Working title: Verifying the Edmonds-Karp Algorithm
A Proof Pearl

S. Reza Sefidgar, Peter Lammich

Technische Universität München, {sefidgar,lammich}@in.tum.de

Abstract. the Ford-Fulkerson algorithm is a generic method that computes the maximum flow in a flow network. This paper outlines a formal proof of the Ford-Fulkerson theorem in Isabelle/HOL which follows the informal proof found in standard algorithm text books. This formalization is then used to develop an implementation of the algorithm in Standard ML. Refinement techniques are used to transform an abstract definition of the algorithm into an executable code.

1 Introduction

The Ford-Fulkerson algorithm [cite] is one of the important results of the graph theory and is used to find a solution to the maximum flow problem in flow networks. Many important mathematical problems like the maximum-bipartite-matching problem, the edge-disjoint-paths problem, the circulation-demand problem, and many other scheduling and resource allocating problems can be reduced to the maximum flow problem. Hence, the Ford-Fulkerson algorithm has major application in the field of mathematical optimization.

Despite its importance, no formalization of the Ford-Fulkerson algorithm has been developed in modern proof assistants. The only similar work that the authors are aware of is formalization of the Ford-Fulkerson algorithm in Mizar [cite]. This formalization defines and proves correctness of the algorithm at the level of graph manipulations without providing concrete implementation of the algorithm. Providing such an implementation is specially important, as it could provide the basis for many verfied programs with practical importance.

This paper present a formalization of the correctness proof of the Ford-Fulkerson algorithm in Isabelle/HOL. Our proof is based on the informal proof of the algorithm which is presented in the book "Introduction to algorithms" [cite]. Due to practical importance of the algorithm, we also present a verified implementation of the algorithm. Isabelle/HOL provides some automation for generating the code corresponding to a verified algorithm, however, in order to use the code generating features, one needs to do the formalization with executability in mind. Being limited only to executable concepts in the formalization has the disadvantage of cluttering the proofs with implementation details. Such approach makes the proofs more complicated, and may even render proofs of medium complex algorithms unmanageable.

One solution for the aforementioned problem is refinement [cite]. In order to generate executable code of an algorithm using refinement, we first formulate the algorithm on an abstract level. The abstract version of the algorithm has a clean correctness proof as it only captures the idea behind the algorithm. Next, we refine the abstract definition of the algorithm towards an executable implementation in possibly multiple refinement steps. During each step, we only need to prove the correctness of the implementation of a particular abstract concept. Hence, we can be sure about the correctness of the resulting executable program as each refinement step preserves the correctness.

There are several approaches to data refinement in Isabelle/HOL. We will be using the Autoref tool [cite], which has been used for proving more complex results such as formalized implementation of Hopcroft's DFA-minimization algorithm [cite]. Given an algorithm phrased over abstract concepts like sets and maps, it automatically synthesizes a concrete, executable algorithm and the corresponding refinement theorem. To make it applicable for the development of actual algorithms, Autoref is integrated with the Isabelle Refinement Framework [cite] and the Isabelle Collection Framework [cite].

2 Short Background on MinCutMaxFlow and FoFu

A flow network is a weighted digraph where the set of vertices (nodes) V is finite, and each edge (u, v) has a real capacity c (u, v) ¿=0. There are two distinct vertices s (source) and t (sink) in the flow network. In this case the network is called an s-t network. To make the formalization simple, some additional assumptions are added to the definition as following: 1) The source only has outgoing edges while the sink only has incoming edges; 2) If the flow network contains an edge (u, v) then there is no edge (v, u) in the reverse direction; and 3) every vertex of the flow network must be on a path from s to t. Although these assumptions seems restrictive, any network may be transformed such that it satisfies all the aformentioned properties.

An s-t flow on a weighted digraph is a function f: E -¿ R which satisfies the following conditions. 1) (capacity constraint) the flow of each edge is a non-negative value which is smaller or equal to the edge capacity; 2) (conservation constraint) For all vertices except s and t, the sum of flows over incoming edges of the vertex is equal to the sum of flows over outgoing edges of the vertex.

The value of an s-t flow f is denoted by —f —, and is computed by subtracting the sum of the flows over all incoming edges of s from the sum of the flows over all outgoing edges of s. Given an s-t network G, the maximum flow problem involves finding a flow f in G such that the value of f is greater or equal to the value of any other s-t flow in G. (Note that the value of the flow in our model of a flow network equals sum of flows over edges going out of s. Because the model prohibits edges coming into the source.)

One of the important results in mathematical optimization is the corelation between flows and cuts in flow networks. An s-t cut in a flow network with source s and sink t is a partition of vertices that puts s and t in different subsets. Then

the capacity of a cut is defined as sum of the edge capacities over all edges going from the source side to the sink side. Consider a valid flow in a flow network G. For any cut in G, the flow that crosses between the two subsets of the cut cannot exceed the capacity of the cut. So cuts define an upper bound for any flow in the flow network. The Ford-Fulkerson theorem tightens this bound and states that the value of the maximum flow is equal to the capacity of the minimum cut.

The Ford-Fulkerson algorithm is the direct consequence of the Ford-Fulkerson theorem. It solves the maximum flow problem in a greedy approach. Assume an instance of the maximum flow problem in an s-t network. The algorithm starts with a zero flow: f (e) = 0 for all the edges e. In each iteration of the algorithm the value of f is increased by pushing flow along a path from s to t up to the limits imposed by the available edge capacities. The process would continue until no more paths can be found to push additional flow from s to t.

The total flow value is increased by manipulating the flow values along different edges. However, the flow along specific edges might decrease during this process. In order to perform these operations, the Ford-Fulkerson algorithm defines the residual graph of the flow network. Intuitively, the residual graph corresponding to a flow in a given flow network, has the same vertices as the flow network. It also and consists of edges with capacities representing how flow can be changed along the edges of the flow network.

In each iteration of the algorithm, the residual graph G_f corresponding to the current flow in the network is constructed. Then the algorithm looks for an augmenting path— a simple path from source to the sink— in the residual graph. If such a path could be found, it will be used to construct a flow in the residual graph. This flow provides a road map for modifying the current flow in the flow network. The process of increasing the current flow using an augmenting flow is called augmentation. Let f be the current flow in an s-t network G = (V, E) with capacity function c, and f be a flow in the corresponding residual graph. The augmentation of f using f' is a function from $V \times V$ to N, defined as following: (???? definition of augment function)

The intuition behind the above definition is based on the definition of the residual graph. The augment procedure increases the flow on (u, v) by f (u, v) but decreases it by f (v, u) . This is due to the fact that pushing flow on the reverse edge on the residual graph signifies decreasing the flow in the original network. In each iteration of the algorithm we replace the current flow with the result of the augment function, and the algorithm terminates if no more augmenting paths can be found in the residual graph. According to the Ford-Fulkerson theorem, the flow computed in this states is indeed the maximum flow in the flow network. The theorem states that following statements are equivalent:

1- f is a maximum s-t flow in flow network G. 2- there is no augmenting path in the residual graph G_f corresponding to flow f and G. 3- there is an s-t cut C in G such that capacity of C is equal to the value of f.

The termination of the Ford-Fulkerson algorithm depends on how the augmenting path is found. For the networks with irrational edge capacities the algorithm might even fail to terminate. In practice, the maximum-flow problem

often arises with integral capacities. Moreover, we may convert rational edge capacities to integral values by multiplying them with a big enough integer. If f* denotes a maximum-flow in the transformed network, then the algorithm will iterate at most $|f^*|$ times, since the augmentation procedure increases the flow value by at least one unit in each iteration. Assuming that we use breadth-first-search (BFS) or depth-first-search (DFS) for finding augmenting paths makes the total execution time of the Ford-Fulkerson algorithm $O(E|f^*|)$, as BFS and DFS run in O (E + V).

The Ford-Fulkerson algorithm is sometimes called a method instead of an algorithm because it does not fully specify the procedure for finding the augmenting path in the residual graph. There are several implementations with different execution times. For instance, the Edmond-Karp algorithm uses breadth-first-search (BFS) for finding augmenting paths and has O (VE 2) running-time. The augmentings paths that are computed using BFS are also shortest paths connecting source to the sink in the residual graph. Edmond and Karp showed that in the Ford-Fulkerson algorithm, if each augmenting path is the shortest one, the length of the augmenting paths is non-decreasing and it always terminates. Which finally gave the polynomial algorithm for solving the maximum flow problem in generic case of real edge capacities.

3 Formalizing MinCutMaxFlow and Fofu

GOAL: Present highlights of our formalization, persuade the reader that we have done something substantial. Reader should think: Yeah, they did cool stuff!

[???* Thematize definition of flows: On Graphs vs. on Networks. You find both in the literature. We decides for [...]. This is better suited, as is gives nice and elegant proofs, even formally, bla bla bla]

Proof of MinCut-MaxFlow: Follows the textbook proof. Uses Isar to even look like a textbook proof, being comprehensible even without running Isabelle/HOL.

For example, *augment_flow_presv_cap*. Present formal proof text. Perhaps oppose it to textbook proof?.

Abstract Algo looks like pseudocode presented in texbooks. Oppose our algo to textbook pseudocode.

4 Refinement to executable code

GOAL: Same as previous section

Introduction to refinement. (On abstract level)

TODO: Show better complexity bound for FoFu with shortest path! –¿ Edmonds-Karp algo!

FoFu-Scheme, can be instantiated with pathfinders now. These are developed independently, they are just search algorithms on graphs. We did BFS and DFS. DFS took from Refinement Framework examples. BFS: Based on our existing formalization for VSTTE12 verification competition, but extended to return shortest path, and refined to imperative code.

??? Highlights of the BFS formalization?

Down to executable code: Implement things that where not yet specified/specified in a non-executable way: Residual graph: By combination of c and f, and pred-succ. TODO: Directly! All we need is successor function. Implemented by tabulating adjacent nodes of each node (pred-succ), and using c and f to filter out actual successor nodes. augpath-spec by BFS/DFS (which work on succ-functions) bottleneck and augment: Give iterative impl.

Use efficient data structures: Arrays for c,f, and ps. c and f in row-major indexing.

List for augmenting path. [TODO/Future work: Iterator scheme]

In BFS, we use lists for the queues (we have split the queue into one for the current level, and one for the next level), and an array for the predecessor map.

The refinement is performed using the Sepref tool.

NetCheck We also implemented an algorithm that reads a list of edges and a source and target node, converts it to a capacity matrix and an adjacency map, and checks whether the resulting graph satisfies our network assumptions. We proved that this algorithm returns the corresponding Network and adjacency function iff the input describes a valid network, and returns a failure value otherwise.

Combining the implementation of the Edmonds-Karp algorithm with the Network checker yields our final algorithm, for which we can export code, and have proved the theorem: [...].

5 Benchmarking

6 Conclusion

... and related work Mizar-Formalization: What did they do.

Contributions

Formal proof of mincut maxflow fofo-scheme inst to edmonds karp refinement down to executable code. Roughly 5 times slower than Java. (What about OCaml) + NetCheck

What shines (its a Pearl) Min-Cut Max-Flow: Textboook like formal reasoning: Comprehensible proof, BUT machine checked (Present one (carefully worked) example in paperI. We could use lemma *augment_flow_presv_cap*)

Refinement based approach: Fofu-Scheme, instantiation to EdsKa. +1: Abstract Algo looks almost like pseudo-code you would expect in textbook. +2: Fofu-Scheme proved correct for all aug-path finders. EdsKa is instantiation of it. +3: Modularity: Fofu-scheme and pathfinder developed+proved independently of each other.

Down to executable code, plugging in efficient data structures.

Some minor contributions: Reusable BFS algorithm Imperative matrix data structure (really minor).

References