, \$100) xor (7, 12, \$150) for entry node A and exit node B , with $\text{fixed}_{\text{entry}}$ but $\neg\text{fixed}_{\text{exit}}$, states that the train agent is willing to pay up to \$100 to enter at A at time 5 and depart before time 10, or up to \$150 to enter at time 7 and depart before time 12.

To keep the winner-determination problem tractable I also find it useful to restrict the number of bids that an agent can place in any round.³

³Experimentally, $B_{\text{max}} = 5$ appears to work well in many problems.

Winner-determination

The auction has multiple rounds. In each round the dispatcher solves the winner-determination problem, computing a provisional allocation to maximize revenue based on bids. The provisional allocation must be consistent with some feasible schedule.

The winner determination problem is formulated as a mixed-integer program that is very similar in form to that for the centralized train scheduling problem. However, it tends to be much easier to solve because the problem is restricted to the space of solutions compatible with the bids submitted by agents and the problem is for only a single territory.

As in the centralized problem, it is not necessary to select a bid from every train agent even if there is a feasible solution involving every agent. Sometimes a schedule with fewer agents will generate more revenue.

Borrowing as much from the earlier global MIP formulation (see Section 9.2.3) as possible, we introduce new zero-one variables $x(i, k) \in \{0, 1\}$ for agent $i \in \mathcal{I}$ and bid $k \in \beta(i)$, where $\beta(i)$ is the set of bids from agent i , with $x(i, k) = 1$ iff agent i 's bid k is in the provisional allocation. The linear objective function is:

$$\max \sum_{i \in \mathcal{I}} \sum_{k \in \beta(i)} p(k)x(i, k)$$

i.e. accept bids to maximize total revenue where $p(k)$ denotes the bid price of bid k from agent i .

Constraints (2a, 2b, 3a, 3b, 4a, 4b, 4c, 4d) are retained from the MIP of the global train scheduling problem, with train times computed on the basis of bids from agents. Allowing for flexible bid times, we write:

$$t(i, s(i)) = \sum_{k \in \beta(i)} t_{\text{entry}}(k)x(i, k) \quad , \text{ if } \textit{fixed}_{\text{entry}}(i) \quad (1a')$$

$$t(i, s(i)) \geq \sum_{k \in \beta(i)} t_{\text{entry}}(k)x(i, k) \quad , \text{ otherwise}$$

$$t(i, d(i)) = \sum_{k \in \beta(i)} t_{\text{exit}}(k)x(i, k) \quad , \text{ if } \textit{fixed}_{\text{exit}}(i) \quad (1b')$$

$$t(i, d(i)) \leq \sum_{k \in \beta(i)} t_{\text{exit}}(k)x(i, k) \quad , \text{ otherwise}$$

$$\sum_{k \in \beta(i)} x(i, k) \leq y(i) \quad (1c')$$

for every train $i \in \mathcal{I}$.

The source node $s(i)$ is the entry node $n_{\text{entry}}(i)$, and the destination node $d(i)$ is the exit node, $n_{\text{exit}}(i)$. Constraints (1a') constrain the schedule for train i to an entry time consistent with its bid, similarly for (1b') for its exit time. Constraint (1c') ensures that at most one bid is accepted per agent (exclusive-or bid constraints), and that no bids are accepted from dropped trains.

Price Updates

Each dispatcher agent maintains ask prices on a discrete price lattice, but without imposing a discretization on the times that a train agent can bid. Ask prices represent a lower-bound on the price that a train agent must bid to have any chance of success in the auction, but do not guarantee that a bid will be successful. The lattice structure is used to approximate a continuous non-linear price space. A smaller unit of discretization leads to a higher computational cost and slower convergence but perhaps to a higher schedule quality. An alternative price structure might explicitly maintain unsuccessful bids and compute ask prices on-the-fly exactly.

Dispatcher agents provide a price-query function for train agents, to allow train agents to compute the ask price for any pair of times on-the-fly. Given a bid for a pair of fixed times the ask price is determined as the price of the nearest point in the lattice. Given a bid containing one or more flexible times, the ask price is determined as the *minimal* price over all lattice points with *consistent* times. The interpretation of consistent is the natural one, a lattice point is consistent with a bid if it satisfies all constraints on entry and exit times.

The *minimal* operator, used to compute ask prices across sets of lattice-times consistent with flexible bids, provides useful price semantics:

$$p(k_1) \geq p(k_2), \quad \text{if } k_1 \subseteq k_2$$

for bids k_1 and k_2 if all times consistent with k_1 are also consistent with k_2 , and the bids have both flexible entry and exit times. This follows immediately from the minimal operator. The minimal price over the set of times consistent with k_2 can be no larger than the minimal price over the set of times consistent with k_1 , because the set of consistent times for k_1 is a subset of the times for k_2 . The relationship allows a train agent to prune its local search when considering different times in its best-response strategy.

Note, however, that it is *not* necessarily the case that $p(k_1) \geq p(k_2)$ for a pair of *fixed* times k_1 and k_2 with the entry time of k_1 later than k_2 and the exit time of k_1 earlier than k_2 ; for example, bid k_2 might hit a *safety* conflict with an accepted bid from another agent, which bid k_1 might escape. In comparison, a flexible bid, k_2 , would escape the safety problem whenever k_1 escaped the safety problem.

An unsuccessful bid increases the price on its nearest lattice point, or multiple consistent lattice points in the case of an unsuccessful constraint-based bid. For each bid in an unsuccessful exclusive-or set of bids we update the ask price on all grid points consistent with the bid times, as follows:

(a) if the bid is for fixed times then find the point on the lattice closest to the bid, otherwise find the set of consistent points,

(b) update the ask price at that lattice point to ϵ above the unsuccessful bid price, or on all lattice points in the set, where $\epsilon > 0$ is the minimal bid-increment in the auction.

The structure of this price-update is motivated by price updates in *iBundle* (Chapters 4–7), which increases prices on bundles of items by ϵ in response to unsuccessful exclusive-or bids.

The price is also increased because of bids submitted by train agents in the provisional allocation that receive the same pair of times from the last round of the auction but are trying to shift away from that allocation. I allow a train agent to indicate when it is merely repeating a bid because that is required under the auction rules, rather than because it really wants that pair of times.

Finally, an “infinite” value is used to represent the case that the *safety* condition will be violated with any bid close to a particular grid point. This is used as items are sold to train agents at particular times (under the continuous clearing rules, see below), to move a train’s bid focus away from a time that cannot be accepted at any price.

Bidding Rules

The bidding rules are quite simple: (1) an agent must bid at least the ask price for a pair of times, computed as appropriate for the flexible/fixed attributes of the bid; and (2) an agent must repeat a bid that supports a pair of times it receives in the current provisional allocation. The two rules ensure that progress is made across rounds of the auction towards a solution.

Clearing and Termination Rules

Every dispatcher auction terminates simultaneously when no new bids are placed by any train agent to any dispatcher agent. In addition, the auctions have a continuous clearing rule, in which a dispatcher commits to a particular pair of times for an agent that has received the times in the provisional allocation for more than a fixed number of successive rounds, T_{clear} .

Continuous clearing helps to reduce bidding complexity, committing trains to particular times (although they can continue to bid for alternate times at an additional cost), and focusing their search. A countervailing force is that early commits can also lock-in a particular pair of times too quickly when continued search might find a better solution.

The MIP formulation for winner-determination is easily adapted to include committed times. These times can be represented with bids from a dummy agent, with acceptance of those bids forced within the MIP solution method.⁴

Speeding-up Winner-Determination

One useful technique to speed-up winner-determination in iterative auctions is to maintain solutions from previous rounds in a cache, indexed against the bids that were submitted. The cache can be checked for a solution before solving the mixed integer program.

A *hit* in the cache depends on the bid times and fixed/flexible attributes in agents' bids. We need a couple of definitions.

DEFINITION 9.1 [less flexible] A bid k_1 for a pair of times at a price is weakly *less flexible* than another bid k_2 if all times that are consistent with bid k_1 are also consistent with bid k_2 , and if the price on bid k_1 is no greater than the price on bid k_2 .

In other words, a bid k_1 which is less flexible than another bid k_2 will never be accepted in the provisional allocation when bid k_2 is not accepted.

DEFINITION 9.2 [supports] A set of bids *support* a bid k' if one or more of the bids in the set is (weakly) more flexible than bid k' .

In other words, the agent that submitted the set of bids would have been happy to submit a successful bid k' .

⁴The flexibility of mixed-integer program formulations of winner-determination problems was previously noted by Andersson *et al.* [ATY00].

Given this, a set of bids received by a dispatcher *match* a set of cached bids if there is some permutation of the new bids with a set of bids in the cache that satisfies:

(1) if bids from agent i and are successful in the cached solution, then the new bids from i must *support* the time corresponding to the successful bid, and be (weakly) *less flexible* than the other times in the cached bid.

(2) if bids from agent i are unsuccessful in the cached solution, then the new bids from i must all be (weakly) *less flexible* than the old bids.

It is quite straightforward to understand why this cached solution is an optimal solution with the new bids. Basically, no agents that are in the provisional allocation in the cached solution submitted stronger bids on anything except the successful bid, and no unsuccessful agents submit any stronger bids.

As a special case, a match also occurs when the new bids from every agent supports a successful bid in a cached solution, and the prices on all the rejected bids from each agent are no greater than on the accepted bids. In this case the rejected bids from an agent do not need to be less flexible than the other times in the cached bid.

9.4 The Bidding Problem

Recall that each train $i \in \mathcal{I}$ has value V_i to complete its journey, subject to a cost $cost_i(t_s, t_d)$ for off-schedule performance, given optimal source and destination times $t_s^*(i)$ and $t_d^*(i)$ and actual times t_s and t_d . The bidding problem is to purchase the right to travel across the network from source to destination at minimal total cost, where cost is the sum of the price it pays in each auction and the cost of off-schedule performance. In addition, if this cost is greater than the train's value then it would prefer to drop out completely.

The bidding problem is difficult for two main reasons:

- (a) *coordination*: a train agent must bid with multiple dispatchers when its route spans more than one territory.
- (b) *dynamic pricing*: a train agent cannot know the prices at which the auction will clear.

Each train agent is assumed to follow a myopic best-response bidding strategy, bidding for the schedule that minimizes total cost given the current ask prices. Myopic best-response provides a good starting point to analyze the performance of the auction method. It would be interesting, but probably quite difficult, to also consider the effect of fully

strategic agent behavior on the quality of scheduling solutions. It is possible that a train agent can achieve a better outcome by anticipating the bids of other agents and considering the effect of current bids on future prices.

9.4.1 Myopic Best-response Bidding Strategy

The myopic best-response bidding problem can be formulated as a *shortest weighted path problem*. The edges in the graph correspond to pairs of entry-exit times at each dispatcher, fixed or flexible as appropriate. Edges are connected if the exit time on one edge is *compatible* with the entry time on the next edge. The compatibility requirement here simply requires that there is enough time for the train to leave the first dispatcher territory at the exit time, travel across the connecting yard, and enter the second dispatcher territory at the entry time. The cost associated with an edge represents the sum of the current ask price, and any cost for off-schedule arrival or departure if the dispatcher is at the source or destination of a train's route.

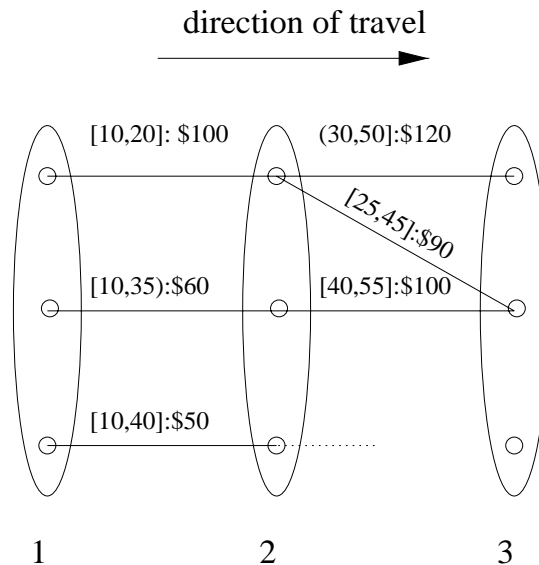


Figure 9.4: The myopic best-response bidding problem.

Figure 9.4 illustrates a partially-completed graph, with nodes 1 , 2 and 3 representing the yard to the West of dispatcher d_1 , between d_1 and d_2 , and to the East of d_2 respectively. For example, the times on the edges between yards 1 and 2 denote entry-exit times for dispatcher d_1 , together with the ask price for those times, summed with any additional cost of early/late departure if dispatcher d_1 is also the first dispatcher in the train's itinerary.

Formulation

A train’s best-response, taking current prices as fixed, is to select a path from source to destination with minimal total cost (or no path at all in the case that the minimal cost is greater than its value for completing the journey).

Given a set of dispatchers, \mathcal{D} , let (d_1, \dots, d_n) represent the dispatchers on the route of a particular train. Let $C_{1 \rightarrow n}^*(t)$ denote the minimal total cost to enter dispatcher d_1 no earlier than time t , travel from d_1 to d_n , and exit from dispatcher d_n . This cost represents the cost of the best schedule, given current ask prices and the train’s costs for off-schedule performance. The solution to C^* can be computed as a recursive relationship:

$$C_{j \rightarrow n}^*(t) = \begin{cases} \min_{\tau > t} \left(c_j(t, \tau) + C_{(j+1) \rightarrow n}^*(\tau) \right) & , \text{ if } j < n \\ \min_{\tau > t} c_j(t, \tau) & , \text{ if } j = n \end{cases}$$

where $c_j(t_1, t_2)$ is the cost to enter dispatcher j at time t_1 (or no earlier than t_1 in the case of a flexible bid time), and exit dispatcher j at time t_2 (or no later than t_2 in the case of a flexible bid time), computed as the sum of the price for times and any additional cost penalty for off-schedule performance if dispatcher j is at the end of the train’s route.

The price is the ask-price if the agent is not yet committed to the good (i.e. it has not cleared), or *zero* otherwise (in which case the price represents a sunk cost). Trains treat offered items in the same way as any other item, making an assumption that it can move away from such an item if necessary without becoming exposed.

Trains consider flexible bid times in the case of non-extremal nodes, but fixed times at source or destination because a cost is incurred for any deviation from optimal departure and arrival times. The intermediate time τ represents the time to cross from dispatcher d_j to d_{j+1} . In this description I have finessed the detail about the time to travel across yards between dispatch territories, which is simply incorporated into the recursion.

Dynamic Programming

Dynamic programming solves this shortest-path formulation of the myopic best-response bidding problem, computing the best solution over a fixed set of time points selected by the train agent, and working from dispatcher d_n to d_1 , pruning any dominated solutions (for example higher cost edges with earlier entry times). In related work, Boutilier *et*

al. [BGS99] proposed a dynamic programming algorithm for agent bidding strategies in *sequential* auctions with complementarities.

We implement the following algorithm:

- (1) determine the maximal *compatible* set of current offers and sold items, that leave enough time for travel across the dispatch territories.
- (2) for each maximal set, use dynamic programming to determine minimal-cost routes in the gaps of the schedule. The gaps are contiguous sequences of dispatchers for which the train is not currently holding a suitable pair of entry and exit times. Flexible time constraints are selected for all entry and exit times except those representing a train's initial departure or arrival time, at which nodes the train is not willing to be flexible.
- (3) fill the gaps and select the solution with the lowest total cost (including the cost for current offers/sold items used in the solution).

Compatible offers allow the train to complete its journey, including the time to travel across dispatchers and across connecting yards, consistent with all entry and exit times in the offers. A maximal set of compatible offers is a set of compatible offers with maximal cardinality, given the current offers and the train's free-running travel times.

Whenever a gap occurs at the start or end dispatcher on a train's route the train agent also considers tradeoffs between bid price and the cost of off-schedule performance for off-time departure and/or arrival times.

This method includes a bias in favor of solutions compatible with times the train receives in the current provisional allocations. This is reasonable, given that the ask prices represent a lower-bound on what might be a successful bid price but provide no guarantees that a bid will actually succeed. That an agent currently receives a pair of times conveys useful information about the fit of those times with bids from other agents.

Finally, a train will submit as many bids that are consistent with its selected solution as possible, making use of multiple exclusive-or bids with individual dispatchers, and using constraints on times to submit multiple bids without compromising the solution. This increases its own likelihood of success, and also helps the dispatchers to coordinate joint search across multiple agents.

9.5 Experimental Results

The performance of the auction-based solution is compared with a centralized solution in networks consisting of *linear chains* of dispatcher territories. Both the global MIP formulation and the MIP for winner-determination in each round of the auction are solved with CPLEX, commercially available linear-programming based branch-and-bound optimization software. The rest of the code (myopic best-response, price-updates, etc.) was written in C++.

9.5.1 Dispatcher model

Each dispatcher territory has the same simple network structure, as depicted in Figure 9.5. The total distance is 157.5 km, consisting of a single-track section followed by a double-track section (siding) followed by a single-track section. Dispatcher territories are connected together with yards. Trains have a maximal speed of 100 km/hr over single- and double-track sections, and 1 km/hr within a yard. For simplicity, I model maximal speed as 100 km/hr throughout the network and re-scale yards from size 0.5 km to a model length of 50 km. The minimum separation distance, Δ_{safety} , between trains is 20 km, corresponding to a run-time of 0.2 hrs at maximum speed.

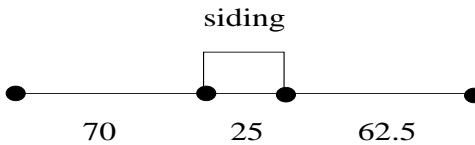
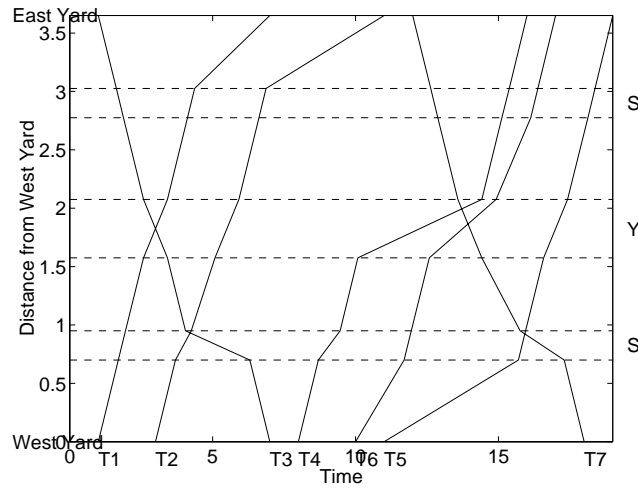


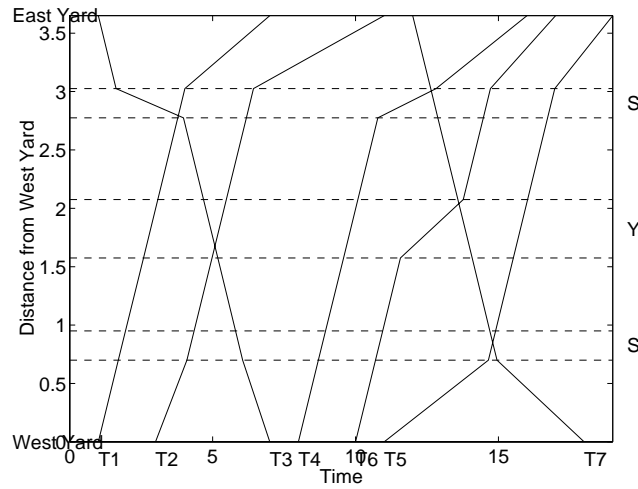
Figure 9.5: The network structure for a single dispatcher, with distances of each section (in km).

9.5.2 Example Problem

Consider a problem with a chain of two dispatchers, and 7 trains, each with value \$200 and cost \$50 per hour of delay. Trains 1, 2, 4, 5, and 6 run east with optimal departure and arrival times (in hours) of $\{ (1, 7), (3, 11), (8, 16), (11, 19), (10, 17) \}$, while trains 3 and 7 run west with optimal times of $\{ (1, 7), (12, 18) \}$. Given a maximal speed of 100 km/hr the *free-running time* of a train across the network is 3.66 hrs. This is how long a train would take with no delays.



(a) Auction solution.



(b) Centralized solution.

Figure 9.6: Example: 7 trains, 1 dispatcher territory. Distance in 100's of kms, time in hrs.

The auction-based and centralized solutions to this problem are illustrated in distance-time charts in Figure 9.6. Both methods find optimal solutions, with value \$1400 (all trains run on-time). This problem is quite under-constrained, with a number of possible optimal schedules.

Notice that the auction-based solution is less extremal than the centralized solution. This is quite typical, a result of the fact that train agents tend to bid less extreme times than those selected with a global LP-based branch-and-bound method such as CPLEX, and achieve a more evenly paced schedule from source to destination. We would expect this property to make auction-based solutions more robust against unexpected minor delays during the execution of a schedule.

9.5.3 Results

The experimental results are for problems with between 2 and 4 dispatcher agents, and between 5 and 15 train agents, and with linear networks—formed from dispatcher territories as shown in in Figure 9.5 and connected with yards. The auction algorithm was parameterized as follows: at most 5 bids per-agent in each round, at most 240 seconds to solve winner-determination in each round (with the best feasible solution used if the optimal solution is not found), and a minimal price increment of \$25. The price lattice was maintained over points with a granularity of 0.2 hrs, and I used a time interval size of 0.3 hrs in the trains' dynamic-programming algorithm.

Problem Generator

A stochastic model is used to generate a set of problem instances. The model is loosely based around instances in Kraay et al. [KHC91]. I refer the reader to Hallowell [Hal93] for an account of other interesting train scheduling problem sets in the literature. The problem sets are parameterized by constants: $\text{prob}(E)$, μ_V , σ_V , μ_C , σ_C , dep_{\max} , μ_{slack} and σ_{slack} , and the number of train and dispatcher agents, as described below

All trains travel from end-to-end over the network, and travel East with probability $\text{prob}(E)$. A train's value is selected from a normal distribution, $V_i \sim N(\mu_V, \sigma_V)$, with mean μ_V and standard deviation σ_V , and its unit cost C_i for off-schedule performance is normally distributed $N(\mu_C, \sigma_C)$. The optimal departure time for a train, $t_d^*(i)$, is uniformly distributed, $t_d^*(i) \sim U(0, dep_{\max})$. Finally, a train's optimal arrival time, $t_s^*(i)$, is computed

num dispatchers	Model Size								
	2			3			4		
num trains	5	10	15	5	10	15	5	10	15
Global-time (s)	97	1438	2022	1161	2442	2495	886	2378	3155
Global-value (\$)	895	1699	2097	854	1561	1177	898	1106	1132
Auc-time (s)	15	792	2568	15	1192	2039	26.9	944	2448
Auc-value (\$)	850	1893	1737	842	1855	2632	768	1832	2162
Agent time (s)	0.4	0.6	2.7	0.9	2.2	3.3	1.9	4.4	8.8
Revenue (\$)	315	698	1045	470	1030	1690	700	1365	2142
num rounds	12	13	25	13	16	18	16	16	23
Cache hit (%)	60	50	30	61	50	47	50	52	37

Table 9.1: Comparative performance: Auction vs. Centralized methods.

so that the *relative slack*, i.e. (available time - free-running time) / free-running time, is normally distributed with mean μ_{slack} (%) and standard deviation σ_{slack} (%).

The complexity of a train-scheduling problem depends on many factors, including the slack time available to each train, the network section types, and the number of “cross-overs”. A *cross-over* is counted whenever two trains traveling on-time must cross at some point in the network. As I scaled the problems, with more train agents and more dispatcher agents, I adjusted the dep_{max} parameter to maintain the same number of average cross-overs per-agent, in an effort to maintain a similar problem complexity. An appropriate dep_{max} value was computed off-line for each problem size to achieve this property. Without this adjustment adding more trains and more dispatchers increases the number of cross-overs and makes problems much more difficult to solve. My goal was to capture the scaling properties of the two solution methods with the same per-train and per-dispatcher complexity.

Results

In the experiments the stochastic problem generator is parameterized with: $\text{prob}(E) = 0.7$, $\mu_V = \$200$, $\sigma_V = 50$, $\mu_C = \$100$, $\sigma_C = 25$, $\mu_{slack} = 100\%$, $\sigma_{slack} = 25\%$, and set dep_{max} to give average cross-over complexity of 2 per-train. I generated 10 problem instances for each problem size, as the number of train agents were increased from 5 to 15 and the number of dispatchers from 2 to 4.

Table 9.1 presents the results. The computation time of the centralized method is bounded at 3600 secs, at which point we take the best available solution. CPLEX also

ran out of memory (at 200 MB) while solving a few of the centralized problems, stopping before 3600 secs but without an optimal solution. The computation time of the auction is the total winner-determination and price-update time, summed over all rounds and all dispatchers.

Notice that the quality of the schedule computed in the auction dominates that from the centralized solution in hard problems, as the number of dispatchers and/or the number of agents increase—the auction computes a higher quality solution in less time. The auction also appears to have reasonable scaling properties, with the number of train agents and in particular with the number of dispatchers. We believe that this advantage arises because the auction is an effective method to decompose the computational problem across dispatchers, with critical (and difficult) computation localized to a single critical dispatcher. It is also noteworthy that the train agents' myopic best-response algorithm is quite fast, indicating that it would be interesting to experiment with a smaller discrete time step. Notice also that the simple cache proves quite effective, finding the optimal solution around 50% of the time.

More experiments are required, both to better understand the average-case scaling properties of the auction and also to look more deeply at agent strategies. Based on these limited results my conjecture is that the average-case run time in the auction scales *quadratically* with the number of train agents, and perhaps *sub-linearly* with the number of dispatch territories. It would be interesting to develop an analytic model to confirm/reject this conjecture.

In terms of agent strategies, I suspect that some agents purchase times that they cannot use as the number of dispatchers increases and as the bid coordination problem gets more difficult. This belief is based on a comparison of the revenue with value in the auction, see Table 9.1. If this is the case it will be necessary to consider more sophisticated bidding strategies to avoid this exposure problem. A similar exposure problem is noted in the FCC spectrum auction problem, in which agents need sets of compatible licenses, and bid across simultaneous auctions [BCL00].

9.6 Related Work

This is not the first study of auction-based methods for train scheduling problems. Brewer & Plott [BP96], proposed the BICAP ascending-price auction for distributed train scheduling. The auction proposed in this chapter is more flexible: we allow trains to construct arbitrary schedules across the network, while BICAP restricts trains to bid from a small set of fixed paths. Returning to centralized approaches to train scheduling, Kreuger *et al.* [KCO97] have proposed a constraint-based method which appear to have better scaling properties than straight applications of MIP methods, but is perhaps less suited to making tradeoffs across schedules with different qualities.

Market-based methods have also been advocated for other distributed scheduling problems, such as for airport take-off and landing slot allocation problems [RSB82]. For example, Wellman *et al.* [WWWMM01] propose an auction-based method for a factory-scheduling problem, in which agents compete for periods of time on (one or more) shared machines. As in the train-scheduling problem, agents in the factory-scheduling problem (representing a job) often require a combination of time periods, perhaps also across multiple machines. The train scheduling problem differs in that it is not possible to define a *static* set of mutually-compatible times, any of which can be safely allocated to any agent. Instead, a feasible allocation of times must be checked dynamically, by computing a safe underlying meet/pass schedule. My approach is also rather different to that adopted by Wellman *et al.*, since I avoid imposing a discretization of time into finite slots, but provide instead a simple and expressive constraint-based bidding language.

9.7 Discussion

The auction-based mechanism for the distributed train scheduling problem presented in this chapter is a better match to the natural information and control structure of modern railroads than traditional centralized scheduling solutions. Moreover, initial experiments on a simple network structure show that the auction-based solution can generate better solutions than a centralized approach, and more quickly. The auction-based approach also appears to have useful scaling properties.

The main weakness in my current results is the lack of an interesting network structure. I assume a single-line network, which while quite a common assumption in the academic

train scheduling literature [KHC91, KH95, Hal93] is perhaps not very realistic. In the context of an auction-based method, relaxing the single line assumption would require two important extensions:

(a) extend the mixed integer program formulation of the winner-determination problem for each train agent to schedule trains across multiple lines,

(b) extend the bidding strategy of train agents to consider alternative routes, *in addition* to alternative times.

Extension (b) does not seem to be too difficult, given that train agents already consider alternative times (but not alternative paths); the current shortest path approach should extend quite readily. In addition, although it is not immediately clear how to formulate a multi-line problem as a mixed integer program, extension (a), the problem is certainly no more difficult than that faced in a centralized solution.

In addition to introducing heuristic methods and approximations for winner-determination in each round of the auction, it would also be interesting to compare the auction-based method with heuristic centralized methods. The current comparison with an integer-programming based centralized solver may be a little unfair, in that one can view the auction-based method (with prices increases, and myopic best-response, etc.) as combining special-case heuristic methods with a general-purpose integer-programming method.