Canonical Generation of Matroids

(from ancient times of matroid computing to the present)

Petr Hliněný*

Faculty of Informatics, Masaryk University

Brno, Czech Republic

1 Bit of Personal History

My getting to *mathematical computing*:

- Early 90's – some tries to generate snarks on a computer; that was when I had first met Brendan's work and `nauty`.

1 Bit of Personal History

My getting to *mathematical computing*:

- Early 90's – some tries to generate snarks on a computer;
  that was when I had first met Brendan's work and `nauty`.
- Late 90's – Negami's planar cover conjecture;
  unsuccessful tries to get a computer-assisted discharging argument, and
  a very successful generation of all possible counterexamples to it.

1 Bit of Personal History

My getting to *mathematical computing*:

- Early 90's – some tries to generate snarks on a computer;
  that was when I had first met Brendan's work and `nauty`.
- Late 90's – Negami's planar cover conjecture;
  unsuccessful tries to get a computer-assisted discharging argument, and a very successful generation of all possible counterexamples to it.
- Since 2000 – MACEK, motivated by Geoff's questions and suggestions.
  Unfortunately, its development not touched since 2006.

1 Bit of Personal History

My getting to *mathematical computing*:

- Early 90's – some tries to generate snarks on a computer;
  that was when I had first met Brendan's work and `nauty`.
- Late 90's – Negami's planar cover conjecture;
  unsuccessful tries to get a computer-assisted discharging argument, and a very successful generation of all possible counterexamples to it.
- Since 2000 – MACEK, motivated by Geoff's questions and suggestions.
  Unfortunately, its development not touched since 2006.
- And nowadays – am I too old for programming? Or too lazy?
  Perhaps, and so I leave the coding work to my students...

1 Bit of Personal History

My getting to *mathematical computing*:

- Early 90's – some tries to generate snarks on a computer;
  that was when I had first met Brendan's work and `nauty`.
- Late 90's – Negami's planar cover conjecture;
  unsuccessful tries to get a computer-assisted discharging argument, and a very successful generation of all possible counterexamples to it.
- Since 2000 – MACEK, motivated by Geoff's questions and suggestions.
  Unfortunately, its development not touched since 2006.
- And nowadays – am I too old for programming? Or too lazy?
  Perhaps, and so I leave the coding work to my students. . .
  For instance;
  - generating all posible nonprojective graphs with planar emulators,
  - and computing good heuristic partitioned branch-decompositions of really huge graphs (e.g. the TIGER/Line road maps of USA).

2 MACEK and Matroid Generation

– MAtroids Computed Efficiently toolKit.

- A system developed under influence of Geoff at VUW since 2000.
- Intended to help with tiresome small case-checking in matroid theory.

2 MACEK and Matroid Generation

– MAtroids Computed Efficiently toolKit.

- A system developed under influence of Geoff at VUW since 2000.
- Intended to help with tiresome small case-checking in matroid theory.
- Handling only represented matroids over small finite fields and partial fields, and richly supporting step-by-step generation of these matroids from specified base minors.

2 MACEK and Matroid Generation

– MAtroids Computed Efficiently toolKit.

- A system developed under influence of Geoff at VUW since 2000.
- Intended to help with tiresome small case-checking in matroid theory.
- Handling only represented matroids over small finite fields and partial fields, and richly supporting step-by-step generation of these matroids from specified base minors.
- Some disadvantages:
  - Nonequivalent representations must be handled each one separately, and
  - no support for abstract matroids (though isomorph. testing works).

2.1 MACEK – a practical example

– the largest *golden-mean* matroids of each rank.

Consider a question;

what are the maximum-size golden-mean matroids of each rank?

2.1 MACEK – a practical example

– the largest *golden-mean* matroids of each rank.

Consider a question;

what are the maximum-size golden-mean matroids of each rank?

- These are 3-connected, and we may use the Wheels-and-Whirls theorem to generate all of them in single-element steps.
- MACEK has been developed right for this kind of tasks...

2.1 MACEK – a practical example

– the largest *golden-mean* matroids of each rank.

Consider a question;

what are the maximum-size golden-mean matroids of each rank?

- These are 3-connected, and we may use the Wheels-and-Whirls theorem to generate all of them in single-element steps.

- MACEK has been developed right for this kind of tasks...

```
{ <Wh3 <W3 }
!represgen (S) allq
!append ((S)) "!extend cccccccccccccccc (T)"
!restart
!prtree
```

– `!restart` is a tricky way to repeat (cycle) in MACEK.

2.2 MACEK – a second practical example

– the *intertwines* of $M(K_{3,3})$ and $M(K_{3,3})^*$.

As Gordon has mentioned in his talk, another interesting task is:

Find the matroids M with both $M(K_{3,3})$ and $M(K_{3,3})^*$ as minors such that no proper minor of M has both $M(K_{3,3})$ and $M(K_{3,3})^*$ as minors.

2.2 MACEK – a second practical example

– the *intertwines* of $M(K_{3,3})$ and $M(K_{3,3})^*$.

As Gordon has mentioned in his talk, another interesting task is:

Find the matroids M with both $M(K_{3,3})$ and $M(K_{3,3})^*$ as minors such that no proper minor of M has both $M(K_{3,3})$ and $M(K_{3,3})^*$ as minors.

- Having such 3-connected M, there is a 3-connected single-element minor N of M containing $M(K_{3,3})$ but not $M(K_{3,3})^*$.

2.2 MACEK – a second practical example

– the *intertwines* of $M(K_{3,3})$ and $M(K_{3,3})^*$.

As Gordon has mentioned in his talk, another interesting task is:

Find the matroids M with both $M(K_{3,3})$ and $M(K_{3,3})^*$ as minors such that no proper minor of M has both $M(K_{3,3})$ and $M(K_{3,3})^*$ as minors.

- Having such 3-connected M, there is a 3-connected single-element minor N of M containing $M(K_{3,3})$ but not $M(K_{3,3})^*$.
  - All such potential matroids N can be generated in step-by-step 3-connected extensions from $M(K_{3,3})$ while excluding $M(K_{3,3})^*$.
  - This is very fast in MACEK.

2.2 MACEK – a second practical example

– the *intertwines* of $M(K_{3,3})$ and $M(K_{3,3})^*$.

As Gordon has mentioned in his talk, another interesting task is:

Find the matroids M with both $M(K_{3,3})$ and $M(K_{3,3})^*$ as minors such that no proper minor of M has both $M(K_{3,3})$ and $M(K_{3,3})^*$ as minors.

- Having such 3-connected M, there is a 3-connected single-element minor N of M containing $M(K_{3,3})$ but not $M(K_{3,3})^*$.
  - All such potential matroids N can be generated in step-by-step 3-connected extensions from $M(K_{3,3})$ while excluding $M(K_{3,3})^*$.
  - This is very fast in MACEK.
- The next step then adds one element to the generated N in all possible ways creating an $M(K_{3,3})^*$-minor. Let N_1 be the extended matroid.

– All single-element removals of N_1 are then checked for $M(K_{3,3})$ and $M(K_{3,3})^*$, which can validate $N_1 = M$ being an intertwine.

The initial generation multi-step

In the rather curious (or even bizzare) language of MACEK this reads:

```
!pfield GF2
!verbose
{ <grK33 }
@name itwi
@ext-forbid grK33#
!extend $param1
!mmove ((S)) >(()(S))
!prtree
!writetreeto itwi-$param1 (()(T))
```

– here `param1` controls the max size of intended N (the number of extension steps we take),

– and `!mmove` is needed to “deprive” the generated matrices of traces (the signatures) of their generating sequences.

Continuing; the one-element addition

This step is already quite slow – having to “forget” the previous generating sequence, we arrive at many duplicates.

So;

```
!quiet
!append (T) "@eraseall ext-forbid"
!extend b (()(S)) >((2)(S))
!prtree
!writetreeto itwi-$param1-b ((2)(T))
{ <grK33# }
!filt-minor ((2)(S)) ((3)(T))
!prtree
!writetreeto itwi-$param1-bm ((2)(T))
```

– all the one-element additions are tried for each potential N,

– and only those extensions having an $M(K_{3,3})^*$ are kept.

Finishing; testing an intertwine

The finishing step done just by brute force – all one-element removals are tested as follows for the presence of $M(K_{3,3})$ and $M(K_{3,3})^*$ minors.

```
!append ((2)(S)) "!remeach (T); !quiet"
!append ((2)(S)) "!mread grK33 >(()(t)); !mread grK33# >((2)(
!append ((2)(S)) "!filt-minor ((S)) (()(T))"
!append ((2)(S)) "!filt-minor ((S)) ((2)(T))"
!append ((2)(S)) "!iflist 0 = ((S)); !writeto itwi-$param1-ok
!restart
!prtree
```

3 Canonical Generation: Generating Sequences

– an idea extending Brendan's orderly generation approach.

The core: Not consider the task as just "constructing a matroid", but "*constructing a particular generating sequence*" (leading to this matroid).

3 Canonical Generation: Generating Sequences

– an idea extending Brendan's orderly generation approach.

The core: Not consider the task as just "constructing a matroid", but "*constructing a particular generating sequence*" (leading to this matroid).

- This is a better framework to capture various involved connectivity restrictions in chain and splitter theorems.

3 Canonical Generation: Generating Sequences

– an idea extending Brendan's orderly generation approach.

The core: Not consider the task as just "constructing a matroid", but "*constructing a particular generating sequence*" (leading to this matroid).

- This is a better framework to capture various involved connectivity restrictions in chain and splitter theorems.
- It is really needed, say (in MACEK), if one wants to stick with a particular matrix representation.

3 Canonical Generation: Generating Sequences

– an idea extending Brendan's orderly generation approach.

The core: Not consider the task as just "constructing a matroid", but "*constructing a particular generating sequence*" (leading to this matroid).

- This is a better framework to capture various involved connectivity restrictions in chain and splitter theorems.
- It is really needed, say (in MACEK), if one wants to stick with a particular matrix representation.

Canonical minimality: Among all *generating sequences* leading to isomorphic "results", define a linear *canonical order*.

Always generate only the canonically minimal sequence among all.

3 Canonical Generation: Generating Sequences

– an idea extending Brendan's orderly generation approach.

The core: Not consider the task as just "constructing a matroid", but "*constructing a particular generating sequence*" (leading to this matroid).

- This is a better framework to capture various involved connectivity restrictions in chain and splitter theorems.
- It is really needed, say (in MACEK), if one wants to stick with a particular matrix representation.

Canonical minimality: Among all *generating sequences* leading to isomorphic "results", define a linear *canonical order*.

Always generate only the canonically minimal sequence among all.

- Of course, this canonical order must be hereditary (on subseq.).

3 Canonical Generation: Generating Sequences

– an idea extending Brendan's orderly generation approach.

The core: Not consider the task as just "constructing a matroid", but "*constructing a particular generating sequence*" (leading to this matroid).

- This is a better framework to capture various involved connectivity restrictions in chain and splitter theorems.
- It is really needed, say (in MACEK), if one wants to stick with a particular matrix representation.

Canonical minimality: Among all *generating sequences* leading to isomorphic "results", define a linear *canonical order*.

Always generate only the canonically minimal sequence among all.

- Of course, this canonical order must be hereditary (on subseq.).
- "All" generating sequences can be easily replaced with "all conforming to some arbitrary criteria" if these criteria are hereditary, too.

The overall advantage: Now, we can define many, even really weird, restrictions on the generating sequences; they just have to be hereditary. . .

The price to pay: Often, the requirement of being *hereditary on subsequences* implies quite "expensive" canonical orderings, such as:

The overall advantage: Now, we can define many, even really weird, restrictions on the generating sequences; they just have to be hereditary...

The price to pay: Often, the requirement of being *hereditary on subsequences* implies quite "expensive" canonical orderings, such as:

- A lexicographical ordering on the sequences with the heavier keys "on the left" (beginning of the sequence) – unlike the orderly generation which simply takes the heaviest key at the sequence end.

The overall advantage: Now, we can define many, even really weird, restrictions on the generating sequences; they just have to be hereditary. . .

The price to pay: Often, the requirement of being *hereditary on subsequences* implies quite "expensive" canonical orderings, such as:

- A lexicographical ordering on the sequences with the heavier keys "on the left" (beginning of the sequence) – unlike the orderly generation which simply takes the heaviest key at the sequence end.
- Consequently, a canonicity test has to evaluate all possible generating sequences, and not only the possible last steps.

The overall advantage: Now, we can define many, even really weird, restrictions on the generating sequences; they just have to be hereditary...

The price to pay: Often, the requirement of being *hereditary on subsequences* implies quite "expensive" canonical orderings, such as:

- A lexicographical ordering on the sequences with the heavier keys "on the left" (beginning of the sequence) – unlike the orderly generation which simply takes the heaviest key at the sequence end.
- Consequently, a canonicity test has to evaluate all possible generating sequences, and not only the possible last steps.
- Yet, a quite efficient implementation is possible, cf. MACEK.

Some implementation details:

- The generating framework (high-level) gets a generating sequence on one side, and a list of all possible one-element additions on the other side.

Some implementation details:

- The generating framework (high-level) gets a generating sequence on one side, and a list of all possible one-element additions on the other side.
- The canonical minimality testing is coded in the framework, calling an external elementary comparison function.

Some implementation details:

- The generating framework (high-level) gets a generating sequence on one side, and a list of all possible one-element additions on the other side.
- The canonical minimality testing is coded in the framework, calling an external elementary comparison function.
- An external function for testing admissibility of a generating sequence is provided as well.

Some implementation details:

- The generating framework (high-level) gets a generating sequence on one side, and a list of all possible one-element additions on the other side.
- The canonical minimality testing is coded in the framework, calling an external elementary comparison function.
- An external function for testing admissibility of a generating sequence is provided as well.
- Both the aforementioned external functions must be sufficiently "fragmented", so that the framework can "gradually" call the admissibility and canonicity test (from the least to the most expensive ones)!

3.1 Back to the second MACEK example

Find the matroids M with both $M(K_{3,3})$ and $M(K_{3,3})^*$ as minors such that no proper minor of M has both $M(K_{3,3})$ and $M(K_{3,3})^*$ as minors.

3.1 Back to the second MACEK example

Find the matroids M with both $M(K_{3,3})$ and $M(K_{3,3})^*$ as minors such that no proper minor of M has both $M(K_{3,3})$ and $M(K_{3,3})^*$ as minors.

- A generating sequence leading to such an intertwine M should

3.1 Back to the second MACEK example

Find the matroids M with both $M(K_{3,3})$ and $M(K_{3,3})^*$ as minors such that no proper minor of M has both $M(K_{3,3})$ and $M(K_{3,3})^*$ as minors.

- A generating sequence leading to such an intertwine M should
  - start from $M(K_{3,3})$ (as a minor),

3.1 Back to the second MACEK example

Find the matroids M with both $M(K_{3,3})$ and $M(K_{3,3})^*$ as minors such that no proper minor of M has both $M(K_{3,3})$ and $M(K_{3,3})^*$ as minors.

- A generating sequence leading to such an intertwine M should
  - start from $M(K_{3,3})$ (as a minor),
  - maintain 3-connectivity by the wheels-and-whirls theorem, and

3.1 Back to the second MACEK example

Find the matroids M with both $M(K_{3,3})$ and $M(K_{3,3})^*$ as minors such that no proper minor of M has both $M(K_{3,3})$ and $M(K_{3,3})^*$ as minors.

- A generating sequence leading to such an intertwine M should
  - start from $M(K_{3,3})$ (as a minor),
  - maintain 3-connectivity by the wheels-and-whirls theorem, and
  - stipulate that there is no $M(K_{3,3})^*$ minor except possibly at the sequence end.

3.1 Back to the second MACEK example

Find the matroids M with both $M(K_{3,3})$ and $M(K_{3,3})^*$ as minors such that no proper minor of M has both $M(K_{3,3})$ and $M(K_{3,3})^*$ as minors.

- A generating sequence leading to such an intertwine M should
  - start from $M(K_{3,3})$ (as a minor),
  - maintain 3-connectivity by the wheels-and-whirls theorem, and
  - stipulate that there is no $M(K_{3,3})^*$ minor except possibly at the sequence end.
- These requirements are hereditary,
  and we avoid duplicates in the previously used one-element addition step.
- The final step of validating an intertwine remains the same.

4 Some Final Thoughts

- **MACEK is over. . .**

4 Some Final Thoughts

- **MACEK is over. . .**

  Yes, it is still out there, and it compiles smoothly in new Linux and gcc4.5. Just its development stopped years ago, and many of its core design ideas are now overcome.

4 Some Final Thoughts

- **MACEK is over...**

  Yes, it is still out there, and it compiles smoothly in new Linux and gcc4.5. Just its development stopped years ago, and many of its core design ideas are now overcome.

- Though, some algorithmic ideas from extensive MACEK documentation might be useful in future development of matroid computation (I hope).

4 Some Final Thoughts

- **MACEK is over. . .**

  Yes, it is still out there, and it compiles smoothly in new Linux and gcc4.5. Just its development stopped years ago, and many of its core design ideas are now overcome.

- Though, some algorithmic ideas from extensive MACEK documentation might be useful in future development of matroid computation (I hope).

- And the idea of enhanced flexible canonical generating sequences could possibly be used in the core of the generating process in future matroid computation kits.

4 Some Final Thoughts

- **MACEK is over. . .**

  Yes, it is still out there, and it compiles smoothly in new Linux and gcc4.5. Just its development stopped years ago, and many of its core design ideas are now overcome.

- Though, some algorithmic ideas from extensive MACEK documentation might be useful in future development of matroid computation (I hope).

- And the idea of enhanced flexible canonical generating sequences could possibly be used in the core of the generating process in future matroid computation kits.

THANK YOU FOR YOUR ATTENTION