EASY AND HARD PROBLEMS IN OPERATIONS RESEARCH

Martin Butler and Derek O'Connor*

The development of operations research since the second World War has been closely parallel to that of Computer Science. Although this statement could be made of any two scientific disciplines, because of the general increase in scientific activity, it is particularly true in the case of operations research and computer science. Both subjects were "born" at the same time and have contributed to each other's development and today there is a large area of operations research that could be classified as computer science and vice versa. For example, the operations research techniques of queuing theory and scheduling theory have been extensively used by computer scientists in the development of operating systems.

The main thread connecting the two subjects has been computation. Operations researchers need to solve practical problems and they require computers to do this. Linear programming and Dantzig's (1951) Simplex Method would be no more than a theoretically interesting remark were it not for the ability of computers to solve such problems. In fact, J.B.J. Fourier in 1824 posed a linear programming problem and outlined a simplex method for solving it [Grattan-Guinness, 1970]. Dantzig persevered in his development of linear programming because he knew that digital computers were being developed and that these could solve real problems formulated as linear programmes. As a result, the advent of commercial computers brought with it a great increase in the use of operations research techniques and in particular the use of linear programming.

Linear programming has been and still is the most successful tool used by operations research. It has been applied to a seemingly endless variety of problems: real-time control of chemical plants and oil refineries; cutting paper to reduce waste; managing growth in forests; mixing feed for livestock; and optimally assigning personnel to jobs. Other tools, such as simulation, queuing theory and PERT, have been successful but not to the same extent as linear programming.

*The authors are Lecturers in Management Information Systems in the Department of Management Information Systems at University College, Dublin.

IBAR – Journal of Irish Business and Administrative Research, Volume 6, No. 2, October, 1984, pp 36-52.

Along with these successes there have been failures. Although many of these have been failures of *implementation* rather than *method*, there are many, apparently simple, problems that have not been solved by operations researchers. Consider the following two problems.

Problem 1: Find the shortest route for a postman delivering letters so that he starts and ends at the sorting office.

Problem 2: Find the shortest route for a travelling salesman visiting customers so that he starts and ends at his home office.

These two problems are apparently similar and they are easy to state and understand. They are also practical problems – they are posed and "solved" every day. Despite their apparent similarity these problems are fundamentally different: Problem 1 is easy to solve whereas Problem 2 is virtually impossible to solve generally.

The purpose of this article is to discuss these *easy* and *hard* problems and to show that many problems in operations research can be classified, in a precise way, as easy or hard. We will see that this classification depends on the ability of computers to solve these problems in a 'reasonable' amount of time. This is why computer scientists are interested in such problems and have contributed much to this area of research, viz., computational complexity.

Problems, Algorithms and Complexity

The classification of problems as easy or hard requires the auxiliary notions of *problem size, algorithm* and *reasonable amount of computation time*. We now explain these notions by way of two very straightforward problems.

Problem 3: Given a set of 100 cards, each with a 4-digit number stamped on it, find the card with the largest number.

This problem can be solved as follows:

Step 1. *Pick up any card.*

Step 2. *Pick up a second card and compare it with the first. If the second card is greater than the first then discard the first card. Otherwise discard the second card.*

Step 3. *Repeat Step 2 for the 3rd, 4th, . . . 100th cards.*

The card remaining in your hand has the largest number on it.

This method is guaranteed to work and it will give the correct result in a finite amount of time. Such a method is called an *algorithm*. The method obviously requires that we examine each of the 100 cards (if we ignored one card it could have the largest number on it). It is also obvious that if we increase the number of cards to 200 that the time to find the card with the largest number also increases. We say therefore, that the *problem size* is 100 in the first case and 200 in the second case. If we assume that it takes 5 seconds to pick up and examine each card then the algorithm (method) will take 500 seconds (8 minutes) to solve a problem of size 100 and 1000 seconds (16 minutes) to solve a problem of size 200. We consider such computation times to be *reasonable* (1000 years would *not* be reasonable) and say that *a problem is easy if it is solvable by an algorithm that requires a reasonable amount of computation time.*

Problem 4: Given a set of 3 cards with the letters A, B, C printed on them, generate all possible arrangements of the three letters.

This problem can be solved as follows:

Generate the arrangements by adding letters one-at-a-time to each partial arrangement as shown in Figure 1. The dots show the positions where each additional letter can be put. This algorithm generates all permutations (arrangements) of the three letters A, B, C and can be used to generate all the permutations of any number of distinct objects, in a finite amount of time. If it takes 1 second to generate each permutation then this permutation problem of size 3 (letters) requires 6 seconds. If we double the size of the problem to 6 letters then it requires 720 seconds (12 minutes) to generate all permutations. If we wanted to find the permutations of all the letters in the alphabet then the algorithm would require 1.28×10^{19} years of computation time. This amount of time is most definitely *unreasonable*. Even if we had a computer capable of generating 1 million permutations/second it would still require 1.28×10^{13} years. (This is longer than the age of the known universe).

Figure 1: *Generating Permutations*

A	.A.					
B	.A.B.			.B.A.		
C	.C.A.B.	.A.C.B.	.A.B.C.	.C.B.A.	.B.C.A.	.B.A.C.

There is a very striking difference between Problems 3 and 4. Although both are easy to state and both are solved by simple algorithms, the

amount of computation time for each is vastly different as the size of each problem increases. The difference between the two problems is not because one algorithm is efficient and the other is inefficient. In fact each algorithm is optimal (the best possible) for the problem it solves. The difference is simply this: one problem is *easy* and the other problem is *hard*. In more formal terms we say that finding the maximum of n numbers (Problem 3) is *computationally tractable* (easy) and generating the permutations of n objects (Problem 4) is *computationally intractable* (hard).

The problem of generating permutations is obviously hard in the computational sense because, of necessity, all n! permutations must be generated. Therefore, any algorithm that generates the n! permutations must spend a time that is at least proportional to n! in doing this. We call such problems *obviously hard*. Apart from such obviously hard problems, there is a large class of problems that are not obviously hard and for which there are no *known* algorithms that solve them in a reasonable amount of time. This class of problems, technically called 'NP Complete', [see Garey and Johnson, 1979] has been the subject of intense interest because it includes such important problems as the travelling salesman, vehicle routing, assembly line balancing and the school timetable problem, to name but a few.

The curious feature that distinguishes this class from *easy* and *obviously hard* classes of problems is this:

> *Finding the solution to a problem in this class is difficult but verifying that the solution is correct is easy.*

For example, it may take many hours of computer time to find, for a travelling salesman, a tour of 200 given towns in Ireland that is less than 1800 miles (say). However, if the salesman claims that his current route through the 200 towns is 1720 miles, we can verify this claim very quickly – simply add the 200 distances between the towns on his current route.

We can now classify problems according to their *complexity* as follows:

Complexity Class	Finding Solution	Verifying Solution
Easy	Easy	Easy
Hard	Hard	Easy
Obviously Hard	Hard	Hard

We will discuss the first two classes of problems in the next two sections and illustrate their easiness or hardness. We do not discuss the *obviously hard* class because no fast method of solving these problems is possible.

Easy Problems

An easy problem is one whose solution requires a reasonable amount of computation time to find and verify. The problems to be discussed below will all be characterized by a single number n, called the size of the problem. We will express the complexity (computational difficulty) of each problem as a function of its size, n. We saw an example of an easy problem earlier: finding the maximum of n numbers requires a time that is linearly proportional to n; verifying the answer is also easy – compare the answer (a single number) with the remaining n-1 numbers.

The Minimum Spanning Tree Problem

This problem arises when we wish to connect, in some way, n objects so that the cost of connection is a minimum. For example, a cable television company wishes to service n customers from a central antenna so that the amount of cable used is a minimum. We can visualize this problem by plotting the locations of each customer and the central antenna on a sheet of paper. This is shown in Figure 2 (a). Figures 2 (b), (c) and (d) show three possible ways to connect 9 customers to the central antenna. It can be seen that the connection patterns have a tree-like structure – hence the name. One possible method of finding the least cost connection of customers is to list all possible connections and then pick the minimum. This method is not practicable because there are $(n+1)^{n-1}$ different ways of connecting n customers. In our example this would mean finding the minimum of $10^8 = 100$ million possible connections. Fortunately there is a very simple algorithm for finding the minimum cost connection.

Finding the Minimum Spanning Tree

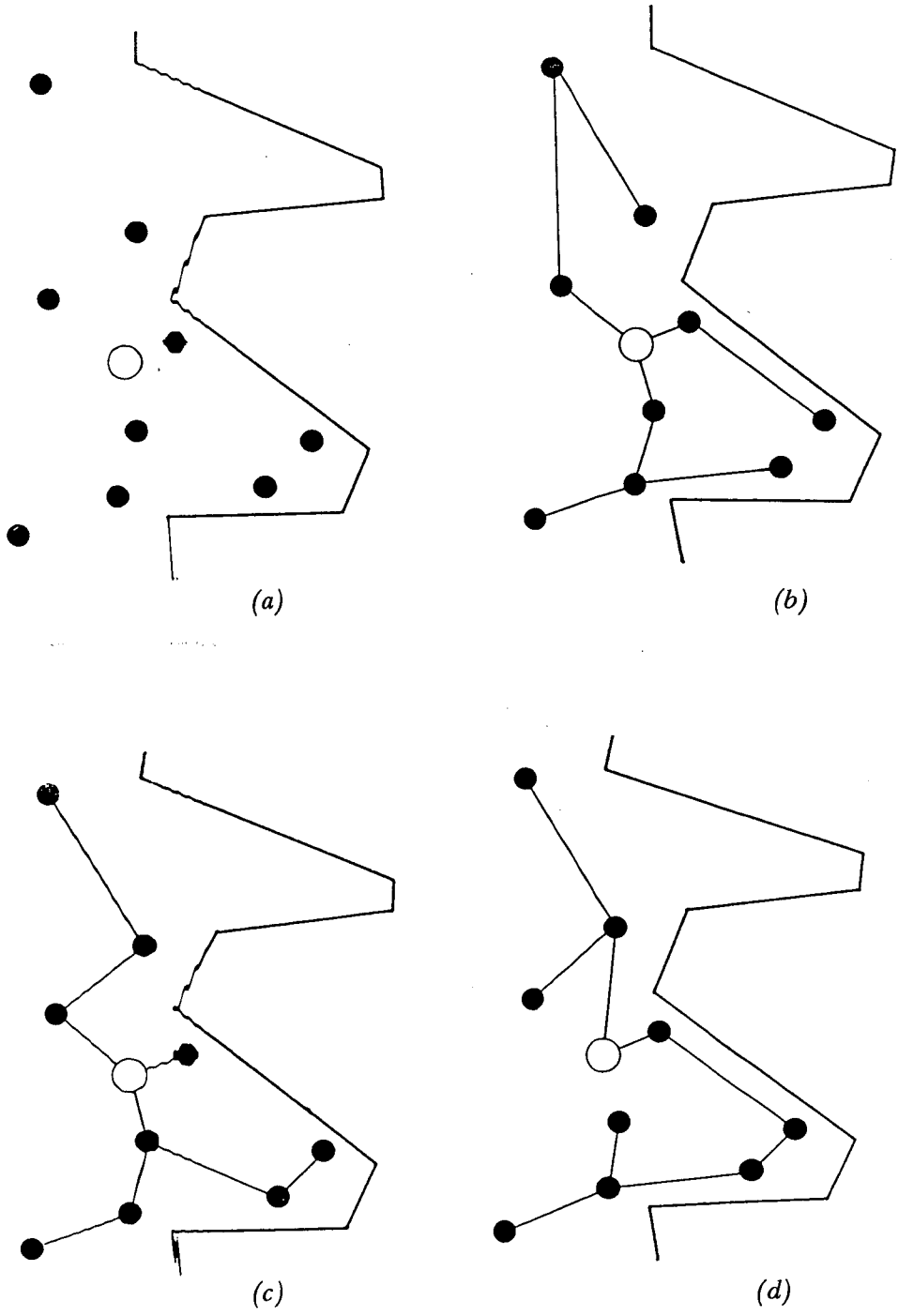
Step 1: *Connect the central antenna and the nearest customer.*

Step 2: *Connect the unconnected customer who is nearest to a connected customer or the central antenna, whichever is nearer.*

Repeat Step 2 until all customers are connected.

The remarkable thing about this algorithm is that it works at all. At each step it chooses the best (nearest) connection of the next customer without worrying about the overall solution. Despite this 'myopic'

Figure 2: *Connecting Cable TV Customers*

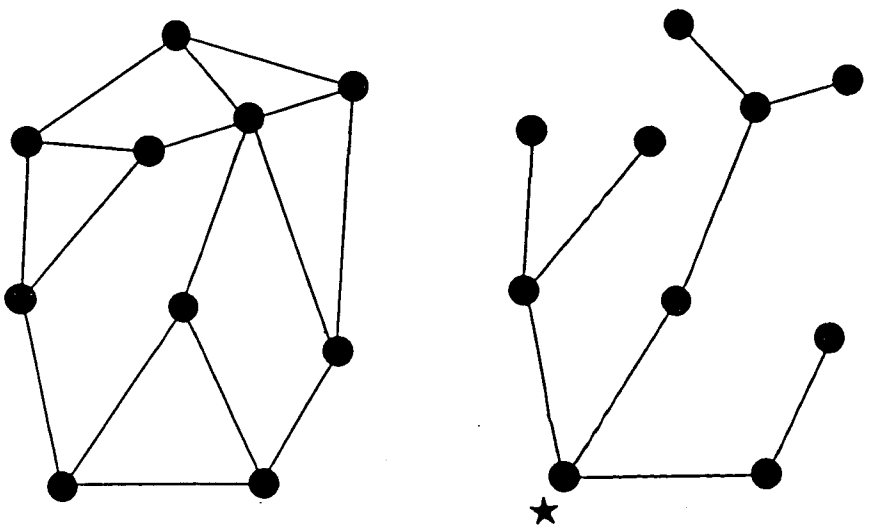


behaviour the algorithm gives the optimum connection. The algorithm is extremely fast and optimum connections for 5000 to 1000 customers can be found in a few seconds on an average mainframe computer. The computational time is proportional to n^2. Thus we say that the minimum spanning tree problem is *easy*. This is fortunate because these trees arise in many, apparently unrelated, practical problems.

The Shortest Path Problem

This problem arises when we are given a road map of towns connected by roads and are asked to find the shortest route (path) between two given towns. In abstract terms a road map is a *network* of n *nodes* (towns) connected by *arcs* (roads) and each arc has a *weight* (distance, time, cost, etc.) associated with it. A typical network is shown in Figure 3 (a). The shortest routes from one town to all others are shown in Figure 3 (b). There are many methods for finding the shortest path between two towns, two of the most popular being due to Bellman (1958) and Dijkstra (1959). Both of these methods are fast and can solve problems with n = 1000 towns in a few minutes on a main-frame computer. The computation time for the shortest path problem is proportional to n^2. We will not go into the details of these algorithms because they require some familiarity with graph theory and computer science. However, both methods are simple and can be programmed easily on a computer.

Figure 3: *(a): Road Network (b): All Shortest Paths from One Town*



A special case of the shortest path problem is the PERT problem which requires that we find the *longest* path through a PERT (project) network. The longest path contains the 'critical' activities of the project

and hence the length of the longest path is the length of the project [Wagner, 1975]. In general, finding the longest path through a network is hard, but PERT networks are special and can be solved easily using a slight modification of any good shortest path method.

It is interesting to note that most people 'solve' a shortest path problem when finding the best route to work. Initially, a route is constructed from memory or with the aid of a map. The initial route is then modified as experience of road and traffic conditions is gained, until a final route is chosen. We can only speculate on whether the final route chosen is the best in most cases — it probably is close to the best. Note, however, that two people travelling between the same two points will often chose different routes — one based on shortest distance and the other on shortest time.

The Simultaneous Linear Equation Problem

This problem occurs in many areas besides operations research and is fundamental to numerical calculations in economics, statistics, engineering and science in general.

Consider the following three linear equations in three unknowns x_1, x_2, x_3.

$$\begin{aligned} x_1 + 3x_2 - 4x_3 &= 8 \quad \text{(i)} \\ x_1 + x_2 - 2x_3 &= 2 \quad \text{(ii)} \\ -x_1 - 2x_2 + 5x_3 &= -1 \quad \text{(iii)} \end{aligned}$$

We eliminate x_1 from (ii) & (iii) by subtracting (i) from (ii) and adding (i) to (iii). This gives

$$\begin{aligned} x_1 + 3x_2 - 4x_3 &= 8 \quad \text{(i)} \\ -2x_2 + 2x_3 &= -6 \quad \text{(ii)'} \\ x_2 + x_3 &= 7 \quad \text{(iii)'} \end{aligned}$$

Eliminating x_2 from the last equation gives

$$\begin{aligned} x_1 + 3x_2 - 4x_3 &= 8 \quad \text{(i)} \\ -2x_2 + 2x_3 &= -6 \quad \text{(ii)'} \\ 2x_3 &= 4 \quad \text{(iii)''} \end{aligned}$$

This immediately gives $x_3 = 2$. Substituting this value back into equation (ii)' gives $x_2 = 5$, and substituting x_2 and x_3 back into (i) gives $x_1 = 1$.

This method can be generalized easily to n equation in a n unknowns. It has two phases: *elimination* of variables and *back substitution* of values. The method is known as *Gaussian Elimination* because the first systematic study of it was done by the famous mathematician C.F. Gauss (1777-1855).

The linear equations problem is classified as easy because problems ranging in size from 100 to 100,000 variables can be solved in times ranging from 1 min. to 1 hr. on a main-frame computer. The computation time for this problem is proportional to n^3.

The Linear Programming Problem

This is, without doubt, the most famous of all operations research problems and has been studied and solved continually since 1947, when Dantzig announced his simplex method. We illustrate the problem with an example taken from Dantzig's (1963) book.

A furniture manufacturer makes 4 types of furniture: cabinets, chairs, desks, and tables. Each piece of furniture is first constructed in a carpentry shop and it is then sent to a finishing shop, where it is varnished, waxed, and polished. The number of hours of labour required and the profit for each piece are as follows:

	Cabinet	*Chair*	*Desk*	*Table*
Carpentry	*4*	*9*	*7*	*10*
Finishing	*1*	*1*	*3*	*40*
Profit	*$12*	*$20*	*$18*	*$40*

Because of limited capacity, only 6000 hrs. can be performed in the carpentry shop and only 4000 hrs. in the finishing shop, per month. Assuming adequate raw materials are available and that all furniture produced can be sold, determine the optimum product mix, i.e., the quantities of each type to make each month to maximize profit.

Letting x_1, x_2, x_3, x_4 denote the quantities of cabinets, chairs, desks, and tables we get the following linear programming problem:

$$\text{Maximize Profit} = 12x_1 + 20x_2 + 18x_3 + 40x_4$$

$$\text{Subject to} \quad 4x_1 + 9x_2 + 7x_3 + 10x_4 \leq 6000 \quad \text{Carpentry Constraint}$$

$$x_1 + x_2 + 3x_3 + 40x_4 \leq 4000 \quad \text{Finishing Constraint}$$

There are many possible ways of solving this problem but the simplex method is by far the best. The method uses the following *Optimum Mix Characteristic*:

> *If n products can be produced using m resources then the optimum mix will contain, at most, m products.* ($n \geq m$)

Hence, in the example above, at most two of the four furniture types will be in the optimum mix. In fact the optimum solution is $x_1 = 1334$ and $x_4 = 67$.

The simplex method starts at a sub-optimal, feasible solution (e.g., $x_1 = x_2 = x_3 = x_4 = 0$) and improves the solution in a step-wise manner until no further improvement can be attained. Because of the optimum mix characteristic the number of steps to the optimum is finite and so the method is guaranteed to work. The main (theoretical) drawback of the simplex method is that it does not take a reasonable amount of computation time for *all* linear programming problems. However, a new method due to the Russian mathematician Khachian (1979) is guaranteed to solve *all* linear programming problems in a reasonable amount of time. Hence the linear programming problem is classified as easy. We must strongly emphasise that the superiority of Khachian's method over the simplex method is theoretical. In practice the simplex method is far superior to all other known methods. This is the reason for its success and indeed today it is used more often than any other computational tool. The size and computation time for problems solved by the simplex method are similar to those for the linear equations problem outlined above.

Conclusion

Easy problems are those that can be solved on a computer in a reasonable amount of time. We have not defined 'reasonable' because this varies widely over the class of easy problems. A great deal of research is still concentrated on finding better methods for easy problems. There are two reasons for this: (1) a better method can save scarce computer time and solve larger problems, and (2) it seems that once a problem is classified as easy then it is possible to find an optimal (fastest) method for its solution. The difference between an easy and a hard problem is rarely obvious. An apparently minor change in an easy problem can turn it into a hard problem. The classic example of this is the linear programming problem: if we require all variables in it to be integer-valued, then we get the integer linear programming problem, and this problem is hard. In the next section we discuss hard problems

and will see how subtle are the differences between easy and hard problems.

Hard Problems

We have defined a *hard* problem as one whose solution is difficult to find but easy to verify. This class of problems was first identified around 1970 and has been the subject of intense research – and a certain amount of controversy – since then. The main theoretical result of this research is surprising: if we can find a good method for solving *one* problem in this class then we can find good methods to solve *all* problems in this class. This result is very surprising because this class includes many problems, such as the travelling salesman problem, for which there are no *known* good methods, despite the efforts of the best researchers in the fields of computer science, mathematics and operations research. At the same time nobody has been able to prove that it is impossible to find good methods for solving these problems.

Proving the possibility or impossibility of finding good methods for this class of hard problems (the NP-Complete class) is the outstanding 'open' question in mathematics today. Answering this 'open' question is not merely a theoretical exercise: if it proves impossible to find good methods for these hard problems then we can stop looking for such methods and concentrate on *approximately* solving these problems rather than *exactly* solving them. We now illustrate and discuss three problems from this class that have practical importance.

The Travelling Salesman Problem (TSP).

This problem consists of many customers each of which must be visited by a single salesman, starting and ending at his head office. The solution is the tour, through all the customers, which minimises the total distance travelled by the salesman. Figure 4 shows a tour of 8 cities which must be visited by a salesman, who starts and finishes his tour at the head office. Figure 5 shows another possible solution to the problem.

Figure 4: *A Tour of 8 Cities*

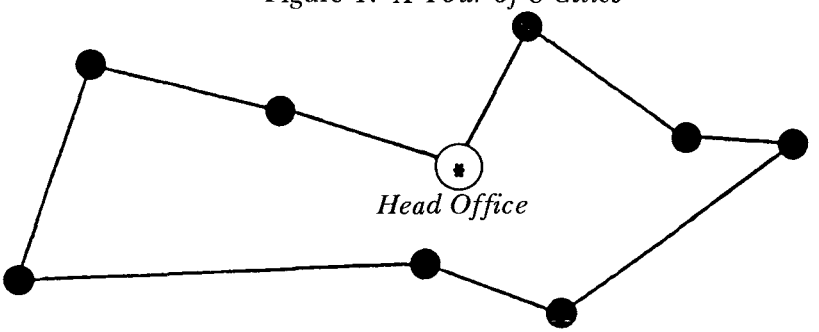
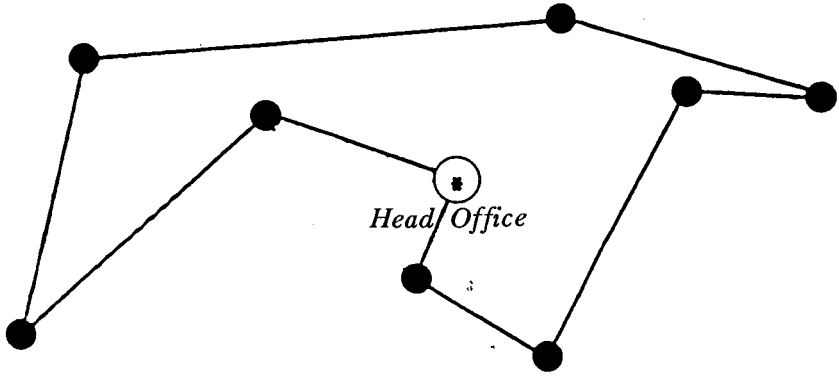


Figure 5: *A Different Tour of 8 Cities*



The TSP has attracted many man-years of research, partly because of its application in routing, but also because many different industrial problems can be transformed into a TSP. For example, the problem of scheduling different jobs on a single machine, where the setup time between jobs depends upon the order in which jobs are performed, can be formulated as a TSP. It is even possible to use the TSP formulation as a means of minimising the amount of wallpaper wasted when papering a wall with a patterned design.

The research into the TSP has resulted in several ingenious exact methods [see, for example, Little, Murty, Sweeney and Karel, 1969; Held and Karp, 1970; Miliotis, 1976; Crowder and Padberg, 1980]. However the TSP has been classified as a hard problem (NP-Complete) and thus the computer time required to solve exactly problems with more than 200 customers is prohibitive. In parallel with the development of exact methods for the TSP, much research has gone into the development of solution methods which, while not guaranteed to provide the exact answer, will provide good answers. These approximate methods use rules-of-thumb called *heuristics*. A fairly simple heuristic method for the travelling salesman problem is known as the 'closest unvisited city' method, in which the tour is constructed by always selecting the closest city not yet visited. This method is extremely fast, and good solutions to large problems (more than 2000 customers) can be obtained easily.

Since the early 1970s there has been an explosion in the number and complexity of heuristic methods for the TSP. Since these methods are not exact, i.e. they will not necessarily produce the optimal solution, it is necessary to test the "goodness" of the answers produced. The most widely used testing procedure is to apply the heuristic method

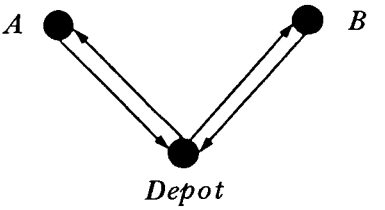
to a set of standard test problems. This empirical testing is not very useful because a good performance on the test problem does not guarantee that a similar good answer will be obtained if the method is used to solve a real-world problem. Researchers have recognised this deficiency of the testing procedure and new testing procedures are being developed to guarantee the quality of answers obtained.

Before leaving the TSP it must be stressed that the importance of the TSP is only partly due to the industrial occurence of the problem. It also has stimulated research and generated solution methods applicable to a much wider range of problems. This wider range includes the vehicle routing problem.

The Vehicle Routing Problem (VRP)

The VRP involves servicing many customers, with vehicles of limited capacity, from a central depot, in such a way as to minimise some operating criterion. Solutions to the problem of routing vehicles to service customers are obtained daily. Every time a truck leaves a distribution depot with goods to be delivered to or collected from a list of customers, then somebody has solved the VRP. Like the TSP, the VRP is a hard problem, and thus the main thrust of research is towards developing good approximate solution methods. The most successful and widely used of these has been the 'savings' method developed by Clarke and Wright (1964). The rule-of-thumb used in this method can be illustrated using just 2 customers, A and B, serviced from a depot. Assume, initially, that the delivery truck makes two separate trips, one to each customer. This is shown in Figure 6.

Figure 6: *Separate Deliveries*

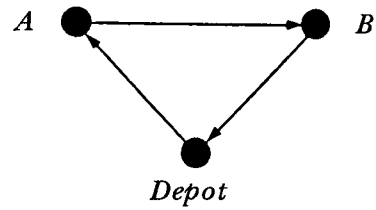


The total distance travelled in Figure 6.3 is:

$$\begin{aligned} Distance1 = \; & Depot\ to\ A + A\ to\ Depot \\ & + Depot\ to\ B + B\ to\ Depot \\ & = 2(Depot\ to\ A + Depot\ to\ B) \end{aligned}$$

Now, if we combine the separate trips into one trip as shown in Figure 7.

Figure 7: *Combined Deliveries*



The total distance now is:

$$Distance2 = (Depot\ to\ A) + (A\ to\ B) + (B\ to\ Depot)$$

The savings in distance resulting from combining the two trips is:

$$Savings = Distance1 - Distance2$$
$$= 2(Depot\ to\ A + Depot\ to\ B) - (Depot\ to\ A + A\ to\ B + B\ to\ Depot)$$
$$= (Depot\ to\ A) + (Depot\ to\ B) - (A\ to\ B)$$

This savings will be positive if (Depot to A + Depot to B) > (A to B). This rule-of-thumb can be generalized to problems with any number of customers.

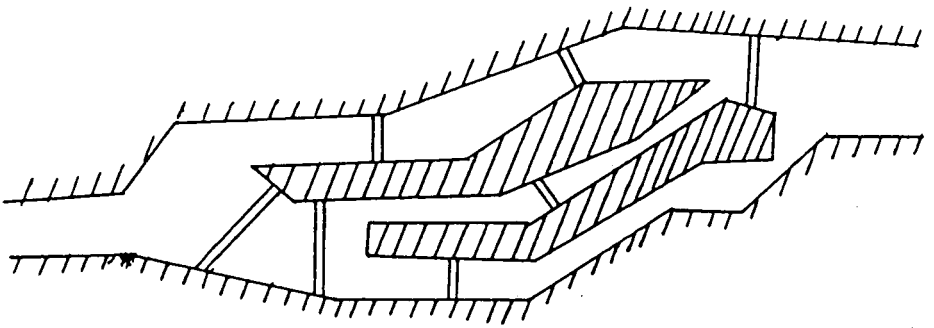
The 'savings' heuristic is incorporated into many of the commercial packages for vehicle routing. The question that must then be asked is: "How good are the routes produced by the heuristic?" To answer this question, and also to evaluate the multitude of other heuristics for the VRP, several test problems were developed. The most celebrated of these is the 100 customer problem of Christofides and Eilon (1969). The 'savings' solution services the 100 customers in 886 kms. The best known solution to this problem requires 826 km. The optimal solution to this problem is unknown. (The largest VRP solved to optimality contained only 40 customers). The above implies that for this simple test problem the 'savings' answer was at least 7% above the optimal. It must be stressed that this performance level cannot be guaranteed if the heuristic is applied to a practical problem.

In parallel with the development of approximate methods, research has gone into the development of exact methods for the VRP. The leading researchers in this area include Dr. N. Christofides and Prof. M. Fisher. Since VRP is a hard problem, this research is not intended to produce exact methods for large problems, but to gain further insight into the problem and incorporate this into new and effective approximate (heuristic) methods.

The Chinese Postman Problem

The Chinese Postman Problem, named after the Chinese researcher Mei-Ko Kwan (1962), is the problem of routing vehicles so as to traverse, at least once, all of the roads in an area. Such problems arise in, for example, refuse collection, milk or postal delivery, salting and gritting of roads to prevent ice formation, and inspection of electrical power networks. Several authors have studied this problem and many approximate and exact methods have been developed. These solution procedures use a technique that was developed by the mathematician Leonhard Euler in 1736. The problem studied by Euler concerned his ability to undertake a Sunday afternoon walk that crossed, only once, each of the 7 bridges, in his home town of Konigsberg, and then returned him to his original starting point. The town of Konigsberg is built on the bank of the river Pregel, with two islands linked to one another and the river bank by bridges, as shown in Figure 8. Euler proved that such a walk was impossible and some bridges must be crossed more than once. If Euler's problem was reformulated as to find the shortest walk that crosses each bridge at least once then we have the Modern Chinese Postman Problem.

Figure 8: *The Bridges of Konigsberg*



The Chinese Postman Problem, as stated above, can be solved exactly in a reasonable amount of time and is thus classified as easy [see Edmonds and Johnson, 1973]. However, minor variations in the problem definition change the computational complexity of the problem, i.e., move it from the easy to the hard class. If the road network consists of all two-way roads, or of all one-way roads then the problem is easy. However, if the network consists of a mixture of one-way and two-way roads then the problem is hard and thus the optimal solution cannot be found for large problems. Thus, we see that the distinction between hard and easy problems can be very subtle.

Conclusion and Summary

We have seen that well-defined problems in operations research can be classified as easy or hard, according to the amount of computation time they require for solution. This classification is important because it saves researchers and practitioners looking for an exact solution method for a hard problem when (probably) none exists. If the problem is easy then a fast – possibly optimal – solution method is available or can be developed. This is particularly important for operations research practitioners – it saves them 're-inventing the wheel'. As an aid to the practitioner we present a short check-list of easy and hard problems below. A more complete list is available in Garey and Johnson (1979) and Bodin, Golden, Assad and Ball (1981).

It may seem 'academic' to be worrying about computer solution time when computers are becoming faster, smaller and cheaper by the day. Such a view is erroneous: no technological breakthrough in digital computer *hardware* will shift a problem from a harder class to an easier class. Recalling the card permutation example, a computer 1000 times faster than the best today would still require millions of years to generate all permutations of the alphabet's letters (this is why chess-playing programmes that try to evaluate all possible moves will never beat a good human player). Another way of stating this is: no matter how fast computers become, we can always find well-defined mathematicial problems that cannot be solved in a reasonable amount of time on these computers.

SUMMARY OF EASY AND HARD PROBLEMS

Easy	*Hard*
1. Sorting Numbers	1. Knapsack Problem
2. Minimum Spanning Tree	2. Capacitated Spanning Tree
3. Shortest Path	3. Longest Path
4. Assignment Problem	4. Travelling Salesman Problem
5. Transportation Problem	5. Vehicle Routing Problem
6. Maximum Flow Problem	6. Network Reliability
7. Minimum Cost Flow Problem	7. Airline Crew Scheduling
8. Chinese Postman Problems	8. Mixed Chinese Postman Problem
9. Solving Linear Equations	9. Timetable Design
10. Linear Programming	10. Integer Programming

REFERENCES

Bellman, R.E., "On the Routing Problem", *Quarterly on Applied Mathematics*, Vol. 16, 87-90, 1958.

Bodin, L., Golden, B., Assad, A., and Ball, M., "The State of the Art in the Routing and Scheduling of Vehicles and Crews", University of Maryland, Working Paper No. 35, 1981.

Christofides, N. and Eilon, S., "An Algorithm for the Vehicle Dispatching Problem", *Operational Research Quarterly*, Vol. 12, 309-318, 1969.

Crowder, H. and Padberg, M., "Solving Large-Scale Travelling Symmetric Travelling Salesman Problems to Optimality", *Management Science*, Vol. 26, 495-509, 1980.

Clarke, G. and Wright, J., "Scheduling of Vehicles from a Central Depot to a Number of Delivery Points", *Operations Research*, Vol. 12, 568-581, 1964.

Dantzig, G.B., "Maximization of a Linear Function of Variables Subject to Linear Inequalities", Chap. 21, in Koopmans, T.C., (ed.), *Activity Analysis of Production and Allocation*, Cowles Commission Monograph No. 13, Wiley, 1951.

Dantzig, G.B., *Linear Programming and Extensions*, Princeton University Press, 1963.

Dijkstra, E.W., "A Note on Two Problems in Connexion with Graphs", *Numerische Math.* Vol. 1, 269-271, 1959.

Edmonds, J. and Johnson, E., "Matchings, Euler Tours and the Chinese Postman Problem", *Mathematical Programming*, Vol. 5, 88-124, 1973.

Grattan-Guinness, I., "Joseph Fourier's Anticipation of Linear Programming", *Operational Research Quarterly*, Vol. 13, 361-364, 1970.

Garey, M.R., and Johnson, D.S., *Computers and Intractability: A Guide to the Theory of NP Completeness*, W.H. Freeman, 1979.

Held, M., and Karp, R., "The Travelling Salesman Problem and Minimum Spanning Trees", *Operations Research*, Vol. 18, 1138-1162, 1970.

Khachian, L.G., "A Polynomial Algorithm for Linear Programming", *Soviet Mathematics Doklady*, Vol. 20, 191-194, 1979.

Kwan, Mei-Ko, "Graphic Programming Using Odd and Even Points", *Chinese Mathematics*, Vol. 1, 273-277, 1962.

Little, J.D.C., Murty, K.G., Sweeney, D.W., and Karel, C., "An Algorithm for the Travelling Salesman Problem", *Operations Research*, Vol. 11, 972-989, 1963.

Miliotis, P., "Integer Programming Approaches to the Travelling Salesman Problem", *Mathematical Programming*, Vol. 10, 367-378, 1976.

Wagner, H.M., *Principles of Operations Research, 2nd Ed.*, Prentice-Hall, 1975.