A reprint from

American Scientist
the magazine of Sigma Xi, The Scientific Research Society

Trains of Thought

Brian Hayes

GUIDED BY AN UNSEEN hand, a grimy railroad tank car negotiates a series of switch points in the tracks, veering right, then right again, then left. Next comes a lime-green box car, which makes two lefts. I observe these events from the control tower of a railroad facility called a hump yard, where freight cars sort themselves into trains bound for various destinations. It is an eerie scene. The cars glide silently downhill through the maze of tracks, seeming to steer themselves, as if each car knows just where it wants to go. This is an illusion; a computer two floors below me is making all the decisions, setting the switches a moment before each car arrives. But I can't shake the impression that the hump yard itself is a kind of computer—that the railroad cars are executing some secret algorithm.

It's not such a far-fetched notion. In 1994 Adam Chalcraft and Michael Greene, who were then at the University of Cambridge, and later Maurice Margenstern of the University of Metz, designed railroad layouts that simulate the operation of a computer. The machine is programmed by setting switch points in a specific initial pattern; then a locomotive running over the tracks resets some of the switches as it passes; the result of the computation is read from the final configuration of the switches.

These constructions are wonderfully ingenious, although admittedly they have little to do with the day-to-day running of real railroads. Even at a more practical level, though, brawny steel rails and brainy silicon chips have surprisingly rich connections. The work of the hump yard is a case in point. Algorithms for sorting are a specialty of computer science, but railroads were sorting freight cars decades before the first electronic computer was built. Methods invented by rail workers have served as metaphor and inspiration for the development of algorithms and data structures in computer science; conversely, the theoretical analysis of algorithms has suggested ways for railroads to improve their operations.

Computing with locomotives and box cars takes a one-track mind

What a Way to Run a Railroad

Railroads were the epitome of high tech in the later years of the 19th century. Even more than dot-com businesses of recent times, they were a magnet for capital investment and intellectual talent. They dominated the economy; nine of the eleven companies in the earliest precursor of the Dow-Jones stock average were railroads. Technological innovations bloomed: Pneumatic brakes in 1869, automatic signals a decade later. Like the Internet today, railroads transformed aspects of daily life and culture, knitting together distant regions and even changing the way people kept time.

Trains have also become a part of our mental furniture. They appear in paintings, poems, novels, songs, legends and figures of speech; children are still strangely enchanted by them. In the sciences, too, trains have made an impression. Einstein worked out some of his ideas on special relativity by thinking about hypothetical events inside railway carriages. Trains are also common props in the problems found at the end of the chapter in mathematics and physics textbooks.

Somewhat more challenging than a typical textbook problem are various railway-switching puzzles that began appearing in the 1880s. W. W. Rouse Ball presented a few of them in his *Mathematical Recreations and Essays*, first published in 1892. A good example of the genre was discussed at length by A. K. Dewdney in 1987. Eastbound and westbound trains are chuffing toward each other on a single track; they stop just in time to avert disaster, at a place where a short siding parallels the main track. The siding, which connects to the main line through switches at both ends, can hold only one car or locomotive. The question is: Can the trains get past each other, so that both of them can continue in their original direction, pulling the same cars in the same order? (You might want to try finding a solution on your own before reading on or peeking at the diagram on page 111.)

If each train consisted of a single engine, unaccompanied by any cars, the problem would be easy: Just have one engine (say the eastbound one) duck into the siding while the other engine proceeds along the main track; afterward, the eastbound engine can exit the siding and continue on its way without obstruction. In fact, this scheme works no matter what the length of the westbound train.

What if *both* trains have several cars? The trains can still pass, but the crews will have a busy day, coupling and uncoupling cars and throwing switches. The basic idea is to break the eastbound train into single cars and pass them through the siding one at a time. (The westbound train remains intact throughout the procedure.) The eastbound engine can slip by in the way described previously; for the rest of the process, the westbound train does

Brian Hayes is Senior Writer for American Scientist. *Additional material related to the "Computing Science" column appears in Hayes's Weblog at http://bit-player.org. Address: 211 Dacian Avenue, Durham, NC 27701. Internet: bhayes@amsci.org*

all the work. First it moves forward and grabs the leading eastbound car; then the train backs up into the siding and uncouples the car, leaving it there. Now the train performs a maneuver called a runaround, backing out of the eastern end of the siding, driving forward along the main track until it is clear of the western switch, then reversing again into the siding in order to push the eastbound car out and hitch it to the waiting eastbound locomotive. The runaround procedure is repeated for each of the remaining eastbound cars; for each one the crew has to perform three hitching and three unhitching operations, and the westbound train reverses direction twice.

Track Topology

Playing with a puzzle like this one will quickly acquaint you with a salient fact about railroads: They are one-dimensional. Rail cars cannot jump over or go around one another. Indeed, if a rail line has no switch points—if it is an unbranched length or loop of track—then the order of the cars is absolutely invariant. So is their orientation, or polarity; each car always faces in the same direction along the track. As a matter of fact, even in the presence of sidings like the one in Dewdney's puzzle, no rearrangement of the cars can ever alter their orientation; you can shuffle their sequence but you cannot change the direction they are facing. Certain other configurations of tracks and switches, however, do allow the orientation to be reversed.

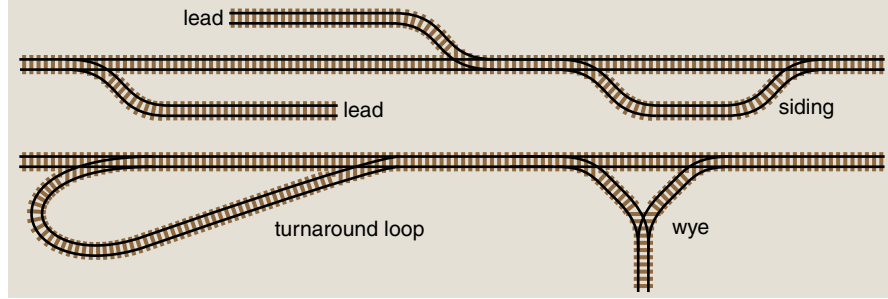
Railroad switches (also known as turnouts, or points) have roughly the same geometry as highway on-ramps and off-ramps:

The action of the switch is asymmetrical. A train departing *A* and moving from right to left can be steered either to *B* or to *C*, but trains traveling from left to right have no choices. The switch does not allow a turn from *B* to *C* or from *C* to *B* (except by going through the switch toward *A* and then backing up). For trains headed right to left, a switch in this configuration is called a *facing-point* switch; for those going left to right it is a *trailing-point* switch.

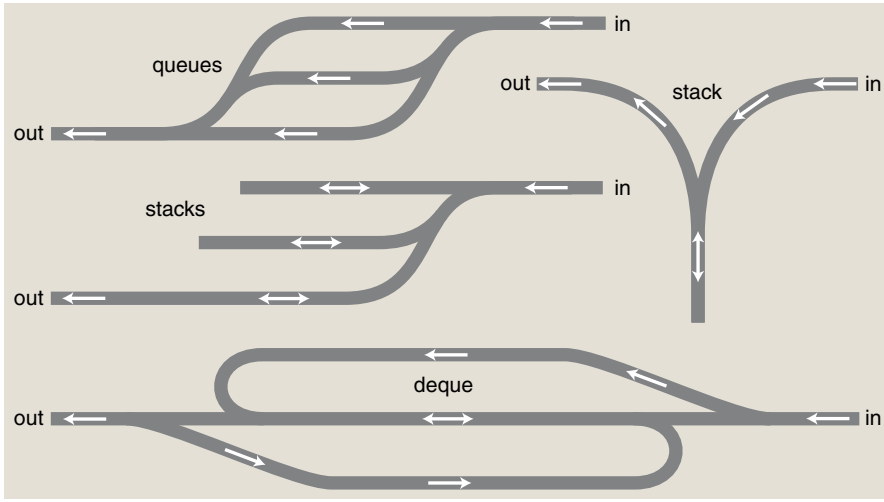
Switches combine with straight and curved sections of track to form the basic structural motifs of railroad layouts. The simplest such element (other than an unbranched length of track) is a *lead*, or *stub*: a dead-end branch connected to the main track by a switch at one end. Depending on a train's direction of travel, leads are entered either head-on through a facing-point switch or by backing up through a trailing-point switch.

Rail cars roll off the hump and into the classification tracks of a freight yard in Linwood, North Carolina, operated by the Norfolk Southern railway. The yard sorts cars onto more than 40 tracks in order to assemble trains bound for various destinations. As each car rolls down a slight grade, a computer identifies it from a radio-frequency tag, regulates its speed and sets the switch points according to the car's destination. (Photograph by the author.)

A *siding*, as mentioned above, is a parallel track with connections to the main line at both ends; the two switches face in opposite directions, which means that a train can run through the siding and return to the main track still going the same way. A *turnaround*



Railroad-track topology can reshuffle the cars and locomotives of a train as it travels through a rail network. Leads and sidings can be used to change the sequence of cars; a jug-handle turnaround loop or a wye can also alter orientation (the direction a car is facing).



Devices and data structures in computer science have railroading counterparts. A *queue* is a first-in, first-out buffer; data elements (or railroad cars) enter and exit in the same order. (But the order can be permuted by the use of multiple queues in parallel, as in the arrangement shown here.) A *stack* enforces the opposite protocol: The first item in is the last one out. The stack at the upper right has the effect of reversing the orientation of each car that passes through it from input to output. The series of dead-end leads at left can also be operated as an array of stacks, which do not change a car's orientation. The elaborate structure at the bottom is a double-ended queue, or *deque*, which allows cars to be added and removed at either end. If engines are allowed to back up, then a simple siding can also function as a deque.

A railroad worker throws a switch lever in a small freight yard in Greensboro, North Carolina, where cars are put in order for delivery. Once the switch is set, the locomotive will couple to and pull out the line of cars farthest to the right. (Photograph by the author.)

loop has a jug-handle shape, with two switches that face in the same direction; turnarounds are rare except at the terminus of a rail line. Finally there is the *wye*, a configuration of three switches joining three tracks, and allowing a train on any track to reach either of the other tracks. Turnarounds and wyes differ from other track elements in that they change a car's orientation: The car can go in facing east and come out facing west.

Some versions of the passing puzzle have the trains confronting each other at a wye instead of a siding. The third track attached to the wye is a short spur, with room for just one car. Despite this change in topology, essentially the same procedure can be used to get the trains past each other. Again the intact westbound train shuttles back and forth, parking the eastbound cars one at a time on the spur, then doing a runaround before pulling them out and shoving them along to the east. But there's a difference: The cars come out of the spur with reversed orientation. Turning them to face in the right direction takes another pass through the wye.

Can the trains pass if all they have to work with is a one-car, dead-end lead? Around 1900 Sam Loyd published a solution for trains of length 4 and 5. It takes 33 reversals of direction.

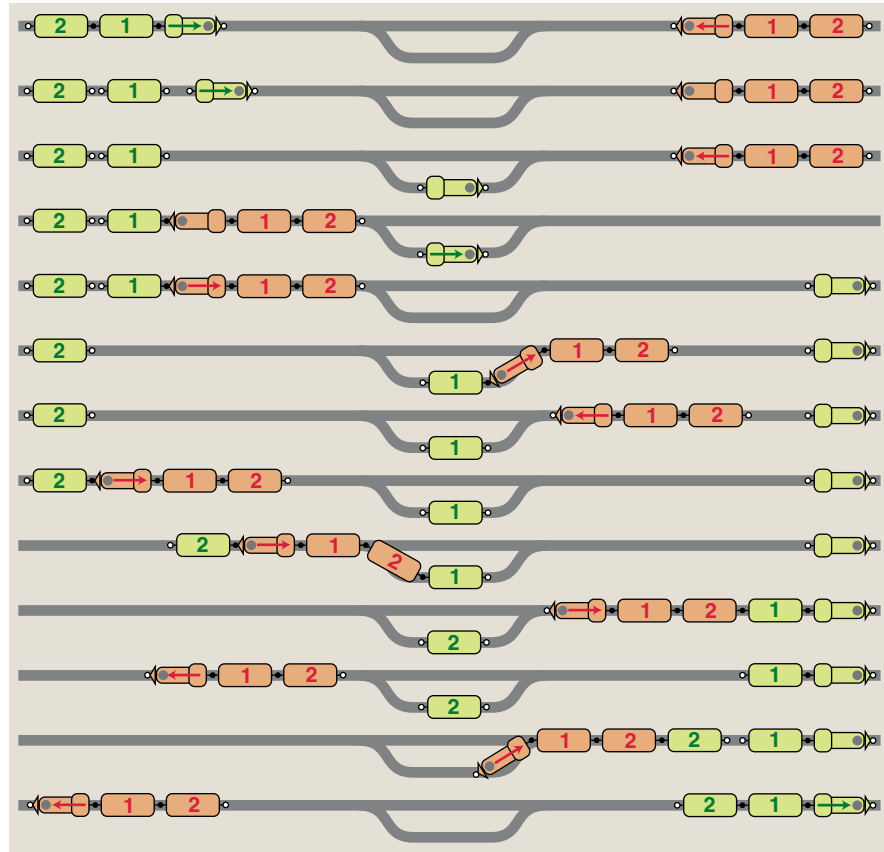
Another famous railroad puzzle asks if a train can make a U-turn at a wye with a one-car-length spur. A suitable unit of measure for the work done in turning the train around is the effort expended in moving a single car through its own length. For a train of n cars, Dewdney gave an algorithm requiring an amount of work proportional to n^3; in essence, n cars move through n car-lengths n times. In 1988 Nancy Amato, Manuel Blum, Sandra Irani and Ronitt Ruvinfeld (all then at the University of California, Berkeley) found an improvement, reducing the effort required to $n^2 \log_2 n$.

Tracking Data

Sidings, wyes and other devices for directing trains inspired some of the earliest ideas about managing the flow of information through a computer program. In 1961 Edsger W. Dijkstra included a diagram of a railroad wye in a memorandum about methods of translating the new programming language Algol 60. In parsing an expression such as $3 \times (5 + 2)$, the seven

symbols are read from left to right, but they must be acted on in a different order: First the subexpression inside the parentheses is evaluated by adding 5 and 2, then the result of this operation is multiplied by 3. Dijkstra showed that the reordering can be accomplished by temporarily storing the operators (such as × and +) in a data structure called a stack. The stack is a first-in, last-out storage device; in this case the × sign goes in first, followed by the + sign, but they come out in the opposite order. Dijkstra chose to explain this principle in terms of a railroad wye. One of the three tracks serves as input, one as output, and the third provides the first-in, last-out storage.

A few years later Donald E. Knuth, in the first volume of *The Art of Computer Programming*, gave railroading interpretations of three important data structures: the stack, the queue and the double-ended queue, or deque. The queue is the simplest of these—at least for railroaders. A queue is a first-in, first-out data structure, and so it is represented by a simple straight length of track. Cars put in at one end of the queue come out the other end in the same order. A deque is an arrangement of tracks allowing cars to be added at either end and removed from either end (but there is no access to cars in the middle of the train). Knuth illustrates a deque by a complicated layout of two back-to-back jug-handle loops, requiring four switches. The same functions can also be accomplished by an ordinary siding, provided that trains are allowed to back up in order to pass through a trailing-point switch. (The siding and the double-jug-handle layouts differ in their effect on the orientation of the cars.)



A train-passing puzzle has variations going back well over a century. The green eastbound train and the red westbound train need to pass each other using a siding that has room for only one car or locomotive at a time. In the solution shown here the westbound train does almost all the work, pulling each car into the siding, going around it, and finally pushing it rearward. The algorithm generalizes to trains with any number of cars.

On the Right Track

Routing and sorting are at the heart of railroad logistics. Cars enter the system from many points of origin, and they must be hauled to many destinations. Other transportation networks, such as shipping lines and the postal system, also sort their cargoes according to destination, but they do not have to deal with the strict one-dimensional constraint of railroad tracks.

Computer science offers a fully-stocked toolbox of methods for sorting—for putting things in order. The textbooks are filled with such algorithms: merge sort, insertion sort, selection sort, shell sort, heap sort, quick sort, bubble sort. It seems there's a sorting algorithm adapted to every imaginable purpose—except maybe the sorting of railroad cars.

When computer scientists evaluate the performance of a sorting algorithm, the usual practice is to measure the mental work done (deciding where each item goes) while ignoring the physical labor of actually moving things. An algorithm is judged to be more efficient if it requires fewer decisions, regardless of how often or how far the data have to be moved. This convention is reasonable when the things being sorted are bits and bytes of data, represented by packets of electric charge with a mass of maybe a zeptogram. The situation is different when you are sorting 100-ton rail cars.

Railroad sorting is usually done in two stages. First, a batch of cars going to the same general area is made up into a train; then the cars within a train are arranged in the best linear order for delivery to their individual destinations. The hump yard is mainly concerned with the first phase of this process. An incoming train is pushed slowly up a hill, and at the crest a worker "pulls the pin" to uncouple each car in turn. The separated car then rolls down the other side of the hill into a fan of diverging tracks. In a large yard there might be 40 or 50 of these "classification" tracks, where trains are assembled.

As each car comes over the hump, switches have to be set to direct it to the correct track. Years ago this was done by workers yanking on iron levers. Other workers, called runners, rode along on the coasting freight cars, cranking the hand brake to regulate speed so that the car would retain just enough momentum to couple with any other cars already on the classification track. The runners and the switch tenders are gone now. An unseen computer sets the switches and controls each car's speed through a mechanism called a retarder, which squeezes a passing car's wheels to slow it down. The speed is measured by radar units much like those used by the highway patrol. Each car is identi-

fied (so that the computer knows where to send it) by a radio-frequency ID tag.

From a computational point of view, the hump yard is an array of queues arranged in parallel. Each car coming over the hump is steered onto a specific track, where the car is appended to the rear of the queue of cars already present there. When the train is complete, a locomotive extracts the line of cars from the far end of the classification track. Because of the first-in, first-out property of a queue, this process does not change the order of the cars within each train. (To be more precise: If car *A* is ahead of car *B* in the incoming train, and if *A* and *B* are both directed to the same classification track, then *A* remains in front of *B* in the outgoing train.)

Solitaire Sorting

The second phase of the freight-car sorting process—putting the cars in order for delivery—is typically done in smaller, local switching yards. These are humpless "flat yards"; the cars are moved by engines rather than gravity. The tracks can be arranged as queues, as in a hump yard, or as stacks—dead-end leads—so that cars have to be pushed in and pulled out from the same end. Suppose a train has cars numbered 1 through *n*, but on arrival at the yard they are scrambled in some arbitrary order; the departing train should have the cars in ascending sequence, with car 1 just behind the engine and car *n* at the end. How many classification tracks are needed to achieve this result? How many times do cars have to be pushed onto and pulled out of the tracks? These are questions of obvious practical importance to railroaders. They are also questions that yield to mathematical and algorithmic analysis.

Robert Tarjan of Stanford University answered some of the questions in 1972. Here are a few of his results:

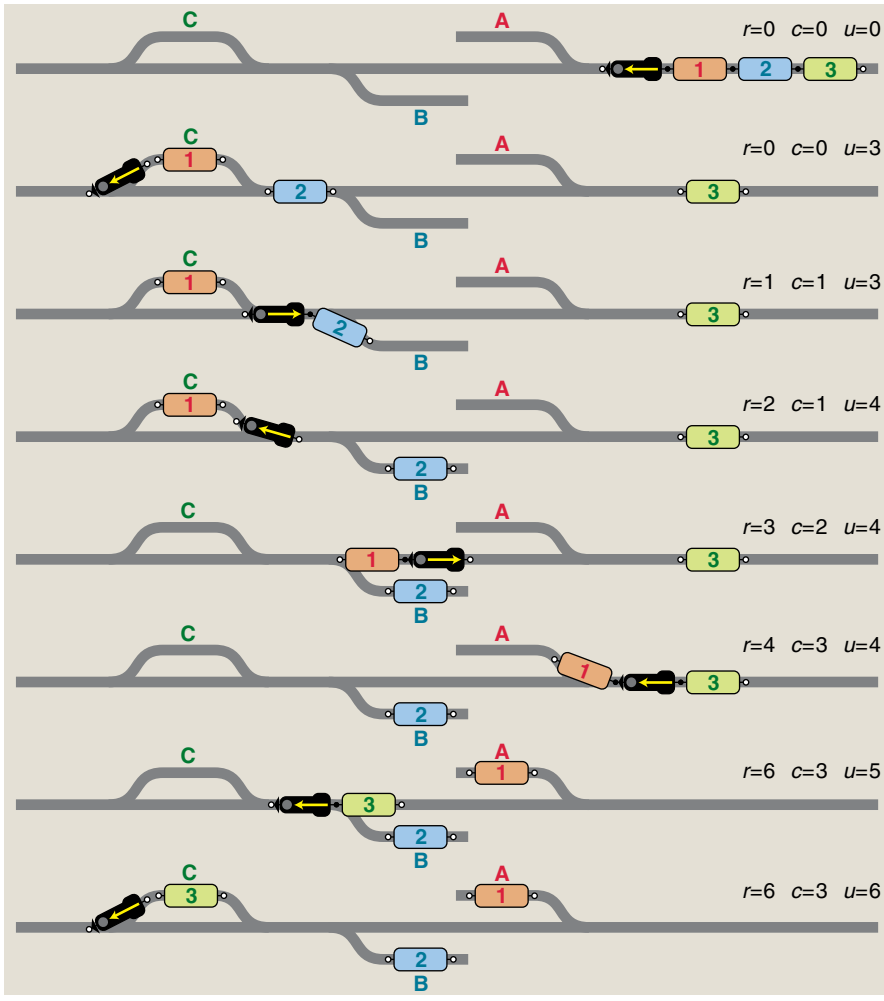
If a switch yard has an internal loop, allowing cars at the output to be brought back to the input for further processing, then *any* sequence can be sorted. Conversely, in the absence of such loops, no finite network of stacks, queues or deques can sort all possible sequences, even if the individual storage elements are of unbounded capacity.

If the yard consists of *m* queues arranged in parallel, then a train can be fully sorted if and only if the longest decreasing subsequence has no more than *m* cars. (The cars of a decreasing subsequence don't have to be consecutive; for example, in the sequence 1 6 3 5 4 9 2 the longest decreasing subsequence is 6 5 4 2.) For a yard with *m* parallel stacks, it's the longest *increasing* subsequence that governs. But these constraints are somewhat artificial. They apply only if cars must always move from the input to a stack or a queue and then directly to the output. Real rail yards are more flexible; cars can be pulled from one stack and pushed onto another. When moves like this are allowed, it's harder to determine which sequences can be sorted.

In 2000 Elias Dahlhaus, Peter Horak, Mirka Miller and Joseph F. Ryan showed that a version of the switch-yard problem is NP-complete (which means, roughly speaking, that there's no efficient algorithm for solving it). Specifically, they proved it is difficult to decide how many tracks are needed.

Chinese mathematicians have taken a somewhat different and more-pragmatic approach to train-sorting problems, apparently in response to a request from Chinese railroad officials. (The exact provenance of these ideas is somewhat murky. In 1976 an American delegation to China heard a lecture on the subject by Ma Chung-fan; notes on this talk were written up by Henry O. Pollak and published in a National Academy of Sciences report. A 1983 paper by Zhu Yongjin and Zhu Ruopeng covers similar ideas but does not mention Ma.)

Pollak's lecture notes present an example: Use an array of stacks to sort the 10-car sequence 6 3 2 4 1 3 5 7 2 6. (Cars with the same number are going to the



Even with the cars of a train sorted in delivery order, getting them to their destinations requires strategy. Car 1 is to be deposited at lead *A*, car 2 at lead *B* and car 3 at siding *C*. The diagram shows only selected steps in the solution. The numbers at the right give the cumulative total of engine reversals (*r*), couplings (*c*) and uncouplings (*u*). The procedure shown includes a total of 15 such events. Can it be done in fewer steps? Would a different initial ordering of the train allow more efficient deliveries? Is any ordering *worse* than this one? The lowest possible score is four reversals, no couplings and three uncouplings: Can this be attained? What if regulations forbid leaving cars unattended on the main line? (Some rules: Each lead or siding holds one car, and there is room for only one car on the main line between *A* and *B*, between *B* and *C* and between the two switches of siding *C*. Cars can be moved only when coupled to the engine.)

same destination and thus should be grouped together.) Here I am going to consider the same example but look at it from a different point of view.

One approach to sorting the example sequence resembles a game of solitaire, building multiple stacks of cars in nondecreasing order. Working from the rear of the train toward the front, we examine the number on each car and push the car onto a stack. Suppose the car we have just reached bears number k. If there is exactly one nonempty stack whose topmost element is greater than or equal to k, then we put the car on that stack. If there are multiple stacks with a top element greater than or equal to k, we choose the stack with the smallest top entry. If no stack qualifies to receive car k, then we have to start a new stack.

For the sequence 6324135726, we begin with the rightmost 6, which necessarily starts a new stack. We push 2 onto the same stack, but the 7 starts a second stack, which can also accept 5 and 3. The 1 then goes on the first stack, and the 4 inaugurates a third stack. Working through the rest of the sequence, we finally reach this configuration of four stacks:

	2		
1	3		
2	5	3	
6	7	4	6

Now the cars can be pulled out of the stacks in nondecreasing order; following the guidelines indicated by the colored blocks, this final assembly step will take seven "pulls." The sorted sequence, of course, is 1223345667.

In his Beijing lecture, Ma gave an alternative sorting procedure; I'm going to call it the Chinese solitaire algorithm. It partitions the sequence in a way that requires just four pulls to assemble the sorted train. Here is the final state of the four stacks:

3			
4			
5	1	6	2
6	2	7	3

It's easy to confirm that this configuration can be reached from the original train order, and that four pulls do indeed yield the properly sorted sequence. But by what rule were the numbers dealt into these particular groups? Both the notes on Ma's lecture and the paper by Zhu and Zhu give a rather convoluted algorithm. In trying to explain it I can do no better than quote the lecture notes:

> Start at the leftmost (in this case the only) 1, put down all 1s, all 2s to the right of the last 1, 3s to the right of the last 2 if you have covered *all* the 2s, etc. In this case, the first subset defined in this way is 12.... The next subset takes the other 2 and the second 3...; it can't get to the first 3. The next subset takes the first 3, the 4, the 5, and second 6; the last subset is 67.

This procedure works, but there's an easier way to generate the same partitioning: Repeatedly scan from left to right, and on each pass extract the longest possible nondecreasing subsequence starting with the leftmost number. In the example considered here, the first such subsequence is 67, followed by 3456, then 23 and finally 12.

Zhu and Zhu give a proof that the Chinese solitaire algorithm allows the train to be assembled with the minimal number of pulls from the classification tracks. But the proof counts *only* pulls. What about "pushes"—the train movements needed to place the cars on the stacks in the first place? For the example sequence, the Chinese algorithm has the worst possible performance in this respect: Ten separate pushes are needed to stack up the 10 cars. The *non*-Chinese solitaire method is somewhat better, at seven pushes. Taking the sum of pushes and pulls, the two methods score a tie at 14. I don't know whether some other technique can do better.

All the Livelong Day

From the mathematical literature on railroad sorting, one might get the impression that putting the train in order is the end of all difficulties. The cars can then be dropped off at their destinations, one by one, without further thought. Train crews tell a different story. A memoir by Ralph E. Fisher, who worked on the Boston and Maine Railroad until the 1950s, refers to the process of making deliveries as a chess game. "Figuring out all these moves required no small skill if they were to be done in the shortest time and the least amount of motion."

Inspired by Fisher's stories, I offer the little puzzle on the opposite page. The task is simply to deliver cars 1, 2 and 3 to destinations *A*, *B* and *C*. The cars are already in delivery order. The procedure shown requires six reversals, three couplings and six uncouplings, for a total of 15 steps. Is there a better solution? Would some other initial permutation of the cars be more efficient? Is there a *worse* permutation?

The chess game of making freight-car deliveries is one aspect of railroading that has gotten easier in recent years. Many of the spur lines used for such local runs have been closed. Much rail freight is now shipped in containers or piggyback trailers that are lifted off the train at a central terminal and delivered by truck. Such "intermodal" transport doubtless has several advantages. One of them is escape from the tyranny of the one-track mind.

Bibliography

Amato, Nancy, Manuel Blum, Sandra Irani and Ronitt Rubinfeld. 1989. Reversing trains: A turn of the century sorting problem. *Journal of Algorithms* 10:413–428.

Ball, W. W. Rouse. 1892. *Mathematical Recreations and Essays*. Thirteenth edition. New York: Dover Publications.

Chalcraft, Adam, and Michael Greene. 1994. Train sets. *Eureka* 53:5–12.

Dahlhaus, Elias, Peter Horak, Mirka Miller and Joseph F. Ryan. 2000. The train marshalling problem. *Discrete Applied Mathematics* 103(1–3):41–54.

Dewdney, A. K. 1987. Algopuzzles: wherein trains of thought follow algorithmic tracks to solutions. *Scientific American* 256(6):128–130.

Dijkstra, Edsger W. 1961. Algol 60 translation. Algol Bulletin Supplement No. 10. Mathematisch Centrum, Amsterdam. http://www.cs.utexas.edu/users/EWD/MCReps/MR35.PDF

Fisher, Ralph E. 1976. *Vanishing Markers: Memories of Boston and Maine Railroading, 1946–1952*. Brattleboro, Vermont: The Stephen Greene Press.

Fitzgerald, Anne, and Saunders Mac Lane. 1977. *Pure and Applied Mathematics in the People's Republic of China: A Trip Report of the American Pure and Applied Mathematics Delegation*. Washington, D.C.: National Academy of Sciences, pp. 42–44.

Knuth, Donald E. 1973. *The Art of Computer Programming: Volume 1, Fundamental Algorithms*; *Volume 3, Sorting and Searching*. Reading, Mass.: Addison-Wesley.

Loyd, Sam. 1914. *Cyclopedia of Puzzles*. New York: Lamb Publishing Co.

Margenstern, Maurice. 2001. Two railway circuits: a universal circuit and an NP-difficult one. *Computer Science Journal of Moldova* 9(1):3–33.

Stewart, Ian. 1994. Mathematical recreations: A subway named Turing. *Scientific American* 271(3):90–92.

Tarjan, Robert. 1972. Sorting using networks of queues and stacks. *Journal of the Association for Computing Machinery* 19:341–346.

Zhu Yongjin and Zhu Ruopeng. 1983. Sequence reconstruction under some order-type constraints. *Scientia Sinica* Series A 26:702–713.