UNIVERSITATEA ”ALEXANDRU-IOAN CUZA” DIN IAȘI

FACULTATEA DE INFORMATICĂ

LUCRARE DE LICENȚĂ

A Heuristic Solver for the Directed Feedback Vertex Set Problem

propusă de

Andrei Arhire

Sesiunea: iulie, 2022

Coordonator științific

Lect. Dr. Paul Diac

UNIVERSITATEA ”ALEXANDRU-IOAN CUZA” DIN IAȘI

FACULTATEA DE INFORMATICĂ

A Heuristic Solver for the Directed Feedback Vertex Set Problem

Andrei Arhire

Sesiunea: iulie, 2022

Coordonator științific

Lect. Dr. Paul Diac

Avizat,

Îndrumător lucrare de licență,

Lect. Dr. Paul Diac.

Data: ..... Semnătura: .....

Declarație privind originalitatea conținutului lucrării de licență

Subsemnatul **Arhire Andrei** domiciliat în **România, jud. Galați, mun. Tecuci, strada Vasile Alecsandri, nr. 1, bl. U, et. 1, ap. 46**, născut la data de **27 august 2000**, identificat prin CNP **5000827171698**, absolvent al Facultății de Informatică, **Facultatea de Informatică** specializarea **Informatică**, promoția 2019-2022, declar pe propria răspundere cunoscând consecințele falsului în declarații în sensul art. 326 din Noul Cod Penal și dispozițiile Legii Educației Naționale nr. 1/2011 art. 143 al. 4 și 5 referitoare la plagiat, că lucrarea de licență cu titlul **A Heuristic Solver for the Directed Feedback Vertex Set Problem** elaborată sub îndrumarea domnului **Lect. Dr. Paul Diac**, pe care urmează să o susțin în fața comisiei este originală, îmi aparține și îmi asum conținutul său în întregime.

De asemenea, declar că sunt de acord ca lucrarea mea de licență să fie verificată prin orice modalitate legală pentru confirmarea originalității, consimțind inclusiv la introducerea conținutului ei într-o bază de date în acest scop.

Am luat la cunoștință despre faptul că este interzisă comercializarea de lucrări științifice în vederea facilitării falsificării de către cumpărător a calității de autor al unei lucrări de licență, de diplomă sau de disertație și în acest sens, declar pe proprie răspundere că lucrarea de față nu a fost copiată ci reprezintă rodul cercetării pe care am întreprins-o.

Data: ..... Semnătura: .....

Declarație de consimțământ

Prin prezenta declar că sunt de acord ca lucrarea de licență cu titlul **A Heuristic Solver for the Directed Feedback Vertex Set Problem**, codul sursă al programelor și celelalte conținuturi (grafice, multimedia, date de test, etc.) care însoțesc această lucrare să fie utilizate în cadrul Facultății de Informatică.

De asemenea, sunt de acord ca Facultatea de Informatică de la Universitatea ”Alexandru-Ioan Cuza” din Iași, să utilizeze, modifice, reproducă și să distribuie în scopuri necomerciale programele-calculator, format executabil și sursă, realizate de mine în cadrul prezentei lucrări de licență.

Absolvent **Andrei Arhire**

Data: ..... Semnătura: .....

Contents

Motivation

The Directed Feedback Vertex Set is an $\mathcal{NP} - complete$ problem with various applications. I decided to research the scientific advances and current solutions that approach the problem exactly and heuristically. At the same time, encouraged by this year's edition of PACE [1], I have created a solver that finds results close to the global optimum in a short time. Through the competition, I examined the solution's performance in large instances, built on different patterns inspired by the real world, and compared the results obtained with those of other competitors. At the end of the competition, I made a short paper[2] which I registered at a conference.

Chapter 1

Introduction

Competitive Programming can be considered a mind sport, just like chess. In general, it refers to solving as quickly as possible a large percentage of a set of problems of an algorithmic nature. The scientific committee has a set of correct solutions for each problem and demonstrations. In essence, the process of solving a problem consists of 2 stages. The first stage is the design of the algorithm used, and the second is the implementation of ideas for the problem's constraints. Thus, depending on the time limits, the memory, and the dimensions of the input data, the competitor will carefully choose the data structures to be used and develop a program with a complexity good enough to pass all the evaluation tests under the specified conditions. The competitor will use his knowledge of mathematics, algorithms, and data structures in the first phase, after which he will choose a programming language that he masters and implement the solution in a straightforward and fast way. The primary skills that a person with experience in competitive programming acquires are solving complicated problems, working in a team, working under stressful conditions, efficiently managing time and deadlines, and minimizing errors. It also shows that the person is disciplined, focused, and fast, which are essential skills. This way, a person with experience in competitive programming can deliver high-quality software products better and deal with research problems.

1.1 Classical Contests

Today, competitive programming is more popular and more accessible than ever. Classic contests mean competitions in which, for each requirement in a problem, either

the total associated score is awarded if the competitor solves correctly, or no points are awarded at all otherwise. Internationally, the most prestigious competitions include those for students in schools, such as the International Olympiad in Informatics, the Baltic Computer Science Olympiad, the Central European Computer Science Olympiad, those for university students, as the International Collegiate Programming Contest, and those open to the general public such as FIICode, Facebook Hacker Cup or Google Code Jam.

1.2 Optimization Challenges

Another type of programming competition closely related to competitive programming is those in which an optimization problem is given. Competitors must develop a program that finds a solution as close as possible to the global optimum. Often, the problem only supports exponential complexity algorithms, and competitors create solutions that use mathematical observations, greedy assumptions, genetic algorithms, or neural networks. Perhaps the most famous such contest is Google Hash Code. Other important international competitions include Parameterized Algorithms and Computational Experiments (PACE), International Student Competition in Structural Optimization (ISCSO), DIMACS Implementation Challenge, Model Counting Competition, and several competitions organized by Huawei. The problem may be a variation of a classic $\mathcal{NP} - hard$ problem chosen to encourage the public to become familiar with the problem and come up with new observations that will later materialize into new efficient algorithms that can solve it. Also, some competitions are organized by companies that bring real problems encountered by them in the industry. For example, in the last two years, Huawei has organized a tournament composed only of optimization problems.

Chapter 2

Pace Challenge

Parameterized Algorithms and Computational Experiments (PACE) Challenge is an annual optimization competition that started in 2016. Anyone can participate alone or as a team member, although most participants are researchers and Ph.D. candidates. University professors from more than 9 European countries form the scientific committee. So far, the committee has chosen the following challenges: **Cluster Editing**, **Treedepth**, **Vertex Cover**, **Hypertree Width**, **Steiner Tree**, **Minimum Fill-In** and **Undirected Feedback Vertex Set** for which the participants had to develop a heuristic track and or an exact one.

The challenge of the 7th edition is **Directed Feedback Vertex Set**. A team can compete with up to 3 solvers who do not share a common code base. For each track, 200 instances are created. Half are public and represent the tests based on which the preliminary ranking is created. After submission, the competitor's sources are re-evaluated, but this time on all 200 tests. Based on the final results, those in the first place are invited to the Award ceremony at the International Symposium on Parameterized and Exact Computing (IPEC 2022) in September, when the challenge of the next edition will be announced.

For both the exact and the heuristic version, a light version of the contest is provided, where the ranking is made only by the ten smallest instances from the public ones. The running time is much shorter than in the official competition, the idea being to familiarize users with the submit system and test specific solutions quickly and compare them. In the official competition, the program is limited to 8 GB of memory per heap, 8 MB per stack, 10 minutes running time for the program, and 6 hours time penalty between two consecutive submissions. Communication with the tech-

nical and scientific committee members is encouraged during the competition. The ranking is built on the last submission of each participant (not the best on public tests). The period in which submissions can be made is about three months, and in the last days, although the participants can send their sources, the scoreboard remains frozen. After the deadline for submitting the code, two weeks are available for the descriptions of the solutions to be uploaded through easychair.org. The descriptions of the best programs will have the chance to be published at a conference.

The goal of the project can be found described on PACE website in [1]. Due to this project, highly appreciated papers were born such as [21], [20], [9], [5].

Chapter 3

Directed Feedback Vertex Set Problem

A feedback vertex set of a graph is a set of nodes with the property that each cycle contains at least one vertex from the set, i.e., the removal of all vertices from a feedback vertex set leads to an acyclic graph. This problem is one of the first shown to be $\mathcal{NP} - complete$, being part of Karp's 21 $\mathcal{NP} - complete$ problems[13]. It is essential to solving the problem quickly because of its applications in various areas, such as Bayesian inference [17] and deadlock recovery. It is an element to characterize the behavior of modeled biological systems by differential equations [6], [8] and is significant in the model reduction of Boolean Networks [14].

The decision problem for FVS is as follows:

FEEDBACK VERTEX SET	
Instance:	An (undirected or directed) graph $G = (V, E)$ and a positive integer k.
Parameter:	k
Question:	Does there exist a vertex subset of size at most k that intersects every cycle in G?

A parameterized problem is called fixed-parameter tractable if it can be solved in $\mathcal{O}(f(k) \cdot p(n))$, where n is related to the size of the problem instance, p is a polynomial function, k is a fixed parameter, and f is an arbitrary function that can be computed. In $\mathcal{NP} - complete$ problems, the function $f(k)$ is not expected to be polynomial. An equivalent parameterized problem (T, q) is a kernel for (G, k) if it can be reduced from it in a time determined only by k, where q is obtained only from k, and H is bounded by a function dependent only by k.

In 2008, Chen et al. proposed an FPT branching-based algorithm [11] with a running time the $4^k k \cdot n^{\mathcal{O}(1)}$ after 15 years when the FPT Status of DFVS had been an open problem. Recently, Benjamin Bergougnoux et al. have made significant progress in resolving the existence of a polynomial kernel, opening new directions for future research [19].

Let us talk about some heuristic approaches to the problem. There must be mention at least three papers that solve the problem through a simulated annealing algorithm [18], a genetic algorithm [7], and a greedy algorithm that uses reductions and include a randomized local search [15]. I implemented a solver for each of the three approaches. I made comparisons using custom tests, tests from problems on the infoarena educational archive, and the 100 public instances from the competition.

A repository where these programs are found along with other auxiliaries such as a checker to verify if the output is a Directed Feedback Vertex Set or another that helps to read the data in the format provided by the organizers can be found in the **pace-2022** release on GitHub [3].

Of all these algorithms, the last one proved to be the most efficient in terms of time and quality of the result. In the continuation of the thesis, the solver I created based on the last approach mentioned above is described and analyzed. Also, a shorter presentation about it, the one submitted to the conference, is loaded in [2] and the source code is available on GitHub [3], and Zenodo [4]

Chapter 4

High-Level Description

The algorithm I used can be structured in 3 stages to be easier to understand. In the first stage, a solution set is obtained by alternating two processes until the graph becomes empty. The first consists of reducing the graph to one with fewer vertices or edges, which allows a common optimum. The second process consists of the heuristic selection of a node that must be part of the solution. In the second stage, I try eliminating the redundant nodes from the solution obtained in the previous step. In the last part, several local searches are performed starting from a subset of the best-known solution.

The problem can be viewed in the following way. I have to place each vertex in a set A or a set B such that at the end, A is a minimum feedback vertex set and B is the acyclic remainder. A group of vertices can be placed in A or B based on some verification done in polynomial time.

4.1 Preliminaries

Let $G = (V, E)$ be a directed graph.

- $N_G^-(v) = \{u \in V \mid \exists (u, v) \in E \wedge \nexists (v, u) \in E\}$, $\quad d_G^-(v) = |N_G^-(v)|$;
- $N_G^+(v) = \{u \in V \mid \nexists (u, v) \in E \wedge \exists (v, u) \in E\}$, $\quad d_G^+(v) = |N_G^+(v)|$;
- $N_G^\pm(v) = \{u \in V \mid \exists (u, v) \in E \wedge \exists (v, u) \in E\}$, $\quad d_G^\pm(v) = |N_G^\pm(v)|$;
- $G - v = (V \backslash \{v\}, E \cap \{\{V \backslash \{v\}\} \times \{V \backslash \{v\}\}\})$;
- $G \circ v$ = $(V \backslash \{v\}, E \cap \{\{\{V \backslash \{v\}\} \times \{V \backslash \{v\}\}\} \cup \{\{N_G^-(v) \cup N_G^\pm(v)\} \times \{N_G^+(v) \cup N_G^\pm(v)\}\}\})$;
- G is a diclique if $\forall x, y \in V, x \neq y$ and $\{(x, y), (y, x)\} \subset E$.

For a node v, elements of $\{N_G^-(v) \cup N_G^\pm(v)\}$ are named predecessors and elements $\{N_G^-(v) \cup N_G^\pm(v)\}$ are named successors. I will refer to $G - v$ operation as vertex v removal (remove v together with all its adjacent edges) and to $G \circ v$ operation as vertex v merger (connect all its predecessors with all its successors and exclude v from the graph). Also, during the paper, by (D/M)FVS, I will refer to the (Directed/Minimum) Feedback Vertex Set.

4.2 Reduction Rules

The following two operations are considered to be reductions:

- If a vertex can be part of set A without affecting the solution's optimality, I remove it and eventually introduce it in the solution set.
- If a vertex can be part of set B without affecting the solution's optimality, then I merge it.

We make usage of 8 reduction rules described in [16], [12] and [15].

Reduction Rule 1. *If there exists a vertex $v \in V$ and an edge $(v, v) \in E$, remove v.*

In this case, vertex v contains a self-loop. It is erased and inserted into the solution.

Reduction Rule 2. *If there exists a vertex* $v \in V$, $(v, v) \notin E$ *with* $d_G^-(v) + d_G^\pm(v) \leq 1$ *or* $d_G^+(v) + d_G^\pm(v) \leq 1$, *merge* v.

Vertex v can have $d_G^+(v) = 0$ or $d_G^-(v) = 0$, then it can not be part of any cycle. Otherwise, $d_G^+(v) = 1$ or $d_G^-(v) = 1$, connecting all its predecessors with all its successors and excluding v from the graph (merge operation) will not affect the optimality. In both scenarios v is not considered in the solution.

In example below vertex Z is merged because $d_G^-(Z) = 0$ and $d_G^-(v) = 1$.

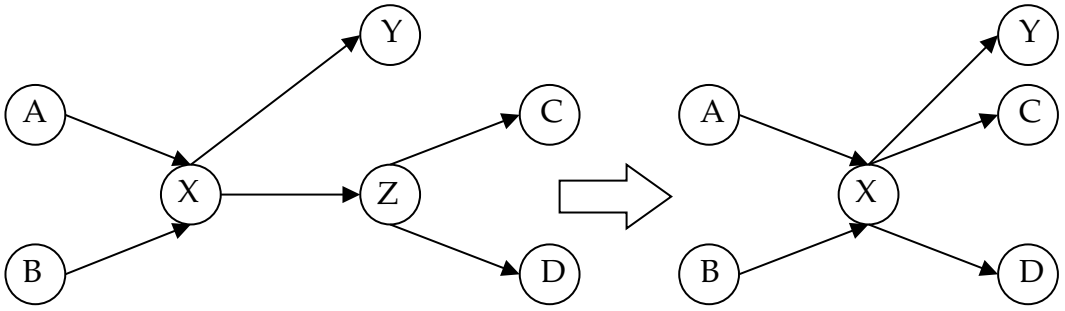


Fig. 1. Concept of the Second Reduction

Reduction Rule 3. *If there exists a vertex* $v \in V$, $(v, v) \notin E$, $\min\left(d_G^-(v), d_G^+(v)\right) = 0$ *and* $N_G^\pm(v)$ *forms a diclique, remove* $N_G^\pm(v)$ *and merge* v.

The critical observation here is that in a diclique, at most, one node will not be part of the solution because any two vertices form a cycle. Thus, the diclique can be viewed as a single node, so rule two can be applied further.

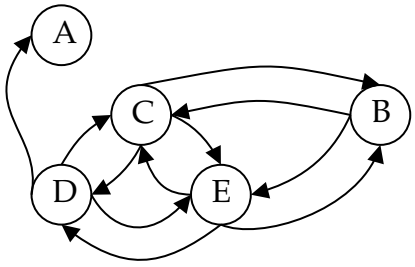


Fig. 2. Concept of the Third Reduction

Node D ca be merged and not included in $DFVS$, $\{C, E\}$ can be removed and included in $DFVS$.

Reduction Rule 4. *If there exist vertices* $u, v \in V$, $(u, v) \in E \wedge (v, u) \notin E$ *and* u *and* v *are in different strongly connected components of the graph* $(V, E \backslash \{E \cap \{(x, y), \forall x \in V, y \in N_G^\pm(x)\}\})$, *erase* (u, v).

Edges between vertices in different strongly connected components are not part of any cycle (otherwise, the vertices would be in the same component). Thus all these

edges can be deleted. Furthermore, in a diclique with two vertices, at least one node will be removed, so edges that are part of a diclique can be ignored when strongly connected components are computed.

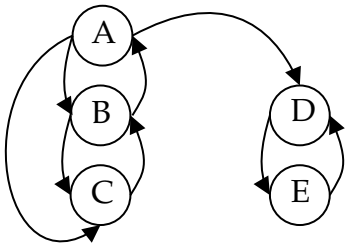


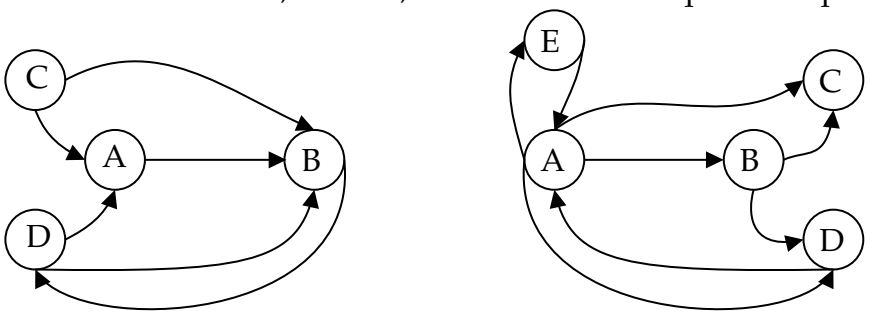
Fig. 3. Concept of the Fourth Reduction

Edges (A, C) and (A, D) can be removed.

Reduction Rule 5. *If there exist vertices* $u, v \in V$, $(u, v) \in E \wedge (v, u) \notin E$ *and* $(N_G^-(u) \subset \{N_G^-(v) \cup N_G^{\pm}(v)\}) \vee (N_G^+(v) \subset \{N_G^+(u) \cup N_G^{\pm}(u)\})$, *erase* (u, v).

The idea with this rule is to delete a set of edges with the property that there is no minimal cycle using them since only minimal cycles need to be broken to compute the feedback vertex set.

The following three reduction rules are obtained based on reduction rule three together with the idea that, at most, one node in a diclique is not part of the solution.



(a) (b)

Fig. 4. Concept of the Fifth Reduction

In both examples, (A, B) can be erased.

Reduction Rule 6. *If there exists a node* $v \in V$, $(v, v) \notin E$ *such that* $\{N_G^+(v) \cup N_G^{\pm}(v)\}$ *or* $\{N_G^-(v) \cup N_G^{\pm}(v)\}$ *forms a diclique, merge* v.

Let's consider an example where $\{N_G^+(v) \cup N_G^{\pm}(v)\}$ is a diclique and $N_G^-(v) \neq \emptyset$. Since $\{N_G^+(v) \cup N_G^{\pm}(v)\}$ is diclique, it can be compressed to at most one node. That node can be either part of $N_G^{\pm}(v)$ or $N_G^+(v)$. In both cases node v can be reduced using second reduction. Such an example is shown in Fig. 5.

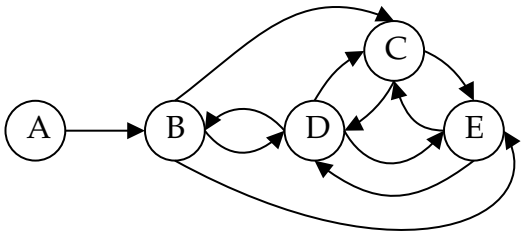


Fig. 5. Concept of the Sixth Reduction

Vertex B can be merged and not included into $DFVS$.

Reduction Rule 7. *If there exists a node $v \in V$, $(v, v) \notin E$ and $\{N_G^-(v) \cup N_G^+(v) \cup N_G^{\pm}(v)\}$ can be split in two sets, $N_G^{\pm}(v)$ is included in one of them and each set forms a diclique, merge v.*

That diclique containing $N_G^{\pm}(v)$ can be reduced to a single vertex, having at most two edges with v.

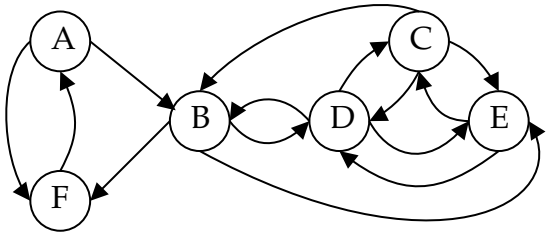


Fig. 6. Concept of the Seventh Reduction

Vertex B can be merged and not included into $DFVS$.

Reduction Rule 8. *If there exists a node $v \in V$, $(v, v) \notin E$, $d_G^{\pm}(v) = 0$ and $\{N_G^+(v) \cup N_G^-(v)\}$ can be split in at most three sets and each set forms a diclique, merge v.*

Each diclique can be reduced to a single vertex. Therefore v will have at most three edges, one for each successor or predecessor.

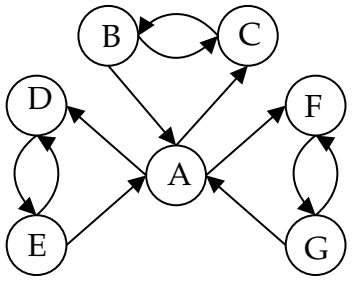


Fig. 7. Concept of the Eighth Reduction

Vertex A can be merged and not included into $DFVS$.

4.3 Stage I

In the first part of the algorithm, reduction rules are applied until the graph no longer supports any. If the graph is not empty, a promising vertex is selected to be part of the solution and removed. These two operations are performed until the graph becomes empty. A vertex v is considered the best candidate in the selection process if it maximizes among all the other vertices either $(d_G^+(v) + d_G^\pm(v)) \cdot (d_G^-(v) + d_G^\pm(v))$ or $d_G^\pm(v) \cdot \infty + d_G^-(v) \cdot d_G^+(v)$. The higher the value in the function, the more promising the node is to be part of the solution set. The first criterion is obtained based on two observations: a node with many neighbors is more likely to be part of many cycles. Therefore, it is chosen to the detriment of another node with few neighbors. At the same time, a node for which the difference between the internal and external degree is small will be chosen in favor of a node with the same number of neighbors but for which the difference is greater. This is because the node with the big difference is closer to the situation where it loses enough neighbors to be reduced by a reduction operation. The second criterion chooses the vertex with the highest $d_G^\pm$ value. In the case of a tie, the same criterion described earlier is applied. Other criterion that I used and obtained a satisfactory result but lower than the two above is $(d_G^+(v) + d_G^-(v) + d_G^\pm(v)) - \alpha \cdot |d_G^+(v) + d_G^-(v)|, \alpha \in (0, 1]$.

Although these three lead to excellent results, they do not always make the best choice.

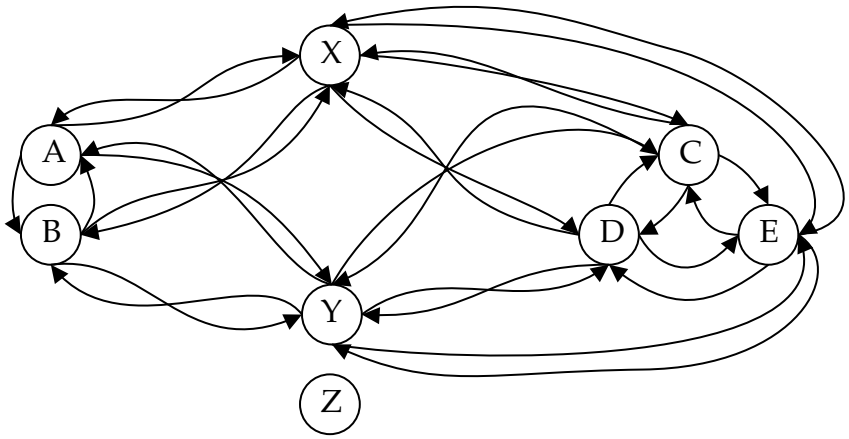


Fig. 8. Example of a Heuristic Selection Failure

$\{A, B\}$ forms a 2-diclique, $d_G^\pm(A) = d_G^\pm(B) = 3$;

$\{C, D, E\}$ forms a 3-diclique, $d_G^\pm(C) = d_G^\pm(D) = d_G^\pm(E) = 4$;

$\{X, Y\}$ are not part of any diclique, $d_G^\pm(X) = d_G^\pm(Y) = 5$.

According to the above criteria, nodes X and Y will be removed first, the rest acyclic consisting of a node of $\{A, B\}$ and a node of $\{C, D, E\}$. If I also have a node Z whose set of neighbors is the same as that of X or Y, the result obtained by applying the criteria would not be optimal because you would have removed the nodes of maximum degree firstly, i.e., X, Y and Z, and the cardinal of the solution would have been 6 (for example $\{X, Y, Z, A, C, D\}$). But the optimal solution contains 5 nodes, namely $\{A, B, C, D, E\}$.

Algorithm 1 Stage I

Require: $G-$ a directed graph, $\alpha \in (0, 1]$

```
F ← ∅
E ← getNumberOfEdges(G)
while G ≠ ∅ do
    G, F ← useLightReductions(G, F)
    if E ≤ α · getNumberOfEdges(G) then
        G, F ← useHeavyReductions(G, F);
        E ← getNumberOfEdges(G)
    end if
    if G ≠ ∅ then
        V ← selectVertex(G)
        G ← remove(G, V)
        F ← F ∪ V
    end if
end while
return F
```

4.4 Stage II

After several vertex selections, some of them may become redundant in the solution, i.e., they are not part of any cycle, so that they can be excluded from the feedback vertex set. To maximize the number of excluded vertices, I take them in the reversed order of their insertion and introduce them in the acyclic graph. At a fixed vertex, I check if the resulting graph is acyclic or not using an optimized version of the Tarjan algorithm for computing strongly connected components. If the graph is acyclic, the vertex will be erased from the solution, and the graph will keep the changes. This step is presented in [15] and [10].

Algorithm 2 Stage II

Require: G - a directed graph, F - a $DFVS$ of G, A - the acyclic remainder of $G \setminus F$

```
for all v ∈ F do
    if isAcyclic(A ∪ {v}) then
        A ← A ∪ {v}
        F ← F \ {v}
    end if
end for
return F
```

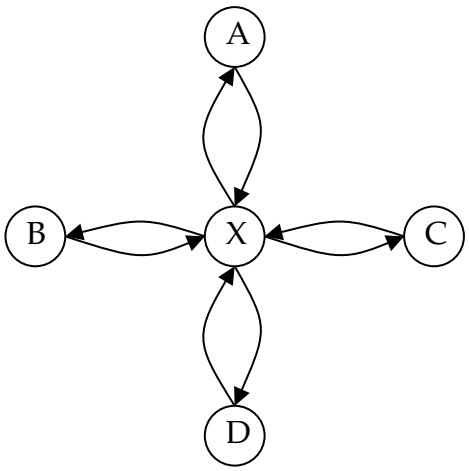


Fig. 8. Example of a redundant vertex

Suppose that nodes A, B, C, D have two more neighbors just like X, so that they all have only three neighbors and $d_G^{\pm}(A) = d_G^{\pm}(B) = d_G^{\pm}(C) = d_G^{\pm}(D) = 3$ and $d_G^{\pm}(X) = 4$. Also, due to the graph's structure, all five nodes are introduced in the solution set within the heuristic choice. In this case, node X (chosen before the others) can be re-

moved from the solution because all its neighbors are in the solution, so it has no way to be part of any cycle.

4.5 Stage III

In the last part of the algorithm, several local searches are performed [15]. A local search runs the same process as the one presented in stage one but on a subgraph of G. Subgraphs are obtained by removing a specific subset uniformly random from the best feedback vertex set found so far, together with a particular subgroup from the acyclic remainder. Additionally, a slightly modified version of the algorithm from stage two is applied to the new solution depending on the remaining time. Although the results of this local search are outstanding and often reach the optimum, finding the optimal solution is not always guaranteed.

Algorithm 3 Stage III

Require: G - a directed graph, F - a $DFVS$ of G, $\alpha \in (0, 1]$

```
while availableTime do
    F' ← getRandomSubset(F);
    G' ← G \ F';
    S ← StageI(G', α)
    S ← StageII(G', S, G' \ S)
    S ← S ∪ F'
    if |S| < |F| then
        F ← S
    end if
end while
return F
```

Chapter 5

Implementation Details

First of all, I mention that I used $C++$ because it is a fast language, $C++$ runtime is more predictable since it has fewer hidden costs, the code is shorter compared to an analog one in $Java$, it uses less RAM, and it has an extensive library called the Standard Template Library, which is a collection of $C++$ templates to provide common programming data structures and functions. During the development of the program and due to the low limit of the stack's memory limit and the relatively low time limit, I minimized as much as possible the number of variables and data structures allocated locally, declaring them globally.

To keep the actual graph in memory, I used adjacent lists implemented as a vector of unordered sets. This is because the adjacent lists of nodes may support changes (delete or insert), allowing operations to reduce or remove nodes based on the heuristic selection. A node v holds three unordered sets with the elements of $N_G^+(v)$, $N_G^-(v)$ and $N_G^\pm(v)$. An unordered set with all the edges of the graph is also kept to check for specific edges. This type of set is implemented using hast tables and has average search, delete and insert time $\mathcal{O}(1)$, making it the ideal data structure for this problem.

Using this data structure, I can identify in $O(1)$ nodes that the first two types of operations can reduce. Each time I delete or add an edge, I check for both incident nodes to see if they can be reduced. If so, I will put them in a stack specific to the reduction criterion that may apply. Thus, I will have a stack for vertices that have self-loop, a stack for vertices that have the external or internal degree equal to zero, and another for nodes with the internal or external degree equal to one. I will use only the nodes in the specific stack when I want to reduce the graph by a particular operation. For the other reduction criteria, it is necessary to go through the entire list of nodes or

edges as the case may be.

Another essential detail to mention is that for the reason of the low memory limit for the stack and the large instances where the number of edges exceeds five million and the number of nodes exceeds 30.000, I had to implement some iterative functionalities, although in the classic format they are recursively coded. Here I refer mainly to the iterative implementation of Tarjan's algorithm for determining strongly connected components. This implementation was used in reduction four and as part of the checker program.

5.1 Complexity Analysis

Both selection criteria are simple but not always optimal, adding significant benefits. Implying only the number of adjacent neighbors allows some reductions to be performed efficiently using hash sets, keeping most of the data in memory. After each update of the graph, reductions one and two can be applied in $\mathcal{O}(1)$. Reduction three is implemented in $\mathcal{O}((|V| + |E|) \cdot \log(|V|))$ and reduction four in $\mathcal{O}(|V| + |E|)$. The rest are run only for vertices with degrees bounded by a small constant, and I assume the complexity is $\mathcal{O}(|V| + |E|)$. The first two reductions are run after each graph change. The others are run after each loss of around $5\% - 25\%$ of the number of edges. For every iteration among T ones, I choose to restore around 30% of the vertices into the acyclic remainder and run the local search. As an instance can be reduced in linear time to a graph with $|V| < |E|$, the final complexity of the program is $\mathcal{O}(T \cdot (|V| + |E| \cdot \log |E|))$, based on Master's Theorem [22], where T is the number of local searches. T is a parameter bounded by the time limit.

Chapter 6

Unsuccessful Optimizations

One idea I tested was to change the output of the solver. Instead of finding a $DFVS$ as small as possible, it aimed to find a DAG as large as possible. The main difference is that within the heuristic selection, the least promising node is chosen now. However, on tests with a large number of nodes and with a vast number of edges, the number of edges was growing too much after every merge of the heuristically selected node, and, on some tests, the memory limit for the stack was exceeded.

In order to optimize the source code, to make it run faster or consume less memory, I tried different experiments. The main focus was on the data structures used. Therefore I tried to find faster variants to replace the unordered sets.

I built several custom hash tables to keep lists of adjacency. Following the change, however, the running time was not stable as I could not find a large prime number that would form the buckets as uniform as possible for all local instances. The complexity on the insert is $\mathcal{O}(1)$, and in search and delete it depends on the size of the bucket, but the average is still $\mathcal{O}(1)$.

In practice, this hash is represented as an array with the size of the maximum number of numbers that can simultaneously be present in the hash. When I want to insert an item, priority is given to the free positions where a number has been deleted over the unused ones. Free positions are held in a stack. Each item also carries a pointer towards the next item from the same bucket. When I want to search, I go through the element by element with those pointers. A delete operation implies a search in which I modify the element predecessor's pointer and put the element's position in the stack.

Chapter 7

Conclusion and Future Work

Following this process, I developed an efficient heuristic solver that finds a Directed Feedback Vertex Set Problem. The solver is public and free to use. Below is the ranking of the contest after more than two months since submissions became available. This ranking is built on the public instances. The final version of the scoreboard includes the results obtained on all the 200 tests, and it is expected to be published at the end of July 2022.

I hope that through this solver, I can open a way to new heuristics with even better performance. For example, I propose a hybrid between the algorithm described and a genetic algorithm. Another idea is to assign each node a specific probability of being inserted or not into the solution within the heuristic selection based on a previously trained neural network.

#	USER	LANGUAGE	TIME [S]	SCORE	1	2	3	4	5	6	7	8
1	_UAIC_AndreiArhire_	C++	57,451.95	**460.14**	46.00	38.00	235.00	**77.00**	33.00	**74.00**	**169.00**	**71.00**
2	florian	Static binary	55,404.43	**459.94**	46.00	38.00	235.00	80.00	33.00	76.00	171.00	74.00
3	Nanored	TGZ	48,661.08	**459.39**	46.00	38.00	**235.00**	85.00	33.00	76.00	175.00	73.00
4	xuxu9110	C++	28,122.86	**458.39**	48.00	38.00	236.00	83.00	35.00	78.00	177.00	75.00
5	dcastro	CMake	59,370.08	**458.11**	46.00	38.00	**235.00**	80.00	33.00	**74.00**	174.00	**71.00**
6	ths_h	Static binary	50,692.55	**454.91**	53.00	43.00	238.00	92.00	39.00	85.00	206.00	89.00
7	GBathie	TGZ	59,603.90	**453.04**	50.00	42.00	247.00	87.00	36.00	83.00	191.00	79.00
8	mt-doom	Static binary	1,400.01	**452.81**	57.00	44.00	247.00	101.00	39.00	88.00	232.00	80.00
9	Timeroot	TGZ	34,772.98	**447.81**	**46.00**	38.00	313.00	83.00	**33.00**	75.00	179.00	73.00
10	PACE2022	C++	6,606.34	**440.57**	63.00	**38.00**	345.00	107.00	37.00	81.00	230.00	79.00
11	GraPA-JAVA	Jar	30,522.70	**438.91**	57.00	41.00	246.00	91.00	38.00	88.00	219.00	83.00
12	BabyShark	Jar	5,015.25	**438.86**	56.00	43.00	244.00	97.00	39.00	87.00	224.00	78.00
13	GRAPA-Rust	Static binary	36,650.77	**433.24**	162.00	126.00	352.00	301.00	101.00	160.00	441.00	217.00
14	anthonimes	Python3	34,630.25	**424.28**	75.00	62.00	377.00	156.00	47.00	113.00	303.00	116.00
15	GOAT	Static binary	6,807.80	**418.02**	226.00	324.00	481.00	273.00	185.00	310.00	420.00	351.00
16	pasa_2	Python3	15,196.27	**415.22**	77.00	73.00	361.00	143.00	80.00	125.00	304.00	123.00
17	check	C++	51,791.50	**405.98**	47.00	39.00	237.00	83.00	34.00	77.00	179.00	74.00
18	prostoyakira	C++	48,707.25	**402.13**	71.00	49.00	278.00	216.00	40.00	97.00	334.00	200.00
19	Juve45	C++	15.83	**389.07**	599.00	626.00	634.00	1,163.00	694.00	720.00	1,344.00	1,237.00
20	jdohea	Python3	57,173.29	**365.37**	WA	WA	WA	WA	WA	WA	WA	WA
21	pace2022	C++	0.00	**365.20**	TLE	TLE	TLE	TLE	TLE	TLE	TLE	TLE
22	swats	TGZ	51,751.27	**365.20**	RTE	RTE	RTE	RTE	RTE	RTE	RTE	RTE
23	pasa_1	Python3	0.00	**365.20**	RTE	RTE	RTE	RTE	RTE	RTE	RTE	RTE
24	convo	C++	96.66	**365.20**	RTE	RTE	RTE	RTE	RTE	RTE	RTE	RTE

Figure 7.1: Scoreboard

Bibliography

[1] PACE Challenge about. https://pacechallenge.org/about/. Accessed: 2022-06-12.

[2] PACE Solver paper. https://andrei-arhire.web.app/assets/PACE2022.pdf. Accessed: 2022-06-12.

[3] Andrei Arhire. Pace solver, 2022. URL: https://github.com/AndreiiArhire/PACE2022.

[4] Andrei Arhire and Paul Diac. _UAIC_ANDREIARHIRE_ - A Heuristic Solver for the Directed Feedback Vertex Set Problem, June 2022. doi:10.5281/zenodo.6646187.

[5] Max Bannach, Sebastian Berndt, and Thorsten Ehlers. Jdrasil: A Modular Library for Computing Tree Decompositions. In Costas S. Iliopoulos, Solon P. Pissis, Simon J. Puglisi, and Rajeev Raman, editors, *16th International Symposium on Experimental Algorithms (SEA 2017)*, volume 75 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 28:1–28:21, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. URL: http://drops.dagstuhl.de/opus/volltexte/2017/7605, doi:10.4230/LIPIcs.SEA.2017.28.

[6] Atsushi Mochizuki... Bernold Fiedler. Dynamics and control at feedback vertex sets. i: Informative and determining nodes in regulatory networks. *Journal of Dynamics and Differential Equations*, 25:563–604, 2013. URL: http://doi.org/10.1007/s10884-013-9312-7, doi:10.1007/s10884-013-9312-7.

[7] Vincenzo Cutello and Francesco Pappalardo. Targeting the minimum vertex set problem with an enhanced genetic algorithm improved with local search strategies. In *International Conference on Intelligent Computing*, pages 177–188. Springer, 2015.

[8] Mochizuki Atsushi; Fiedler Bernold; Kurosawa Gen; Saito Daisuke. Dynamics and control at feedback vertex sets. ii: A faithful monitor to determine the diversity of molecular activities in regulatory networks. *Journal of Theoretical Biology*, 335:130–146, 2013. URL: http://doi.org/10.1016/j.jtbi.2013.06.009, doi:10.1016/j.jtbi.2013.06.009.

[9] Serge Gaspers, Joachim Gudmundsson, Mitchell Jones, Julián Mestre, and Stefan Rümmele. Turbocharging Treewidth Heuristics. In Jiong Guo and Danny Hermelin, editors, *11th International Symposium on Parameterized and Exact Computation (IPEC 2016)*, volume 63 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 13:1–13:13, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. URL: http://drops.dagstuhl.de/opus/volltexte/2017/6932, doi:10.4230/LIPIcs.IPEC.2016.13.

[10] Berend Hasselman. An efficient method for detecting redundant feedback vertices. CPB Discussion Paper 29, CPB Netherlands Bureau for Economic Policy Analysis, April 2004. URL: https://ideas.repec.org/p/cpb/discus/29.html.

[11] Chen Jianer; Liu Yang; Lu Songjian; O'sullivan Barry; Razgon Igor. A fixed-parameter algorithm for the directed feedback vertex set problem. *Journal of the ACM*, 55:1–19, 2008. URL: http://doi.org/10.1145/1411509.1411511, doi:10.1145/1411509.1411511.

[12] Hen-Ming Lin; Jing-Yang Jou. On computing the minimum feedback vertex set of a directed graph by contraction operations. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 19:295–307, 2000. URL: http://doi.org/10.1109/43.833199, doi:10.1109/43.833199.

[13] Richard M. Karp. *Reducibility among Combinatorial Problems*, pages 85–103. Springer US, Boston, MA, 1972. doi:10.1007/978-1-4684-2001-2_9.

[14] Koichi Kobayashi. Design of fixed points in boolean networks using feedback vertex sets and model reduction. *Complexity*, 2019:1–9, 2019. URL: http://doi.org/10.1155/2019%2F9261793, doi:10.1155/2019/9261793.

[15] Mile Lemaic. *Markov-Chain-Based Heuristics for the Feedback Vertex Set Problem for Digraphs*. PhD thesis, Universität zu Köln, 2008.

[16] Hanoch Levy; David W Low. A contraction algorithm for finding small cycle cutsets. *Journal of Algorithms*, 9:470–493, 1988. URL: http://doi.org/10.1016/0196-6774%2888%2990013-2, doi:10.1016/0196-6774(88)90013-2.

[17] Bar-Yehuda Reuven; Geiger Dan; Naor Joseph (Seffi); Roth Ron M. Approximation algorithms for the feedback vertex set problem with applications to constraint satisfaction and bayesian inference. *SIAM Journal on Computing*, 27:942–959, 1998. URL: http://doi.org/10.1137/s0097539796305109, doi:10.1137/s0097539796305109.

[18] Tang Zhipeng; Feng Qilong; Zhong Ping. Nonuniform neighborhood sampling based simulated annealing for the directed feedback vertex set problem. *IEEE Access*, 5:12353–12363, 2017. URL: http://doi.org/10.1109/ACCESS.2017.2724065, doi:10.1109/ACCESS.2017.2724065.

[19] Bergougnoux Benjamin; Eiben Eduard; Ganian Robert; Ordyniak Sebastian; Ramanujan M. S. Towards a polynomial kernel for directed feedback vertex set. *Algorithmica*, 2020. URL: http://doi.org/10.1007/s00453-020-00777-5, doi:10.1007/s00453-020-00777-5.

[20] Ben Strasser. Computing tree decompositions with flowcutter: PACE 2017 submission. *CoRR*, abs/1709.08949, 2017. URL: http://arxiv.org/abs/1709.08949, arXiv:1709.08949.

[21] Hisao Tamaki. Positive-Instance Driven Dynamic Programming for Treewidth. In Kirk Pruhs and Christian Sohler, editors, *25th Annual European Symposium on Algorithms (ESA 2017)*, volume 87 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 68:1–68:13, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. URL: http://drops.dagstuhl.de/opus/volltexte/2017/7880, doi:10.4230/LIPIcs.ESA.2017.68.

[22] R.M. Verma. A general method and a master theorem for divide-and-conquer recurrences with applications. *Journal of Algorithms*, 16:67–79, 1994. URL: http://doi.org/10.1006/jagm.1994.1004, doi:10.1006/jagm.1994.1004.