Parallel Graph Algorithms for GPU

Radek Cichra

Faculty of Nuclear Sciences and Physical Engineering
Czech Technical University in Prague
Supervisor: doc. Ing. Tomáš Oberhuber, Ph.D.

August 28th, 2024

Table of Contents

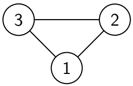
Graphs

Graph $\approx$ Mathematical tool describing a system using relationships between its parts $\approx$ Cities with highways between them

Graph

Graph is an ordered set $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ where

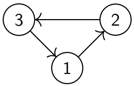
- $\mathbf{V}$ is a set of vertices
- $\mathbf{E}$ is a set of **edges**
- **edge** is a pair of vertices $\in \mathbf{V}$



Undirected Graph

$\mathbf{V} = \{1, 2, 3\}$

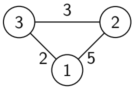
$\mathbf{E} = (\{1, 2\}, \{1, 3\}, \{2, 3\})$



Directed Graph

$\mathbf{V} = \{1, 2, 3\}$

$\mathbf{E} = ((1, 2), (2, 3), (3, 1))$



Weighted Graph

$\mathbf{V} = \{1, 2, 3\}$

$\mathbf{E} = (\{1, 2\}, \{1, 3\}, \{2, 3\})$

$w : \mathbf{E} \to \mathbb{R}^+$

Maximal Independent Set

Defined on an undirected graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$.

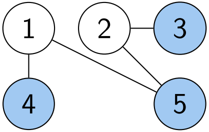
Independent Set

Independent set $\mathbf{IS} \subseteq \mathbf{V}$ is a set of vertices where no two vertices are directly connected by an edge.

$$(\forall u, v \in \mathbf{IS})(\{u, v\} \notin \mathbf{E})$$

Maximal Independent Set

Maximal independent set $\mathbf{MIS} \subseteq \mathbf{V}$ is an independent set to which no vertices can be added anymore.



Connected Components

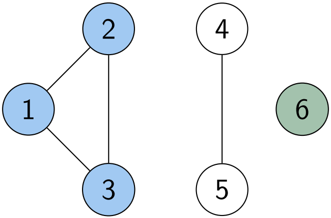
Defined on an undirected graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$.

Connected Component

Connected Component $\mathbf{CC} \subseteq \mathbf{V}$ is a set of vertices that are all *mutually reachable* - ie. there exists a sequence of edges in **E** connecting them.

$$(\forall u, v \in \mathbf{CC})(\exists \text{ a sequence of edges } \{u, x_1\}, \{x_1, x_2\} \ldots \{x_n, v\}) \in \mathbf{E})$$

We also require **CC** to be maximal - all reachable vertices must be added to **CC**.



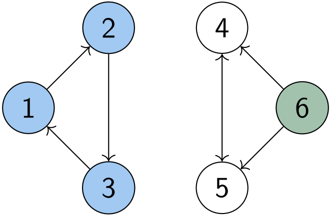
Strongly Connected Components

Defined on a directed graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$.

Strongly Connected Component

Strongly connected component $\mathbf{SCC} \subseteq \mathbf{V}$ is a set of vertices that are all mutually reachable *with respect to edge directions*.

We also require **SCC** to be maximal - all mutually reachable vertices must be added to **SCC**.



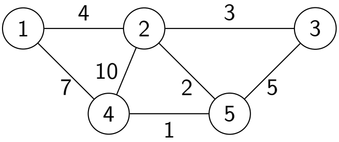
Minimal Spanning Tree

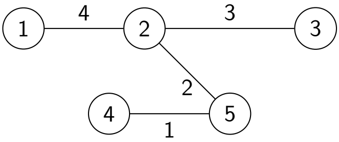
Defined on an undirected weighted graph $\mathbf{G} = (\mathbf{V}, \mathbf{E},\ w : \mathbf{E} \to \mathbb{R}^+)$.

Minimal Spanning Tree

Minimal spanning tree is an undirected weighted subgraph of **G** that:

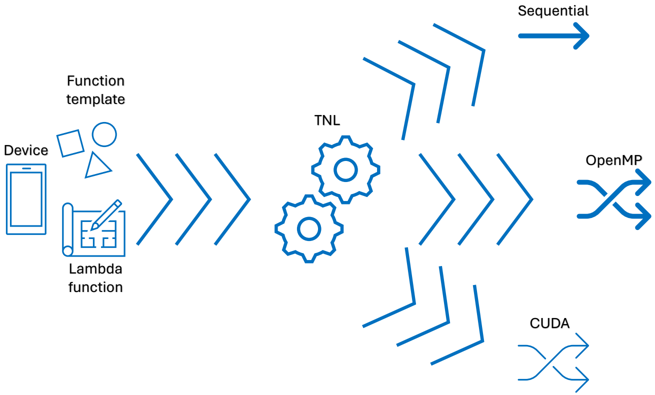
1. is a *tree*
2. contains all vertices of the original graph (ie. *spanning*)
3. has the smallest total sum of edge weights out of all spanning trees (ie. *minimal*)





TNL

Open-source library and a flexible toolkit for the developement of numerical solvers and HPC algorithms. Supports parallelism on both GPU (CUDA, HIP, or ROCm) and CPU (OpenMP).



$\Longrightarrow$ TNL allows for writing device (CPU or GPU) *independent* code.*

Project Objectives

Implement, test, and benchmark solutions to the following graph problems:

1. **Find maximal independent set** of an undirected graph.
2. **Find connected components** of an undirected graph.
3. **Find strongly connected components** of a directed graph.
4. **Find minimal spanning tree** of an undirected weighted graph.

- **Implementation** - TNL
- **(Unit) Testing** - Gtest, General check functions
- **Benchmarking** - CPU $\times$ GPU, Methods to improve GPU

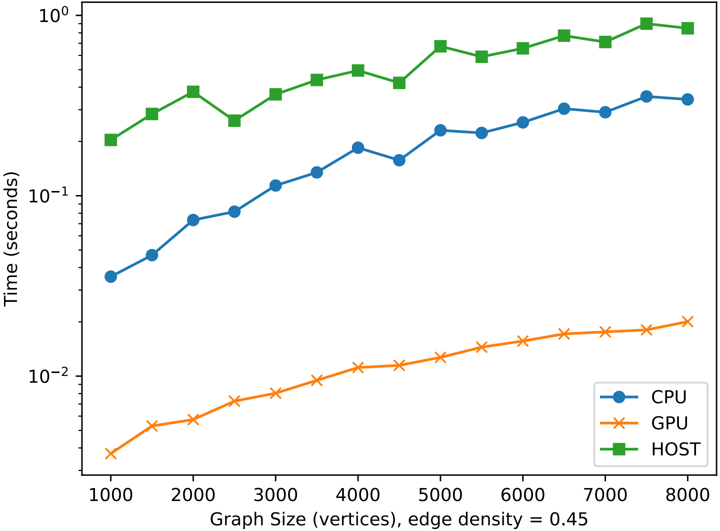
MIS - Luby's algorithm

Algorithm utilizing random subset selection of graph vertices favoring *lonely* vertices.[3]

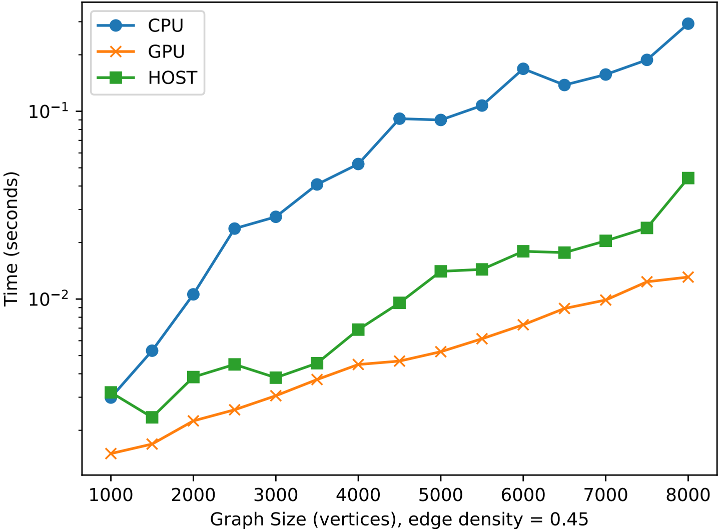
Algorithm 1 Luby on an undirected graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$

Initialize MIS $= \emptyset$
while $V \neq \emptyset$ **do**
    Select subset $S \subseteq V$, $n \in V$ is selected with the probability $\frac{1}{2d(n)}$
    for all $\{u, v\} \in E$ **do**
        if both endpoints are in S **then**
            Remove endpoints with $< d$ and break ties arbitrarily
        end if
    end for
    MIS $=$ MIS $\cup$ S
    Remove S and its neighbours from V
end while
return MIS

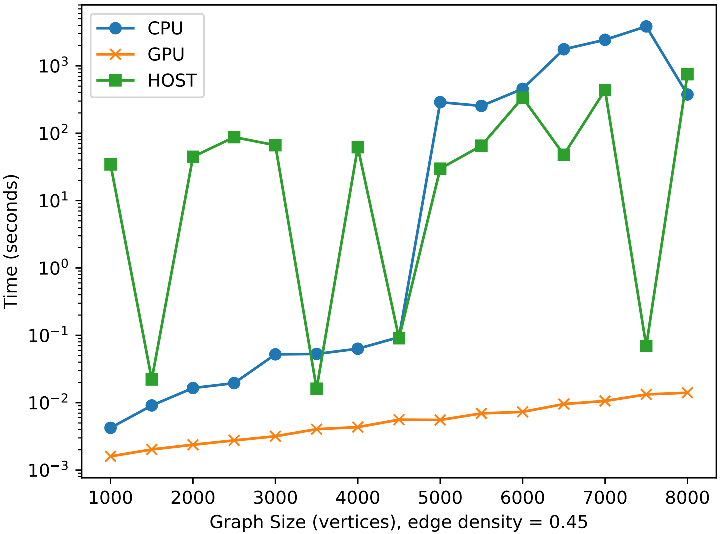
MIS_BASE - Benchmark



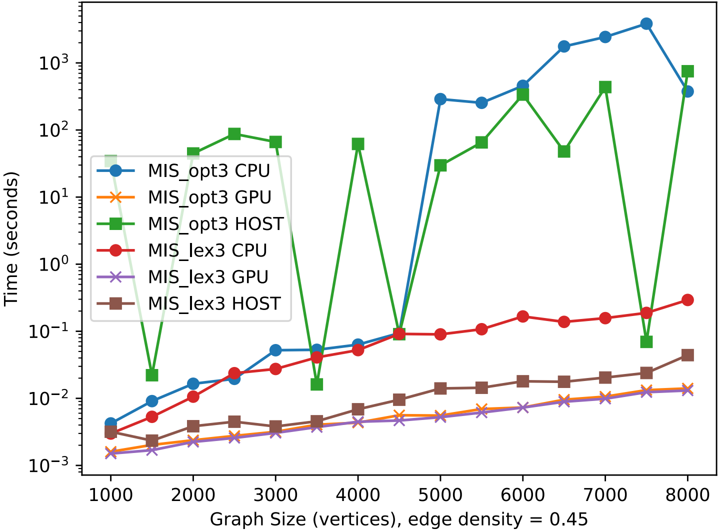
MIS_LEX3 - Benchmark



MIS_OPT3 - Benchmark



Comparison



Time (seconds)
10^3
10^2
10^1
10^0
10^{-1}
10^{-2}
10^{-3}
1000
2000
3000
4000
5000
6000
7000
8000
Graph Size (vertices), edge density = 0.45
MIS_opt3 CPU
MIS_opt3 GPU
MIS_opt3 HOST
MIS_lex3 CPU
MIS_lex3 GPU
MIS_lex3 HOST

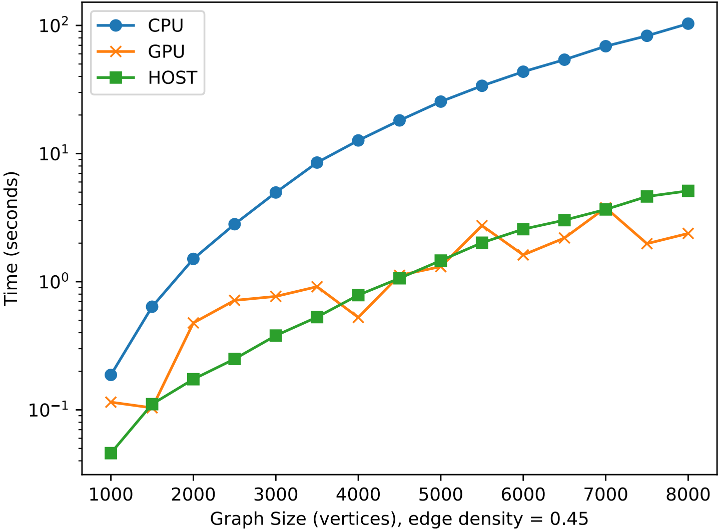
CC - FastSV

Algorithm that finds connected components by initially assuming every vertex to be a rooted tree, then repeatedly hooking these trees with suitable edges between them.[4]

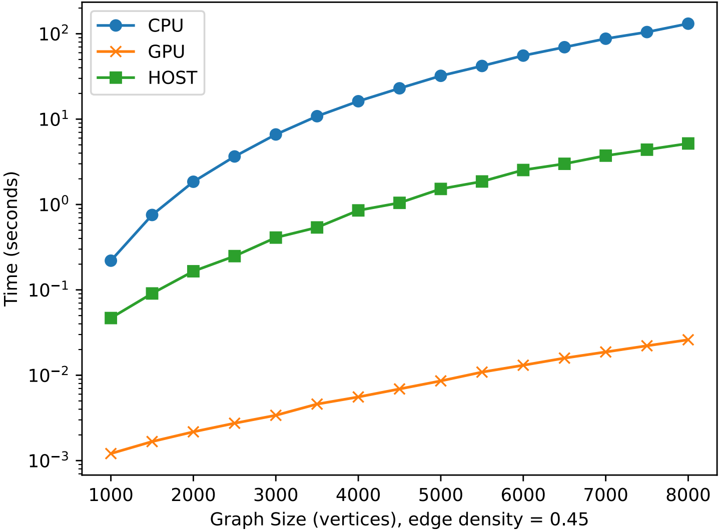
Algorithm 2 FastSV on an undirected graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$

Initialize vectors **p**, **gp** to track parents and grandparents of each vertex
Assume $\forall n \in V$ to be rooted tree $\rightarrow \mathbf{p_n} = \mathbf{gp_n} = \mathbf{n}$
repeat
    for all $\{u, v\} \in E$ (in parallel) **do**
        if $\mathbf{gp_u} > \mathbf{gp_v}$ **then**
            set $\mathbf{p_{p_u}} = \mathbf{gp_v}$
            set $\mathbf{p_u} = \mathbf{gp_v}$
        end if
    end for
    for all $n \in V$ (in parallel) **do**
        if $\mathbf{p_n} > \mathbf{gp_n}$ **then**
            set $\mathbf{p_n} = \mathbf{gp_n}$
        end if
    end for
    for all $n \in V$ (in parallel) **do**
        set $\mathbf{gp_n} = \mathbf{p_{p_n}}$
    end for
until vector **gp** stops changing
return **p**

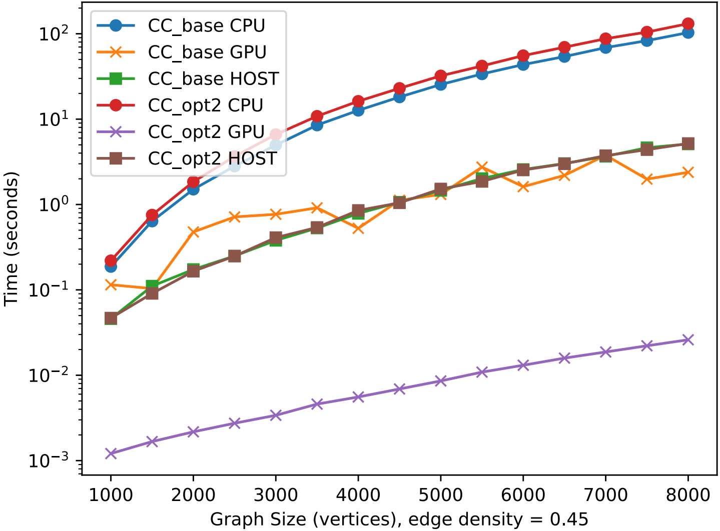
CC_BASE - Benchmark



CC_OPT2 - Benchmark



Comparison



CC_base CPU
CC_base GPU
CC_base HOST
CC_opt2 CPU
CC_opt2 GPU
CC_opt2 HOST
10^2
10^1
10^0
10^{-1}
10^{-2}
10^{-3}
Time (seconds)
1000
2000
3000
4000
5000
6000
7000
8000
Graph Size (vertices), edge density = 0.45

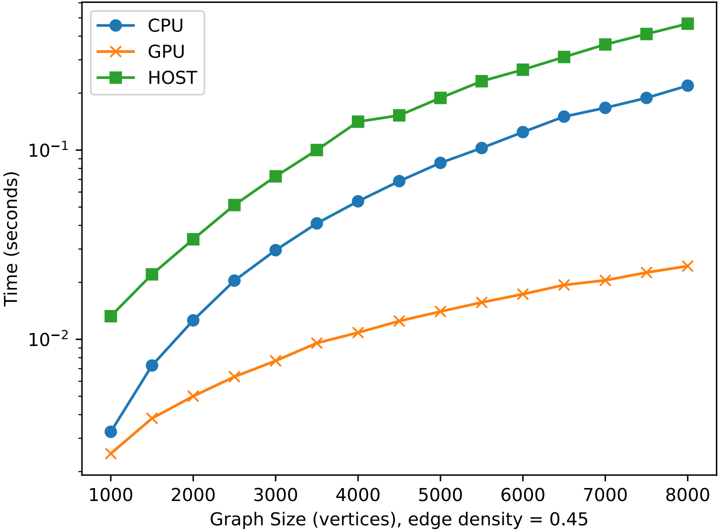
SCC - (C)FHP algorithm

Initially a Divide-and-Conquer algorithm which selects a **pivot** vertex, then finds SCC by intersecting its *predecessors* (ie. vertices, from which the pivot is reachable) and *descendants* (ie. vertices reachable from the pivot).[2]

Algorithm 3 CFHP on a directed graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$

```
initialize vector SCCs and color c to track colored vertices
repeat
    select uncolored vertex p ∈ V
    find predecessors and descendants of p → PRED, DESC
    for all u ∈ V found in both PRED and DESC do
        color SCCs_u with c
    end for
    increase c
until ∀v ∈ SCCs are colored
return SCCs
```

SCC_BASE - Benchmark



MST - Algebraic MST algorithm

Algorithm to calculate the edge weight sum of **Minimal Spanning Tree**, as well as the tree itself.

Initially views the graph as a collection of rooted trees. Then for stars (trees of height ≤ 2), a minimal outgoing edge is found and propagated to the root. Star roots are then hooked with these edges, which are added to MST. Lastly, height of all trees is reduced, so that they may become stars in future iterations.

We repeat steps above, until all trees stop changing between iterations.[1]

Conclusion

To reiterate, this thesis encompasses:

1. Getting acquainted with the basics of GPU programming using CUDA and TNL.
2. Implementing fully working solutions to 3 out of the 4 aforementioned graph problems. For the MST problem, a partially working implementation was created.
3. Unit testing all implementations, as well as creating general check functions for 2 of them (MIS and CC).
4. Explaining all implementations, as well as providing context in the form of *sequential-only* counterparts to each algorithm.
5. Benchmarking performance across hardware devices and different versions of implementations.

Thank you for your attention

Citations

[1] Tim Baer, Raghavendra Kanakagiri, and Edgar Solomonik. Parallel minimum spanning forest computation using sparse matrix kernels. *Proceedings of the 2022 SIAM Conference on Parallel Processing for Scientific Computing (PP)*, page 72–83, Jan 2022.

[2] Don Coppersmith, Lisa Fleischer, Bruce Hendrickson, and Ali Pinar. *A divide-and-conquer algorithm for identifying strongly connected components*, Jan 2006.

[3] Michael Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM Journal on Computing*, 15(4):1036–1053, 1986.

[4] Yongzhe Zhang, Ariful Azad, and Aydın Buluç. Parallel algorithms for finding connected components using linear algebra. *Journal of Parallel and Distributed Computing*, 144:14–27, May 2020.

Q1

Q: *Byl algoritmus FastSV implementován správně; viz. předchozí část posudku "Odborná úroveň"?*

A: Ano.

Je pravda, že ve článku Zhang-Azad-Hu je pseudokód (Algorithm 2), který obsahuje oddělené cykly pro *hooking* operace. V práci je ale zároveň také citován novější článek Zhang-Azad-Buluç (v práci citace ZAB20), ve kterém je algoritmus (Algorithm 3) se společným *hooking phase* cyklem.

Q1

Algorithm 2 The FastSV algorithm. **Input:** $G(V, E)$. **Output:** The parent vector f

procedure FASTSV(V, E)
  for every vertex $u \in V$ **do**
    $f[u], f_{next}[u] \leftarrow u$
  repeat
    ▷ Step 1: Stochastic hooking
    for every $(u, v) \in E$ **do in parallel**
      $f_{next}[f[u]] \xleftarrow{\min} f[f[v]]$
    ▷ Step 2: Aggressive hooking
    for every $(u, v) \in E$ **do in parallel**
      $f_{next}[u] \xleftarrow{\min} f[f[v]]$
    ▷ Step 3: Shortcutting
    for every $u \in V$ **do in parallel**
      $f_{next}[u] \xleftarrow{\min} f[f[u]]$
    $f \leftarrow f_{next}$
  until $f[f]$ remains unchanged

Algorithm 3 The skeleton of the FastSV algorithm. **Input:** a graph $G(V, E)$. **Output:** The parent vector f

procedure FASTSV$(G(V, E))$
  for every vertex $u \in V$ **do in parallel** ▷ Initialize
    $f[u], gf[u] \leftarrow v$
  repeat
    ▷ Step 1: Hooking phase
    for every edge $\{u, v\}$ in E **do in parallel**
      if $gf[u] > gf[v]$ **then**
        $f[f[u]] \leftarrow gf[v]$
        $f[u] \leftarrow gf[v]$
    ▷ Step 2: Shortcutting
    for every vertex u in V **do in parallel**
      if $f[u] > gf[u]$ **then**
        $f[u] \leftarrow gf[u]$
    ▷ Step 3: Calculate grandparent
    for every vertex u in V **do in parallel**
      $gf[u] \leftarrow f[f[u]]$
  until gf remains unchanged
  return f

V článku je řečeno "*If the condition gf[u] > gf[v] is satisfied, we hook* ***both*** *f[u] and u onto gf[v], v's grandparent in the previous iteration. Here, hooking f[u] to gf[v] corresponds to the stochastic hooking and hooking u to gf[v] corresponds to the aggressive hooking.*"

Q2

Q: *Jaký je rozdíl mezi "stochastic hooking" a "agressive hooking"? Je za volbou těchto názvů nějaká motivace?*

A: *stochastic hooking* je hlavní krok posouvající algoritmus ke správnému výsledku, tj. do situace, kde každá souvislá komponenta je reprezentovaná hvězdou. *agressive hooking* se snaží maximálně urychlit algoritmus snižováním výšky stromů, což z něho dělá takový druhý *shortcutting* krok v rámci stejné iterace.

Název *agressive* mluví vcelku za sebe. Název *stochastic* existuje asi kvůli kontrastu oproti podmínkám pro *hooking* v původním SV algoritmu. Něco v tomto duchu je řečeno v již zmíněném článku Zhang-Azad-Hu.

Q3

Q: *Jaké hodnoty ve vaší definici "Adjacency Matrix" nabývají prvky této matice v případě grafu bez vah?*

A: Pokud se jedná o graf bez vah, lze jakýkoliv nenulový prvek vnímat jako indikaci toho, že jemu odpovídající hrana je v grafu. V praxi (implementaci) s prvky pracujeme jako s datovým typem `bool`, tedy algoritmy pro MIS, CC, SCC jsou schopné zpracovat i vážené grafy, přičemž hodnota váhy nehraje roli.

Q4

Q: *Co znamená "mutually reachable"?*

A: V kontextu práce pojem definujeme tak, že v grafu jsou 2 vrcholy *u,v* vzájemně dosažitelné (*mutually reachable*) pouze tehdy, pokud v něm existují sekvence hran $((u, x_1)(x_1, x_2) \dots (x_{n-1}, x_n)(x_n, v))$ a $((v, y_1)(y_1, y_2) \dots (y_{m-1}, y_m)(y_m, u))$, které je spojují. Navíc definujeme každý vrchol jako vzájemně dosažitelný sám se sebou.

(Po zpětné kontrole textu práce je třeba podotknout, že v ní je tato definice neúplná. Naštěstí se jedná pouze o chybu textu, ve zbytku práce a všech úvahách počítáme s kompletní definicí)

Q5

Q: *Porovnával jste nějak u variant pro MIS (MIS_Base, MIS_lex3, MIS_opt3) krom času také kardinalitu příslušným algoritmem nalezené nezávislé množiny pro stejný vstup...?*

A: Bohužel neporovnával, ale musím souhlasit, že se jedná o velice vhodnou věc, kterou by bylo dobré porovnat. Z toho, co jsem četl/pochytil, je zpravidla snaha kardinalitu nalezených MIS maximalizovat - jedná se tedy o významnou vlastnost MIS, kterou by bylo dobré sledovat.

Po úvaze jsem došel k závěru, že benchmarkovací procedura by měla naštěstí jít velmi jednodušše pro tyto účely modifikovat. Je škoda, že jsem na toto nepomyslel během práce.