Jonas Hafermas - Zoya Masih

Parallelization of Maximum Flow Problem on Big Graphs

A Status Report

Table of contents

1 Recap: Max Flow Problem

2 Dinitz's Algorithm

3 Sequential implementation and Evaluation

4 Parallelization Ideas

5 Challenges and Outlook

Table of Contents

1 Recap: Max Flow Problem

2 Dinitz’s Algorithm

3 Sequential implementation and Evaluation

4 Parallelization Ideas

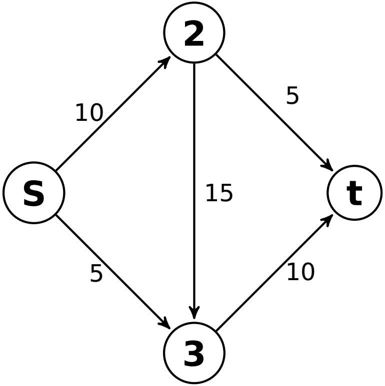
5 Challenges and Outlook

Maximum Flow Problem

- Our simplified definition:
  'The Maximum Flow problem is about
  finding the maximum possible flow through a flow network'

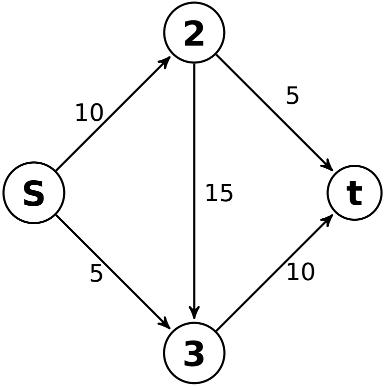
Example

- Suppose that we have a network with 4 nodes
- We want to transfer data from the source (S) to the target (t)
- Each edge has limited flow capacity for data propagation



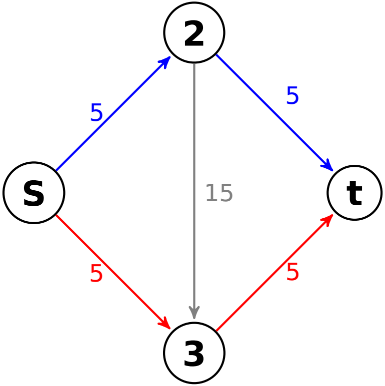
Example: Assumptions

- Flow on a node cannot exceed its capacity
- The total incoming and outgoing flow equals on each node (*conservation of flow*)
- There could be several paths of our (data) flow routed through



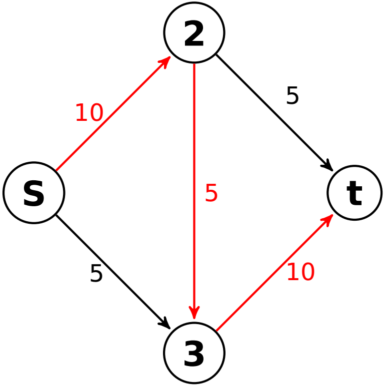
Example: One Solution

- One possible way is to split flow across multiple paths (blue and red)
- Incoming Flow to **t** is 10
- Is this the maximum flow achievable?



Example: Optimal Solution

- We can also pass flow = 15
- Solution: pass 10 from **s** to **2**, and use the edge 2-3 for the excessive flow
- The sum of incoming flows at **t** equals 15



Motivation and applications

- The problem has many applications in real world. Following are 2 examples

1. Circulation with Demands:
   - A collection of supply nodes that want to ship products or goods
   - A collection of demand nodes that want to receive the products
2. Airline scheduling
   - Adjusting the number of passengers and the amount of loading on air networks

Motivation and applications

- It can also be used in IO problems and optimization
  - Task scheduling
  - Data transfer
  - Network Routing

Table of Contents

1 Recap: Max Flow Problem

2 Dinitz's Algorithm

3 Sequential implementation and Evaluation

4 Parallelization Ideas

5 Challenges and Outlook

Dinitz's Algorithm - Overview

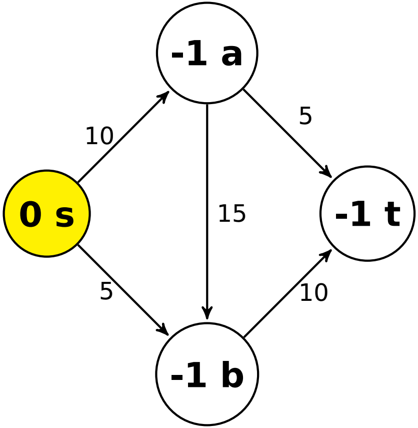
- One of the several algorithms used for solving the max flow problem
- Invented by Yefim Dinitz in 1970,
  - Prior to Edmonds-Karp algorithm, some internal similarities exist

Dinitz's Algorithm – Advantages

- Better runtime complexity than Ford-Fulkerson and Edmonds-Karp
  - $\mathcal{O}(V^2E)$ compared to $\mathcal{O}(VE^2)$ for Edmonds-Karp
  - This improves scalability significantly for dense graphs
- Relatively easy to implement
- Uses so-called level graphs and the concept of blocking flow
  - This helps to achieve its superior performance

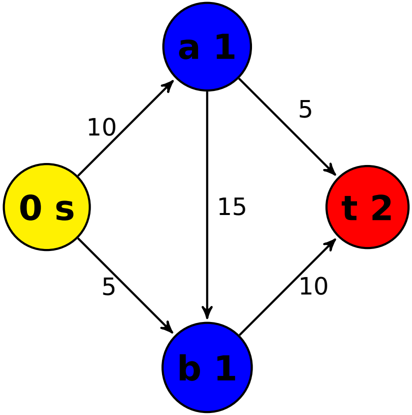
Dinitz's Algorithm – How it works

1. Initially, the source is labelled 0 and other vertices are labelled -1



Dinitz's Algorithm – How it works

2 Each unvisited child of vertex u with label i, receives label i+1 (BFS)



Dinitz's Algorithm – How it works

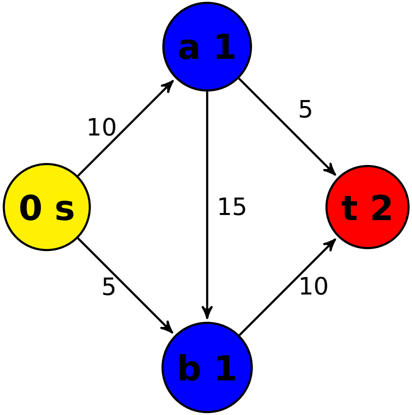
3. For the paths in order of the labels, the algorithm finds the min capacity
   - Finding the paths with DFS method

Dinitz's Algorithm – How it works

3. For the paths in order of the labels, the algorithm finds the min capacity
   - Finding the paths with DFS method

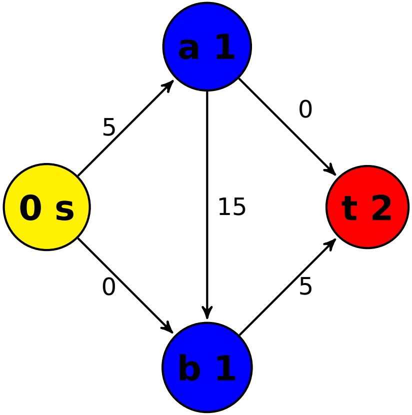
0	1	2	Path order
s	a	t	min{10,5}=5
s	b	t	min{5,10}=5

- The flow is 5+5=10



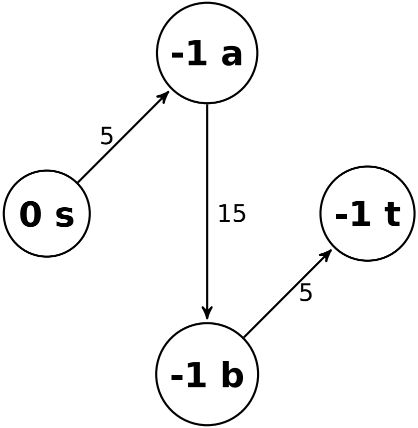
Dinitz's Algorithm – How it works

4. The capacities get updated, and the min flow is added to total flow



Dinitz's Algorithm – How it works

5 The procedure iterates for updated, residual graph



Dinitz's Algorithm – How it works

5. The procedure iterates for updated, residual graph

0	1	2	3	Path order
s	a	b	t	min{10,5,15 }=5

- The flow is 10+5=15

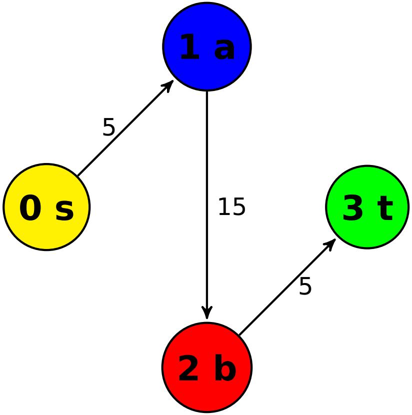


Table of Contents

1 Recap: Max Flow Problem

2 Dinitz's Algorithm

3 Sequential implementation and Evaluation

4 Parallelization Ideas

5 Challenges and Outlook

C - The Programming Language of Choice

- C is used as the programming language of choice
  - Fine-grained options for performance tuning and native bindings for OpenMPI
  - Many POSIX-compatible libraries available for further architectural tailoring

- Main challenge: robust and scalable memory management
  - Point of ongoing optimisation

Implementing Dinitz's Algorithm in C

- In the 1970s, B. Cherkassy proposed good coding practices for graph algos
- For Dinitz's Algorithm, some of these practices were followed
  - No level graph is built, manage level (aka label) array where:
    - $level[v]$ = level of vertex v
  - Our DFS-implementation ignores saturated edges and equally levelled edges
  - No edge removals take place

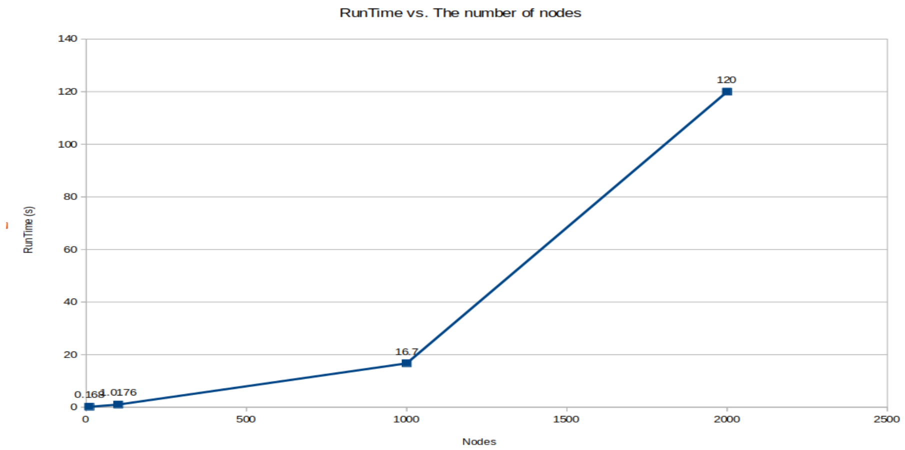
Implementing Dinitz's Algorithm in C

- Object-like and edge-based approach to graph processing
  - Using an adjacency list for storing the edges of each vertex
  - Adjacency matrix transformation possible for parallel approaches
- Implementing BFS using TAIL queue macros provided by the BSD sys library
  - No other external libraries needed right now, might change finally

Implementing Dinitz's Algorithm in C

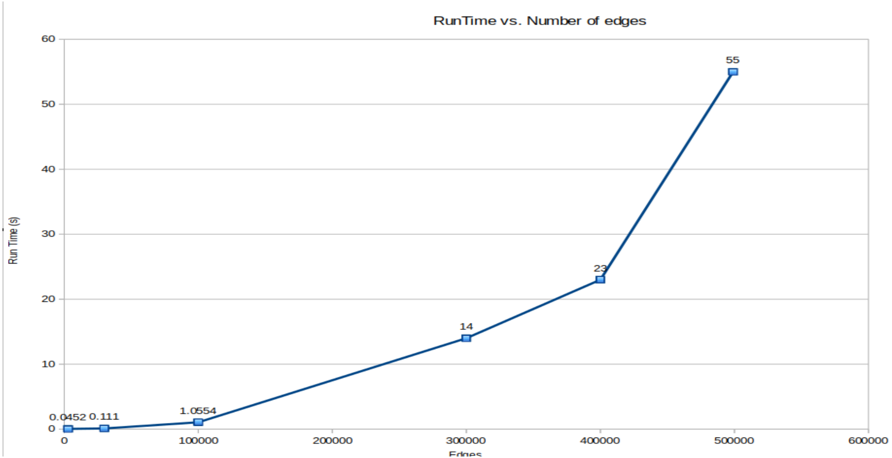
- Right now, graphs are parsed as .csv files
  - But: not a robust way of processing graphs comprising of millions of nodes
  - Moving forward, we'll switch to .json files for storing our graphs

Preliminary Evaluation of Sequential Implementation



Runtime scaling over an increasing number of vertices

Preliminary Evaluation of Sequential Implementation



Runtime scaling over an increasing number of edges (base: 1000 vertices)

Table of Contents

1 Recap: Max Flow Problem

2 Dinitz's Algorithm

3 Sequential implementation and Evaluation

4 Parallelization Ideas

5 Challenges and Outlook

Parallelization Ideas

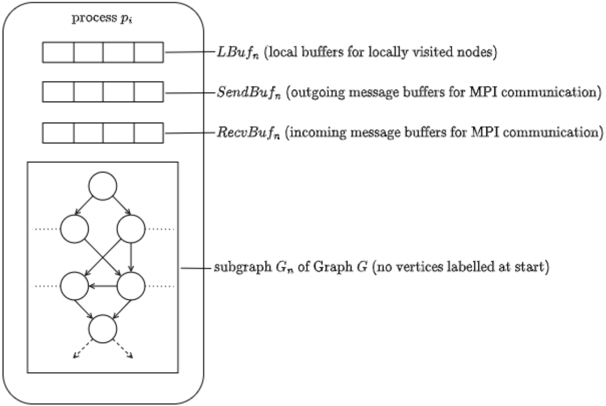
- Dinitz’s algo is (on a very basic level) a modified combination of BFS & DFS
  - Extra parameters make parallelising DFS difficult
  - However, BFS can be parallelised quite nicely

Parallelization Ideas – BFS

- BFS can be parallelized in a variety of ways, depending on the target graph
  - Classical top-down approach for low-diameter (sparse) graphs
  - Bottom-up approach for high-diameter graphs (children search for parents)
  - Dynamic optimization algo combining TD/BU depending on the graph
- Partitioning has to be done to allow for parallelization
  - 1D (which we used) and 2D (splitting adjacency matrix among CPUs)

Parallelization Ideas – BFS

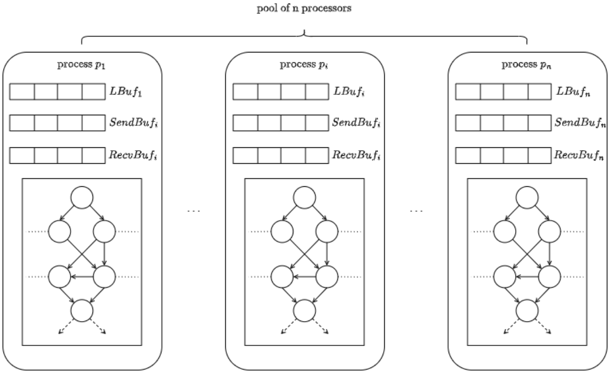
First off: Schematics of a process



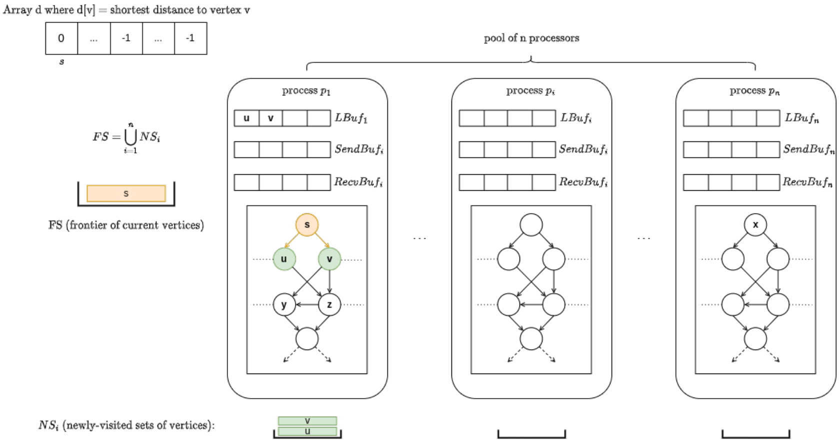
Parallelization Ideas – BFS

INPUT: n subgraphs G_n of $G(V, E)$, source vertex s, sink vertex t

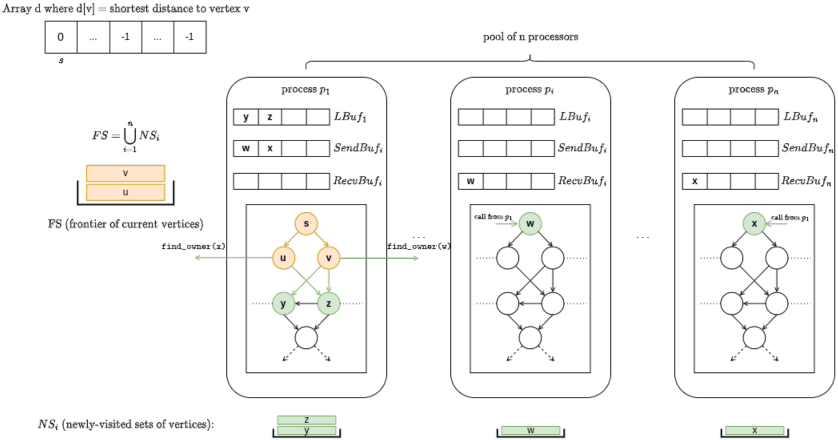
OUTPUT: Truth value: Is t reachable from s? ($\overrightarrow{st} \neq -1$)



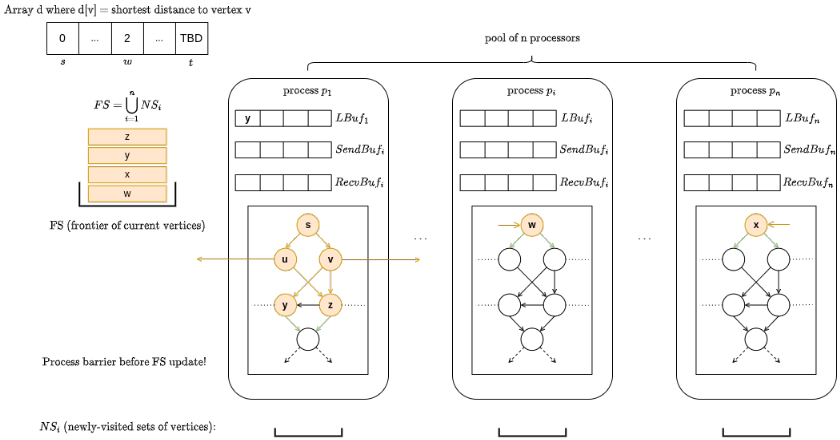
Parallelization Ideas – BFS



Parallelization Ideas – BFS



Parallelization Ideas – BFS



Parallelization ideas – BFS

- As seen in the model, each vertex with a shared neighbour owner processes
  - Vertices might be checked more than once (possibly by all the processes!)
  - Here using the bottom-up approach mentioned can mitigate revisiting vertices
  - Each process can also benefit from multithreading via OpenMP or Pthreads
- 2D partitioning is more scalable overall
  - Adjacency matrices are very space-efficient (basically bitmaps)
  - Libraries such as GSL (GNU Scientific Library) offer efficient linear algebra ops
- BUT: 1D partitioning is easier to implement quickly and correctly

Parallelization Ideas – DFS

- DFS is described as a nightmare for parallel processing"
- We can parallelize finding the shortest augmenting paths
  - Updating and calculation of minimal flow capacity cannot be done in parallel
- Not as straightforward as BFS since flow capacities actually matter here
  - Can’t visit edges in parallel if we don’t know about their residual flow capacity

Table of Contents

1 Recap: Max Flow Problem

2 Dinitz's Algorithm

3 Sequential implementation and Evaluation

4 Parallelization Ideas

5 Challenges and Outlook

Challenges with constructing Big Graphs

- An average graph with 1 million nodes can have 300 billion(3×10^{11}) edges
- Can't be held by each CPU in local memory
  - partitioning and memory management are important
- Not all graphs are processed equally
  - proper data structures (bitmaps, adjacency matrices, etc.) are important
- To address this problem we will use partitioning and dynamic optimization

What we are still working on

- Refining memory management to make memory accesses more local
- Parallelising each step of the algorithm at least somewhat meaningfully
  - find a way to combine flow management and existing ideas for parallel DFS
- Evaluate performance more rigorously
  - More data, relevant graph types (e.g. small-world graphs)
- Implementing a robust way of generating and processing large graphs

What we might tackle if there is time

- Implementing a 2D partitioned approach with adjacency matrices
- Further increasing the performance of our parallelised code
  - BFS: Implement dynamic optimisation combining top-down/bottom-up
  - Decreasing IO overhead through the use of multithreading for sequential parts
  - Maybe also using an external memory algo for more memory access locality
- Using OpenMP to decrease communication overhead on multicore systems

References

Beamer, S., K. Asanović, and D. Patterson. "Direction-Optimizing Breadth-First Search". In: *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. SC '12. Salt Lake City, Utah: IEEE Computer Society Press, 2012. ISBN: 9781467308045.

Beamer, S., A. Buluç, et al. *Distributed Memory Breadth-First Search Revisited: Enabling Bottom-Up Search*. Tech. rep. UCB/EECS-2013-2. EECS Department, University of California, Berkeley, Jan. 2013. URL: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2013/EECS-2013-2.html.

Buluç, A. and K. Madduri. "Parallel Breadth-First Search on Distributed Memory Systems". In: *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. SC '11. Seattle, Washington: Association for Computing Machinery, 2011. ISBN: 9781450307710. DOI: 10.1145/2063384.2063471. URL: https://doi.org/10.1145/2063384.2063471.

Dinitz, E. A. "Algorithm for solution of a problem of maximum flow in a network with power estimation". In: *Doklady Akademii Nauk SSSR* 11 (1970), pp. 1277–1280.

Goldreich, O., A. L. Rosenberg, and A. L. Selman. *Theoretical Computer Science: Essays in Memory of Shimon Even*. Springer, pp. 218–240.

Kumar, R. and V. N. Vipin. "Parallel depth first search. Part I. Implementation". In: *International Journal of Parallel Programming* 16 (Dec. 1987), pp. 479–499. URL: https://rdcu.be/dgUPq.

References

- https://en.wikipedia.org/wiki/Maximum_flow_problem
- https://www.geeksforgeeks.org/dinics-algorithm-maximum-flow/
- https://gitlab.gwdg.de/jonas.hafermas/max_flow_hpc
- https://www.researchgate.net/publication/331967163_Airline_Scheduling_with_Max_Flow_algorithm/citation/download
- http://worldcomp-proceedings.com/proc/p2013/PDP3767.pdf