Round eliminator: a tool for automatic speedup simulation

Dennis Olivetti

1 Introduction

This tool is an implementation of the main result of [1], that states that, under some assumptions, given a problem Π having a distributed complexity of exactly T rounds, it is possible to automatically compute a new problem Π' that has a distributed complexity of exactly $T - 1$ rounds.

This document gives a high level overview of how (and when) it is possible to use this tool to obtain, in a semi-automatic way, lower and upper bounds on the complexity of distributed problems. We will first see, in Section 2, what are the assumptions under which the result of [1] can be applied, which kind of problems can be fed into the program, and how to properly encode them. Then, in Section 3 we will see some basic examples of how to use the program. In Sections 4 and 5 we will see how to use this tool to get proper lower and upper bounds. Finally, in Section 6 we will see some additional things left out in the previous sections. Do not be scared by the length of this guide, it is full of pictures!

2 Assumptions

Infinite regular trees   Consider an infinite 2-colored tree, where all white nodes have the same degree Δ, and all black nodes have the same degree δ. This is the most general family where the technique applies, and notice that it also includes the family of Δ-regular trees (without the 2-coloring), by setting $\delta = 2$ and treating black nodes as edges. This implies, for example, that we cannot apply this technique to get a lower bound for a problem that is trivial (0-rounds solvable) on trees, and that, if we use this technique to get an upper bound, the result does not directly apply to general graphs.

Anonymous networks   The theorem of [1] assumes that the network is anonymous and that algorithms are deterministic. That is, if we get a lower bound using this technique, it does not directly apply to the LOCAL model of distributed computing, but to a much weaker setting. Usually this is not an issue: we can lift lower bounds obtained with this technique to lower bounds that apply to the LOCAL model (see e.g. [2]), also for randomized algorithms.

Local Checkability   This is the most important assumption: our problem must be *locally checkable*. That is, there should be a set of constraints that, if they are satisfied on each node, then the solution is globally valid. In order to use the round eliminator on a problem, the only thing that we need to type is the set of local constraints. In particular, a problem Π is described by a triple (L, W, B):

- $L = \{l_1, \ldots, l_{|L|}\}$ is a set of labels.

- W represents the white constraint. That is, W is the set of allowed white configurations. Each configuration is a multiset of size Δ, where each element is in L.

- B represents the black constraint. That is, B is the set of allowed black configurations. Each configuration is a multiset of size δ, where each element is in L.

The assumption is that there is a *white* algorithm that solves the problem. That is, white nodes are active entities, and they must output a label $l \in L$ on each incident edge (they can output different labels on different incident edges). The output is valid if the following holds:
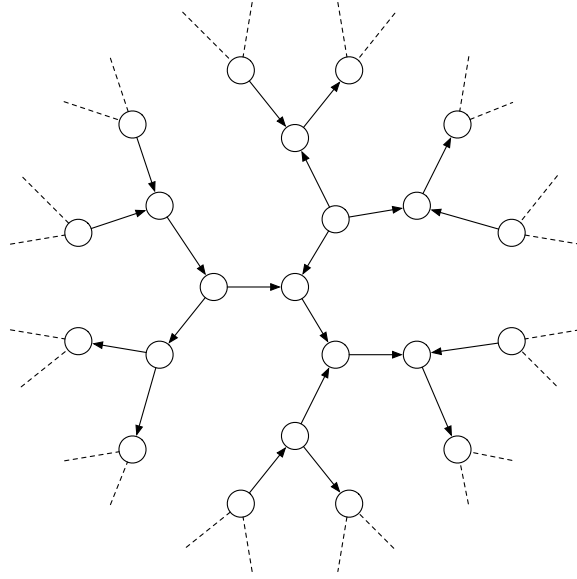
- For each white node, consider the multiset of size Δ containing the labels that appear on its incident edges. This multiset must be in W.

- For each black node, consider the multiset of size δ containing the labels that appear on its incident edges. This multiset must be in B.

Note that black nodes are passive, that is, they do not output anything. We can imagine black nodes as entities that just propagate messages that white nodes send between each other. It is easier to understand this if we consider the case where black nodes have degree 2, that is, when they represent edges. In this case we have white nodes that communicate with each other, and messages pass through edges, that are passive entities.

The idea is that if we apply the technique on a problem that *white* nodes can solve in exactly T rounds, then we get a problem that can be solved in $T - 1$ rounds by *black* nodes, that is, we save one round but we reverse the roles of active and passive. By repeating the same procedure we would get a third problem, that requires $T - 2$ rounds where the active entities are again the *white* nodes.

Inputs   Currently, the constraints that the round eliminator can handle cannot depend on inputs (except for the bipartition side), that is, all white nodes have the same view and the same constraints, and all black nodes have the same view and the same constraints. The round eliminator can actually handle the case where some input coloring is given, and this is used to check for 0-rounds solvability. For more details, see Section 6.
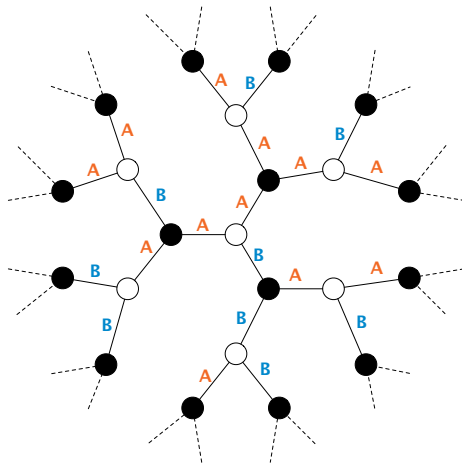
Example: Bipartite Sinkless Orientation (BSO)   Sinkless orientation requires to orient each edge such that there are no sinks. That is, orient each edge such that each node has at least one outgoing edge. A solution looks like the following:

It is easy to see that this problem is locally checkable: each edge must be oriented, and if each node satisfies that it has at least one outgoing edge then the solution is globally valid. We will now see how to encode this problem in the 2-colored case, in particular in the case where $\Delta = \delta = 3$.

We need to define the set of labels L, and the set of white and black constraints, W and B. We can define L as a set containing 2 labels $\mathbf{A}$ and $\mathbf{B}$. The label $\mathbf{A}$ will represent an edge oriented from black to white, and the label $\mathbf{B}$ will represent an edge oriented from white to black. Note that $\mathbf{A}$ and $\mathbf{B}$ are just two arbitrary names.

We now need to define the constraints. White nodes must satisfy to have at least one outgoing edge, that is an edge oriented from white to black, that is an edge labeled $\mathbf{B}$. The other edges are unconstrained, that is, they can be oriented either ways, and thus be labeled either $\mathbf{A}$ or $\mathbf{B}$. Thus, we can define $W = \{\{\mathbf{B}, \mathbf{A}, \mathbf{A}\}, \{\mathbf{B}, \mathbf{A}, \mathbf{B}\}, \{\mathbf{B}, \mathbf{B}, \mathbf{B}\}\}$. Notice that the order in the set, and in each multiset, does not matter: the set W is just the set containing all possible multisets that contain at least one $\mathbf{B}$. Similarly, the black nodes must have at least one incident edge labeled $\mathbf{A}$. Thus, we define $B = \{\{\mathbf{A}, \mathbf{A}, \mathbf{A}\}, \{\mathbf{A}, \mathbf{A}, \mathbf{B}\}, \{\mathbf{A}, \mathbf{B}, \mathbf{B}\}\}$. A correct solution looks like the following:

In order to feed the set of constraints to the round eliminator, we use a more compact notation.

In order to express the problem $\Pi = (L, W, B)$ we will omit L (it can be inferred from W and B), and we will write the set W as follows (each horizontal line is a tuple of W):
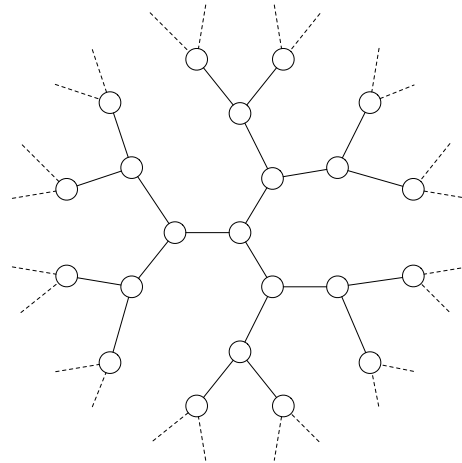
B	A	A
B	A	B
B	B	B

Similarly, we will write the set B as follows:

A	A	A
A	A	B
A	B	B

Also we will use an even more compact notation, that is, the set W will be written as $\boxed{\textbf{B AB AB}}$, while the set B will be written as $\boxed{\textbf{A AB AB}}$. This compact notation should be interpreted as follows: any choice that we make on each group is allowed. For example, from $\boxed{\textbf{B AB AB}}$, we are forced to pick **B** from the first group, but we can pick either **A** or **B** from the second and the third. Also, these two notations can be mixed. For example, B could be written as follows:

A	A	AB
A	B	B

Example: Non-bipartite Sinkless Orientation (SO) We will now see an alternative way to encode the sinkless orientation problem, that may help to understand how to encode problems defined on non-2-colored graphs. We will assume $\Delta = 3$ and $\delta = 2$. White nodes will represent proper nodes, while black nodes will represent edges. In fact, consider a 3 regular tree:
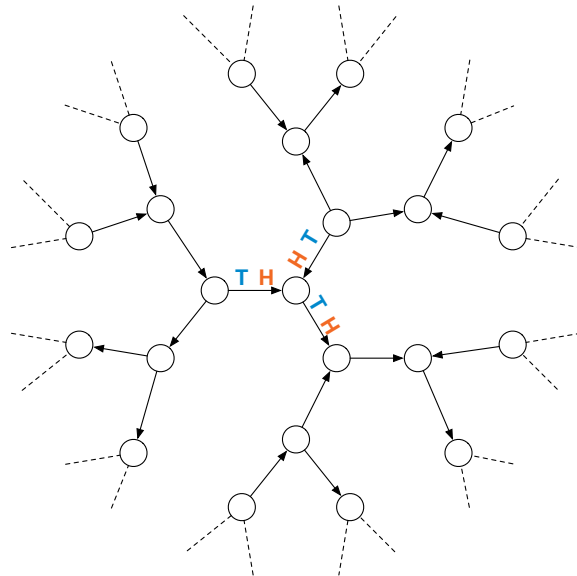

Let us add a black node in the middle of each edge, obtaining the following graph:

It should be clear that given an algorithm that runs on one graph, we can easily simulate it on the other. That is, these two graphs are basically equivalent.

As in the bipartite case, we want to orient edges of the original graph such that no node is a sink. In this case we want no sinks in the original graph (the one before adding the black nodes), but we need to define the constraints on the second graph (the one where we can apply the theorem of [1]). Thus, we can think of white constraints as *node constraints*, and black constraints as *edge constraints*.

We will have two labels, $\mathbf{H}$ and $\mathbf{T}$, that represent, respectively, the head and the tail of an oriented edge. An edge must be properly oriented, that is, $B = \{\{\mathbf{H}, \mathbf{T}\}\}$. In other words, if a white node writes $\mathbf{H}$ on its incident edge, the other white node that is incident to the same edge must write $\mathbf{T}$. In other words, each black node is a neighbor of exactly two white nodes, and it must see different labels. Also, white constraints must satisfy that each white node is incident to at least one $\mathbf{T}$, that is $W = \{\{\mathbf{T}, \mathbf{H}, \mathbf{H}\}, \{\mathbf{T}, \mathbf{H}, \mathbf{T}\}, \{\mathbf{T}, \mathbf{T}, \mathbf{T}\}\}$. A solution looks like the following:

A compact way of writing the constraints is the following. Node constraints are $\boxed{\mathbf{T}\ \mathbf{HT}\ \mathbf{HT}}$, while edge constraints are $\boxed{\mathbf{H}\ \mathbf{T}}$. If we are given a solution for the problem encoded in this way, we

can obtain an output for the sinkless orientation problem by orienting the edges as indicated by **T** and **H**. Also, given an edge orientation, we can easily produce a solution for the problem that we encoded. Thus, the problem that we encoded is equivalent to SO.

Example: Maximal Independent Set (MIS)   While sinkless orientation is a problem that we could either define on 2-colored graphs or on non-2-colored graphs, for MIS it really only makes sense to define it on non-2-colored graphs. In fact, nothing stops us from defining it on 2-colored graphs, but we would obtain a 0-round solvable problem.

   Let us see how to define this problem on 3-regular trees with node and edge constraints, that is, let us assume that $\Delta = 3$ and $\delta = 2$. One may think that we could use 2 labels, one used by nodes that are in the MIS, and the other used by nodes that are not in the MIS. This is actually not possible [2], and the problem is that in order to express *maximality*, we need to somehow ensure that if a node is not in the MIS, then at least one neighbor is in the MIS, and this requires an additional label.

   The idea is that we use 3 labels, **M**, **O** and **P**. Being in the MIS is encoded by having **M** on each incident edge. Nodes that are not in the MIS will need to *point* to one neighbor that is in the MIS, that is, they will output **P** on one edge and **O** on all the others. That is, W is defined as follows:

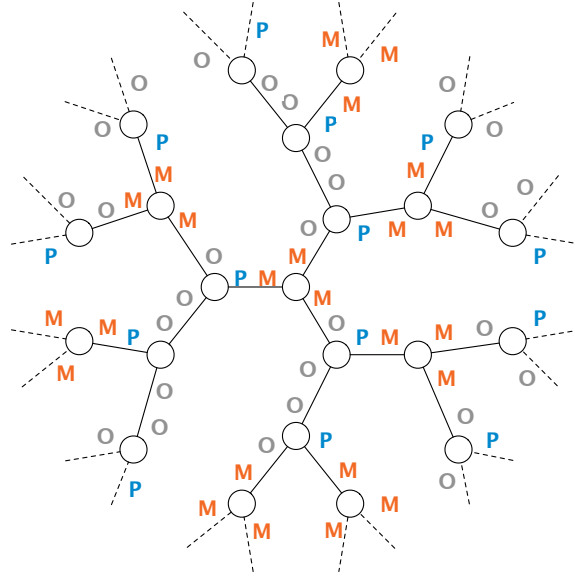| M	M	M
P	O	O

Let us now define edge constraints. The idea is that a pointer must point to a node in the MIS, so, on an edge, **P** can be compatible only with **M**. Also two neighboring nodes must not be both in the MIS, so **M** must *not* be compatible with itself. Finally, if a node is not in the MIS, and it already pointed to some neighbor with a **P**, then all other neighbors can be anything, so **O** can be compatible with both **O** and **M**. This gives the following set B.
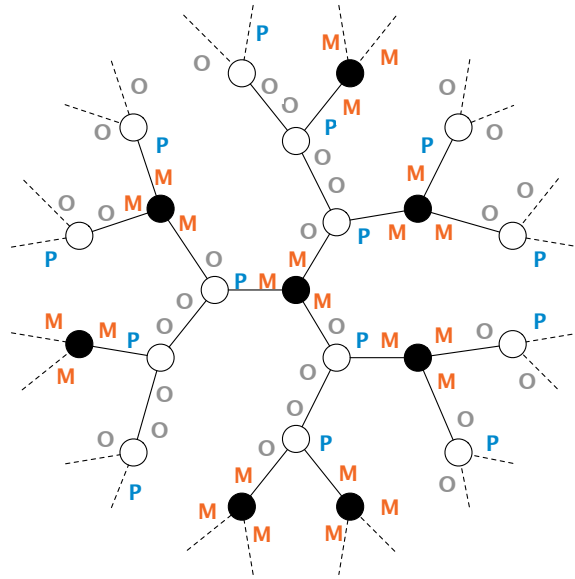
M	P
O	OM

Please note that there is no unique way of writing these constraints. In fact, an alternative way is the following:

M	OP
O	O

An example of a solution is the following.

This labeling represents the following MIS, where nodes of the MIS are colored black.

Note that what we defined is not exactly the MIS problem. In fact, given an MIS, it requires exactly one round for nodes not in the MIS to know where to point, and thus output a valid solution. Nevertheless, since the complexity of the two problems differs by at most 1 round, if we prove a lower bound for one problem, we directly get a lower bound for the other as well.

Example: Bipartite Maximal Matching (BMM)   We have a bipartite 2-colored graph, and we want to choose a subset of edges that are independent, such that this set cannot be extended. Again, 2 labels are not enough to encode the problem: if a node has no incident edges in the matching, then *all* neighbors must have one edge in the matching, so we need to use pointers like in the MIS case. The white constraints are defined as follows.

M	O	O
P	P	P

That is, a white node either outputs **M** on an edge (the edge in the matching) and **O** on all the others, or it outputs **P** on all incident edges (all black neighbors must be matched).
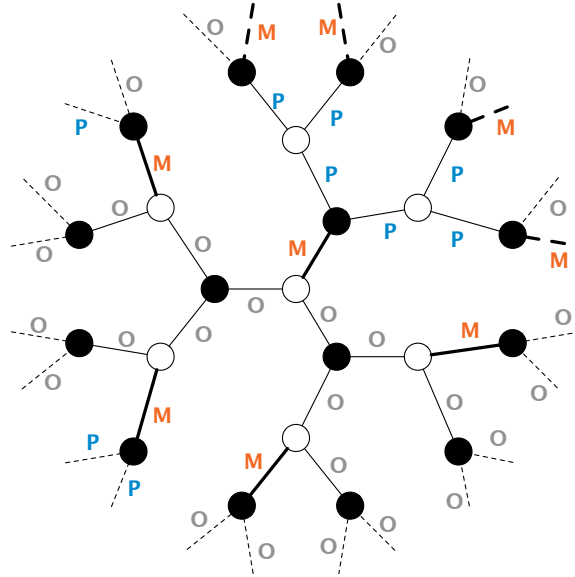
Let us now define the black constraints. The idea is that a black node can accept a pointer only if it has an edge in the matching, that is, **P** can appear only if **M** appears as well. Also, two **M**s cannot appear together. Thus, the black constraint is defined as follows.

M	OP	OP
O	O	O

We can see that the independence of the edges is guaranteed by the fact that for both white and black nodes it holds that at most one incident edge is labeled with **M**. Then, maximality for white nodes is guaranteed by the fact that if a node has no incident edges labeled **M**, then it is outputting **P** everywhere, and these **P**s are accepted by black nodes only if they are matched. Similarly, maximality for black nodes is guaranteed by the fact that if a black node is unmatched, it writes **O** everywhere, and a white node accepts an **O** only if it also has an **M**.

An example of a solution is the following.

This labeling represents the following Maximal Matching.

Example: $\Delta + 1$ Vertex Coloring   This problem makes sense on non-2-colored graphs. Thus we will see an example where $\Delta = 3$ and $\delta = 2$, that is, white nodes represent nodes and black nodes represent edges. We will use 4 labels, **R**,**G**,**B**, and **Y**, that correspond to the 4 possible colors. Nodes must output a color consistently, that is, they need to write the same color on each incident edge. That is, W is defined as follows.
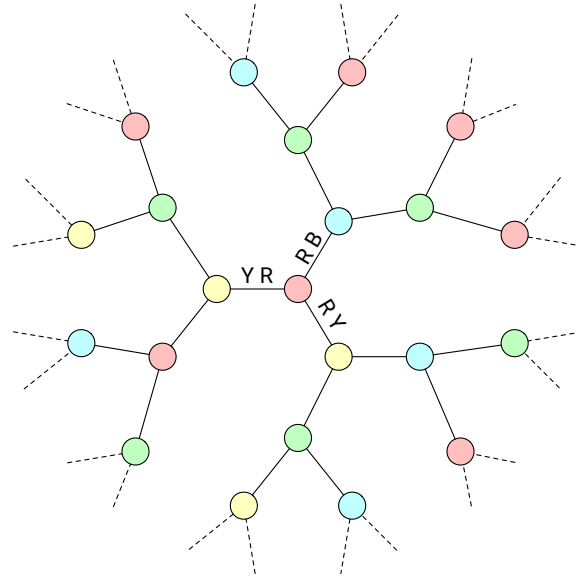
R	**R**	**R**
G	**G**	**G**
B	**B**	**B**
Y	**Y**	**Y**

Then, edges must satisfy that if on one side there is one color, on the other side there must be a different color. Thus, B is defined as follows.

R	**GBY**
G	**RBY**
B	**RGY**
Y	**RGB**

Note that some options are repeated multiple times (for example **R** with **B** on the first line, and **B** with **R** on the third line). This is an allowed input for the round eliminator and does not make any difference.

An example of a solution, with the corresponding coloring, is the following:

Giving inputs to the round eliminator   By opening the round eliminator, we get the following interface.

There are two input areas, the left one corresponds to the constraints for the active side, that at the beginning corresponds to the white nodes, while the right one corresponds to the constraints for the passive side, that at the beginning corresponds to the black nodes. Also, there is a button, "Start", that can be used to send the entered input to the program. For example, we can write $\boxed{\textbf{B AB AB}}$ on the left, and $\boxed{\textbf{A AB AB}}$ on the right, and press the button "Start". This gives the BSO problem as input to the round eliminator. The program will parse this input and output some useful information about the problem.

In particular, please note the following parts:

- The program detected that this problem is not 0-rounds solvable.

- The active and passive constraints are just printed back.

- The diagram.

- The tools, that will be explained in the next sections. They are necessary to perform operations on the problem.

The diagram is a useful part of the interface, that can be used to better understand the *strength* of the labels. For example, for the BSO problem, we can see that there is a direct edge from the label **B** to the label **A**. This means that, from the point of view of the passive side (the black constraints), each time the label **B** is allowed, the label **A** is allowed as well. In more detail, if some configuration containing a **B** is in the black constraint, we can replace **B** with **A** and obtain a new configuration that is also in the black constraint.

Sometimes we want to be more explicit about the labels, and instead of just using arbitrary letters, we prefer to give more sensible names. This can be done in the round eliminator by delimiting a label with parentheses. For example, while "**AB**" basically means "**A** or **B**", we could write "(**AB**)" and this would represent a single label. To give another example, in order to encode BSO, we could use the two labels "**B->W**" and "**W->B**", and give the following input to the round eliminator (please notice the two spaces between the three groups in each input box).

3 Basics

In order to play with the round eliminator, it is not really necessary to know the details of [1], but once we get a nice result, for example a nice lower bound for BMM for some specific values of Δ and δ, we would like to get a general proof for any value of Δ and δ. For this purpose, it is necessary to know how the problem Π' (the problem that requires 1 round less to be solved) is defined as a function of Π (the original problem). Thus, we will now briefly and informally see how Π' is defined, and an example of how to compute Π' starting from Π.

The problem Π' We are given a problem $\Pi = (L, W, B)$ having complexity T rounds, for some unknown value of T, where white nodes are the active entity. We want to compute a problem $\Pi' = (L', B', W')$ of complexity $T - 1$ rounds, where the black nodes are the active entity. Let us start with the new label set L'. The new labels are all the possible sets of old labels, excluding the empty set. That is, $L' = 2^L \setminus \{\{\}\}$. Then, the set B' is the *maximal* set satisfying the following:

$$\forall\, Y = \{Y_1, \ldots, Y_d\} \in B', \ \forall\, (y_1, \ldots, y_d) \in Y_1 \times \ldots \times Y_d : \ \{y_1, \ldots, y_d\} \in B$$

Finally, the set W' is the *maximal* set satisfying the following:

$$\forall\, X = \{X_1, \ldots, X_\Delta\} \in W', \ \exists\, (x_1, \ldots, x_\Delta) \in X_1 \times \ldots \times X_\Delta : \ \{x_1, \ldots, x_\Delta\} \in W$$

In other words, the new black constraints ask black nodes to output a set on each incident edge, and these sets must satisfy that for any choice that we make by picking one element for each set, this choice should be part of the original black constraints. Similarly, the new white constraints ask that the sets present on the incident edges of white nodes (given as output by the black nodes, since in the new problem white nodes are passive) satisfy that there exists a choice that is part of the original white constraints. The reason for why this problem has complexity exactly $T - 1$ rounds can be found in [1]. There are some simplifications that we can always perform that do not change the round complexity of the problems, and the round elimination always performs them. We will see an example of such simplification in the following paragraph, that shows how we can compute the new problem Π' by hand.

Manually computing Π' We will now see an example of how to compute Π', assuming that Π is the BSO problem. Recall that:

$$L = \{\mathbf{A}, \mathbf{B}\}$$
$$W = \{\{\mathbf{B}, \mathbf{A}, \mathbf{A}\}, \{\mathbf{B}, \mathbf{A}, \mathbf{B}\}, \{\mathbf{B}, \mathbf{B}, \mathbf{B}\}\}$$
$$B = \{\{\mathbf{A}, \mathbf{A}, \mathbf{A}\}, \{\mathbf{A}, \mathbf{A}, \mathbf{B}\}, \{\mathbf{A}, \mathbf{B}, \mathbf{B}\}\}$$

First of all, $L' = \{\{\mathbf{A}\}, \{\mathbf{B}\}, \{\mathbf{A}, \mathbf{B}\}\}$. In order to compute B', we start by listing all possible multisets using labels of L', and then exclude the invalid ones. The candidates are:

$$\{\{\mathbf{A}\}, \{\mathbf{A}\}, \{\mathbf{A}\}\} \qquad \{\{\mathbf{A}\}, \{\mathbf{A}\}, \{\mathbf{B}\}\} \qquad \{\{\mathbf{A}\}, \{\mathbf{A}\}, \{\mathbf{A}, \mathbf{B}\}\}$$
$$\{\{\mathbf{A}\}, \{\mathbf{B}\}, \{\mathbf{B}\}\} \qquad \{\{\mathbf{A}\}, \{\mathbf{B}\}, \{\mathbf{A}, \mathbf{B}\}\} \qquad \{\{\mathbf{A}\}, \{\mathbf{A}, \mathbf{B}\}, \{\mathbf{A}, \mathbf{B}\}\}$$
$$\{\{\mathbf{B}\}, \{\mathbf{B}\}, \{\mathbf{B}\}\} \qquad \{\{\mathbf{B}\}, \{\mathbf{B}\}, \{\mathbf{A}, \mathbf{B}\}\} \qquad \{\{\mathbf{B}\}, \{\mathbf{A}, \mathbf{B}\}, \{\mathbf{A}, \mathbf{B}\}\}$$
$$\{\{\mathbf{A}, \mathbf{B}\}, \{\mathbf{A}, \mathbf{B}\}, \{\mathbf{A}, \mathbf{B}\}\}$$

We now need to keep only candidates for which any possible choice is in the set **B**. For example, $\{\{\mathbf{A},\mathbf{B}\},\{\mathbf{A},\mathbf{B}\},\{\mathbf{A},\mathbf{B}\}\}$ is invalid, since it is possible to pick **B** from each subset, and $\{\mathbf{B},\mathbf{B},\mathbf{B}\}$ is not in B. The valid multisets are:

$$\{\{\mathbf{A}\},\{\mathbf{A}\},\{\mathbf{A}\}\} \qquad \{\{\mathbf{A}\},\{\mathbf{A}\},\{\mathbf{B}\}\} \qquad \{\{\mathbf{A}\},\{\mathbf{A}\},\{\mathbf{A},\mathbf{B}\}\}$$
$$\{\{\mathbf{A}\},\{\mathbf{B}\},\{\mathbf{B}\}\} \qquad \{\{\mathbf{A}\},\{\mathbf{B}\},\{\mathbf{A},\mathbf{B}\}\} \qquad \{\{\mathbf{A}\},\{\mathbf{A},\mathbf{B}\},\{\mathbf{A},\mathbf{B}\}\}$$

Now, we could just rename the sets to obtain something more readable. For example, let $\mathbf{X} = \{\mathbf{A}\}$, $\mathbf{Y} = \{\mathbf{B}\}$, $\mathbf{Z} = \{\mathbf{A},\mathbf{B}\}$. We would get that B' is $\boxed{\mathbf{X\ XYZ\ XYZ}}$. Instead, we will first get rid of more multisets, without changing the complexity of the problem. In fact, one can prove that if a multiset is not *maximal*, then it is not needed [1]. For example, consider the two multisets $\{\{\mathbf{A}\},\{\mathbf{A}\},\{\mathbf{A},\mathbf{B}\}\}$ and $\{\{\mathbf{A}\},\{\mathbf{A},\mathbf{B}\},\{\mathbf{A},\mathbf{B}\}\}$. We can find a perfect matching between the sets of the first and the second, such that each set of the first is included in its matched set. For this reason, we can safely ignore $\{\{\mathbf{A}\},\{\mathbf{A}\},\{\mathbf{A},\mathbf{B}\}\}$. By removing the non maximal multisets, we can notice that only one remains:

$$\{\{\mathbf{A}\},\{\mathbf{A},\mathbf{B}\},\{\mathbf{A},\mathbf{B}\}\}$$

Notice that the set $\{\mathbf{B}\}$ never appears, so we can actually redefine the set L' to contain only $\{\mathbf{A}\}$ and $\{\mathbf{A},\mathbf{B}\}$. By using the aforementioned renaming we get that the new black constraints B' are just $\boxed{\mathbf{X\ Z\ Z}}$.

Let us now see how to compute W'. Again, we can start from all the possible candidates:

$$\{\{\mathbf{A}\},\{\mathbf{A}\},\{\mathbf{A}\}\} \qquad \{\{\mathbf{A}\},\{\mathbf{A}\},\{\mathbf{B}\}\} \qquad \{\{\mathbf{A}\},\{\mathbf{A}\},\{\mathbf{A},\mathbf{B}\}\}$$
$$\{\{\mathbf{A}\},\{\mathbf{B}\},\{\mathbf{B}\}\} \qquad \{\{\mathbf{A}\},\{\mathbf{B}\},\{\mathbf{A},\mathbf{B}\}\} \qquad \{\{\mathbf{A}\},\{\mathbf{A},\mathbf{B}\},\{\mathbf{A},\mathbf{B}\}\}$$
$$\{\{\mathbf{B}\},\{\mathbf{B}\},\{\mathbf{B}\}\} \qquad \{\{\mathbf{B}\},\{\mathbf{B}\},\{\mathbf{A},\mathbf{B}\}\} \qquad \{\{\mathbf{B}\},\{\mathbf{A},\mathbf{B}\},\{\mathbf{A},\mathbf{B}\}\}$$
$$\{\{\mathbf{A},\mathbf{B}\},\{\mathbf{A},\mathbf{B}\},\{\mathbf{A},\mathbf{B}\}\}$$

We now need to keep only candidates for which there exists a choice that is in the set W. In other words, we need to exclude all candidates for which it is not possible to pick at least one **B**. The only bad candidate is $\{\{\mathbf{A}\},\{\mathbf{A}\},\{\mathbf{A}\}\}$. Thus, we get the following:

$$\{\{\mathbf{A}\},\{\mathbf{A}\},\{\mathbf{B}\}\} \qquad \{\{\mathbf{A}\},\{\mathbf{A}\},\{\mathbf{A},\mathbf{B}\}\} \qquad \{\{\mathbf{A}\},\{\mathbf{B}\},\{\mathbf{B}\}\}$$
$$\{\{\mathbf{A}\},\{\mathbf{B}\},\{\mathbf{A},\mathbf{B}\}\} \qquad \{\{\mathbf{A}\},\{\mathbf{A},\mathbf{B}\},\{\mathbf{A},\mathbf{B}\}\} \qquad \{\{\mathbf{B}\},\{\mathbf{B}\},\{\mathbf{B}\}\}$$
$$\{\{\mathbf{B}\},\{\mathbf{B}\},\{\mathbf{A},\mathbf{B}\}\} \qquad \{\{\mathbf{B}\},\{\mathbf{A},\mathbf{B}\},\{\mathbf{A},\mathbf{B}\}\} \qquad \{\{\mathbf{A},\mathbf{B}\},\{\mathbf{A},\mathbf{B}\},\{\mathbf{A},\mathbf{B}\}\}$$

Now, since $\{\mathbf{B}\}$ never appears in B', it never makes sense to have it in W', so we can remove all candidates containg $\{\mathbf{B}\}$. Thus, we get the following:

$$\{\{\mathbf{A}\},\{\mathbf{A}\},\{\mathbf{A},\mathbf{B}\}\} \qquad \{\{\mathbf{A}\},\{\mathbf{A},\mathbf{B}\},\{\mathbf{A},\mathbf{B}\}\} \qquad \{\{\mathbf{A},\mathbf{B}\},\{\mathbf{A},\mathbf{B}\},\{\mathbf{A},\mathbf{B}\}\}$$

By renaming, we get that the new white constraints W' are $\boxed{\mathbf{Z\ XZ\ XZ}}$.

Automatically computing Π' After having given in input a problem to the round eliminator, we can press the button "Speedup" to let the program compute the new problem Π' automatically. For example, starting from the BSO problem, thus by writing $\boxed{\mathbf{B\ AB\ AB}}$ on the left and $\boxed{\mathbf{A\ AB\ AB}}$ on the right, and then pressing "Start", and then "Speedup", we get the following result:

Close

The problem is NOT zero rounds solvable.

Active (Before Renaming)	Passive (Before Renaming)	Renaming
Any choice satisfies previous Passive	Exists choice satisfying previous Active	Old and new labels

Active	Passive	Diagram	Tools
Any choice satisfies previous Passive	Exists choice satisfying previous Active	Strength of right side labels	Speedup, edit, simplifications, ...

Active:
B   B   A

Passive:
AB   AB   B

Diagram: B → A

Tools:
Speedup   Edit

Simplifications

Harden v1

Harden v2

Automatic Lower Bound

Automatic Upper Bound

New Renaming

This is the same result that we obtained manually, just with a different renaming. By default, the sets are just renamed in alphabetical order. In fact, we can suggest to the round eliminator a different renaming, by pressing "New Renaming". We can for example suggest the same renaming used before:

New Renaming

A   X

AB   Z

Rename

By pressing "Rename" we get the following result:

If we are interested in seeing the result before the automatic renaming, or in just seeing what is the renaming used by the program, we can press either "Active (Before Renaming)", "Passive (Before Renaming)", or "Renaming". A set of labels will be represented by enclosing multiple labels in a box. We get the following:

If we are interested in modifying the current problem, we can press the button "Edit". By doing so, we can notice that the input areas (that can be found by scrolling again to the top) now contain the problem Π':

Active	Passive
Z Z X	XZ XZ Z

Start Clear

4 Lower Bounds

We have seen how to feed a problem to the round eliminator and how to get, by pressing a button, a problem that is exactly one round easier. It is time to see how to use this tool to get proper lower bounds.

Bipartite Sinkless Orientation Start from $\Pi_0 = \text{BSO}$, that is, by writing $\boxed{\textbf{B AB AB}}$ on the left side, and $\boxed{\textbf{A AB AB}}$ on the right side. Press "Start" and then "Speedup". What is shown is the problem Π_1, that has complexity exactly one round less than BSO. On one side we have the constraint $\boxed{\textbf{A B B}}$, while on the other side we have the constraint $\boxed{\textbf{B AB AB}}$. Let us press again the button "Speedup". We get the following:

This is not an error, we got a problem Π_2 that is exactly the problem Π_1 on which we pressed the button! In other words, Π_1 is a fixed point: by applying the round elimination technique on this problem we get back to the same problem. This is somehow a contradiction, how can Π_1 and Π_2 have the same complexity? Π_2 should require exactly one round less than Π_1. The solution to this enigma is that, in order to apply the technique, we need to assume that $T = o(\log n)$, and a fixed point basically shows that under this assumption we get a contradiction. In other words, a fixed point given by a non 0-rounds solvable problem directly implies a lower bound of $\Omega(\log n)$.

Typically, this kind of lower bounds can be lifted to the LOCAL model to obtain an $\Omega(\log n)$ lower bound for deterministic algorithms and an $\Omega(\log \log n)$ lower bound for randomized algorithms (see e.g. the FDSO problem in [2]).

Sinkless Orientation (SO)   Sometimes we cannot hope to get lower bounds on the bipartite case, since many problems are trivial given a bipartition. Also, it is not possible to get fixed points directly in the non bipartite case, since $\delta = 2$ but typically $\Delta > 2$. So the resulting problem cannot be the same as the original problem, since at least the degrees differ. Nevertheless we can get something very similar to a fixed point, that is a cycle, and SO is an example of such behavior. Start with $\Pi_0 = $ SO: $\boxed{\textbf{T HT HT}}$ on the left side, and $\boxed{\textbf{H T}}$ on the right side. Press "Start". Press the button "Speedup" *three* times (each time on the new obtained problem). Note that the last problem that we get, Π_3, is different from the previous one, Π_2, but is the same as Π_1. That is, we did not get a fixed point, but something very similar. We have a problem Π_3 that is 2 (half) rounds easier than itself! For the same reasoning used in the BSO case, this directly gives an $\Omega(\log n)$ lower bound.

Bipartite Sourceless and Sinkless Orientation (BSSO)   Fixed points are good. If we get a fixed point we get a lower bound of $\Omega(\log n)$ for free. Do we get a fixed point for any problem that is $\Omega(\log n)$? Unfortunately no, we do not. Understanding when and why we get fixed points is an open question. In the BSSO problem, each node must have at least one incoming *and* one outgoing edge. BSSO is an example of a problem for which we know a lower bound of $\Omega(\log n)$ (since it is at least as hard as BSO), but it does not give a fixed point. It can be described by having both white and black constraint equal to $\boxed{\textbf{A B AB}}$. If we feed this problem to the round eliminator, and press "Speedup" multiple times (each time on the new obtained problem), we can see that we do *not* get a fixed point. Instead, each time we get a new problem that uses one label more than the previous one. Here is an example of the result after 8 steps:

How can we prove lower bounds for this kind of problems? One way would be to characterize all problems Π_i, and prove that none of them is 0-rounds solvable. While this may be doable for BSSO, this may be very challenging for more complicated problems. An alternative way to get a lower bound for a problem Π is to try to relax the problem (that is, make it easier) obtaining a new problem Π', and hope that Π' gives a fixed point. This would directly imply a lower bound for the harder problem Π. For example, it is easy to do so for BSSO: we can relax it to BSO! More generally, what we can always do, is to add more configurations to the black and white constraints, and this can only make the problem easier. In particular, if we add $\boxed{\textbf{B B B}}$ to the white constraint of BSSO, and add $\boxed{\textbf{A A A}}$ to the black constraint, we get the BSO problem, for which we know a lower bound.

Making a problem too easy Can we always get a fixed point for problems that are $\Omega(\log n)$ by relaxing them? We do not know! What we know is that if we make the wrong relaxations we get a problem that is too easy, and thus we do not get a lower bound. For example, for BSSO, if instead of adding $\boxed{\textbf{A A A}}$ to one side and $\boxed{\textbf{B B B}}$ to the other we add $\boxed{\textbf{A A A}}$ to both sides, we get a 0-rounds solvable problem (in fact, if all nodes output $\boxed{\textbf{A A A}}$ we get a solution that satisfies both white and black constraint). The round eliminator would inform us about that:

Thus, if we are aiming to prove an $\Omega(\log n)$ lower bound, one possible way is to try to relax the problem and hope to get a fixed point, while trying to avoid to relax it too much and get a trivial problem.

Diagram Before seeing more techniques that can be used to prove lower bounds, we need to understand the diagram associated to a problem. For example, by giving BSO as input, we get the following:

Close

The problem is NOT zero rounds solvable.

Active	Passive	Diagram	Tools
Any choice satisfies previous Passive	Exists choice satisfying previous Active	Strength of right side labels	Speedup, edit, simplifications, ...

AB   AB   B

AB   AB   A

B

A

Speedup   Edit

Simplifications

Harden v1

Harden v2

Automatic Lower Bound

Automatic Upper Bound

If we look at the diagram, we can see that there is a direct edge from the label **B** to the label **A**. This means that, on the passive side (the black constraint), each time the label **B** is allowed, the label **A** is allowed as well. In more detail, if some configuration containing a **B** is in the black constraint, we can replace **B** with **A** and obtain a new configuration that is also in the black constraint. This diagram is important for multiple reasons.

We know that the labels of the new problem are a subset of all possible sets of old labels. We know something more: one can prove that we do not have to consider all possible sets, we only have to consider *right closed subsets* with regards to the diagram of the original problem [1]. In other words, if on the passive side a label **A** is allowed each time a label **B** is allowed, then the sets of the new problem will satisfy that if a set contains **B**, then it must contain **A** as well. That's the reason why, by applying the round elimination technique on the BSO problem, we do *not* get the set {**B**} as a new label. Instead, it disappears, and we only get {**A**} and {**A**, **B**}. Thus, we can somehow control the number of labels of the new problem, by trying to limit the number of right closed subsets of the old problem.
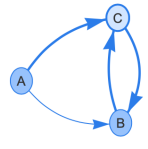
Another reason why we care about the diagram is the following. Consider the following example:

This is an arbitrary artificial problem that is at least as hard as BSO. We can notice that there is a bidirectional edge between the labels B and C, and the round eliminator suggests that these labels are equivalent. Why are they equivalent? It is true that, from the point of view of black nodes, each time B is allowed, C is allowed as well, and vice versa, but the same is not true for the white nodes. A white node can have 2 edges labeled B, but only one labeled C. Nevertheless, if we define a new problem where we "identify" the labels B and C, this will have the same complexity as the original problem. The idea is that, since black nodes do not really care whether white nodes write B or C (remember that white nodes are the only ones writing outputs), then in 0 rounds we could transform the solution of one problem to a solution for the other problem (and this really works in both ways). This merging of labels can be achieved by pressing "Simplifications" and then either the button "$B \rightarrow C$" or the button "$C \rightarrow B$". We obtain a problem that uses fewer labels, and it has the same complexity as the original one. This is the best thing we could wish for, since, as we will see later, having few labels helps in speedup process.

Guaranteed Fixed Point Should we just try to randomly add allowed configurations until we get a fixed point? Well, that may work, but it may also require a lot of time. The main issue is that we really do not know why and when we get fixed points. We know something, though. We know that the labels of the new problem can only be *right closed subsets* with regards to the diagram. This in turn implies that, if a problem uses k labels, and these k labels form a diagram that is a directed path, we get the following: the number of new labels is also upper bounded by k, since that is the number of right closed subsets in the directed path, and the new diagram is also a directed path (with possibly additional arrows). In other words, these problems behave really nicely using the round elimination technique. That is, they do not *grow*. We can press "Speedup" as many times as we want, and we will always obtain a problem with at most k labels. Since the number of labels is bounded, by keeping pressing "Speedup", we cannot get a new problem forever: at some point we must get an old problem, that is, we get a cycle. Basically, for these problems we are guaranteed to find either a fixed point or a cycle (a cycle is as good as a fixed point to prove lower bounds). Does this mean that all problems of this kind are $\Omega(\log n)$? Unfortunately not: during this process we may obtain a 0-rounds solvable problem, and get a fixed point only after this. An example is the

problem having constraint "A AB AB" on both sides: if we feed this into the round eliminator we notice two things:

- It is 0-rounds solvable.

- The diagram is a directed path of length 2 (that is equal to the number of labels).
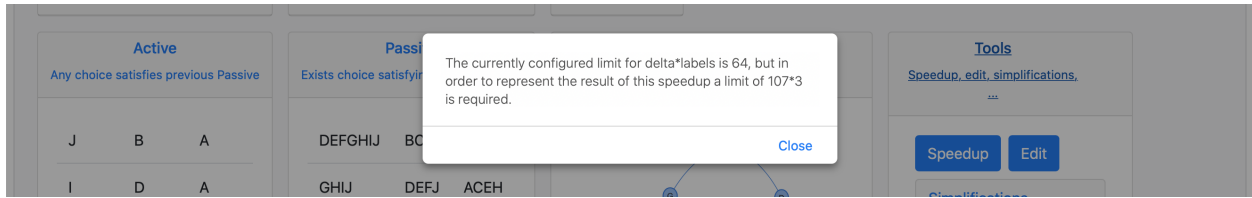
By pressing "Speedup" multiple times, we can notice that $\Pi_3 = \Pi_2$, that is, we got a fixed point, but it is a fixed point given by a 0-round problem, thus we got no lower bound for free.

Can we get fixed points only for problems having a path as diagram? No, there are others, and an example can be found in [2], where a problem called FDSO is defined. That problem gives a fixed point, but the diagram is a square. Understanding why we get fixed points is an open question, and answering this question may really help in proving lower bounds.

Bipartite $2\Delta - 2$ **edge coloring** We now know a class of problems that are guaranteed to give a fixed point (sometimes a useless one, though). Can we use this knowledge to prove a lower bound for problems that are *not* in this class? An example of a problem that does not give a fixed point, and that is hard to handle, is the bipartite $2\Delta - 2$ edge coloring, that can be encoded (for $\Delta = \delta = 3$) by using the following constraint for both white and black nodes.

A	B	CD
A	C	D
B	C	D

In fact, by feeding this problem to the round eliminator, and by pressing "Speedup" twice, we get an error, saying that the resulting problem is too big to be handled by the program, as it contains 107 different labels. While the round eliminator can be configured to handle more labels, it may be hopeless to try to read a huge resulting problem and try to understand it.

Instead, we could try a different approach. We could try to add allowed configurations and hope for a fixed point. We now know a class of problems that are guaranteed to give a fixed point, the ones where the diagram contains all labels and forms a directed path. What we could try to do is to add configurations, in such a way that the diagram changes shape, and becomes one that looks like the desired path. Let us recap:

- We have a problem that uses 4 labels, its diagram does not contain any edge.

- We want to obtain a diagram that is a directed path of 4 nodes.

One way to do that is the following: we assign an arbitrary order to the labels, such as $\mathbf{A} < \mathbf{B} < \mathbf{C} < \mathbf{D}$. We now process labels in order. We relax the constraint such that each time $\mathbf{A}$ is allowed, $\mathbf{B}$ is allowed as well, thus, each time there is an $\mathbf{A}$, we add a $\mathbf{B}$. This will ensure that the edge $\mathbf{A} \rightarrow \mathbf{B}$ is in the diagram. We repeat the same procedure for the other labels. Each time $\mathbf{B}$ appears, we add the label $\mathbf{C}$. Each time the label $\mathbf{C}$ appears, we add the label $\mathbf{D}$. This procedure must be done only on the passive side (since the diagram depends only on that). Thus, we obtain the following problem. The white constraint is the same as before, while the black constraint is the following:

ABCD	BCD	CD
ABCD	CD	D
BCD	CD	D

By performing this simplification, we are sure that the diagram contains a path $\mathbf{A} \rightarrow \mathbf{B} \rightarrow \mathbf{C} \rightarrow \mathbf{D}$. It may actually contain more edges, in the case where, by adding allowed configurations, we made two labels interchangeable (from the point of view of the passive side). This is not an issue, the guaranteed fixed point still applies. This is what we get by feeding the new problem to the round eliminator.

Close

The problem is NOT zero rounds solvable.

The following labels could be merged without changing the complexity of the problem: CD

Active			**Passive**			**Diagram**	**Tools**
Any choice satisfies previous Passive			Exists choice satisfying previous Active			Strength of right side labels	Speedup, edit, simplifications, ...
CD	B	A	CD	BCD	ABCD		
D	C	A					
D	C	B					

Tools:

Speedup   Edit

Simplifications

Harden v1

Harden v2

Automatic Lower Bound

Automatic Upper Bound

We can note that, at least at this point, the problem is still not 0-rounds solvable. Also, we can note that the diagram is not exactly a path, since labels **C** and **D** became equivalent. The round eliminator actually informs about that. We could merge these labels, but it is not necessary. We can now press "Speedup" multiple times, and note that we get a fixed point, and it is the same fixed point that we obtained with BSO. By generalizing this reasoning for all values of Δ, we get that $2\Delta - 2$ edge coloring requires $\Omega(\log n)$, even if nodes are 2-colored.

An alternative way to obtain a diagram that looks like a path is by using the "Add arrows" feature of the round eliminator.

Initial problem.

Close

The problem is NOT zero rounds solvable.

Active	Passive	Diagram
Any choice satisfies previous Passive	Exists choice satisfying previous Active	Strength of right side labels

Active

CD	B	A
D	C	A
D	C	B

Passive

CD	B	A
D	C	A
D	C	B

Diagram

D

A

C

B

Tools

Speedup, edit, simplifications, ...

Speedup   Edit

Simplifications

Add arrows

Available simplifications:

A→B   A→C   A→D   B→A   B→C   B→D   C→A   C→B   C→D   D→A   D→B   D→C

Harden v1

Harden v2

Automatic Lower Bound

Automatic Upper Bound

This feature allows us to add an arrow between labels that in the diagram are not connected. For example, by pressing the button "$\mathbf{A}{\to}\mathbf{B}$", we get a new problem where each time $\mathbf{A}$ is allowed $\mathbf{B}$ is allowed as well.

Simplifications   Sometimes we know that a problem can be solved in $o(\log n)$, so it does not make sense to try to get a fixed point. Sometimes we just want to prove some $\Omega(f(\Delta))$ lower bound, for some unknown function f that we want to infer. For example, we may want to prove a linear in Δ lower bound for some local problem. One thing that we could try to do is to press the "Speedup" button Δ times and see if the result that we get is a problem that cannot be solved in 0 rounds. Unfortunately, it often happens that the problem that we get after pressing "Speedup" is much

larger than the original one. In fact, the new number of labels can be exponentially larger than the number of labels of the original problem, in the worst case. This creates multiple issues:

- The round eliminator could become too slow, if we give it a problem with too many labels.

- Even if we get some meaningful result, it could be so large that it is hopeless to try to read it, understand it, and try to generalize it for any value of Δ.
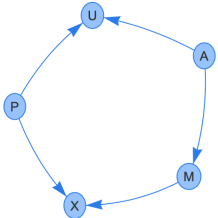
Thus, one way to solve this issue is to try to reduce the number of labels. If we want to prove a lower bound, we cannot just remove some labels, as it would make the problem harder. The only thing that we can do is to try to reduce the number of labels by making the problem easier. Thus we need to find a way to get rid of labels by adding allowed configurations to the black and white constraints. The idea is to *merge* labels. Merging labels means that we add allowed configurations such that each time one label is allowed, we allow the other as well. In this way, we obtain two labels that are equivalent, thus we can just remove one without making the problem harder. How to find the *right* labels to merge? It is not easy, and sometimes we have some large problem that we cannot understand, and the only thing that we can do is to try to merge labels randomly and hope for the best. Sometimes instead, the problem that we have can be interpreted in some meaningful manner, and we can try to find two labels that we think are similar enough, such that if we merge them, we get a problem that is not much easier.

In order to merge labels in the round eliminator, we have two ways. One is to press the "Edit" button, modify the constraints as described, and press "Start" again. Another is to press "Simplifications" and choose the desired merging. For example, consider the following (artificial) problem:

By choosing the label **A** as the label to replace, the label **M** as the replacement, and by pressing "Simplify", the round eliminator merges the labels **A** and **M** and gets rid of the label **A**. From now on, we will refer to this operation by just saying to perform the simplification $\mathbf{A} \to \mathbf{M}$. The result that we get is the following:

We still do not understand what are the right simplifications that we should do, but we have some examples of lower bounds that give some ideas about strategies that sometimes work [3]. A nice scenario is the one where each time we get the *same* number of labels, with problems that are all similar to each other. That is, we start from a problem with k labels, we press "Speedup" and we get some large problem, we make some simplifications and we get a problem with k labels. We press "Speedup" again and we get again some large problem, we make the *same* simplifications done before and we get a problem with k labels again, and we repeat. Usually this gives a nice problem family, that can be described in a compact way, and this makes it doable to prove properties on this family. One trick that we can use to get this nice family is to try to make the diagram look the same at each step. This somehow captures the case in which we have two problems that are similar to each other. This does not always work, and sometimes we just get 0-rounds solvable problems, but sometimes it works, and it allows us to prove lower bounds.

Bipartite Maximal Matching (BMM)   This is an example of a problem for which we can prove a nice lower bound easily, if we do the right simplifications. We will see how, by just looking at the diagram and by trying to make it look the same at each step, we can easily prove a $2\Delta - 1$ lower bound for this problem. Notice that this problem can be actually solved in $2\Delta - 1$ rounds, so we really get a tight result here. We will see an example where $\Delta = 4$, but it can be generalized. Thus, we will start with the problem where the white and black constraints are the following.

M	O	O
P	P	P

M	OP	OP
O	O	O

Let us start by pressing "Start" and then "Speedup" twice. We get the following.

Close

The problem is NOT zero rounds solvable.

Active (Before Renaming)	Passive (Before Renaming)	Renaming
Any choice satisfies previous Passive	Exists choice satisfying previous Active	Old and new labels

Active	Passive	Diagram	Tools
Any choice satisfies previous Passive	Exists choice satisfying previous Active	Strength of right side labels	Speedup, edit, simplifications, ...

Active

F	E	E	A
E	E	E	B
D	C	C	C

Passive

CDEF	CDEF	CDEF	BDF
EF	ABCDEF	EF	EF

Tools

Speedup · Edit

Simplifications

Harden v1

Harden v2

Automatic Lower Bound

Automatic Upper Bound

New Renaming

We will now do the only guess that is needed to get the lower bound, that is, we will perform the simplification $\mathbf{D} \to \mathbf{F}$. We get the following.

Close

The problem is NOT zero rounds solvable.

Active (Before Renaming)	Passive (Before Renaming)	Renaming
Any choice satisfies previous Passive	Exists choice satisfying previous Active	Old and new labels

Active	Passive	Diagram	Tools
Any choice satisfies previous Passive	Exists choice satisfying previous Active	Strength of right side labels	Speedup, edit, simplifications, ...

Active

F	E	E	A
E	E	E	B
F	C	C	C

Passive

CEF	CEF	CEF	BF
EF	ABCEF	EF	EF

Tools

Speedup · Edit

Simplifications

Harden v1

Harden v2

Automatic Lower Bound

Automatic Upper Bound

New Renaming

We now press the "Speedup" button, and we get the following.

27

Close

The problem is NOT zero rounds solvable.

Active (Before Renaming)	Passive (Before Renaming)	Renaming
Any choice satisfies previous Passive	Exists choice satisfying previous Active	Old and new labels

Active	Passive	Diagram	Tools
Any choice satisfies previous Passive	Exists choice satisfying previous Active	Strength of right side labels	Speedup, edit, simplifications, ...

	Active				Passive		
F	E	E	A	CDEFG	ABCDEFG	CDEFG	G
E	E	E	B	CDEFG	CDEFG	CDEFG	BDFG
G	C	C	C	EFG	ABCDEFG	EFG	EFG
E	D	C	C				

Speedup   Edit

Simplifications

Harden v1

Harden v2

Automatic Lower Bound

Automatic Upper Bound

New Renaming

Notice that we now got a diagram that is very similar to what we got before the $\mathbf{D} \to \mathbf{F}$ simplification. Thus, we can try to obtain the same diagram again. The idea is that by performing the simplification $\mathbf{X} \to \mathbf{Y}$, we basically push the node $\mathbf{X}$ of the diagram to $\mathbf{Y}$. For example, if we now want to get the same rectangle that we had before, we have to simplify $\mathbf{F} \to \mathbf{G}$. In fact we obtain the following:

Performed simplification F→G

Close

The problem is NOT zero rounds solvable.

Active (Before Renaming)	Passive (Before Renaming)	Renaming
Any choice satisfies previous Passive	Exists choice satisfying previous Active	Old and new labels

Active	Passive	Diagram	Tools
Any choice satisfies previous Passive	Exists choice satisfying previous Active	Strength of right side labels	Speedup, edit, simplifications, ...

	Active				Passive		
G	E	E	A	CDEG	ABCDEG	CDEG	G
E	E	E	B	CDEG	CDEG	CDEG	BDG
G	C	C	C	EG	ABCDEG	EG	EG
E	D	C	C				

Speedup   Edit

Simplifications

Harden v1

Harden v2

Automatic Lower Bound

Automatic Upper Bound

New Renaming

We can now notice that we have the same rectangular diagram that we had few steps back (up to relabeling). What we did at that point was to do the simplification $\mathbf{D} \to \mathbf{F}$. Now, the labels that

are in the same position of **D** and **F** are **D** and **G**. Thus, we can try to do the same simplification that we did before, by simplifying $\mathbf{D} \to \mathbf{G}$. By doing so, we get again a diagram that looks like a cycle of 5 nodes, and by pressing "Speedup" we again get the rectangular diagram:

We can now continue, each time we do the same simplification $\mathbf{D} \to \mathbf{F}$. We can notice that we can repeat this many times before getting a 0-rounds solvable problem. In particular, we get a 0-rounds solvable problem only after having pressed "Speedup" 7 times. This basically gives a 7 rounds lower bound for BMM when $\Delta = 4$. We can then try different values of Δ, to see if this was just a coincidence, or if this sequence of simplifications actually works for different values of Δ. By doing so, one can notice that this chain of simplifications actually works.

The next step is to write a formal proof. How to do that? It is now time to look at the problems that we got at each step. We can notice that they all look very similar. There is a common structure in them. Thus, we can try to write a parametrized family of problems, that, for any Δ and i, describes what is the problem that we get after i steps by performing speedups and simplifications. Once we have this family, we need to prove a generic round elimination result, that is, that if we start from one problem of the family and we apply the round elimination technique, we get a problem that is harder than the next problem of the family. This is basically the only step that we cannot do automatically, since it requires to prove what is the result of the round elimination technique on an infinite family of problems. A proof for the BMM problem can be found in [4]. An example for a different problem, called FDSO, can be found in [2].

Automatic lower bounds   In the BMM example, we had to guess a simplification, and then, by trying to keep the diagram the same, we got a nice lower bound. We do not actually need to perform manual guessing: the round eliminator can do that for us. Let us start again from the BMM problem for the case where $\Delta = 4$. This time let us press "Automatic Lower Bound". We get the following.

Initial problem.

Close

The problem is NOT zero rounds solvable.

Active	Passive	Diagram	Tools
Any choice satisfies previous Passive	Exists choice satisfying previous Active	Strength of right side labels	Speedup, edit, simplifications, ...

Active:
U U U M
P P P P

Passive:
PU PU PU M
U U U U

Diagram:
P → U
M

Tools:

Speedup   Edit

Simplifications

Add arrows

Harden v1

Harden v2

Automatic Lower Bound

Maximum number of iterations:
15

Maximum number of labels:
5

Maximum number of right closed subsets:
20

○ Try to merge unreachable labels
● Try to merge diagram neighbors
○ Try to merge indirect neighbors
● Try to add diagram edges

Automatic Lower Bound

Automatic Upper Bound

There are few parameters that we can tweak:

- Maximum number of iterations. You do not have to worry about this parameter. Just give it some value that is higher than what you think is the lower bound that you should get. This just gives a maximum on the number of rounds that we can get as lower bound. The reason why we need to give a number is the following: if the round eliminator manages to find simplifications that induce a fixed point (or a cycle), then the round eliminator itself would start looping indefinitely by finding each time better and better lower bounds (since it currently does not detect fixed points or cycles automatically).

- Maximum number of labels and maximum number of right closed subsets. What the round eliminator does is the following. After each speedup step, it tries to merge labels until some number of labels is reached. In particular, if we set these limits to k_1 and k_2, it tries all possible mergings that give at most k_1 labels as result *and* a diagram that contains at most k_2 right closed subsets.

- Try to merge unreachable labels. Among the simplifications that are tried, the round eliminator tries to merge labels that are unreachable in the diagram.

- Try to merge diagram neighbors. Among the simplifications that are tried, the round eliminator tries to merge direct neighbors of the diagram.

- Try to merge indirect neighbors. Among the simplifications that are tried, the round eliminator tries to merge two labels if one is reachable from the other in the diagram.

- Try to add diagram edges. Among the simplifications that are tried, the round eliminator tries to add arrows between nodes that are unreachable in the diagram.

By enabling more options, more simplifications are tried, and it requires more time. Usually, the one that works best is "Try to merge diagram neighbors". Let us leave the default values, 15 and 5, and let us keep active only the "Try to merge diagram neighbors" option. Then, press the "Automatic Lower Bound" button. After a while, we get the following:

> Lower bound of 7 rounds.

The round eliminator automatically found the right simplifications to do to get a 7 rounds lower bound! We can then read the result, and we can see what simplifications the program did. What we find is that it basically did the same simplifications that we did manually before. If we try to do the same for the case where $\Delta = 5$, we automatically get a 9 rounds lower bound (it takes more time though). Thus, the only thing that we need to do is to read the simplifications that the program did, see what is the pattern, and try to generalize it.

5 Upper Bounds

Until now we have seen how to apply this technique to get lower bounds. This technique can be actually applied to get upper bounds! The only issue is that the upper bounds that we get are not general. That is, they only hold in a regular tree and high-girth graphs. Thus, if we get something with this technique, we then have to try to generalize it.

The idea is that, while for proving lower bounds we have to make the problem easier at each step and hope to not get a problem that is 0-round solvable, here we have to make the problem harder and hope to *get* a problem that is 0-rounds solvable. How to make the problem harder? It is very easy, just remove labels! In the round eliminator program there are two ways to do that. By pressing "Harden v1" we can select which labels we want to keep. By pressing "Harden v2" we can press a button to remove a label. This is what we see if we start from BMM.

Close

The problem is NOT zero rounds solvable.

Active	Passive	Diagram	Tools
Any choice satisfies previous Passive	Exists choice satisfying previous Active	Strength of right side labels	Speedup, edit, simplifications, ...

Active:
U U U M
P P P P

Passive:
PU PU PU M
U U U U

Diagram:
U ← P

Tools:

Speedup   Edit

Simplifications

Harden v1

Please choose which labels should be kept.

M
P
U

Add diagram predecessors

Harden

Harden v2

Click on the label that you want to remove

M   P   U

Add diagram predecessors

Automatic Lower Bound

Automatic Upper Bound

For example, let us press the button "**P**". The result is the following.

Removed label P

Close

The problem is NOT zero rounds solvable.

Active	Passive	Diagram	Tools
Any choice satisfies previous Passive	Exists choice satisfying previous Active	Strength of right side labels	Speedup, edit, simplifications, ...

Active:
U U U M

Passive:
U U U M
U U U U

Diagram:
U ← M

Tools:

Speedup   Edit

Simplifications

Harden v1

Harden v2

Automatic Lower Bound

Automatic Upper Bound

Often, by removing labels we get problems that are way harder than the original one. For example, we can notice that after removing the label $\mathbf{P}$, if we press "Speedup" many times, we get a fixed point. It means that we made the problem way harder! While BMM can be solved in $O(\Delta)$ rounds, this new problem requires $\Omega(\log n)$, that is the highest possible lower bound that we can get with this technique.

Are there ways to remove labels without making the problem too hard? Actually, yes, and it is the default option in the round eliminator program. Please note that there is an option called "Add diagram predecessors" in both "Harden v1" and "Harden v2", that is enabled by default. The idea is that we do not just remove a label, but instead we replace it with its diagram predecessors, and this allows us to get a problem that is potentially easier than the one that we would get by just removing labels. If we want to prove an upper bound, this is the best option: we reduce the number of labels, so the result after a speedup step does not get too large, but we make the problem as less hard as possible.

Automatic Upper Bounds Again, we do not really need to manually try all possible buttons, and the round eliminator can do that for us. For example, let us start from the BMM problem where $\Delta = 4$, and let us press "Automatic Upper Bound". We get the following.

Here again there are two parameters:

- Maximum number of iterations. Here you should put a value that represents your target complexity, that is, the number of rounds that you think are enough to solve the problem. By

putting high values, the program gets slower (the program currently tries to find a solution in a DFS-like fashion).

- Maximum number of labels and maximum number of right closed subsets. Like in the automatic lower bound case, the round eliminator tries to limit the number of labels in all possible ways at each step. Increasing these values may help in finding an upper bound, but it may make the program slower.

Since we got a 7 rounds lower bound, let us try to get a 7 rounds upper bound. Also, we used a limit of 5 labels to get a lower bound, but let us be optimistic and try to get an upper bound with just 4 labels. Thus, let us put the values 7 and 4 and press "Automatic Upper Bound". We get the following:

> Upper bound of 7 rounds.

The round eliminator found a 7 rounds algorithm, meaning that BMM requires exactly 7 rounds if $\Delta = 4$. By scrolling down we can also see what labels got removed at each step. There are other options that can be used while trying to get an upper bound automatically:

- Slow and Accurate. This enables the "Add diagram predecessors" option while removing labels. This typically makes the result larger, and slows down the process, but it can actually help to get upper bounds (without this option we may get none, since the problem may always get too hard).

- Test. This is some experimental technique that sometimes can help to get upper bounds very fast. It usually does not find a solution.

Defective edge coloring   Very often, it is very hard to understand what is the actual algorithm that we get using this technique. We need to start from the 0-rounds solvable problem that we got, and go backwards and understand what is going on. There are some exceptions though. Sometimes we get very easy and a bit surprising algorithms. Consider the following problem: color the edges with 3 colors, such that each node has at most $\lceil \frac{2}{3}\Delta \rceil$ edges colored with the same color. Let us consider the easiest possible case where $\Delta = 3$. Each color must be used at most twice. This problem can be encoded as follows:

A	B	C
A	A	BC
B	B	AC
C	C	AB

A	A
B	B
C	C

That is, edges have 3 possible colors, and we either use 3 different colors on each node, or we use one color twice and then a different one. Let us feed this problem to the round eliminator, and perform two steps of round elimination. We get the following:

Close

The problem is zero rounds solvable.

Active (Before Renaming)
Any choice satisfies previous Passive

Passive (Before Renaming)
Exists choice satisfying previous Active

Renaming
Old and new labels

Active			**Passive**		**Diagram**	**Tools**
Any choice satisfies previous Passive			Exists choice satisfying previous Active		Strength of right side labels	Speedup, edit, simplifications, ...

G	F	A	ACEG	ACEG
G	E	B	BCFG	BCFG
G	D	C	DEFG	DEFG
F	E	C		

Tools

Speedup   Edit

Simplifications

Harden v1

Harden v2

Automatic Lower Bound

Automatic Upper Bound

New Renaming

The round eliminator is telling us that this new problem is 0-rounds solvable, meaning that the original one is 2 rounds solvable. Actually, since we are in the non bipartite setting, we are counting *half* rounds, instead of full rounds. This means that we actually got a 1 round algorithm.

How to get an algorithm out of this? The first thing that we need to do is to identify which is the output that gives 0-rounds solvability. We can notice that there is a line, $\boxed{\textbf{F E C}}$, that satisfies the following: any pair of labels of this line is allowed in the edge constraint. For example, $\boxed{\textbf{F F}}$ appears on the edge constraint on the second line, while $\boxed{\textbf{E C}}$ appears on the first line. This means that if all nodes output $\boxed{\textbf{F E C}}$, then we have a valid solution. By clicking "Renaming" we can see what these labels correspond to:

Renaming
Old and new labels

Ⓐ	A
Ⓑ	B
A̲B̲	C
Ⓒ	D
A̲C̲	E
B̲C̲	F
A̲B̲C̲	G

So what are nodes doing? They are just writing three different sets, $\{\mathbf{A}, \mathbf{B}\}$, $\{\mathbf{A}, \mathbf{C}\}$, and $\{\mathbf{B}, \mathbf{C}\}$, on their 3 incident edges, in some arbitrary order. As we have seen when we manually computed the round elimination result on the BSO problem, these sets must satisfy a forall quantifier. This means that no matter what nodes write on an edge, the edge should be able to pick something that is valid. Let us see what is allowed on the previous step:

Easy, just the three colorings are allowed. So why are these sets good? Well, no matter how we choose the sets, their intersection is not empty! So we can always pick a common element and

36

decide that it is the color of the edge. For example, given the sets $\{\mathbf{A}, \mathbf{B}\}$ and $\{\mathbf{A}, \mathbf{C}\}$ we can pick the element $\mathbf{A}$ that is in both. The only remaining thing to see is why this is good for the original problem. Nodes must have at most 2 edges with the same color, and since they wrote 3 different sets on the edges, it is not possible to pick the same color from all of them!

Thus, the algorithm is the following. Each node sends the set $\{A, B\}$ to one arbitrary neighbor, $\{B, C\}$ to another, and $\{A, C\}$ to the remaining one. Nodes will then receive the sets sent from the neighbors, and for each edge they will pick the smallest element in the intersection between the set sent on that edge and the set received on the same edge. This algorithm requires just one round of communication, and by generalizing it to larger values of Δ we get an algorithm that solves the desired problem.

6 Extra

Input coloring Sometimes we are interested in the following question: what is the complexity of a problem if we are given a $\Delta + 1$ coloring as input? The round eliminator gives some hints about that, if used in the case where black nodes represent edges (that is, when the black constraint has degree $\delta = 2$). For example, starting from the MIS problem, we can see that the round eliminator informs us that this problem is 0-rounds solvable, if a 2 coloring is given:

No surprise here, this problem is easy on 2 colored graphs. Let us press "Speedup" 4 times. We get the following:

Close

The problem is NOT zero rounds solvable. It can be solved in zero rounds given a 3 coloring.

Active (Before Renaming)	**Passive (Before Renaming)**	**Renaming**
Any choice satisfies previous Passive	Exists choice satisfying previous Active	Old and new labels

Active			**Passive**		**Diagram**	**Tools**
Any choice satisfies previous Passive			Exists choice satisfying previous Active		Strength of right side labels	Speedup, edit, simplifications, ...

Active			**Passive**	
Q	A	A	BCDEFGHIJKLMNOPQRS	Q
S	E	B	GHIJKLMNOPQRS	FKNPQS
P	C	C	ACDEFHIJKLMNOPQRS	RS
N	D	D	DEFIJKLMNOPQRS	OPQRS
K	E	E	LMNOPQRS	EFJKMNOPQRS
F	F	F		
R	R	G		
O	H	H		
M	I	I		
J	J	J		
L	L	L		

Tools: Speedup, edit, simplifications, ...

Speedup | Edit

Simplifications

Harden v1

Harden v2

Automatic Lower Bound

Automatic Upper Bound

New Renaming

In this case the round eliminator tells us that the problem that is 4 (half) rounds easier than MIS does not need a 2 coloring to be solved in 0 rounds, but that actually 3 colors are enough.

Also, in the automatic lower and upper bound features, we can specify how many colors we assume are given, and this influences the 0-rounds solvability.

More labels Sometimes, if the result of the round elimination technique gives a problem with too many labels, the round eliminator program gives an error.

Active		**Passi**			**Tools**
Any choice satisfies previous Passive		Exists choice satisfyin			Speedup, edit, simplifications, ...

The currently configured limit for delta*labels is 64, but in order to represent the result of this speedup a limit of 107*3 is required.

Close

J	B	A	DEFGHIJ	BC

Speedup | Edit

I	D	A	GHIJ	DEFJ	ACEH

What if we are actually interested in seeing this big result? In this case the error says that we need a limit of $107 \cdot 3$, that is the number of labels of the new problem times Δ, and that the current limit is 64. Thus, we need a limit of at least 321. To solve the issue just compute $k = \lceil 321/64 \rceil = 6$. Then, open in your browser https://users.aalto.fi/~olivetd1/round-eliminator6/ (replace the last number in the URL with your value of k, and values in $1, \ldots, 10$ are allowed). This is a different version of the round eliminator, compiled with a different limit. The downside of using this version is that every operation will be roughly 6 times *slower*.

Constraints that depend on inputs Sometimes we want to encode constraints such as "nodes that have as input a 1 must do something, while nodes that have as input a 0 must do something else". It is actually possible to apply the round elimination technique also in this case! The program does not support this kind of constraints at the moment, but there is a plan to add this feature at some point.

7   Acknowledgments

References

[1] Sebastian Brandt. An Automatic Speedup Theorem for Distributed Problems. In *Proc. 38th ACM Symposium on Principles of Distributed Computing (PODC 2019)*, 2019.

[2] Alkida Balliu, Sebastian Brandt, Yuval Efron, Juho Hirvonen, Yannic Maus, Dennis Olivetti, and Jukka Suomela. Classification of distributed binary labeling problems. *CoRR*, abs/1911.13294, 2019.

[3] Alkida Balliu, Sebastian Brandt, Juho Hirvonen, Dennis Olivetti, Mikaël Rabie, and Jukka Suomela. Lower bounds for maximal matchings and maximal independent sets. In *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 481–497, 2019.

[4] Sebastian Brandt and Dennis Olivetti. Truly tight-in-Δ bounds for bipartite maximal matching and variants. *CoRR*, abs/2002.08216, 2020.