Are Attention Flows All You Need?

Stanford CS224N Custom Project

Tomas M. Bosschieter
Institute for Computational and Mathematical Engineering
Stanford University
tomasbos@stanford.edu

Key Information

- **Name**: Tomas M. Bosschieter
- **Student ID**: 06261732
- **Topic**: Explainable NLP
- **External collaborators**: None
- **TA Mentor**: Isabel Papadimitriou
- **Sharing project**: No.

Abstract

While deep learning has achieved many successes in fields such as computer vision and NLP, the lack of explainability remains a severe problem, especially in NLP. While Shapley values, a solution concept in cooperative game theory, have been heavily popularized, they suffer from mathematical incompatibilities and not providing natural human-centric explanations. As one alternative to Shapley values, attention weights have been put forward as explainable in recent literature, although there is no one-fold interpretation they provide. One promising concept is that of Attention Flows, which state that the total outgoing flow of a node in a max flow graph of a neural network with attention values as weights is the node's Shapley value at the layer-level. We first discuss attention versus Shapley values, and then do a case study how these attention flows differ from the raw attention weights.

1 Introduction

As machine learning has become ubiquitous, with applications ranging from from Computer Vision [1, 2] to soccer match predictions [3], the demand for explainability methods for machine learning has also skyrocketed. Reasons thereof include discovering potential biases in data, providing medical professionals with explanations of what story statistics tell regarding e.g. a disease, and many more. Explainability in machine learning could also be used to help understand linguistic data artifacts [4], and aid those making real-world big-impact decisions understand potential effects of their deicisions, e.g. policy makers.

To meet these needs, explainability methods such as SHAP [5] and LIME [6] gained much traction in the wider research community. SHAP and LIME aim at providing (primarily) global and local explanations respectively. SHAP provides feature importances by aggregating the features' mean absolute contributions to predictions and also generates partial dependence plots, while LIME is a model-agnostic local surrogate model for specific predictions. One exciting breakthrough in ML is the introduction of attention [7], which has not only revolutionized the architectures of machine learning models to improve accuracy, but it has also given notoriously opaque models such as NLP models hope to become more interpretable. Specifically, attention weights could help understand

which words (or phrases) are most important in e.g. text classification (TC) and neural machine translation (NMT). Intuitively, the attention weights learned by a model indicate how important the corresponding words are in NLP tasks such as Text Classification (TC) and Neural Machine Translation (NMT): there appears to be a monotonic relationship between a word's attention weights and the contribution it has to final predictions. In other words, the larger the assigned weights, the more important that word appears to be.

However, there are also many challenges that come with attention [8] and Shapley-value-based explanations [9]. For example, at each layer each feature computes the attention to the other features, and after one iteration of backpropagation is done, the cycle continues, and so it's unclear how to specifically interpret attention as an explanation, given that it has been heavily influenced by other features' embeddings after perhaps only few iterations. On the other hand, interpreting initial attention weights might not be as informative, though, given the effects of the randomized initialization. Even after disregarding these points, there is still a lot of controversy. Some papers (e.g. [10]) argue attention provides no explanations by showing that perturbed attention weights yield equivalent predictions as without any perturbation. On the other hand, [11] refutes this claim by observing that [10] only considers text classification as task, in which attention has little impact on accuracy. [11] shows that on other tasks, such as pair sequence tasks and Neural Machine Translation (NMT), attention does improve accuracy while also providing intelligibility. They do this by setting the attention weights to uniform, random, and permutations and comparing their respective performances. In order to estimate a feature's importance, they compare attention weights before and after zeroing out a feature's attention and see what the Jensen-Shannon (JS) divergence is between the two distributions.

As both Shapley values and attention values are argued to be 'interpretable'[1], a natural question to ask is if they are similar, if not equivalent somehow? Ethayarajh and Jurafsky [13] show that they are, in fact, not equal. They observe that the max flow throughput of a neural network with attention weights has an interesting property: for a node n at layer ℓ, the sum ϕ_n of its attention weights to all the nodes in layer $\ell + 1$ is the 'contribution' of node n to the total max flow in the network. The total throughput in the network would decrease by ϕ_n if the node's attention weights are zeroed out. [13] shows that this satisfies the axioms of Shapley values. Thus, the sum of the attention weights of the outgoing edges of a node represents the node's Shapley value. In other words, 'attention flows' are Shapley value explanations. The main focus of [13] is theoretically proving that these attention flows do indeed satisfy the axioms of Shapley values, but no experiments are shown, while empirical results are often well-appreciated in NLP [7].

One of the main issues with the discussion on whether attention values are interpretable is that arguments to refute others' claims are often based on only a very specific subset of tasks, data sets, or else, which might not be representative of attention overall [8, 10]. This miscalibration hinders converging to a consensus about the interpretability of attention. We aim to make a step towards bridging this gap by outlining these different settings, what some of the main shortcomings of attention and Shapley values are, and what challenges remain. Additionally, we make an experimental contribution to [13] by computing the max flow throughput and attention-value-distributions for several examples in the max-flow network. We compare this to the raw attention values for reference.

The main contributions of this paper are:

- Giving an overview of the discussion on attention versus Shapley values, as many arguments and counterarguments are based on different contexts, data sets, and models.
- We perform a case study in which we compute attention values in the max flow network and compare this to the attention values in the model-trained network (i.e. the network that max flow is performed on).

2 The Theoretical Minimum

We give a brief overview of the attention mechanism and Shapley values in this section.

[1] Note that the notion of interpretability is not well-defined [12].

2.1 Sequence-to-Sequence Models and Attention

Sequence-to-sequence models are a type of Recurrent Neural Network (RNN) [14] that are popular for MNT, question answering, and so on. These recurrent models enable parsing text sequentially so that the model can be used to model corpuses and texts of different lengths. Models like Long Short-Term Memory (LSTM) models [15] are considered state-of-the-art recurrent networks to solve sequence-to-sequence problems, but are computationally expensive in memory and hard to parallelize due to their recurrent nature. Additionally, LSTMs tend to overfit, which is exacerbated by the fact Dropout [16] is non-trivial to incorporate into LSTM models given its inner gates.

The sequential nature of recurrent models is a bottleneck for the use of such recurrent models in big-data modelling, especially NLP. There is a tremendous amount of data available on the internet, and going through it sequentially without any parallelization is infeasible. Observing that recurrent models are constructed to connect different inputs and layers, [7] propose using attention to construct the Transformer architecture, the current state-of-the-art. In transformers, the recurrent connections are replaced by the attention mechanism as a whole rather than being used alongside a recurrent network.

2.1.1 Attention Mechanism

The attention mechanism consists of a mapping that maps queries Q and key-value pairs (K, V) to an output, the attention weight(s). Two common attention functions are (i) a scaled dot-product of these queries and key-value pairs [7], and (ii) additive attention [17]. The additive attention mechanism is defined as

$$\alpha = \text{softmax}(\mathbf{w_3}^\top \tanh(\mathbf{W_1}K + \mathbf{W_2}Q)), \tag{1}$$

where the weight matrices $\mathbf{w_3}^\top, \mathbf{W_1}, \mathbf{W_2}$ are to be learned by the model. On the other hand, the multiplicative attention mechanism computes attention weights as

$$\alpha = \text{softmax}(\frac{QK^\top}{\sqrt{d_k}})V, \tag{2}$$

where $d_k = \dim(K)$ represents the dimension of the keys K. Specifically, for nodes i, j, consider the query $\mathbf{q_i} = Q\mathbf{x_i}$, key $\mathbf{k_i} = K\mathbf{x_i}$ and value $\mathbf{v_i} = V\mathbf{x_i}$ for an embedding $\mathbf{x_i} = E\mathbf{w_i}$ of a word w_i given an embedding matrix $E \in \mathbb{R}^{d \times |V|}$. Then, the non-scaled attention weights are computed as cosine similarity scores, i.e. $\mathbf{e_{i,j}} = \mathbf{q_i}^\top \mathbf{k_j}$, so that finally the attention score from node i to node j is yielded by putting the distribution of $\mathbf{e_{i,j}}$ through the softmax distribution, i.e.

$$\alpha_{\mathbf{i,j}} = \frac{\exp(\mathbf{e_{i,j}})}{\sum_k \exp(\mathbf{e_{i,k}})}. \tag{3}$$

The final attention output of a word is then the weighted sum of all the attention scores between said word and the words it attends to, i.e. $\sum_j \alpha_{\mathbf{i,j}} \mathbf{v_j}$.

Now, while the attention mechanism allows every node to attend to the other nodes in the subsequent layer, the attention weights might represent how layer $\ell + 1$ depends on layer ℓ in one particular way, while it may also depend on layer ℓ in other ways. Having multiple self-attention processes at each layer allows the model to capture different linguistic constructs and phenomena, rather than focusing on only one. This enriches the model's understanding of the corpus texts and thus hopefully better equipped to make well-informed, high-accuracy predictions. This multi-head attention mechanism is defined by concatenating the attention vectors for each head and scaling by a parameter matrix W [7], i.e.

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_n)W. \tag{4}$$

2.2 Shapley Values

Shapley values are a game-theoretic concept that have been popularized to explain machine learning models [5]. At its essence, they measure the expected marginal contribution of a feature, averaged over all possible combinations of contributions. Consider playing a game with N players to achieve a payoff p, where each player contributes to achieving this payoff. Shapley values aim to answer the question 'How does one fairly distribute the payoff over the players according to their contribution to the achieved payoff?' The analogy to features in a data set is that features 'work together' in a

model (e.g. through a linear combination) to come up with a certain prediction by the model. Which feature was most important in making this prediction? How about which feature was most important on average in making predictions? How do we distribute this 'importance' (i.e. average absolute contribution)?

Shapley values are defined through value functions $\text{val}(\cdot)$ of subsets S of players in a set $N = \{1, 2, \ldots, n\}$. Specifically, the Shapley value $\phi_j(\text{val})$ of player (or feature) j in a game with total payoff v is given by

$$\phi_j(v) = \sum_{S \subset N \setminus \{j\}} \frac{|S|!(n - |S| - 1)!}{n!} (\text{val}(S \cup \{j\}) - \text{val}(S)), \tag{5}$$

following. These Shapley values are the only constructs that jointly satisfy the Efficiency, Symmetric, Dummy and Additivity axioms [18]:

- **Efficiency**: For a model prediction x, the Shapley values much add up to the difference between the prediction of model $\hat{f}$ and the average over all values $X \ni x$: $\sum_{j=1}^{p} \phi_j = \hat{f}(x) - \mathbb{E}_X(\hat{f}(X))$.
- **Symmetry**: If two players j, k induce the same change in payoff, and thus contribute equally, for all coalitions S, then their Shapley values must be equal. That is, $\text{val}(S \cup \{j\}) = \text{val}(S \cup \{k\}) \;\; \forall S \subset N \setminus \{j, k\} \implies \phi_j = \phi_k$.
- **Dummy**: If a player j induces no change in payoff in all coalitions, it has a Shapley value of 0, i.e. if $\text{val}(S \cup \{j\}) = \text{val}(S) \;\; \forall S \subset N \setminus \{j\}$, then $\phi_j = 0$.
- **Additivity**: If a player j contributes $\phi_j(A)$ to a game with payoff A and $\phi_j(B)$ to another game with payoff B, then the Shapley value of that player is the sum of its Shapley values in the different games, i.e. $\phi_j(A + B) = \phi_j(A) + \phi_j(B)$.

Computing Shapley values precisely is arduous and often computationally infeasible. A popular Python module to efficiently approximate Shapley values is SHAP, which includes KernelSHAP, TreeSHAP, DeepSHAP, and more [5].

2.2.1 Shapley Value Inconsistencies

While Shapley values seem to be well-grounded theoretically, a variety of problems have been laid bare, particularly in [9]. They argue that (i) there are mathematical issues with the axioms of Shapley value, and (ii) there is sometimes a gap between Shapley values as feature importance values and how humans would interpret feature importance. The main argument made is that there's a fundamental difference between interventional and conditional value functions. Given a function f with domain $\mathbb{R}^d$, then if for some feature i it's true that $f(\mathbf{x}) = f(\mathbf{x}')$ whenever its j components are the same, $x_j = x'_j$, then that feature j is redundant for prediction using f as it has no interventional effect and no contribution, hence a Shapley value of 0. If x_i is a proxy for x_j (as is the case in many data sets) that does affect prediction, however, then x_j should have a non-zero Shapley value. Such conditional versus interventional distributions are explored and discussed in more detail in [9].

3 Methods

3.1 Attention Flows

Imagine a (neural) network as a graph that contains a total payoff equal to the max flow of that network. Then, [13] shows that the total throughput of a node (i.e. sum of its outgoing flows) is the Shapley value of that node. See Figure 1, taken from [13], for an illustration. [13] shows this by noting that all the players are disjoint and the max flow is computed over the entire graph, not all subgraphs. After all, max flow algorithms like the Edmund-Karp algorithm [19] and Dinic's algorithm [20] output only one graph. Then, for a value function $v(\cdot)$ measuring the total flow, note that $v(S \cup \{i\}) = v(S) + |f_o(i)|$ for output flow f_o of node i. This follows directly from the definition of a max flow graph, and is the main building block of the proof. Additionally, they note that for the group of leave-one-out methods [10] as a subset of erasure methods, the leave-one-out values are not Shapley values except when the leave-out player is a null player, i.e. if it satisfies the dummy property from Section 2.2.

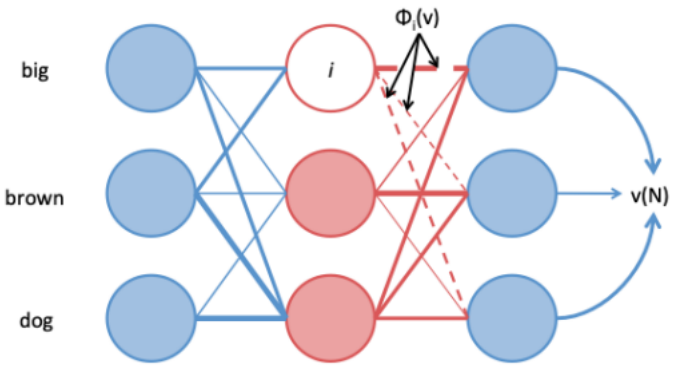

Figure 1: Consider the first node, i, in the second layer of the neural network where the weights are the attention weights. Then, the sum of the outgoing flows of node i in the max flow graph of this network is equal to the Shapley value ϕ_i of i. That is, if these weights were zeroed out, then the total payoff would be reduced by ϕ_i. (This image is taken from [13].)

3.2 Experimental Setup

In order to run experiments with attention flows, we first need a pretrained Transformer model from Hugging Face: we use the pre-trained distilbert-base-cased AutoTokenizer and the distilbert-base-cased AutoModel on evaluation mode. (The adoption of a pre-trained transformer model is warranted here as I have a suspicion my $10 Colab Pro credit isn't going to cut it to train a full model...) For each of the 6 layers and 12 heads in the multi-head attention mechanism we track the attention weights for easy visualization later on.

3.2.1 Feature Engineering

Selecting one particular head, we select all layers in the network and add a source s and target (or sink) t. The source s is connected to all nodes in the first layer with capacity ∞, whereas the nodes in the last layer are connected to the target t with capacity ∞, too. The network is connected layer-by-layer through the attention weights. We compute the max flow of this network using Dinic's algorithm [20], a specific implementation of the Ford-Fulkerson algorithm, but note that Dinic's algorithm demands that all weights be integers, since convergence is not guaranteed for arbitrary float capacities [19]. To that end, we compute $\lambda = \min_a \lfloor \log_{10}(a) \rfloor$ over all attention weights a in the network to obtain the lowest order of the attention weights. Given the finite precision and the fact the network is finite, λ is well-defined, and we scale all attention weights by λ. This yields an attention-based network of integer weights that includes a source and target, which is what we apply Dinic's algorithm to.

For that purpose, we first construct the adjacency matrix by mapping the network with T tokens and L layers as follows: the attention weight between node i in layer $0 \leq \ell < L$ and node j in layer $0 < \ell + 1 \leq L$ in the network is mapped to $(1 + T\ell + i, 1 + T(\ell + 1) + j)$ with the scaled attention weight $\lfloor \lambda a \rfloor$. This establishes the following block structure of the adjacency matrix, where $A(\ell_i, \ell_j)$ represents the attention weights from layer i to layer j as a square matrix, while $a(s, \ell_0)$ and $a(\ell_{L-1}, t)$ are row vectors from the source to layer $\ell = 0$ and layer $\ell = L - 1$ to the target respectively (and contain infinite capacity):

$$\text{Adj} = \begin{bmatrix} 0 & a(s,\ell_0) & & & & & \\ & & A(\ell_0,\ell_1) & & & & \\ & & & A(\ell_1,\ell_2) & & & \\ & & & & \ddots & & \\ & & & & & A(\ell_{L-2},\ell_{L-1}) & \\ & & & & & & a(\ell_{L-1},t)^\top \\ & & & & & & 0 \end{bmatrix}. \tag{6}$$

Note that the source has no incoming edges, and the target has no outgoing edges. For computational efficiency, we transform the adjacency matrix to a compressed sparse row (CSR) format [21], a sparse matrix, which is then fed into Dinic's algorithm.

4 Results

We first query the model using the string "Isabel is pretty cool!". The distilbert AutoTokenizer splits this into the tokens

['[CLS]', 'Isabel', 'is', 'pretty', 'cool', '!', '[SEP]']

While the model has 6 layers and 12 heads in its multi-head self-attention mechanism, we focus on the first head to allow for detailed inspection of attention weights. The results and findings for other heads are similar.

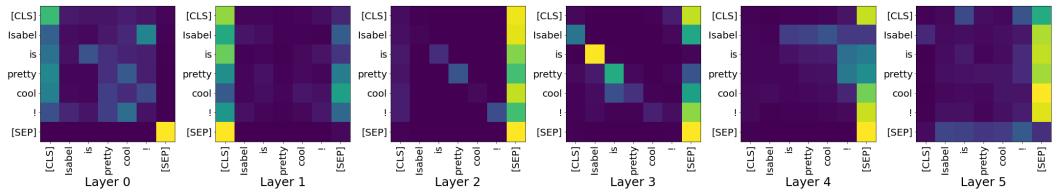

Figure 2: Attention weights in the first head for the query "Isabel is pretty cool!".

See Figure 2 for a visualization of the attention weights in the model when queried with "Isabel is pretty cool!". Note that as these are self-attention weights, they are within a single layer, i.e. not connected to other layers. As such, we ought to connect these weights in a graph so we can compute the sequential attention flows. This could be achieved by attaching each attention layer to the next, which matches the original construction of [13] in Figure 1. See Equation (6) for the specific block matrix construction. After computing the max flow using Dinic's algorithm, we obtain the attention weights shown in Figure 3.

Interestingly, the 'normal' attention weights in Figure 2 show some large attention weights, but these locations are not at fixed locations throughout the layers. In the max flow graph, if one layer does not have a high attention value at a particular location even though it's high at that location in the other layers, this forms a bottleneck and only little flow is sent through that pipeline. This might explain why the the largest attention flow weights are much more concentrated at particular nodes after computing the max flow and min cut, rather than being spread out. Computing the attention flow values is trivial given the matrices that underlie Figures 2, 3. Additionally, a detailed analysis of the weights themselves[2] reveals that many dark-colored attention weights are non-trivial, although 2-10x smaller than the maximum value, resulting in a perhaps slightly skewed coloring.

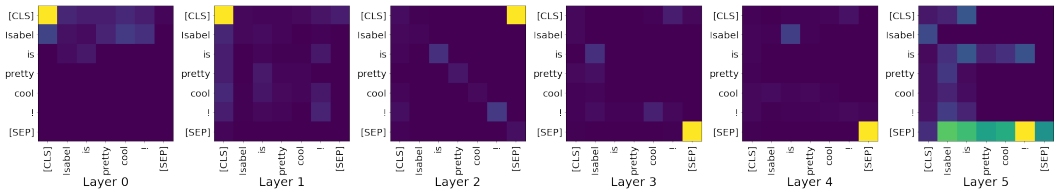

Figure 3: Attention weights in the first head for the query "Isabel is pretty cool!" after performing the max flow algorithm using Ford-Fulkerson and Dinic's algorithm.

[2]The code is publicly available at `https://colab.research.google.com/drive/1kso_IOtzKCyuVb2xPcUSoWAHKXuPP4KO?usp=sharing`.

5 Conclusions

We have performed a detailed analysis of Shapley values and attention and considered arguments for and against the interpretability of these constructs. One promising direction to connect these concepts is the use of attention flows, and while theoretical work has shown that a node's sum of outgoing attention weights in a max flow graph is the node's Shapley value, we perform a case study to compare these attention flows to the 'raw' attention weights before applying the Ford-Fulkerson algorithm. It appears some bottlenecks in an attention weights graph might concentrate attention weight more locally, but more research is needed to conclude whether attention flows could beat state-of-the-art explainable models for model interpretation.

References

[1] Zhengwei Wang, Qi She, and Tomas E Ward. Generative adversarial networks in computer vision: A survey and taxonomy. *ACM Computing Surveys (CSUR)*, 54(2):1–38, 2021.

[2] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. In *International Conference on Machine Learning*, pages 8821–8831. PMLR, 2021.

[3] Tom Decroos and Jesse Davis. Interpretable prediction of goals in soccer. In *Proceedings of the AAAI-20 Workshop on Artificial Intelligence in Team Sports*, 2019.

[4] Xiaochuang Han, Byron C Wallace, and Yulia Tsvetkov. Explaining black box predictions and unveiling data artifacts through influence functions. *arXiv preprint arXiv:2005.06676*, 2020.

[5] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. *Advances in neural information processing systems*, 30, 2017.

[6] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should I trust you?": Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pages 1135–1144, 2016.

[7] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[8] Adrien Bibal, Rémi Cardon, David Alfter, Rodrigo Wilkens, Xiaoou Wang, Thomas François, and Patrick Watrin. Is attention explanation? an introduction to the debate. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3889–3900, 2022.

[9] I Elizabeth Kumar, Suresh Venkatasubramanian, Carlos Scheidegger, and Sorelle Friedler. Problems with shapley-value-based explanations as feature importance measures. In *International Conference on Machine Learning*, pages 5491–5500. PMLR, 2020.

[10] Sarthak Jain and Byron C Wallace. Attention is not explanation. *arXiv preprint arXiv:1902.10186*, 2019.

[11] Shikhar Vashishth, Shyam Upadhyay, Gaurav Singh Tomar, and Manaal Faruqui. Attention interpretability across nlp tasks. *arXiv preprint arXiv:1909.11218*, 2019.

[12] Finale Doshi-Velez and Been Kim. Towards a rigorous science of interpretable machine learning. *arXiv preprint arXiv:1702.08608*, 2017.

[13] Kawin Ethayarajh and Dan Jurafsky. Attention flows are shapley value explanations. *arXiv preprint arXiv:2105.14652*, 2021.

[14] Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Interspeech*, volume 2, pages 1045–1048. Makuhari, 2010.

[15] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[16] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[17] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[18] Christoph Molnar. *Interpretable machine learning*. Lulu. com, 2020.

[19] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2022.

[20] Efim A. Dinic. Algorithm for solution of a problem of maximum flow in networks with power estimatio. *Soviet Math. Doklady, vol. 11, pp. 1277-1280*, 1970.

[21] Aydin Buluç, Jeremy T Fineman, Matteo Frigo, John R Gilbert, and Charles E Leiserson. Parallel sparse matrix-vector and matrix-transpose-vector multiplication using compressed sparse blocks. In *Proceedings of the twenty-first annual symposium on Parallelism in algorithms and architectures*, pages 233–244, 2009.