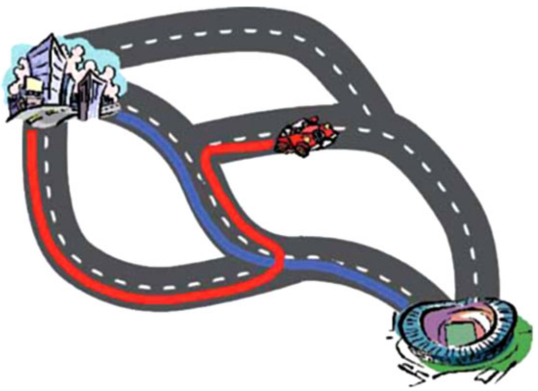
34

Maximum Flows – Towards the Stadium During Rush Hour

Robert Görke, Steffen Mecke, and Dorothea Wagner

Karlsruher Institut für Technologie (KIT), Karlsruhe, Germany

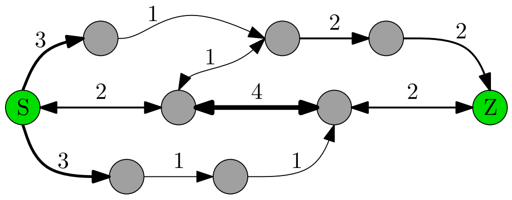
“What the heck is this? We’ll never get to the soccer stadium this way!” Jogi sat in the car, next to his mother, and was beginning to get nervous. “It’s not my fault that everybody’s using this street to the stadium. Now there is a traffic jam,” she said. “Then simply turn around, and let’s take Ford Street over there. Nobody uses that route.” Jogi’s mother did not really buy this, but for the sake of peace she drove back, and, indeed, on Ford Street there was less traffic. Up to the next junction at least. There, Ford Street led into busy and broad Station Street, and there was the traffic jam again. “These idiots don’t know what they’re doing. Turn left, mum!” – “But the stadium is straight ahead,” she replied. “That’s true,” Jogi returned, “but we can take Karl Street over there. While that is a detour, that street will definitely be free.” Jogi’s mum remained skeptical, but she gave it a try. And it actually appeared that Jogi was right. Nobody was taking that detour. Jogi even made a futile attempt to encourage other cars coming towards them to turn around, but nobody followed them. “Those fools! In a second they’ll be stuck in the traffic jam although they could easily get through here!”



B. Vöcking et al. (eds.), *Algorithms Unplugged*,
DOI 10.1007/978-3-642-15328-0_34, © Springer-Verlag Berlin Heidelberg 2011

Jogi arrived at the stadium in time, but the match turned out to be a boring waste of time, which made Jogi think about the traffic situation earlier. "Letting drivers choose their routes by themselves easily leads to congestion. Traffic signs should be put up at each junction in order to route the cars in such a way that traffic keeps flowing and as many cars as possible can reach their goal soon. But how can we find the best solution for this routing problem?" He did not arrive at a satisfactory answer immediately. However, a few days later he spoke to his older sister about the incident. She studied computer science but nonetheless did not have a solution at hand.

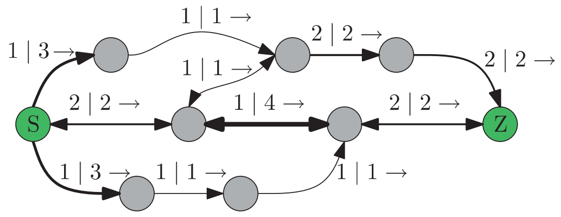
"Let's simplify the problem as much as possible first: Let's assume that all drivers start from the same point ..." She marked the point on a piece of paper and labeled it S for "start." "... and want to go to another point." She marked that one with a Z for Zuse Stadium. "In between, there are streets which meet at the junctions." She drew several more points and lines between the start and Zuse Stadium.



"But the streets can differ in capacity. Let's put the number of lanes right next to each street and thicken the lines in the drawing accordingly. Hmm, we learned about shortest paths in a lecture recently, but that doesn't help much here. Of course we can start by finding the shortest path from S to Z." – "This one here?" Jogi marked it in the drawing. "But now we can continue looking for more routes. We just have to record how many cars are already using each street."

Let us leave Jogi and his sister alone for now and continue their approach: We are given a road network with road capacities, and we would like to know how to route the cars in order to let traffic flow optimally. To keep things simple, we assume that all cars start at S and aim for Z. Usually, car drivers try to take the shortest path to their goal. But when too many cars take a route at the same time, the traffic gets stuck. In our case "at the same time" means "in the hour before the soccer match starts." Each street can only handle a certain number of cars passing through it (the *capacity* of the street). This number does not so much depend on the street's length but rather on its width (the number of lanes). For example, a street with 1 lane can be used by 1000 cars per hour. However, we still write 1 (the number of lanes) instead of 1000, since we want to avoid dealing with such big numbers.

Other critical points in road networks are junctions. In case more cars arrive at a junction than can continue, we have a congestion. By contrast, if the number of cars that can leave a junction does not fall short of the number of arriving cars, all is well. We now ask ourselves how many cars we can simultaneously push through the network from S to Z. ("Simultaneously" means "per hour" here.) A solution to the problem in our example would be the following "traffic flow":



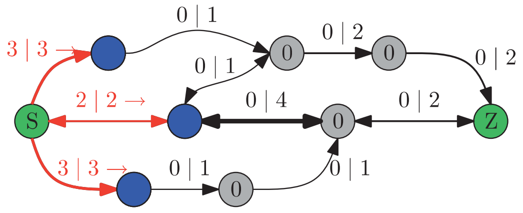
Each street is now additionally labeled by the number of cars using it and an arrow denoting their driving direction. Thus, 1 | 3 → means that one of three available lanes is in use by cars going to the right. The first number must never exceed the number of lanes (i.e., something like 3 | 2 is not allowed). This rule is so important that we give it a name. We call it the *capacity rule.* The requirement that the numbers of cars leaving and arriving at a junction are equal (otherwise cars will get stuck) is called *flow conservation rule* (as it ensures a steady flow at junctions). Only S and Z are an exception to this rule.

Among all traffic flows fulfilling these two rules, we now seek one which allows as many cars as possible to start off simultaneously or – which is the same – to arrive at the goal. Computer scientists call the result a *maximum flow.*

This kind of problem does not only occur in the field of traffic routing. For instance, you could also think about how to evacuate buildings as fast as possible or how to route data through a computer network. Can you think of other examples?

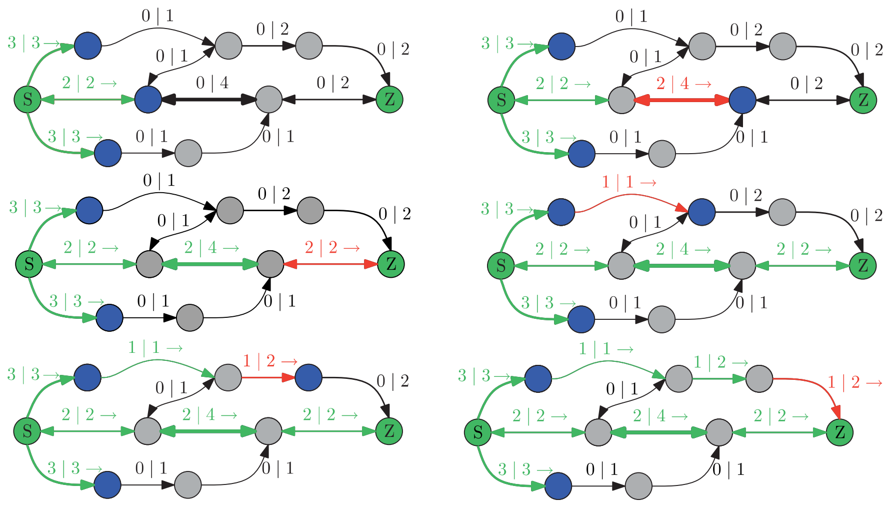
The Algorithm

So how do we find a maximum flow? Let's just give it a try: to start with, there are no cars at all in our road network. We start by sending out (red in the picture) as many cars as possible from S without violating the capacity rule.



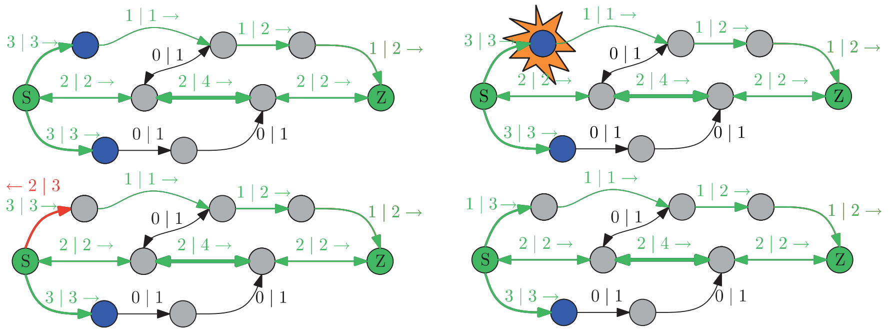
On each street leaving S, there are now as many cars as there are free lanes (green). Of course, we do not use real cars for this purpose but, e.g., toy cars instead. Or simply pen and paper. As soon as we have determined the best solution for our problem, we can start routing real traffic on real streets with real cars.

Now, obviously, the junctions these streets end at have an *excess flow* (blue) of cars, that we have to send somewhere



such that the flow conservation rule is not violated. In order to get rid of this congestion, we just push cars along some street leaving this junction (again red). However, we always have to comply with the capacity rule and must not push forth more cars than there are lanes. Inevitably, this causes a new congestion at the next junction (blue). But when the cars finally arrive at Z, we need not push them any further (but simply put them in the parking lot).

We have seen that we cannot always push forth as many cars as we would like to. The crucial point is: We always push forth as many cars as possible but never more cars than there are lanes and, of course, at most as many as are stuck at the junction. And, by any means, we have to obey one-way streets. In case we are stuck altogether – because more cars arrive at a junction than can possibly depart by all the streets leaving it – we have to be allowed to "push back" cars (like in the junction at the upper left in the following picture).



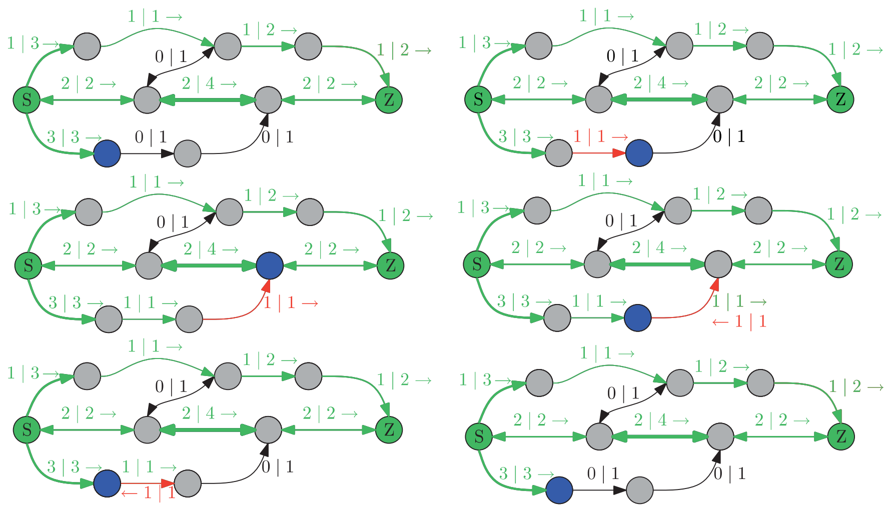
By doing so we reduce the number of cars that use a street. In fact it is not allowed to go backwards on one-way streets, but remember that we are only simulating here. Needless to say, we only push back as many cars as necessary in order to get rid of the excess flow at the junction (after that the junction is grey). And naturally we cannot push back more cars than have arrived in the first place.

To sum up: In every step we select a junction with excess flow (i.e., a junction where more cars arrive than depart) and push forth as big a portion of those excess cars as possible. This results in the following procedure:

The procedure Push pushes cars from a junction with excess flow (either forward or backward).

```
procedure Push ($C$)
precondition $C$ is a junction with excess flow
begin
    choose one of the following:
        Select among the streets leaving $C$ one with free lanes and push
        forth as many cars as possible on it.
    or
        Select among the streets leading to $C$ one with cars driving on it
        and push back as many of the excess cars as possible.
    end of choice
end
```

Unfortunately, this procedure does not yet lead to success. The reason for this is that it can easily happen that two junctions (or more than two) simply push their excess flow back and forth forever and we never come to an end, as happens with the three junctions in the following six pictures. Computer scientists say: “The algorithm doesn’t *terminate*.”



Thus, we need a good idea for making our search for the best flow more goal-directed. Let's introduce the following additional rule: Each junction is assigned a *height*. At the start all junctions have height 0. Later on we gradually raise the junctions in the following fashion: We stipulate that cars may only be pushed *downwards*. Hence, in order to push forth an excess flow from a junction we first need to raise it to, say, height 1. Then we are allowed to push (forth or back) excess cars only on streets leading to lower junctions. In the beginning we raise S to height 1 and push, like before, as many cars as possible away from S. Afterwards, we raise the next junction with excess flow to height 1, push forth, then raise the next junction, and so on. We never have to raise the goal Z because once the cars have arrived there, they have reached their final destination.

Usually, many cars arrive at Z in this fashion. Nevertheless we might end up with junctions having excess flow but no neighboring junctions at height 0 to get rid of their excess flow. Then we are allowed to raise them even higher. How high? Well, at least by 1. Chances are that this does not yield a new possibility for pushing downwards. In that case we may continue raising the junction. But only far enough to arrive at a height that allows pushing forth (or backwards). Naturally, we raise the junction only in our model. Once we have found our final solution, there is no need to call the construction workers with their diggers to raise real junctions.

The procedure RAISE raises a junction C if it has excess flow but can push it neither forward nor backward.

```
1  procedure RAISE ($C$)
2  precondition Pushing from $C$ not possible.
3  begin
4      Raise $C$ until there is an opportunity to push flow to a lower junction.
5  end
```

We amend the procedure PUSH with the prerequisite that excess flow must only be pushed downwards.

This PUSH procedure has – on top of the previous version – the restriction that flow may only be pushed downwards.

```
1  procedure PUSH ($C$)
2  begin
3      choose one of the following:
4          Select among the streets leaving $C$ one that leads to, say, $N$ and
           that is not full yet. If $C$ is higher than $N$, push excess cars along
           this street; but neither more than fit on the street nor more than
           there is excess.
5      or
6          Select among the streets leading from, say, $N$ to $C$ one with cars
           driving on it. If $C$ is higher than $N$, then push back as many of
           the excess cars as possible; but never more than $C$ has excess.
7      end of choice
       [sometimes neither is possible]
8  end
```

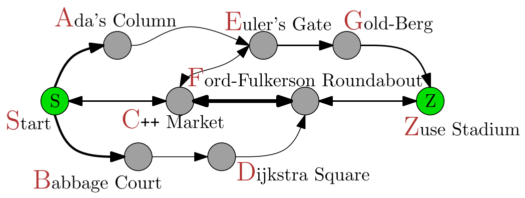
We can now repeat these two procedures (PUSH and RAISE) until there is no junction with excess flow left. We can stop raising S as soon as it reaches height n, where n denotes the number of junctions in our network. After that we merely deal with the excess flow at the remaining junctions, and then we are done. The reason why stopping then is okay will be explained in the section "Why does it work?" below. Note that we could also simply raise S to height n directly at the beginning. In our example S has height 9.

The actual algorithm thus looks like this:

The algorithm MAXIMUM FLOW finds a maximum flow from the start S to the target Z, by repeatedly calling the procedures RAISE and PUSH.

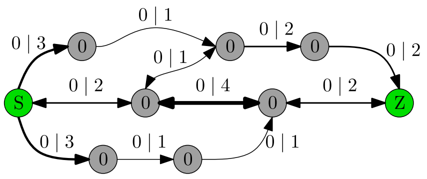
```
1   procedure MAXIMUM FLOW $(G, S, Z)$
2   begin
3       Raise $S$ to height $n$ ($n$ is the number of junctions).
4       For each street leaving $S$, push as many cars as possible away from $S$.
5       Leave all other junctions (except $S$) at height 0.
6       while there is a junction $C$ with excess flow do
7           if pushing from $C$ possible
8               Apply procedure PUSH (at junction $C$).
9           else
10              Apply procedure RAISE (at junction $C$).
11      endwhile
12  end
```

In order to describe a complete run of our algorithm, we first have to give names to the junctions:

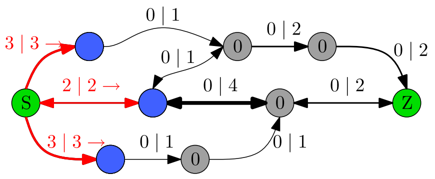


The following pictures show the progress of the algorithm. We annotated each junction with its current height.

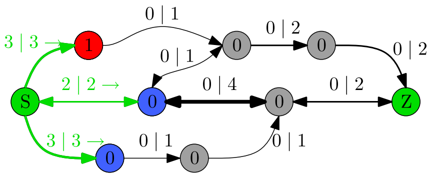
Initial state:



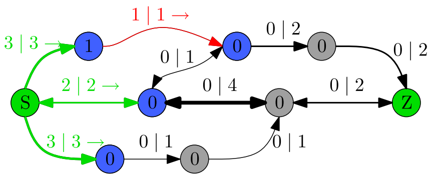
0. Push all cars away from the start:



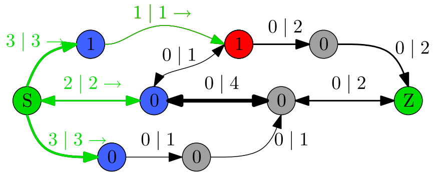
1. RAISE (Ada's Column) to 1:



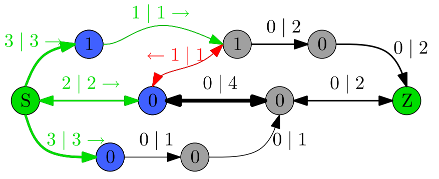
2. PUSH (Ada's Column): one car to Euler's Gate:



3. Raise (Euler's Gate) to 1

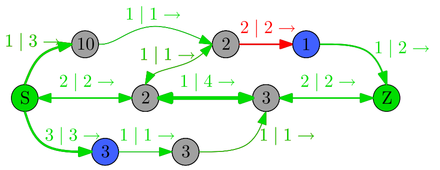


4. Push (E): 1 to C

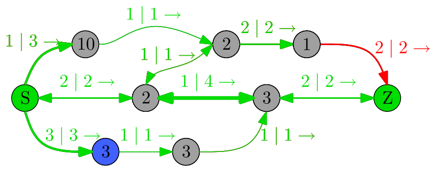


We list some intermediate steps in short only:

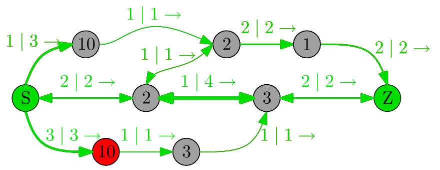
5. Raise (C) to 1
6. Push (C): 3 to F
7. Raise (F) to 1
8. Push (F): 2 to Z
9. Raise (F) to 2
10. Push (F): 1 to C
11. Raise (C) to 2
12. Push (C): 1 to E
13. Push (E): 1 to G
14. Raise (G) to 1
15. Push (G): 1 to Z
16. Raise (B) to 1
17. Push (B): 1 to D
18. Raise (D) to 2
19. Push (D): 1 to B
20. Raise (B) to 3
21. Push (B): 1 to D
22. Raise (D) to 3
23. Push (D): 1 to F
24. Raise (A) to 10
25. Push (A): 2 to S
26. Raise (F) to 3
27. Push (F): 1 to C
28. Push (C): 1 to E
29. Raise (E) to 2
30. Push (E): 1 to G



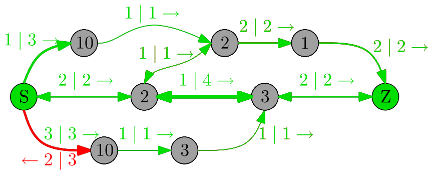
31. Push (G): 1 to Z



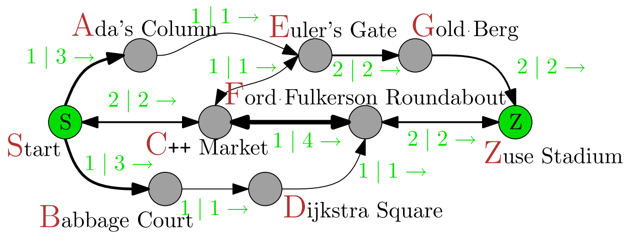
32. Raise (B) to 10



33. Push (B): 2 to S



This is the solution to our problem:



We could now use this solution to route traffic in the real world, avoiding congestions.

Some Open Questions

- The algorithm does not specify which junction to choose each time step 6 is called. Any choice works as long as we obey the rules for pushing ("always downwards") and raising ("only if no more pushing is possible and then only as high as necessary to push again")! Our example needs 33 steps: 15 times Raise and 18 times Push (excluding the initial raising of and pushing from the start). Try coming up with a different sequence of operations for our example. Can you find one that reaches a solution more quickly?
- Have you realized that excess flow may only be pushed back to S, after a junction has been raised higher than n?

Why Does It Work?

As if by magic, the algorithm always works correctly. If you are in the mood, try playing around with different road networks. But if you are interested in why the algorithm works, continue reading:

First of all, we should convince ourselves that the algorithm outputs a valid traffic flow. To this end we must merely realize that:

- We have never pushed more cars along a road than fit on it (capacity rule), and
- no junction ends up with excess flow (flow conservation rule).

Thus the traffic can flow unhindered.

Having tried a few examples, you will notice that there is a crucial difference between junctions which are higher than n, and those that are not. From high junctions traffic is always pushed back towards S. These are the junctions having no chance any more to pass their excess flow on towards Z. But what happens before this?

In the beginning excess flow is only pushed from junctions at height 1 to junctions at height 0. In a way these are the simple cases. Only after all simple options are exhausted, are junctions gradually raised higher and higher. An important observation is that excess flow is always only pushed down one level, i.e., from junction C at height h to a neighboring junction D at height $h-1$. It can never happen that this neighboring junction D has, e.g., height $h-2$. After all, we would then not have been allowed to raise C that high in the first place. Cars that are supposed to arrive at Z must gradually descend from h to $h-1$, then from $h-1$ to $h-2$ and so on, until they arrive at Z, which always remains at height 0. Therefore at least h different stages are visited.

This ensures that excess flow cannot be pushed back and forth perpetually between two junctions. This is due to the fact that there can be at most $n-1$ different stages (not n because we never pass through S). Moreover, this guarantees that really all options to push forth excess flow are exhausted, before pushing back flow to S. If by no means excess flow can be routed to Z, backward routing toward S works by the same principle. Ultimately all excess flow ends up at either Z or S, and we have finally found the best traffic flow.

Epilogue

Some time later Jogi's sister learned how to meticulously prove that this algorithm finds the best traffic flow. In fact, this is quite involved. It turned out, that the order in which junctions are selected for Push and Raise operations does not affect the correctness of the algorithm. She also learned that Andrew Goldberg and Robert Tarjan found this algorithm in 1988. Jogi often still gets stuck in traffic jams on his way to the stadium.

Solution

There is actually a sequence of operations that requires only 19 steps to find the best traffic flow:

1. Raise (A) to 1
2. Push (A): 1 to E
3. Raise (C) to 1
4. Push (C): 1 to E
5. Push (C): 1 to F
6. Raise (B) to 1
7. Push (B): 1 to D
8. Raise (D) to 1
9. Push (D): 1 to F
10. Raise (F) to 1
11. Push (F): 2 to Z
12. Raise (E) to 1
13. Push (E): 2 to G
14. Raise (G) to 1
15. Push (G): 2 to Z
16. Raise (A) to 10
17. Push (A): 2 to S
18. Raise (B) to 10
19. Push (B): 2 to S

Further Reading

1. Chapter 7 (Depth-First Search)
   Many flow algorithms are based on a so-called depth-first search or breadth-first search in the graph. This is also true for the algorithm of Ford–Fulkerson, for instance. In this chapter you can read up on how a depth-first search in a graph works, and how it can be used.

2. Chapter 9 (Cycles in Graphs)
   In rare cases, the algorithm of Goldberg and Tarjan sends a few units of flow around in a circle somewhere in the graph that does not contribute to the actual flow from the start to the target. By means of a cycle search, these circular paths can be removed after the maximum flow has been determined. This chapter explains how this can be done.
3. Chapter 32 (Shortest Paths)
   A problem closely related to maximum flows is the search for a shortest path. Here the goal is not to route a whole flow of cars from the start to the target but to find the quickest route for a single car. You can read up on how these shortest paths can be found in this chapter.
4. The 3D-animation *Flow Commander* at
   http://i11www.iti.uni-karlsruhe.de/adw/jaws/GTVisualizer3D.jnlp (requires Java Web Start).
   Why not look at height as an actual third dimension? Fly through the graph and behold PUSH and RAISE operations taking place in 3D! If Java is installed on your computer (which is most likely the case), you can install and launch *Flow Commander* at this URL.
5. One of the quickest algorithms for maximum flows: Lestor R. Ford, Jr. and D.R. Fulkerson. *Flows in Networks*. Princeton University Press, 1962.
6. The original publication of the presented algorithm: Andrew V. Goldberg and Robert E. Tarjan: *A new approach to the maximum-flow problem*. Journal of the ACM 35:921–940, 1988.
   http://dx.doi.org/10.1145/48014.61051
7. The English Wikipedia article on the algorithm of Goldberg and Tarjan:
   http://en.wikipedia.org/wiki/Push-relabel_algorithm
8. The Wikipedia article on network flow:
   http://en.wikipedia.org/wiki/Flow_network
   Among other things this article explains how flows are interrelated to so-called *cuts*.