Maximum Independent Set Algorithms on the GPU

Eric Waldherr

March 7, 2025

4231309

Bachelor Thesis

at

Algorithm Engineering Group Heidelberg
Heidelberg University

Supervisor:
Univ.-Prof. PD. Dr. rer. nat. Christian Schulz

Co-Supervisor:
Ernestine Großmann

Acknowledgments

First of all, I would like to thank Prof. Dr. Christian Schulz for allowing me to work on this project. Additionally, I would like to thank Ernestine Großman and Prof. Dr. Christian Schulz for addressing all my questions with patience and giving me valuable advice throughout this project. Furthermore, I acknowledge the support provided by the state of Baden-Württemberg through bwHPC. Finally, I thank my parents and my girlfriend for the unwavering support I received from them on every step of the way to my bachelor's degree.

Hiermit versichere ich, dass ich die Arbeit selbst verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und wörtlich oder inhaltlich aus fremden Werken Übernommenes als fremd kenntlich gemacht habe. Ferner versichere ich, dass die übermittelte elektronische Version in Inhalt und Wortlaut mit der gedruckten Version meiner Arbeit vollständig übereinstimmt. Ich bin einverstanden, dass diese elektronische Fassung universitätsintern anhand einer Plagiatssoftware auf Plagiate überprüft wird.

Heidelberg, March 7, 2025

Eric Waldherr

Abstract

The *maximum independent set problem* describes the task of computing a set of vertices of maximum cardinality such that no vertices of the set are adjacent. Since the problem is NP-hard, we focus on heuristic algorithms in this thesis. There are multiple closely related problems to the maximum independent set problem, including the maximum clique and the minimum vertex cover problem. Therefore, there exist applications for the problem in various fields of computer science, for instance, to computer graphics and route planning.

In this thesis, we develop GPU-accelerated algorithms to address the maximum independent set problem. Furthermore, we apply known improvements to our algorithms, leading to the design of our three algorithms: LUMIS, DUMIS, and IDUMIS. LUMIS is based on Luby's Monte Carlo algorithm and forms the basis of our work. Mainly, our improved algorithm DUMIS prioritizes low-degree over high-degree vertices. This concept improves the runtime and quality of our algorithm. Moreover, IDUMIS improves an initial solution over a given time limit by applying randomized adjustments to the preceding solution.

We investigated the performance of our algorithms through numerous experiments by comparing them with recently published algorithms in the field. We show that IDUMIS can outperform the MMWIS algorithm for short time limits. In addition, DUMIS is capable of producing larger independent sets than the similarly designed algorithm ECL-MIS, while requiring less execution time for large graph instances. Finally, a reflection on the results of this thesis is provided, along with input for potential future work.

Contents

CHAPTER 1

Introduction

1.1 Motivation

The maximum independent set (MIS) problem describes the issue of finding a set of vertices in a graph such that no vertices in that set are adjacent. Moreover, the objective is to find the largest possible set that fulfills this requirement. There are many other problems in graph theory that are closely related to the MIS problem, such as the minimum vertex cover, the maximum clique [46], the maximum matching, and the maximal set packing problem [6]. Consequently, advances in algorithms for solving the MIS problem can have a significant impact on multiple fields of computer science. In addition, there are various applications for the MIS problem in many fields of interest, including computer graphics [15], route planning [17], and RNA kinetics [36]. Thus, there exist numerous different approaches to solve the MIS problem (for instance, see [7, 20, 42, 43]).

The MIS problem is known to be NP-hard [46]. Therefore, heuristic algorithms are a common approach to the problem. The objective of such algorithms is to identify near-optimal solutions in polynomial time. Consequently, these algorithms do not require exponential execution times in contrast to exact MIS algorithms. In this thesis, we focus on heuristic algorithms.

Furthermore, graphics processing units (GPUs) have emerged as an acceleration tool in the field of graph processing. The massive degree of parallelism that GPUs provide has the potential to significantly accelerate graph processing algorithms [33]. Therefore, the main objective of this work was to design GPU-accelerated MIS algorithms that provide high-quality maximal independent sets. Moreover, our goal is to achieve low execution times due to the usage of GPUs.

1.2 Our Contribution

There are numerous different approaches to solving the MIS problem effectively. Recently, studies have been published that apply machine learning and graph neural networks to the issue [42, 44]. Other state-of-the-art algorithms are based on local search [20, 28, 41]. Despite the various advantages of these approaches, we engineered an algorithm based on Luby's algorithm [7], which was among the first heuristic algorithms solving the problem in parallel. This decision was made based on the assumption that Luby's algorithm is more suitable for GPU algorithms. Therefore, we designed our algorithm to take advantage of the large degree of parallelism of GPUs to significantly accelerate execution time. The result is our proposed algorithm, LUMIS. Additionally, we introduce known improvements to Luby's algorithm to improve the quality of the results. Eventually, this led to the design of our improved algorithm, DUMIS. Both of these algorithms compute a solution once. Contrary to other algorithms, these algorithms do not improve their initial solutions. Therefore, we designed IDUMIS, an algorithm that runs multiple rounds of DUMIS. Furthermore, IDUMIS improves the solution over a given time limit by applying randomized adjustments to the preceding solution.

Moreover, there are different reductions that can be applied to the MIS problem [29, 35]. Therefore, we included a portion of these reductions in our approach. We designed the reductions for parallel execution on GPUs. Our experiments show, that the reductions can significantly increase the quality of a MIS algorithm.
In order to investigate the performance of the proposed algorithms, we conducted experiments on various graph instances. Furthermore, we evaluated the results and compared our algorithms to recently published MIS algorithms. We found that the convergence time of our algorithm IDUMIS is faster than the convergence time of other MIS algorithms. Therefore, the proposed algorithms can outperform competing algorithms for applications with limited time available for computing a maximal independent set. Moreover, DUMIS is capable of producing larger sets than a comparable MIS GPU algorithm, while requiring less execution time on large graph instances.

1.3 Structure

The remainder of this thesis is structured as follows: First, the fundamental and problem-related definitions required to understand the following contents are given in Chapter 2. Subsequently, different previously developed approaches to the MIS problem are presented briefly in Chapter 3. Chapter 4 outlines the proposed algorithms and illustrates how they are implemented in detail. Afterwards, in Chapter 5, reductions for the MIS problem that were implemented are described and explained. Thereafter, the experiments conducted in the course of this work are presented in Chapter 6. Lastly, the results of the thesis and potential future work are discussed in Chapter 7.

CHAPTER 2

Fundamentals

2.1 General Definitions

Graphs: Let $G = (V, E)$ be a *graph* with a set of *vertices* V and a set of *edges* $E \subseteq V \times V$. The number of vertices in G is denoted by $n = |V|$ and the number of edges by $m = |E|$. If G is *undirected*, each edge $(u, v) \in E, u, v \in V$ connects u to v and vice versa. Therefore, (u, v) and (v, u) refer to the same edge in an undirected graph.

IDs: To distinguish vertices from each other, each vertex $v \in V$ is assigned a *vertex ID*. A vertex v with ID i is denoted as v_i. Additionally, v_i can be referred to as vertex i.

Self-loops: A *self-loop* is an edge that goes from a vertex $v \in V$ to itself: $(v, v) \in E$. In this thesis, we only consider graphs that do not contain self-loops. Therefore, all self-loops were removed from the graph instances used in the experiments.

Neighborhoods: Each vertex $v \in V$ in the graph has its own *neighborhood*. It is defined as $N(v) = \{u \in V : (u, v) \in E\}$. In addition, $N[v] = N(v) \cup \{v\}$ denotes the *closed neighborhood* of vertex v.

Degrees: The *degree* of a vertex is an important property, especially in the context of the MIS problem. The degree of a vertex $v \in V$ in an undirected graph is defined as $deg(v) = |N(v)|$. Including vertices with high degrees is disadvantageous because it is unlikely that such vertices are in large independent sets [28, 32]. Therefore, this property is essential for finding large maximal independent sets.

Cliques: A *clique* is a set of vertices. In a clique $C \subseteq V$, each vertex is adjacent to every other vertex of the clique: $\forall u, v \in C$ with $u \neq v : (u, v) \in E$.

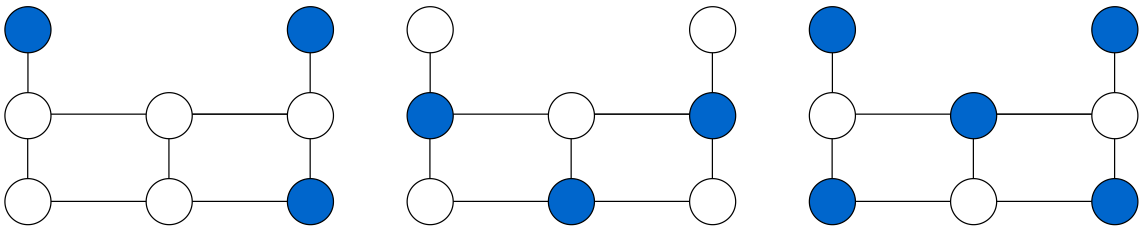


Figure 2.1: Illustration of different categories of independent sets.

2.2 Maximum Independent Set Problem

An *independent set* of an undirected graph G is defined as a subset of vertices $I \subseteq V$ such that no two vertices in I are adjacent: $\forall u, v \in I : (u, v) \notin E$. Furthermore, an independent set I of G is *maximal* if $\nexists v \in V, v \notin I : I \cup \{v\}$ is independent. In other words, an independent set is maximal if there is no vertex in the graph that can be added to the set without breaking its independence constraint. I is a *maximum* independent set if it is a maximal independent set with the largest cardinality achievable: $\nexists J \subseteq V : J$ is independent $\wedge\ |I| < |J|$. The cardinality of a maximum independent set of the graph G is denoted as $|MIS(G)|$. Figure 2.1 illustrates the difference between the categories of independent sets in an example graph. In this example, vertices marked in blue are part of the independent set.

2.3 Data Reductions

Data reductions are rules or algorithms that can be applied to the input graph $G = (V, E)$. The objective of reductions is to decrease the size of the original graph and form a reduced graph $G' = (V', E')$: $|V'| < |V|$. In this thesis, we use two types of reductions. On one hand, there are reductions that remove vertices from the original graph. This can be done by proving that there exists a MIS that does not contain these specific vertices. On the other hand, there are reductions that add vertices to the independent set because it can be proven that there is a MIS that contains these vertices. Subsequently, the neighborhoods of such vertices can be removed from the graph. This process simplifies the input graph and can potentially increase the quality of MIS algorithms.

CHAPTER 3

Related Work

In this chapter, we present an overview of different approaches to the MIS problem. First, we give a brief overview of exact MIS algorithms. Thereafter, we focus on heuristic algorithms, especially parallel algorithms that inspired our work.

3.1 Exact Algorithms

Brach-and-bound algorithms are an approach for a variety of combinatorial optimization problems [5], especially for the MIS problem. The fundamental idea of branch-and-bound algorithms is to branch on a specific vertex $v \in V$. This process divides the problem into two subproblems, one with v in the independent set and one with v not part of the independent set. This way, the solution space is searched systematically, and an optimal solution can be identified. Furthermore, data reductions are an effective technique for MIS algorithms [27]. Therefore, various exact MIS algorithms use a *branch-and-reduce* approach. This combines the concept of branch-and-bound with reductions that are applied before each branching phase.

There exist numerous different branching strategies for such algorithms. Branching on the vertex with the highest degree and with a minimal number of edges connecting its neighbors is a frequently used method (for instance, see [21, 31]). Akiba and Iwata [27] propose a state-of-the-art algorithm that uses that strategy. More recently, Hespe et al. [37] propose a branching strategy that branches on vertices that hinder certain reductions. Therefore, data reductions can be applied to more vertices. In their study, they demonstrate that this strategy achieves equal or improved performance compared with highest degree branching.

3.2 Heuristic Algorithms

As noted above, the MIS problem is NP-hard. Thus, known exact algorithms addressing this problem require exponential time. Consequently, there are many heuristic algorithms that address this problem, which will be examined in the following section.

3.2.1 Local Search Algorithms

Local search is a common approach of different state-of-the-art MIS algorithms. The fundamental idea of local search is to modify and improve the current solution trough insertion, deletion, and swap operations. Andrade et al. [20] introduced one of the first successful heuristic local search MIS algorithms called ARW [28]. Their most interesting idea is called *(1,2) swap*. Essentially, *(1,2) swaps* are local situations that allow one vertex to be removed from the solution and two neighboring vertices to be added to the solution instead. This process can remarkably improve the solution over time.

After the introduction of ARW, there were multiple attempts to improve it. For example, Dahlum et al. [28] introduced OnlineMIS, an algorithm that accelerates the local search introduced by ARW. OnlineMIS performs exact reductions as well as inexact reduction rules to significantly reduce the size of the graph. Thereafter, the reduced graph is handled by a modified and efficient version of ARW. Furthermore, Chang et al. [30] introduce multiple MIS algorithms, especially a linear and a near-linear time algorithm. These are called LinearTime and NearLinear, respectively. They used these algorithms to successfully accelerate the ARW algorithm. Their Reducing-Peeling framework applies reduction rules on low-degree vertices (*reducing*). In addition, the vertex with the highest degree is removed from the graph (*peeling*). To date, these local search approaches represent state-of-the-art MIS algorithms [41].

3.2.2 Evolutionary Algorithms

There have been multiple evolutionary approaches to the MIS problem (for example, see [8, 25]). In this thesis, we focus on the evolutionary algorithm MMWIS [43]. MMWIS is an algorithm for the maximum *weighted* independent set problem (MWIS). However, it is capable of producing solutions for unweighted graphs. It was demonstrated that MMWIS can produce larger independent sets than other state-of-the-art MWIS algorithms. Therefore, we used MMWIS in the experiments.

MMWIS operates by repeating three fundamental steps: exact reduction, memetic search, and heuristic reduction. Since the reductions implemented in MMWIS are reductions of the MWIS problem, we do not examine them in further detail. These reductions are used to reduce the size of the graph by including vertices in the solution and removing their neighborhood from the graph. Thereafter, a *population* is generated consisting of multiple maximal independent sets. Subsequently, the individual solutions of the population

are combined, which creates new solutions, called *offsprings*. This process improves the quality of the population in multiple rounds. Finally, the best individual of the population is used to perform heuristic reductions. Vertices of this individual solution are then added to the independent set, and their neighborhoods are removed from the graph. Since this process reduces the size of the graph, a portion of the exact reductions can potentially be applied again. Therefore, the three steps are repeated until a given time limit is reached.

3.2.3 Parallel Algorithms

Despite being one of the first parallel algorithms for the MIS problem, the fundamental concepts of Luby's algorithm [7] are of great importance for modern MIS algorithms (for example, see [32, 34]). The algorithm assigns random numbers to each vertex in the graph during each iteration. Subsequently, a vertex is added to the independent set if it has the lowest random number assigned in its neighborhood. Afterwards, the entire closed neighborhood of each vertex added to the independent set is removed from the graph. This procedure is repeated until the residual graph is empty. This procedure computes maximal independent sets and can be executed in parallel effectively.

In their study, Imanaga et al. [34] present a parallel algorithm for GPUs that addresses the MIS problem. The algorithm they propose is based on the fundamental concepts of Luby's algorithm. In contrast to Luby's algorithm, their algorithm chooses a vertex if it has the lowest degree of its neighborhood. They only use random numbers, similar to Luby's approach, for tie-breaking situations that include multiple vertices with the same degree. Furthermore, they propose an algorithm that calculates a large number of solutions for each graph instance. After each computation of a solution, the algorithm removes vertices of the maximal independent set at random. Afterwards, these vertices and their neighborhoods are added back to the residual graph. Based on the resulting graph, a new solution is computed. During this process, the algorithm stores the best solution that was found overall. Furthermore, the authors demonstrate that their algorithm requires shorter execution times than other algorithms for GPUs or CPUs on random graphs. Additionally, their algorithm produced 22.6%-38.5% larger independent sets for large graph instances relative to the alternative algorithms employed in their experiment.

Burtscher et al. [32] propose an algorithm that is based on Luby's algorithm. They present improvements to the memory footprint, the execution time, and the quality of Luby's approach. In contrast to Luby's algorithm, the vertex with the *highest* number assigned in its neighborhood is included in the independent set. Moreover, instead of using purely random numbers, they define a function that assigns a high number, called priority, to low-degree vertices. Vertices with a high degree get a low priority assigned. Furthermore, this priority function is based on random numbers to differentiate the priorities of vertices that share the same degree. These improvements have a significant impact on the performance of their proposed algorithm. Therefore, the algorithm outperformed other MIS GPU implementations, both in speed and quality.

These studies illustrate the potential for implementing Luby's algorithm on GPUs. Furthermore, they show the significant impact of preferring low-degree vertices and modifying Luby's approach accordingly. These results are fundamental for our approach to the problem and have inspired our work.

CHAPTER 4

Proposed Algorithms

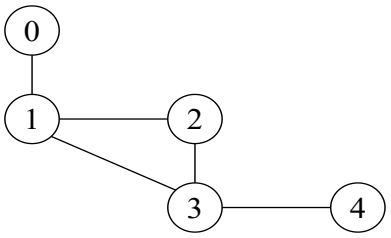
In this chapter, the framework used to develop GPU algorithms is examined. In addition, the data structure utilized by all proposed algorithms is explained. Lastly, the three algorithms developed in the course of this work are described in detail.

4.1 Development with Kokkos

The Kokkos framework [23, 38, 40] was employed to achieve the objective of solving the MIS problem on GPUs. This section focuses on the fundamentals of developing an algorithm with the Kokkos framework because this is crucial to understand the following algorithms in detail.

In general, the purpose of Kokkos is to develop performance-portable code that can be executed on a variety of architectures, especially on GPUs. Kokkos enables the creation of functions that work similarly to the kernel functions of CUDA [45]. These functions are then called in parallel by a user-specified amount of threads. Depending on the number of threads that are executed in parallel, the Kokkos framework can run the code on different parts of the underlying architecture (CPU or GPU). For instance, it is advantageous to run a large amount of threads on a GPU, while it is beneficial to run the code on the CPU cores if only a few threads are requested.

Furthermore, Kokkos allows one to execute the exact same code specifically on GPUs or CPUs in parallel as well as on a single CPU core in serial. This was helpful for conducting the experiments because it provided an excellent comparison of the proposed algorithms executed on different architectures. Therefore, the impact of the high degree of parallelism of GPUs could be analyzed more thoroughly. Further details on the development with Kokkos can be found in Section A.1.



i	0	1	2	3	4	5	6	7	8	9
$xadj[i]$	0	1	4	6	9	10				
$adjncy[i]$	1	0	2	3	1	3	1	2	4	3

Figure 4.1: An example graph and the two arrays to store its data.

4.2 Data structure

This section focuses on the data structure utilized by the algorithms. An adjacency matrix is a common data structure for storing graph data. However, this structure requires a large amount of memory. In addition, GPUs have limited memory available. Therefore, the Compressed Sparse Row (CSR) format is used for the algorithms, which requires less memory. This format stores the entire graph data in two arrays, $xadj$ and $adjncy$. The array $adjncy$ stores the destination of each edge, starting from the edges of the vertex v_0. For a vertex $v_i \in V$, $xadj[i]$ points to the first entry of $adjncy$ that belongs to v_i. Thus, all edges of v_i are listed from $adjncy[xadj[i]]$ to $adjncy[xadj[i+1]-1]$. The size of $adjncy$ is $2 * m$ and the size of $xadj$ is $n + 1$, because the last entry of $xadj$ points to the end of the $adjncy$ array. Furthermore, the degree of each vertex v can be obtained by $xadj$: $deg(v_i) = xadj[i+1] - xadj[i]$. An example graph and its corresponding arrays $xadj$ and $adjncy$ are shown in Figure 4.1.

An additional array $state$ is used to determine whether a specific vertex is inside of the calculated independent set. If a vertex v_i is part of the independent set, $state[i]$ equals 1 and 0 otherwise. Initially, the entry of each vertex in the array $state$ is set to -1, representing that a vertex is still in the residual graph, and a decision on whether it is in the independent set is yet to be made.

The algorithms DUMIS and IDUMIS require information on the degree of the vertices in the residual graph. Therefore, an array $degree$ is used to store and maintain this information. Lastly, another array $priority$ saves a randomly generated number between 0 and 1 for each vertex in the graph. This array is required for the proposed algorithms and its purpose is explained in further detail in Section 4.3 and Section 4.4.

4.3 LUMIS

This first algorithm that was implemented is based on Luby's algorithm. The fundamental idea of LUMIS is to assign a random priority to each vertex. In each iteration of the algorithm, a vertex is added to the solution if it has the highest priority in its neighborhood. If a vertex is added to the graph, its neighborhood is removed from the graph. This is

Algorithm 1 LUMIS

1: **Input:** Graph $G = (V, E)$
2: **Output:** Maximal independent set $I \subseteq V$
3: Initialize $I \leftarrow \emptyset$
4: Initialize array $state$
5: **for all** $v \in V$ **do**
6: $\quad state[v] \leftarrow -1$
7: **end for**
8: Initialize array $priority \leftarrow InitializePriorities()$ ▷ Assigns a random priority to each vertex
9: **while** $\exists v \in V : state[v] = -1$ **do**
10: $\quad$ **for all** $v \in V$ with $state[v] = -1$ **do**
11: $\quad\quad$ **if** $state[u] \neq 1 \; \forall u \in N(v)$ **then**
12: $\quad\quad\quad$ **if** $priority[v] > priority[u] \; \forall u \in N(v)$ with $state[u] = -1$ **then**
13: $\quad\quad\quad\quad state[v] \leftarrow 1$
14: $\quad\quad\quad\quad state[u] \leftarrow 0 \; \forall u \in N(v)$ ▷ Remove neighborhood of vertex in solution
15: $\quad\quad\quad$ **end if**
16: $\quad\quad$ **end if**
17: $\quad$ **end for**
18: **end while**
19: $I \leftarrow \{v \in V \; : state[v] = 1\}$
20: **return** I

repeated until the residual graph is empty. This procedure ensures both independence and maximality of the solution by design. Algorithm 1 outlines the functionality of LUMIS.

The actual implementation for GPUs is more complex and structured as follows: Initially, each vertex v_i is assigned a priority $priority[v_i] \in [0; 1]$ at random. To establish that, a kernel function was implemented. This kernel function is called for each vertex in the graph and utilizes the Kokkos-Random-Library. This is crucial because common libraries and functions for random number generation, like the *random* library of C++, cannot be executed on GPUs. This form of parallelism that executes a specific kernel function for each vertex in the graph is used throughout the project.

Thereafter, another kernel function is called that checks for each vertex if it is the vertex with the highest priority in its neighborhood. This corresponds to Step 12 of Algorithm 1. It is essential to note that vertices that have been removed from the graph in previous iterations are not considered. If a vertex has the highest priority in its neighborhood, its entry in the $state$ array is set to 1. Afterwards, the neighborhood of that vertex gets removed from the graph by setting the $state$ array entry of each vertex in the neighborhood to 0. Figure 4.2 illustrates the functionality of LUMIS on an example graph.

Additionally, a counter is incremented that represents the number of vertices that were added to the solution during the current iteration. This counter is shared among all threads

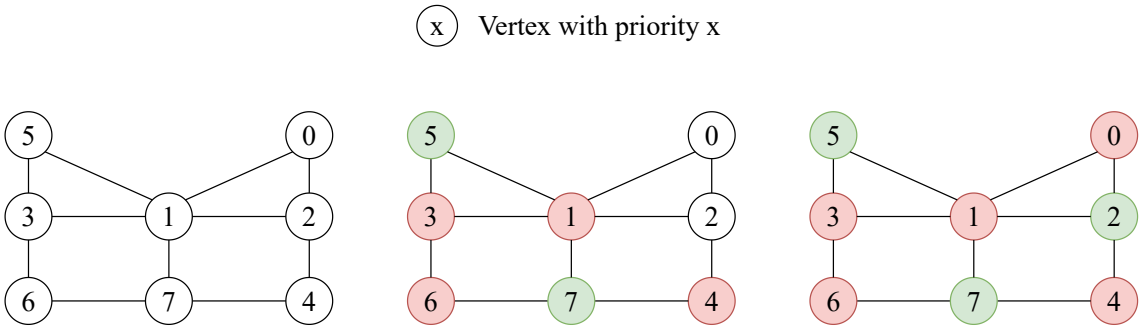


Figure 4.2: Illustration of the execution of the LUMIS algorithm.

and is accessed with atomic operations to prevent race conditions. If the counter is zero at the end of an iteration, the algorithm concludes and the calculated independent set is returned. Thus, in contrast to Step 9 of Algorithm 1, the algorithm does not check if there are any vertices left in the residual graph. Using the counter instead avoids the need for an additional kernel function. Consequently, the algorithm runs for an additional iteration in which no changes are made. However, we found that employing the counter requires less computation time.

Furthermore, the algorithm only includes a vertex in the solution if it has the highest priority in its neighborhood. This design of Luby's algorithm ensures the independence of the calculated solution. In addition, this prevents race conditions, because no adjacent vertices can be added to the independent set at the same time due to their different priorities. Thus, the main steps of the algorithm can be executed in parallel using a single kernel function.

4.4 DUMIS

This algorithm is designed similarly to the LUMIS algorithm, which is described in detail in Section 4.3. The fundamental idea of DUMIS is to add the vertex with the lowest degree in its neighborhood to the independent set. The enforcement of this idea likely increases the quality of the solution [30], because adding vertices with a low degree to the independent set leads to fewer vertices that are removed from the graph afterwards. In a graph, multiple vertices can have the same degree in a neighborhood. In that case, random priorities are used to handle these tie-breaks. In the scenario of a tie, the vertex with the highest priority is selected over the other vertices of the same degree. Each iteration of this algorithm works similarly to an iteration of LUMIS. The only differences are additional checks that prioritize a low-degree vertices over vertices of higher degree. This fundamental idea is taken from Imanaga et al. [34]. Their proposed algorithm is explained in Chapter 3. The functionality of DUMIS is outlined in Algorithm 2.

Initially, the degree of each vertex is determined by the information stored in the array $xadj$, as explained in Section 4.2. Similarly to the algorithm of Imanaga et al. [34], the

Algorithm 2 DUMIS

Input: Graph $G = (V, E)$
Output: Maximal independent set $I \subseteq V$
Initialize $I \leftarrow \emptyset$
Initialize array $state$
for all $v \in V$ **do**
    $state[v] \leftarrow -1$
end for
Initialize array $priority \leftarrow InitializePriorities()$ ▷ Assigns a random priority to each vertex
while $\exists v \in V : state[v] = -1$ **do**
    for all $v \in V$ with $state[v] = -1$ **do**
        if $state[u] \neq 1 \; \forall u \in N(v)$ **then**
            if $IsPrioritizedOver(v, u) \; \forall u \in N(v)$ with $state[u] = -1$ **then**
                $state[v] \leftarrow 1$
                $state[u] \leftarrow 0 \; \forall u \in N(v)$ ▷ Remove neighborhood of vertex in solution
            end if
        end if
    end for
end while
$I \leftarrow \{v \in V \; : state[v] = 1\}$
return I

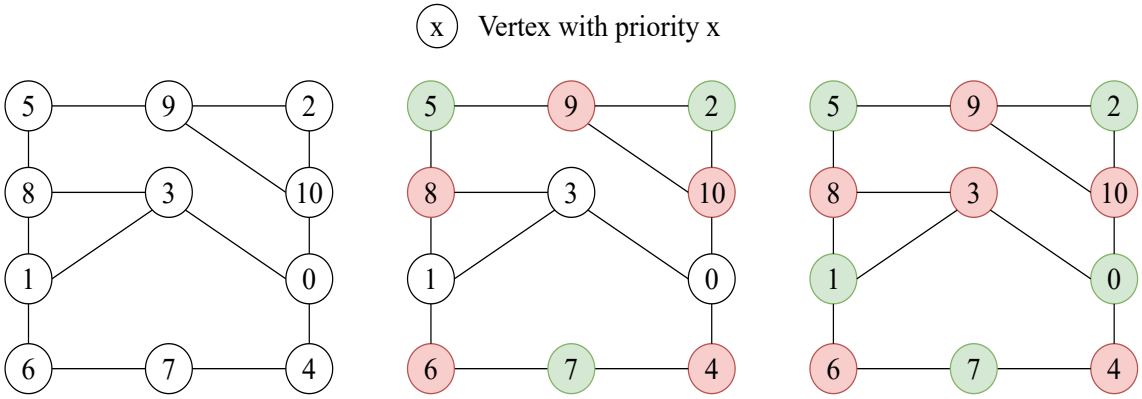
function ISPRIORITIZEDOVER(Vertex v, Vertex u)
    if $degree[v] < degree[u]$ **then**
        return $True$
    else if $degree[v] = degree[u] \wedge priority[v] > priority[u]$ **then**
        return $True$
    else
        return $False$
    end if
end function

degree is not updated between iterations of the algorithm by default. In later stages of development, a parameter was introduced to DUMIS to modify the degree updates. The *update frequency* $f \in \mathbb{N}$ determines how often the degree of each vertex in the residual graph is updated. An update frequency of f implies that degrees are updated every f-th iteration. We denote a *small* number for f as a *high* frequency and vice versa. A higher update frequency can significantly increase the quality of the calculated solution. However, more updates come with a crucial computational cost and thus increase the runtime of the algorithm. An update frequency of 2 was found to provide a high-quality solution without

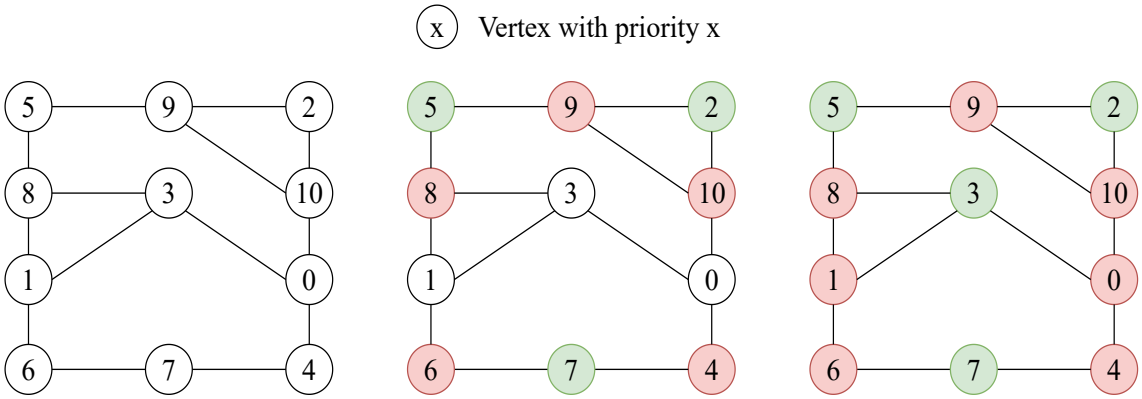
requiring a large amount of additional execution time. This is discussed in more detail in Section 6.3.

The update of the degrees works as follows: for each vertex v that is inside the residual graph, meaning that $state[v] = -1$, a thread is launched that counts the adjacent vertices of v that are in the remaining graph. This information is then stored in the array $degree$. This array is required because the $xadj$ array is not modified based on the residual graph. Therefore, $xadj$ is only capable of displaying the information of the degree of each vertex at the beginning of the algorithm.

Figure 4.3 demonstrates how the DUMIS algorithm computes a maximal independent set while selecting vertices with the lowest degree of a neighborhood. In addition, it illustrates the impact of the update frequency on the quality of the solution. Due to the higher update frequency, DUMIS computes a larger independent set, shown in Figure 4.3a, than without intermediate degree updates, displayed in Figure 4.3b. The first iteration is identical for both instances. The difference lies in the second iteration. With an update frequency of one, the degrees of the vertices with priorities 0 and 1 are updated to one, while the degree of the



(a) Illustration of the execution of the DUMIS algorithm using an update frequency of one.



(b) Illustration of the execution of the DUMIS algorithm using no intermediate degree updates.

Figure 4.3: Comparison of the DUMIS algorithm with different update frequencies

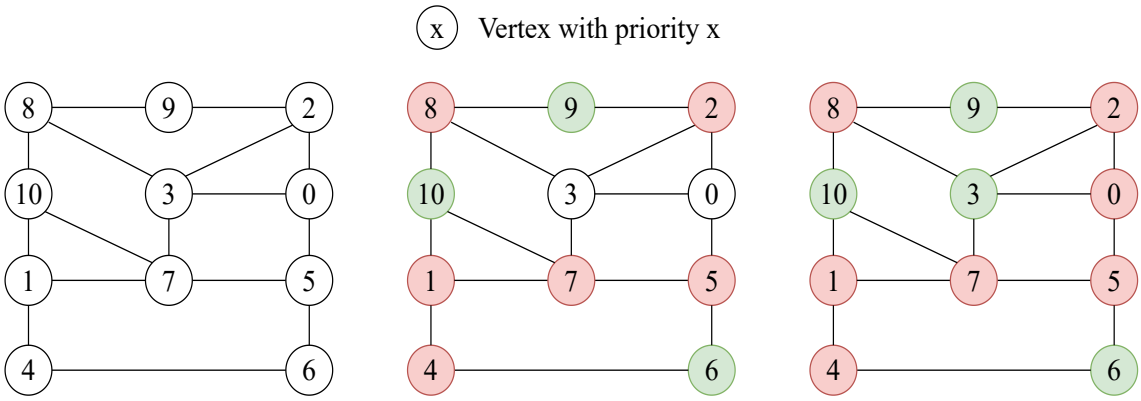
vertex with priority 3 is adjusted to two, respectively. Therefore, the vertices with degree one are selected over the other vertex. Without updating the degrees, all three vertices in the residual graph have degree three. Thus, the priorities induce a tie-break in which only the vertex with the highest priority is added to the set. This observation demonstrates that a higher update frequency can lead to the computation of larger independent sets.

4.5 IDUMIS

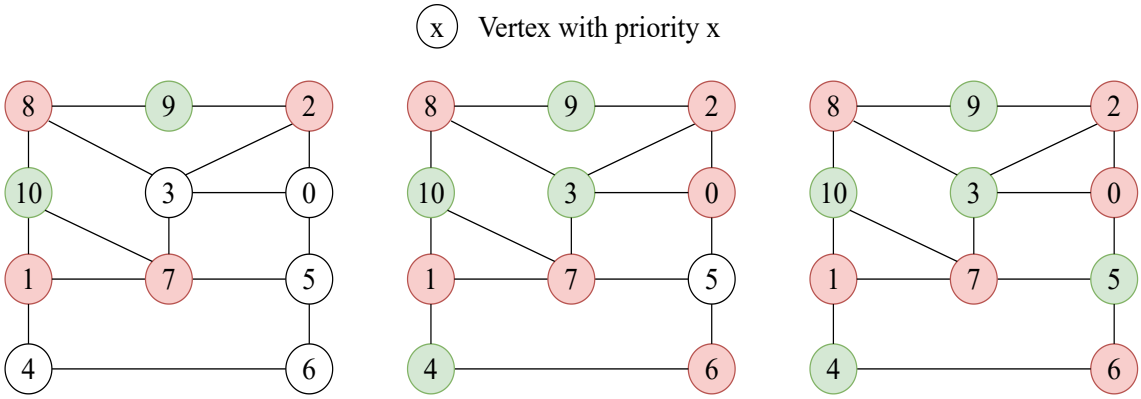
Imanaga et al. [34] present another crucial idea that is part of the approach used in this thesis. LUMIS and DUMIS only calculate a solution once and then conclude. In contrast, many MIS algorithms calculate an initial solution that is then improved over time until a local maximum is found or a time limit is reached (for instance, see [20, 28, 41, 43]). Therefore, we designed the algorithm IDUMIS that runs multiple rounds of DUMIS improving the initial solution over time. However, IDUMIS does not strategically improve the initial maximal independent set I provided by the DUMIS algorithm. Instead, each vertex $v \in I$ is removed from the solution by a chance of 50%. Subsequently, the removed vertices and their respective neighborhoods are added back to the residual graph. Thereafter, DUMIS is executed again on the resulting graph. This process likely computes a different solution as in the previous round. This procedure is repeated until a user-specified time limit is reached. During this process, the best solution found so far is stored. Technically, this process can be performed with the LUMIS and the DUMIS algorithms. However, we show that DUMIS produces significantly larger solutions than LUMIS. Therefore, DUMIS is used with an update frequency of 1, since the objective of this approach is to achieve a high-quality solution within a given time limit. The fundamental concepts of IDUMIS are examined in Algorithm 3.

In detail, if a vertex v of the current solution is removed, $state[v]$ and $state[u]$ are set to -1 $\forall u \in N(v)$. This process adds the closed neighborhood of v back to the residual graph. Thus, they are involved in the next execution of the DUMIS algorithm. However, there can be conflicts by adding back the whole closed neighborhood of the vertices that have been removed from the solution. It is possible that these vertices remain adjacent to a different vertex that was not removed from the solution during the preceding step. Therefore, a kernel function is called for each vertex v that remains in the solution. The entry in the array $state$ of each vertex adjacent to v is then set to zero. This ensures the independence of the consecutive solution and resolves the conflict.

The concept of IDUMIS can produce larger maximal independent sets. This is illustrated in Figure 4.4. The initial round of IDUMIS produces an independent set of size four, which can be observed in Figure 4.4a. Subsequently, the vertices with priorities 3 and 6 are removed from the independent set. Consequently, both the vertices and part of their neighborhoods are added to the residual graph. Not all vertices of their neighborhoods can be added back to the residual graph because a portion of them is adjacent to a vertex that remains in the solution. For example, the vertex with priority 2 is in the neighborhood of



(a) Round one of the IDUMIS algorithm.



(b) Round two of the IDUMIS algorithm.

Figure 4.4: Illustration of the execution of the IDUMIS algorithm.

the vertex with priority 3. However, it cannot be added to the residual graph because it is adjacent to the vertex with priority 9, which is part of the independent set. Figure 4.4b shows the second round of IDUMIS that is performed on the residual graph. Since IDUMIS uses DUMIS with an update frequency of 1, the degree of each vertex in the residual graph is updated. Therefore, the resulting independent sets after the first and second round differ. Since the solution after the second round is larger, IDUMIS stores that solution. This overwrites the initial independent set improving the quality.

Algorithm 3 IDUMIS

Input: Graph $G = (V, E)$
Output: Maximal independent set $I \subseteq V$
Initialize $I_{max} \leftarrow \emptyset$
Initialize $I \leftarrow \emptyset$
Initialize $J \leftarrow \emptyset$
while time limit is not reached **do**
    $J \leftarrow DUMIS(G \setminus I)$
    $I \leftarrow I \cup J$
    if $\mid I \mid > \mid I_{max} \mid$ **then**
        $I_{max} \leftarrow I$
    end if
    $I \leftarrow RemoveVertices(I)$
end while
return I

function REMOVEVERTICES(Set of Vertices I)
    for all $v \in I$ **do**
        Initialize random number $r \in [0; 1]$
        if $r < 0.5$ **then**
            $I \leftarrow I \setminus \{v\}$
        end if
    end for
    return I
end function

CHAPTER 5

Data Reductions

There are multiple known data reductions of the MIS problem. Using data reductions can significantly improve the performance and quality of MIS algorithms [29, 35]. Hence, three reductions were implemented. In this chapter, the reductions are outlined and their impact is analyzed in the experiments in Chapter 6.

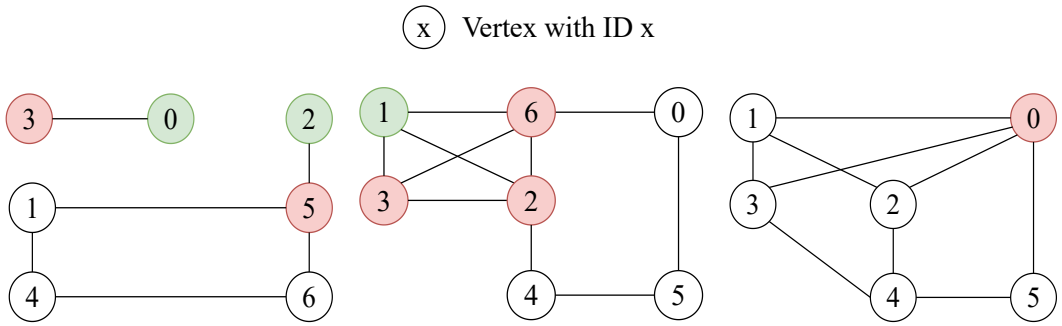
5.1 Description of the Data Reductions

Theorem 1 (Degree-one reduction [35])
Vertices of degree one are always part of at least one maximum independent set. Therefore, $|MIS(G)| = |MIS(G \setminus N[v]) \cup \{v\}|$ *for* $v \in V$ *with* $deg(v) = 1$*. If two vertices of degree one are adjacent to each other, only one of those vertices can be inside of the MIS.*

The first and most simple reduction is called *degree-one reduction* (see Theorem 1). This reduction rule is illustrated in Figure 5.1a. The vertices 0, 2 and 3 have degree one in this example, while vertices 0 and 3 are adjacent to each other. To resolve the tie-break of vertices 0 and 3, only one of them is added to the independent set (selecting the vertex with lower vertex ID). Subsequently, the neighbors of the added vertices are removed from the graph.

Definition 1 (Isolated vertex)
An isolated vertex v *is a vertex that is part of a clique* C *and has no adjacent vertices outside of* C*:* $N[v] = C$*.*

Theorem 2 (Isolated vertex reduction [29])
Consider the isolated vertex $v \in V$ *of the clique* $C \subseteq V$*. By definition,* v *is only adjacent to other vertices of* C*. Vertex* v *must be in at least one maximum independent set. Thus,* $|MIS(G)| = |MIS(G \setminus C) \cup \{v\}|$*. If multiple vertices of the same clique are isolated, either one of them can be included in the MIS.*



(a) Degree-one reduction. **(b)** Isolated vertex reduction. **(c)** Domination reduction.

Figure 5.1: Illustration of the reduction rules.

The next reduction, outlined in Theorem 2, is connected to cliques, which are a structure that can be found within graphs. Furthermore, a clique is a crucial structure for the MIS problem, because one vertex of a clique can be part of a MIS at most. Figure 5.1b shows a graph that contains a clique, made up of vertices 0, 1, 2 and 3. By definition, vertices 0 and 1 are isolated. Therefore, either one of the two vertices can be added to the independent set, while the remainder of the clique can be removed.

Definition 2 (Domination)
Consider two vertices $u, v \in V$. The vertex u dominates v if $N[u] \subseteq N[v]$.

Theorem 3 (Domination reduction [35])
Consider two vertices $u, v \in V$ such that u dominates v. Then v can be removed from the graph: $|MIS(G)| = |MIS(G \setminus \{v\})|$. In the special case that two vertices share the same closed neighborhood ($N[u] = N[v]$), either of the two vertices can be removed from the graph.

The *domination reduction* (see Theorem 3) is illustrated in Figure 5.1c. In this example, vertex 1 has a closed neighborhood consisting of the vertices 0, 1, 2, and 3. The closed neighborhood of vertex 0 contains the same vertices, but in addition, vertex 5 is also a part of it. Therefore, vertex 1 dominates vertex 0 and vertex 0 is removed from the graph.

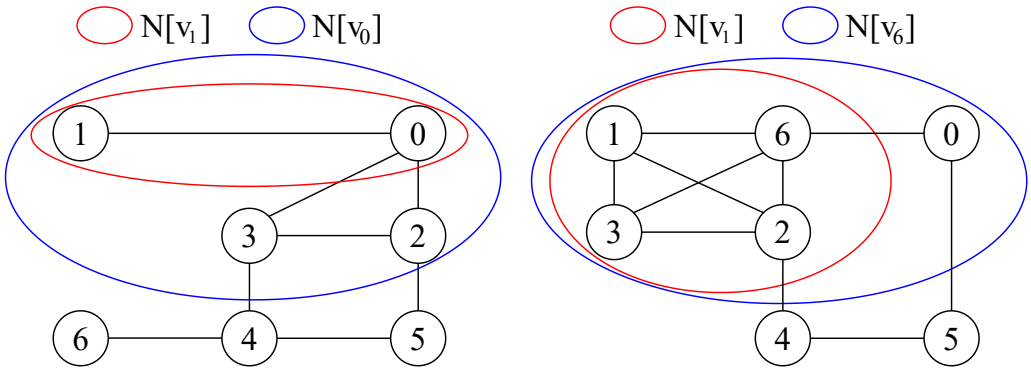
5.2 Implementation of the Data Reductions

The objective was to implement the reductions for parallel execution on GPUs. The reason for that was to potentially reduce the time required to apply the reductions and hence improve the performance of the algorithm. It should be noted that the degree one and the isolated vertex reduction do not improve the quality of the calculated solution for the DU-MIS algorithm. This is the case because those reductions strongly depend on the degrees

of the vertices. As described in detail in Section 4.4, DUMIS includes vertices with the lowest degree in its neighborhood to the solution. Therefore, DUMIS conducts the same decisions that are induced by these reductions. The only difference is in the handling of tie-breaks. While DUMIS relies on random numbers for tie-breaks, the reductions prefer the vertex with the lower vertex ID. Therefore, only the domination reduction has a positive effect on the independent set size produced by DUMIS. Thus, we conducted experiments with the reductions using the LUMIS algorithm in Section 6.4.

When investigating the reductions thoroughly, one can notice that implementing the domination reduction already includes the isolated vertex reduction as well as the degree-one vertex reduction. In detail, a vertex $v \in V$ with degree one always dominates its only adjacent vertex $u \in V$. The adjacent vertex u must have at least degree one because it is adjacent to v. However, it may have additional neighbors. Therefore, $N[v] \subseteq N[u]$, which implies that v dominates u. Thus, according to the domination reduction, u can be removed from the graph. Figure 5.2a demonstrates this principle.

Furthermore, an isolated vertex $v \in V$ is part of a clique C. By definition, all vertices of C are adjacent to each other, while v is only adjacent to the members of the clique. Therefore, the closed neighborhood of v and the clique C represent the same set of vertices: $N[v] = C$. Moreover, this implies $N[v] \subseteq N[u]\ \forall u \in C$. Thus, v dominates each vertex of the clique. According to the domination reduction, this means that all vertices of C, except for v, can be removed from the graph. Figure 5.2b illustrates this principle in detail. The only difference is that the domination reduction only removes vertices. In contrast, the other reductions add vertices to the independent set and remove their neighborhood. However, the domination reduction removes the entire neighborhood of degree-one vertices and isolated vertices. Therefore, it is ensured that the subsequently executed algorithm includes such vertices in the independent set.



(a) Degree-one reduction based on the domination reduction.

(b) Isolated vertex reduction based on the domination reduction.

Figure 5.2: Illustration of the relationship between the reductions.

As shown, properly implementing the domination reduction covers the degree-one reduction and the isolated vertex reduction. The domination reduction was added to the implementation by defining a kernel function that is executed for each vertex in the graph. An outline of the implementation is shown in Algorithm 4. In detail, the algorithm determines for each vertex if it is dominated by another vertex. We found that sorting the edges of the input graph allows us to implement the reductions more effectively, so that they require significantly less execution time. Therefore, using reductions in our approach requires sorted input graphs. To check if a vertex $v \in V$ is dominated by another vertex, we check each adjacent vertex $u \in N(v)$ at a time. It is checked if each neighbor of u is also a neighbor of v. As we presuppose sorted input graphs for this process, we only need to go through the neighborhoods of v and u once. If u dominates v, v is removed from the graph. Otherwise, this process is repeated for the next vertex in $N(v)$. If v is not dominated by another vertex, it remains in the graph. This process reduces the size of the graph. Consequently, the reduction can be repeated if vertices have been removed during the preceding execution.

Algorithm 4 Domination Reduction

```
1:  Input: Graph G = (V, E), Array state
2:  Output: Array state
3:  for all v ∈ V do
4:      for all u ∈ N(v) do
5:          if N[u] ⊆ N[v] then
6:              if N[u] != N[v] ∨ v_ID > u_ID then        ▷ Tie-breaks via vertex ID
7:                  state[v] ← 0
8:              end if
9:          end if
10:     end for
11: end for
12: return state
```

CHAPTER 6

Experimental Evaluation

In this chapter, the experiments conducted over the course of this work are presented. First, the experimental environment is described in detail in Section 6.1. Thereafter, information about the datasets used for the experiments is provided in Section 6.2. Subsequently, the performances of the proposed algorithms are compared for different parameters in Section 6.3. Section 6.4 focuses on the runtime and the impact on the quality of the solution introduced by the reductions. Finally, the proposed algorithms are compared with recently published algorithms in the field in Section 6.5.

6.1 Methodology

The proposed algorithms are executed on a CPU and on a GPU. This allows an analysis of the impact of the high degree of parallelism of GPUs on a MIS algorithm. Furthermore, the MMWIS algorithm [43] was used in the experiment, which is a CPU algorithm. MMWIS was executed with the *mmwiss* configuration. In addition, ECL-MIS [32] was used in the experiment. There exist two versions of ECL-MIS: one using OpenMP for the CPU, the other one utilizing CUDA for the GPU. For the following experiments, only the GPU-based version of ECL was used. Therefore, it provides a comparison of the proposed algorithms with another GPU implementation. ECL utilizes deterministic random numbers. Additionally, ECL does not accept arguments like seeds to adjust the generation of random numbers. Therefore, each execution of the algorithm produces the same maximal independent set.

All experiments were carried out on the BwUniCluster2.0 [1]. The algorithms running on CPUs were executed on the *single* partition of the cluster with an Intel Xeon Gold 6230 (20 cores) running at 2.1 GHz. The GPU-based algorithms were executed on the *gpu_4* partition using an Intel Xeon Gold 6230 (20 cores) running at 2.1 GHz and a NVIDIA Tesla V100 with 32 GB VRAM. However, MMWIS is a serial algorithm. Therefore, MMWIS could only utilize one of the 20 CPU cores provided. To compare the proposed algorithms

more accurately to MMWIS, the proposed algorithms were also executed in serial mode. Moreover, all experiments were executed with 150GB of RAM available.

The proposed algorithms are implemented in C++ and compiled with gcc 13.3.0 using the -O3 flag for full optimization. For executions on the GPU, the code was compiled with nvcc 12.2. To run the proposed algorithms on the different architectures (GPU, CPU, and in serial), the project was compiled using three different Kokkos installations: one that uses OpenMP, another that uses CUDA, and finally one that executes the code in serial on a single CPU core. To distinguish the different architectures used for the proposed algorithms, the results are denoted as follows: Algorithm$_{Architecture}$. For example, DUMIS$_{GPU}$ denotes the results of the DUMIS algorithm executed on the GPU. Moreover, for some of the results, LUMIS and DUMIS are abbreviated as L and D, respectively.

If not stated otherwise, each algorithm was executed five times for each graph instance using seeds 1000, 2000, 3000, 4000, and 5000 for random number generation. Afterwards, the average size of the maximal independent sets as well as the average execution time were calculated to produce the displayed results. As previously mentioned, ECL is completely deterministic. Therefore, each execution produced the same solution size. Despite that, the average of the five runs of ECL was calculated for its execution times for each instance. For progress charts showing the size of the best solution found over time (see Figure 6.5), the algorithms were executed once for each graph instance with a time limit of one hour using seed 1000 for random number generation.

Furthermore, we use *performance profiles* [10] to compare the execution times and solution sizes of LUMIS, DUMIS, and ECL. Let P be the set of graph instances on which the experiment was conducted. In addition, t^p_{Alg} denotes the runtime and s^p_{Alg} the solution size the Algorithm Alg delivered for the graph instance $p \in P$. Moreover, t^p_{best} and s^p_{best} describe the best time and size achieved for graph p among all algorithms, respectively. For execution times, a performance profile shows the fraction of all graphs an algorithm Alg solved within a factor of τ of the best execution time: $\frac{|\{p \in P : t^p_{Alg} \leq \tau \cdot t^p_{Best}\}|}{|P|}$. If execution times are displayed, $\tau \geq 1$. For solution quality, performance profiles display the fraction of all graphs algorithm Alg solved with a solution size greater than τ times the largest solution: $\frac{|\{p \in P : s^p_{Alg} \geq \tau \cdot s^p_{Best}\}|}{|P|}$. In this case, $0 < \tau \leq 1$. In general, if an algorithm shows that it solved instances with $\tau = 1$, it performed better than the competing algorithms for such instances. Furthermore, the higher the fraction of instances that were solved with a value for τ close to 1, the better the algorithm performed.

6.2 Datasets

In total, 32 different graph instances were used in the experiment. Their proportions and the source they originate from are displayed in Table 6.1. The graphs were converted from the SNAP to the METIS format, since all of our proposed algorithms and MMWIS use this graph format. Thereafter, self-loops were removed from each instance, and the

Graph	Vertices	Edges	Origin
webbase-2001	118 142 155	868 338 910	[12, 19]
friendster	65 608 366	1 806 067 135	[24, 26]
twitter-2010	41 652 230	1 202 513 046	[12, 19, 18]
it-2004	41 291 594	1 034 978 210	[12, 19]
uk-2005	39 454 746	1 566 054 250	[12, 19, 11]
gsh-2015-tpd	30 809 122	489 675 683	[12, 19, 22]
arabic-2005	22 744 080	558 325 967	[12, 19, 11]
uk-2002	18 520 486	264 722 307	[12, 19, 11]
eu-2015-tpd	6 650 532	112 106 091	[12, 19, 22]
enwiki-2013	4 206 785	91 961 847	[12, 19]
livejournal	3 997 962	34 681 189	[24, 26]
orkut	3 072 441	117 185 083	[24, 26]
hollywood-2011	2 180 759	114 492 816	[12, 19]
roadNet-CA	1 965 206	2 766 607	[24, 16]
as-skitter	1 696 415	11 095 298	[24, 13]
dewiki-2013	1 532 354	33 095 282	[12, 19]
in-2004	1 382 870	13 591 473	[12, 19]
roadNet-TX	1 379 917	1 921 660	[24, 16]
youtube	1 134 890	2 987 624	[24, 26]
roadNet-PA	1 088 092	1 541 898	[24, 16]
web-Google	875 713	4 322 051	[24, 16]
eu-2005	862 664	16 138 468	[12, 19]
amazon-2008	735 323	3 523 472	[12, 19]
web-BerkStan	685 230	6 649 470	[24, 16]
web-NotreDam	325 729	1 103 835	[24, 16, 9]
cnr-2000	325 557	2 738 969	[12, 19]
web-Stanford	281 903	1 992 636	[24, 16]
ca-CondMat	23 133	93 468	[24, 14]
ca-AstroPh	18 772	198 080	[24, 14]
ca-HepPh	12 008	118 505	[24, 14]
ca-HepTh	9 877	25 985	[24, 14]
ca-GrQc	5 242	14 490	[24, 14]

Table 6.1: Origin and number of vertices and edges of the graphs used for the experiment.

graphs were converted to undirected graphs. This conversion process was performed using the networkit framework [39]. MMWIS as well as the proposed reductions in Chapter 5 require sorted METIS graph files. Therefore, the edges were sorted using an algorithm provided by KaMIS[2], the framework from which MMWIS originates. ECL requires graphs in a different graph format. However, the developers of ECL provided a conversion

algorithm in their framework, which allowed us to convert the sorted METIS graph files to the required format. Furthermore, the graphs were obtained from the Laboratory for Web Algorithmics[4] as well as the Stanford Large Network Dataset Collection [3].

6.3 Proposed Algorithms

The results of the experiments presented in this section demonstrate the impact of certain design decisions made during the development of the proposed algorithms.

The most significant design decision was to introduce the fundamental concept of preferring low-degree vertices over high-degree vertices. The impact of this concept can be demonstrated by comparing the quality and execution time of LUMIS and DUMIS. Table 6.2 shows the execution time of both algorithms on the different architectures for a portion of the graph instances. Furthermore, it is notable that the algorithms produce different maximal independent sets on the different architectures. This is the case because the algorithms involve random number generation, which generates different outputs for the same seed when executed on different architectures. However, the proposed algorithms produce similar independent set sizes for all of the architectures. Therefore, only the sizes produced on the GPU are displayed in Figure 6.1. Furthermore, we executed MMWIS for this chart once for each graph instance using seed 1000 and a time limit of 10 hours. In that time, MMWIS is able to produce exact or high-quality solutions for most graph instances. This provides a comparative value for the proposed algorithms and allows us to evaluate their produced sets more accurately. Moreover, the figure displays the sizes produced by IDUMIS with a time limit of one hour.

Figure 6.1 shows that the selection of low-degree vertices over high-degree vertices has a crucial impact on the size of the solution. For each graph, DUMIS computes a larger maximal independent set than LUMIS. In addition, the figure shows that the concept of IDUMIS works, since it produces even larger independent sets than DUMIS for all in-

Graphs	$\mathbf{L}_{GPU}$	$\mathbf{L}_{CPU}$	$\mathbf{L}_{SERIAL}$	$\mathbf{D}_{GPU}$	$\mathbf{D}_{CPU}$	$\mathbf{D}_{SERIAL}$
webbase-2001	**77.48**	711.53	5 516.43	79.78	702.34	5 273.47
it-2004	131.02	402.26	2 582.63	**126.44**	443.40	3 023.72
arabic-2005	72.30	216.10	1 463.24	**62.83**	223.02	1 535.69
uk-2002	25.83	186.40	970.37	**22.58**	186.34	926.96
as-skitter	3.97	16.31	86.93	**3.08**	14.72	80.34
dewiki-2013	6.11	32.81	151.58	**5.07**	26.24	132.64
web-Google	1.51	7.04	48.41	**1.27**	5.87	38.21
ca-HepTh	0.19	0.15	0.33	0.12	**0.10**	0.30

Table 6.2: Time required to compute MIS for the different approaches and architectures in milliseconds.

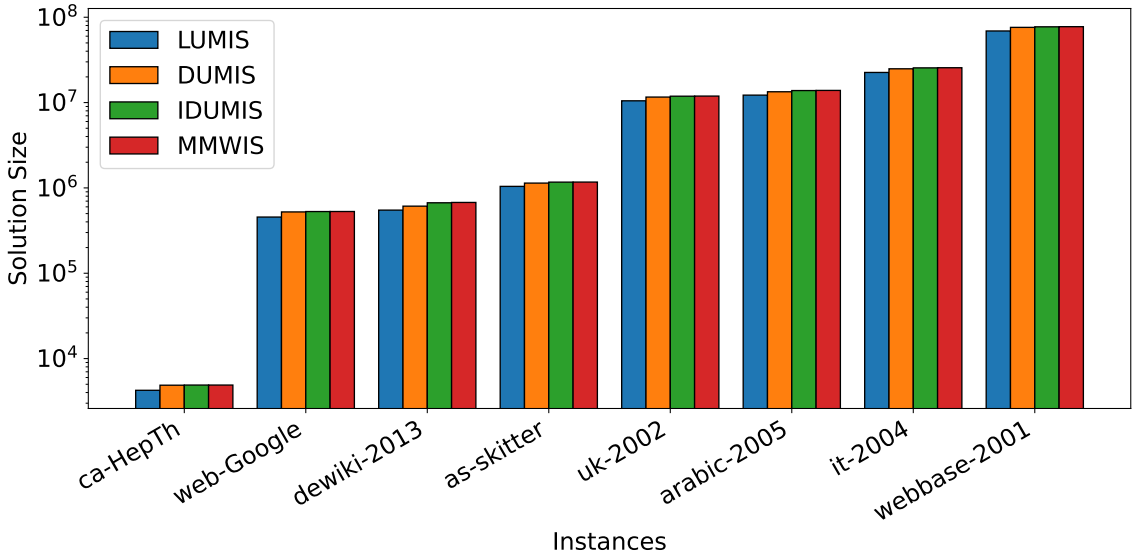


Figure 6.1: Average solution sizes of LUMIS, DUMIS, and IDUMIS compared to the solution of MMWIS. IDUMIS was executed with a time limit of one hour and MMWIS with a ten hour time limit.

Graphs	LUMIS	DUMIS
webbase-2001	7.8	11.0
it-2004	7.8	28.2
arabic-2005	6.6	10.0
uk-2002	6.6	7.2
as-skitter	6.4	6.0
dewiki-2013	7.2	6.0
web-Google	6.2	5.4
ca-HepTh	5.6	3.4

Table 6.3: Amount of iterations the different algorithms required when executed on the GPU.

stances. Table A.1 shows further results of this comparison. On average, LUMIS produced 87.3%, DUMIS 97.1%, and IDUMIS 99.6% of the independent set sizes of MMWIS on the graph instances displayed in the table.
Furthermore, the results in Table 6.2 demonstrate that the algorithms are most effective when executed on the GPU. The only exception is the smallest graph, which is displayed at the bottom of the table. In that case, the CPU architecture provides the solution faster. This can be explained by the small size of the graph. The high degree of parallelism offered by GPUs outperforms the CPU execution only for larger graphs. As expected, executing the algorithms in serial results in the slowest execution times.

Moreover, DUMIS outperforms LUMIS in terms of runtime in most cases, which may seem unexpected because LUMIS and DUMIS are structured similarly, but DUMIS requires additional checks and functions. To determine the cause of this observation, the number of iterations required by the algorithms is measured for the GPU. The results of this analysis are shown in Table 6.3. Each iteration required by the algorithms comes with a computational cost. Thus, if LUMIS needs more iterations than DUMIS for the same graph instance, it could explain the observed shorter runtimes of DUMIS. In general, there are different structures that lead to more iterations. For example, the graph shown in Figure 6.2 is solved by DUMIS in a single iteration, while LUMIS requires an additional iteration. Figure 6.3 displays another example graph. In this case, LUMIS solves the problem in a single iteration, while DUMIS needs two iterations. Table 6.3 shows that LUMIS requires more iterations on average than DUMIS for smaller graph instances. However, especially for the largest graph instances, at the top of the table, DUMIS requires significantly more iterations.

Another aspect to consider is the number of vertices that remain in the residual graph per iteration. The runtime of a single iteration is strongly dependent on the amount of vertices that remain to be checked. Figure 6.4 compares LUMIS and DUMIS on that property. All of these figures show a similar pattern: although DUMIS needs more iterations than LUMIS for these graphs, the additional iterations are executed with a low amount of vertices. In most cases with fewer than 10^3 vertices. Therefore, these additional iterations

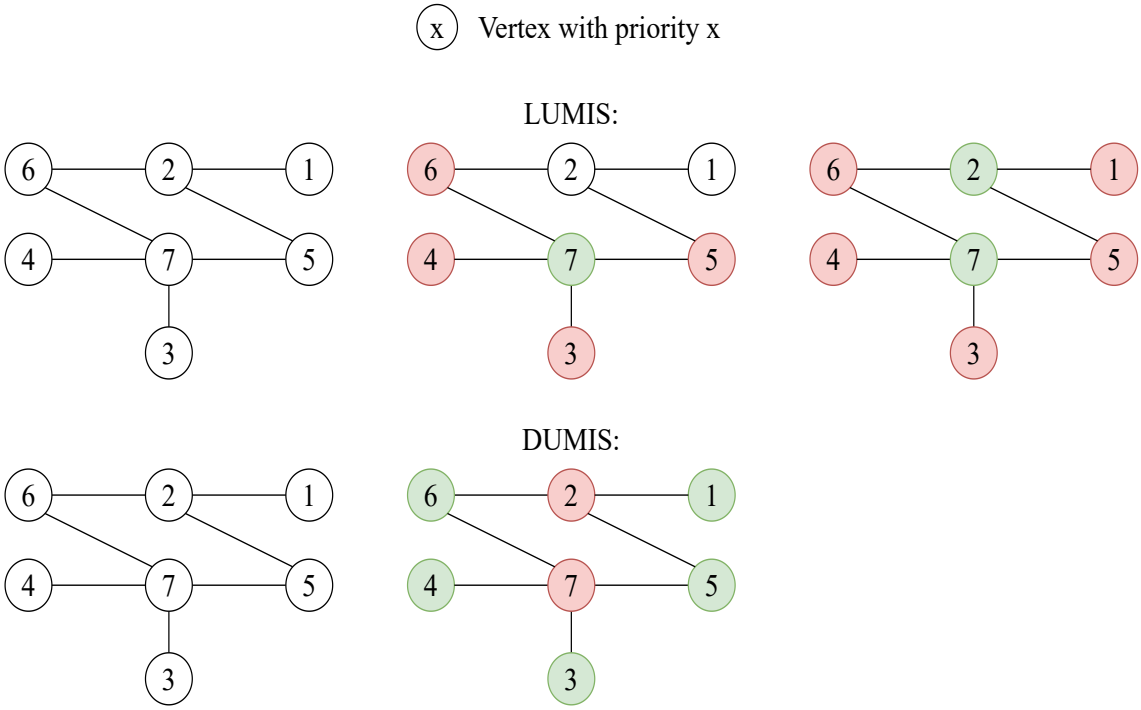


Figure 6.2: Example graph that requires more iterations to solve for LUMIS.

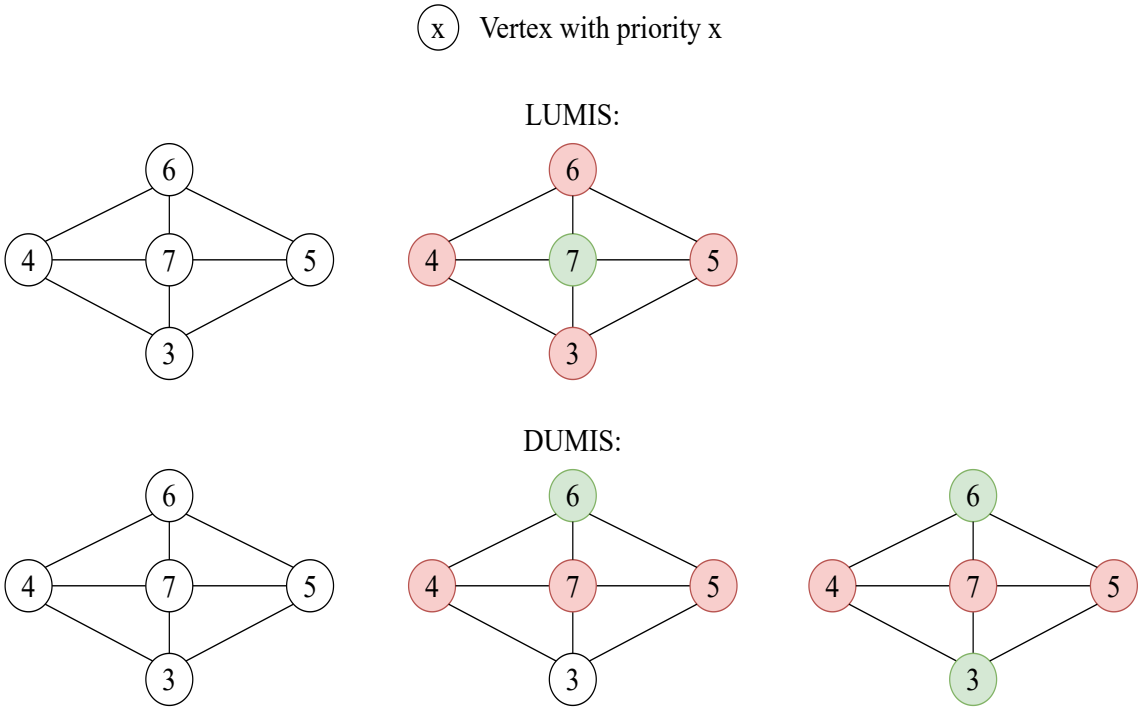


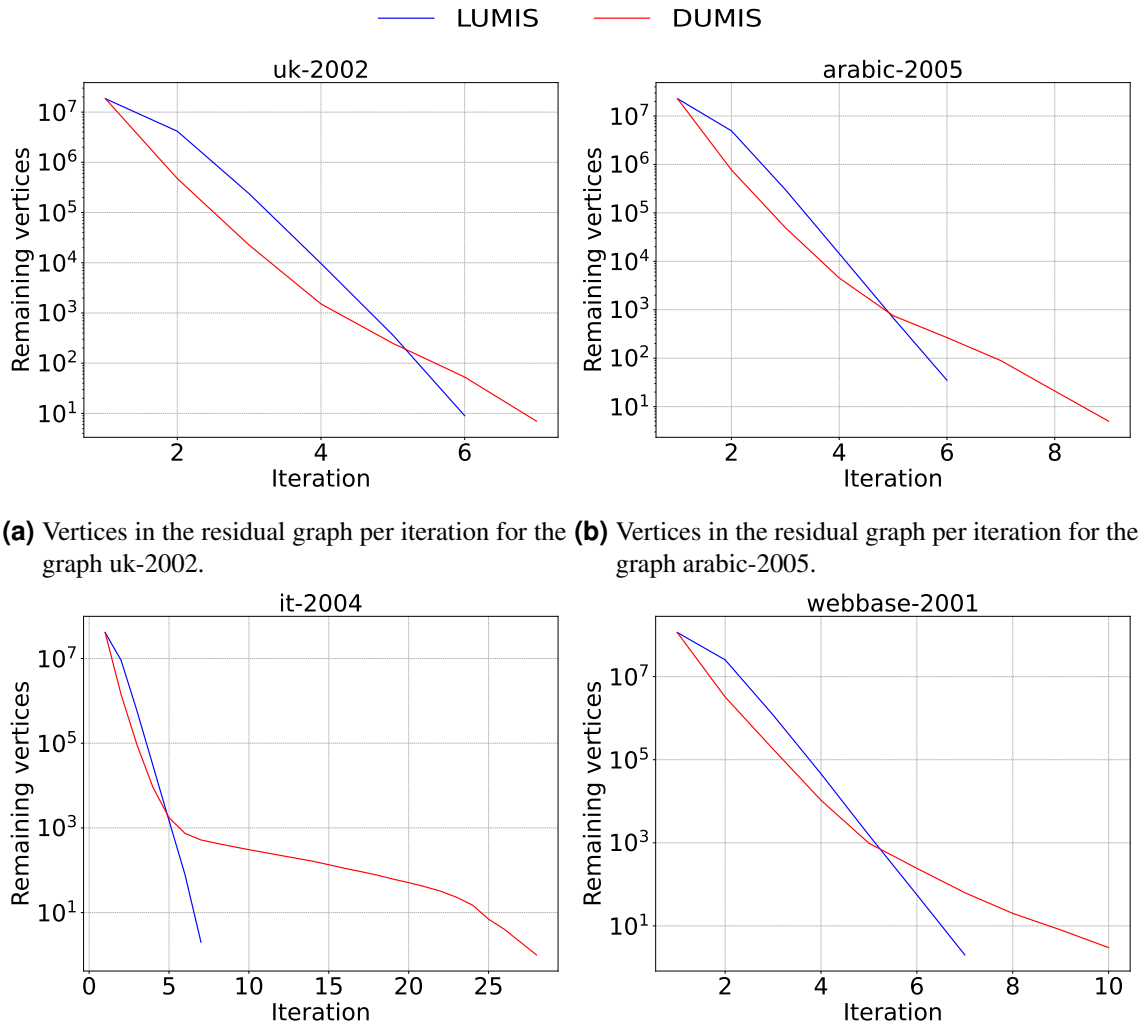
Figure 6.3: Example graph that requires more iterations to solve for DUMIS.

consume a small amount of time compared to the iterations at the beginning with a larger number of vertices. Especially during the first three iterations, DUMIS removed most of the vertices from the graph. At the same time, LUMIS removes notably fewer vertices. Thus, these iterations consume a large amount of time. This observation explains why DUMIS outperforms LUMIS in terms of execution time for most graph instances.

Overall, these experiments show that GPUs can significantly accelerate MIS algorithms. Furthermore, they demonstrate that DUMIS introduces crucial improvements and outperforms LUMIS in quality and runtime. Even for the graph instance LUMIS solves slightly faster, the lack of quality of the solutions is decisive. For that reason, IDUMIS is based on DUMIS. Furthermore, these results show that IDUMIS is capable of significantly improving the initial solution provided.

In Section 4.4, we introduced the update frequency parameter. To analyze its impact on DUMIS, we measured the execution time and independent set size for different update frequency values. The results are listed in Table 6.4. Furthermore, we denote DUMIS^f as DUMIS with an update frequency of f. As expected, a higher update frequency increases the execution time of DUMIS for most instances. Additionally, a higher update frequency results in larger independent sets. However, we want to highlight the difference between the update frequencies 1 and 2. The quality of the solutions produced by DUMIS^1 is only slightly better than the quality of DUMIS^2. However, the execution time of DUMIS^1 is significantly longer. Consequently, we conclude that 2 is the optimal update

frequency for DUMIS, since it provides comparably low execution times while achieving relatively large solutions.



(a) Vertices in the residual graph per iteration for the graph uk-2002.

(b) Vertices in the residual graph per iteration for the graph arabic-2005.

(c) Vertices in the residual graph per iteration for the graph it-2004.

(d) Vertices in the residual graph per iteration for the graph webbase-2001.

Figure 6.4: Diagrams that investigate the amount of vertices that are in the residual graph for LUMIS and DUMIS per iteration.

Frequencies	**1**		**2**		**3**		**None**	
Instance	s	t	s	t	s	t	s	t
friendster	**35 880 760**	1 205.43	35 880 750	264.10	35 856 011	250.24	35 855 467	**249.40**
gsh-2015-tpd	**20 594 726**	1 146.79	20 594 678	187.00	20 578 861	179.02	20 578 315	**176.95**
livejournal	2 058 441	20.47	**2 058 442**	8.20	2 057 180	7.72	2 057 160	**7.68**
hollywood-2011	**327 857**	28.08	**327 857**	7.30	327 847	7.21	327 847	**7.18**
roadNet-CA	**896 238**	1.45	896 237	**1.18**	892 977	**1.18**	892 811	1.20
youtube	**855 275**	8.27	855 274	1.70	855 199	1.60	855 199	**1.57**
cnr-2000	**225 492**	6.85	**225 492**	1.32	225 405	1.08	225 394	**1.03**
ca-HepPh	**4 986**	0.37	**4 986**	**0.20**	**4 986**	0.21	**4 986**	**0.20**

Table 6.4: Average best solution size s and time t in milliseconds for different update frequencies of DUMIS. Best sizes and times among all update frequencies are marked bold.

6.4 Data Reductions

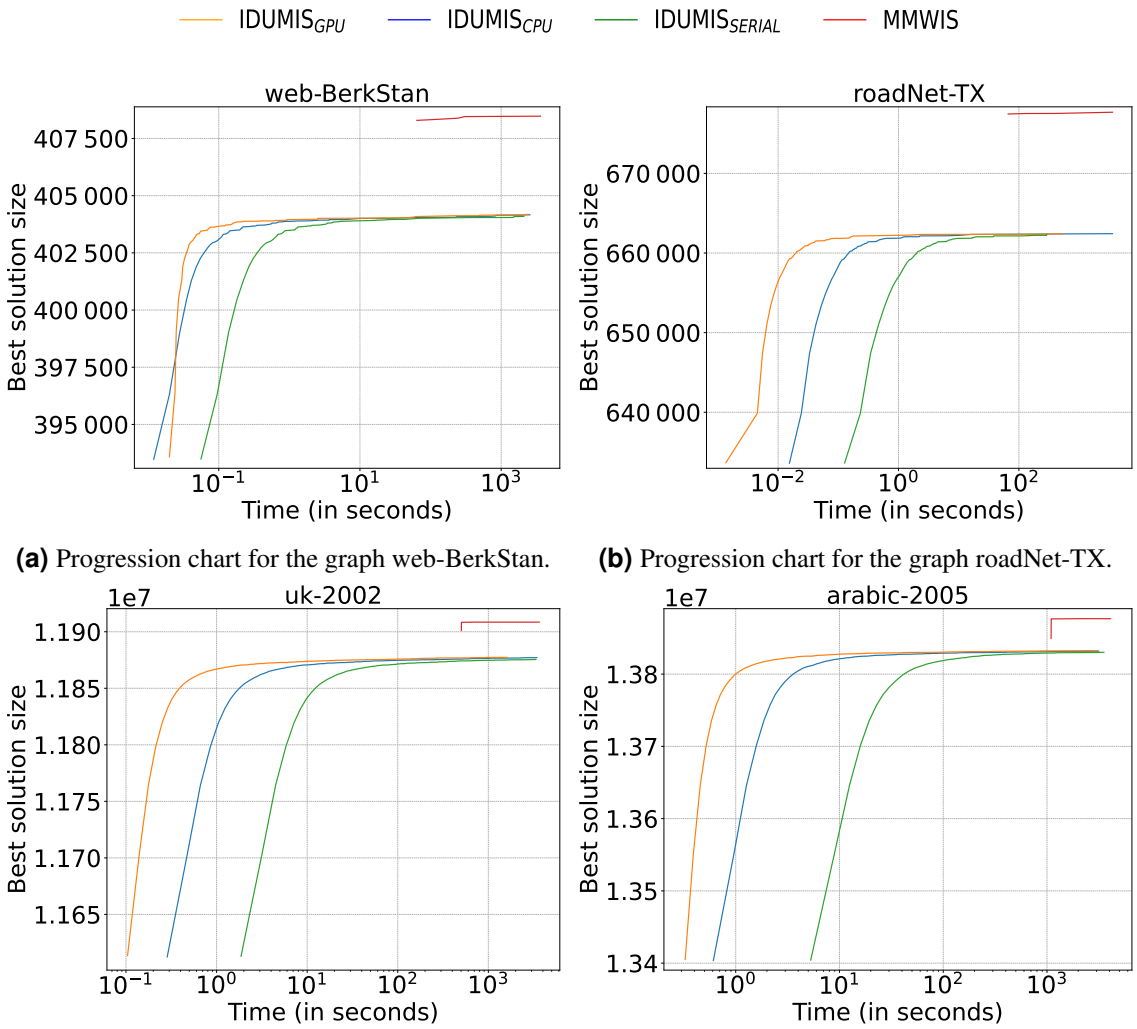
This section analyzes the performance of the data reductions implemented. As previously explained, DUMIS is not affected by the reductions to the same extent as LUMIS. Therefore, we compare LUMIS with and without reductions applied to the graph. The results of the LUMIS algorithm with reductions enabled are denoted with LUMIS$_{RED}$. After applying the reductions to the initial graph, its size is reduced. Therefore, the connectivity of the graph has changed, and it is possible that the reductions can be executed again, reducing the size of the graph even further. We applied the reductions only once, denoted by LUMIS$^1_{RED}$. Furthermore, we applied the reductions repeatedly until no further reductions were possible. This process is denoted by LUMIS$^R_{RED}$.

Table A.2 displays the results of the experiment for each graph instance. Most importantly, it can be observed that the reductions largely increase the sizes of the independent sets produced by LUMIS. However, the execution time increases significantly when reductions are used, especially for larger graphs. The time required for the reductions varies greatly for different graph instances. As expected, applying the reductions repeatedly increases the quality as well as the execution time even further. Considering the high execution times of LUMIS$_{RED}$ for large graph instances, the reductions should only be used in non-time-critical applications or on small graphs. Therefore, it can be advantageous to use LUMIS$^1_{RED}$ instead of LUMIS$^R_{RED}$, depending on the available time. However, the experiments show that GPUs are capable of performing reductions on various graph instances effectively. In addition, we note that LUMIS$^R_{RED}$ solves the five smallest graph instances and the graph hollywood-2011 optimally, showing the potential of implementing data reductions for MIS algorithm.

6.5 Comparison with Competing Algorithms

In this section, the proposed algorithms are compared with the other MIS algorithms, which we mention in Section 6.1. First, we present experiments comparing the proposed algorithms with the serial CPU implementation of MMWIS. Subsequently, experiments that compare the proposed algorithms with the GPU implementation ECL are evaluated.

Figures 6.5a and 6.5b illustrate the development of the maximal independent set size for the different algorithms over time for small and medium-sized graphs. The time limit of all progression experiments was set to one hour for each algorithm. Firstly, it can be observed that IDUMIS converges quickly for all of the different architectures used in the experiment. Moreover, the execution on the GPU is the quickest, followed by the execution on multiple CPU cores. As expected, serial execution of IDUMIS is the slowest of the three architectures. However, IDUMIS converges faster than MMWIS for all architectures. During the later stages of the execution, MMWIS manages to find larger maximal independent sets than IDUMIS.



(a) Progression chart for the graph web-BerkStan.

(b) Progression chart for the graph roadNet-TX.

(c) Progression chart for the graph uk-2002.

(d) Progression chart for the graph arabic-2005.

Figure 6.5: Progression charts for different graph instances that show the development of the computed independent set over time for the algorithms.

For larger instances, shown in Figures 6.5c and 6.5d, a similar performance can be observed. IDUMIS also converges quickly for large instances. However, towards the end of the time limit, MMWIS finds solutions of higher quality than IDUMIS. In summary, the progression charts show that MMWIS outperforms IDUMIS for long time limits. However, IDUMIS shows excellent scalability and convergence speed. Therefore, IDUMIS outperforms MMWIS for short time limits. Furthermore, the results demonstrate the acceleration that can be achieved by the execution of IDUMIS$_{GPU}$ over the other architectures. Further results of this experiment are shown in Figure A.1.

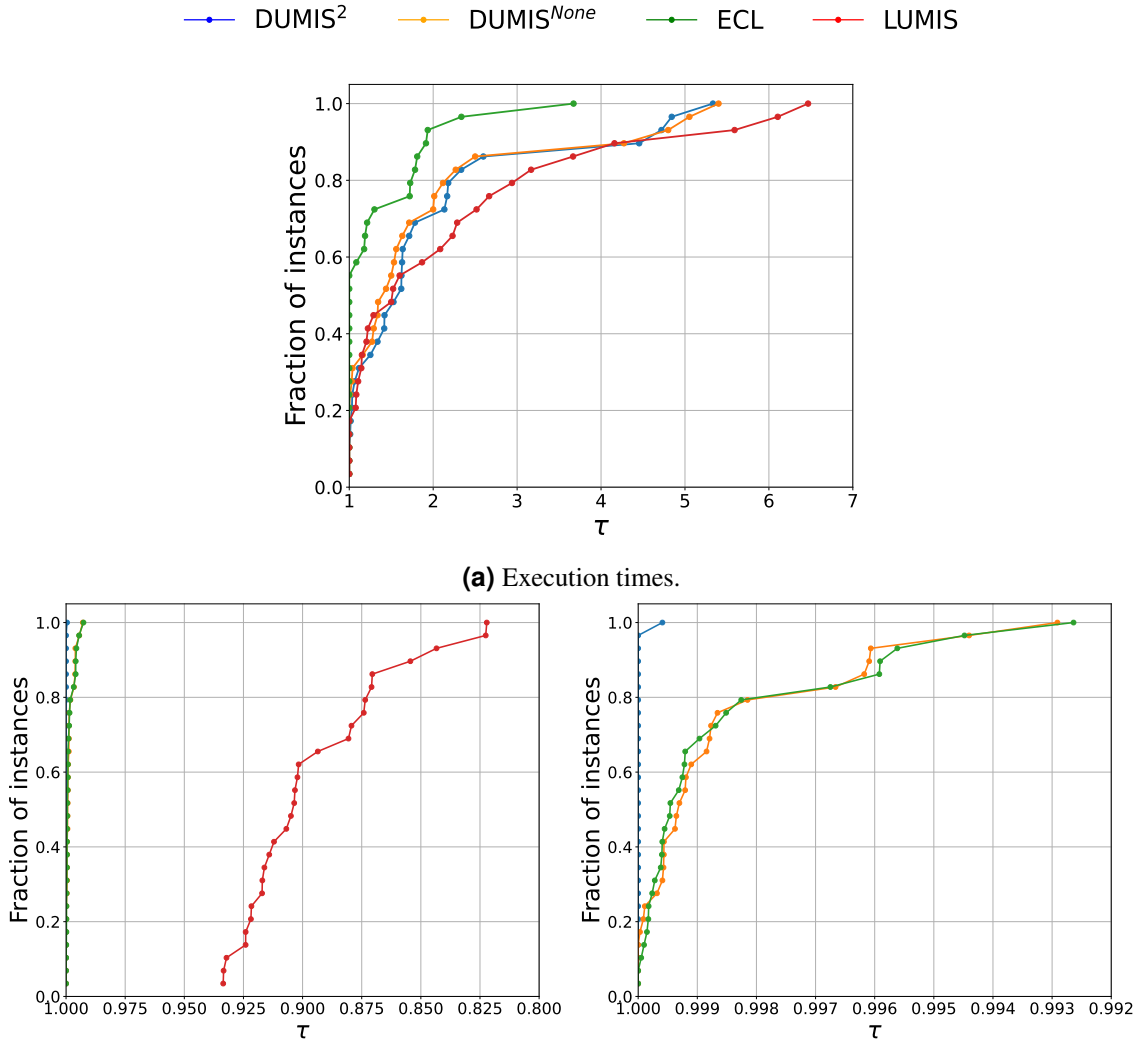
Table A.3 shows the average sizes of the maximal independent sets that were produced by the algorithms within a time limit of one hour for all graph instances. It can be observed that MMWIS produced a significantly larger independent set for nearly all graph instances. Moreover, these results show that the GPU execution achieves the best solutions in comparison to the other architectures on which IDUMIS was executed. Each round on the GPU is notably faster than a round on the CPU (see Table 6.2). Therefore, the GPU execution performs more rounds within the given time limit. Thus, it achieves better solutions for the majority of instances. Furthermore, MMWIS was unable to compute a solution in the given time limit of one hour for four graph instances. The three graphs with the highest number of edges (uk-2005, twitter-2010, friendster) exceed the graph size limit of MMWIS. It can only compute solutions of graphs that have 2^{31} edges or less because it uses 32-bit integers to store edge data. In addition, each algorithm used in the experiments stores both directions of each edge $(u, v) \in E$: (u, v) and (v, u). Therefore, graphs with more than 2^{31} edges exceed that limit.

To analyze the convergence speed of each algorithm, Table A.3 shows the average time required to produce a solution within 99.5% of the size of the largest solution found by each algorithm. The results show that IDUMIS converges significantly quicker than MMWIS. This highlights that IDUMIS outperforms MMWIS for small time limits. However, IDUMIS converges within 1% of the given time limit for nearly all graph instances. Consequently, IDUMIS should be improved to utilize the given time more effectively, especially for large time limits. For example, the reductions explained in Chapter 5 can be applied to the graph before IDUMIS is executed. Half of the given time limit could be used to perform the reductions repeatedly. This adjustment could lead to a more efficient usage of the given time and could increase the independent set size produced by IDUMIS.

Finally, we compare the proposed algorithms executed on the GPU with ECL. ECL only produces one solution when executed. In contrast, IDUMIS produces a large number of solutions within a given time limit. Therefore, ECL is compared to LUMIS and DUMIS in this section, since these algorithms are designed similarly.

We compared the solution size and execution times of the algorithms using performance profiles in Figure 6.6. The graphs uk-2005, twitter-2010, and friendster were excluded from this experiment since ECL is unable to provide a solution for these instances (see Table A.4). Figure 6.6a displays the performance profile for execution times. ECL solves approximately 55% of the instances fastest. In contrast, LUMIS and DUMIS^2 solved approximately 20% of the instances faster than the other algorithms. In addition, ECL shows the least deviation of the fastest time for the instances it did not solve fastest. Therefore, ECL performed best in terms of execution time among the algorithms. Contrary, LUMIS showed the longest execution times with a factor of up to almost 6.5 compared to the best runtime, indicating it performs poorly in this regard. Moreover, DUMIS^2 and DUMIS^{None} achieved almost identical runtimes.

The performance profile for the solution sizes is shown in Figure 6.6b. This figure clearly illustrates that LUMIS achieved the smallest independent sets by a large margin. LUMIS produced solutions ranging from approximately 93 to 82 % of the best solutions. We



(a) Execution times.

(b) Independent set sizes, LUMIS included.

(c) Independent set sizes, LUMIS excluded.

Figure 6.6: Performance profiles for solution size and execution time. ECL cannot solve the graphs uk-2005, twitter-2010, and friendster. Therefore, these instances were excluded from this experiment.

adjusted the limits of the x-axis of the diagram to analyze the performance of the other algorithms more thoroughly. The resulting performance profile is shown in Figure 6.6c. The figure illustrates that DUMIS^2 achieves the best solution sizes for over 95% of the instances. In terms of solution sizes, DUMIS^{None} and ECL perform almost identically. Both algorithms show a decrease in quality of up to 0.7% and solve approximately 80% of the graphs within 99.8% of the best solution size achieved.

Furthermore, the exact results for each graph instance are shown in Table A.4. ECL did not compute a solution for the graphs uk-2005, twitter-2010, and friendster due to its 32-bit integer size limit. The execution time of ECL is significantly lower than the execution times of the proposed algorithms for small and medium-sized graphs. However, for large graph instances, LUMIS and DUMIS compute their solutions faster than ECL. This demonstrates that the proposed algorithms achieve better scalability than ECL.

The experiments conducted identify weaknesses of the proposed algorithms. Especially IDUMIS indicates room for improvement when executed with large time limits. Furthermore, the execution time of LUMIS and DUMIS is relatively long for small graph instances. However, we showed that DUMIS and IDUMIS can outperform other MIS algorithms for certain time limits or graph sizes. In addition, we demonstrated that LUMIS and DUMIS show remarkable scalability.

CHAPTER 7

Discussion

Finally, we reflect on the experiments of this thesis and summarize the issues and advantages of the proposed algorithms in Section 7.1. Thereafter, we provide input for potential future work in Section 7.2.

7.1 Conclusion

In this thesis, we explained in detail how we implemented Luby's algorithm for GPUs using the Kokkos framework. Furthermore, we proposed three algorithms: LUMIS, DUMIS, and IDUMIS. We implemented various improvements to Luby's approach. The main improvement was prioritizing low-degree over high-degree vertices. These improvements led to DUMIS, which produces larger maximal independent sets than LUMIS, while requiring less execution time for most graph instances.

Furthermore, we demonstrated that the frequency of updating the degrees of the vertices in the residual graph has a significant impact on the performance of DUMIS. A higher update frequency requires more runtime for most graph instances, but produces larger maximal independent sets at the same time. DUMIS with a high update frequency produces larger independent sets than ECL. Although DUMIS is slower for small and medium-sized graph instances, DUMIS is capable of outperforming ECL for large graph instances in terms of speed and quality.

In order to improve the produced set sizes even further, we introduced IDUMIS, which calculates a large amount of solutions for the input graph instances. By utilizing this concept, we were able to increase the independent set size over a given time limit. IDUMIS converges quickly compared to MMWIS. This is an advantage for applications where speed is prioritized over quality. However, after IDUMIS converges, it only finds small improvements for the remaining duration of the time limit. Therefore, it is ultimately outperformed by high-quality MIS algorithms after a certain point in time.

Moreover, we introduced known reductions to the MIS problem to LUMIS. We implemented these reductions to be executed on the GPU. We demonstrated that the reductions remarkably improve the size of the independent sets produced by LUMIS. LUMIS is able to solve certain small graph instances optimally due to the reductions. However, especially for large graphs, the reduction process requires a large amount of time. Depending on the input proportions, reductions can be a valuable tool for MIS algorithms.

Additionally, we showed that the proposed algorithms run on the GPU more effectively than on the CPU. For small graph instances, the CPU achieves solutions slightly faster. However, for larger instances, the usage of a GPU significantly accelerated our algorithms. This demonstrates the potential of designing GPU-accelerated algorithms.

7.2 Future Work

This thesis suggests various possibilities for future work. Mainly, the proposed algorithms can be improved even further. Although the reductions introduced improve the quality of LUMIS, they come with a significant computational cost. Therefore, the reductions implemented in the current state can be improved in terms of execution time.

Especially IDUMIS can be improved significantly. While it provides relatively good solutions very quickly, only small improvements are made during the majority of the execution time. Instead of searching for improvements at random, techniques of local search algorithms can be introduced. The current state of IDUMIS could be used to find a large solution using 1% of the available runtime, for instance. The rest of the time, the solution could be strategically improved by local search techniques, resulting in remarkably larger independent sets. Therefore, we hope that future experiments investigate this idea thoroughly.

Furthermore, the size of the input graph is limited by the available memory. Especially for the execution on the GPU, this is an issue, since GPUs have limited memory resources. However, using multiple GPUs simultaneously could solve this issue. This would require an initial step where the graph is partitioned into multiple parts. For each part, the algorithms could then be executed on different GPUs. This could improve the issue of memory limitations. Moreover, the utilization of multiple GPUs has the potential to further accelerate the algorithms. However, this requires additional functions that prevent conflicts that may occur between the different parts of the graph.

In conclusion, it is our hope that our research will lead to more experiments and improved algorithms in the field of the MIS problem in the future. Furthermore, it is our hope that such improvements will lead to an acceleration of the various applications and closely related issues of the MIS problem.

APPENDIX A

Appendix

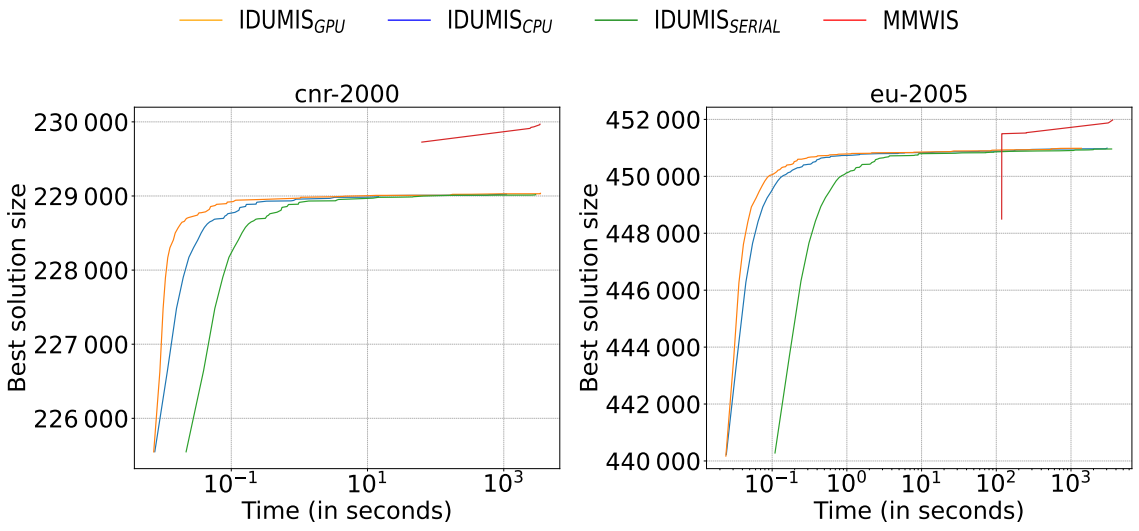
A.1 Implementation Details

As mentioned in Section 4.1, the proposed algorithms were implemented utilizing the Kokkos framework. To execute code on GPUs with Kokkos, it is essential to transfer the graph data from the main memory of the program to the memory of the GPU and vice versa. This is especially important for I/O operations because they cannot be executed by a GPU. Therefore, this process was used throughout the implementation to enable the utilization of input data files and the generation of output files that contain the computed maximal independent set. The data transfer enables the GPU to access and manipulate data that originates from the main memory. After the data is transferred back to the CPU, the data can be stored in an output file or displayed on the console.

Furthermore, the Kokkos framework introduces multidimensional arrays, called Views. Due to their multi-dimensionality, Views can store matrices or higher-dimensional structures. However, the graph data was stored in the CSR format as explained in detail in Section 4.2. Therefore, only one-dimensional Views were used in the implementation, which behave identically to arrays. This was done to minimize the memory usage of the program because GPUs usually have limited memory resources. This allowed us to execute the proposed algorithms for large graph instances.

Moreover, the kernel functions of the implementation are designed to terminate as quickly as possible for each vertex in the graph. In detail, a vertex is only included in the maximal independent set if multiple conditions are fulfilled. As soon as one of those conditions is broken, the thread immediately moves on to the next vertex in the graph that has not been checked yet. This process slightly improves the runtime of the algorithm. However, for large graph instances, this behavior remarkably improves the execution time.

A.2 Further Results



(a) Progression chart for the graph cnr-2000.

(b) Progression chart for the graph eu-2005.

(c) Progression chart for the graph dewiki-2013.

(d) Progression chart for the graph enwiki-2013.

Figure A.1: Progression charts for different graph instances that show the development of the computed independent set over time for the algorithms.

Graphs	LUMIS	DUMIS	IDUMIS	MMWIS
webbase-2001	69 038 402	76 079 178	77 242 628	77 516 755
it-2004	22 543 494	24 881 230	25 511 639	25 608 217
uk-2005	20 588 802	23 116 429	23 626 679	23 684 619
gsh-2015-tpd	18 980 774	20 578 315	20 868 912	20 892 996
arabic-2005	12 225 004	13 387 812	13 832 410	13 884 838
uk-2002	10 488 832	11 603 198	11 877 495	11 914 750
eu-2015-tpd	4 397 816	4 708 783	4 761 632	4 765 583
enwiki-2013	1 837 983	2 030 483	2 162 041	2 174 945
livejournal	1 812 659	2 057 160	2 078 773	2 085 626
hollywood-2011	286 392	327 847	327 947	327 949
roadNet-CA	828 190	892 811	937 370	960 777
as-skitter	1 042 260	1 138 832	1 168 811	1 170 580
dewiki-2013	549 273	611 259	669 477	675 682
in-2004	805 597	878 223	891 482	896 724
roadNet-TX	585 565	631 192	662 433	677 674
youtube	797 264	855 199	857 772	857 945
roadNet-PA	459 449	496 456	520 732	533 208
web-Google	455 562	523 093	528 480	529 138
eu-2005	396 821	439 704	450 995	452 300
amazon-2008	248 554	301 773	307 749	309 793
web-BerkStan	355 572	393 002	404 195	408 478
web-NotreDame	232 584	249 153	251 143	251 849
cnr-2000	206 584	225 394	229 035	230 030
web-Stanford	142 087	154 818	160 815	163 387
ca-CondMat	8 200	9 597	9 612	9 612
ca-AstroPh	5 681	6 735	6 760	6 760
ca-HepPh	4 342	4 986	4 996	4 996
ca-HepTh	4 250	4 876	4 896	4 896
ca-GrQc	2 152	2 453	2 459	2 459

Table A.1: Average solution size of the proposed algorithms compared to the results of MMWIS. MMWIS was executed once with seed 1000 and a time limit of 10 hours. IDUMIS was executed with a time limit of one hour. Graphs for which MMWIS did not produce feasible solutions are excluded. All results of the proposed algorithms were computed by execution on the GPU.

Table A.2: Average solution size s and time t in milliseconds. Best sizes and times among all algorithms are marked bold. Cases in which an algorithm did not find a solution in sufficient time (one hour) are indicated by '—'. All results were computed by execution on the GPU.

	LUMIS		**LUMIS$^1_{RED}$**		**LUMIS$^R_{RED}$**	
Instance	s	t	s	t	s	t
webbase-2001	**69 038 402**	**77.48**	—	—	—	—
friendster	32 474 405	**239.43**	34 115 270	2 564.64	**35 835 640**	79 863.88
twitter-2010	26 555 759	**204.67**	26 790 344	91 057.08	**28 738 422**	130 881.92
it-2004	22 543 494	**131.02**	**24 200 686**	81 835.46	—	—
uk-2005	20 588 802	**130.37**	22 759 514	1 344 810.16	**23 276 587**	185 454.78
gsh-2015-tpd	18 980 774	**131.68**	19 702 388	17 845.31	**20 783 188**	173 853.64
arabic-2005	12 225 004	**72.30**	**13 099 300**	376 891.03	—	—
uk-2002	10 488 832	**25.83**	11 354 708	7 060.51	**11 631 067**	506 381.85
eu-2015-tpd	4 397 816	**16.31**	4 546 966	1 233.36	**4 759 531**	3 179.05
enwiki-2013	1 837 983	**15.01**	1 858 954	1 705.94	**2 089 174**	30 432.13
livejournal	1 812 659	**10.44**	1 976 396	142.42	**2 081 292**	1 871.94
orkut	651 252	**15.03**	662 987	593.11	**669 015**	15 679.29
hollywood-2011	286 392	**7.94**	327 616	28.83	**327 949**	450.65
roadNet-CA	828 190	**1.04**	872 112	1.34	**900 512**	17.51
as-skitter	1 042 260	**3.97**	1 094 638	12 397.02	**1 151 797**	1 443.12
dewiki-2013	549 273	**6.11**	555 399	935.93	**609 218**	40 362.01
in-2004	805 597	**4.82**	863 935	1 944.76	**885 186**	303 568.48
roadNet-TX	585 565	**1.16**	619 287	1.31	**640 708**	12.40
youtube	797 264	**1.74**	850 952	26.47	**857 931**	47.07
roadNet-PA	459 449	**0.97**	485 195	1.23	**501 807**	7.97
web-Google	455 562	**1.51**	510 621	6.97	**526 099**	55.46
eu-2005	396 821	**3.36**	420 904	591.83	**434 768**	199 028.14

Continued on next page

Table A.2 – *Continued from previous page*

	LUMIS		**LUMIS**$^1_{RED}$		**LUMIS**$^R_{RED}$	
Instance	s	t	s	t	s	t
amazon-2008	248 554	**1.23**	277 002	6.05	**302 214**	46.87
web-BerkStan	355 572	**1.83**	381 641	287.04	**395 814**	3 118.70
web-NotreDame	232 584	**0.80**	245 935	407.73	**249 178**	60 866.72
cnr-2000	206 584	**0.99**	219 755	463.95	**223 961**	30 303.68
web-Stanford	142 087	**1.03**	151 055	36.32	**158 190**	6 132.74
ca-CondMat	8 200	**0.22**	9 556	0.50	**9 612**	0.82
ca-AstroPh	5 681	**0.32**	6 682	0.89	**6 760**	2.46
ca-HepPh	4 342	**0.30**	4 944	1.34	**4 996**	2.16
ca-HepTh	4 264	**0.19**	4 841	0.38	**4 896**	0.59
ca-GrQc	2 157	**0.16**	2 445	0.28	**2 459**	0.44

Table A.3: Average solution size s after one hour runtime and average time t in milliseconds to achieve 99.5% of the best solution of each algorithm. Best sizes and times among all algorithms are marked bold. Instances for which an algorithm did not find a solution within the time limit of one hour are indicated by '—'.

	IDUMIS$_{GPU}$		**IDUMIS**$_{CPU}$		**IDUMIS**$_{SERIAL}$		**MMWIS**	
Instance	s	t	s	t	s	t	s	t
webbase-2001	77 242 628	**514.24**	77 240 816	4 535.83	77 236 859	41 569.87	**77 501 671**	1 886 839.66
friendster	36 196 982	**1 729.80**	36 197 014	13 901.04	**36 197 259**	82 606.10	—	—
twitter-2010	**28 702 301**	**2 271.03**	28 702 264	15 577.44	28 702 273	45 299.08	—	—
it-2004	25 511 639	**951.03**	25 508 751	4 127.41	25 506 195	22 816.13	**25 577 685**	2 999 531.86
uk-2005	**23 626 679**	**1 204.71**	23 625 611	3 967.04	23 624 088	17 772.31	—	—
gsh-2015-tpd	**20 868 912**	**1 433.19**	20 868 587	4 188.79	20 868 233	16 238.94	—	—
arabic-2005	13 832 410	**595.53**	13 831 885	2 499.82	13 830 461	14 791.25	**13 876 714**	1 123 095.30
uk-2002	11 877 495	**200.68**	11 877 078	1 057.37	11 875 568	7 287.41	**11 909 624**	544 943.04
eu-2015-tpd	4 761 632	**157.25**	4 761 540	392.54	4 761 487	2 512.32	**4 765 581**	148 636.64
enwiki-2013	2 162 041	**246.87**	2 161 976	811.50	2 161 863	5 499.55	**2 170 001**	219 547.04
livejournal	2 078 773	**34.06**	2 078 756	276.33	2 078 818	1 368.21	**2 085 626**	8 744.31
orkut	**819 525**	**164.09**	819 519	554.63	819 524	3 812.56	806 262	3 115 002.98
hollywood-2011	327 947	**27.42**	327 946	126.64	327 946	685.90	**327 949**	34 926.68
roadNet-CA	937 370	**16.32**	937 242	155.12	937 220	1 245.16	**973 352**	69 760.44
as-skitter	1 168 811	**25.93**	1 168 795	94.56	1 168 766	601.25	**1 170 579**	87 367.38
dewiki-2013	669 477	**81.84**	669 456	410.90	669 410	1 991.27	**671 743**	192 196.47
in-2004	891 482	**26.74**	891 453	59.72	891 453	341.79	**896 724**	11 331.92
roadNet-TX	662 433	**13.80**	662 431	119.24	662 328	1 250.16	**677 705**	66 364.48
youtube	857 772	**7.67**	857 759	18.27	857 760	77.29	**857 945**	553.13
roadNet-PA	520 732	**11.77**	520 719	111.76	520 699	820.94	**533 184**	65 358.52
web-Google	528 480	**6.34**	528 468	23.25	528 465	194.96	**529 138**	911.61
eu-2005	450 995	**51.77**	450 987	78.25	450 952	432.49	**451 994**	98 146.70

Continued on next page

Table A.3 – *Continued from previous page*

	IDUMIS$_{GPU}$		**IDUMIS**$_{CPU}$		**IDUMIS**$_{SERIAL}$		**MMWIS**	
Instance	s	t	s	t	s	t	s	t
amazon-2008	307 749	**5.42**	307 745	35.59	307 723	321.95	**309 632**	66 272.52
web-BerkStan	404 195	**24.83**	404 164	58.68	404 093	302.74	**408 476**	63 749.50
web-NotreDame	251 143	**3.71**	251 145	7.28	251 125	41.2	**251 849**	66 285.16
cnr-2000	229 035	**10.38**	229 025	31.21	229 017	97.06	**229 961**	62 340.18
web-Stanford	160 815	**15.99**	160 792	26.76	160 763	175.24	**163 375**	60 759.26
ca-CondMat	**9 612**	**0.34**	**9 612**	0.48	**9 612**	1.43	**9 612**	3.82
ca-AstroPh	**6 760**	**0.41**	**6 760**	0.56	**6 760**	1.56	**6 760**	3.78
ca-HepPh	**4 996**	0.50	**4 996**	**0.39**	**4 996**	1.03	**4 996**	2.04
ca-HepTh	**4 896**	0.29	**4 896**	**0.25**	**4 896**	0.58	**4 896**	1.06
ca-GrQc	**2 459**	0.32	**2 459**	**0.18**	**2 459**	0.33	**2 459**	0.53

Table A.4: Average solution size s and time t in milliseconds. Best solution size and time among all algorithms are marked bold. Instances for which an algorithm did not find a solution due to graph size limits (32-bit edges) are indicated by '—'. All results were computed by execution on the GPU.

	LUMIS		**DUMIS**^{2}		**DUMIS**^{None}		**ECL**	
Instance	s	t	s	t	s	t	s	t
webbase-2001	69 038 402	**77.48**	**76 128 351**	82.48	76 079 178	79.78	76 086 905	284.53
friendster	32 474 405	**239.43**	**35 880 750**	264.10	35 855 467	249.40	—	—
twitter-2010	26 555 759	**204.67**	**28 182 002**	293.68	28 147 055	279.38	—	—
it-2004	22 543 494	131.02	**24 911 930**	**121.71**	24 881 230	126.44	24 894 824	232.96
uk-2005	20 588 802	130.37	**23 136 326**	105.98	23 116 429	**105.11**	—	—
gsh-2015-tpd	18 980 774	**131.68**	**20 594 678**	187.00	20 578 315	176.95	20 578 562	234.89
arabic-2005	12 225 004	72.30	**13 403 311**	63.52	13 387 812	**62.83**	13 392 648	113.71
uk-2002	10 488 832	25.83	**11 612 577**	23.30	11 603 198	**22.58**	11 606 381	52.75
eu-2015-tpd	4 397 816	**16.31**	**4 710 802**	24.91	4 708 783	23.46	4 708 695	31.56
enwiki-2013	1 837 983	**15.01**	**2 037 282**	18.79	2 030 483	17.41	2 030 659	17.67
livejournal	1 812 659	10.44	**2 058 442**	8.20	2 057 160	7.68	2 056 904	**5.01**
orkut	651 252	15.03	**791 724**	17.59	786 111	16.11	785 895	**9.87**
hollywood-2011	286 392	7.94	**327 857**	7.30	327 847	**7.18**	327 840	12.35
roadNet-CA	828 190	1.04	**896 237**	1.18	892 811	1.20	892 582	**0.25**
as-skitter	1 042 260	3.97	**1 140 208**	3.19	1 138 832	**3.08**	1 138 512	3.66
dewiki-2013	549 273	6.11	**614 699**	5.66	611 259	**5.07**	611 307	6.15
in-2004	805 597	4.82	**878 502**	4.26	878 223	3.72	878 294	**1.64**
roadNet-TX	585 565	1.16	**633 685**	0.92	631 192	0.96	630 908	**0.19**
youtube	797 264	1.74	**855 274**	1.70	855 199	1.57	855 189	**0.78**
roadNet-PA	459 449	0.97	**498 403**	0.80	496 456	0.81	496 364	**0.15**
web-Google	455 562	1.51	**523 321**	1.28	523 093	1.27	523 231	**0.60**
eu-2005	396 821	**3.36**	**440 099**	5.45	439 704	4.34	440 021	5.80

Continued on next page

Table A.4 – *Continued from previous page*

	LUMIS		**DUMIS**^{2}		**DUMIS**^{None}		**ECL**	
Instance	s	t	s	t	s	t	s	t
amazon-2008	248 554	1.23	**302 332**	0.98	301 773	0.94	301 936	**0.22**
web-BerkStan	355 572	1.83	**393 530**	1.31	393 002	1.31	393 122	**0.98**
web-NotreDame	232 584	0.80	**249 181**	0.81	249 153	0.78	249 111	**0.50**
cnr-2000	206 584	0.99	**225 492**	1.32	225 394	1.03	225 406	**0.81**
web-Stanford	142 087	1.03	**154 926**	**0.95**	154 818	0.96	154 656	1.03
ca-CondMat	8 200	0.22	**9 597**	0.14	**9 597**	0.15	**9 597**	**0.06**
ca-AstroPh	5 681	0.32	**6 736**	0.17	**6 736**	0.18	6 735	**0.12**
ca-HepPh	4 342	0.30	**4 986**	**0.20**	**4 986**	**0.20**	4 984	0.26
ca-HepTh	4 264	0.19	4 876	0.13	4 876	0.12	**4 878**	**0.06**
ca-GrQc	2 157	0.16	**2 453**	0.12	**2 453**	0.12	2 452	**0.07**

Zusammenfassung

Das *Problem der maximalen unabhängigen Menge* beschreibt die Aufgabe, eine Menge von Knoten maximaler Kardinalität zu berechnen, sodass keine Knoten der Menge benachbart sind. Da das Problem NP-schwer ist, konzentrieren wir uns in dieser Arbeit auf heuristische Algorithmen. Es gibt mehrere eng verwandte Probleme mit dem Problem der maximalen unabhängigen Menge, darunter das Problem der maximalen Clique und das Problem der minimalen Knotenüberdeckung. Daher gibt es Anwendungen für das Problem in verschiedenen Bereichen der Informatik, wie zum Beispiel in der Computergrafik und der Routenplanung.

In dieser Arbeit entwickeln wir GPU-beschleunigte Algorithmen zur Lösung des Problems der maximalen unabhängigen Menge. Darüber hinaus wenden wir bekannte Verbesserungen an unseren Algorithmen an, was zur Entwicklung unserer drei Algorithmen führt: LUMIS, DUMIS, und IDUMIS. LUMIS basiert auf dem Monte-Carlo-Algorithmus von Luby und bildet die Grundlage unserer Arbeit. Unser verbesserter Algorithmus DUMIS priorisiert hauptsächlich Knoten mit niedrigem Grad gegenüber Knoten mit hohem Grad. Dieses Konzept verbessert die Laufzeit und Qualität unseres Algorithmus. Des Weiteren verbessert IDUMIS eine anfängliche Lösung über einen bestimmten Zeitraum hinweg, indem er zufällige Anpassungen an der vorhergehenden Lösung vornimmt.

Wir haben die Leistung unserer Algorithmen durch zahlreiche Experimente untersucht, indem wir sie mit kürzlich veröffentlichten Algorithmen auf diesem Gebiet verglichen haben. Wir zeigen, dass IDUMIS den MMWIS-Algorithmus für kurze Zeitlimits übertreffen kann. Darüber hinaus ist DUMIS in der Lage, größere unabhängige Mengen zu erzeugen als der ähnlich konzipierte Algorithmus ECL-MIS, während für große Graphinstanzen weniger Ausführungszeit erforderlich ist. Abschließend wird eine Reflexion über die Ergebnisse dieser Arbeit sowie Input für mögliche zukünftige Arbeiten bereitgestellt.

Bibliography

[1] bwUniCluster 2.0: High Performance Computing for Universities in Baden-Württemberg. URL: https://wiki.bwhpc.de/e/BwUniCluster2.0.

[2] KaMIS Source Code. URL: https://github.com/KarlsruheMIS/KaMIS.

[3] Stanford Large Network Dataset Collection. URL: https://snap.stanford.edu/data/.

[4] Laboratory for Web Algorithmics. URL: https://law.di.unimi.it/datasets.php.

[5] Ailsa H. Land and Alison G. Doig. An automatic method of solving discrete programming problems. *Econometrica*, 28(3):497–520, 1960.

[6] Richard M. Karp and Avi Wigderson. A fast parallel algorithm for the maximal independent set problem. *J. ACM*, 32(4):762–773, 1985.

[7] Michael Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM J. Comput.*, 15(4):1036–1053, 1986.

[8] Thomas Bäck and Sami Khuri. An evolutionary heuristic for the maximum independent set problem. In *Proceedings of the First IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence, Orlando, Florida, USA, June 27-29, 1994*, pages 531–535. IEEE, 1994.

[9] Réka Albert, Hawoong Jeong, and Albert-László Barabási. The diameter of the world wide web. *CoRR*, cond-mat/9907038, 1999.

[10] Elizabeth D. Dolan and Jorge J. Moré. Benchmarking optimization software with performance profiles. *Math. Program.*, 91(2):201–213, 2002.

[11] Paolo Boldi, Bruno Codenotti, Massimo Santini, and Sebastiano Vigna. Ubicrawler: A scalable fully distributed web crawler. *Software: Practice & Experience*, 34(8):711–726, 2004.

[12] Paolo Boldi and Sebastiano Vigna. The WebGraph framework I: Compression techniques. In *Proc. of the Thirteenth International World Wide Web Conference (WWW 2004)*, pages 595–601, Manhattan, USA, 2004. ACM Press.

[13] Jure Leskovec, Jon M. Kleinberg, and Christos Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In Robert Grossman, Roberto J. Bayardo, and Kristin P. Bennett, editors, *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Chicago, Illinois, USA, August 21-24, 2005*, pages 177–187. ACM, 2005.

[14] Jure Leskovec, Jon M. Kleinberg, and Christos Faloutsos. Graph evolution: Densification and shrinking diameters. *ACM Trans. Knowl. Discov. Data*, 1(1):2, 2007.

[15] Pedro V. Sander, Diego Nehab, Eden Chlamtac, and Hugues Hoppe. Efficient traversal of mesh edges using adjacency primitives. *ACM Trans. Graph.*, 27(5):144, 2008.

[16] Jure Leskovec, Kevin J. Lang, Anirban Dasgupta, and Michael W. Mahoney. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Math.*, 6(1):29–123, 2009.

[17] Tim Kieritz, Dennis Luxen, Peter Sanders, and Christian Vetter. Distributed time-dependent contraction hierarchies. In Paola Festa, editor, *Experimental Algorithms, 9th International Symposium, SEA 2010, Ischia Island, Naples, Italy, May 20-22, 2010. Proceedings*, volume 6049 of *Lecture Notes in Computer Science*, pages 83–93. Springer, 2010.

[18] Haewoon Kwak, Changhyun Lee, Hosung Park, and Sue B. Moon. What is twitter, a social network or a news media? In Michael Rappa, Paul Jones, Juliana Freire, and Soumen Chakrabarti, editors, *Proceedings of the 19th International Conference on World Wide Web, WWW 2010, Raleigh, North Carolina, USA, April 26-30, 2010*, pages 591–600. ACM, 2010.

[19] Paolo Boldi, Marco Rosa, Massimo Santini, and Sebastiano Vigna. Layered label propagation: A multiresolution coordinate-free ordering for compressing social networks. In Sadagopan Srinivasan, Krithi Ramamritham, Arun Kumar, M. P. Ravindra, Elisa Bertino, and Ravi Kumar, editors, *Proceedings of the 20th international conference on World Wide Web*, pages 587–596. ACM Press, 2011.

[20] Diogo Vieira Andrade, Mauricio G. C. Resende, and Renato Fonseca F. Werneck. Fast local search for the maximum independent set problem. *J. Heuristics*, 18(4):525–547, 2012.

[21] Nicolas Bourgeois, Bruno Escoffier, Vangelis Th. Paschos, and Johan M. M. van Rooij. Fast algorithms for max independent set. *Algorithmica*, 62(1-2):382–415, 2012.

[22] Paolo Boldi, Andrea Marino, Massimo Santini, and Sebastiano Vigna. BUbiNG: Massive crawling for the masses. In *Proceedings of the Companion Publication of the 23rd International Conference on World Wide Web*, pages 227–228. International World Wide Web Conferences Steering Committee, 2014.

[23] H. Carter Edwards, Christian R. Trott, and Daniel Sunderland. Kokkos: Enabling manycore performance portability through polymorphic memory access patterns. *J. Parallel Distributed Comput.*, 74(12):3202–3216, 2014.

[24] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, jun 2014.

[25] Sebastian Lamm, Peter Sanders, and Christian Schulz. Graph partitioning for independent sets. In Evripidis Bampis, editor, *Experimental Algorithms - 14th International Symposium, SEA 2015, Paris, France, June 29 - July 1, 2015, Proceedings*, volume 9125 of *Lecture Notes in Computer Science*, pages 68–81. Springer, 2015.

[26] Jaewon Yang and Jure Leskovec. Defining and evaluating network communities based on ground-truth. *Knowl. Inf. Syst.*, 42(1):181–213, 2015.

[27] Takuya Akiba and Yoichi Iwata. Branch-and-reduce exponential/fpt algorithms in practice: A case study of vertex cover. *Theor. Comput. Sci.*, 609:211–225, 2016.

[28] Jakob Dahlum, Sebastian Lamm, Peter Sanders, Christian Schulz, Darren Strash, and Renato F. Werneck. Accelerating local search for the maximum independent set problem. In Andrew V. Goldberg and Alexander S. Kulikov, editors, *Experimental Algorithms - 15th International Symposium, SEA 2016, St. Petersburg, Russia, June 5-8, 2016, Proceedings*, volume 9685 of *Lecture Notes in Computer Science*, pages 118–133. Springer, 2016.

[29] Darren Strash. On the power of simple reductions for the maximum independent set problem. *CoRR*, abs/1608.00724, 2016.

[30] Lijun Chang, Wei Li, and Wenjie Zhang. Computing A near-maximum independent set in linear time by reducing-peeling. In Semih Salihoglu, Wenchao Zhou, Rada Chirkova, Jun Yang, and Dan Suciu, editors, *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017*, pages 1181–1196. ACM, 2017.

[31] Mingyu Xiao and Hiroshi Nagamochi. Exact algorithms for maximum independent set. *Inf. Comput.*, 255:126–146, 2017.

[32] Martin Burtscher, Sindhu Devale, Sahar Azimi, Jayadharini Jaiganesh, and Evan Powers. A high-quality and fast maximal independent set implementation for gpus. *ACM Trans. Parallel Comput.*, 5(2):8:1–8:27, 2018.

[33] Xuanhua Shi, Zhigao Zheng, Yongluan Zhou, Hai Jin, Ligang He, Bo Liu, and Qiang-Sheng Hua. Graph processing on gpus: A survey. *ACM Comput. Surv.*, 50(6):81:1–81:35, 2018.

[34] Tomohiro Imanaga, Koji Nakano, Masaki Tao, Ryota Yasudo, Yasuaki Ito, Yuya Kawamata, Ryota Katsuki, Yusuke Tabata, Takashi Yazane, and Kenichiro Hamano. Efficient GPU implementation for solving the maximum independent set problem. In *Eighth International Symposium on Computing and Networking, CANDAR 2020, Naha, Japan, November 24-27, 2020*, pages 29–38. IEEE, 2020.

[35] Chengzhi Piao, Weiguo Zheng, Yu Rong, and Hong Cheng. Maximizing the reduction ability for near-maximum independent set computation. *Proc. VLDB Endow.*, 13(11):2466–2478, 2020.

[36] Laurent Bulteau, Bertrand Marchand, and Yann Ponty. A new parametrization for independent set reconfiguration and applications to RNA kinetics. In Petr A. Golovach and Meirav Zehavi, editors, *16th International Symposium on Parameterized and Exact Computation, IPEC 2021, September 8-10, 2021, Lisbon, Portugal*, volume 214 of *LIPIcs*, pages 11:1–11:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[37] Demian Hespe, Sebastian Lamm, and Christian Schorr. Targeted branching for the maximum independent set problem. In David Coudert and Emanuele Natale, editors, *19th International Symposium on Experimental Algorithms, SEA 2021, June 7-9, 2021, Nice, France*, volume 190 of *LIPIcs*, pages 17:1–17:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[38] Christian Trott, Luc Berger-Vergiat, David Poliakoff, Sivasankaran Rajamanickam, Damien Lebrun-Grandié, Jonathan R. Madsen, Nader Al Awar, Milos Gligoric, Galen M. Shipman, and Geoff Womeldorff. The kokkos ecosystem: Comprehensive performance portability for high performance computing. *Comput. Sci. Eng.*, 23(5):10–18, 2021.

[39] Eugenio Angriman, Alexander van der Grinten, Michael Hamann, Henning Meyerhenke, and Manuel Penschuck. Algorithms for large-scale network analysis and the networkit toolkit. In Hannah Bast, Claudius Korzen, Ulrich Meyer, and Manuel Penschuck, editors, *Algorithms for Big Data - DFG Priority Program 1736*, volume 13201 of *Lecture Notes in Computer Science*, pages 3–20. Springer, 2022.

[40] Christian R. Trott, Damien Lebrun-Grandié, Daniel Arndt, Jan Ciesko, Vinh Q. Dang, Nathan D. Ellingwood, Rahulkumar Gayatri, Evan Harvey, Daisy S. Hollman, Dan Ibanez, Nevin Liber, Jonathan R. Madsen, Jeff Miles, David Poliakoff, Amy Powell, Sivasankaran Rajamanickam, Mikael Simberg, Dan Sunderland, Bruno Turcksin, and Jeremiah J. Wilke. Kokkos 3: Programming model extensions for the exascale era. *IEEE Trans. Parallel Distributed Syst.*, 33(4):805–817, 2022.

[41] Enqiang Zhu, Yu Zhang, and Chanjuan Liu. An adaptive repeated-intersection-reduction local search for the maximum independent set problem. *CoRR*, abs/2208.07777, 2022.

[42] Oualid Elissaouy and Karam Allali. Minimizing the maximum tardiness for a permutation flow shop problem under the constraint of sequence independent setup time. *RAIRO Oper. Res.*, 58(1):373–395, 2024.

[43] Ernestine Großmann, Sebastian Lamm, Christian Schulz, and Darren Strash. Finding near-optimal weight independent sets at scale. *J. Graph Algorithms Appl.*, 28(1):439–473, 2024.

[44] Kenneth Langedal, Demian Hespe, and Peter Sanders. Targeted branching for the maximum independent set problem using graph neural networks. In Leo Liberti, editor, *22nd International Symposium on Experimental Algorithms, SEA 2024, July 23-26, 2024, Vienna, Austria*, volume 301 of *LIPIcs*, pages 20:1–20:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024.

[45] NVIDIA Corporation. *CUDA Toolkit Documentation*, 2023. Version 12.2, Available at `https://developer.nvidia.com/cuda-toolkit`.

[46] M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.