Boosting Data Reduction for the Maximum Weight Independent Set Problem Using Increasing Transformations

Alexander Gellner, Sebastian Lamm, Christian Schulz,
Darren Strash, Bogdán Zaválnij

Article summary

Chipăruș Alexandru-Denis - MOC 2

January 2022

1   Introduction

In this article, the authors introduce new improvements to the de facto standard approach to solve the Maximum Weight Independent Set (MWIS) problem, which is the branch-and-reduce algorithm. New generalized data reduction and transformation rules for the framework of this algorithm are presented and tested on well-known problem instances, giving promising results.

2   Problem definition

Given a graph $G = (V, E)$ and a weight function $w : V \to \mathbb{R}^+$, the goal of the maximum weight independent set problem is to compute a set of pair-wise non-adjacent vertices $I \subseteq V$, whose total weight is maximum. This NP-hard problem has several real-world applications in various domains such as map labeling, combinatorial auctions, workload scheduling, computer vision, protein structure prediction and many more.

3   Related work

In practice, for graphs with hundreds of vertices it is enough to use a traditional branch-and-bound method. However, larger graphs are infeasible for this method, that's why other techniques were developed to obtain good solutions on larger graphs:

1. Exact methods: This type of methods are dominated by branch-and-bound with special pruning rules so that the explored space it's reduced significantly. Also, the most promising one is the branch-and-reduce algorithm for MWIS, presented by Lamm et al. [2].

2. Heuristic methods: The most used heuristic approach is the Local Search (or variants of this method), which usually computes an initial solution and then tries to improve it by simple operations (insertions, removals or swaps).

4 Structions

4.1 Original struction

Originally, the struction (STability number RedUCTION) was introduced by Ebenegger et al. [3] and was later improved by Alexe et al. [4]. This method is a graph transformation that can be applied to an arbitrary vertex and reduces the stability number by exactly one. Thus, after successive applications of the struction, the stability number of the graph can be determined.

The first struction described in the article, is the one introduces by Ebenegger et al. , on which they base their own struction variants.

In general, a struction is applied on a center vertex v, which has the neighborhood $N(v)$. The struction removes the vertex v from the graph G, producing a new graph G', and reduces the weighted independence number of the graph G by its weight, i.e. $\alpha_w(G) = \alpha_w(G') + w(v)$.

Layering is a method that partitions a set M that contains vertices $v_{x,y}$. These vertices are indexed by two parameters $x \in X, y \in Y$, where the sets X and Y either contain vertices or vertex sets. A layer L_k is defined as the set of all vertices having k as first parameter ($k \in X$). Conversely, the layer of a vertex $v_{x,y}$ is $L(v_{x,y}) = k$.

4.1.1 Formal definition

Let $v \in V$ be a vertex with minimum weight $w(v)$ among its neighbors. The graph it's transformed as follows:

1. Remove v and lower the weight of each neighbor by $w(v)$;

2. For each pair of non-adjacent neighbors $x < y$, create a new vertex $v_{x,y}$ with weight $w(v_{x,y}) = w(v)$;

3. Insert edges between $v_{q,x}$ and $v_{r,y}$ if either x and y are adjacent or $L_q \neq L_r$;

4. Each vertex $v_{x,y}$ is also connected to vertex $w \in V \setminus \{v\}$ adjacent to either x or y.

After obtaining an indepenent set I' of graph G', the original independent set I (of graph G) can be reconstructed using $I = I' \cup \{v\}$, if $I' \cap N(v) = \emptyset$, or $I = I' \cap V$, otherwise.

(a) Original Graph (b) Original Struction (c) Modified Struction
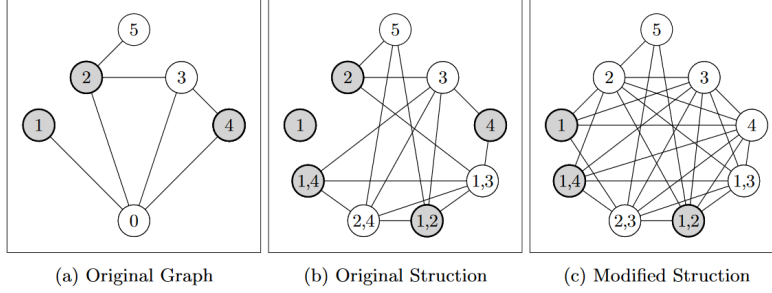
4.2 New structions

4.2.1 Modified Weighted Struction

In the original struction the number of vertices that are part of an MWIS in the transformed graph is generally larger. The modified struction tries to alleviate this issue by ensuring that the number of vertices of an MWIS stays the same in both graphs.

When using this struction, the newly created vertex for each pair of non-adjacent neighbors $x, y \in N(v)$ with $x < y$ is now assigned weight $w(v_{x,y}) = w(y)$. Furthermore, in addition to the edges created in the original struction, each neighbor $k \in N(v)$ is connected to each vertex $v_{x,y}$ belonging to a different layer than k and $N(v)$ is extended to a clique by adding edges between any vertices $x, y \in N(v)$.

To get the original I from the resulted I', we use $I = I' \cup \{v\}$, if $I' \cap N(v) = \emptyset$, or $I = (I' \cap V) \cup \{v_y | v_{x,y} \in I' \setminus V\}$, otherwise.

4.2.2 Extended Weighted Struction

The second new struction is defined as follows: let $v \in V$ be an arbitrary vertex and C the set of all independent sets c in $G[N(v)]$ (the subgraph induced by $N(v)$) with $w(c) > w(v)$. The transformed graph G' is derived using the following steps:

1. Remove v together with its neighbors;

2. Create a new vertex v_c with weight $w(v_c) = w(c) - w(v)$ for each independent set $c \in C$;

3. Each vertex v_c is connected to each non-neighbor (of v) w adjacent to at least one vertex in c;

4. The vertices v_c are connected with each other to form a clique.

For an MWIS I' (of G') we obtain I (of G) as follows: if $I' \setminus V = \{v_c\}$, then $I = (I' \cap V) \cup c$, otherwise $I = I' \cup \{v\}$.

3

4.2.3 Extended Reduced Weighted Struction

The last new struction presented in the article is the extended reduced weighted struction. This struction is a variant of the previous one which can potentially reduce the number of newly created vertices.

A subset of C is defined as follows: $C' = \{c \in C | w(c) - w(M(c)) \leq w(v)\}$, where $M(c)$ is the vertex from $N(v)$ with the highest index in an arbitrary (but fixed) ordering. Then, the same construction as for the extended struction is used, but only create vertices for the subset C'. The resulting set is denoted by V_C.

Since this construction might not be valid anymore, additional vertices, that are connected to each other with layering, are added. For each pair of an independent set $c \in C'$ and a vertex $y \in N(v)$, a vertex $v_{c,y}$ with weight $w(v_{c,y}) = w(y)$ is created, if y is not adjacent to any vertex in c. The resulting set is denoted by V_E. Then new edges are inserted between two vertices $v_{c,y}, v_{c',y'}$ if they either belong to different layers or y and y' are adjacent. Moreover, each vertex $v_{c,y}$ is connected to each non-neighbour w (of v), if w has been connected to either y or a vertex $x \in c$. Also, each vertex v_c to each vertex $v_{c',y}$ belonging to a different layer than c.

For an MWIS I' the authors obtain an MWIS I as follows: if $I' \cap V_C = \emptyset$, then $I = I' \cup \{v\}$, otherwise, $I = (I' \cap V) \cup c \cup \{v_y | v_{c,y} \in I' \cap V_E\}$.

5 The new branch-and-transform

The structions described in the previous section are used in three different forms, called transformations:

1. Decreasing Transformations : transformations where graph G' has less vertices than G;

2. Plateau Transformations : transformations where graph G' has the same number of vertices as G (but the size and weight of the MWIS are reduced);

3. Increasing Transformations : transformations where graph G' has more vertices than G.

5.1 Non-Increasing Reduction Algorithm

The authors use the standard branch-and-reduce framework and apply transformations instead of simple reductions. When applying a struction variant, in the form of a transformation, the new method generally keep track of the number of vertices that will be created. If this number exceeds a given maximum value n_{max}, the corresponding struction is discarded to ensure that not too many vertices are created.

.

Algorithm 1 Branch-and-Reduce Algorithm for MWIS
___
input graph $G = (V, E)$, current solution weight $curr$
(initially zero), best solution weight $\mathcal{W}$ (initially zero)
procedure Solve(G, $curr$, $\mathcal{W}$)
    $(G, curr) \leftarrow$ Reduce($G, curr$)
    if $\mathcal{W} = 0$ **then** $\mathcal{W} \leftarrow curr + $ ILS(G)
    if $curr + $ UpperBound(G) $\leq \mathcal{W}$ **then return** $\mathcal{W}$
    if G is empty **then return** $\max\{\mathcal{W}, curr\}$
    if G is not connected **then**
        for all $G_i \in$ Components(G) **do**
            $curr \leftarrow curr + $ Solve(G_i, 0, 0)
        return $\max(\mathcal{W}, curr)$
    $(G_1, curr_1), (G_2, curr_2) \leftarrow$ Branch($G, curr$)
    {Run 1st case, update currently best solution}
    $\mathcal{W} \leftarrow$ Solve($G_1, curr_1, \mathcal{W}$)
    {Use updated $\mathcal{W}$ to shrink the search space}
    $\mathcal{W} \leftarrow$ Solve($G_2, curr_2, \mathcal{W}$)
return $\mathcal{W}$
___

Algorithm 2 Cyclic Blow-Up Algorithm
___
input graph $G = (V, E)$, unsuccessful iteration
threshold $X \in [1, \infty)$, maximum blowup $\alpha \in [1, \infty)$
procedure CyclicBlowUp(G, X)
    $K \leftarrow$ Reduce(G)
    $K^\star \leftarrow K$
    count $\leftarrow 0$
    while $|V(K)| \leq \alpha \cdot |V(K^\star)|$ **and** count $< X$ **do**
        $K' \leftarrow$ BlowUp(K)
        if $K' = K$ **then**
            return $K^\star$
        $K'' \leftarrow$ Reduce(K')
        $K \leftarrow$ Accept(K'', K)
        if $K < K^\star$ **then**
            $K^\star \leftarrow K$
return $K^\star$
___

5.2 Cyclic Blow-Up Algorithm

Another method used in the article is an extension of the previous one. The main idea is to alternate between computing an irreducible graph using the non-increasing algorithm (the Reduce Phase) and then apply increasing transformations (structions) while ensuring that the graph size does not increase too much (the Blow-Up Phase).
.

6 Experimental results

The experimental configuration included the standard branch-and-reduce algorithm, where the 2 new preprocessing algorithms are applied. Cyclic Blow-Up Algorithm is used in two forms: C_{strong}, with max number of unsuccessful

Graph	n	t_r	n	t_r	n	t_r	n	t_r	n	t_r
OSM instances	BASIC-DENSE		BASIC-SPARSE		NONINCREASING		CYCLIC-FAST		CYCLIC-STRONG	
alabama-AM2	173	0.06	173	0.07	0	0.01	0	0.01	0	0.01
district-of-columbia-AM2	6360	11.86	6360	14.39	5606	0.85	1855	2.51	1484	84.91
florida-AM3	1069	31.52	1069	35.20	814	0.13	661	0.44	267	42.26
georgia-AM3	861	8.99	861	10.14	796	0.08	587	0.69	425	12.84
greenland-AM3	3942	3.81	3942	24.77	3953	3.94	3339	10.27	3339	54.44
new-hampshire-AM3	247	4.99	247	5.69	164	0.02	0	0.07	0	0.09
rhode-island-AM2	1103	0.55	1103	0.68	845	0.17	0	0.53	0	4.57
utah-AM3	568	8.21	568	8.97	396	0.03	0	0.09	0	0.40
Empty graphs	0% (0/34)		0% (0/34)		11.8% (4/34)		41.2% (14/34)		50% (17/34)	
SNAP instances	BASIC-DENSE		BASIC-SPARSE		NONINCREASING		CYCLIC-FAST		CYCLIC-STRONG	
as-skitter	26584	25.82	8585	36.69	3426	4.75	2782	5.50	2343	6.80
ca-AstroPh	0	0.02	0	0.02	0	0.02	0	0.03	0	0.03
email-EuAll	0	0.08	0	0.09	0	0.06	0	0.09	0	0.07
p2p-Gnutella06	0	0.01	0	0.01	0	0.01	0	0.01	0	0.01
roadNet-PA	133814	2.43	35442	7.73	300	1.05	0	1.19	0	1.14
soc-LiveJournal1	60041	236.88	29508	213.74	4319	22.27	3530	24.13	1314	37.77
web-Google	2810	1.57	1254	2.42	361	1.75	46	1.88	46	7.97
wiki-Vote	477	0.03	0	0.02	0	0.02	0	0.02	0	0.02
Empty graphs	58.1% (18/31)		67.7% (21/31)		67.7% (21/34)		80.6% (25/31)		80.6% (25/31)	
mesh instances	BASIC-DENSE		BASIC-SPARSE		NONINCREASING		CYCLIC-FAST		CYCLIC-STRONG	
buddha	380315	5.56	107265	26.19	86	1.83	0	1.87	0	1.91
dragon	51885	0.89	12893	1.34	0	0.18	0	0.19	0	0.21
ecat	239787	4.07	26270	10.09	274	2.12	0	2.12	0	2.14
Empty graphs	0% (0/15)		0% (0/15)		66.7% (10/15)		100% (15/15)		100% (15/15)	
FE instances	BASIC-DENSE		BASIC-SPARSE		NONINCREASING		CYCLIC-FAST		CYCLIC-STRONG	
fe_ocean	141283	1.05	0	5.94	138338	8.90	138134	9.61	138049	10.78
fe_sphere	15269	0.21	15269	1.47	2961	0.34	147	0.62	0	0.75
Empty graphs	0% (0/7)		14.3% (1/7)		0% (0/7)		28.6% (2/7)		42.9% (3/7)	

Table 1: Smallest irreducible graph found by each algorithm and time (in seconds) required to compute it. Rows are highlighted in gray if one of our algorithms is able to obtain an empty graph.
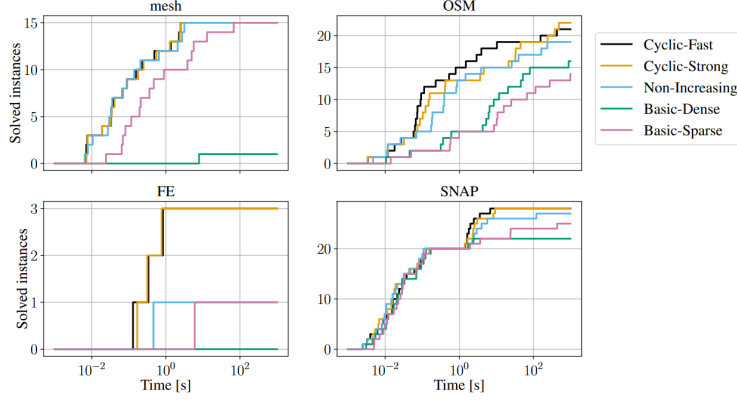

Figure 3: Cactus plots for the different instance families and evaluated solvers.

blow-up phases $X = 64$ and $n_{max} = 2048$, and C_{fast}, with $X = 25$ and $n_{max} = 512$. Also, instead of using ILS, the authors use hybrid iterated local search (HILS).

.

.

.

7 Conclusion

The experimental results show that the new methods presented in this article outperformes existing algorithms on a wide variety of instances. The authors suggest that in the future, more work can be done for the transformations used to have a much better version of the branch-and-transform.

Graph	t_{max}	w_{max}	t_{max}	w_{max}	t_{max}	w_{max}	t_{max}	w_{max}
OSM instances	DynWVC2		HILS		Cyclic-Fast		Cyclic-Strong	
alabama-AM2	0.24	174 269	0.03	**174 309**	0.01	**174 309**	0.01	**174 309**
district-of-columbia-AM2	915.18	208 977	400.69	**209 132**	4.21	**209 132**	84.21	209 131
florida-AM3	862.04	237 120	3.98	**237 333**	1.57	**237 333**	40.97	**237 333**
georgia-AM3	1.31	**222 652**	0.04	**222 652**	0.98	**222 652**	12.97	**222 652**
greenland-AM3	640.46	14 010	1.18	**14 011**	10.95	**14 011**	58.24	14 008
new-hampshire-AM3	1.63	**116 060**	0.03	**116 060**	0.05	**116 060**	0.08	**116 060**
rhode-island-AM2	13.90	184 576	0.24	**184 596**	0.41	**184 596**	4.37	**184 596**
utah-AM3	136.90	**98 847**	0.07	**98 847**	0.09	**98 847**	0.27	**98 847**
Solved instances					61.8% (21/34)		64.7% (22/34)	
Optimal weight	68.2% (15/22)		100.0% (22/22)					
SNAP instances	DynWVC2		HILS		Cyclic-Fast		Cyclic-Strong	
as-skitter	383.97	123 273 938	999.32	122 658 804	346.69	124 137 148	354.71	**124 137 365**
ca-AstroPh	125.05	797 480	13.47	**797 510**	0.02	**797 510**	0.02	**797 510**
email-EuAll	132.62	**25 286 322**	338.14	**25 286 322**	0.07	**25 286 322**	0.07	**25 286 322**
p2p-Gnutella06	186.97	548 611	1.29	**548 612**	0.01	**548 612**	0.01	**548 612**
roadNet-PA	469.18	60 990 177	999.94	60 037 011	0.96	**61 731 589**	1.04	**61 731 589**
soc-LiveJournal1	999.99	279 231 875	1 000.00	255 079 926	51.33	284 036 222	44.19	**284 036 239**
web-Google	324.65	56 206 250	995.92	56 008 278	1.72	**56 326 504**	6.44	**56 326 504**
wiki-Vote	0.32	**500 079**	10.34	**500 079**	0.02	**500 079**	0.02	**500 079**
Solved instances					90.3% (28/31)		90.3% (28/31)	
Optimal weight	28.6% (8/28)		57.1% (16/28)					
mesh instances	DynWVC2		HILS		Cyclic-Fast		Cyclic-Strong	
buddha	797.35	56 757 052	999.94	55 490 134	1.75	**57 555 880**	1.77	**57 555 880**
dragon	981.51	7 944 042	996.01	7 940 422	0.21	**7 956 530**	0.22	**7 956 530**
ecat	542.87	36 129 804	999.91	35 512 644	2.19	**36 650 298**	2.29	**36 650 298**
Solved instances					100.0% (15/15)		100.0% (15/15)	
Optimal weight	0.0% (0/15)		0.0% (0/15)					
FE instances	DynWVC1		HILS		Cyclic-Fast		Cyclic-Strong	
fe_ocean	983.53	**7 222 521**	999.57	7 069 279	18.85	6 591 832	19.04	6 591 537
fe_sphere	875.87	**616 978**	843.67	616 528	0.63	**617 816**	0.67	**617 816**
Solved instances					42.9% (3/7)		42.9% (3/7)	
Optimal weight	0.0% (0/3)		0.0% (0/3)					

Table 2: Best solution found by each algorithm and time (in seconds) required to compute it. The global best solution is highlighted in bold. Rows are highlighted in gray if one of our exact solvers is able to solve the corresponding instances.

References

[1] Gellner, Alexander, et al. "Boosting Data Reduction for the Maximum Weight Independent Set Problem Using Increasing Transformations." 2021 Proceedings of the Workshop on Algorithm Engineering and Experiments (ALENEX). Society for Industrial and Applied Mathematics, 2021.

[2] S. Lamm, C. Schulz, D. Strash, R. Williger, and H. Zhang. Exactly solving the maximum weight independent set problem on large real-world graphs. In 2019 Proceedings of the Meeting on Algorithm Engineering and Experiments (ALENEX), pages 144–158. SIAM, 2019.

[3] C. Ebenegger, P. Hammer, and D. De Werra. Pseudo-boolean functions and stability of graphs. In North-Holland Mathematics Studies, volume 95,pages 83–97. Elsevier, 1984.

[4] G. Alexe, P. L. Hammer, V. V. Lozin, and D. de Werra. Struction revisited. Discrete applied mathematics, 132(1-3):27–46, 2003.