RALF BORNDÖRFER MARTIN GRÖTSCHEL ANDREAS LÖBEL

The Quickest Path to the Goal

To appear as Chapter 5 in Martin Aigner, Ehrhard Behrends (Editors) *All Mathematics*, American Mathematical Society, 2010.

ZIB-Report 10-21 (July 2010)

The Quickest Path to the Goal

Ralf Borndörfer* Martin Grötschel* Andreas Löbel*

July 11, 2010

Abstract. We provide an introduction into the mathematics of and with paths. Not on the shortest, but hopefully on an entertaining path!

Keywords. shortest paths, combinatorial optimization, operations research

Mathematics Subject Classification (1991). 90C10

1 Historical Overture

The theme "paths" evokes associations with streets, transport, traffic ... and mathematics! Here are four examples.

The Königsberg bridge problem. In the year 1736 the mathematician, astronomer and physicist Leonhard Euler (1707–1783) studied the simple and at the same time mysterious sketch reproduced as Figure 1.

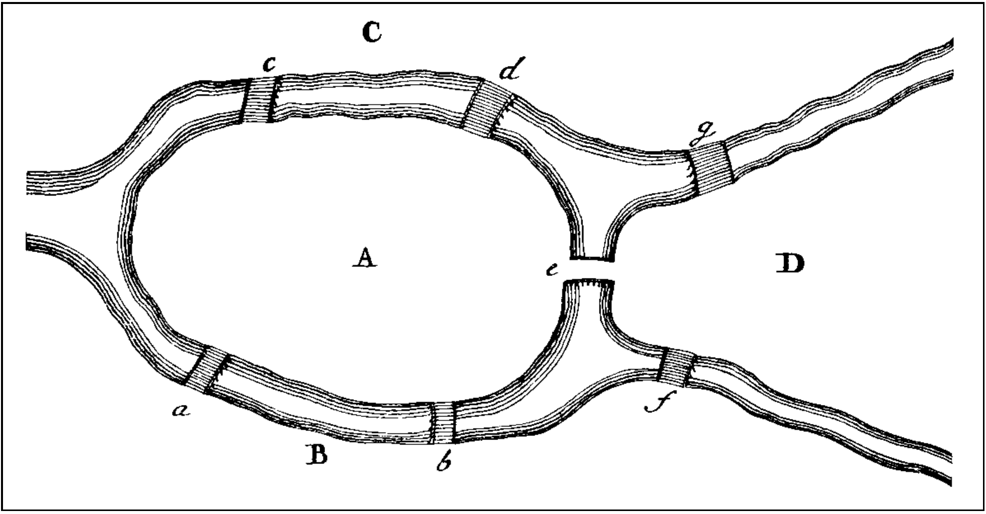


Fig. 1: The Königsberg bridge problem.

[1]Zuse-Institute Berlin, Takustraße 7, 14195 Berlin, Germany, http://www.zib.de

The problem, which ought to be familiar, was the following: At Königsberg in Pruſſia there is an iſland A, called „the Kneiphof", and the river that flows by it divides into two arms as indicated in Figure 1. Seven bridges, a, b, c, d, e, f and g, paſſ over the arms of the river. The queſtion is whether one can deviſe a round walk that paſſes over each of theſe bridges exactly once. I have heard that ſome deny this poſſibility, while others are doubtful, and that nobody corroborates this. From this I frame the following very general problem: How from the ſhape of the river and its diviſions into arms, and alſo the number of bridges, to determine whether it is poſſible to paſſ over each bridge exactly once, or not.

This was the *Königsberg Bridge Problem.* It was most unconventional: undoubtedly mathematical in nature, and yet so framed that it neither required the calculation of a number, nor admitted a ſolution with the help of numerical calculation. Euler called this new mathematics without numbers, in which only the structure of the configuration plays a rôle, but not size and form, *Geometria situs*, *geometry of position* (he took the concept from a letter of Leibniz from the year 1679).

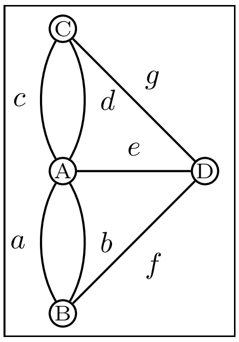


Fig. 2: Graph.

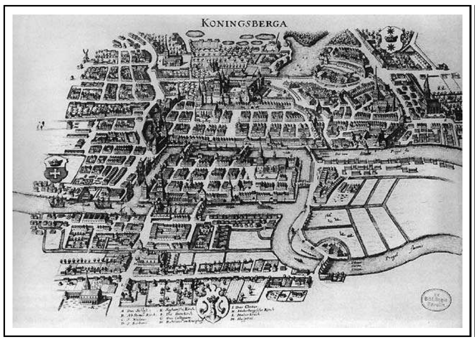


Fig. 3: Königsberg 1736.

Fig. 4: L. Euler.

Euler's first step into this uncharted territory was an abstraction of genius. From the map in Figure 1 he produced the diagram in Figure 2. In this *graph* the islands have been transformed into formless nodes and the bridges into lines (today we call them edges). When he studied this representation Euler noticed that the number of edges that lead into a node, its *degree*, plays an important rôle. This quantity was the key to the solution, not only of the Königsberg, but even all bridge problems, through two wonderful results:

Theorem 1. *The number of nodes with odd degree is even.*

Theorem 2. *There is a walk that traverses each edge exactly once if and only if there are at most two nodes of odd degree.*

To find such a walk one must identify a starting node and an end node, which may, but need not, coincide.

Euler's theorems were the first contributions to *topology*, as the geometry of position is now called: more precisely, to the mathematical discipline now called *graph theory.* The highest honor in mathematics is the naming of a discovery after a person. The term *Euler tour* for such a walk is one of these references through which mathematicians today still honor Euler's work. The question remains for you, dear readers:

Exercise 1 (Solution in Section 6)**.** *Is there an Euler path over the Königsberg bridges?*

The Hamiltonian circuit problem. In 1857 the Irish mathematician Sir William Rowan Hamilton (1805–1865) invented the "Icosian Game", see Figure 7, which resembles the Königsberg Bridge Problem at first glance. In the graph in Figure 5, whose nodes represent places such as Brussels, Canton, Delhi to Zanzibar, one is to find a *closed tour* (a cycle) that visits each place exactly once. Just try it! Try the "honeycomb" in Figure 6 beside it too: this was invented at about the same time by Hamilton's English colleague Thomas Penyngton Kirkman (1806–1895). You will find a difference!

Exercise 2 (Solution in Section 6)**.** *Is there a closed tour in the Icosian Game?*

Exercise 3 (Solution in Section 6)**.** *Is there a closed tour in the honeycomb?*

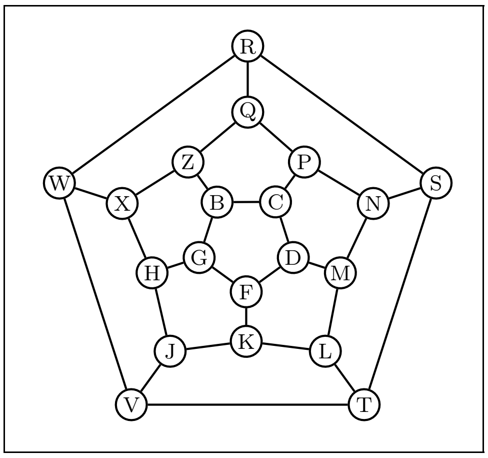


Fig. 5: Graph of the Icosian Game.

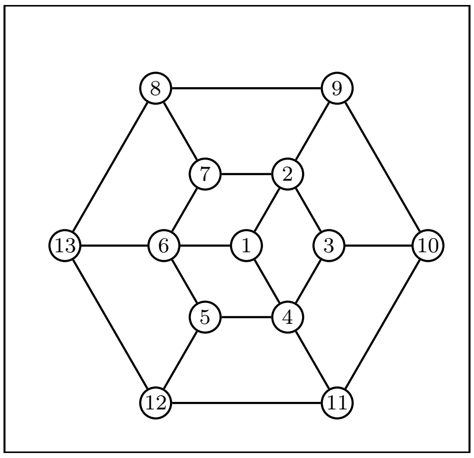


Fig. 6: The "honeycomb".

Closed tours in graphs are called *Hamiltonian circuits*, and the question whether a graph contains such a path is the *Hamiltonian Circuit Problem* (HCP for short). What is astonishing about the HCP is that although the problem resembles the Bridge Problem closely it admits no similar method of solution. All known methods must reckon in the worst case with having to try all possibilities by *enumeration*. The amazing reason for this is that in all probability there is nothing better! *Complexity theory* shows that the HCP belongs to the class of *NP-complete problems*, which are difficult in a way that can be made mathematically precise. However, a conclusive proof of the inevitability of enumeration for NP-complete problems has not been found yet, despite intensive research.

For special graphs one can do more, and develop specific algorithms. Hamilton knew a method of solution for his Icosian game, and he could even prescribe the

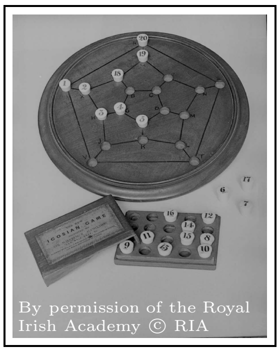


Fig. 7: Icosian Game.

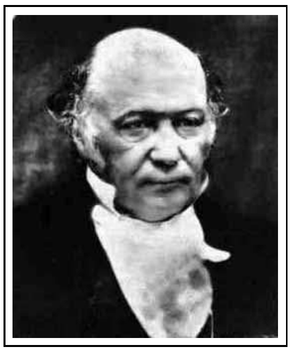


Fig. 8: W.R. Hamilton.

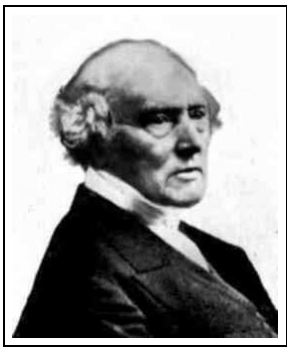


Fig. 9: T.P. Kirkman.

starting cities. His method was based on an "Icosian Algebra" that he invented, which had to do with the symmetry properties of the icosahedron. Hamilton sold his game for £25 to a games company in 1859, which marketed it under the name "Around the World". It was a shelf warmer, and no wonder. Just the beginning of the instructions, where Hamilton explains why the *Icosian* game takes place on a *dodecahedron* (see Figure 10), can take the fun away. Hamilton knew something of mathematics, but clearly nothing of marketing!

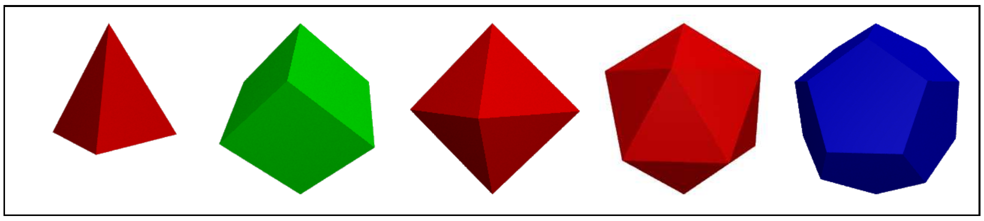


Fig. 10: The Platonic solids: tetra-, hexa-, octa-, icosa-, dodecahedron.

The traveling salesman problem. An optimization variant of the HCP is the *Traveling Salesman Problem* (TSP for short). One seeks the shortest closed tour in a *complete graph* (with all imaginable edges) whose edges have *lengths*. This is relevant for commercial travelers and automatic drilling machines, but the greatest importance of the TSP is as *the benchmark problem* in combinatorial optimization. In contrast to the HCP the difficulty in the TSP does not lie in finding a tour: In a complete graph with n nodes one may choose from $\frac{1}{2}(n-1)!$ tours. Figure 11 indicates the true problem. One can transform each HCP (here the honeycomb) into a TSP: A tour of length 0 exists if and only if the initial HCP admits a Hamiltonian circuit. This *problem transformation* shows that the TSP is *$\mathcal{NP}$-hard.* ($\mathcal{NP}$-complete optimization problems are called $\mathcal{NP}$-hard.)

All the same, $\mathcal{NP}$-hard does not mean that trivial enumeration is the only possibility for solution. One idea is to combine enumeration with empirically efficient techniques known from experience to run in tolerable time for problems up to a certain size. *Branch & Bound*, *Branch & Cut* and *Branch & Price*

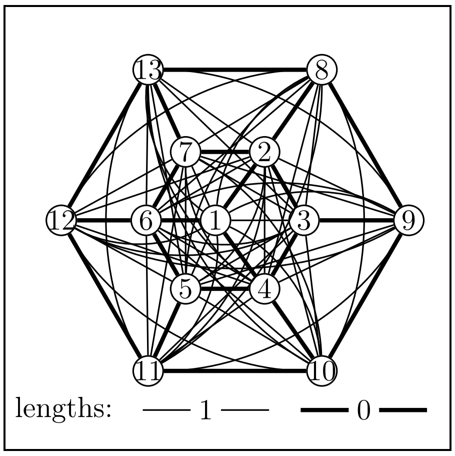


Fig. 11: The HCP as TSP.

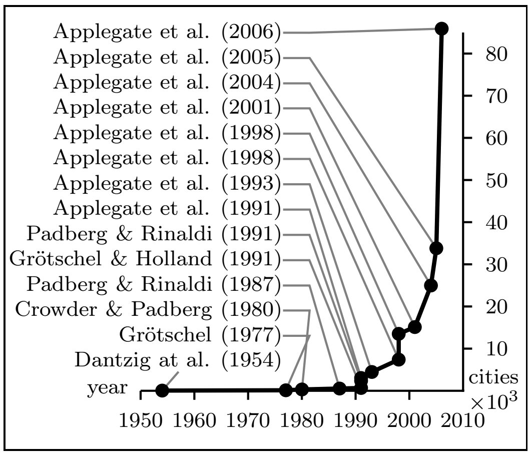


Fig. 12: TSP world records.

are the main representatives of this type of algorithm. “Branch” stands for enumeration, “Bound”, “Cut” and “Price” for various acceleration techniques. Just tuned enumeration? From a complexity theory point of view, yes, but enormously effective in the *concrete case*. Figure 12 shows the developments with TSP. In 1991 the 666-city tour in Figure 13 was the limit of possibility. Today one can solve this problem in seconds, and the world record, set in 2006 by the team-of-seven, David Applegate, Bob Bixby, Bill Cook, Vasěk Chvátal, Daniel Espinoza, Marcos Goycoolea and Keld Helsgaun, stands at 85,900 cities! As against the 666 city problem there are $129\times$ more cities, $16,660\times$ more edges, and $7.4 \cdot 10^{348,931}\times$ more tours! Presently the same group is attempting to find the shortest tour through all the 1,904,711 inhabited places of the world. The length of the best tour found to date measures 7,516,043,366 m, and one knows that the tour cannot be shorter than 7,511,705,615 m: that is, one can improve this result by at most 0.05%. The algorithmic progress is naturally difficult to quantify. But certainly one cannot dispute that *Branch & Co* is more than trivial enumeration.

An alternative is to accept “good” solutions, ones that differ from the (unknown!) optimum by not more than a given percentage, in place of an exact solution. But are there fast *approximation algorithms* with such *performance guarantees* for $\mathcal{NP}$-hard problems? Often yes! For *euclidean TSPs* (with beeline distances, still an $\mathcal{NP}$-hard variant) the Indian mathematician Sanjeev Arora discovered such a method in 1996, where the goodness can be stipulated in advance. One can show that there can be no such method for TSPs in general.

The shortest path problem. The oldest path problem known to us arises from a classical source: Friedrich Schiller’s (1759–1805) play “Wilhelm Tell”. For already by 1291 he could not just shoot well, but he could also optimize. And only with this combination was he able to free Switzerland! After the

Fig. 13: The shortest journey round the world.

apple shot Tell found himself at the shore of Lake Lucerne, not far from Altdorf. At all costs he had to reach the Hohle Gasse in Küssnacht before the Bailiff Hermann Gessler: see Figure 15. Schiller reports[1]:

Tell.	Which is the neareſt way to Arth and Küſſnacht?
Fischer.	The open route's by Steinen. But my boy
	Can bring you by a quicker leſſ-known way
	Acroſſ Lowerz.

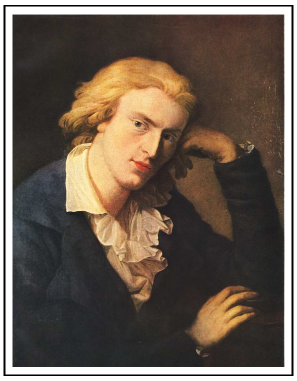


Fig. 14: F. Schiller.

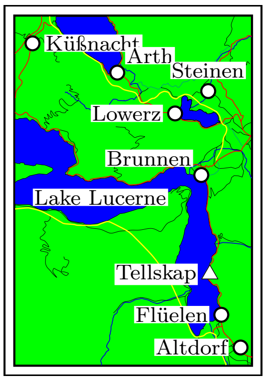


Fig. 15: Lake Lucerne.

Fig. 16: W. Tell.

In this scene Tell solves a graph theoretic optimization problem. The shortest path between two predetermined points (Altdorf and Küssnacht) in a graph

[1]Translation taken from version of John Prudhoe, Manchester University Press 1970.

(the road system at the Vierwaldstätter See) with edge lengths (travel time) is to be found. This is a *Shortest Path Problem* (Single Source Shortest Path [problem]: SSSP for short, the second P being dropped). Tell has to deal with a complicated variant with an extra *constraint*: The sum of the "arrest coefficients" has to be kept below a safety margin. Literature and mathematics – no way in contradiction!

*

Path problems arise everywhere in connection with networks: In route and personnel planning, in logistics and project management, in the design of integrated circuits, in the design of telecommunication networks, in routing telephone calls and data, etc. Mathematics can help to lower costs, raise quality, and plan quicker.

In this article we intend to illuminate several questions, methods and possibilities in the mathematical treatment of path problems. The examples of passenger information in public transport and the planning of bus services form the basis of our presentation. The prelude is the fundamental shortest path problem and its treatment. For the real time planning of bus driver duties the heavy Branch & Price cannon must be brought up. Shortest paths play an important rôle there too. At the end there is a survey of the field, hints for future reading, and the solutions to the problems posed in the text. Enjoy your path through the mathematics of paths!

2 Combinatorics of Shortest Paths

2.1 Local Transport and Graph Theory

A shortest path problem familiar to everyone is choosing a route in public transport. Figure 17 shows the example of the Berlin rapid transit network (subways and commuter trains). 306 stations and 445 legs do not make the decision easy. What should one optimize: time, legs, transfers, price? Let us consider a model.

Definition 3 (Shortest path problem (SSSP))**.**
Given: *Graph $G = (V, E)$ (node set V, edge set E)*
nonnegative lengths or weights w_{uv} for all edges $uv \in E$
two nodes s and t
Sought: *a shortest path from s to t in G*

The advantage of this *abstraction* is that the problem data may be interpreted as needed. The edge lengths may be prices, legs, times, etc.: the mathematics is the same. The versatility of the model goes even much further. By clever variation of the lengths and manipulation of the structure one can reduce many

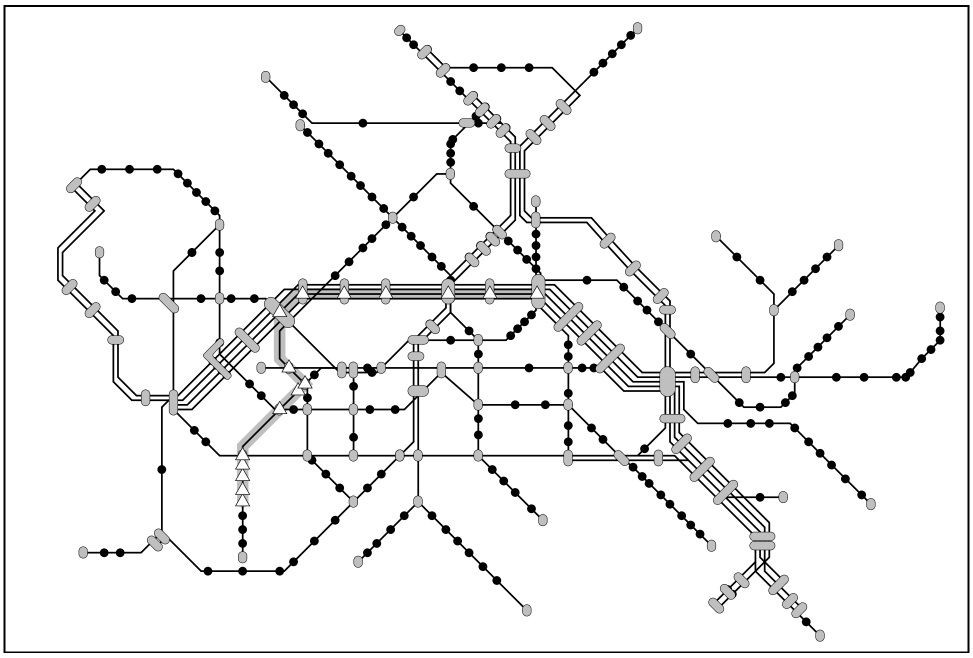


Fig. 17: The rapid transit network of Berlin.

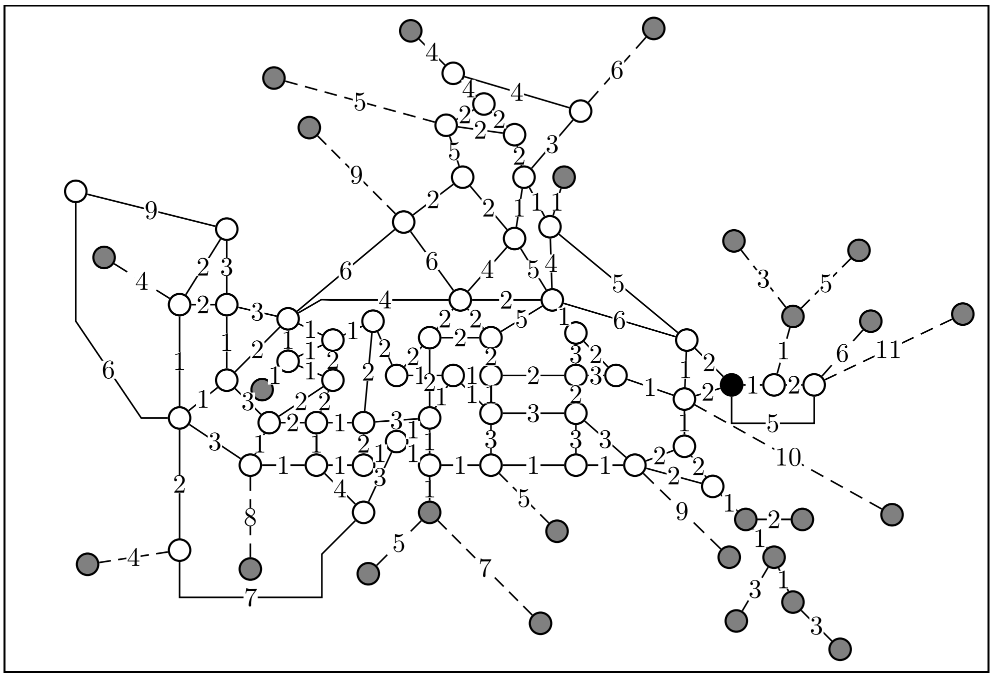


Fig. 18: Reduced rapid transit network.

variant problems to the basic model. We give three examples of such *modeling tricks*.

Fare minimization in a zone system. Many public transport companies have tariffs under which a supplementary fare is due for each new zone. Figure 19 shows an edge weighting for constant supplements (always the same). More realistic, however, are decreasing zone supplements; we will investigate this case in Section 2.3.

Node weights. One can "transfer" weights (costs, prices) at the nodes to the edges. The formulae for this are indicated in Figure 20 (with special treatment for start and goal).

Transfers. Figure 21 shows a treatment of transfer times at intermediate stations as an example of a structural transformation.

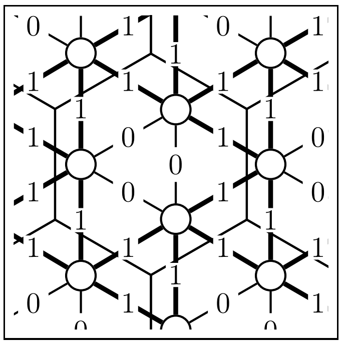
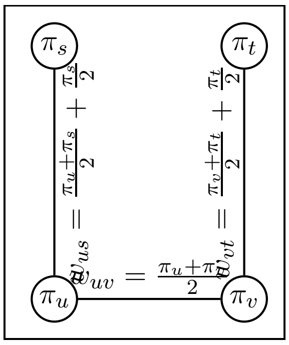


Fig. 19: Zone tariffs.


π_s
π_t
$w_{us} = \frac{\pi_u+\pi_s}{2} + \frac{\pi_s}{2}$
$w_{vt} = \frac{\pi_v+\pi_t}{2} + \frac{\pi_t}{2}$
$w_{uv} = \frac{\pi_u+\pi_v}{2}$
π_u
π_v
Fig. 20: Node weights.

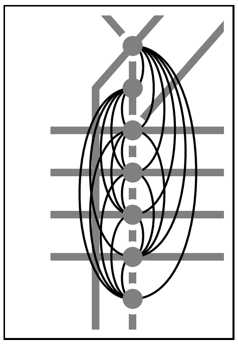


Fig. 21: Transfers.

Enthusiastic as one may be for such techniques, nevertheless: In modeling less is often more. It is neither sensible nor necessary to underline every inessential detail. Often the data available will set narrower limits than the power of the algorithms!

We shall remain with the 'Simple', and "complete" the rapid transit example, in which we assign the length 1 to each edge in Figure 17. The length of a path is then the number of edges traveled. The path marked by in Figure 17 from Alexanderplatz to Dahlem-Dorf (headquarters of the Zuse Institute), for example, has a length of 15 (edges).

One can simplify such path length computations greatly at little cost. Figure 18 shows how amalgamating a sequence of edges with no transfer possibilities into an edge with the corresponding length (a new edge of length k replaces k old edges) one achieves a considerable reduction in the size of the problem: to 80 nodes and 122 edges. The price for this *preprocessing* is that now one can compute shortest paths directly only between the end and interchange nodes of the lines. But one can then derive all shortest paths easily by considering cases. (The path from and to a node between two transfer-/ end nodes A and

B passes either through A or through B. The same holds for the other end with nodes C and D. So there are four cases: AC, AD, BC and BD.)

The reduction can be taken further. One can eliminate the gray/dashed *trees* in Figure 18, where one has no choice. One can decompose the graph at the black *articulation nodes* and so on. At some point overkill sets in, when the cost of computation and implementation outweighs the usefulness. Even so: The right dose of reprocessing is a must if one is to solve practical optimization problems.

2.2 On the Tracks of Chance

The simplest approach to determining a shortest path is enumeration. But which transport company would advise its passengers to do so? The number of paths is too large. How large? Let us find out. Unfortunately there is no formula for the exact number of paths in an arbitrary graph. This question is too complex. Astonishingly, however, there is a formula for the "approximate number of paths in an average graph with n nodes".

A simple method of generating an "average" graph by coin tossing is shown in Figure 22: For each of the $(n^2 - n)/2$ possible edges one flips a Euro coin. 'Number' accepts the edge, 'Eagle' rejects. We are tossing 'German Euros' with eagles.) This provides a *realization* of the *random graph* G_n on n nodes. Each realization may contain different edges, yet on the average one knows a lot about this graph. For example, G_n has an *expected number* of $(n^2 - n)/4$ edges, a half of those possible, since each exists with probability $1/2$.

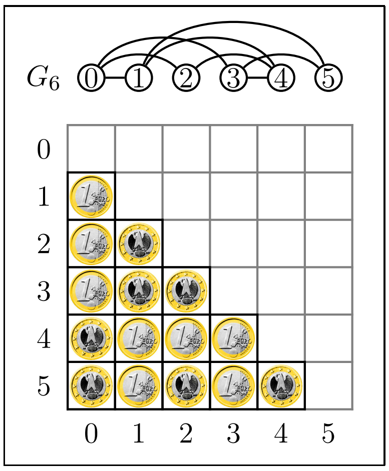


Fig. 22: A random graph.

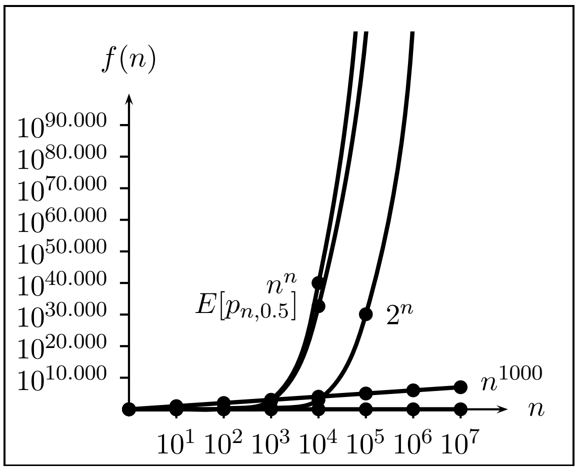


Fig. 23: Combinatorial explosion.

Similarly one can compute the *expected number of paths* in G_n between two randomly chosen nodes s and t. First let us consider only the longest possible (s, t)-paths, the Hamiltonian paths. These have $n-1$ edges. There are $(n-2)!$

such paths, one for each configuration of the $n-2$ "inner" nodes. But not all these paths exist in each realization of the random graph G_n. The first edge exists in 1/2 of the cases, the first and the second in 1/4 of the cases, etc. All $n-1$ edges exist in $1/2^{n-1}$ of the cases. On average one can expect

$$\begin{aligned}\frac{(n-2)!}{2^{n-1}} &= \frac{n!}{2^{n-1}} \cdot \frac{1}{n(n-1)} \approx \frac{n^n e^{-n}}{2^{n-1}} \cdot \frac{\sqrt{2\pi n}}{n(n-1)} \\ &= \left(\frac{n}{2e}\right)^{n-2} \cdot \frac{n^2\sqrt{2\pi n}}{2n(n-1)e^2} \geq \left(\frac{n}{2e}\right)^{n-2} \cdot \frac{\sqrt{n}}{e^2}\end{aligned}$$

(s,t)-paths of length $n-1$. From *Stirling's formula*

$$n! \approx n^n e^{-n}\sqrt{2\pi n}$$

the relative error of this approximation is smaller than 1% for $n \geq 10$. With a little more algebra one can estimate the expected number $E[p_{n,\rho}]$ of all (s,t)-paths in G_n (with 1 to $n-1$ edges) as follows:

$$E[p_{n,\rho}] \approx \left(\frac{n\rho}{e}\right)^{n-2} \cdot e^{1/\rho-2} \cdot \rho \cdot \sqrt{2\pi n}$$

where ρ is the probability of the occurrence of an edge ($\rho = 1/2$ for coin tossing). All these numbers have an *order of magnitude* of more than $(cn)^{n-2}$ (c a constant). Figure 23 shows the enormous growth of such functions on a doubly logarithmic scale. Already in our little rapid transit example with only 336 nodes and edge probability $\rho = 445/\frac{336^2-336}{2} = 0.008$ there are no fewer than $E[p_{336,0.008}] = 1.977 \cdot 10^{50}$ paths to expect! This growth phenomenon is known as the *combinatorial explosion.* There is no escaping it. The way out is to search through the giant *solution space* purposively. Euler had already recognized this:

In regard to the Königsberg problem with the ſeven bridges, one could ſolve by an exact enumeration of all poſſible paths; one would then know whether one ſatiſfies the condition, or none. This mode of ſolution is, however, becauſe of the large number of combinations, too laborious and difficult; and moreover, could no longer be applied to other queſtions with very many bridges. If one were to examine it in the way alluded to above one would find much not in queſtion; this is doubtleſſ the reaſon why this route would be ſo burdenſome. This is why I have rejected this method and looked for another, to reach ſo far as to eſtabliſh whether ſuch a walk can be found, or not; for I conjectured that ſuch a method would be much ſimpler.

2.3 Münchhausen versus Archimedes

Euler's advice, not to undertake superfluous computations, may well be applied to the shortest path problem. Clearly many paths are not the shortest. The shortest path from Alexanderplatz to Dahlem-Dorf does not pass through the dark gray articulation node "Lichtenberg" in Figure 18, for the path Lichtberg-Dahlem Dorf alone is already longer than 15 edges – or is it? Now we require two shortest paths and the continuation of this argument is strongly reminiscent of the method invented by the Freiherr Karl Friedrich Hieronymus von Münchhausen (1720–1796) to pull himself out of the swamp by his own hair.

Fig. 24: Münchhausen's method.

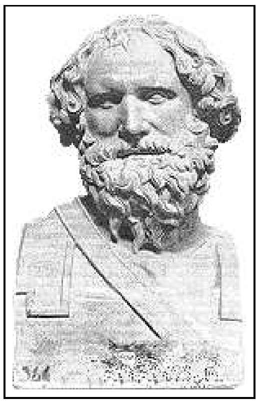


Fig. 25: Archimedes of Syracuse.

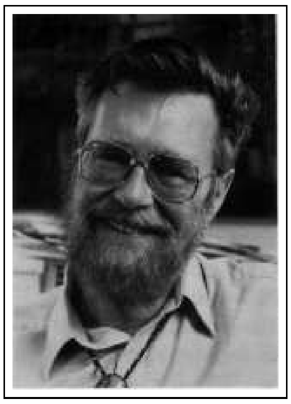


Fig. 26: E.W. Dijkstra.

"Unfortunately" the liar-baron had invented the story only to entertain his guests. We have to establish a firm foundation for a genuine shortest path to emerge from the swamp. Here is one: The "empty path" of length 0 from Alexanderplatz to itself. Do you find this somewhat sparse? Archimedes of Syracuse (287–212) had another opinion! "Give me a place to stand and I will move the Earth" said he. Might Archimedes's point be preferable to Münchhausen's method?

The "flip book" Figure 27–32 shows how from an Archimedean point one can makes the whole graph traversable by shortest paths. The method is based on the concept of a *distance mark* (distance label) which for each (reached) node states the length of the shortest path so far discovered. The paths themselves form a *shortest path tree.*

Initialization. The node "Alexanderplatz" is *temporarily marked* with distance 0.

Figure 27. The node Alexanderplatz for the moment has the smallest temporary distance mark and is *selected.* The node cannot be reached by a shorter path. The temporary distance mark for Alexanderplatz is therefore made *permanent.*

Figure 28. The nodes neighboring Alexanderplatz are *marked* with the distances to them, and the edges are entered into the *shortest path tree.* The marking and tree are *temporary.* (There may be shorter paths that do not come directly from Alexanderplatz.)

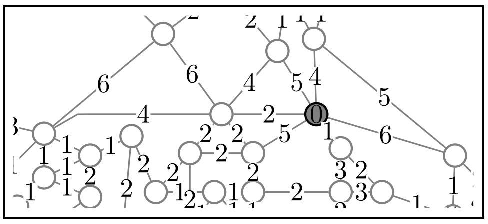


Fig. 27: Dijkstra's algorithm (0).

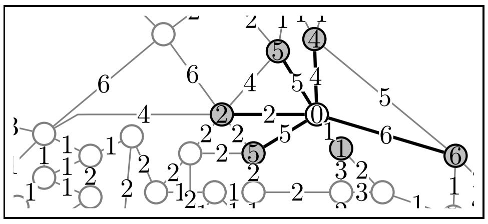


Fig. 28: Dijkstra's algorithm (1).

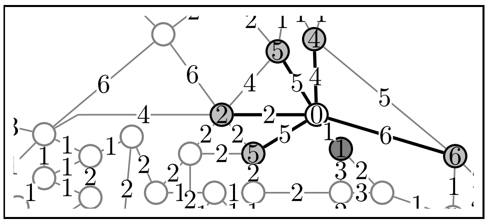


Fig. 29: Dijkstra's algorithm (2).

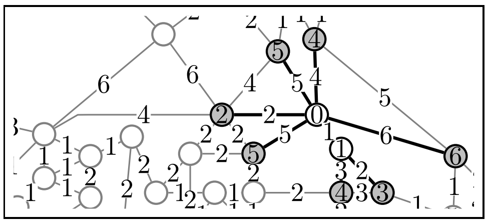


Fig. 30: Dijkstra's algorithm (3).

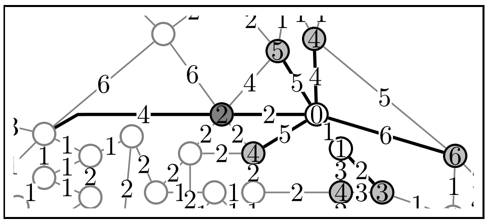


Fig. 31: Dijkstra's algorithm (4).

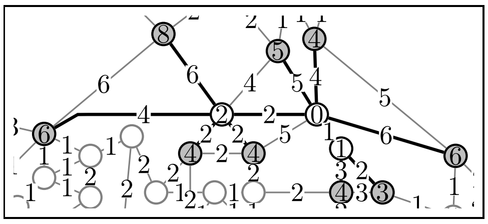


Fig. 32: Dijkstra's algorithm (5).

Figure 29. The node "Jannowitzbrücke" with the distance mark 1 cannot be reached by a shorter path. This pivotal observation provides the second fixed Archimedean point. The node Jannowitzbrücke is *selected*, the marking and edge are made *permanent* in the shortest path tree.

Figure 30. The temporary distance marks of the neighbors of the node Jannowitzbrücke are *refreshed* (label update). Two new nodes are discovered and *temporarily marked.*

Figure 31. As in step 0 the node "Friedrichstraße" is *selected* with the presently smallest temporary label 2. Label and shortest path edge are made *permanent.*

Figure 32. On the distance-update of the neighbor nodes three new nodes are *temporarily marked.* The distance to a temporarily marked node decreases from 5 to 4. Marks and shortest path tree are correspondingly refreshed.

. . .

End. The computation ends when the goal node, here the (black) "Heidelberger Platz", is marked permanent (or when all nodes are marked permanent).

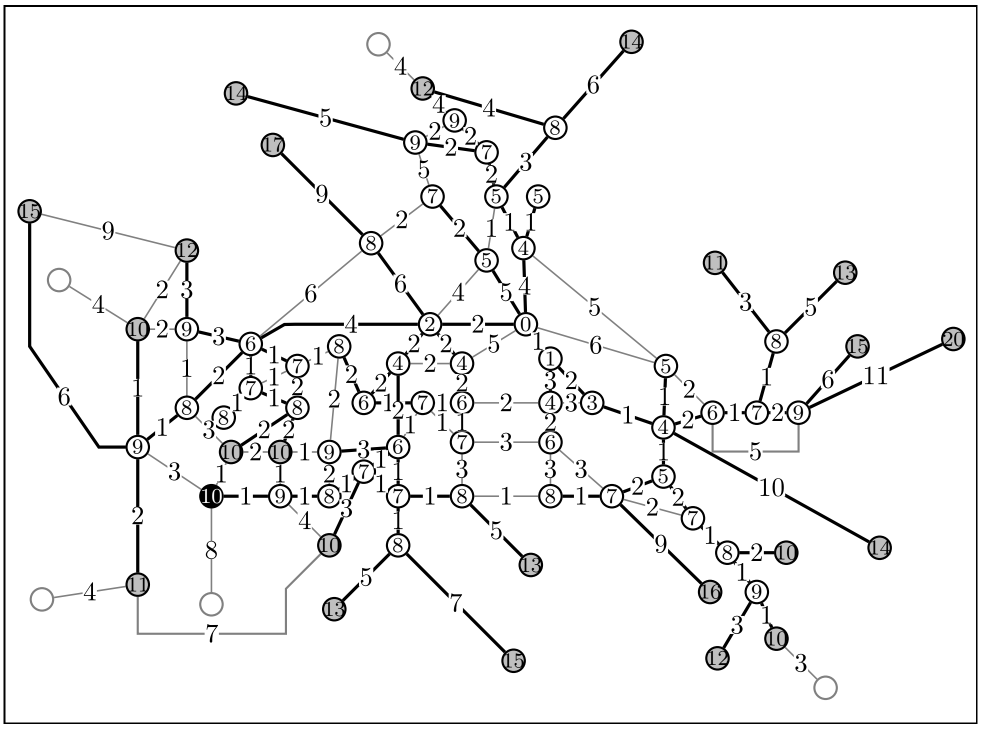


Fig. 33: A shortest path tree.

Try it out! You will obtain marks and a tree as in Figure 33. (The shortest path tree may look a little different, because of free choices at the same distance labels.) The tree contains a surprise: Instead of reaching Heidelberger Platz in 11 edges, as in Figure 17, one can also do so in 10, and so reach Dahlem-Dorf in 14 instead of 15 edges. Did you know this?

*

The *pseudocode* in Figure 34 is a general description of this process. `d(v)` and `pred(v)` are `arrays` for distance marks and predecessor nodes on the shortest path from the start s. The set of permanently marked nodes is denoted by T (tree), and $\delta(v)$ is the list of neighbors of the node v. Also, $w(u, v)$ is the length of the edge from u to v.

This process is called *Dijkstra's algorithm* after its inventor, the Dutch mathematician Edsger Wybe Dijkstra, who first proposed it in 1959. The first node marking algorithm was actually that of the American mathematician L.R. Ford Jr., of 1956, and Dijkstra added the concept of the permanent marking of nodes and the specification of a rule for choosing them from this; Ford's algorithm marked in an arbitrary sequence. A small but fine difference for its efficiency! With its permanent marking Dijkstra's algorithm saves work in enumeration, in that all paths are discarded if their starts are not contained in the growing shortest path tree.

```
algorithm Dijkstra
  forall $v \in V$ do d(v) ← $\infty$;
  d(s) ← 0; pred(s) ← s; $T$ ← $\{s\}$;
  while ($T \neq \emptyset$) do
    determine $v \in T$ such that d(v) = min$_{u \in T}$ d(u);
    $T$ ← $T \cup \{v\}$;
    forall $u \in \delta(v)$ do
      if d(u) > d(v) + w(v,u) then
        d(u) ← d(v) + w(v,u); pred(u) ← v;
      endif
    endforall
  endwhile
endalgorithm
```

Fig. 34: Dijkstra's algorithm.

Flexibility is another plus of Dijkstra's algorithm. We give three examples.

Directed graphs. The process functions without alteration also for any *directed graph* (digraph, for short), whose *arcs* may, like one way streets, be traversed in only one direction.

Zone systems with decreasing zone supplements (Section 2.1) can be handled by adding the extra cost appropriately when updating the distances.

Timetables require time marks and a corresponding search for transfers.

Company	`URL (http://)`
Berliner Verkehrsbetriebe	`www.fahrinfo-berlin.de/Fahrinfo`
Deutsche Bahn	`www.bahn.de`
Norddeutsche Verkehrsbetriebe	`www.efa.de`

Table 1: Passenger information on the Internet.

Passenger information systems such as those mentioned in Table 1 are all based on Dijkstra's algorithm. Only the "Mathematics Inside" can often hardly be recognized with all the surfaces and visualizations. Naturally this is not so for the readers of this article!

2.4 Runtime Law for Algorithms

Theorem 4 (Runtime of Dijkstra's algorithm).
Dijkstra's algorithm can be implemented on a Random Access Machine (computer type with address arithmetic: RAM for short) so that the computing time for a graph with n nodes requires $O(n^2)$ operations.

```
algorithm Dijkstra
  forall $v \in V$ do d(v) $\leftarrow \infty$; T(v) $\leftarrow$ 0;     $n\times O(1)$                     $= O(n)$
  d(s) $\leftarrow$ 0; pred(s) $\leftarrow$ s;                                                         $O(1)$
  forever do                                         $n\times$
    min_d $\leftarrow \infty$;                                                        $O(1)$
    forall $u \in V$ do                                   $n\times$
      if T(u) = 0 && d(u) < min_d then                      $O(1)$
        min_d $\leftarrow$ d(u); v $\leftarrow$ u;                          $O(1)$
      endif
    endforall                                             $\overline{n\times O(1)} = O(n)$
    if min_d = $\infty$ then break;                                             $O(1)$
    $T$(v) $\leftarrow$ 1;                                                            $O(1)$
    forall $u \in \delta$(v) do                                $n\times$
      if d(u) > d(v) + w(v,u) then                          $O(1)$
        d(u) $\leftarrow$ d(v) + w(v,u); pred(u) $\leftarrow$ v;          $O(1)$
      endif
    endforall                                             $\overline{n\times O(1)} = O(n)$
  endforever                                         $\overline{n\times \qquad O(n)} = \underline{O(n^2)}$
endalgorithm                                                                       $\overline{\overline{O(n^2)}}$
```

Fig. 35: Runtime of Dijkstra's algorithm.

Figure 35 shows a suitable implementation. The code indicates that one can determine the computing time as one advances row by row (repetitions in loops are counted multiply) and tots up. How this computation is performed, and what Theorem 4 means in detail, is the concern of the remainder of this section.

O-Notation is a concept for comparing the rates of growth of functions.

$$g(n) = O\big(f(n)\big) \iff \text{there is a constant } C \text{ with } g(n) \le C \cdot f(n) \text{ for all } n,$$

i.e. $g(n) = O\big(f(n)\big)$ (read: g is of the *order* of f) means that up to multiplication by a constant $g(n)$ is always $\le f(n)$. The most important *calculation rules* here are

$$O(1) + O(1) = O(1), \quad O(1) + O(n) = O(n), \quad O(1) + O(n) + O(n^2) = O(n^2)$$

etc.; for a polynomial the highest power determines the order (whence the name). This "Landau O" was invented by the number theoretician Paul Gustav Heinrich Bachmann (1837–1920) and popularized by Edmund Georg Hermann Landau (1877–1938).

The size of the input data can be measured according to the O-notation. If one stores a graph such as in Figure 22 in an *adjacency matrix*, coding Number and Eagle by the whole numbers 1 and 0 of type `int`, there is a storage requirement of $\frac{n^2-n}{2} \cdot$ `sizeof(int)` bits. The value `sizeof(int)` is a *hardware dependent* constant, the bit length of a whole number. Data structures and

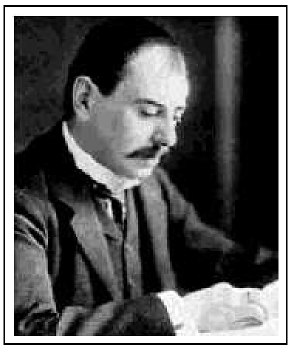


Fig. 36: E.G.H. Landau.

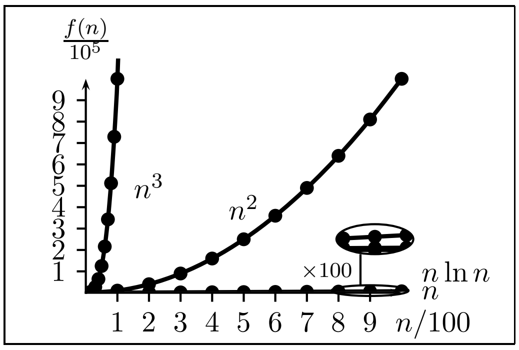


Fig. 37: Polynomial functions.

hardware lead to different storage requirements. Most variations do not affect the *order* of the storage needed (as it depends on n), but affect only constants like `sizeof(int)`. For a graph with n nodes the storage need is at most of the order of magnitude $O(n^2)$ (bits).

Worst case analysis. The cost of thinking in terms of orders of magnitude is that one must always reckon for the worst case, both for input data and runtime.

The random access machine (RAM) is a type of computer that performs only the four *elementary operations*: $+, -$ (addition), $*, /$ (multiplication), $<, >, =$ (comparison) and $\leftarrow$ (assignment). Random access means that there are no limitations on storage access. (The important subtleties of this definition are explained in Mehlhorn's book – see the References.) The RAM-model levels hardware and system differences in microprogramming: counting the elementary operations gives the number of CPU-cycles needed on a real computer.

The runtime of a program on a RAM is the number of elementary operations performed. One can show that this simplified measure really does determine the order of magnitude of the CPU cycles. (This holds for today's computers: there will be different laws for the parallel computers of the future!) The greatest advantage of this concept is that the order of magnitude of the runtime is also independent of the implementation details: A command here or there, or even ten times as many, makes no difference. This invariance allows one to speak of the *runtimes of algorithms* instead of the runtimes of implementations.

$*$

A worst case analysis for Dijkstra's algorithm is shown in Figure 35: The complete runtime is $O(n^2)$. This *polynomial function* presents a different behavior from the *(super-)exponential functions* in Figure 23 that describe the runtime for enumeration. This quality jump is the quantitative basis for the superiority of Dijkstra's algorithm. The linear scale in Figure 37 shows that there are also clear differences in the polynomial domain. The quantum jumps in the growth behavior are a great incentive to reducing the order of the runtime. Dijkstra's algorithm offers potential here. With better list and heap data structures one can reduce the runtime relatively easily to $O(m \ln n)$ (for $m < n^2/\ln n$), where m is the number the edges. Data structures even more finely tuned to this purpose, such as "Fibonacci heaps", free further potential. Faster than $O(m)$ is impossible because one needs this much time to read the data. Is there an *optimal SSSP-algorithm* with this runtime? The Danish mathematician Mikkel Thorup showed in 1996 that this is actually possible! There is, however, a little catch: The process is based on an "atomic heap" data structure proposed by the American mathematicians Michael L. Fredman and Daniel E. Willard, applicable for $n > 2^{12^{20}}$. Although $2^{12^{20}} = O(1)$ *this* constant is so large that the algorithm is not implementable! What next? Thorup proposes a "slimmer" variant, where the runtime is "worsened" to $O(\ln C + m + n \ln\ln\ln n)$ (C is the largest edge weight). As you can see: The topic SSSP remains to be developed!

2.5 Limited Resources

Shortest path problems rarely appear in pure form. Even for Tell there were *constraints* on the form of the paths. *Resource constraints* are a useful framework. Such models envisage, along with the objective function, a number of *resources*, to be used along the edges. At the nodes the constraints restrict the forms of the incoming paths to admissible *states* of resource use. We give two examples.

Time windows are intervals $[a_v, b_v]$ at the nodes $v \in V$ that restrict the use of a time resource t_{uv} for traversing the edge $uv \in E$.

Path length constraints are handled like time windows. When the term length is applied to a resource one speaks of weight in the objective function.

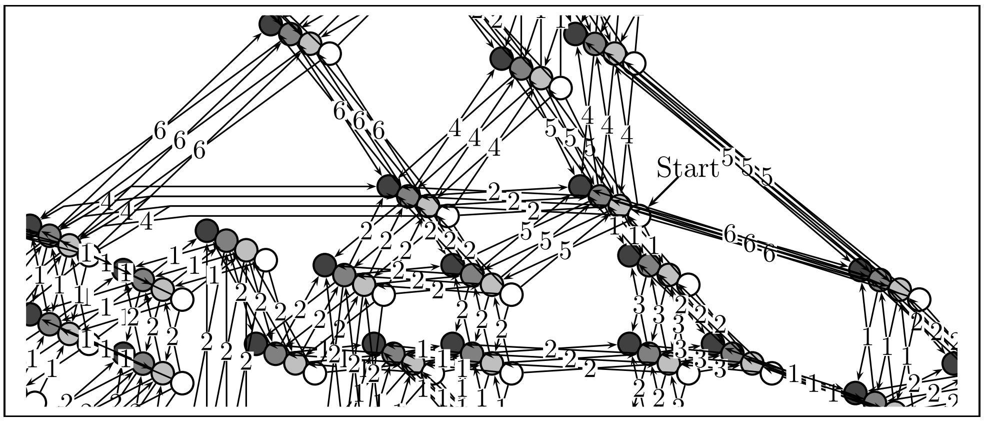


Fig. 38: Resource Graph.

Shortest path problems with (discrete) resource constraints can be reduced by a *transformation* to the *directed* standard form (cf. Page 15). Figure 38 illustrates this principle for a variant of the S-Bahn example with length constraints: To find the shortest paths with exactly 3 edges starting from Alexanderplatz. A unit of edge resource is consumed along each edge. At each node v in the original graph there are four possible admissible resource states $R_v = \{0, 1, 2, 3\}$ (an incoming path has already “used” 0, 1, 2 or 3 edges). The transformation builds four copies from each original node, one for each resource state. In Figure 38 the copies are arranged in four planes 0–3. Directed arcs join copies of nodes whose originals are neighbors; the arcs always run “upwards” from the lower to the higher numbered plane, the weights are the same as for the original edges. The *(resource)state digraph* so constructed has the property that the shortest paths from Alexanderplatz in Plane 0 to the nodes in Plane 3 correspond exactly to the shortest paths with 3 edges in the original graph. The directed arcs prevent longer paths arising from “jumping back”.

This construction works generally for discrete resource states R_v. From one node v in the original graph one generates $|R_v|$ copies in the state digraph, joined by arcs according to the possible resource consumptions. Then the runtime of Dijkstra’s algorithm is $O\big((\sum_{v\in V} |R_v|)^2\big)$. This quantity is in general *not* polynomial in the number n of nodes of the initial digraph. Only in special circumstances, e.g. when $\sum_{v\in V} |R_v|$ is polynomial in n, do we have polynomial runtimes. One calls such a runtime behavior *pseudopolynomial*.

The construction of the state digraph offers room for improvement. The idea is to perform the transformation only *implicitly* and to work in the original digraph with several marks per node. These *Multilabel SSSP algorithms* require less storage space, and often (as in the timetable example of Page 15) the runtime is also more favorable.

Exercise 4 (Solution in Section 6)**.** *Why does an n plane transformation not deliver a polynomial algorithm for the Hamiltonian Circuit Problem?*

3 Combinations of Paths

After the shortest path problem the next level is the simultaneous planning of *several paths*. The typical difficulty is to make the “right detour” occasionally. We will not try to provide a survey of covering, packing and partitioning problems for paths, but only to describe the best known types. We restrict ourselves here to a representative example: Branch & Price methods for Set Partitioning Models in Bus Driver Scheduling.

3.1 Duty Scheduling 'Light'

As of 31 December 2006 the Berliner Verkehrsbetriebe (BVG) deploys 1,310 buses, driven by about 4,000 bus drivers. During the year there were 403.8 million passenger journeys on 147 lines with a total length of 1,656 km. With these numbers it is clear that buses and drivers must be employed as efficiently as possible. This is not so simple. In particular, a driver's duty is subject to complicated regulations: among others, to ensure sufficient breaks. But these necessary breaks should not lead to further waiting times, with duties so unfavorable to the timetable that drivers may be available but no journeys are to be made. Moreover, one wants to devise the most convenient duties for the workers and, e.g., keep small the number of split duties that require people to come in to work twice in a day.

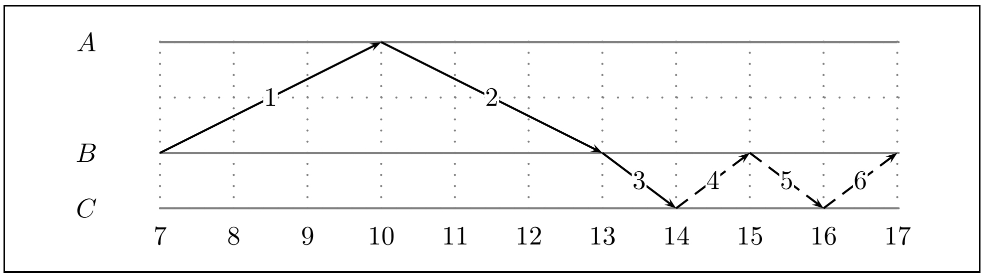


Fig. 39: Simplified duty scheduling problem.

Figure 39 illustrates a simplified duty scheduling problem: To service trips 1–6 between the termini A, B and C. The trip times are indicated by the time bar at the bottom edge of the figure. Trip 1 begins at 7 o'clock and ends at 10 o'clock, etc. We need to specify duties to cover these trips, and agree on the following rules. The driving time is not to exceed 7 hours, and there should be no break of more than 3 hours between two trips. The costs of a duty are determined as the duty length (that is, driving time plus break resp. transfer times) and an additional supplement of 2 hours per shift. The 1-2-3 duty, shown by an unbroken line in Figure 39, is permitted, with a driving time of 7 hours and 0 hours between trips; its cost is $3 + 3 + 1 + 2 = 9$. Similarly, duty 4-5-6, shown by a dashed line, with driving of 3 hours and changes of 0 hours, is also admissible; its cost is $1 + 1 + 1 + 2 = 5$. Together these two duties form a schedule with a total cost of $9 + 5 = 14$, with all the trips covered.

For such a small example with simple rules the scheduling is simple. In reality there are very many duty elements to cover. The duties have to conform to complicated rules such as those in Figure 40, an example of a European Union rule. A further difficulty is that the rules and objectives change regularly (because of company agreements, political guidelines, etc.). One can imagine the complications this brings to a proper scheduling. The danger in this situation is that one may, with *ad hoc methods*, develop a complex and confusing model that will not lend itself easily to changes, and may deliver solutions of doubtful

Council Regulation (EEC) No 3820/85 of 20 December 1985 on the harmonisation of certain social legislation relating to road transport.

SECTION V Breaks and Rest Periods

Article 7.

(1) After 4.5 hours' driving, the driver shall observe a break of at least 45 minutes, unless he begins a rest period.

(2) This break may be replaced by breaks of at least 15 minutes each distributed over the driving period or immediately after this period in such a way as to comply with the provisions of paragraph 1.

(3) By way of exception from paragraph 1, in the case of national carriage of passengers on regular services Member States may fix the minimum break within at not less than 30 minutes after a driving period not exceeding four hours. Such exceptions may be granted only in cases where breaks in driving of over 30 minutes could hamper the flow of urban traffic and where it is not possible for drivers to take a 15 minute break within 4.5 hours driving prior to a 30 minute break.

Fig. 40: From the EU Break Regulations for Bus Drivers.

quality. We present the mathematical alternative in the following section.

3.2 Duties and Paths

The first step towards a better method is the development of a mathematical model. For this, duties are represented as paths in an appropriate scheduling graph that "cover" the trip elements. One speaks of a "path covering problem".

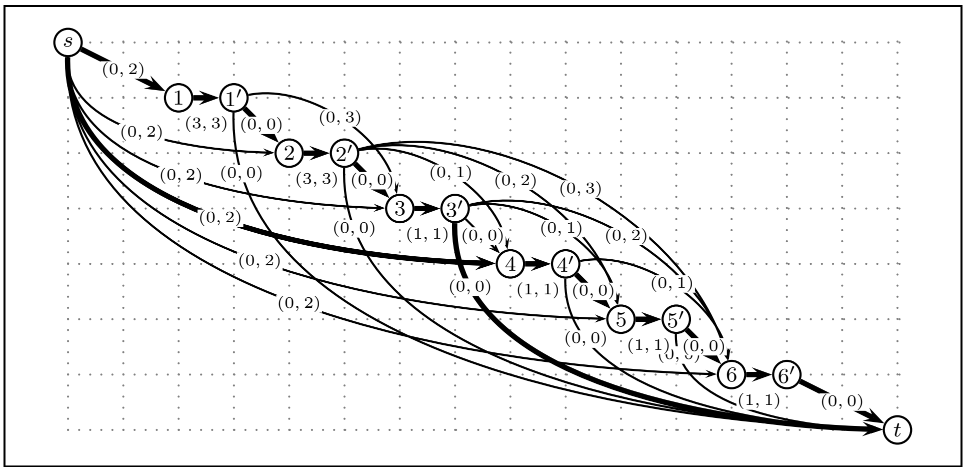


Fig. 41: Shift planning graph; $(t_{ij}, c_{ij}) = (\text{driving time}, \text{costs})$.

Figure 41 shows the construction for the example of Figure 39. For each trip element i there are two nodes i and i', to denote the start and finish of the trip. Also two "artificial nodes", s and t, are introduced to represent the start and finish of the duty. The nodes are connected by arcs representing the possible transitions between the events. The arcs are marked by number pairs (t_{ij}, c_{ij}),

c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}	c_{11}	c_{12}	c_{13}	c_{14}	c_{15}	c_{16}	c_{17}	c_{18}	c_{19}	c_{20}	c_{21}	c_{22}	c_{23}	c_{24}	c_{25}	c_{26}	c_{27}	c_{28}	c_{29}	c_{30}	c_{31}	c_{32}	c_{33}	c_{34}	c_{35}	c_{36}	c_{37}
5	5	3	3	3	3	8	9	6	7	8	9	4	5	6	4	5	4	9	10	11	12	7	8	9	5	6	5	11	12	12	8	9	9	6	12	9
$a_{\cdot 1}$	$a_{\cdot 2}$	$a_{\cdot 3}$	$a_{\cdot 4}$	$a_{\cdot 5}$	$a_{\cdot 6}$	$a_{\cdot 7}$	$a_{\cdot 8}$	$a_{\cdot 9}$	$a_{\cdot 10}$	$a_{\cdot 11}$	$a_{\cdot 12}$	$a_{\cdot 13}$	$a_{\cdot 14}$	$a_{\cdot 15}$	$a_{\cdot 16}$	$a_{\cdot 17}$	$a_{\cdot 18}$	$a_{\cdot 19}$	$a_{\cdot 20}$	$a_{\cdot 21}$	$a_{\cdot 22}$	$a_{\cdot 23}$	$a_{\cdot 24}$	$a_{\cdot 25}$	$a_{\cdot 26}$	$a_{\cdot 27}$	$a_{\cdot 28}$	$a_{\cdot 29}$	$a_{\cdot 30}$	$a_{\cdot 31}$	$a_{\cdot 32}$	$a_{\cdot 33}$	$a_{\cdot 34}$	$a_{\cdot 35}$	$a_{\cdot 36}$	$a_{\cdot 37}$
1	.	.	.	.	.	1	1	.	.	.	.	.	.	.	.	.	.	1	1	1	1	.	.	.	.	.	.	1	1	1	.	.	.	.	1	.
.	1	.	.	.	.	1	.	1	1	1	1	.	.	.	.	.	.	1	1	1	1	1	1	1	.	.	.	.	.	.	1	1	1	.	.	1
.	.	1	.	.	.	.	1	1	.	.	.	1	1	1	.	.	.	1	.	.	.	1	1	1	1	1	.	1	1	.	1	1	.	1	1	1
.	.	.	1	.	.	.	.	.	1	.	.	1	.	.	1	1	.	.	1	.	.	1	.	.	1	1	1	1	1	1	1	1	1	1	1	1
.	.	.	.	1	.	.	.	.	.	1	.	.	1	.	1	.	1	.	.	1	.	.	1	.	1	.	1	1	.	1	1	.	1	1	1	1
.	.	.	.	.	1	.	.	.	.	.	1	.	.	1	.	1	1	.	.	.	1	.	.	1	.	1	1	.	1	1	.	1	1	1	1	1
x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}	x_{16}	x_{17}	x_{18}	x_{19}	x_{20}	x_{21}	x_{22}	x_{23}	x_{24}	x_{25}	x_{26}	x_{27}	x_{28}	x_{29}	x_{30}	x_{31}	x_{32}	x_{33}	x_{34}	x_{35}	x_{36}	x_{37}

Table 2: All duties.

where t_{ij} is the driving time on going from node i to node j, and c_{ij} is the cost.

The duty scheduling graph is so constructed that every admissible duty corresponds to a path from s to t: but not exactly the converse. For an (s,t)-path to correspond to a duty the sum of the driving times on its arcs must not exceed 7. With respect to the resource "driving time" the duties correspond to resource constrained paths, as discussed in Section 2.5. Please note, dear reader, that the second scheduling rule, restricting the break between duties to at most 3 hours, can be treated in a very simple way, by introducing, or not introducing, arcs into the scheduling graph. For example, the arc $1'4$ must be removed since it represents an illegal change from the end of trip 1 to the beginning of trip 4 with a gap of 4 hours. In all, a schedule corresponds to a set of driving time constrained (s,t)-paths with all the trip elements covered exactly once. That is, we aim at a "path covering" at minimal total cost. The question remains: how can one find the right paths?

3.3 Set Partitioning Models

The starting point for a solution to this question is a *conceptional enumeration* of all duties. Table 2 lists the results in a compressed form. Each column stands for one of 37 possible duties. Above, for each duty j the cost coefficient c_j is listed. Then follows the trip *incidence vector* $a_{\cdot j}$ (here $\cdot$ is a placeholder for an index). It specifies which trips the duty j implements (1) or does not implement (.). Row i thus belongs to trip i, $i = 1, \ldots, 6$. The duties 1-2-3 and 4-5-6 appear in columns 19 and 28.

This preparation makes it easy to formulate the duty scheduling problem as an *integer program*, a model type of discrete optimization. Figure 42 shows the program in the "LP-format" accepted by many code packages. For each duty the program holds a 0/1 *decision variable* x_j (`Binaries`, see also Table 2) for implementation of a duty j ($x_j = 1$) or nonimplementation ($x_j = 0$). The objective function row (`obj`) totals the costs of the incorporated duties, and is to be minimized (`Minimize`). Each of the 6 equations (`c1-c6`) ensures that of all the duties that cover a trip element exactly one will be implemented (`ci` belongs to trip i). The *solutions* of this program correspond exactly to the possible duty schedules. Since the trips have to be partitioned into duties one calls this type of integer program a *Set Partitioning Problem*: SPP for short.

```
Minimize
obj: + 5x1 + 5x2 +3x3 +3x4 +3x5 + 3x6 +8x7 +9x8 + 6x9 +7x10
     + 8x11 + 9x12 +4x13 +5x14 +6x15 + 4x16 +5x17 +4x18 + 9x19 +10x20
     +11x21 +12x22 +7x23 +8x24 +9x25 + 5x26 +6x27 +5x28 +11x29 +12x30
     +12x31 + 8x32 +9x33 +9x34 +6x35 +12x36 +9x37
Subject To
c1:  x1+x7+x8+x19+x20+x21+x22+x29+x30+x31+x36=1
c2:  x2+x7+x9+x10+x11+x12+x19+x20+x21+x22+x23+x24+x25+x32+x33+x34+x37=1
c3:  x3+x8+x9+x13+x14+x15+x19+x23+x24+x25+x26+x27+x29+x30+x32+x33+x35+x36+x37=1
c4:  x4+x10+x13+x16+x17+x20+x23+x26+x27+x28+x29+x30+x31+x32+x33+x34+x35+x36+x37=1
c5:  x5+x11+x14+x16+x18+x21+x24+x26+x28+x29+x31+x32+x34+x35+x36+x37=1
c6:  x6+x12+x15+x17+x18+x22+x25+x27+x28+x30+x31+x33+x34+x35+x36+x37=1
Binaries
     x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16 x17 x18 x19 x20
     x21 x22 x23 x24 x25 x26 x27 x28 x29 x30 x31 x32 x33 x34 x35 x36 x37
End
```

Fig. 42: A set partitioning problem.

Any serious code will solve problems with 37 variables and 6 equations like lightning. Our solver needs less than a millisecond to find the solution and show that it is optimal. We already know the result: $x_{19} = x_{28} = 1$, the other variables being 0.

Real scheduling does not go so easily. With 2,000 trip elements there are at least $\binom{2.000}{10} \geq 10^{20}$ duties, and one can no longer enumerate them. Was all this set partitioning for nothing? Certainly not! But we must say farewell to the idea that to solve a model we have to write it down completely. Only a few duties are relevant to the optimal solution (in the example, just 2), the numerous others (here 35) are only ballast at the end. Again, as Euler said: If one were to examine it in the way alluded to above one would find much not in queſtion; this is doubtleſſ the reaſon why this route would be ſo burdenſome.

3.4 Plans = Paths + Prices + Programs

The fundamental idea for solving larger set partitioning problems is astonishingly simple: Start with a (however cleverly generated) small set of duties and *generate* the missing duties as needed. This *column generation* is the most important component of all *Branch & Price* algorithms. The decisive trick is to generate the columns in a way that guarantees an improvement. We shall now give an impression of how this works.

Initialization. We need some solution to start from, so we choose the expensive, but always possible, “1s disposition” $x_1 = \cdots = x_6 = 1$ with cost

$5+5+3+3+3+3 = 22$, in which every trip is handled by its own driver. The 6 incidence vectors $a_{\cdot 1}$–$a_{\cdot 6}$ and the costs c_1–c_6 are all that currently is known for the SPP of Figure 42.

BTRAN. The *backward transformation* is a *cost centre computation*: The total cost is apportioned from the duties to the trips "caused". (Shadow-) *prices* π_i for the trips result. These prices are found as solutions of the equations $\sum_{i=1}^{6} a_{ij}\pi_i = c_j$ for all duties j with $x_j = 1$. In the example these are the equations

$$\begin{array}{llllllll}
\pi_1 & & & & & & = 5 = c_1 & \text{(price equation duty 1)} \\
& \pi_2 & & & & & = 5 = c_2 & \text{(price equation duty 2)} \\
& & \pi_3 & & & & = 3 = c_3 & \text{(price equation duty 3)} \\
& & & \pi_4 & & & = 3 = c_4 & \text{(price equation duty 4)} \\
& & & & \pi_5 & & = 3 = c_5 & \text{(price equation duty 5)} \\
& & & & & \pi_6 & = 3 = c_6 & \text{(price equation duty 6).}
\end{array}$$

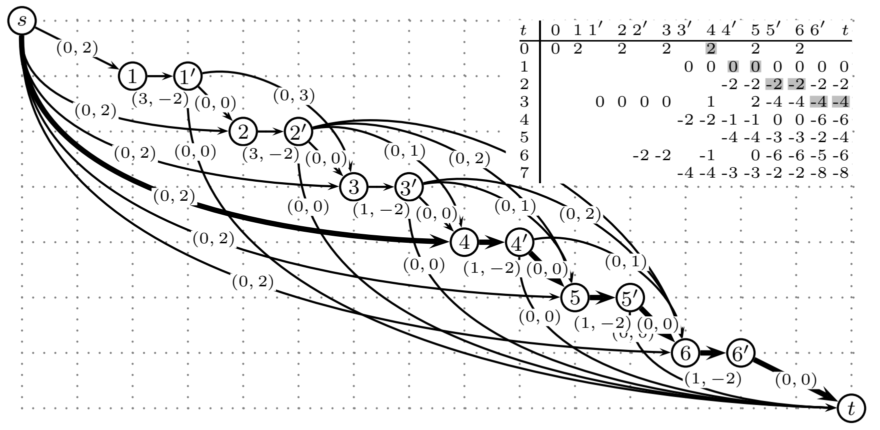
The solution is $\pi_1 = \pi_2 = 5$, $\pi_3 = \pi_4 = \pi_5 = \pi_6 = 3$.

Pricing. This step is an *investment appraisal*: One looks for alternative duties which undercut the shadow price. The difference, the *reduced cost*, does exactly this. The formula for the reduced cost $\overline{c}_j$ of duty j is

$$\overline{c}_j := c_j - \sum_{i=1}^{6} a_{ij}\pi_i. \tag{1}$$

Determining a duty with negative reduced costs is – and so we come back to the paths – a shortest path problem with resource constraints. Figure 43 shows the digraph. In order that the reduced costs will correspond to path lengths the prices are transferred, as in Figure 20, to the arcs pointing out from the starting node of the trips. The prices go into formula (1) with negative sign so are subtracted. Thus we have *negative arc weights*. SSSP algorithms have difficulties with negative weights, which hinder the permanent marking of nodes (one can *lower* the cost on a path, and indeed SSSPs with negative weights are $\mathcal{NP}$-hard). In our case this is not so: In an *acyclic digraph* (without directed cycles) Dijkstra's algorithm works with negative arc weights too, if one marks the nodes in the correct sequence (a *topological order*) "from front to back".

For the proper calculation one needs marks for the driving time t already consumed (which can take the values $t = 0, 1, 2, \ldots, 7$) at the nodes, in addition to the distance marks. The result is presented in the Table in Figure 43. In row t, $t = 0, \ldots, 7$, it holds the lengths of the shortest paths with a driving time to the respective nodes of at most t. The computation of the paths proceeds by rows in the sequence of increasing time marks. The entry -4 in position $(3, t)$ shows that there is an (s, t)-path with a driving time of 3 units and reduced cost of -4. Backtracking the calculation shows that this is for the path $s44'55'66't$, corresponding to duty No. 28, that is, to the duty 4-5-6. This duty has a cost of $2 + 1 + 1 + 1 = 5$, while the sum of the shadow prices for the trips 4, 5 and 6 it contains has the value $3 + 3 + 3 = 9$. The difference $5 - 9 = -4$ is precisely the reduced cost.



t	0	1	1'	2	2'	3	3'	4	4'	5	5'	6	6'	t
0	0	2		2		2		2		2		2		
1							0	0	0	0	0	0	0	0
2									-2	-2	-2	-2	-2	-2
3			0	0	0	0		1		2	-4	-4	-4	-4
4							-2	-2	-1	-1	0	0	-6	-6
5									-4	-4	-3	-3	-2	-4
6					-2	-2		-1		0	-6	-6	-5	-6
7							-4	-4	-3	-3	-2	-2	-8	-8

Fig. 43: Column generation.

FTRAN. The *forward transformation* is a *production assessment*: The new duty 28 is to replace several old duties. How is clear: duty 28 replaces the duties 4, 5 and 6; the new solution is $x_1 = x_2 = x_3 = x_{28} = 1$. The costs are $c_1 + c_2 + c_3 + c_{28} = 5 + 5 + 3 + 5 = 18$, a saving against the value 22 of the previous solution of $\overline{c}_{28} = -4$.

Optimality criterion. Now that the total cost has sunk from 22 to 18 the shadow prices have to be adjusted for the next iteration. Subsequently one seeks an improved duty and builds it into the solution. And so the costs sink further. This proceeds until there are no paths with negative length. It is shown in linear optimization theory that: If the reduced costs of all duties are nonnegative then the current solution is optimal. This *stopping criterion* is the signal to finish the computation. The optimal solution has been found!

Runtime. An important question arises: How many iterations must be performed, how many duties will be generated? If one does it right, not very many. The second author has developed a theory of *polynomial equivalence of separation and optimization* (here applied dually) together with the Dutch mathematician Alexander Schrijver and the Hungarian mathematician László Lovász which shows that one can succeed with a polynomial number of duties. These results are the theoretical foundation of the empirical efficiency of column generation.

Branch & Price. Together with a few technical problems we have suppressed a difficulty in our presentation so far. This is the fact that incorporating a newly determined duty into a solution often cannot be effected by a swap. A mathematical expedient is to allow “fractional solutions” too, where fractions of tours appear. Such a solution has no meaning in the real world. It is merely an intermediate result in a computation at the end of which there should be an integer solution. But what if the column generation (the Price in Branch & Price) terminates with a fractional optimal solution? Then an integer solution can be guaranteed only with a downstream enumeration (Branch), and this generally leads to an exponential total runtime.

One is often lucky and column generation provides a integer solution quickly. This is not the rule, but often, with simple *heuristics*, one can construct integer solutions from fractional ones, the objective functions being close (in the ideal case they are the same). In between is the *duality gap* in which the unknown optimum lies. The size of the gap, in *ignorance of the exact optimum*, is a measure of the *quality* of a heuristic solution. Statements of this sort are important in assessing *possible savings*.

*

The authors have developed a duty schedule optimization system DS-OPT with the help of column generation methods. It has been integrated into the planning system `ivu.plan` (formerly called MICROBUS 2) of IVU Traffic Technologies AG, and is employed by many transport companies here and abroad, also by BVG. Mathematics has been able to contribute to making public transport more attractive and to remain affordable.

4 Outlook

Discrete optimization, and with it the mathematics of paths, has experienced an upswing over sixty years. Today good models, algorithms, and a general theory are available.

Unfortunately their application does not always reflect the state of the mathematics. In contrast to the permeation of the methods of the differential calculus into technical engineering disciplines, discrete methods remain exotica in planning, logistics, decision support, and supply chain management. There are many reasons: The sizes of discrete models, lack of acquaintance with the necessary mathematics, or even skepticism concerning the mathematics in many planning circles, and a lack of interest in optimization because of monopolistic structures in important fields such as public transport.

This state of affairs is changing. The performance of computers and processes is now at a useful level, and is improving. Monopolies are being dissolved. There is a chance of bringing discrete methods into planning and logistics. Our vision is that *Computer Aided Scheduling* (CAS) will assume the same significance in logistics as CAD and CAM in production. The possibilities lie to hand: cost cutting, quality improvement, planning tempo and flexibility, scenario analyses, etc. Set partitioning models have proved their value in scheduling in public transport, and are already the industry standard in the competitive air transport. Scheduling of all the vehicles of concerns such as the BVG (the fourth largest public transit company in the world) is possible with multicommodity flow methods. One can no longer imagine doing without methods for solving path problems in telecommunications network design. Where next? The future will show!

5 Further Reading

You have acquired a taste for discrete optimization and want to know more about the mathematics of paths? This section gives a list of articles for further reading, sorted by topics, many of which can be downloaded from the Internet.

Shortest paths. The articles of the *9th DIMACS Implementation Challenge – Shortest Path* give a survey on the newest research results on shortest paths. They are available at `http://www.dis.uniroma1.it/~challenge9/papers.shtml`.

Transportation problems. Our article *Alcuin's Transportation Problems and Integer Programming* in Volume 2 of the book *Charlemagne and his Heritage: 1200 Years of Civilization and Science in Europe* by Paul Leo Butzer, Hubertus Th. Jongen and Walter Oberschelp, published in 1998 by Brepols, Turnhout, Belgium, gives an entertaining introduction to mathematical transport optimization, based on the well-known wolf-goat-cabbage problems that Charlemagne and his chief counsellor Alcuin of York invented 1200 years ago in order to improve the teaching of Mathematics in Franconian schools. The article is available on the Internet at `http://www.zib.de/ZIBbib/Publications/` as ZIB Preprint SC 95-27.

The Königsberg Bridge problem. Don't miss out on reading Leonhard Euler's article (of the same name) of 1736! An English translation was published in 1953 under the title *Leonhard Euler and the Koenigsberg Bridges* in *Scientific American*, Volume 189, pages 66–70, edited by J.R. Newman. The Latin original is available from the Euler archive at `http://www.math.dartmouth.edu/~euler/docs/originals/E053.pdf`.

Traveling salesman problem. Martin Grötschel's and Manfred Padberg's article *The Optimized Odyssey*, published in 2001 in *AIROnews*, Volume VI, No. 3, pages 6–9, available on the Internet at `http://www.zib.de/groetschel/pubnew/paper/groetschelpadberg2001a.pdf`, informs about the most famous optimization problems. The latest news is reported in the 606 page book *The Traveling Salesman Problem: A Computational Study* by David Applegate, Robert Bixby, Vašek Chvátal and William Cook. It was published by Princeton University Press in 2007.

Public transport. Vehicle scheduling and duty scheduling in public transport is described by Ralf Borndörfer, Martin Grötschel and Marc Pfetsch in the article *Public transport to the fORe*, published in 2006 in *OR/MS Today*, Volume 33, No. 2, pages 30–40. The article is available online at http://www.lionhrtpub.com/orms/orms-4-06/frtransport.html.

Algorithmic Graph Theory. The books *Graph Theory with Applications* by Adrian Bondy and U.S.R. Murty, published in 1976 by Elsevier Science, New York, and *Graph Theory* by Reinhard Diestel, published in 2005 by Springer Verlag, Berlin, are excellent references. Both books are available online at http://www.ecp6.jussieu.fr/pageperso/bondy/books/gtwa/gtwa.html and at http://www.math.uni-hamburg.de/home/diestel/books/graph.theory/GraphTheoryIII.counted.pdf, respectively.

Linear Programming. We recommend the books *Linear Programming* by Vašek Chvátal, published in 1980 by Freeman, New York, and *Linear Programming: Foundations and Extensions* by Robert Vanderbei, published in 2007 by Springer Verlag. The latter book is available online at http://www.princeton.edu/~rvdb/LPbook/onlinebook.pdf.

Complexity, Runtime, Computers. Kurt Mehlhorn's books *Data Structures and Efficient Algorithms*, Volumes 1-3, published in 1984 in the EATCS Mongraphs series by Springer Verlag, Berlin, are a standard reference. These books are out of print, but available online at http://www.mpi-inf.mpg.de/~mehlhorn/DatAlgbooks.html.

6 Solutions to the Questions

Euler tour. Four nodes have odd degree. By Theorem 2 there is no Euler tour.

Icosian game. Two original Hamilton cycles: B C P N M D F K L T S R Q Z X W V J H G and B C P N M D F G H X W V J K L T S R Q Z.

Honeycomb. There is no Hamiltonian circuit. The honeycomb is *bipartite* (two colorable): All edges lead from nodes with even number (g) to odd (u) nodes. A Hamiltonian circuit would have the form g u g u . . . g u and thus an even number of nodes. But the honeycomb has 13 nodes.

HCP as SSSP, page 19. Node repetitions can occur in the original graph.