# Module 6: Management—Vehicle Fleets

(Originally compiled by Eric Gonzales and Josh Pilachowski, April, 2008)

(Last updated 4-11-2012)

### Outline

* Introduction
* Schedule Covering 1 Bus Route
  + Fleet Size: Graphical and Numerical Analyses
  + Determination of Terminus Locations and Bus Runs
* Schedule Covering *N* Bus Routes
  + Single Terminus Close to a Depot
  + Multiple Termini and Deadheading Heuristics
* Discussion: Effect of Deadheading
* Appendix A: Proof--LIFO and Greedy Methods use Fewest Buses.
* Appendix B: The Vehicle Routing Problem and Meta-Heuristic Solution Methods

The block diagram below illustrates that a transit agency is in essence a mechanism (the middle box) for transforming money inputs from both, government and users into transportation service. In this course we focus on the workings of this mechanism, treating the arrows pointing into the middle box as constraints and those pointing out as the objective function. We have just finished a set of (planning) modules that explore the ideal structure of this mechanism, focusing on the long term.

We are now about to start a set of modules that will explore what needs to be done in the short and medium term to execute the long term plans. This involves medium-term investment and deployment decisions of the transit agency’s manageable resources, which mainly consist of vehicles and personnel. Invisible to the public, we call these actions “management decisions”. This module deals with vehicle fleet management. Module 7 will deal with personnel management. A transit agency also needs to make other medium-term and short-term operational decisions that are visible to the public. Module 8 will deal with these.

Although our attention will continue to be focused in the middle box we should not lose sight that it is only part of the whole picture. A transit agency is also concerned with the arrows. The issues of finance and governance (inbound arrows) and public relations and information dissemination (outbound arrows) are of much importance to the success of a transit operation. These issues, however, are not transit-specific and will therefore not be addressed in these notes. So let us now return to the inside of the box, with a focus on management.

$

Government

Transit Agency:

* Experts
* Labor
* Infrastructure
* Vehicles

Users

Service

# *Introduction*

To this point, in the planning part of the course, we have assumed that agency costs (including operating costs and amortized capital costs) are proportional to the vehicle hours and vehicle kilometers of service provided. This would be exact if vehicles could be rented for only the time that they are needed for use in service, and drivers could be hired and fired so that people only worked the hours that buses are in operation. In reality, vehicles are purchased or leased for more than a few hours at a time and labor unions place restrictions on the number of hours that drivers can work. In this and the next module we will develop vehicle operating plans and driver staffing plans recognizing these limitations. Homework exercises will compare these more realistic operating costs experienced by the agency and those assumed in the planning stage. We will find that the assumptions made during the planning part of the course were not bad approximations.

### Definitions

These definitions will be used in the two management modules.

Schedule – set of routes and scheduled services advertised by the transit agency.

Depot – location where buses are stored without drivers.

Terminus – part of a transit line (i.e., route) where buses are empty of passengers and buses can be changed. Routes often have two termini.

Interchanges – points on a transit line where drivers can be interchanged. Termini are usually interchanges, as is the depot. Because drivers can be changed with buses full, routes can also have interchanges that are not termini.

Bus task – smallest part of the schedule that must be covered by a single bus. It usually consists of a bus trip between consecutive termini of a single route.

Loop – part of the schedule that consists of a bus trip between consecutive visits to *the same* terminus of a route. If a route has only one terminus, bus tasks are loops. Otherwise loops can be divided into tasks.

Bus run – time-space path of one specific transit vehicle from and returning to a depot. The vehicle needs a driver during the entire run. The run may include coverage of more than one transit line. The driver may change.

Driver Task – indivisible part of a bus run that must be covered by the same driver. It usually consists of a bus trip between consecutive interchanges.

Worker Type – work pattern characterized by pay rate and properties of the shift; more about this later.

Job – sequence of driver tasks covered by one specific worker type in a single day.

Allocating vehicles and drivers to provide the schedule promised by the transit agency is a two-step process:

1. Find a fleet operating plan to cover the schedule 🡪 {# of vehicles, specific runs}. This involves dividing the schedule into bus tasks and then allocating buses to cover the tasks. In the end some vehicles may have to redeployed (deadheaded) or sit unused part of the time.
2. Find a staffing plan to cover the runs 🡪 {# of workers by type, specific jobs}. This involves cutting the runs found in step 1 into driver tasks and then allocating workers to cover the tasks. In the end some workers may have to be idle part of the time.

These steps are parallel in structure. Both answer the question: how many items are required to cover a set of requirements? This Module is concerned with step 1.

### Schedule covering: 1 bus route with a single terminus

The data for this problem (the schedule) can be represented in a time space diagram showing each of the buses traveling along a route from a terminus (at *x* = 0) and looping back to it. Each bus requires a cycle time *T* to make a full loop of length *L* and return to the terminal. Then, we define *N*(*t*, 0) as the cumulative number of dispatches from the origin over time (also denoted *D*(*t*)), and *N*(*t*, *L*) as the cumulative number of returns, which is *D*(*t–T*) if the cycle time is fixed.



### Fleet Size: Graphical Analysis

We analyze this system as a queuing system from the perspective of the terminus—imagining for the time being that the depot is on top of the terminus and that the depot supplies buses when needed. What is the minimum number of buses needed to sustain the schedule?

Each bus can be classified as waiting in reserve at the terminal until dispatch or circulating in service. The transitions between reserve and circulation are the dispatches and returns.



A cumulative plot of buses available *A*(*t*), buses dispatched *D*(*t*), and returned *R*(*t*) shows graphically how the number of buses in reserve and service evolves over time. See below. The curve *D*(*t*) is given and the other two are derived. The number of available buses is equal to those initially available, *M*, and those returned: *A*(*t*) = *M* + *D*(*t* – *T*). Note how for this closed queuing system the sum of reserve and circulating buses is the total number of vehicles, *M*, which remains constant over the course of the day. Note: curve *A* is obtained from curve *D* with a vector shift (*T*, *M*).

Since the number of buses in reserve has to be positive, we require: *A*(*t*) ≥ *D*(*t*); so the minimum fleet size is obtained when *A*(*t*) and *D*(*t*) are tangent, as shown.



The tangency point is where the cumulative dispatch and cumulative return curves are maximally separated; i.e., where the number of circulating buses, *U*(*t*) is maximum. So, we have:

.

Note from the figure that *U*(*t*) is the number of buses to have been dispatched in the interval (*t−T*, *t*]; i.e., *D*(*t*) − *D*(*t* – *T*).

It is possible to obtain bounds for *M* without knowing the shape of *D*(*t*) by looking at the headways. We know that the number of buses in circulation *U*(*t*)must be maximized immediately after a bus dispatch because until the next dispatch the number of buses in circulation can only decline. The number of dispatches in the interval (*t−T*, *t*] where *t* occurs immediately after a dispatch, i.e. the value of *U*(*t*) at said instant, is at least 1. Therefore, since the number of dispatches in (*t−T*, *t*] is inversely related to the average headway in (*t−T*, *t*], we see that *U*(*t*) must satisfy:

, ∀ *t*,

and therefore,  also satisfies:

.

You can convince yourselves that if the headways vary slowly with times comparable with the bus cycle, then the upper bound is tight. Furthermore, in the time-independent case, where the headways do not vary at all, the lower and upper bounds coincide and we obtain the well-known result:

.

*Fleet Size: Numerical Analysis*

Here we show how to calculate *M* when the cumulative dispatch curve *D*(*t*) is known and the above bounds are insufficient. We first consider a (tentative) fleet size *j* and determine if it is feasible. Note from the picture below that if *Tn+j* – *Tn* ≥ *T* for all *n*, then *A*(*t*) is always to the left of *D*(*t*) for all *t*, as we want so that *j* is feasible. This inequality merely ensures that every bus is available before it is dispatched again. The inequality holds if and only if there are no more than *j* dispatches in time *T*, which is the feasibility condition we used above to develop the headway-based bounds.



The minimum fleet size is the smallest feasible *j.* It can be determined with a spreadsheet such as the one below that checks the feasibility condition, *Tn+j* – *Tn* − *T* ≥ 0, for different tentative fleet sizes, *j*. The lowest value of *j* corresponding to a column with all values greater than or equal to 0 is the sought fleet size. With the found fleet size the reserve of buses is never empty.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | *j* = 1 | *j* = 2 |  |
| *n* | *Tn* |  |  | … |
| 0 | time data |  |  |  |
| 1 | time data |  |  |  |
| 2 | time data |  |  |  |
|  |  |  |  |  |

*Terminus Location*

So far, we assumed that the terminus was at *x* = 0. Could it be possible to reduce the number of buses by locating the terminus in a different place along the route? The answer is no if bus trajectories are the same through the day. Here is why: Let ∆*n* be the travel time of bus n from the old to the new terminus (see figure) and note that the time when bus *n* is dispatched from the new terminus is:. Now, if , then  because ∆ cancels out and we can put the terminus wherever we want without a penalty. This is good because it gives us the feasibility to put the termini at favorable locations (e.g. where buses are nearly empty).



*Run Determination*

The bus scheduling problem is to determine which bus is associated with each bus task (or loop since we have only one terminus); i.e. figuring out what each bus will do: the bus runs. We discuss two heuristic methods for determining the specific run for each vehicle, but there are many more. In method 1 we step through time and as tasks arise we use a rule to choose a bus from those that are available. We examine here the last-in-first-out (LIFO) strategy; new buses are only introduced when absolutely necessary. With LIFO, as we step through time and find that a task is scheduled to start, we assign to it the bus that returned most recently. If there is no such bus then a bus is selected from the initial pool. This strategy is good because it keeps some individual buses running while others experience long periods of idling. The latter can be returned to the depot for driver relief.

An alternative strategy with the same goal is a greedy strategy that steps through buses, assigning to each bus consecutive tasks (loops) until one reaches the end of the day. In other words, each bus is redeployed as early as possible after returning to the terminus and the first possible task is then assigned to it. After each bus is processed in this way, the tasks covered by the bus are removed and the next bus is processed. The procedure is so simple that it can be implemented graphically by hand by plotting each scheduled loop from the route’s terminus against time as shown below. Could you organize this in a spreadsheet?



It is perhaps intuitive, and can be proven (see Appendix A), that both the LIFO and greedy methods are feasible with the minimum fleet size we calculated earlier:

*.*

*Example: Qualitative difference between LIFO and Greedy Methods*

The simple three-run, two-bus system below shows why the two methods differ:



The LIFO method assigns the third loop to the bus returning most recently (Bus 2). The Greedy method assigns it to the lowest indexed available bus (Bus 1). The methods only differ in the bus that is selected from the pool of idle buses for the next run. Since we are assuming that all buses are identical, the choice has no future repercussions on the availability of buses. In fact, one could have selected the bus at random, or with any other rule, and the strategy would perform similarly. Thus, the specific bus choice can be made with other (non-bus) criteria in mind.

### Schedule covering N bus routes

Imagine a map of many routes all passing converging at a centrally located terminus. We could imagine this is a bus station in the center of a city or at a busy rail transfer station. The terminus may be close or far from the depot. We imagine for now that it is close.



*Single Terminus Close to the Depot*

We could treat each route independently as before, but on the other hand, it may be possible to reduce the fleet size by sharing buses between routes. On the left below is the model for dedicating bus fleets to separate routes in isolation. On the right, this model is modified so that rather than a reserve of buses for each route, the terminus holds a reserve of available vehicles for all routes. Each route, *i*, is characterized by loops of different cycle times, *T*(*i*), so the time until a dispatched bus returns is no longer uniform but depends on which route the bus has been dispatched.



The aggregated cumulative count of dispatched and returned vehicles is now expressed as





And since the fleet (*M*) is shared, the cumulative number of buses made available for collective use is still:



Curves *D*(*t*), *R*(*t*), and *A*(*t*) can be plotted as before to determine the minimum fleet size, and the formula *M* = max{*D*(*t*)–*R*(*t*)} continues to hold. The only difference is that *R*(*t*) is no longer related to *D*(*t*) by a shift.

The main advantage of sharing is that the necessary fleet size is always less than or equal to the one without sharing. The reason is that the assignment obtained without sharing can also be achieved with sharing, but sharing allows us to do more things.

*Dispersed Termini and Deadheading Heuristics*

Consider now the case where the termini and depot are dispersed, and there can be more than one terminus per line. Perhaps the termini are at ends of the lines, and there may be some cost, *ckk’*, of moving a bus from task *k* to task *k’*. Recall that a bus task is the smallest part of the schedule, which cannot be subdivided anymore and must be covered by the same bus; i.e., a trip between consecutive termini and not necessarily a full loop. In what follows, *k* and *k’* index bus tasks. To include deadheading from the depot, we use *k* = 0 for the depot and *k* = 1, 2,… for the bus tasks.



A simple heuristic method can be used to solve this problem approximately. This method is good if *ckk’* << *T*(*k*)., where *T*(*k*)  is the duration of task *k.* Otherwise, it produces solutions that may need improvement. We imagine that all tasks *k* are started a time



earlier than their real start time. This buffer that can accommodate the deadheading time required to pull a bus from any other terminus *k’*.[[1]](#footnote-1) The duration of the imagined bus tasks, which include deadheading to pull a bus from anywhere, plus the performance of the actual task, are then:



Since deadheading is included as part of the tasks, we can treat this new problem (with *T’*(*k*)) as previously, with zero deadheading times. This is a way to obtain a tentative fleet size and set of bus runs which can be improved using a computer.

Fortunately, the problem we are solving is analogous to the vehicle routing problem (VRP); a famous problem that has been extensively studied. So, we don’t have to do this from scratch. (Appendix B gives some background on the VRP and a simple computer method that can be used to improve tentative solutions.) The VRP is analogous to the schedule covering problem that we want to solve because we are looking for the least costly way to cover a set of requirements. The analogy is presented in the table below.

The penalty can be defined by any function that maps idle time between bus tasks to a penalty. This may be a function that increases as the idle time wastes money until some point when the bus can be returned to the depot.

|  |  |
| --- | --- |
| *Vehicle Routing Problem (VRP)* | *Schedule Covering Problem* |
| points *i*,*j* | bus tasks *k*, *k’* |
| distance, *cij* | penalty, p*kk’* |
| vehicle | Bus |
| vehicle load | tasks covered by a bus; i.e., the bus run |
| capacity | ∞ |

### Discussion: Effects of Deadheading

To illustrate the potential benefits of deadheading, suppose we have two bus lines with different peaking patterns, such as a commuter route running heavily in the morning and evening, paired with a route that is run most heavily during the middle of the day for something like an athletic event. The figure below displays these patterns by means of two solid curves. The dotted line (not drawn to scale) is the sum of these curves.



Compare the fleet requirement if the routes were considered separately (the sum of the maximum route requirements considering each route individually) to the fleet requirement if the two routes share resources even if deadheading is required (the maximum of the sum of route requirements given by the dotted curve). It always happens that:



So, some savings are possible with deadheading, but these are offset against the cost of deadheading itself. The greatest benefit is from routes that peak at different times.

**Appendix A: Proof--LIFO and Greedy Methods use Fewest Buses.**

Any method that steps through time, such as the LIFO method, is feasible with only *M* buses because, as the text proved with queuing diagrams, the pool of idle buses always includes at least 1 bus. Thus, the focus here is on the greedy algorithm.

Consider the Gantt spreadsheet of the text with all the tasks plotted versus time, and the set of times where a critical number of tasks,, have to be simultaneously performed. This set will be the union of one or more (critical) time intervals. Note that at least a new task must begin at the beginning of each one of these intervals for otherwise the interval could be lengthened. Therefore, because the algorithm is greedy, the first bus must enter the first interval with a task assigned to it (i.e., in a busy state) for if the bus was idle immediately before the start, it would be assigned the task that starts at the interval’s beginning. Now, also note that if a bus enters an interval in a busy state it must remain busy until it exits because any task that ends within an interval must be immediately followed by another (otherwise the number of tasks in the interval would decline, which is impossible). Thus, on finishing a task the bus would continue with another because the algorithm is greedy. Therefore, the bus remains busy as it crosses the first interval, hence decreasing the number of simultaneous tasks in said interval by 1.

For the same reasons, the first bus must cross the second, and all other critical intervals, in a busy state. Thus, the bus also reduces the number of simultaneous tasks left to be processed in these intervals by 1. As a result, the number of critical tasks remaining after removing the tasks processed by bus 1 is *M*−1.

When bus 2 is considered the problem is analogous to the original albeit with *M*−1 critical tasks. Thus, bus 2 also reduces the number of critical tasks by 1. The same happens for buses 3, 4, etc. so that when *M* buses have been processed, the number of critical tasks is 0; i.e., all the tasks have been completed. Therefore the greedy algorithm is feasible with *M* buses.

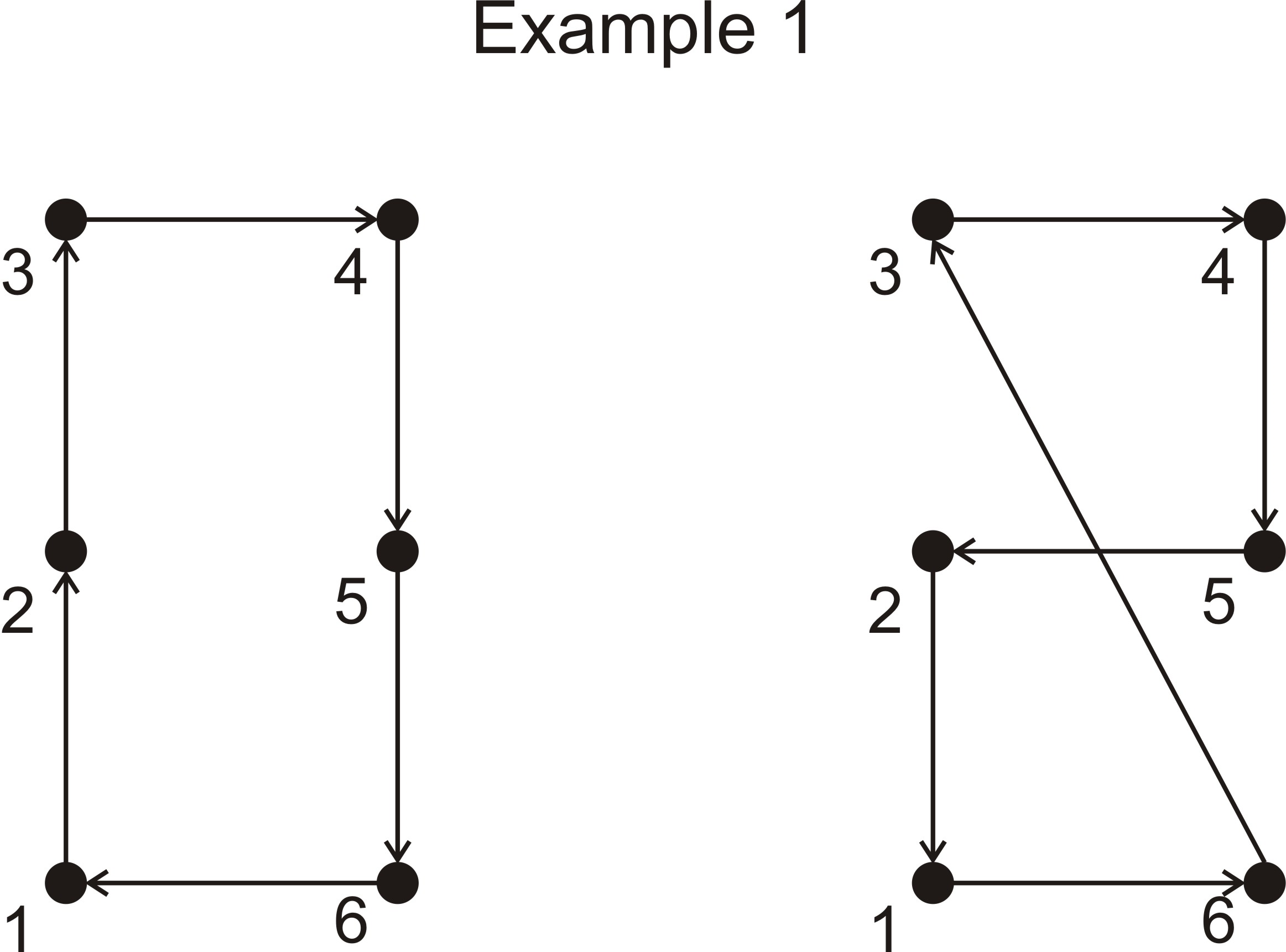
**Appendix B: Introduction to the “Vehicle Routing Problem” and Meta-Heuristic Solution Methods.**

Here we describe some combinatorial optimization concepts that are useful for scheduling public transportation workers and vehicles.

***Local Search Methods and Meta-Heuristics***

The basic idea behind local search methods is to guess solutions that get increasingly better as the procedure develops. Solutions are characterized by a “state” which is a string of numbers. This can be illustrated with the TSP. Given are *N* points (or cities), *i* = 1, 2…. *N*, and a matrix of distances {*eij*} between every pair of points. In the TSP, we look for a tour that visits all the points with the least total distance (or “cost”). Since the positions in which the cities appear in a tour are uniquely defined by an ordering of the first *N* integers (a permutation), any such ordering is a state of the TSP problem. Example 3 below shows 2 possible states for a 6 point TSP problem: (1, 2, 3, 4, 5, 6) and (1, 6, 3, 4, 5, 2). It is assumed in this example that costs are given by the Euclidean distances of the links. Thus, the cost of each state is the length of the tour one would measure with a ruler.

Any “local search” is based on perturbations that transform a state into a similar state, hopefully with lesser cost. For the TSP, a perturbation could be choosing 2 consecutive cities and swapping their order. For example, from (1, 2, 3, 4, 5, 6) we could go to (1, 3, 2, 4, 5, 6,) and from this to (1, 3, 4, 2, 5, 6). The set of states that can be reached in one step (one perturbation) is the state’s local neighborhood. Perturbations should be simple (so they are easy to make and evaluate), but also comprehensive, in the sense that they should allow the system to reach any state from any other state. Consecutive city swaps have these two properties and are therefore acceptable perturbations for the TSP.



Given a current state, a “greedy” local search would evaluate the cost of all the states in its neighborhood and move to the one with the least cost if such a state exists; otherwise the search ends. This procedure is then repeated using this new state as the current state, and then repeated iteratively until the search ends because no improvement can be found. The termination point is called a “local” optimum. Local optima are generally not unique for the TSP. For example, you can verify that the two tours of Example 3 are locally optimal, even though tour (1, 6, 3, 4, 5, 2) on the right is quite bad.

In view of this, people have created “meta-heuristic” methods that in theory can avoid being trapped in local optima and converge to the global optimum. The simplest meta-heuristic method is called simulated annealing (SA). It differs from the greedy method in that it randomly chooses a single perturbation from the current state to identify a single new state. A coin is then flipped to see whether the new state is accepted and becomes the new current state, or one stays put. The probability of success “*p*” is chosen to be the following function of the change in cost, Δ*e*, and the iteration number, *n*: *p* = 1, if Δ*e* ≤ 0; but if Δ*e* > 0 then *p* = exp{-(*n+a*)Δ*e*}, where “*a*” is a positive constant. Note that at the start of the search (*n* = 1) there can be a significant probability of accepting a more costly state (with Δ*e* > 0) but this probability declines as the simulation progresses. This probabilistic feature of SA allows the algorithm to jump out of local optima and, given enough time, to converge to the global optimum. Unfortunately convergence is slow for problems with more than (say) 100 points. Even in these cases, though, the method can be used to fine tune solutions obtained with other methods. A large value of “*a*” is normally chosen for this type of application.

***The Vehicle Routing Problem (VRP)***

The VRP arises in practice more often than the TSP, and many variants of it exist (e.g. with route length restrictions, time-windows, etc.). In its most basic forms it seeks vehicle routes to serve a set of *N* customers distributed in space. Customers have items to be carried, which take up vehicle space. Vehicles have finite capacity.

*Given are:*

*N* points, *i* = 1, 2… *N*

*M* vehicles, *m* = 1, 2… *M*

A deport at *i* = 0

A matrix of distances, *eij*

A demand di for every point (city) (in units of “quantity”)

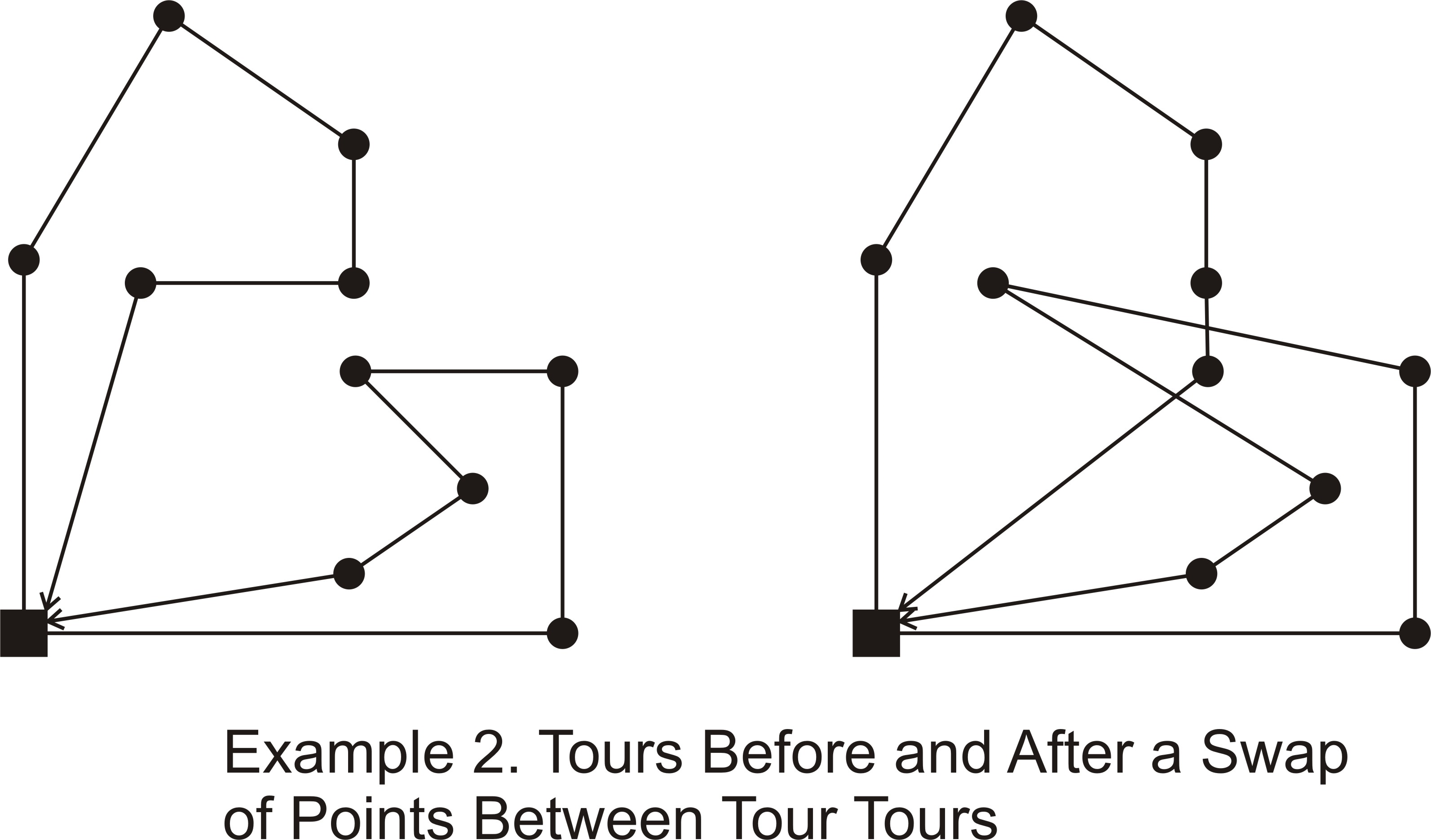
A vehicle capacity, *Vm* for every vehicle (also in units of quantity)

*We look for:*

An allocation of points to vehicles and a set of vehicle routes ending and beginning at the deport that minimizes either vehicle distance, number of vehicles or a combination of the two.

The VRP can also be attacked with meta-heuristics such as simulation annealing (SA), and these techniques still give reasonable results for problems with up to (about) 100 points. Instead of a single permutation, a “state” now consists of an ordered allocation of cities to vehicles. Note, some of these states may be infeasible--if the total demand for vehicle m exceeds *Vm*.

The SA algorithm would work as before. One defines perturbations, which can be swaps of points (also called “customers”) within a tour, or swaps of groups of customers among tours. Example 1 shows the result of swapping the last customer of the tour on the left with the middle customer of the tour on the right. It should be clear that any state whatsoever can be reached from any other state if one uses a proper sequence of swaps. Therefore, the SA approach with random swaps should (theoretically) work. In practice, experience with the VRP has been good with problems as large as ~100 points. For larger problems SA can be used as a fine-tuning tool with a large value of its parameter “a”. A demonstration of this approach can be found in Robuste, et al. (1990), which applied the SA annealing algorithm to a problem with about 200 points).



As is explained in the text, many transit problems can be cast in the form of a VRP-like problem that can be solved or fine-tuned with SA. This technique can be quickly mastered and applied. The case study in Robuste et al (1990) took less than 1 week from conception to completion.

***More Information:***

The following elementary readings could be of use. Section 10.9 of “Numerical Recipes: The Art of Scientific Computing” by W. Press et al., Cambridge 1987, pp. 326-334, describes simulated annealing in the context of the Traveling Salesman Problem (TSP), and shows some computer code. A short description can also be found in Appendix B of Daganzo (2005), *Logistics Systems Analysis*, Springer. Section *4.5.2* of this reference (*Fine-tuning Possibilities*) summarizes the case study in Robuste et al, (1990).

1. If buses are not shared across routes, the maximum operation in the definition of Δ(*k*) should only be taken over the set of termini from which buses can be pulled to begin task *k*; i.e., termini of the same route. Thus, the values of Δ(*k*) will be generally smaller if buses are not shared than if they are shared [↑](#footnote-ref-1)