EFFICIENT LOADING OF INTERMODAL CONTAINER TRAINS

Andreas Ernst[1] and Peter Pudney[2]

Efficient loading and unloading of trains is crucial if rail transport is to compete with road transport. The cost of rail transport can be reduced by improving the placement of containers on trains so that the number of wagons required is minimised and so that the train can be operated safely and efficiently. The cost of terminal operations, and the delays experienced by customers' trucks, can be reduced by improving the procedures used to load and unload trains.

We divided the problem into three subtasks. The first task is to develop a load plan for a train based on the expected mix of containers. An ideal plan specifies positions on the train for various container types in a way that minimises the number of wagons required while meeting a number of packing, safety and aerodynamic constraints. The second task is truck dispatching — when a truck arrives at the terminal, where should it be sent? This problem is not straightforward because trucks arrive randomly and loading and unloading occur simultaneously, and so the ideal position for a container may not yet be available. The third task is to dispatch the stackers that transfer containers between trains and trucks in such a way that truck waiting time is minimised.

The train planning problem can be formulated as an integer programming problem, but it can also be solved using the OASIS software already owned by National Rail Corporation. Train plans can be made more flexible by using container classes rather than specific container details.

The study group suggested several truck dispatch schemes, but did not have sufficient data to evaluate the schemes.

[1]CSIRO Mathematical and Information Sciences, Private Bag 10, Clayton South MDC, VIC 3169, Australia. Email andreas.ernst@cmis.csiro.au

[2]Centre for Industrial and Applicable Mathematics, University of South Australia, Mawson Lakes SA 5095, Australia. Email peter.pudney@unisa.edu.au

The study group also developed and evaluated several stacker dispatch policies for a single stacker. These policies are easy to implement; multi-stacker versions of two of the policies should be developed and evaluated using real data.

1. Introduction

National Rail Corporation operates rail freight terminals in Adelaide, Melbourne, Sydney, Brisbane, Alice Springs and Perth. Customers deliver containers to these terminals, where they are loaded onto trains. At the other end of the train journeys the containers are unloaded from the trains onto the customers' trucks.

The times and costs associated with loading and unloading are major impediments to the competitiveness of rail with road transport. There are two ways these times and costs can be improved:

- The cost of rail transport can be reduced by improving the way containers are stacked on the train, so that the number of wagons required to carry a given load and the running costs of the train are minimised.
- The operating costs of the terminal and the delays experienced by customers' trucks can be reduced by improving the procedures used to load and unload the trains.

A typical train is composed of a variety of wagon types and carries a variety of container types, with up to 150 containers. A full train may have a nominal 8 hours in which to unload the incoming containers and load the outgoing containers. Both activities occur in parallel, with several trains being worked simultaneously using common lifting equipment and labour.

The primary problem is to assign containers to wagons in such a way that the number of wagons required is minimised. But the load planning problem is not straighforward:

- there are several wagon types, with different lengths, deck heights and mass limits;
- there are many container types, with different lengths and heights, and masses that vary between 2 and 35 tonnes;
- some containers can be stacked on top of others, but some containers cannot have another on top; and

- the train must have reasonably good aerodynamics and mass distribution.

The problem is made even more difficult by the fact that the trucks that deliver and collect containers arrive at random times. When a truck arrives at the terminal it may be delivering one or more containers, collecting one or more containers, or both. The truck must be sent to a point alongside a wagon, where it waits for a stacker to load or unload it. Containers are usually transferred directly between trucks and wagons, but if necessary a container can be unloaded from the train onto the ground to free up a wagon slot.

Containers are transferred between trucks and wagons using overhead gantry cranes or reach stackers. A reach stacker is a mobile crane with a lifting frame that attaches to a container. The Melbourne terminal has a pair of gantries straddling four sidings, another pair of gantries straddling one track, and reach stackers; the Adelaide terminal has three reach stackers. The time taken to move a crane or stacker from one position to another is significant, and so it is important that waiting trucks are not scattered widely around the terminal, and that the trucks are served in an efficient order.

2. Subdividing the problem

The problem as described in the previous section is complex and difficult to manage without some form of simplification. After some discussion it was decided that to make the problem manageable it was sensible to divide it into three parts that require different types of decisions to be made. Each part can then be solved by a different type of algorithm in order to give an overall solution to the problem. The three parts of the problem are:

Load Planning: At the highest level the decision to be made is how the load should be arranged on the train. This problem assumes that we have a known list of containers and simply want to arrange these in an optimal way on the train. All operational restrictions due to the (as yet unknown) arrival times of the containers are ignored. Also, no consideration is given to the unloading of containers.

The 'ideal' loading plan can be calculated ahead of time, before the first truck arrives. Hence it is possible to spend more computational time in arriving at a good solution than is possible during the ongoing loading and unloading of the train.

The loading plan arrived at in this step provides a goal to aim for. Naturally the fact that the order in which trucks arrive is not known may make it impossible to follow the loading plan exactly. Nevertheless, to the degree to which it is possible to follow this plan, it should provide a useful guide. Furthermore, when changes need to be made to the plan due to operational considerations, it is not necessary to re-optimise the complete loading plan but simply to try to find some minimal change to the plan which can accommodate the new requirements.

Truck Dispatch: When a truck arrives at the gate it must be assigned a position on the train for its container. The choice of position should be guided both by the loading plan arrived at earlier as well as the operational considerations of the stacker movement. In essence this phase holds together the other two parts of the problem. Note that trucks collecting containers may also have a choice of container if the company has more than one container on the train.

Stacker Dispatch: The short term decision problem is to determine the order in which to service the trucks that have been assigned a position next to the train. For this problem, no distinction needs to be made between loading and unloading of the train; it is simply a matter of minimising the time spent traveling by the stackers while maintaining a good service standard for the trucks.

Each of these three problems is discussed in more detail in the following sections.

3. Load planning

Load planning is concerned with determining a good stacking plan for the containers on the train. This plan will be referred to as the *ideal plan* since it ignores the operational constraints involved in loading and unloading the train. The aim is to provide a guide to how the train should be loaded in order to fit all of the containers with a nice overall container distribution for the final loaded train. While it may not be possible to follow this ideal plan exactly, it provides a guide to be used in the next phase (truck dispatch).

There are a number of objectives and constraints that need to be considered in generating the ideal plan:

- Length Restriction: The most obvious constraint is imposed by the length of the containers. Obviously the combined length of containers that can be placed onto a platform cannot exceed the length of the platform. In practice there are only a very limited number of combinations possible. Containers are usually 20, 40 or 48 feet long, while platforms come in lengths of 40, 48, 60 or 80 feet.

- Weight Restriction: The maximum weight that each bogie (set of wheels) can carry is limited. In practice this restriction causes a problem for only a small number of very heavy containers which can only be placed efficiently in a small number of places.

- Height Restriction: The total height of any wagon in the train needs to be restricted to ensure that the train fits under all bridges it encounters. This limits the amount of double stacking that can be done. 'Well wagons' are special kinds of wagons that have lowered platforms to make it easier to stack two containers.

- Aerodynamics: Fuel cost for long journeys is considerable, but can be reduced by ensuring that the train has a 'nice' shape. The ideal shape starts relatively low, increases in height (double stacked wagons), then is followed by more single stacked wagons. Gaps must be avoided since they create more turbulence.

- Weight Distribution: The overall distribution of weight must be biased towards the front of the train for safety reasons.

- The total number of wagons on the train should be minimised.

Given a configuration of wagons for the train (that is, the types of wagons used in the train and the order in which they are strung together) as well as the anticipated set of containers to be loaded, the load planning problem tries to create an optimal plan based on the above considerations.

It is useful to split the containers into a number of categories, so that rather than assigning a unique identifier to each container, only its general characteristics, such as length, approximate weight and height, are considered. It may suffice to split the containers into 12 categories: three lengths (20, 40 or 48 foot) and four weight types (light, medium, heavy and very heavy). We will assume that all containers are approximately the same height. If necessary the containers could be split into two or three categories based on height as well.

The container data for this problem can be taken from the bookings available prior to the arrival of the train. This can be augmented by some additional generic containers of various types to ensure that the ideal plan reserves some spots for last minute additions to the train that were not booked beforehand.

One way to solve this problem is as an integer programming problem. In the formulation below we will assume that all possible configurations of loading different types of containers onto a given platform type have been enumerated. For example, consider a 60 foot platform. Assuming that it cannot be double stacked, there are about 100 different feasible combinations (three 20 foot containers each with one of four different weights gives 64 combinations, some of which may exceed the weight limit, two 20 foot containers each with four possible weight classes, one 20 and one 40 foot container etc).

Let x_{wi} be the number of wagon platforms of type w that use the i^{th} possible configuration of containers. Then the load planning problem can be formulated as:

$$\min \sum_w \sum_i C_{wi} x_{wi} \tag{1}$$

$$\text{Subject to} \quad \sum_i x_{wi} \leq m_w \quad \forall\, w \tag{2}$$

$$\sum_w \sum_i L^c_{wi} x_{wi} = n_c \quad \forall\, c \tag{3}$$

$$x_{wi} \geq 0 \text{ and integer} \quad \forall\, w, i. \tag{4}$$

Here constraint (2) ensures that only as many platforms are used of each type as are actually available. L^c_{wi} represents the number of containers of type c that are to be loaded onto platform of type w according to the i^{th} possible packing. Hence constraint (3) ensures that there are enough spaces to load each container exactly once. The length, height and weight restrictions for each platform are considered in defining the set of possible configurations. The objectives of ensuring a nice weight distribution and an aerodynamic profile for the train can be taken care of in two ways:

1. The cost coefficients C_{wi} can be used to appropriately penalise undesirable patterns, such an 80 foot platform containing only one 48 foot container. Also, because the approximate position within the train can be deduced from the wagon type, it is possible to include in C_{wi} some penalty factors to discourage heavy wagons at the rear of the train or having tall arrangements at the front or rear of the train.

2. Any solution to the above integer program still allows some freedom in choosing how to arrange the train, as there is a significant number of platforms of the same type. The choice of configuration from the solution of the integer program to specific wagons can be made in a greedy heuristic fashion or by solving a matching problem in which possible assignments of configurations to individual wagons are weighted based on the height and weight of the configuration and the position of the wagon in the train.

   Note, however, that the ideal load plan may not be followed exactly during the truck dispatch phase, and so it may be better not to spend too much effort on the assignment of configurations to specific wagons. Configurations can be swapped between equivalent platforms during the truck dispatch phase. This may be necessary if a container arrives for a platform that has not yet been unloaded.

The only thing not considered in the above model is the case where the loading of adjacent platforms cannot be done independently. This is the case for 'skel' wagons, where adjacent platforms have a shared bogie and hence a shared weight restriction. If the shared weight restriction cannot be satisfied as a post processing step when assigning configurations to individual platforms then the integer program could be modified to include all configurations for the whole wagon, rather than for just an individual platform. Such a modification would significantly increase the number of variables and probably require some form of column generation to deal with the increased problem size. Nevertheless, the problem given in (1)–(3) is likely to remain manageable as the number of constraints is small (< 100).

An entirely different approach is to use the OASIS software already owned by National Rail Corporation. The main problem with this software is that it does not deal with the operational constraints of loading and unloading the train simultaneously, but simply assumes that all containers are available to be picked up from a trailer pool in whatever sequence it determines. Since our loading problem also ignores the operational constraints, it would seem easiest as a first step to use OASIS to create the ideal plan. While OASIS creates a schedule and loading plan for individual containers, the container identifiers can be ignored to give a plan by container category.

The recommendation of the MISG working group on this problem is that OASIS should be tried for this problem before custom software based on the above integer programming formulation or heuristics is developed.

4. Truck dispatcher

Each time a truck arrives at the entrance to the terminal, the exact type of containers it is delivering (including their weights) and the containers that the truck is supposed to collect are recorded. At this point in time the truck needs to be assigned specific positions on the train where it is to deliver and collect each container. For collection there is often no choice as the trucks are generally sent to collect a specific container which has a known position on the train. For containers delivered to the terminal, however, a choice needs to be made. This choice should allow efficient loading of the train while ensuring that a good loading plan is maintained.

The truck dispatcher module ties together the other two components of the overall system. It takes the ideal plan produced by the load planner to determine where to send the trucks while allowing for efficient movement of the stackers. So the truck dispatcher module simply follows the following generic algorithm

```
Truck Dispatcher
    Wait for next truck to arrive
    For each container on the truck
        Assign a position on the train where it is to be loaded
    For each container to be collected
        Determine position on the train
```

As mentioned previously, the position of containers to be collected is often fixed, and so the main decision to be made is the position at which the incoming containers are to be dropped off. Most trucks only deliver or collect a single container, but in some cases two containers can be carried on a single truck, in which case the order in which to process the containers also needs to be determined.

A number of ways to determine a good assignment of the containers to positions on the train have been suggested:

1. Consider the ideal loading plan produced earlier and restrict the possible positions for each incoming container to those that have a container of matching type earmarked in the ideal loading plan and for which the space already exists on the train.

   For example, suppose a truck arrives with a medium weight, 40 foot container. The ideal load plan may have several possible positions for such a container. However, some of these options may not be feasible at the current time as the positions are blocked by containers that are yet to be unloaded. After eliminating such infeasible positions there are two cases:

   - If there are multiple options left, one of them can be chosen arbitrarily or else the dispatcher heuristic could be run for each possible position to see which choice would impact most favourably on the stacking problem.
   - If no feasible positions remain then a local search could be used to modify the ideal plan. As mentioned in point 2 on page 129, the assignment of platform configurations to specific wagons can be interchanged without significantly affecting the ideal plan. Alternatively,

one could look at simple modifications such as swapping two containers of the same size but different weight characteristics between two platforms (provided the height and weight constraints are not violated). Finally, in the unlikely case that none of the above methods arrives at a new ideal plan that can accommodate the container being considered, then one of the greedy heuristic rules described below needs to be used.

2. Make use of appropriate functionality in OASIS. This strategy would require co-operation from the company that produces the OASIS software. However, as part of the GRASP heuristic [4] used in OASIS there should already be functionality that evaluates the feasibility and fitness for placing a container in any possible position. All that is required is to make this functionality available directly to the user on a truck by truck basis.

3. A greedy heuristic could be used to assign each container to a position. There are a number of ways of implementing a greedy heuristic, but they all share the following basic behaviour in common. For each possible position to which a container can be assigned, a score is evaluated. The container is then assigned to the position with the best score. The method is myopic in that it does not consider possible future arrivals directly. Hence the scoring system needs to be designed in such a way that the current assignment will not prevent containers arriving in future from being loaded.

   The following scoring methods have been suggested. Note that each of these could be used alone or a combination of scores could be used in order to allow a trade off between different factors. Consider the potential assignment of container c to platform w:

   (a) Maximise the length of remaining gaps: This measure is designed to allow for the greatest flexibility in loading containers still to arrive by assigning (c, w) a score equal to the longest gap remaining on platform w after c is loaded on it, with a very large score if the remaining gap is exactly zero.

   (b) A slightly more sophisticated variation of the above performance measure is to define the following function for a partially loaded train:

   $$f = \sum_{\text{gaps g}} (\text{value of gap type } g) * (\#\text{ gaps of type } g \text{ remaining})$$

   This function can then be evaluated before and after the assignment of c to w and the difference in f values is the score of the assignment, with lower value being better. The advantage of this system is that it can be better tuned to the different container types. For example, any gap of less than 20 feet cannot be filled by any containers and so should receive a score of zero. Also, a 40 foot gap is much more versatile than two gaps of 20 feet, and so the longer gap should receive more than twice the score of the shorter one.

A further refinement of this scoring function would be to base the weights of gaps at least in part on the expected distribution of containers yet to come. For example, if there is a very low probability of any more 48 foot containers being delivered for the given train then the score of a 48 foot gap should be no higher than that of a 40 foot gap (if anything it should be lower, as it is likely to lead to a gap in the final train which is undesirable from the aerodynamic viewpoint).

(c) None of the above measures take into account the difficulty of having to simultaneously load and unload containers from the same set of platforms. Consider assigning a score to the assignment (c, w) equal to the total length of the sequence of containers on either side of c that have already been loaded. Maximising this measure means that containers are, whenever possible, placed next to others that have already been loaded. Hence this heuristic tries to keep gaps next to containers that have yet to be unloaded. Apart from making it easier to load the remaining containers, it should also improve the efficiency of the stacker scheduling as this method will try to build up long stretches of platforms that have been finalised with no need for a stacker to return to them.

(d) One way to directly minimise the impact of the container assignment on the stacking problem is to run the stacker dispatcher module for each possible assignment of container to a platform. By comparing the time required to load waiting trucks before and after the assignment one can use the greedy heuristic to minimise the expected waiting time of the trucks.

While it is not expected that this measure will produce particularly good train loading patterns by itself, it could be used in two ways. First, it can be used as an additional term or tie-breaker for one of the other scoring methods, to bias the greedy heuristic towards assignments that are easier to schedule. Second, this method provides a way of determining the preferable order of delivery or collection when there are more than two containers are on the truck.

Which of these methods is best suited to producing a good solution for the container loading problem of National Rail Corporation cannot be ascertained without performing some empirical testing with real data. Since no such data was available, the study group did not come to any definite conclusions.

It should be noted that the general framework for the truck dispatcher could be implemented to be performed manually with only minimal decision support software. For example, provided the ideal plan was available in some form, it could easily be used as a guide to assigning containers to platforms.

5. Stacker dispatcher

5.1 The stacker dispatch problem

Trains are loaded and unloaded using stackers, which are either overhead gantry cranes or mobile cranes fitted with container lifting frames. At any instant there could be several trains in the terminal, each being simultaneously unloaded and loaded. Trucks arrive at random times, bringing new containers to be loaded onto a train, collecting containers from a train, or both. When a truck arrives at the terminal the truck dispatcher sends the truck to a position alongside a train, where it waits for a stacker to serve it. The stacker dispatch problem is to determine the order the stackers will visit the waiting trucks in a way that minimises truck waiting time and the cost of operating the stackers.

5.2 A single stacker dispatch problem

The stacker dispatch problem is easier to analyse if there is only one stacker.

When a truck arrives at the terminal the truck dispatcher sends it to a location adjacent to a wagon, where it waits for the stacker to load or unload a container. If the truck has more transactions it then moves to the next location, otherwise it leaves the terminal.

From the stacker's point of view, it must service a sequence of requests $T_1, T_2, \ldots, T_n$. Each request T_i is defined by the tuple $T_i = (a_i, x_i, d_i)$ where a_i is the arrival time of a truck at location x_i, and $d_i > a_i$ is the time at which the loading or unloading is completed and the truck departs.

At time t the set of waiting requests is $W(t) = \{T_i \mid a_i \leq t < d_i\}$. Future arrival times are not known in advance, and so cannot be used to plan stacker movements.

If the stacker is at location x at time t and choses to service waiting request $T_i \in W(t)$ then the departure time d_i will be

$$d_i = t + \tau(x, x_i) + l$$

where $\tau(x, x_i)$ is the time taken for the stacker to travel from location x to location x_i and l is the time required to load or unload the truck.

The waiting time for each request is $w_i = d_i - a_i$. The objective is to minimise some function of the waiting times, such as:

- mean waiting time
- RMS waiting time
- maximum waiting time

5.3 Dispatch policies

A policy is a function that is used to decide which request should be serviced next. It depends on the current time, the location of the stacker, and the set of waiting requests. It may also depend on an internal stacker 'state' that depends on the history of the stacker. For example, stacker state may be used to encode the direction the stacker is travelling.

We developed five different policies during the MISG workshop.

- The **FIFO** (First In, First Out) policy services requests in order of arrival time, a_i.
- The **nearest** policy selects the waiting request i that minimises the travelling time $\tau(x, x_i)$ from the stacker's current location x.
- The **loopy** policy serves the next truck to the right or, if there are none, the leftmost truck. That is, the stacker moves right along the train servicing trucks until there are no more trucks to the right, then jumps back to the leftmost truck.
- The **nearest-longest** policy serves the nearest truck next unless a truck has been waiting longer than time $t_{\max}$, in which case the truck that has been waiting longest is served next.
- The **mirage** policy makes trucks that have been waiting a long time appear closer than they really are, then serves the truck that appears closest.

None of these policies require the stacker to have an internal state. Following the MISG workshop, Alan Brown and Patrick Tobin suggested another strategy:

- The **sweep** policy selects the next truck in the current direction, or, if there are none, changes direction.

For each of these policies we must also specify what the stacker should do if there are no pending requests. We chose to leave the stacker at its last location until another request was received.

5.4 Loopy and sweep analysis

Alan Brown analysed the loopy and sweep policies for a terminal with overhead cranes straddling several sidings, as shown in Figure 1.

Figure 1: Three sidings, two overhead cranes.

Estimates for the mean and maximum waiting times can be derived by considering a deterministic model. Suppose each stacker services a segment of length L/np, where L is the total train length, p is the number of parallel tracks and n is the number of stackers. The time required for each stacker to complete a loop without stopping to load or unload containers is $t = 2L/vnp$, where v is the travel speed of the stackers.

The average loop time for a stacker is the travel time plus the loading time for the expected arrivals. If trucks arrive at a rate λ and are loaded at a rate μ then the traffic intensity is $\psi = \lambda/\mu$ and the average loop time is given by

$$T = \frac{2L}{vnp} + T\psi$$

which can be solved to give

$$T = \frac{2L}{vnp(1-\psi)}.$$

In the deterministic case, the maximum waiting time will be T and the mean waiting time will be $T/2$.

With stochastic arrivals, the mean waiting time is likely to be less than $T/2$ because the stackers do not have to travel a complete loop; they can turn when there are no more trucks waiting in the current direction. On the other hand, the maximum waiting time is likely to be greater than T because of the stochastic variation in arrivals.

The loopy policy controls the maximum expected waiting time by providing the same level of service to all trucks. The sweep policy provides a better level of service to those trucks near the middle of a loop, but provides two chances of serving those trucks near the ends of a loop, which in turn increases the possibility of reducing the size of the next loop. Simulation results show that the sweep policy is usually better than the loopy policy.

5.5 Choosing the cut-off time for the nearest-longest policy

The nearest-longest policy serves the closest truck next unless there is a truck that has been waiting longer than time $t_{\max}$. What should $t_{\max}$ be? Alan Brown suggested that special action is required whenever a waiting time exceeds the mean waiting time by one standard deviation.

Many of the stacker policies we tested result in unimodal distributions of truck waiting time. These distributions are often skewed heavily to the right, and so maximum waiting times become high. (An exception is the loopy policy, which is designed to control the upper bound on the waiting time.) Policies which lead to skewed distributions of waiting time need to be modified to avoid extreme waiting times.

We want to select a cut-off time $t_{\max}$ that defines the start of the distribution tail. We will then modify the policy for waiting times within the tail. A sensible choice of cut-off time is $t_{\max} = \mu + \sigma$, where μ and σ are the mean and standard deviation of the waiting times for the unmodified policy. The Normal Power approximation to the waiting time distribution is

$$F(x) = \mu + \sigma \left(y + \frac{\gamma}{6} \left(y^2 - 1 \right) + O(1/n) \right)$$

where γ is the skewness of the distribution and $\Phi(y) = 1 - \epsilon$ is the standard normal distribution [1, 2]. When $y = 1$ the skewness term vanishes for all waiting time distributions. Furthermore, this is the point of inflection for the standard normal distribution, and so seems to be a reasonable point to define the start of the tail. The probability of a waiting time in the tail of the distribution is $\epsilon = 0.17$, which means we will be intervening for about 17% of the trucks.

The moments of the waiting time distribution for a loopy policy provide useful measures for the mean and standard deviation when small samples are available, as the higher moments are quite small under this policy.

The simulation results in Section 5.8 show that intervention at about one standard deviation from the mean does indeed minimise the maximum waiting time.

5.6 Designing the mirage function

The mirage policy uses a 'mirage function' to make trucks that have been waiting a long time appear closer than they really are, then chooses the truck that appears to be closest. The apparent time required to travel to a request T_i is

$$\tau^*(t, a_i) = m(t, a_i)\tau(t, a_i)$$

where m is the mirage function. The mirage function should have the following characteristics:

- $m = 1$ for trucks that have not been waiting too long;
- m decreases to a small value $\epsilon > 0$ as the waiting time approaches $t_{\max}$;
- $m > 0$ for $t - a_i > t_{\max}$ so that requests that have been waiting longer than time $t_{\max}$ are served in (approximate) order of arrival.

The mirage function used to test the mirage policy was

$$m(t, a) = \sqrt{\min\{1, \frac{10}{3}\max\{0.0001, 1 - \frac{t-a}{t_{\max}}\}\}}$$

which has $m = 1$ for $t - a_i \geq 0.7\, t_{\max}$, then follows the square-root curve down to $m = 0.0001$ for $t - a_i \geq t_{\max}$.

As with the nearest-longest policy, the maximum waiting time can be minimised by setting $t_{\max} \approx \mu + \sigma$.

5.7 Test problem

The stacker dispatch policies were evaluated using the following problem:

- 100 requests;
- random arrival times a_i distributed uniformly over the interval $[0, 14400]$ seconds (4 hours);
- locations selected randomly without replacement from $\{10, 20, \ldots, 1400\}$ metres;
- stacker travel speed of 5ms^{-1}, giving travel times $\tau(x, x_i) = |x - x_i|/5$;
- loading time $l = 120$ seconds.

The sequence of locations was found by forming the sequence of pairs

$$(r_1, 10), (r_2, 20), \ldots, (r_{140}, 1400)$$

where r_i were random numbers distributed uniformly on the interval $[0, 1]$, and then sorting these pairs by the first element. The locations from the first 100 pairs of the sorted list were assigned to the request locations x_i.

We also used a sequence of autocorrelated locations, to simulate the effect of a truck dispatcher that tries to put each truck close to the previous truck. Given a sequence of uniformly distributed random numbers s_i, the autocorrelated sequence was given by $r_i = \alpha r_{i-1} + (1 - \alpha)s_i$ with $r_0 = 0$. The autocorrelated sequence r_i was used as before to generate a sequence of request locations. For the test problems we used $\alpha = 0.8$.

5.8 Results

Figures 2–7 show results of each of the policies. The horizontal axis is time and the vertical axis is location. The horizontal lines show the waiting time for each request; the lighter line shows the movement of the stacker. The request locations are not autocorrelated.

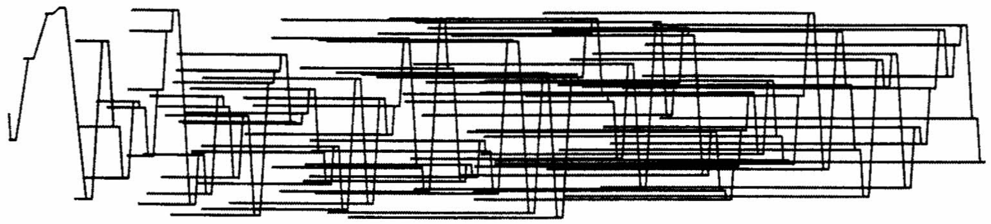

Figure 2: Stacker movement and waiting times for the **FIFO** policy.

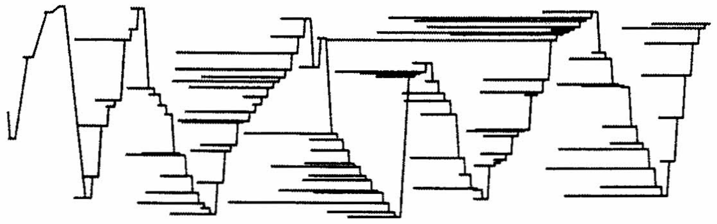

Figure 3: Stacker movement and waiting times for the **nearest** policy.

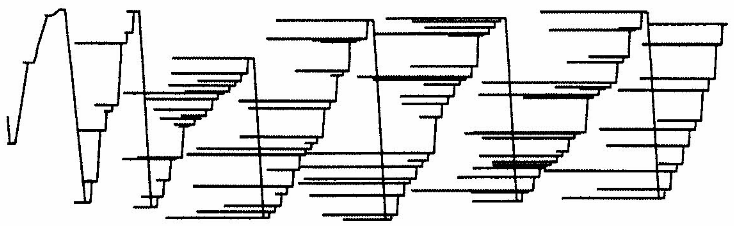

Figure 4: Stacker movement and waiting times for the **loopy** policy.

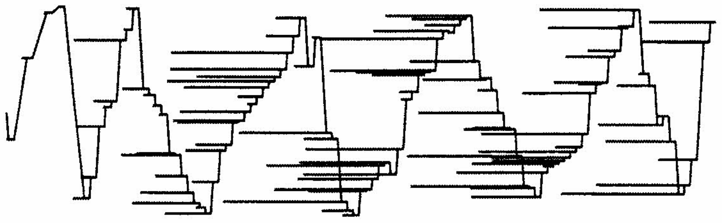

Figure 5: Stacker movement and waiting times for the **nearest-longest** policy, $t_{\max} = 2400$.

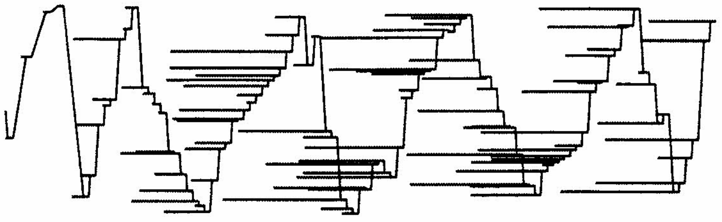

Figure 6: Stacker movement and waiting times for the **mirage** policy, $t_{\max} =$ 2400.

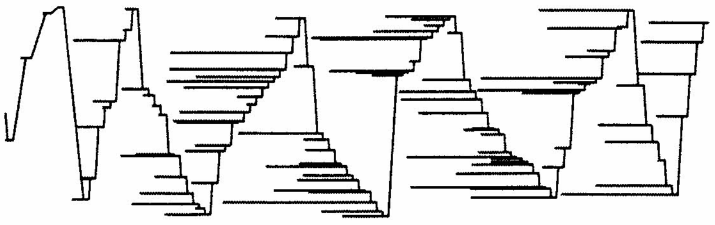

Figure 7: Stacker movement and waiting times for the **sweep** policy.

Tables 1 and 2 show the means of the mean waiting time, RMS waiting time and maximum waiting time, in minutes, from ten runs with each of the policies. In the first table the request locations were uncorrelated; in the second table the request locations are autocorrelated with $\alpha = 0.8$. The parameter for each of the nearest-longest and mirage policies is the cut-off time, t_{max}, in minutes.

Table 1: Waiting times, in minutes, for the various stacker policies, $\alpha = 0$.

policy	mean	RMS	maximum
FIFO	68.19	77.45	128.44
nearest	18.06	23.45	69.53
loopy	22.31	26.26	56.06
nearest-longest 40	24.84	30.45	58.48
nearest-longest 45	21.05	26.40	55.52
nearest-longest 50	19.61	24.82	55.61
nearest-longest 55	18.75	24.12	58.67
nearest-longest 60	18.46	23.79	61.71
mirage 40	21.27	25.92	51.78
mirage 45	19.82	24.70	51.75
mirage 50	18.93	23.93	54.09
mirage 55	18.78	24.09	57.83
mirage 60	18.57	23.90	61.69
sweep	17.92	22.04	56.31

5.9 Mathematical programming

In order to obtain 'optimal' solutions to the stacker dispatch problem a MILP approach can be used. In this section we describe one such formulation that captures much of the complexity of the stacker dispatch problem. For the purpose of this section a *job* is the task of either loading or unloading a container at a certain position on the train. Since the position on the train is known, the travel time between jobs is also known.

One formulation for this problem is described below:

Sets	
T	= Set of trucks
J	= Set of jobs
J^0	= Jobs already commenced
J_t	= Jobs required by truck t

Table 2: Waiting times, in minutes, for the various stacker policies, $\alpha = 0.8$.

policy	mean	RMS	maximum
FIFO	64.81	73.26	119.55
nearest	17.07	22.54	65.66
loopy	21.03	25.10	56.78
nearest-longest 40	21.16	26.43	54.08
nearest-longest 45	18.85	23.81	51.54
nearest-longest 50	18.08	23.25	54.75
nearest-longest 55	17.66	23.00	58.00
nearest-longest 60	17.42	22.83	60.59
mirage 30	23.46	27.38	54.05
mirage 35	20.59	24.71	48.79
mirage 40	19.31	23.85	51.58
mirage 45	18.47	23.15	51.74
mirage 50	17.76	22.79	54.66
sweep	17.08	21.13	56.06

Variables	
x_{ij}	$=$ 1 if jobs i and j are performed by the same stacker, 0 otherwise
y_j	$=$ start time of job j
w_t	$=$ total time spent by truck t in the system
v_t	$=$ penalty for truck $t = \max\{0, w_t - D\}$
z	$=$ maximum time required to service any truck

Parameters	
a_t	$=$ arrival time of truck t
D	$=$ service standard: maximum duration for servicing a truck
d_j	$=$ deadline for job j $(= r_j + D)$
p_j	$=$ processing time for job j
m_{ij}	$=$ movement time for relocating the truck between jobs i and j
M	$=$ a big constant
r_j	$=$ release time of job j (earliest time it can be done)
s_j	$=$ start time of job $j \in J^0$
τ_{ij}	$=$ time for stacker to travel between jobs i and j

$$\min \sum_{t \in T} w_t \tag{5}$$

$$\text{or} \quad \min \sum_{t \in T} v_t \tag{6}$$

$$\text{or} \quad \min \; z \tag{7}$$

$$
\begin{array}{rrcll r}
\text{Subject to} & w_t & \leq & D + v_t & \forall\, t \in T & (8)\\
& z & \geq & w_t & \forall\, t \in T & (9)\\
& \sum_{i \neq j} x_{ij} & = & 1 & \forall\, j \in J \setminus J^0 & (10)\\
& \sum_{j \in J \setminus J^0, j \neq i} x_{ij} & \leq & 1 & \forall\, i \in J & (11)\\
& y_j & = & s_j & \forall\, j \in J^0 & (12)\\
& y_i + p_i + \tau_{ij} - M\,(1 - x_{ij}) & \leq & y_j & \forall\, i \in J,\ j \in J \setminus J^0,\ i \neq j & (13)\\
& y_i + p_i + m_{ij} & \leq & y_j & \forall\, i, j \text{ on same truck, } i \text{ before } j & (14)\\
& y_i + p_i - a_t & \leq & w_t & \forall\, t \in T,\ i \in J_t & (15)\\
& x_{ij} & \in & \{0,1\} & \forall\, i, j \in J & (16)\\
& v_t,\ w_t,\ y_j,\ z & \geq & 0. & & (17)
\end{array}
$$

Here objective (5) minimises the total time required to process all of the trucks. Alternatively objective (6) could be used in order to minimise the amount of time by which the service standard is violated. Finally objective (7) minimises the maximum time spent waiting by an individual truck. This last objective serves mainly to maximise throughput. Constraints (8) and (9) only serve to define the v and z variables for the second and third objective respectively. Equations (10) and (11) are used to ensure that each job is placed somewhere into the sequence of jobs to be completed by one of the stackers. We assume here that each stacker is assigned some job in J^0 that defines its current position — if this is not the case a dummy job of zero duration could easily be added to satisfy this assumption. The start time of each job j, y_j, is given by equations (12)–(14). Finally constraint (15) ensures that each w_t variable takes on an appropriate value.

Some numerical experiments conducted using randomly generated data indicate that it is difficult to produce good solutions with this approach. As the LP relaxation is not very tight, the MILP solver spends a lot of time in the branch and bound before finding a good solution. Given that the dispatch problem needs to be solved quickly, this approach is not recommended for use by National Rail.

Another exact method proposed is based on an algorithm for optimally landing aircraft on a single runway, see Ernst *et al.* [3]. The aircraft landing problem consists of landing a given set of aircraft, each with its own landing time window. The landing times are constrained by minimum separation distances between planes that depend on the aircraft involved, as the separation requirements are at least partly due to turbulence considerations that vary between aircraft types. The aim of the aircraft landing problem is to assign to each plane a feasible landing time so that the deviation from the desired landing time of the plane is minimised.

The stacker dispatch problem can be transformed into an aircraft landing problem as follows. Each job that needs to be done corresponds to an aircraft that needs to land. The arrival time of the truck next to the train corresponds to the earliest possible landing time of the aircraft while the latest time is constrained to be within a fixed time D of the arrival time. The place of the minimum inter-arrival time between aircraft is taken by the processing time of the job plus the travel time of the stacker from one job to the next. Finally in the aircraft landing problem the deviation from the desired landing time is minimised. Here we simply take the desired time for starting the job to be the arrival time and hence minimise the total waiting time by all of the jobs currently being considered.

Numerical testing of this method shows that it works reasonably well. The main difficulty is that as the number of jobs waiting gets large, the algorithm often has difficulty proving optimality. This is because there are many nearly equal solutions when there are several jobs close together. Furthermore the jobs are not distinguished by different cost penalties. Thus there are many solutions with very similar costs that need to be searched before the algorithm can declare that it has found a provably optimal solution. Hence for problems of up to 10 jobs an optimal solution can be found in reasonable time ($\leq$ 5 seconds on a 666MHz Alpha processor). For larger problems it is advisable to limit the CPU time or use a heuristic to prevent unreasonable running times (for problems with 15 jobs, for example, the running time can easily be several hours for the exact method).

6. Conclusions and recommendations

Train running costs can be reduced by improving the way containers are placed onto trains. The costs associated with loading and unloading trains can be reduced by improving the way containers are assigned positions on the train and by improving the order in which waiting trucks are served.

We divided the problem into three tasks: load planning, truck dispatch and stacker dispatch.

A load planning system determines the ideal placement of container types onto a train for the expected mix of container types. The problem was formulated as an integer programming problem. This problem can also be solved using the OASIS software currently owned by National Rail Corporation.

- Train load should be planned using OASIS, but using container classes rather than specific container details so that containers can be more easily rearranged during loading.

- If necessary, custom software could be developed based on the integer programming model developed during the workshop.

When a truck delivers a container to the terminal, the truck dispatcher must decide where the container should be placed on the train. The load plan will provide guidance, but the ideal positions may still be occupied by containers waiting to be unloaded. Several truck dispatch schemes were suggested, but we did not have the data needed to determine the best scheme.

At any instant there could be several trucks waiting alongside trains to be loaded or unloaded. The stacker dispatch problem is to assign stackers to waiting trucks in an order that minimises truck waiting time. We developed and evaluated several stacker dispatch policies for the case of a terminal served by a single stacker. Further work is required to develope multi-stacker versions of the best policies.

- A multi-stacker mirage policy should be developed, evaluated and tuned using real truck arrival times and positions, and the results compared to actual stacker operations. A multi-stacker sweep policy should also be developed and evaluated using real data.

- These policies are designed to be easy to implement at the terminal, and should be implemented if the evaluation indicates a potential for improving terminal operations.

Acknowledgements

The project moderators, Andreas Ernst and Peter Pudney, would like to thank Gary Alexander (National Rail Corporation) for his clear and patient explanations of the problem, and for withholding the complicated bits until we were ready to deal with them.

Thanks also to the MISG participants who contributed to this problem: N.R. Achuthan, Alona Ben-Tal, Igor Boglaev, Natashia Boland, Alan Brown, Grant Cox, Paul Haynes, Frank Jackson, Houyuan Jiang, Philip Kilby, Vicky Mak, David Panton, Andrew Percy, Susanne Scott, David Sier, Patrick Tobin, Kevin Werksman, and Kevin White.

References

[1] R.E. Beard, T. Pentikainen and E. Pesonen, *Risk Theory* (Methuen, London, 1969).

[2] E.A. Cornish and R.A. Fischer, “Moments and cumulants in the specification of distributions”, *Rev. Inst. Int. Statist.* **5** (1937), 307.

[3] A.T. Ernst, M. Krishnamoorthy and R. Storer, “Exact and heuristic algorithms for scheduling aircraft landings”, *Networks* **34** (1999), 229–241.

[4] T.A. Feo and M.G.C. Resende, “Greedy randomized adaptive search procedures”, *Journal of Global Optimization* **6** (1995), 109–133.