Solving 3-SAT and 3-dimensional matching in polynomial time (draft-01)

Frederic Gillet
(frederic.gillet@gmail.com)

October 6, 2013

Abstract:

We show how the implementation of conservative logic gates on flow networks allows us to solve 3SAT and 3-dimensional matching problems in polynomial time by using standard minimum-cost flow methods.

1) Flow Networks and minimum cost flow problem.

A flow network is a directed graph defined by a set of vertices and edges (V,E). The system models a physical system were flows travel along edges and through the nodes.
An edge $e \in E$ is characterized by different quantities:

- $u(e)$, the edge capacity, i.e. the maximum flow allowed along edge e. A positive integer quantity.
- $c(e)$, a cost per unit of flow along edge e.
- $l(e)$, a lower bound on the flow along edge e, a positive integer quantity.
  We have $0 \leq l(e) \leq u(e)$.

The flow along edge e is $f(e)$ with $0 \leq f(e) \leq u(e)$.

The set of edges leaving vertex v is $E^+(v)$ and the set of edges arriving at vertex v is $E^-(v)$.
A vertex $v \in V$ is characterized by the quantity $b(v)$ representing the flow injected ($b(v) > 0$) or absorbed ($b(v) < 0$) at the vertex. If $b(v) = 0$ then the total flow through the vertex is conserved (the flow entering the vertex is equal to the total flow leaving the vertex).

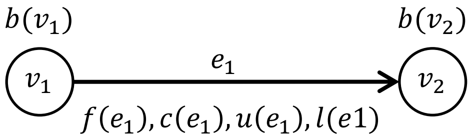


Figure 1: two vertices and an edge e_1 with realized flow $f(e_1)$

A network with multiple demand vertices and multiple supply vertices can always be transformed without loss of generality into a flow network with a single source node and a single sink node.

The minimum flow problem consists in finding a set of valid flows in the network which has the lowest cost and respect the various constraints on all the vertices and edges:

- Minimize $\sum_{e \in E} f(e)c(e)$
- $b(v) = \sum_{e \in E^+(v)} f(e) - \sum_{e \in E^-(v)} f(e)$ for all $v \in V$
- $l(e) \leq f(e) \leq u(e)$ for all $e \in E$

Note that a solution does not necessarily exist.

Several algorithms exist that can solve minimum cost flows efficiently (polynomial both in time and in space), like the "minimum-mean cycle-canceling" algorithm [ref 1, 2].

Figure 2 shows an example flow network and a possible minimum cost flow solution.

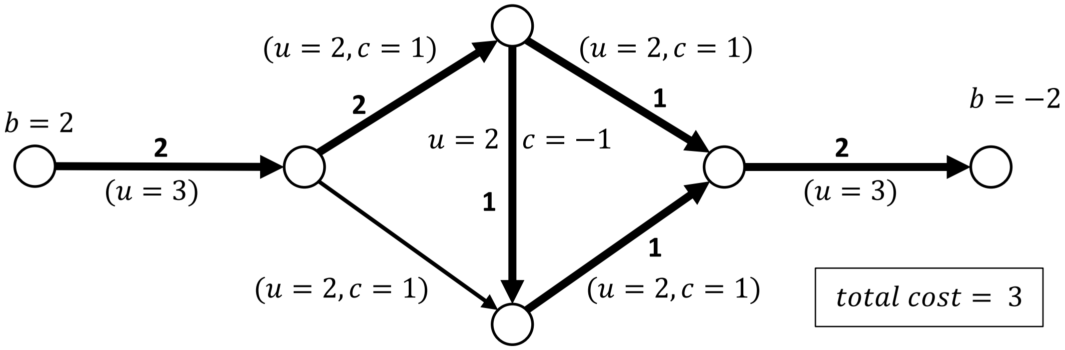


Figure 2: a flow that respects all the constraints and has minimum cost.

2) Conservative logical gates.

In the paper "Conservative Logic" [ref 3, 4], Fredkin and Toffoli showed how it's possible to build logical gates that are better suited to represent actual physical systems. Those gates are conservative (some physical quantity that is flowing through the gate is entering and leaving in equal amount) and reversible (at the microscopic level, physical systems are reversible).

To illustrate the idea of conservation and physical system implementation, Fredkin and Toffoli used a billiard ball model. Figure 3 shows such model for an AND gate.

a	b	$a\ AND\ b$
0	0	0
0	1	0
1	0	0
1	1	1

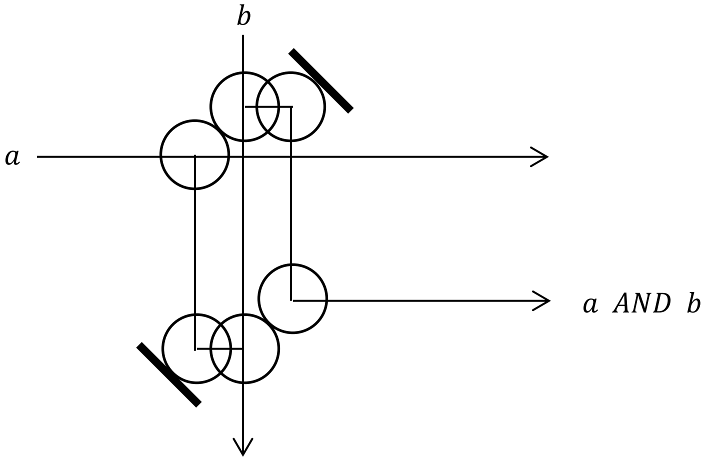


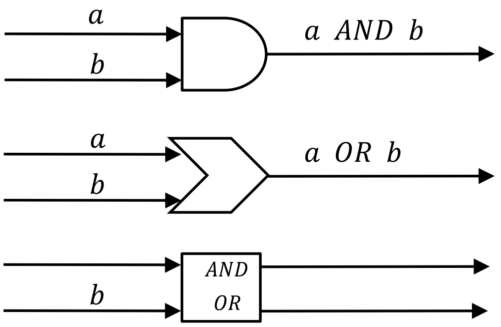
Figure 3: Fredkin and Toffoli’s realization of a conservative AND gate using billiard ball physics.

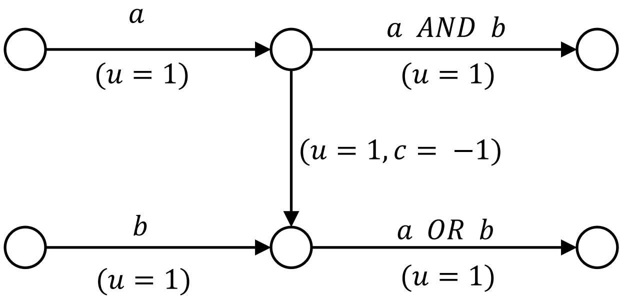
3) Conservative Logic Gates on Flow Networks.

In this section we show that it’s possible to implement various logical gates on flow networks (as defined in section 1).

The first logic gates we consider are the AND and OR gates.

a	b	$a\ ANDb$	$a\ OR\ b$
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1





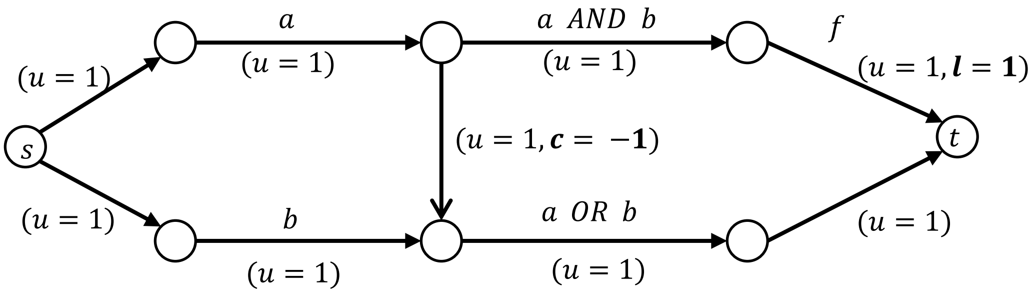
In the flow network we labeled certain edges with the name of a Boolean variable (a, b, a AND b). The value of the corresponding variable is the realized flow on the edge. Those edges have a capacity of 1 unit, so their flow can only be 0 or 1.

To illustrate how the gate works in practice, we use an electronic circuit analogy. A unit flow is similar to an electric current of one unit. We use a vertex as a power source for the various input flows/currents. We also use a drain vertex to absorb the unused flows/currents and achieve overall flow/current conservation.

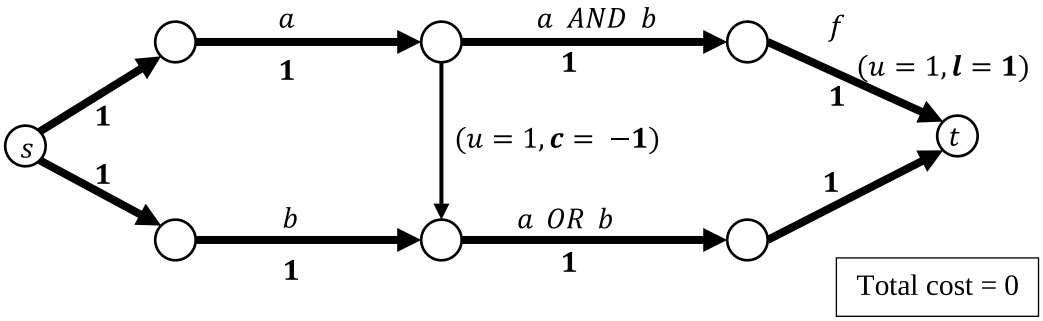
We show that we can create a flow network such that its minimum cost flow solution represent the state of the input variables that satisfy the imposed value of the output function.

In the first example we implement the logical function$f = a\ AND\ b$, with the imposed output value $f = 1$. This condition is realized by imposing a minimum flow value $l(f) = 1$ on the output edge.

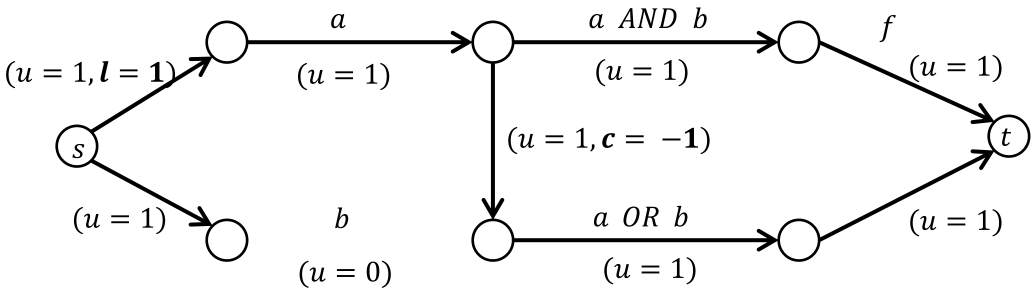
We use a source node with two unit output capacities to possibly feed the two input edges a and b.



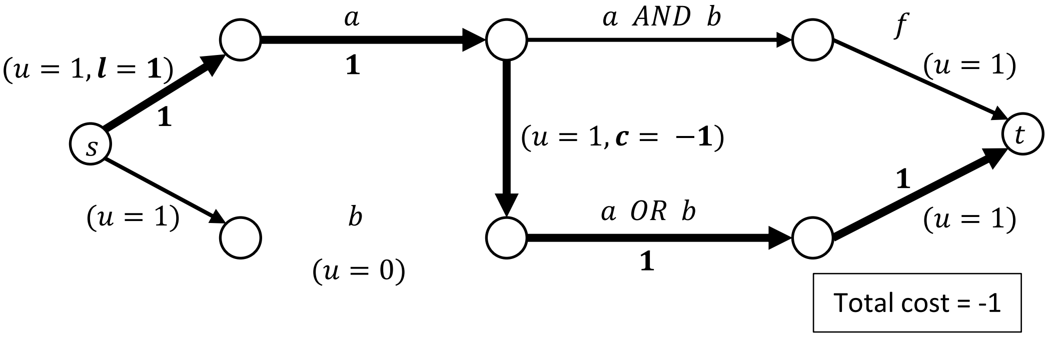
After solving for minimum cost flow, the only feasible set of flows is



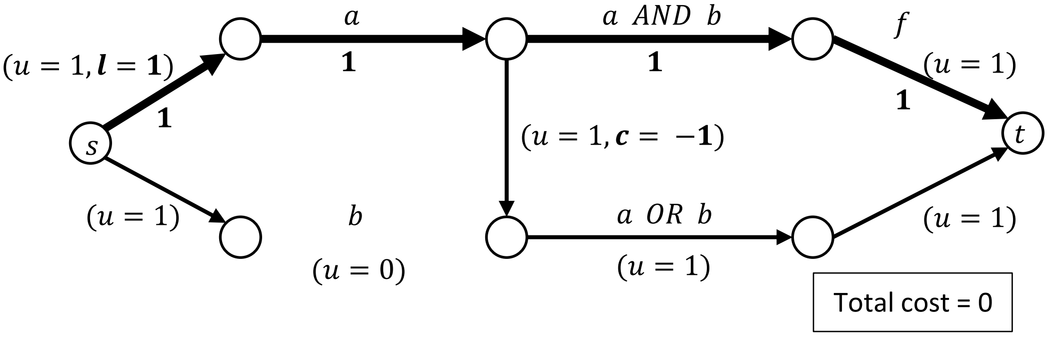
As a second example, we force the variable a at 1 by setting the corresponding edge minimum flow condition $l(a) = 1$. We also set the variable b to 0 by setting a null capacity on the b edge with $u(b) = 0$ (equivalent to severing edge b). Note that we removed any minimum condition on the output edge f with $l(f) = 0$.



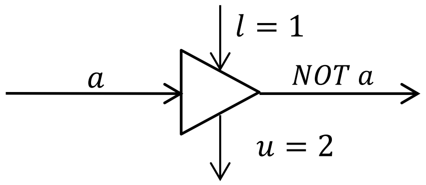
Solving for minimum cost flow shows that the output value is f=0.



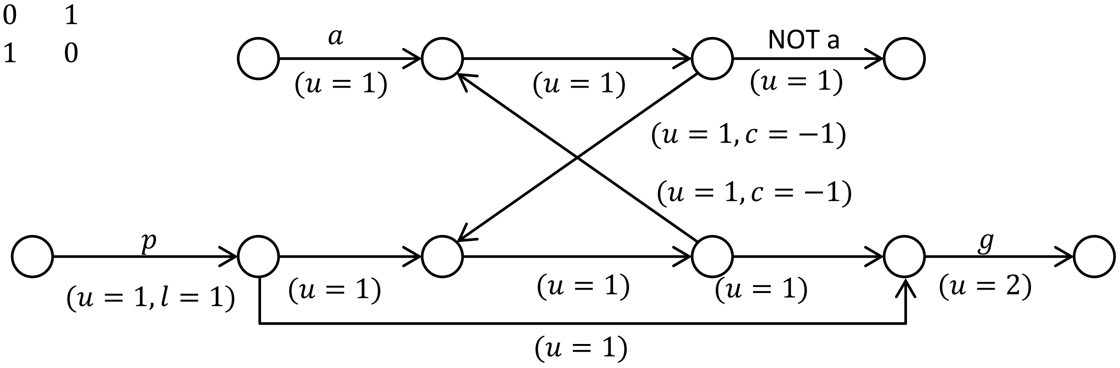
Note that the total cost of the realized flow is -1 thanks to the negative cost of the center edge. It is that negative cost that prevents the flow through edge a to avoid edge f. The following flow is not a valid solution because its total cost is superior:



The next gate we realize is the NOT function.

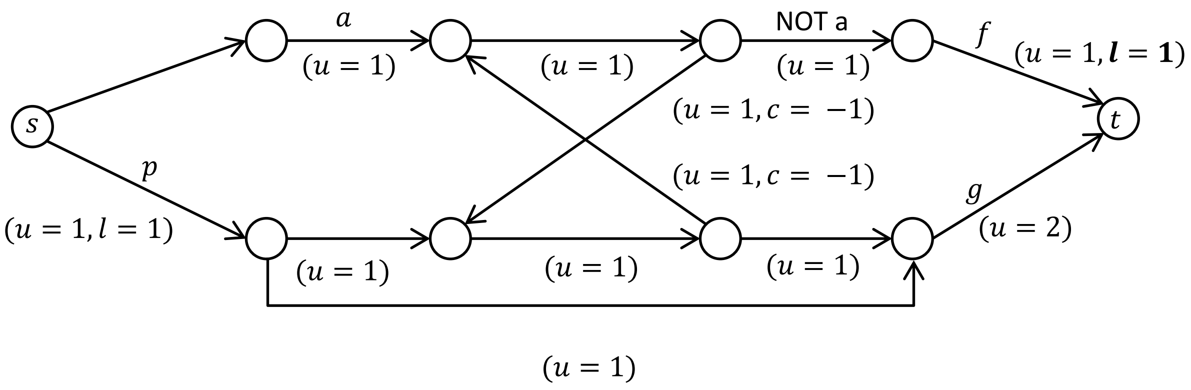


a	$NOT\ a$
0	1
1	0

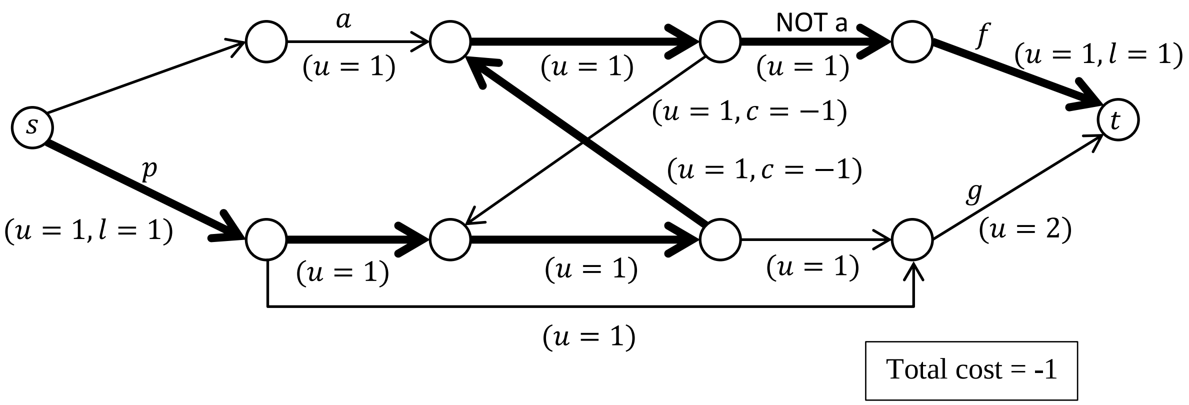


This function needs an edge providing a constant unit flow supply (p for power) since an output flow needs to be generated when the input is at zero. There is also an edge (g for ground) to absorb any unused flows.

An example flow network realizing the function $f(a) = NOT\ a = 1$ (the lower bound on edge f is set to one unit with $l(f) = 1$)

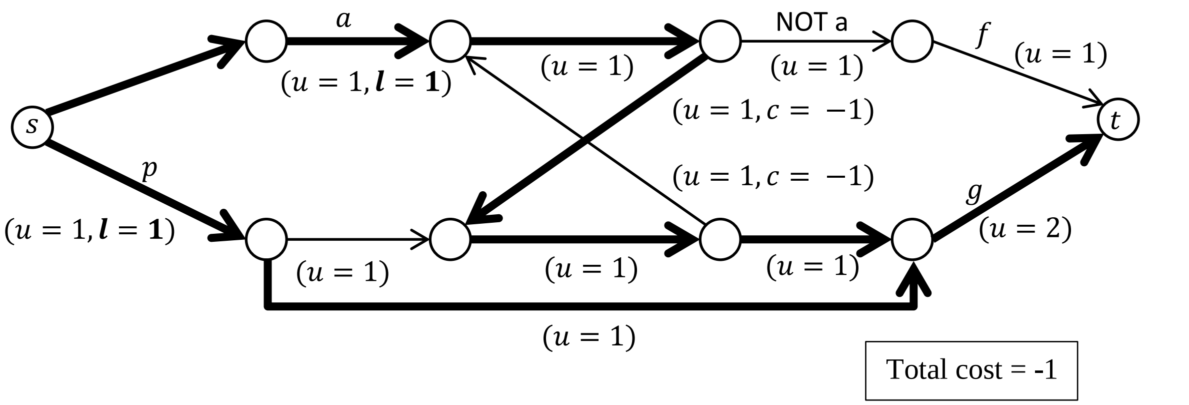


After solving for minimum cost flow, the only feasible set of flows is



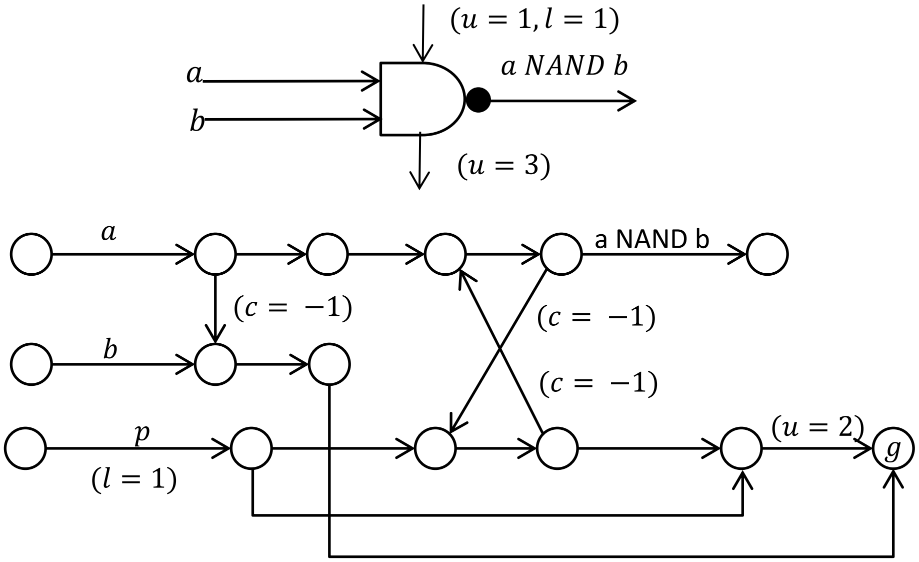
The zero flow on a is the only feasible flow with minimum total cost of -1.

Similarly, forcing the flow f(a)=1 gives a zero flow on the output function f(f)=0:

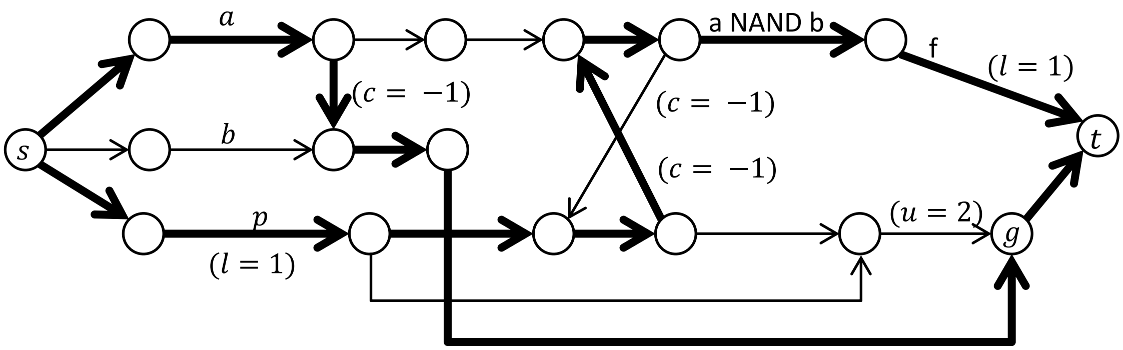


By putting an AND gate and a NOT gate in series we can build a NAND gate. NAND gates allow us to synthesize arbitrary Boolean functions.

a	b	$a\ NAND\ b$
0	0	1
0	1	1
1	0	1
1	1	0

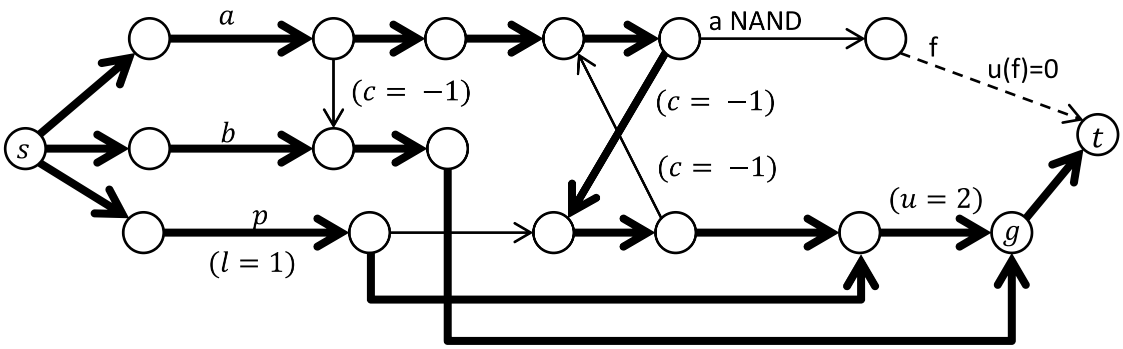


The flow network for the function f(a,b) = a NAND b == 1 is:

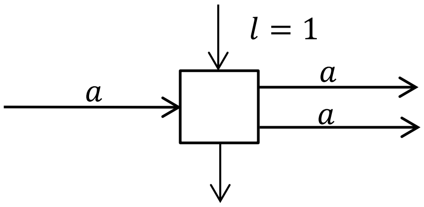


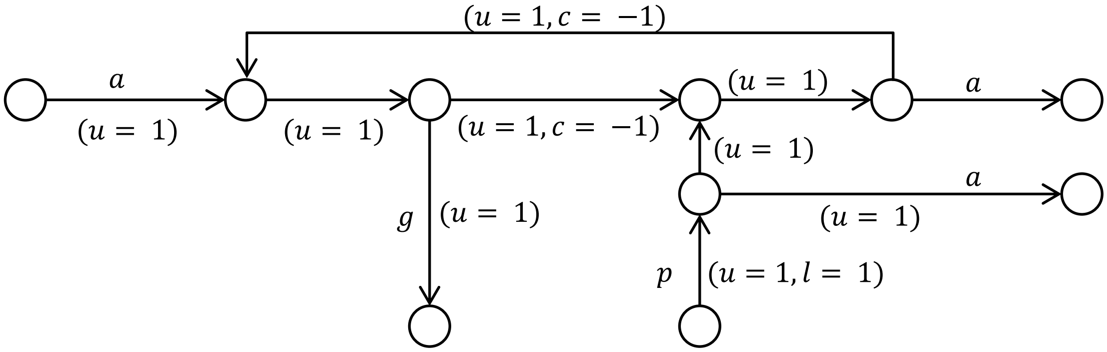
Among the three feasible solutions that respect the constraints: (a=0, b=0, total cost = -1), (a=1,b=0, total cost = -2), (a=0,b=1, total cost = -1), the one with the lowest total cost is selected.

To find the input values such that f(a,b)=0, we set the capacity of edge f to 0 (u(f)=0), which gives the unique feasible solution (a=1, b=1, total cost = -2).

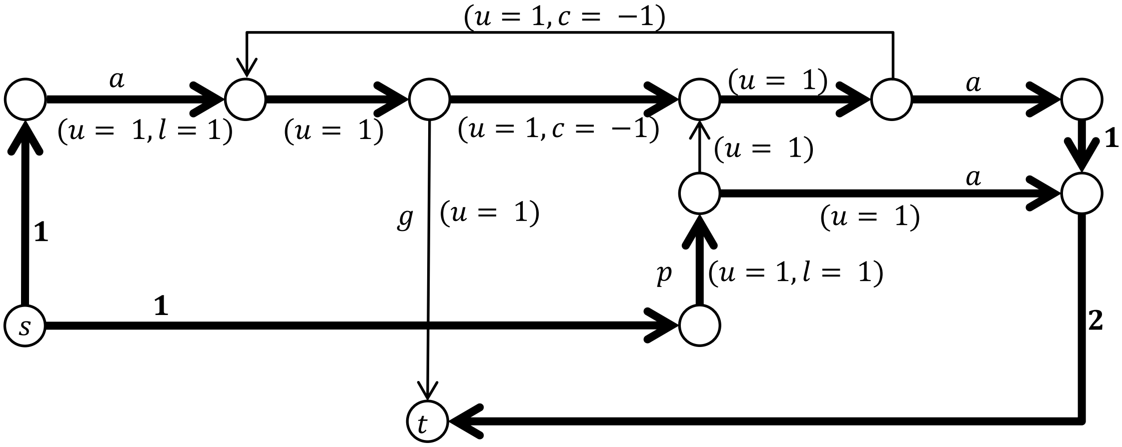


Another useful circuit element is a fan-out gate, allowing duplication of a given flow. The circuit is the same as for the NOT gate, but uses different output paths. The signal requires a “power” flow since the total output flow is doubled when the input flow is at one:

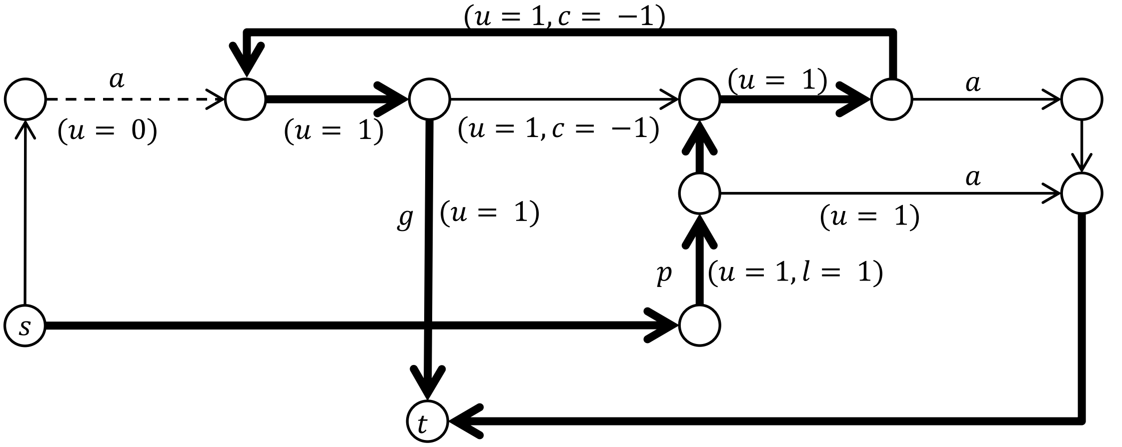




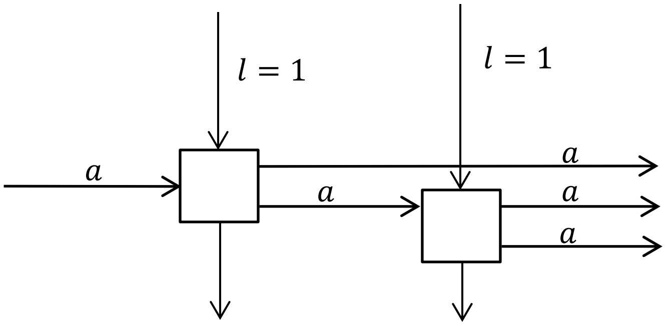
When the input flow a = 1 with l$(a) = 1$:



When the input flow a = 0 with u(a)=0:

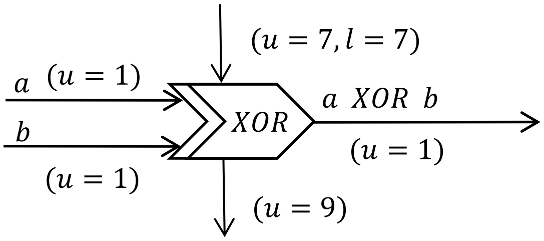


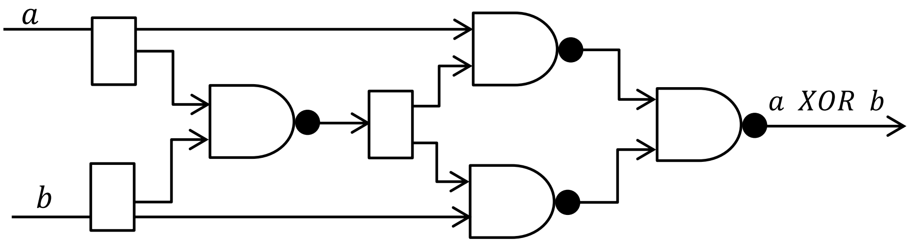
When extra duplicates are needed, multiple fan-out gates can be cascaded:



The last classical logic gate we implement is XOR (exclusive OR), which can be realized using NAND and fan-out gates.

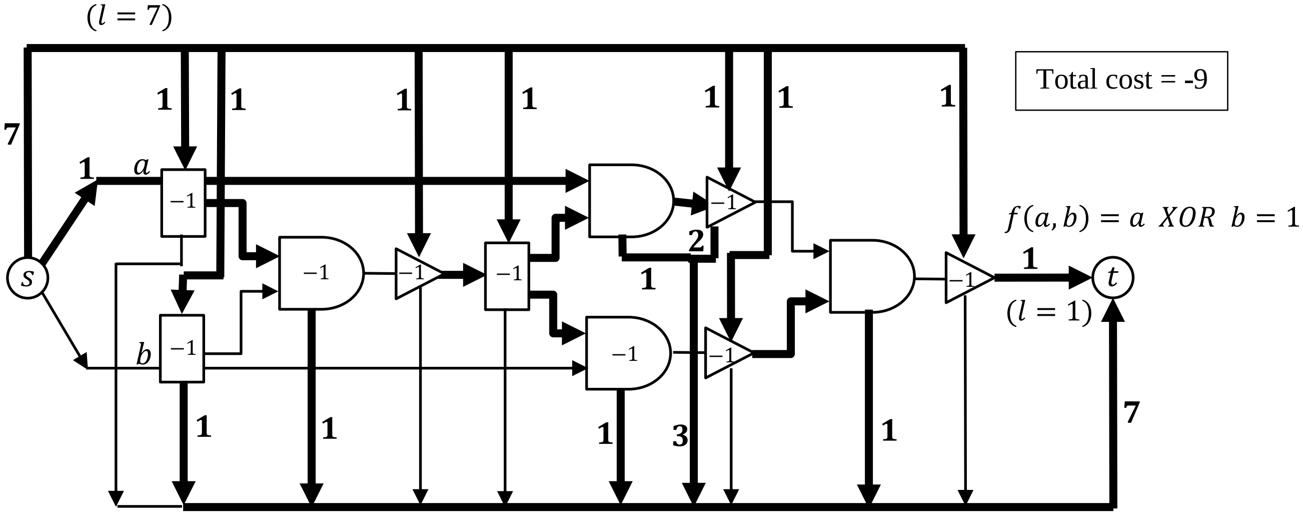
a	b	$a\,XOR\,b$
0	0	0
0	1	1
1	0	1
1	1	0





A flow network realizing XOR, with output forced to 1 (l(f)=1). Note also the condition forcing all the “power” flows to 1. Also not the conservation of flows (total output flow at source vertex s = total input flow at target vertex t = 8).

The flow costs through each gate are indicated in the gate symbols. The total cost of the flow solution is -9.



In the next sections we explore the applications of “Flow Logic Gates” to NP-Complete problems.

4) 3-Dimensional-Matching problem.

The maximum flow problem on a flow network is about finding the maximum feasible flow from a single source vertex to a single target vertex. Over the years many efficient (polynomial) solutions to this problem were discovered (Edmonds-Karp algorithm, push-relabel algorithm, etc). [ref 5]

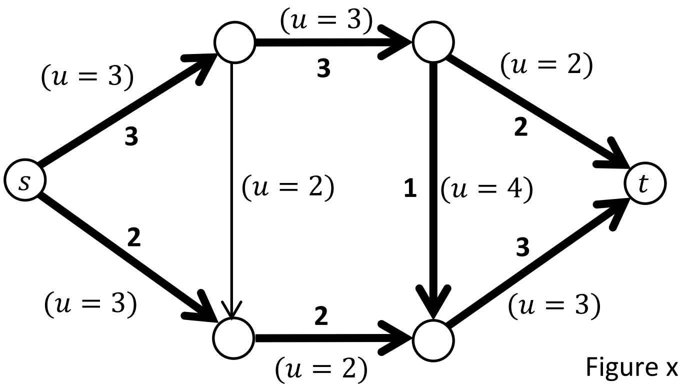


Figure xx: a maximum flow between source and target

The maximum flow algorithm can be applied to solve the Maximum Bipartite Matching Problem. A bipartite graph consists in two sets of vertices X and Y (of same dimension) connected by a collection E of directed edges (unit capacity), without any cycle. An associated decision problem consists in answering the question as to whether there exists or not a subset of E (Es) that connects all the vertices without any overlap.

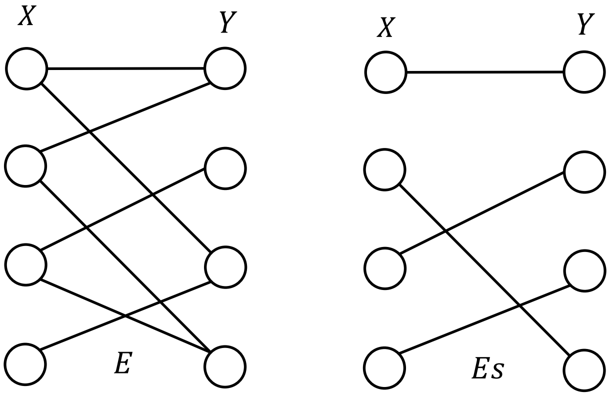


Figure xx: a bipartite graph and a maximum matching subset solution

Without knowing any better it would appear that all the different possible subsets have to be tested to find a solution, which would involve an exhaustive search that grows exponentially with the size of the problem. But a valid solution can be found very efficiently (in polynomial time) by transforming the problem into a maximum flow problem.

We augment the bipartite graph by adding a source vertex (s) and a target vertex (t), and solving for maximum flow (all capacities are unit).

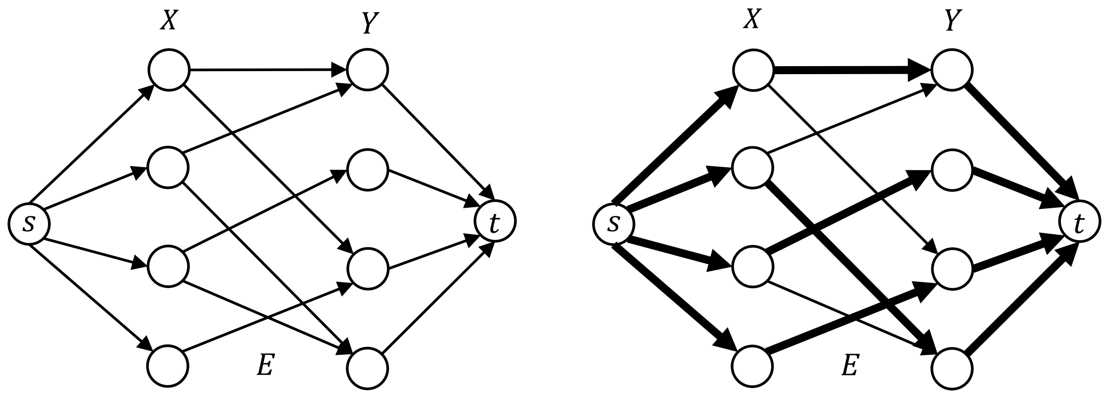


Figure xx: a flow network and its maximum flow.

This transformation allows us to show that the bipartite matching problem can be solved efficiently. One cannot fully appreciate the elegant nature of this solution without doing it once by hand using for example the classical Ford-Fulkerson algorithm (the way one single extra augmented path can exclude multiple edges at once is almost magical).

The 3-dimensional-matching problem is similar to the bipartite matching problem, but it involves an extra set of vertices (X, Y, Z), and instead of edges we are considering a set of triplets E (the elements). We have N triplet elements and M dimension points. We denotes all the triplet elements as $[e_1, e_2, e_3, \dots e_N] \in E$and the different dimensional nodes as $[x_1, x_2, x_3, \dots x_M] \in X, [y_1, y_2, y_3, \dots y_M] \in Y, [z_1, z_2, z_3, \dots z_M] \in Z$. The question we try to answer is “is there a subset of E that covers all the dimensional nodes without any overlap?” Unlike the bipartite matching problem, there is no known efficient method to answer that question.

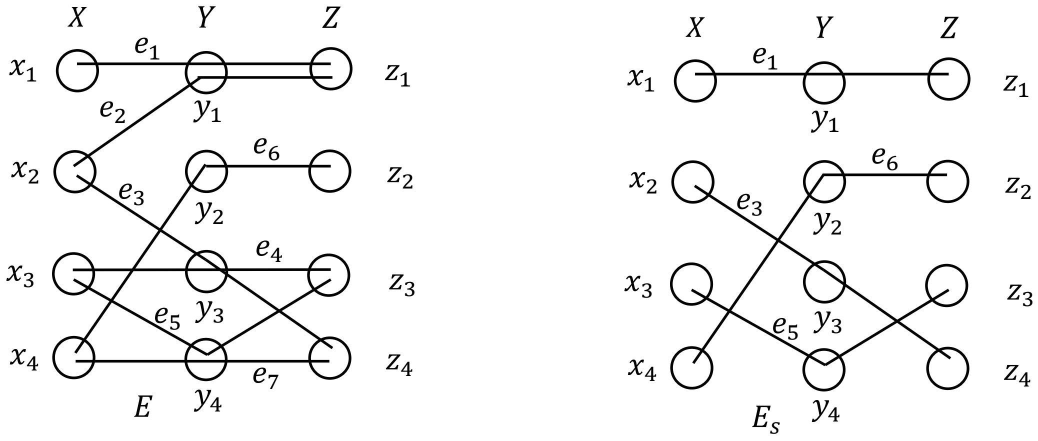
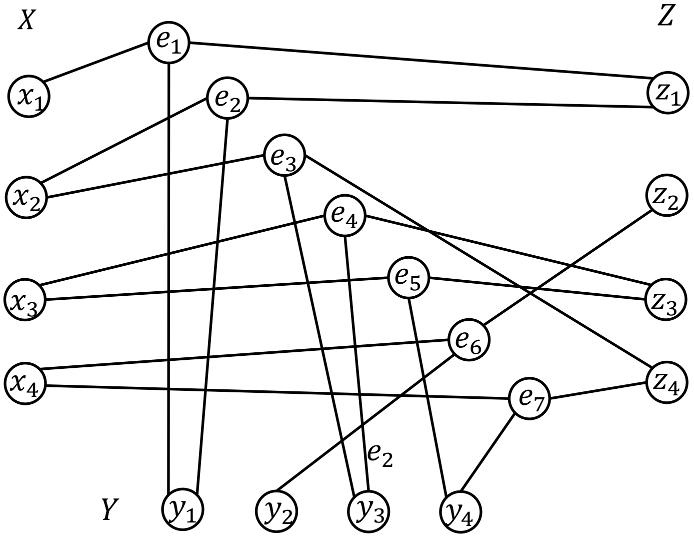
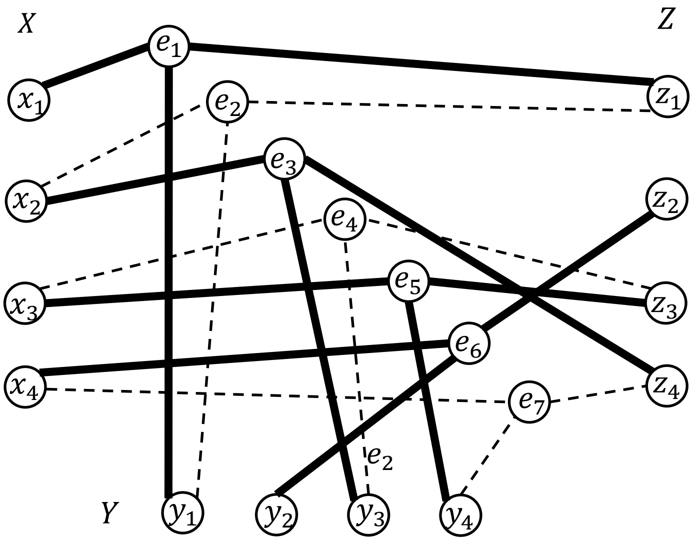


Figure xx: a 3-dim matching problem and a valid solution Es

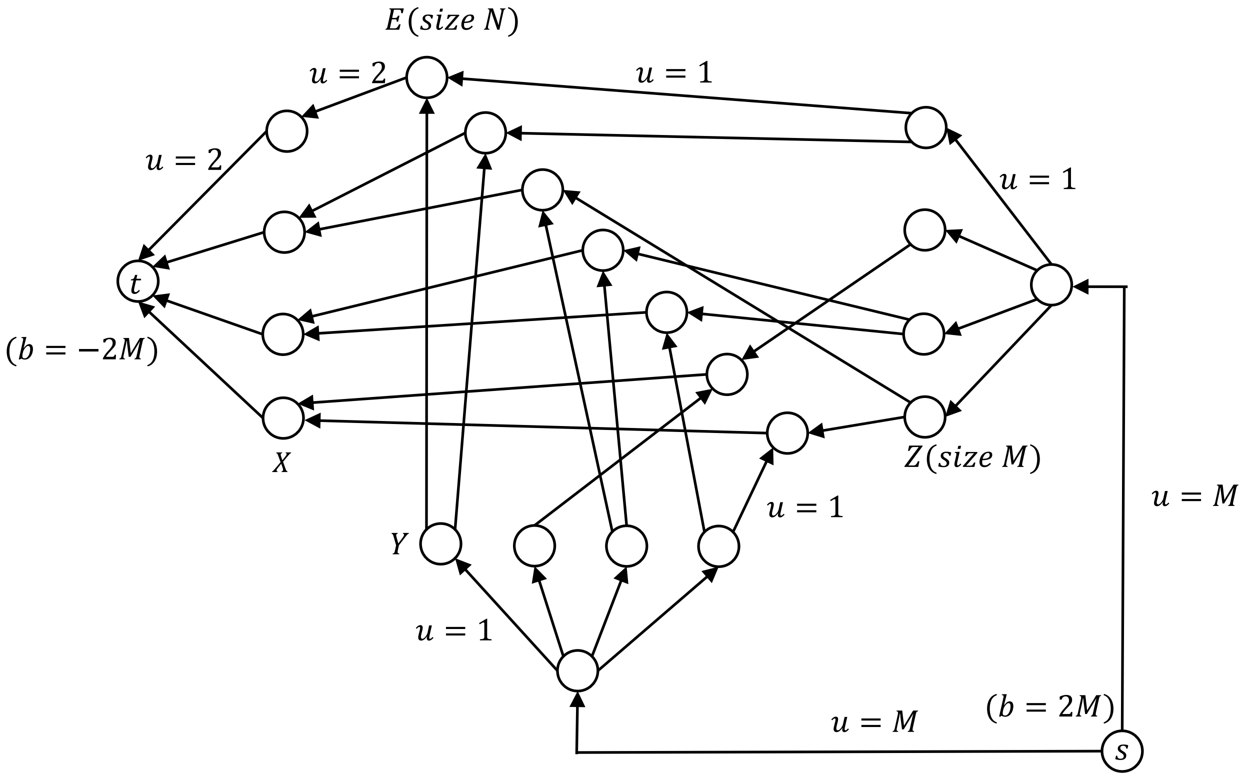
We first apply a visual transformation to the representation by representing the triplet elements and the dimension nodes as vertices in a flow network.



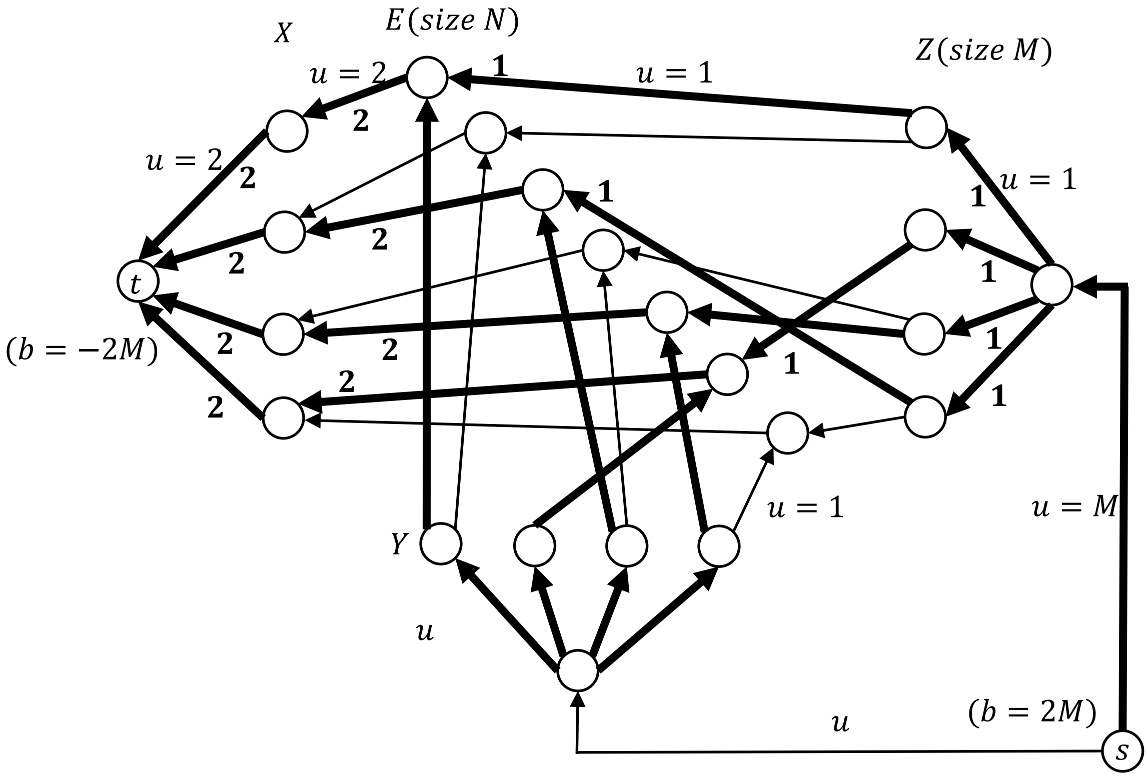


Given this new representation of the problem, it's tempting to map it to a flow network with a source vertex connected to X and a target vertex connected to Y and Z.

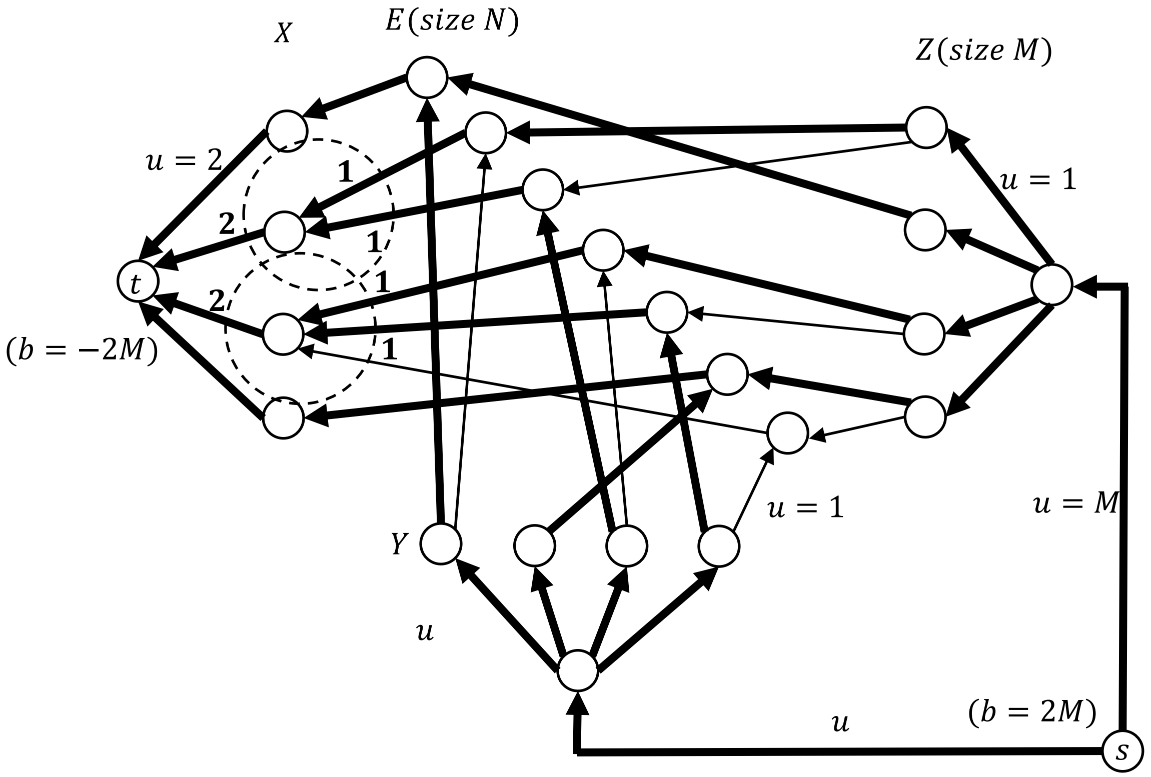
The source tries to inject M unit flows into the Z nodes and M unit flows into the Y nodes (total = -2M), and the target tries to consume 2M unit flows from the X nodes.



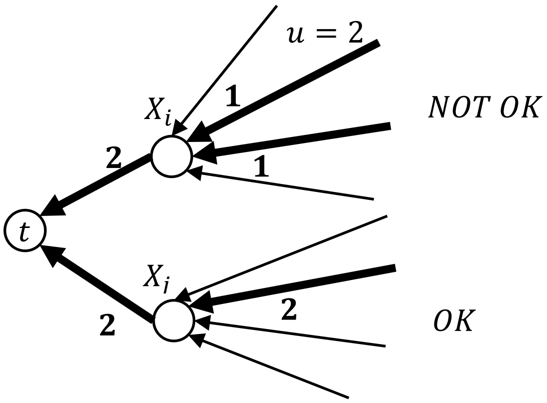
By applying a maximum flow algorithm on this network, we hope that if the maximum flow is 2M this would tell us that the corresponding 3-dimensional matching problem has a solution. Unfortunately this is not the case – the maximum flow could indeed return us the correct answer:



But we would most likely select a lot of invalid solutions where two unit flows arrive on some x elements rather than a unique flow of value 2:

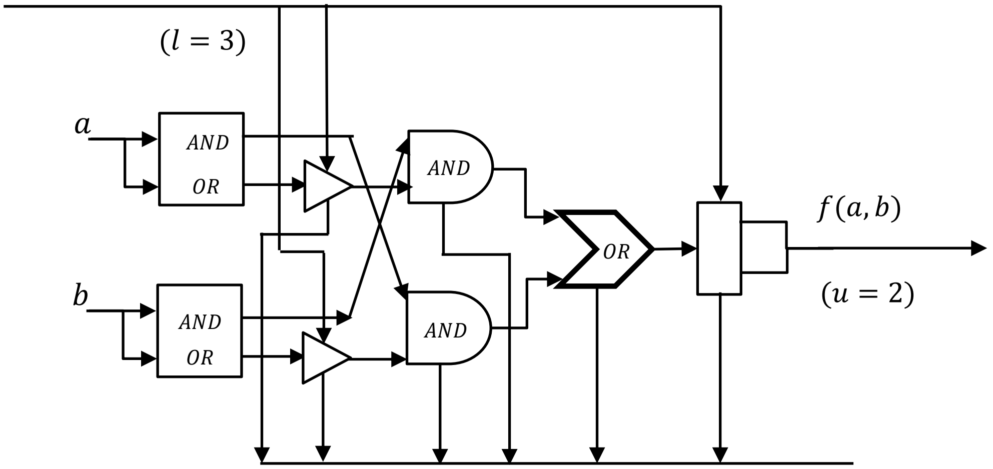


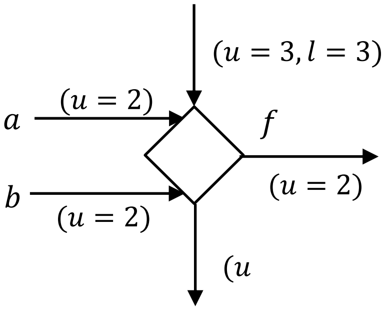
As it is the method is solving a bipartite matching on X/Y and a bipartite matching on X/Z and claiming that the 3-dim matching solution is the union of the two. What is required is some way to impose that the incoming flow on an node x has to be unique instead of a pair of unit flows:



To overcome this issue we synthesize a new flow gate using the logic gates created in the previous section (AND, OR, fan-out, NOT). Note that this new gate is built to handle input flows of values 0,1,2 (so it's not strictly a logic gate).

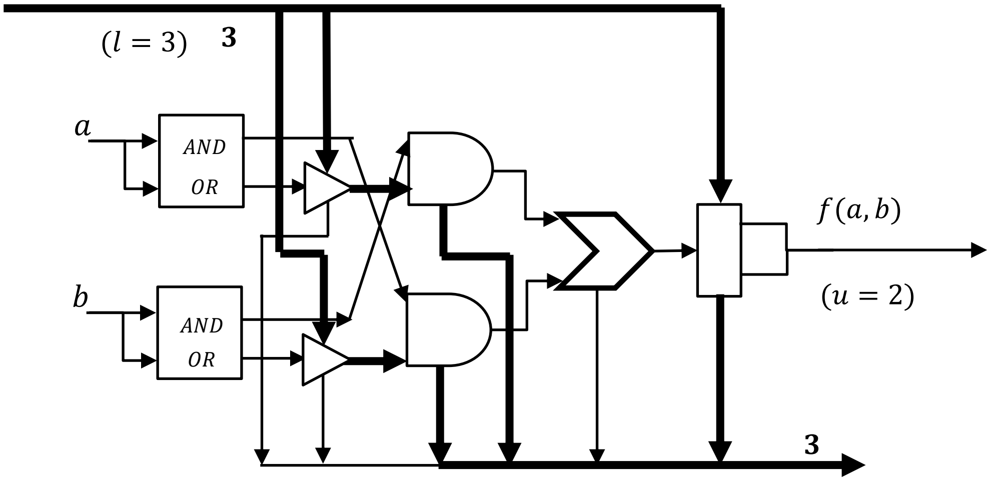
a	b	f
0	0	0
0	1	0
0	2	2
1	0	0
1	1	0
1	2	0
2	0	2
2	1	0
2	2	0



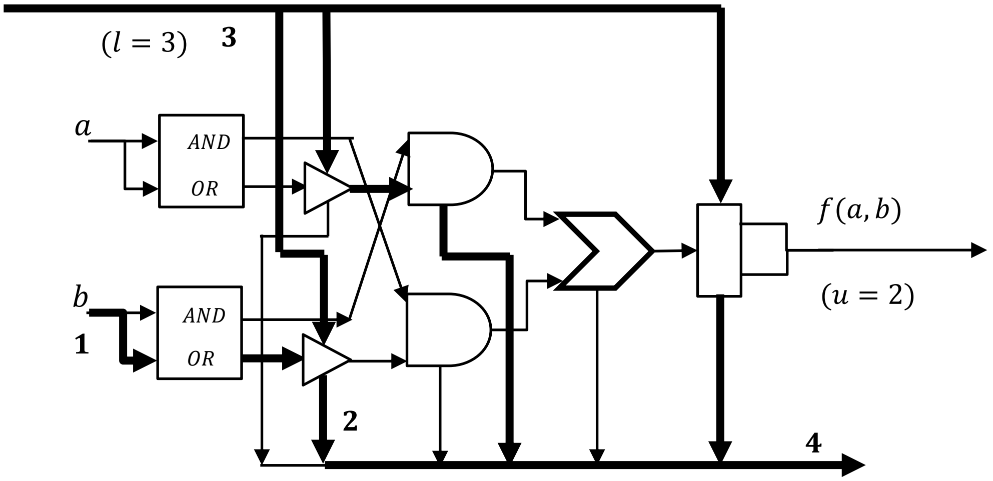


The different possible states of the gate:

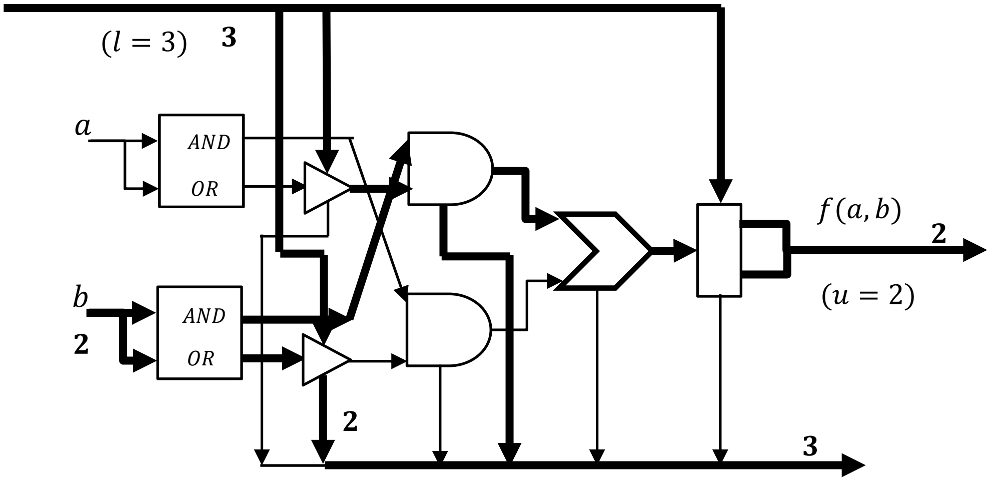
a	b	f
0	**0**	**0**
0	1	0
0	2	2
1	0	0
1	1	0
1	2	0
2	0	2
2	1	0
2	2	0



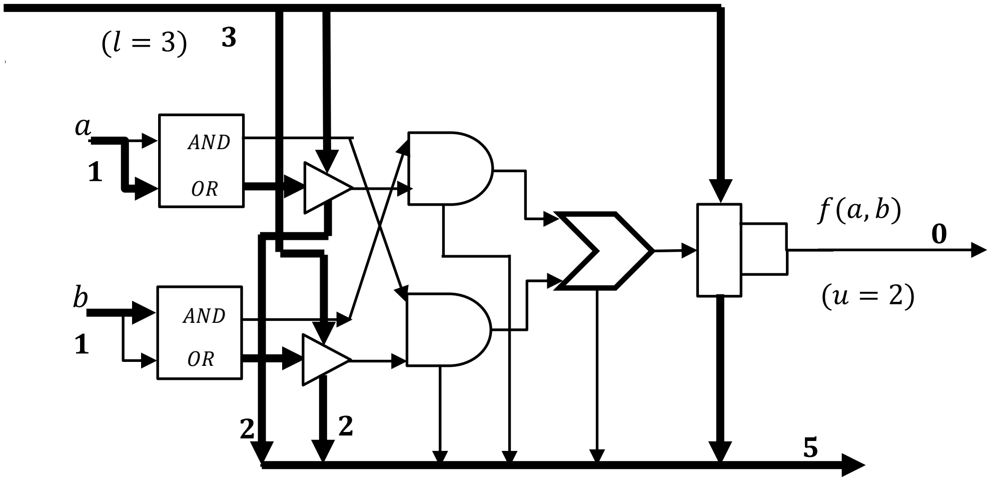
a	b	f
0	0	0
0	**1**	**0**
0	2	2
1	0	0
1	1	0
1	2	0
2	0	2
2	1	0
2	2	0



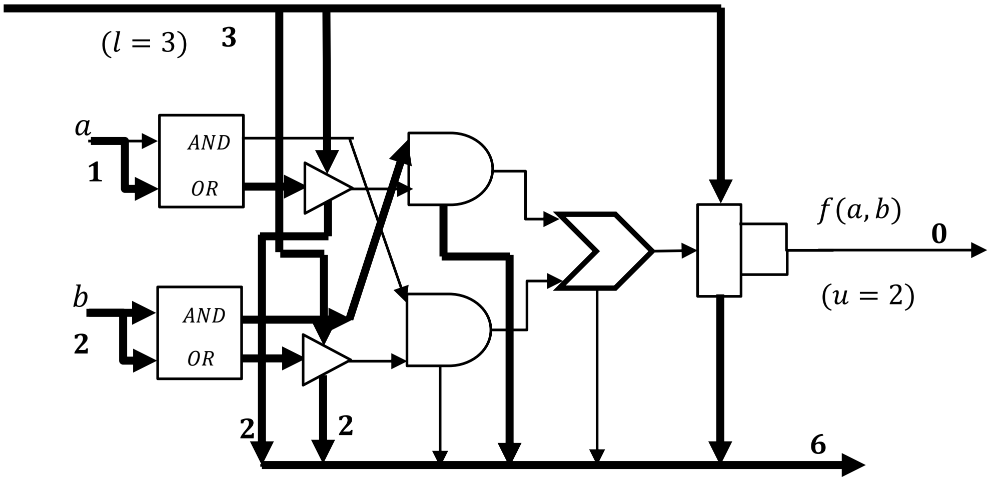
a	b	f
0	0	0
0	1	0
0	**2**	**2**
1	0	0
1	1	0
1	2	0
2	0	2
2	1	0
2	2	0



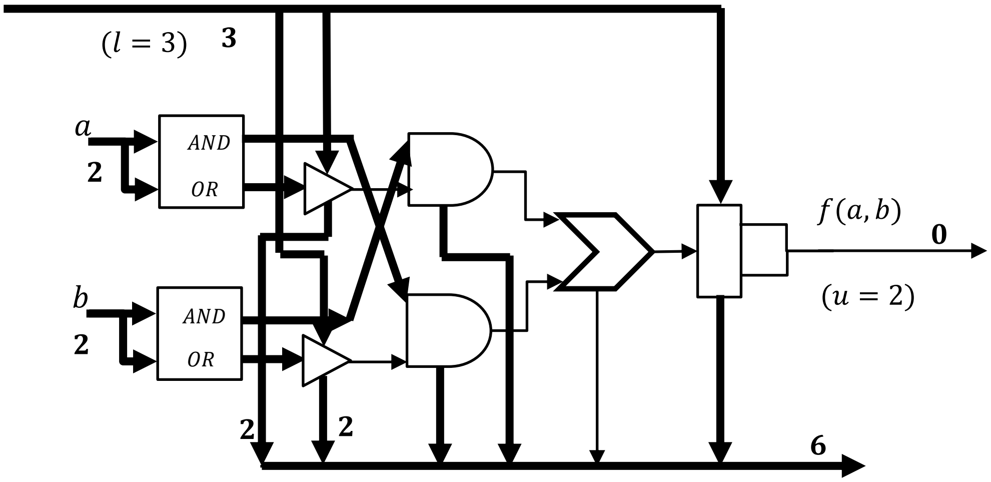
a	b	f
0	0	0
0	1	0
0	2	2
1	0	0
1	**1**	**0**
1	2	0
2	0	2
2	1	0
2	2	0



a	b	f
0	0	0
0	1	0
0	2	2
1	0	0
1	1	0
1	**2**	**0**
2	0	2
2	1	0
2	2	0

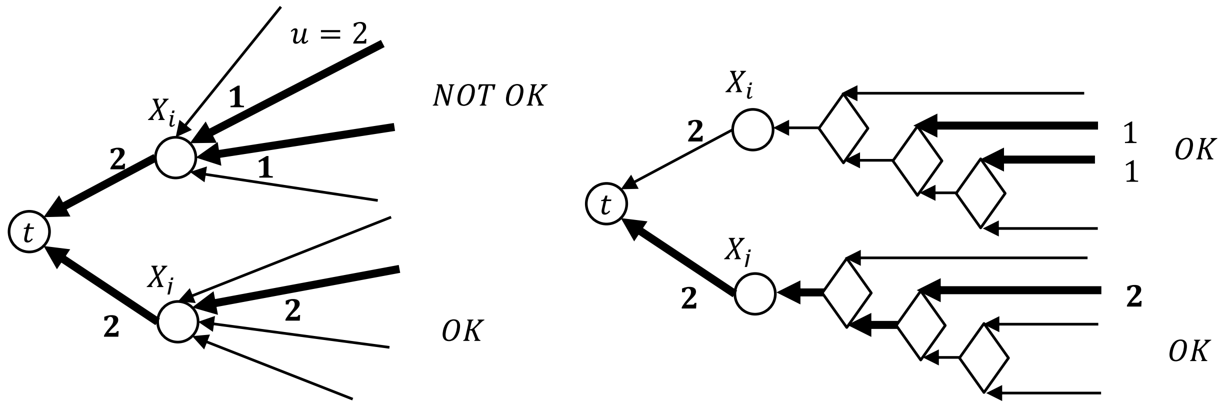


a	b	f
0	0	0
0	1	0
0	2	2
1	0	0
1	1	0
1	2	0
2	0	2
2	1	0
2	**2**	**0**

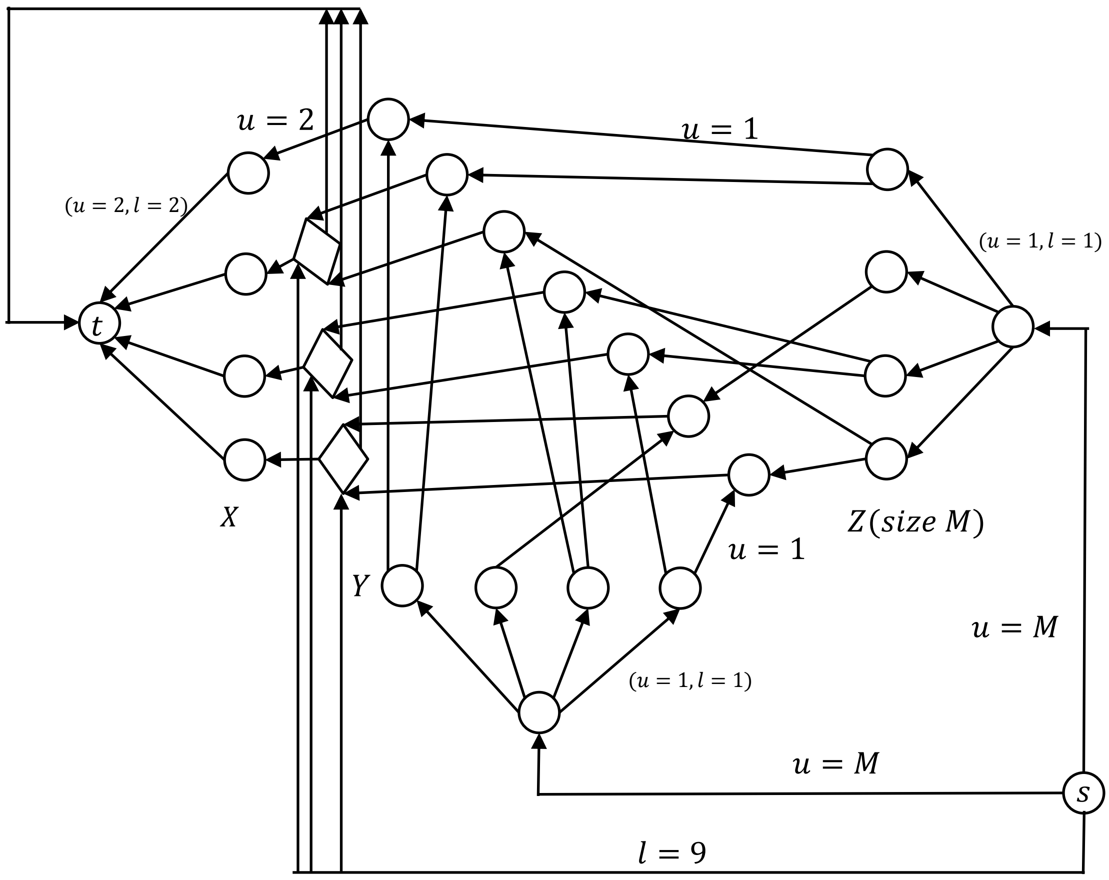


The original flow network is modified with instances of the new gate. Multiple gates can be used in cascade to constraint multiple incoming edges.

The constraints imposed on the flow network are a minimum flow of 2 on each outbound edge of every x node, a minimum flow of 1 on each inbound edge of every y and z node, and a power flow of 3 units for each instance of the diamond gate.



$u = 2$
X_i
1
2
1
t
$NOT\ OK$
X_i
2
2
OK
X_i
2
t
1
1
OK
X_i
2
2
OK



$u = 2$
$u = 1$
$(u = 2, l = 2)$
$(u = 1, l = 1)$
t
X
Y
$Z(size\ M)$
$u = 1$
$u = M$
$(u = 1, l = 1)$
$u = M$
s
$l = 9$

With the new gates it becomes possible to solve the problem using minimum cost optimization on the flow network and solve the underlying 3-dimensional matching problem efficiently.

On average the solution requires the addition of one diamond gate per element. So, the size of the minimum cost problem differs from the size of the underlying 3-dimension matching problem by a constant factor.

5) Application to 3-SAT problem.

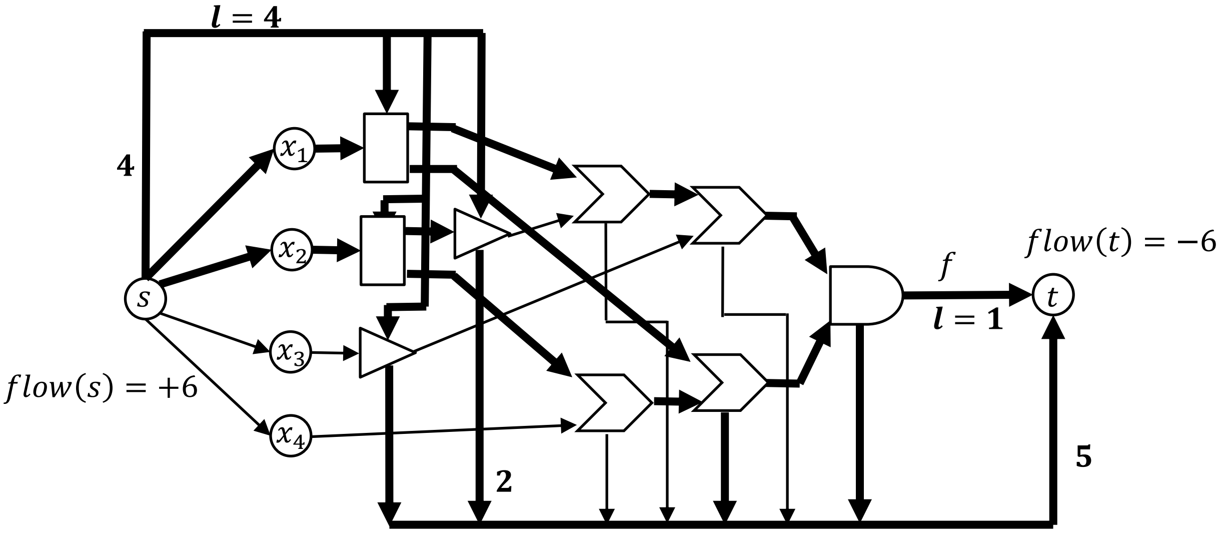
The NP-complete 3-SAT problem consists in a set of input Boolean variable $\{x_1, x_2, x_3, x_4, \dots x_N\}$.
The problem defines a conjunctive normal form with multiple (M) clauses. [ref 5]

An example consists in four variables $\{x_1, x_2, x_3, x_4\}$ and two clauses f= $(x_1 + \overline{x2} + \overline{x3}).(x1 + x2 + x4)$.

The problem consists in answering the question *“is there a set of value of the input variables that sets f as true?”*

In our example the answer is positive with the set of values $\{x_1 = 1, x_2 = 1, x_3 = 1, x_4 = 1\}$

Thanks to our defined “flow logic gates” we can easily transform any 3-SAT problem into a flow network and solve it by computing its minimum cost flow (which can be done efficiently). If multiple solution sets are feasible, the one with the lowest flow cost will be realized.
For our example, we set a constraint $l(p) = 4$ on the power edge and a constraint $l(f) = 1$ on the output link.
Solving for minimum cost flows reveals the possible solution $\{x_1 = 1, x_2 = 1, x_3 = 0, x_4 = 0\}$

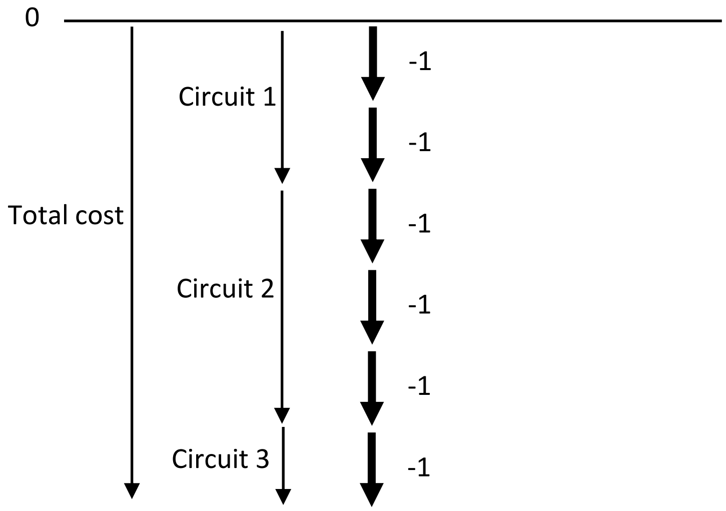


6) Validity and Conclusion.

The proposed approach relies on the hypothesis that flow networks with costs can be used to model logical circuits (Turing Complete).

The work lacks a theoretical demonstration of this assumption. One could try to show that arbitrary logic blocks can be put in series and parallel so that the various costs of the realized paths do not interfere with each other.

On interesting point is that all the costs used in the circuits are either 0 or -1. If there was a mix of positive and negative costs, a positive cost in a circuit could cancel out a negative cost in another circuit (since the cost optimization is done on the whole network). But with strictly negative costs the global cost function can only decrease and costs from various circuits are somewhat distinct:



A practical approach consists in implementing the proposed method and show that it simply works (currently a work in progress). There are plenty of algorithms to compute minimum cost flow efficiently, but even a basic inefficient negative-cycle cancelation algorithm would be sufficient (the difficulty is in doing all the correct transformations to generate a correct flow network).

If correct the approach would seem to suggest that P=NP.

6) References:

[ref 1] Andrew V. Goldberg and Robert E. Tarjan (1990). "Finding minimum-cost circulations by successive approximation".

[ref 2] Jean Bertrand Gauthier, Jacques Desrosiers and Marco E. Lubbecke. "About the minimum mean cycle-canceling algorithm"

[ref 3] Fredkin, Edward; Toffoli, Tommaso (1982), "Conservative logic", International Journal of Theoretical Physics 21 (3-4): 219–253

[ref 4] Richard P. Feynman, “Feynman Lectures on Computation” Addison Wesley, isbn 0-201-48991-0

[ref 5] Michael R. Garey and David S. Johnson, “Computers and intractability – A Guide to the Theory of NP-Completeness”.