CSE291-14: The Number Field Sieve

https://cseweb.ucsd.edu/classes/wi22/cse291-14

Emmanuel Thomé

February 10, 2022

Part 6a

The filtering step

Plan

The filtering step

Duplicates

Singletons and “clique” removal

Merge

Practical aspects

Plan

Once we have collected all relations

The output of the relation collection can be viewed as a matrix

- a row corresponds to a relation;
- a column corresponds to a prime ideal;
- the coefficient is the valuation of the corresponding ideal in the corresponding relation.

Orders of magnitude

This is the dataset that comes directly from the relation collection.

	what?				
year		bits	dd	rows	columns
2010	RSA	768	232	64e9	35e9
2017	DLP	768	232	9.1e9	?
2019	RSA	795	240	8.9e9	2.4e9
2019	DLP	795	240	3.8e9	1.0e9
2020	RSA	829	250	8.7e9	6.5e9

Note: vast difference in the numbers comes from very different parameter choices.
The dataset weighs between hundreds of GB and several TB.

Sparsity

Another trait of the dataset.

Relations (rows) are VERY sparse.

Typical: 20 to 25 non-zero entries per row on average.

Rows and columns

There is no connection between rows and columns in this matrix.

We may sort rows and columns in any way we like, provided that we keep track of the permutation.

This is in contrast with other linear algebra problems for which the notion of "diagonal element" is meaningful. Here it is not.

Plan

This is already linear algebra

We need to do some linear algebra

- In the factoring case, the matrix is defined over $\mathbb{F}_2$.
  We are looking for several left nullspace elements.
- In the discrete logarithm case (later!), the matrix is defined over $\mathbb{Z}/\ell\mathbb{Z}$ for some large prime number ℓ.
  We are looking for a right kernel vector.

The filtering step is the beginning of the linear algebra step.
(but they are often regarded as separate steps.)

Domain of the matrix coefficients

Factoring usage

Since we ultimately work modulo 2, all coefficients are either 0 or 1.

Discrete logarithm usage

Coefficients can be larger integers a priori, BUT the vast majority are ± 1. We can really assume that this is always the case.

Goal of the filtering step

The filtering step is a preprocessing step, which aims at reducing the matrix size.

Another appropriate term can be that filtering is actually creating some preconditioner.

Preprocessing operations

We want to apply a chain of elementary matrix transformations

$$M \to M'$$

such that a solution to the linear system $xM' = 0$ (or $M'x = 0$) leads to a solution of the linear system we started with.

Transformations

We can do the following things:

- Sort rows and columns.
- Remove duplicate rows.
- We will also see how, under certain conditions, we can:
  - Take out some rows or columns.
  - Replace rows by linear combinations (as is done in Gaussian elimination).

Old terminology

Until the late 1990s, the term structured Gaussian elimination was often used to refer to what is now known as filtering.

The filtering step

- The filtering step modifies the matrix built from the relations.
- It outputs a new matrix that is used as input by the linear algebra computation that comes next.
- The "quality" of the matrix produced by the filtering step impacts the time spent in the linear algebra step.

Quality metrics

- We want the new matrix to be much smaller
- We want to maintain sparsity.
- Ultimately we want to minimize the cost of the subsequent linear algebra step.

The filtering step is memory-bound and I/O-bound. CPU-wise, no serious computation is done during filtering.

Plan

Definitions

Definition (Excess)

The excess of set of relations (resp. a matrix) is the difference between the number of relations (resp. the number of rows) and the number of prime ideals (resp. the number of columns).

Definitions

Definition (Weight)

- The weight of a column (resp. a row) in a matrix is the number of non-zero elements in this column (resp. this row).
- The weight of an ideal (resp. a relation) is the weight of the associated column (resp. row).
- The total weight of a matrix is the number of non-zero elements of this matrix.

Definitions

Definition (Density)

- The density of a column (resp. a row) in its weight divided by its size.
- The density of a matrix is its average row density.

Inside the filtering step

The filtering step is split in 4 stages:

- duplicate removal: very easy, uses hash tables to remove relations that appear more than once;
- singleton removal: remove useless rows and columns;
- "clique" removal: use the excess to reduce the size of the matrix;
- merge: beginning of a Gaussian elimination.

Plan

The filtering step

Duplicates

Singletons and “clique” removal

Merge

Practical aspects

Plan

- Duplicates
  - Analysis of the cause
  - Dealing with it

Why do we have duplicates?

Duplicate relations are completely identical rows.

This can only happen if the corresponding pairs (a, b) and (a', b') are such that:

- $(a - b\alpha) = (a' - b'\alpha) \times$ a unit in $\mathcal{O}_K$.
- and the same thing on the other side, which probably means $a = \pm a'$ and $b = \pm b'$ (if we have a rational side).

In most configurations, duplicate relations mean identical (a, b)'s.

Can we really have identical (a, b)'s?

If we do only line sieving: we can't.

- By design, we only process an (a, b) pair once.
- For a large scale computation, accidentally sieving the same sub-range twice may happen, though.

For special-$\mathfrak{q}$ sieving, the situation is a bit different, since a given (a, b) pair may belong to two distinct lattices $\mathcal{L}_{\mathfrak{q}}$ and $\mathcal{L}_{\mathfrak{q}'}$.

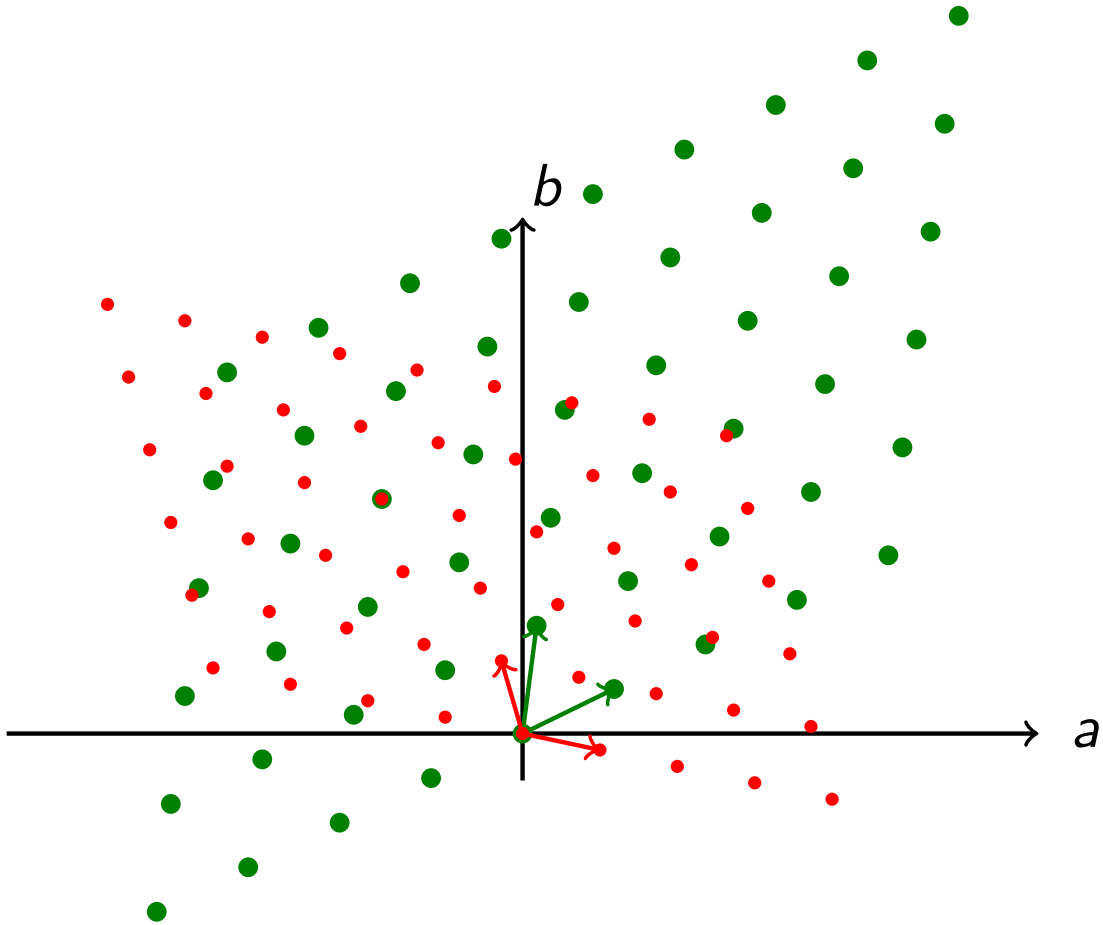
Points in two different lattices

If (a, b) is in two different lattices $\mathcal{L}_{\mathfrak{q}}$ and $\mathcal{L}_{\mathfrak{q}'}$, then the factorization of $\langle a - b\alpha \rangle \times J$ involves both $\mathfrak{q}$ and $\mathfrak{q}'$.

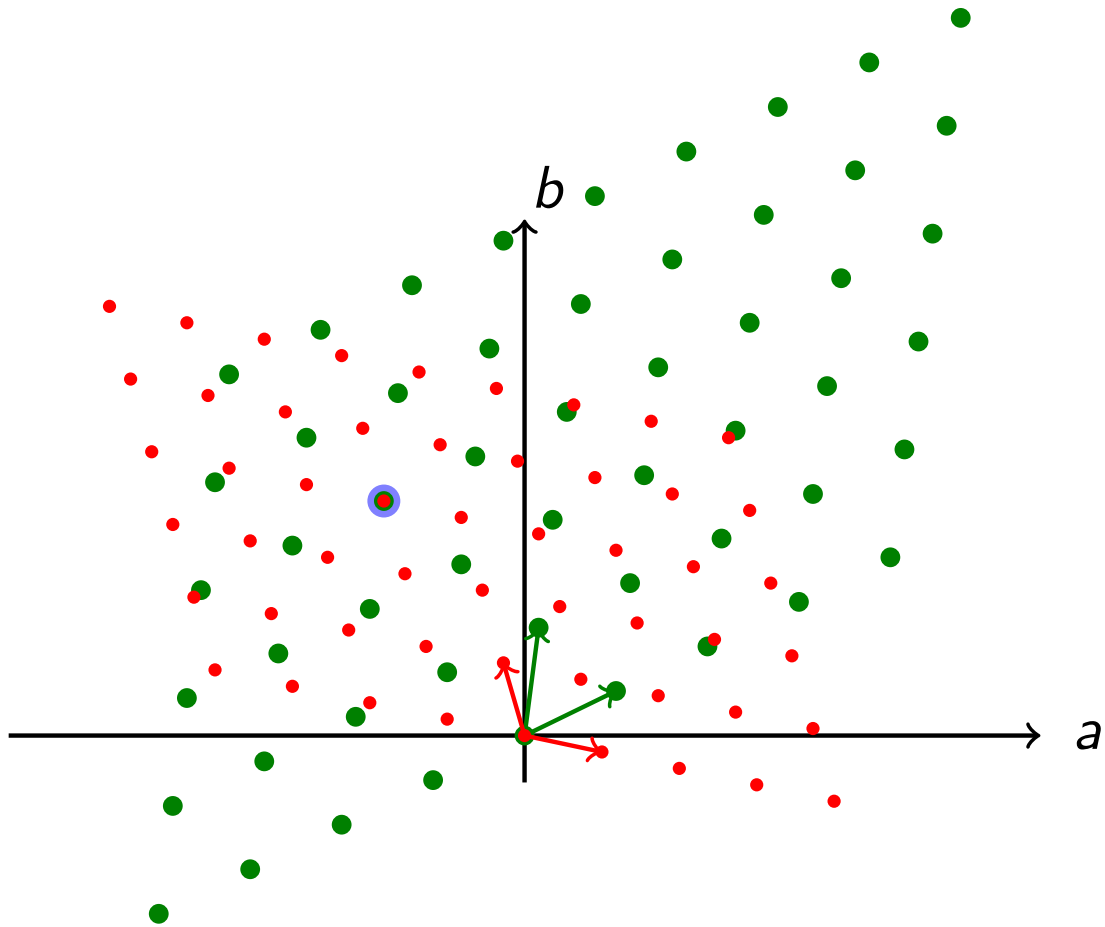
An approach that does not work to avoid duplicates:

If two (or more) prime ideals from the special-$\mathfrak{q}$ range appear in the factorization of $\langle a - b\alpha \rangle \times J$, keep the relation only if the current special-$\mathfrak{q}$ is the largest of the two (or more).

Intersections of two lattices

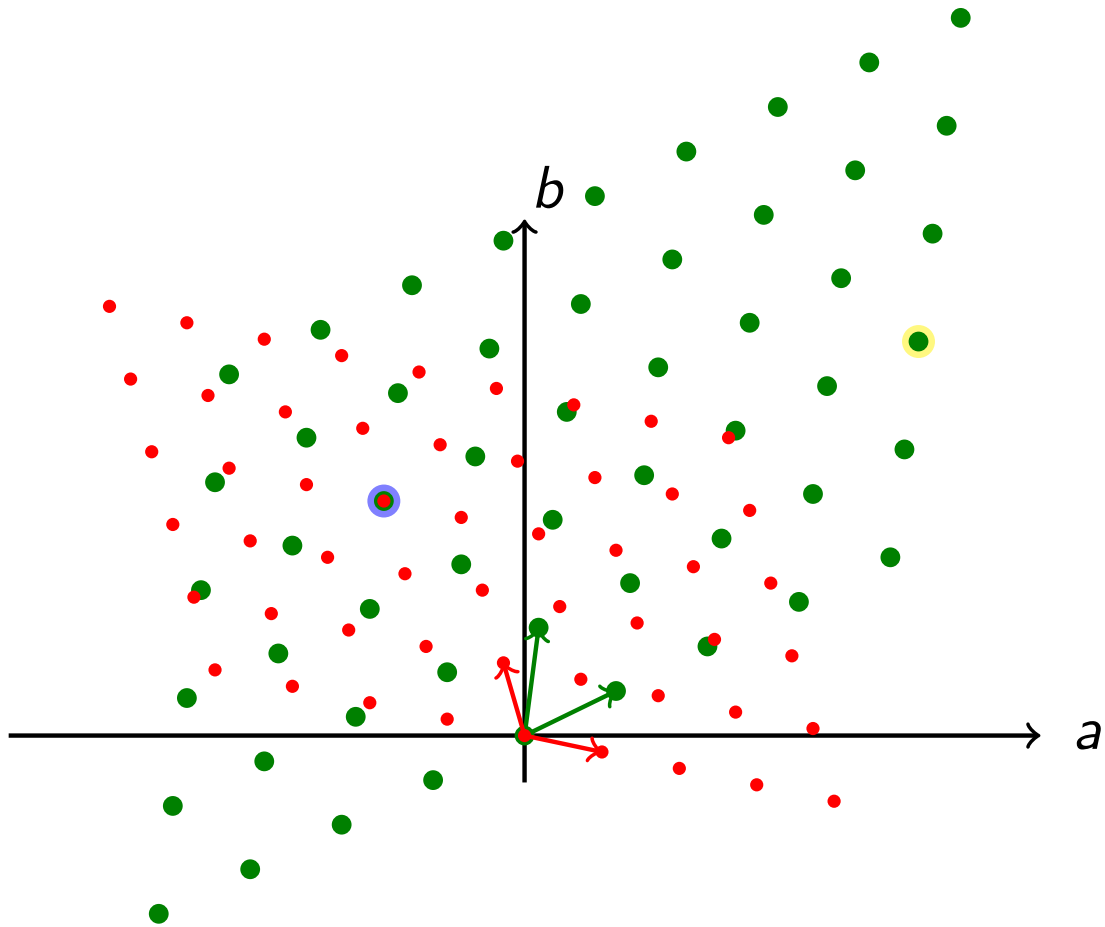


Intersections of two lattices



Point ● is in $\mathcal{L}_{\mathfrak{q}} \cap \mathcal{L}_{\mathfrak{q}'}$ and is reached twice.

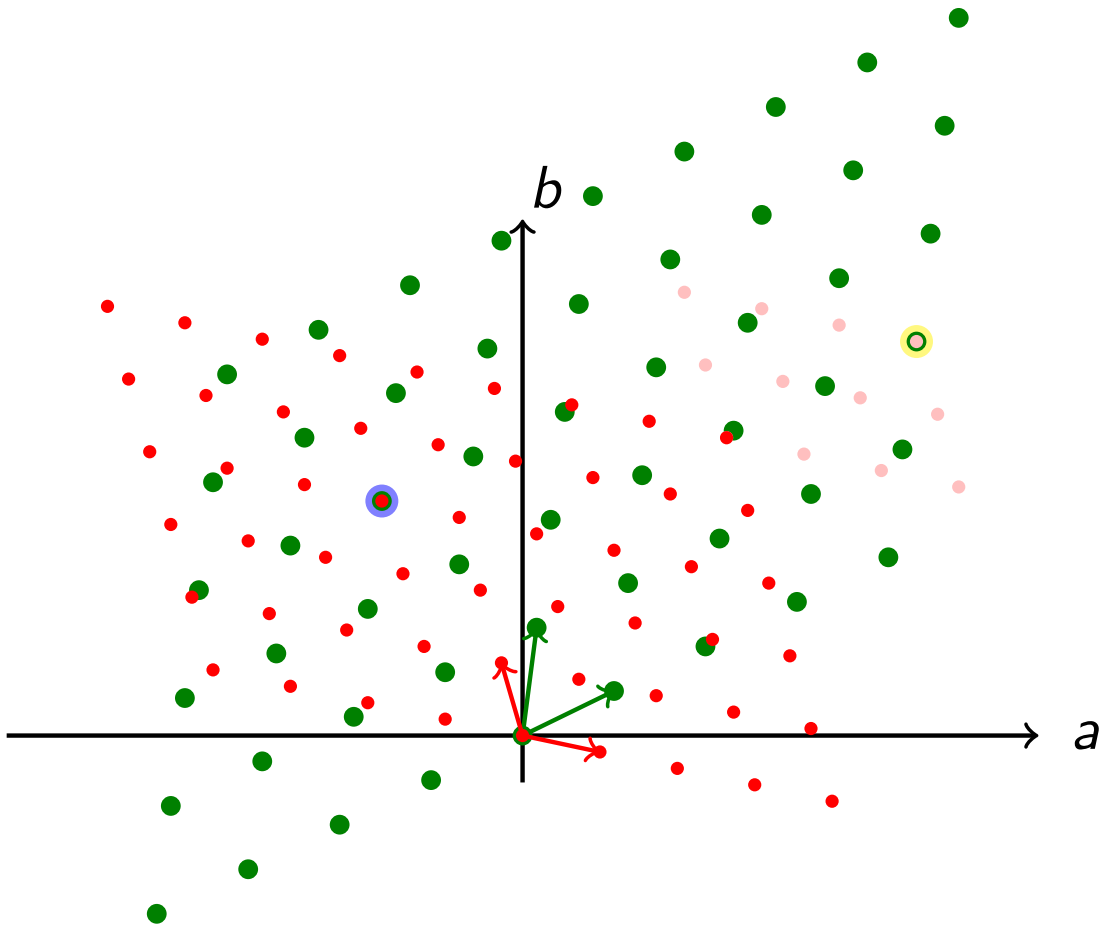
Intersections of two lattices



Point ● is in $\mathcal{L}_{\mathfrak{q}} \cap \mathcal{L}_{\mathfrak{q}'}$ and is reached twice.

Point ● is in $\mathcal{L}_{\mathfrak{q}} \cap \mathcal{L}_{\mathfrak{q}'}$ and is reached only once.

Intersections of two lattices



Point ● is in $\mathcal{L}_q \cap \mathcal{L}_{q'}$ and is reached twice.
Point ● is in $\mathcal{L}_q \cap \mathcal{L}_{q'}$ and is reached only once.

Avoiding duplicates is not trivial

On-the-fly duplicate removal

Trying to avoid duplicates altogether with special-$\mathfrak{q}$ sieving is a more subtle task than one may think.

It is possible to do it, though, with moderate overhead cost. Cado-NFS has this functionality.

Some factors affect the number of duplicates, such as the size of the special-$\mathfrak{q}$ range.

Good rule of thumb: keep $q_{\max}/q_{\min}$ under control (say 2).

Plan

- Duplicates
  - Analysis of the cause
  - Dealing with it

Dealing with duplicates

Good news: dealing with duplicates is very easy.

- Only (a, b) matters.
- We can use a hash table. But a hash table with several billion entries could require some thought.

Cado-NFS uses a few very basic techniques:

- Hash (a, b),
- split the input according to $H(a, b) \bmod K$ (on disk),
- de-duplicate each subset.

Very simple-minded and strongly I/O-bound, but does the job.

Plan

The filtering step

Duplicates

Singletons and “clique” removal

Merge

Practical aspects

Singleton removal

- A singleton is a column of weight 1.
- The removal of a singleton is the removal of the column and of the row corresponding to this single non-zero coefficient.
- Example:

$$\begin{pmatrix} 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

- Removing a singleton reduces the total weight of the matrix but cannot reduce the excess.

Singleton removal

- A singleton is a column of weight 1.
- The removal of a singleton is the removal of the column and of the row corresponding to this single non-zero coefficient.
- Example:

$$\begin{pmatrix} 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

- Removing a singleton reduces the total weight of the matrix but cannot reduce the excess.

Singleton removal

- A singleton is a column of weight 1.
- The removal of a singleton is the removal of the column and of the row corresponding to this single non-zero coefficient.
- Example:

$$\left(\begin{array}{c|ccccc} 0 & 1 & 1 & 0 & 1 & 1 \\ \hline 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right)$$

- Removing a singleton reduces the total weight of the matrix but cannot reduce the excess.

Singleton removal

- A singleton is a column of weight 1.
- The removal of a singleton is the removal of the column and of the row corresponding to this single non-zero coefficient.
- Example:

$$\left(\begin{array}{cc|cccc} 0 & 1 & 1 & 0 & 1 & 1 \\ \hline 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right)$$

- Removing a singleton reduces the total weight of the matrix but cannot reduce the excess.

Singleton removal

- A singleton is a column of weight 1.
- The removal of a singleton is the removal of the column and of the row corresponding to this single non-zero coefficient.
- Example:

$$\left(\begin{array}{cccccc}
\cancel{0} & \cancel{1} & \cancel{1} & \cancel{0} & \cancel{1} & \cancel{1} \\
\cancel{1} & \cancel{1} & \cancel{0} & \cancel{1} & \cancel{0} & \cancel{1} \\
0 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 1
\end{array}\right)$$

- Removing a singleton reduces the total weight of the matrix but cannot reduce the excess.

Singleton removal

Why can we remove singletons?

Any solution to $xM' = 0$ or $M'x = 0$ immediately leads to a solution for the original matrix M.

Note: this "leads to" part may require us to memorize the deleted row. (This is only for DLP.)

Singleton removal

- Only need to know if a coefficient is non-zero or not, not the actual value.
- For each row, we store the indices of the columns with non-zero coefficients (but not the value of the coefficients).
- We also store and maintain the weight of each column.
- Removing a singleton can create other singletons, it may be necessary to loop through the matrix more than once.

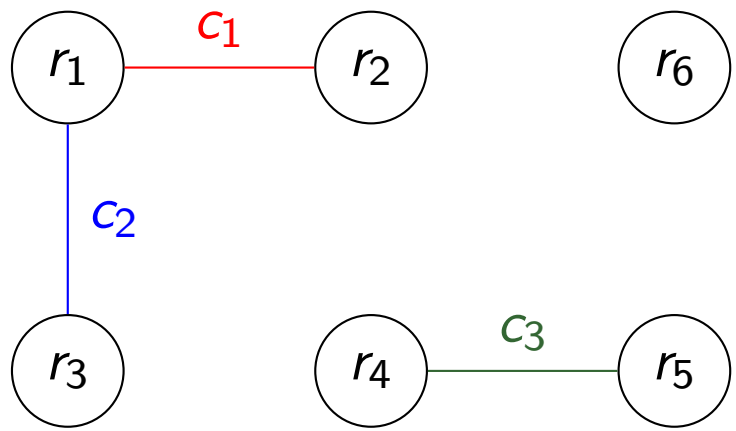
"Clique" removal

- While the excess is larger that what is needed, it is possible to remove some rows. (we may need to memorize them, though.)
- We can choose which row is deleted, how do we choose?
- Put otherwise: what is the best possible use of the excess that we have?

"Clique" removal

- Remark: if a row containing a column of weight 2 is removed, this column becomes a singleton and can be removed.
- A "clique" is a connected component of the graph where the nodes are the rows and the edges are the columns of weight 2.
- It has NOTHING TO DO with cliques in graph theory.
- If any row in a "clique" is removed, the creation of singletons leads to the removal of all rows of the "cliques".
- Example:

$$\begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



"Clique" removal

- When removing a "clique", one more row than column is removed, so the excess is reduced by 1. And the total weight of the matrix is reduced.
- The "clique" removal algorithm is quite simple:

  while the excess allows it, remove a "clique".
- How do we choose the "clique" to remove?
  Multiple cost functions have been explored over the years.
  Current wisdom is mainly based on empirical evidence.
- In order to compute and remove "cliques", we only need to know if a coefficient is non-zero or not, not the actual value.

The purge binary in Cado-NFS

- In Cado-NFS, the singleton and "clique" removal are done by the purge binary.
- Algorithm:
  - Input: the relations and a target excess.
  - Output: the remaining relations (and separately, the deleted relations if needed).
  - 1. Remove all singletons.
  - 2. Remove some "cliques".
  - 3. Go to 1 if the excess is larger than the target value.
- The purge binary is the same for factorization and discrete logarithm computations.

Plan

The filtering step

Duplicates

Singletons and “clique” removal

Merge

Practical aspects

Merge

- Merge is the beginning of a Gaussian elimination: combinations of rows are performed to create singletons that are then removed.

Definition (k-merge)

Let $k \geq 2$ be an integer. Let C be a column of weight k and $r_1, \ldots, r_k$ the k rows corresponding to the k non-zero coefficients of C. A k-merge is a way to perform successive rows additions of the form $r_i \leftarrow \alpha_{ij} r_i + \beta_{ij} r_j$, with $i \neq j$ and $1 \leq i, j \leq k$, such that the column C becomes a singleton, which will be deleted.

- For factorization, $\alpha_{ij} = \beta_{ij} = 1$; for discrete logarithm, they are usually very small (almost always ± 1).
- Merge does not change the right kernel (if we memorize the removed rows) nor the left kernel.

Merge – example

Example of a 2-merge and a 3-merge in the case of factorization.

$$\begin{pmatrix} 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{pmatrix}$$

Merge – example

Example of a 2-merge and a 3-merge in the case of factorization.

$$\begin{pmatrix} 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & \textcircled{1} \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & \textcircled{1} \\ 1 & 1 & 1 & 1 & 0 \end{pmatrix}$$

Merge – example

Example of a 2-merge and a 3-merge in the case of factorization.

$$\begin{pmatrix} 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{pmatrix}$$

Merge – example

Example of a 2-merge and a 3-merge in the case of factorization.

$$\left(\begin{array}{ccccc} 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ \hline 1 & 1 & 0 & 1 & 1 \\ \hline 1 & 1 & 1 & 1 & 0 \end{array}\right)$$

Merge – example

Example of a 2-merge and a 3-merge in the case of factorization.

$$\left(\begin{array}{cccc|c} \textcircled{1} & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ \textcircled{1} & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ \hline 1 & 1 & 0 & 1 & 1 \\ \hline \textcircled{1} & 1 & 1 & 1 & 0 \end{array}\right)$$

Merge – example

Example of a 2-merge and a 3-merge in the case of factorization.

$$\left(\begin{array}{ccccc}
1 & 1 & 0 & 1 & 0 \\
0 & 1 & 1 & 1 & 0 \\
0 & 0 & 1 & 1 & 0 \\
0 & 1 & 1 & 1 & 0 \\
\hline
1 & 1 & 0 & 1 & 1 \\
\hline
0 & 0 & 1 & 0 & 0
\end{array}\right)$$

Merge – example

Example of a 2-merge and a 3-merge in the case of factorization.

$$\left(\begin{array}{ccccc}
1 & 1 & 0 & 1 & 0 \\
\hline
0 & 1 & 1 & 1 & 0 \\
0 & 0 & 1 & 1 & 0 \\
0 & 1 & 1 & 1 & 0 \\
\hline
1 & 1 & 0 & 1 & 1 \\
\hline
0 & 0 & 1 & 0 & 0
\end{array}\right)$$

Merge – how do we do it?

- For a k-merge with $k \geq 3$, there is more than one way to perform the additions to create a singleton.
- We want to minimize the fill-in of the matrix, i.e., we want to minimize the increase of the total weight of the matrix.

Merge – simple approach

Simple approach:

- Select the row with smallest weight among our set of k rows.
- Subtract an appropriate combination of this row to the other rows.

The Markowitz rule

If the lightest row among k has weight w, the incurred fill-in (difference in total matrix weight) is:

$$\begin{aligned}(k-1)(w-1) - (k-1) - w \\ = (k-1)(w-2) - w + 2 - 2 \\ = (k-2)(w-2) - 2\end{aligned}$$

Merge – minimal spanning tree

The simple approach may be refined:

- In order to find the way of performing a k-merge that minimize the fill-in, we compute a minimal spanning tree.
- The minimal spanning tree is computed on the graph where:
  - the nodes are the k rows involved in the k-merge;
  - the weight of the edge between two rows is the weight of the sum of these two rows.
- This minimal fill-in may be below the Markowitz estimate.

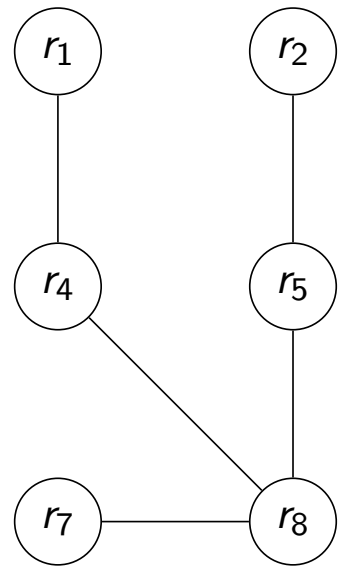
This approach matches the simple one if the MST happens to be a star (with arm length 1).

Merge – example

Example of the use of a minimal spanning tree for a 6-merge in the case of factorization.

$$\begin{pmatrix} 0 & 1 & 1 & 0 & 1 & \textcircled{1} \\ 1 & 1 & 0 & 1 & 0 & \textcircled{1} \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & \textcircled{1} \\ 0 & 0 & 0 & 1 & 0 & \textcircled{1} \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & \textcircled{1} \\ 0 & 0 & 0 & 0 & 0 & \textcircled{1} \end{pmatrix}$$

	r_2	r_4	r_5	r_7	r_8
r_1	4	2	4	2	3
r_2		4	2	4	3
r_4			2	2	1
r_5				2	1
r_7					1



Merge – implementation

- All possible k-merges (for $2 \leq k \leq k_{\max}$) are in a priority queue sorted (in ascending order) by the fill-in implied by the merges.
- Each time a merge is performed, the matrix changes and the minimal spanning trees of the other merges can change.
- Unfortunately, it is too costly to recompute all minimal spanning trees after each change. We approximate the fill-in with Markowitz count.

$$(k-2)(w_{\min}-2)-2$$

Merge – implementation

- Merge reduces the size of the matrix but increases the total weight of the matrix:
- Performing a k-merge reduces the number of rows and columns of the matrix by 1.
- The excess is not modified.
- A 2-merge reduce the total weight of the matrix by at least 2.
- But, in general, performing a k-merge, with $k \geq 3$, increases the total weight of the matrix.

- In merge, the values of the non-zero coefficients matter.

Merge – stopping criteria

- `merge` must be stopped while the matrix is still sparse.
- A possible criteria is to use an estimate of the time of the linear algebra step, based on information such as:
  - The current matrix size. (which $\searrow$ as time goes).
  - The current total matrix weight. (which $\nearrow$ as time goes).
  - Finer grained data can also be used.
- We can stop `merge` once the estimate that is used starts to increase.
- In practice, `merge` is performed until a given average weight per row is reached.

The binaries for merge in Cado-NFS

- The merge stage is split in two binaries in Cado-NFS.
- For factorization:
  - `merge`
  - `replay`
- For discrete logarithm:
  - `merge-dl`
  - `replay-dl`
- The binaries `replay` and `replay-dl` build the matrix for the linear algebra step from the results of the merge algorithm.

Plan

The filtering step

Duplicates

Singletons and “clique” removal

Merge

Practical aspects

Oversieving

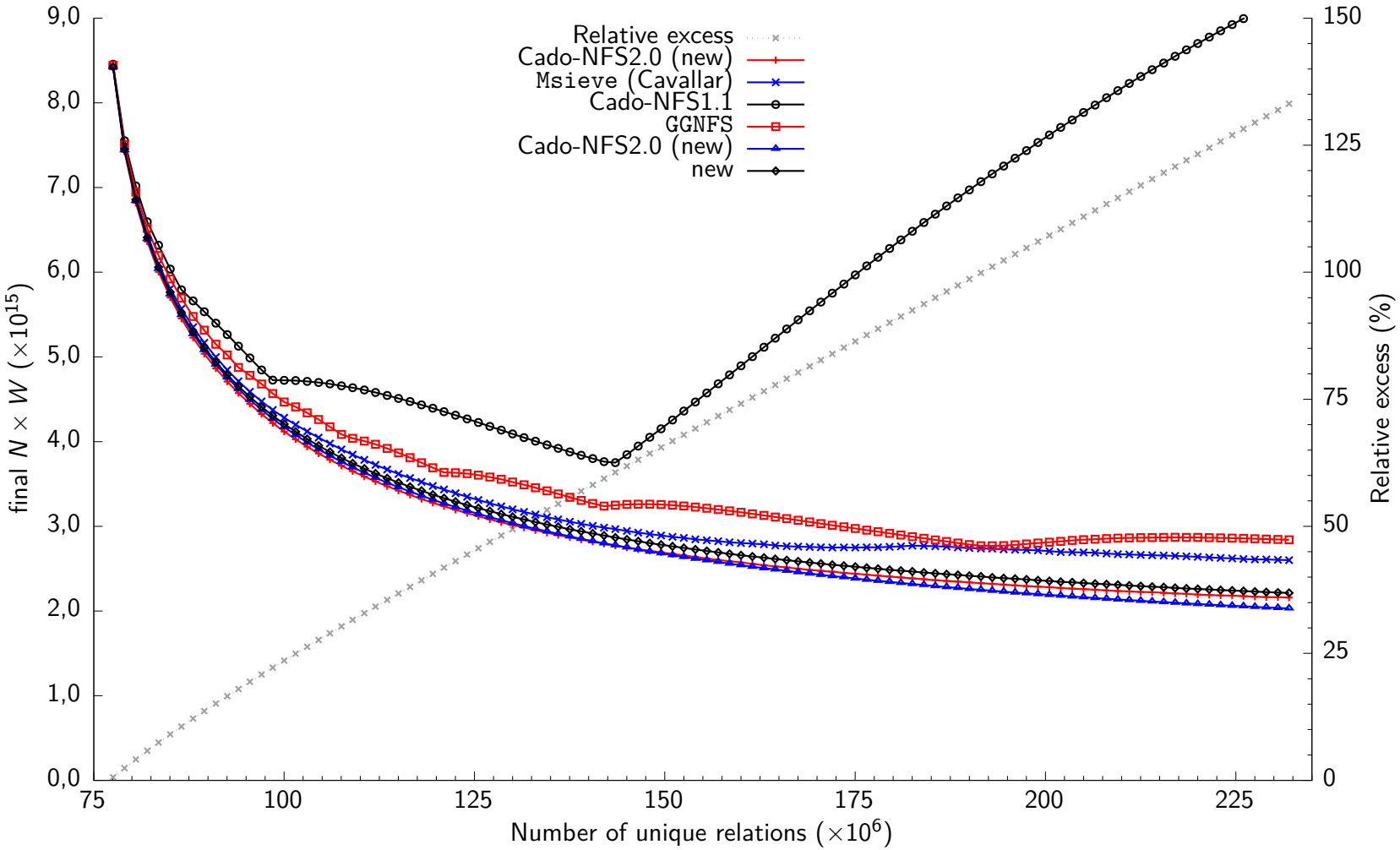
- By computing more relations in the sieving step, we can increase the excess at the beginning of `purge`.
- So more "cliques" can be removed.
- If the weight function used in the "clique" removal algorithm is well-designed, the filtering step should produce a better matrix.
- Trade-off: between the additional time spent in the sieving step and the gain in the linear algebra step.
- The additional time spent in the filtering step due to the additional relations is negligible compared to the time spent in the other steps.

Oversieving – experiment

- Illustrate oversieving with RSA-155.
- We computed a lot more relations that is needed.
- We performed the filtering step with different weight functions and increasing sets of relations.

- To compare results, we used $N \times W$, where
  - N = number of rows of the final matrix
  - W = total weight of the final matrix

Oversieving – results



Oversieving lessons

State-of-the-art implementations have a very flat curve that gives the linear algebra cost estimate as a function of the stopping point.

This explains why the stopping point of the `merge` process is approached in a fairly relaxed way.

- Aim at a ballpark estimate of an average row density ≈ 200.
- See where this brings us.
- Possibly look at what happens for 150, 175, 225, 250.
- It probably makes little difference in the end.

Some experimental data

year	what?	bits	dd	relations	unique	singl	merged
2010	RSA	768	232	64e9	48e9	2.5e9	192e6
2017	DLP	768	232	11e9	9.1e9	?	23.5e6
2019	RSA	795	240	8.9e9	6.0e9	1.2e9	282e6
2019	DLP	795	240	3.8e9	2.4e9	150e6	36e6
2020	RSA	829	250	8.7e9	6.1e9	1.8e9	405e6

Memory footprint

This all depends on the choice of parameters, but overall the `purge` step (singleton and "clique" removal) has the largest memory requirement.

- Input and output data sets are bulky.
- Need to keep lots of info in RAM.
- RSA-829: 1.4TB RAM needed (this has significantly improved since).

The `merge` step generally has a slightly smaller memory footprint. RSA-829: 450GB.

Parallelizing

Recent work has proven that `merge` can be efficiently parallelized, with no noticeable impact on the output quality.

- This is still single-machine parallelization. Multi-machine parallelization attempts have been unsuccessful thus far.
- The `merge` process remains strongly I/O-bound.

Simulation of the filter step

How can we predict the size of the matrix that we obtain after the filter step?

Simulation of the filter step

How can we predict the size of the matrix that we obtain after the filter step?

- Approach 1: create fake relations up to the expected required number, and see what we get after duplicates/singleton/clique/merge.
  - Data processing cost is identical to the real computation.
  - May be hard to use if we want to explore many different parameter sets.

Simulation of the filter step

How can we predict the size of the matrix that we obtain after the filter step?

- Approach 1: create fake relations up to the expected required number, and see what we get after duplicates/singleton/clique/merge.
  - Data processing cost is identical to the real computation.
  - May be hard to use if we want to explore many different parameter sets.
- Approach 2: do the same on a reduced-scale model, say 10- or 100-fold.
  - This works surprisingly well.
  - Still WIP in Cado-NFS, but results are very promising.