Train Outstable Scheduling as Constraint Satisfaction

Andy Hon Wai Chun
Chief Information Officer
City University of Hong Kong
Tat Chee Avenue, Kowloon Tong
Hong Kong SAR
andy.chun@cityu.edu.hk

Abstract

This paper outlines the design of a scheduling algorithm that allocates outstabling locations to railway trains. From time to time railway trains may need to be outstabled to temporary locations, such as stations, sidings, depots, etc., until they are needed for regular operations. This is common for urban rail transit, and especially so for those that do not operate 24 hours. During non-traffic hours (NTH), trains are outstabled to various locations along the rail network so that when operations start again next day, the trains will be nearby their originating station or conveniently located so that they can be put into service whenever needed. However, this is complicated by the fact that engineering works, such as rail testing, installation, regular maintenance, etc. are done during the NTH. Therefore, passenger trains must be outstabled in such a way that they do not interfere with night-time engineering works or the movements of associated engineering trains. Since the engineering works scheduling is done separate to outstabling, this is a mixed-system problem. This paper shows how we modeled this as a constraint-satisfaction problem (CSP) and implemented into an "Outstabling System" (OSS) for the Hong Kong Mass Transit Railway (MTR) using a two-stage search algorithm.

Introduction

The outstable scheduling work was performed for the MTR Corporation Limited (MTRC) in Hong Kong. The MTRC operates both the metro network and the commuter rail network. In Hong Kong, MTRC operates 10 railway lines with 86 railway stations, and a Light Rail with 69 light rail stops. The system spans 211.6 km (131.5 miles) of rail covering a wide area of the city and servicing on average 4 million passengers each weekday (MTR homepage 2013, MTR Wikipedia entry 2013). Fig. 1 is a map of the MTRC network. Our outstable scheduling work supports all the 10 railway lines, but excludes the Light Rail.

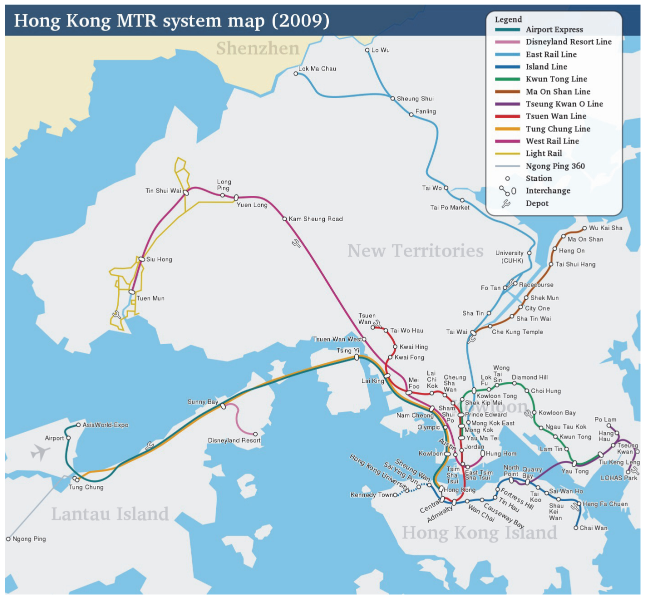

Figure 1. Hong Kong railway route map (public domain map by Sameboat: http://en.wikipedia.org/wiki/File:Hong_Kong_Railway_Route_Map_en.svg)

To "outstable" a railway vehicle is to park it at a station, siding or other location away from the depots (Wiktionary 2013). These passenger trains will stay at their outstabled locations overnight. Those locations are usually at or nearby stations where the trains will start service next morning. This eliminates/minimizes travel time from train depot next morning when service resumes. However, outstabling locations must be carefully assigned so that they do not interfere with any overnight engineering works that might need to be performed on the lines. For example, allocating all available tracks at one location to passenger trains overnight will block any engineering train that might need to pass through.

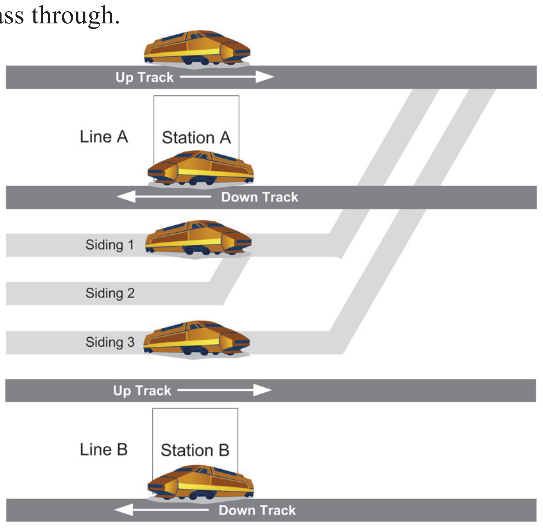

Figure 2. Example of outstabling with two railway lines and 3 sidings.

Fig. 2 illustrates some of the complexity in allocating outstables. One constraint, for example, is that if trains are outstabled at both Station A up and down tracks, then at least one of the sidings must be vacant, otherwise engineering trains will not be able to maneuver. Similar outstabling constraints occur throughout the railway network at different locations.

The project objective is to automate the assignment of outstabling locations to trains using AI while ensuring all safety and operational constraints/guidelines are met.

Project Background

MTR has an IT system called the "Engineering Works and Traffic Information Management System" (ETMS) that manages all information relating to the scheduling, processing, and management of engineering works. Within ETMS is an "AI Engine" that we designed and developed to schedule and optimize the assignment of NTH engineering works. Version 1 of the "AI Engine" went into production in 1995 for the 7 metro lines (Chun, et.al, 2005). Version 2 of the "AI Engine" extends AI scheduling to the remaining 3 commuter railway lines, and is targeted for deployment in summer 2013.

The "Outstabling System" (OSS) is a separate, but integrated, sub-system of the Version 2 of the "AI Engine" to perform train outstabling.

Related Research

In the past, the AI research community has been focused on train scheduling/timetabling (Chiu 2002, Abril 2006), train driver/crew rostering/scheduling (Martins 2003), rolling stock maintenance scheduling (Otsuki 2011), track maintenance scheduling (Peng 2010, Zhang 2012) or maintenance team scheduling (Peng 2012). We have not found any AI research work relating to the scheduling and optimization of railway line "engineering works" besides an earlier CHIP implementation used by MTR (Cheung 1999). Engineering works include any type of testing, installation, repair, inspection, etc., i.e. any work that needs to be done along the railway lines. Usually, it will require the assignment of a team of workers and possibly an engineering train. Track maintenance scheduling is related, but deals mainly with timetabling rather than resource optimization. The research described in this paper on outstable scheduling is also unique. We have not found any research paper documenting this problem yet.

The Outstabling System (OSS)

The input to OSS is a set of trains (usually passenger trains) to be assigned overnight outstabling locations. Because of operational needs, each train will only have a limited number of potential outstabling locations to select from. For instance, these locations might be at or near the station the train will start its morning operations. These sets of potential outstabling locations are predefined and are called "Outstabling Sets" in our system. More than one train may have the same set of potential outstabling locations.

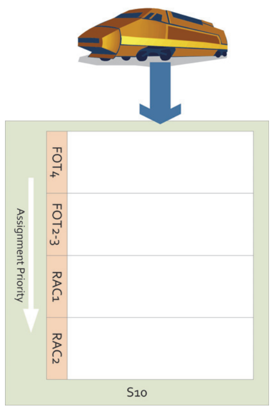

Figure 3. An example of an "outstabling set" used by OSS.

Each "outstabling set" is an ordered list of potential "outstabling locations" for a train. These locations might be adjacent to each other, as in sidings, or at different stations. Each outstabling location can only accommodate one train at a time. For example, Fig. 3 is the "S10" outstabling set, defined for the East Rail Line (EAL). In total, the EAL has roughly 25 such "outstabling sets" defined. In Fig. 3, the outstabling locations FOT4 and FOT2-3 are locations at the Fo Tan Station (FOT), while RAC1 and RAC2 are locations at the Racecourse Station (RAC). The locations within the set are ordered according to preference (or priority). The first empty slot with the highest priority will be selected for the train. Since locations can be at different places, the allocation priority might or might not be related to the actual physical proximity of each outstabling location (Fig. 3 is only a logical view).

Since "outstabling set" is just a list of potential locations to assign to a train, the same "outstabling location" candidate may be included in more than one "outstabling set" and may be considered by more than one train. Fig. 4 shows an example where outstabling locations FOT2-3, RAC1 and RAC2 occur in more than one "outstabling set." Since each location can only be used by one train, if any train is assigned one of those shared locations, then no other train can be assigned to that same location. For example, if the train in "S10" is assigned the location "RAC1", then "S9" and "S22" cannot use the same outstabling location. The label "X (=)" is used to indicate that the outstabling location cannot be used because it is already assigned to another train.

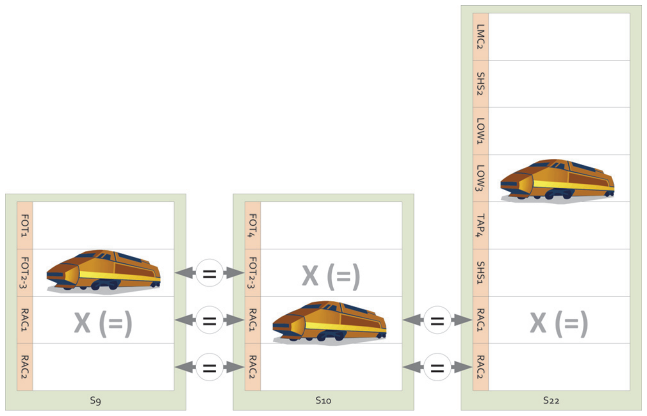

Figure 4. Multiple "outstabling sets" with overlapping outstable locations.

The task of the OSS allocation algorithm is to assign one outstabling location for each train that needs to be outstabled overnight. Besides the above physical constraint, the scheduling algorithm needs to obey a set of additional constraints to ensure outstabling will not interfere with night time engineering works. These constraints are described in the following section.

The AI Model

In OSS, we modeled the outstabling problem as a constraint-satisfaction problem (CSP) (Cohen 1990, Van hentenryck 1989, Steele 1980, Kumar 1992, Tsang 1993). A solution is found through the assignment of values to variables subjected to a set of constraints.

The CSP Model

CSP can be defined as consisting of a finite set of n variables v_1, v_2, ..., v_n, a set of domains d_1, d_2, ..., d_n, and a set of constraint relations c_1, c_2, ..., c_m. Each d_i defines a finite set of values (labels) that variable v_i may be assigned. A constraint c_j specifies the consistent or inconsistent choices among variables and is defined as a subset of the Cartesian product: $c_j \subseteq d_1 \, x \, d_2 \, x \, ... \, x \, d_n$. The goal of a CSP algorithm is to find one tuple from $d_1 \, x \, d_2 \, x \, ... \, x \, d_n$ such that n assignments of values to variables satisfy all constraints simultaneously.

When the railway outstabling problem is formulated as a CSP, each variable v_n represents a train that needs to be assigned an outstabling location for the night. Each variable in OSS is associated with an "outstabling set" that defines the set of all possible outstabling locations that can be assigned; this is the domain of the variable, d_n. The CSP constraints, c_j, are restrictions on how values, i.e. trains, can be assigned to variables, i.e. outstabling locations.
In OSS, since it is a mixed-system problem, the CSP model is implemented as a two-stage search algorithm, where the first stage generates the initial solution. This solution guides the engineering works scheduling. After the engineering works scheduling is finalized, a second-stage iterative repair is performed to "fix" any potential conflicts caused by the engineering works schedule.

Outstabling Constraints

The CSP algorithm for OSS needs to satisfy the following constraint types. There can be many instances of each constraint type in the system.

- **No Overlap** – No two trains can be parked at the same outstabling location at the same time. This is related to the fact that the same outstabling location value may exist in different domains of the CSP variables. If one variable uses that value, other variables cannot be assigned the same value, i.e. outstabling location. This is the common "all different" constraint.
- **Non-Blocking** – Certain collection of outstabling locations cannot be occupied by trains at the same time, to facilitate engineering train movements and/or other operational needs. Two or more outstabling locations can be defined in a "non-blocking" set. The constraint is that at least one of the locations in the set must be clear of any overnight trains. This constraint potentially restricts how other variables can be assigned values through domain reduction.
- **Max Outstabling** – For certain collection of outstabling locations, there can be a "max outstabling" limit that defines the maximum number of trains that can be outstabled in that area. This ensures that the outstabling area will not be too congested and also facilitates overnight train movements if needed.

Fig. 5 illustrates an example of how these constraints are used and the effect of domain reduction. On the links in the diagram, the symbol "=" represents the "no overlap" constraint, "nb" is the "non-blocking" constraint, and "max" is the "max outstabling" constraint. Certain outstabling locations are crossed out due to domain reduction. The label "X (max)" means that location cannot be assigned due to the "max outstabling" constraint. The label "X (nb)" means that location cannot be assigned due to the "non-blocking" constraint. And the label "X (=)" means the location cannot be assigned due to the "no overlap" constraint.

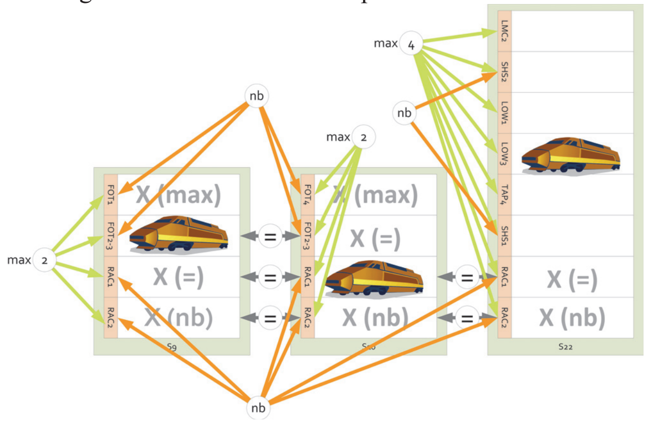

Figure 5. An example to illustrate the "no overlap", "non-blocking", and "max outstabling" contraints.

Take location "RAC1" as an example. It is defined in sets S9, S10, and S22. Linking these sets is the "=" (no overlap) constraint, which restricts RAC1 to be used by only one set. Since a train is already outstabled to RAC1 in S10, neither S9 nor S22 can use that location, indicated by the label "X (=)". In addition, RAC1 and RAC2 have a "nb" (non-blocking) constraint, which means they cannot be occupied at the same time. Since there is a train in RAC1, all RAC2 locations within sets S9, S10, and S22 cannot be assigned to a train, hence the "X (nb)" label. In S10, RAC1 is part of a "max" (max outstabling) constraint, which restricts the total number of locations to be occupied at 2. This reduces the remaining choices of outstabling in S10 to only FOT2-3, which is used by the train in S9.

Search Heuristics

In OSS, we used both "select variable" and "select value" search heuristics to improve search efficiency. The select variable heuristics control the ordering or selection of CSP variables, i.e. v_n, to instantiate. The select value heuristics, on the other hand, control the preferred ordering of values within each domain, d_i, to use for instantiation of a variable v_n.

- **Select variable heuristic** – In OSS, each variable, v_n, represents a train to be outstabled. In selecting which variable to instantiate, OSS uses the minimum domain size heuristics; i.e. the train that has the least number of available outstabling options after domain reduction
- **Select value heuristic** – Once a variable (train) is selected, the algorithm assigns a value (outstabling location) from the variable domain d_i. The value is selected based on a priority level which is usually related to proximity to station that the train will serve next morning

Two-Stage Search Algorithm

The OSS schedule algorithm consists of two stages – the first stage does an initial assignment of trains to outstabling locations based on the CSP model described above (this is done before overnight engineering works are assigned); the second stage is an iterative repair algorithm to adjust the initial plan after engineering works have been finalized.

Some engineering works will require certain tracks to be cleared, so that repair, installation or other work can be done. If an overnight train is already scheduled to be parked there, it must be moved to another outstabling location within the variable's original domain. During the second stage scheduling, any engineering work requests that involve track clearing will be processed. A two stage scheduling algorithm is needed as satisfying all engineering works would most likely be impossible; i.e. there are not enough outstabling locations for all overnight trains. Any engineering work that causes the problem to be over-constrained will need to be rejected. Outstabling will have higher priority. The second stage iterative repair is also needed because engineering work assignment can change dynamically depending on changing business needs.

During the second stage iterative repair search, there is an additional constraint:

- **Mutually Exclusive Track Clear** – Certain engineering works will require some outstabling locations to be cleared of any overnight trains, so that engineering work can be performed at those locations or to allow other engineering trains to pass through those locations. For operational reasons, some combinations of outstabling locations should not be cleared at the same time, i.e. those locations do not allow engineering works to be performed on both tracks at the same time. The "mutually exclusive track clear" constraint defines sets of outstabling locations that cannot be cleared at the same time.

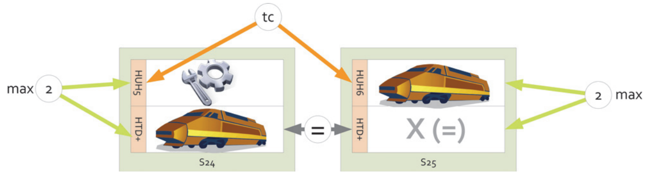

Figure 6. An example to illustrate the "mutually exclusive track clear" contraint.

The example shown in Fig. 6 is taken from the East Rail Line. The symbol "tc" represents the "mutually exclusive track clear" constraint. In the example, outstabling locations "HUH5" and "HUH6" cannot be cleared at the same time. If "HUH5" needs to be cleared for an engineering work (as Fig. 6 illustrates), then "HUH6" should not be cleared for another engineering work, it can either remain empty or be occupied by an overnight staying train. If indeed another engineering works requires "HUH6" to be cleared, then the constraint will be violated. A violation of this constraint does not affect the outstabling plan, but is passed back to the engineering works scheduling algorithm, potentially resulting in the rejection of one of the engineering works.

The AI scheduling algorithm works on top of our CSP model. For our outstabling problem, the first stage schedule is generated by repeated selection of a variable, v_n (a train to be parked) , assigning a value (from its outstabling set), and then verifying the solution with rules (using a rule engine) (Apt 2001). The following is the pseudo code for the first stage of our scheduling algorithm:

```
Let   V be a set of variables
   D be a set of domain values
   C be a set of constraints

OSS_Stage_1_Scheduler(V, D, C):
  while un-instantiated variable in V do:
    v_i = SelectVariableHeuristics (V)
    foreach value x_i in SelectValueHeuristics(D) do:
      if RuleViolation(V, C): continue
      else assign x_i to v_i & Propagate(x_i, v_i): break
      end if
    end foreach
    if v_i is still unassigned: assign none to v_i
  end while

SelectVariableHeuristics(V):
  returns the next un-instantiated variable from V based on
minimum domain size

SelectValueHeuristics(D):
  returns the next value from D based on a set of preference
ordering, usually proximity to a particular station

RuleViolation(V, C):
  returns True if any constraints from C are violated

Propagate(x_i, v_i):
  propagate consequence of assigning x_i to v_i,
```

For MTR, the first stage represents the baseline operation, i.e. how trains will be parked overnight if there are no engineering works (in a highly under-constrained situation). Therefore the first stage scheduling is very fast without the need to backtrack, as trains that cannot be outstabled will be parked in the depot instead. The main challenge is in the problem solving performed in stage two when trains need to be moved around to accommodate the overnight engineering works. The main purpose of stage one is to set up the AI data structures and mechanisms so that stage two can be performed.

After engineering works are assigned or after any changes, the second stage iterative repair will be executed. The repair algorithm reassigns variable values if engineering works require tracks to be clear. In essence, these engineering works define a set of values to be removed from variable domains, i.e. "R" in the following pseudo-code. If the value to be removed is already assigned to a variable, that variable will need to be reassigned another value if possible. If not, then the associated engineering work will not be allowed to be performed that night, i.e. "X" in the following pseudo-code. The set "R" is ordered according to the job priority of the engineering work, to ensure higher priority jobs have better chance of being approved.

```
Let   V be a set of variables
   D be a set of domain values
   C be a set of constraints
   R be a set of domain values to remove
   X be a set of domain values rejected

OSS_Stage_2_Repair (V, D, C, R):
  foreach value x_i in SelectValueHeurstics2(R) do:
    foreach variable v_i in V do:
      remove value x_i from D
      if v_i becomes uninstantiated (i.e. value of v_i is no longer in
          domain) do:
      // repair if needed:
      foreach value x_i in SelectValueHeuristics(D) do:
        if RuleViolation(V, C): continue
        else assign x_i to v_i & Propagate(x_i, v_i): break
        end if
      end foreach
    end foreach
    // reject if no solution
    if V contains uninstantiated variable do:
      reinstate value x_i into D
      add x_i into X
  end foreach
  return X

SelectValueHeuristics2(R):
  returns the next value from R based on priority of the associ-
ated engineering work
```

In the event that the stage two iterative repair algorithm failed to repair the schedule, the engineering work requests that caused the failure will not be allowed to proceed (the set "X" in the above pseudo-code); possibly the engineering works will be rescheduled to be performed in another day (such as a violation of the "mutually exclusive track clear" constraint) or have the outstabled train be moved to the depot instead.

The two stage approach provides some consistency in daily outstabling plans, which makes it easier for train drivers to locate their trains in the morning, and yet at the same time provide some flexibility to overnight engineering works. The alternative is to totally recompute a new solution after engineering works scheduling and after each time an engineering work is changed. Although this single stage approach may lead to a more optimal solution, it is not operationally desirable to do so, as the generated plan may deviate considerably each day and cause confusion and inconvenience to train drivers.

Implementation

The OSS schedule algorithm described in this paper has been completed and is already integrated into the testing environment of the Version 2 of the "AI Engine" at MTR. The OSS and the "AI Engine" are implemented using C# within a Microsoft .NET platform. The "AI Engine" features are provided as web services. Although we coded our own AI algorithms, we believe the design and concepts can be easily adapted and coded in off-the-shelf AI solvers. The OSS is currently undergoing user acceptance testing, and is scheduled for full deployment by mid-2013.

Evaluation

From our testing, the outstabling plan generated is equal if not better in quality to those generated by humans in terms of the total number of trains that can be outstabled each night. However, unlike humans, who might accidentally overlook constraints/guidelines by mistake, the advantage of using an AI approach is that all rules and constraints are guaranteed to be satisfied. In addition, the OSS only takes a few seconds to perform outstabling, whereas a human planner may take a much longer time to resolve all the intertwining constraints. Furthermore, the OSS is integrated into the engineering works scheduling system, so that outstabling can be considered at the same time as engineering works allocation, reducing the need for the outstable planner to negotiate with the engineering works planning team, thus saving even more time. We estimate the complete process can reduce at least one full day of many human planners' time to only a few minutes through AI.

Conclusion

This paper presented our design of a CSP-based scheduling algorithm that performs railway train outstabling. The design is generic enough that it can potentially be applied to railway lines in other cities. The use of AI to perform this task seems unique; we have not seen any other railway line use an automated approach yet.

References

Chun, H.W.; Yeung, W.M.; Lam, P.S.; Lai, D.; Keefe, R.; Lam, J.; and Chan, H. 2005. Scheduling engineering works for the MTR corporation in Hong Kong. In *Proceedings of the 17th Conference on Innovative Applications of Artificial Intelligence* - Volume 3 (IAAI'05), Bruce Porter (Ed.), Vol. 3:1467-1474. AAAI Press.

Abril, M.; Salido, M. A.; Barber, F.; Ingolotti, L.; Tormos, P.; and Lova, A. 2006. Distributed Constraint Satisfaction Problems to Model Railway Scheduling Problems. In *Proceedings of the International Conference on Automated Planning and Scheduling*, The English Lake District, Cumbria, UK, June 6-10.

Apt, K.R.; and Monfroy, E. 2001. Constraint Programming viewed as Rule-based Programming. In *Theory and Practice of Logic Programming*, 1(6):713-750.

Cheung, B.S.N.; Chow, K.P.; Hui, L.C.K.; Yong, A.M.K. 1999. Railway track possession assignment using constraint satisfaction. In *Engineering Applications of Artificial Intelligence*, 12(5):599–611.

Chiu, C.K.; Chou, C.M.; Lee, J.H.M.; Leung, H.F.; and Leung Y.W. 2002. A Constraint-Based Interactive Train Rescheduling Tool. In *Constraints*, 7(2):167-198.

Cohen, J. 1990. Constraint Logic Programming. *Communications of the ACM.* 33(7):52-68.

Martins, J.P.; Morgado, E.; and Haugen, R. 2003. TPO: A System for Scheduling and Managing Train Crew in Norway. In *Proceedings of the 15th conference on Innovative applications of artificial intelligence*, Aug. 12-15, Acapulco, Mexico, pp.25-32.

Otsuki, T.; Aisu, H.; Tanaka, T. 2011. A Search-Based Approach to Railway Rolling Stock Allocation Problems. In *Discrete Mathematics, Algorithms and Applications*, 3(4):443-456.

Peng, F.; Kang, S.; Li, X.; Ouyang, Y.; Somani, K.; Acharya, D. 2010. A Heuristic Approach to the Railroad Track Maintenance Scheduling Problem. In *Computer-Aided Civil and Infrastructure Engineering*, 26(2):129-145.

Peng, F.; and Ouyang, Y. 2012. Track maintenance production team scheduling in railroad networks. In *Transportation Research Part B*, 46(10):1474–1488.

Steele, G.L. Jr. 1980. The Definition and Implementation of a Computer Programming Language Based on Constraints, Ph.D. Thesis, MIT.

Tsang, E. 1993, *Foundations of Constraint Satisfaction*, Academic Press.

Van Hentenryck, P. 1989. *Constraint Satisfaction in Logic Programming*, MIT Press.

Wikipedia contributors (n.d.), "MTR," Wikipedia, The Free Encyclopedia, http://en.wikipedia.org/wiki/MTR (accessed Jan 8, 2013).

MTR homepage (n.d.), http://www.mtr.com.hk/eng/homepage/cust_index.html (accessed Jan 8, 2013).

Wiktionary contributors, "outstable," Wiktionary, The Free Dictionary, http://en.wiktionary.org/w/index.php?title=outstable&oldid=5684250 (accessed Jan 8, 2013).

Sameboat, "File:Hong Kong Railway Route Map en.svg," Wikipedia, The Free Encyclopedia, http://en.wikipedia.org/wiki/File:Hong_Kong_Railway_Route_Map_en.svg (accessed Jan 8, 2013).

Zhang, T.; Andrews, J.; Wang, R. 2013. Optimal Scheduling of Track Maintenance on a Railway Network. In *Quality and Reliability Engineering International*, 29(2):285-297.