# Dependency Graphs, a Multidisciplinary Problem Solving Technique

Omar Khaled

May 2017

## 1 Motivation

Problems where there exists multiple objects and predefined relations between them and the objective is to find if there exists an ordering of execution or evaluation of these objects that respects the relations among them are often met in multiple disciplines. After a certain scale, solving these problems can become fairly hard if there is not a well defined method to deal with these kind of problems systematically.

## 2 Preliminaries

To proceed some *Graph theory* terms must be defined, familiar readers could skip to the next section.

- A *graph* is an ordered pair $G = (V, E)$ comprising a set $V$ of vertices, nodes or points together with a set $E$ of edges(relations), arcs or lines, which are 2-element subsets of $V$.
- A *graph* is *directed* if an edge between vertex $u$ and vertex $v$ doesn't imply a relation between $v$ and vertex $u$ also exits.
- A *graph* contains a *cycle iff* there exists two nodes $u$ and $v$ and there exists a path from $u$ to $v$ and another one from $v$ to $u$
- A *directed acyclic graph (DAG)*, is a directed graph with no directed cycles.
- The *out-degree* of a node is the number of its out-coming edges
- A *dependency graph* is a *directed graph* where an edge $(u, v)$ implies that node $v$ depends on $u$. Nodes in a dependency graph can represent a variety of objects depending on the type of the problem in hand, for example, they can be courses in a degree, mathematical terms in a system of equations, packages to be installed on a computer.

# 3 Method

When a dependency graph is being dealt with, there are 2 cases

1. The dependency graph contains cyclic dependencies, in this case, the cycles of dependencies (also called circular dependencies) lead to a situation in which no valid evaluation order exists, because none of the objects in the cycle may be evaluated first.[3]

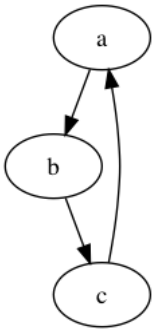


Figure 1: A cyclic directed graph

2. The dependency graph is a *DAG*. It is clear that in this case there exists a certain ordering for the objects that if followed, the terms can be computed or executed one after another respecting the dependencies among them.

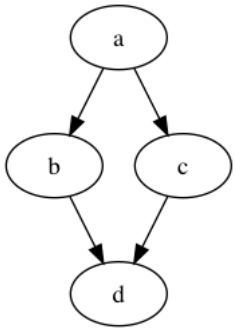


Figure 2: A directed acyclic graph (DAG)

Every directed acyclic dependency graph has a topological ordering, an ordering of the vertices such that in the case of dependency graphs the ending endpoint of every edge occurs earlier in the ordering than the starting endpoint of the edge. The existence of such an ordering can be used to characterize DAGs: a directed graph is a DAG if and only if it has a topological ordering. In general, this ordering is not always unique. [4]

Fortunately, there is a number of algorithms that can compute a topological ordering efficiently, particularly in $O(V + E)$, additionally, some of them detect an existence of a cycle as a bi-product. An algorithm described by *Kahn(1962)* is presented below

**Algorithm 1** Kahn's algorithm

```
1: procedure TopolocialSort(G)   ▷ G is the representation of the graph
2:     Q : Queue
3:     Order : List
4:     Initialize Q with nodes that have zero out-degree   ▷ Independent nodes
5:     while Q.size > 0 do
6:         u ← Q.dequeue
7:         Order.append(u)
8:         for each node v with an edge e from v to u do
9:             remove e from the graph
10:            if out-degree of v = 0 then   ▷ v's dependencies are all fulfilled
11:                Q.dequeue(v)
12:            end if
13:        end for
14:    end while
15:    if Order.size < total numbers of nodes then
16:        The graph is cyclic and no ordering exists
17:    end if
18: end procedure
```

# 4 Applications

Dependency graphs are extremely useful and can be utilized to model a variety of tasks, one of the most popular are fulfilling prerequisites of courses in a college degree. A surprising usage is in circuit analysis, an example from Digital Electronics is presented.

One of the most fundamental devices in Digital Electronics are Bistable vibrators, also known as *Flip Flops*. During the analysis of a Bistable vibrator, an important step is to assume that a transistor, let it be $Q_2$ is in the *ON* (*active*) state and the other one $Q_1$ is in the *OFF* (*Cut-off*) state. The next step is to make sure the assumption is correct by verifying that

$$I_{B2} >> 1.5I_{B2(min)}$$

this reduces the problem to calculating $I_{B2}$ and $I_{B2(min)}$, given the circuit characteristics, the following equations can be devised

$$\begin{aligned} I_1 &= \frac{V_{CC} - V_{CE(sat)}}{R_C + R_1} \\ I_2 &= \frac{V_{BE(sat)} + V_{BB}}{R_2} \\ I_{B2} &= I_1 - I_2 \\ I_3 &= \frac{V_{CC} - V_{CE(sat)}}{R_C} \\ I_4 &= \frac{V_{CE(sat)} + V_{BB}}{R_1 + R_2} \\ I_{C2} &= I_3 - I_4 \end{aligned}$$

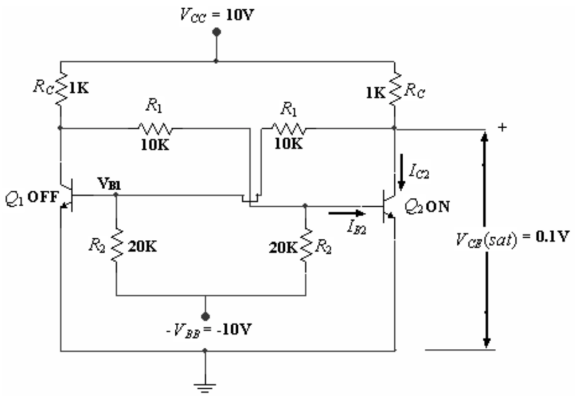


Figure 3: The fixed-bias bistable multivibrator
[2]

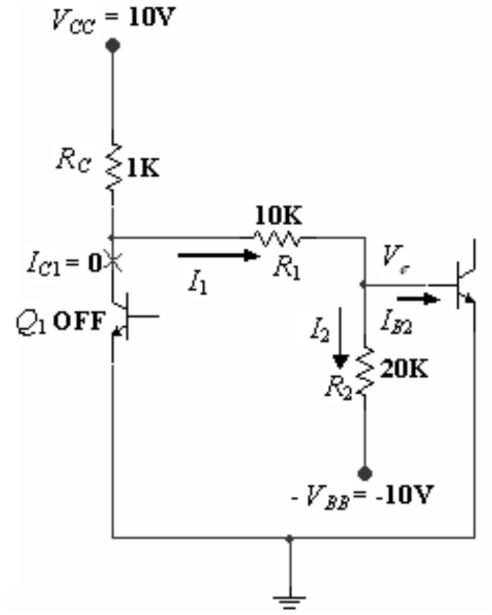


Figure 4: Circuit to calculate $I_{B2min}$
[2]

Now, these equations could be used to construct the following dependency graph to compute $I_{B2(min)}$

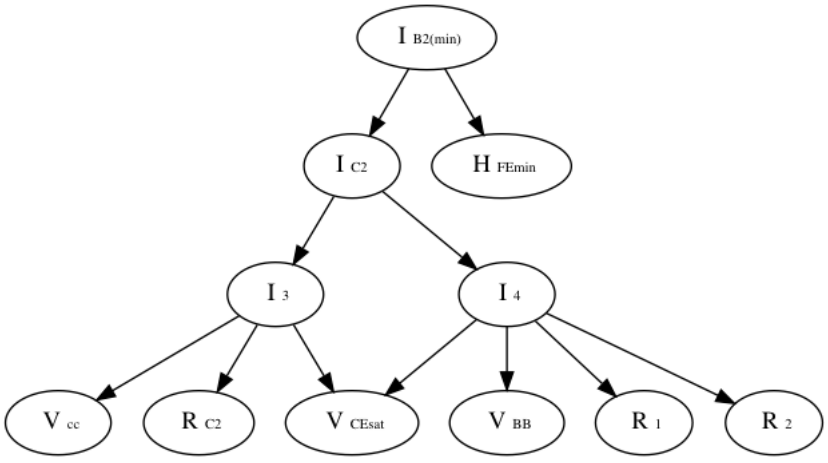


Figure 5: The dependency graph of $I_{B2min}$
[1]

and another one for $I_{B2}$

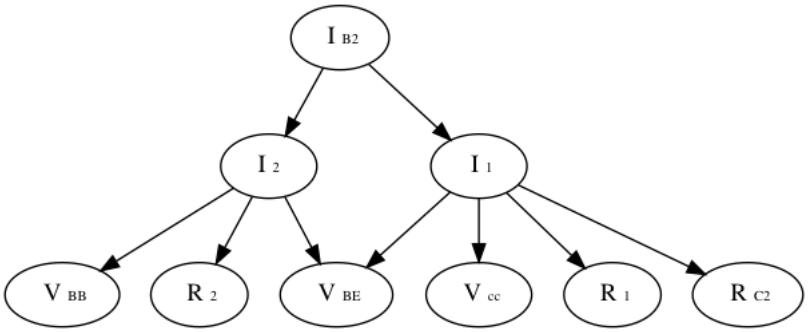


Figure 6: The dependency graph of $I_{B2}$
[1]

Note that these two graphs are a *DAGs*, thus, using *Kahn's* algorithm a topological ordering can be computed which will allow calculating each unknown one after the other with the assurance that any existing term in their equations has already been computed, this is the real advantage of dependencies graphs. Although in the previous example it may be obvious how the calculation could be done without the usage of a dependency graph, in larger problems where the number of equations and unknown terms are more, dependencies diagrams will prove to be extremely useful.

# References


[1] Prof. Taher El-Sony. Digital electronics, bistable multivibrator. University Lecture, 2017.

[2] Manmadha Rao G. Venkata Rao K., Rama Sudha. *Pulse and Digital Circuits*. Pearson.

[3] Wikipedia. Dependency graph — wikipedia, the free encyclopedia, 2017.

[4] Wikipedia. Directed acyclic graph — wikipedia, the free encyclopedia, 2017.