Engineering motif search for large graphs

Andreas Björklund
Lund University

Petteri Kaski
Aalto University, Helsinki

Łukasz Kowalik
Warsaw University

Juho Lauri
Tampere University of Technology

Simons Institute for the Theory of Computing
Thursday 5 November 2015

Aalto University
School of Science

01010111010101
01001010101010
10101010101010

10101011110101
01010101011101
01010111010110
10101101010110
10101110101010
11101010101101
01110111010110
10111011010101
11110101010101
00010101010101
01011010101110
10101010100101
01101010101011

Tight results

Satisfiability Lower Bounds and Tight Results for Parameterized and Exponential-Time Algorithms

Nov. 2 – Nov. 6, 2015

Program: Fine-Grained Complexity and Algorithm Design

Are tight algorithms useful,
***in practice*?**

[here: practice ~ proof-of-concept algorithm engineering]

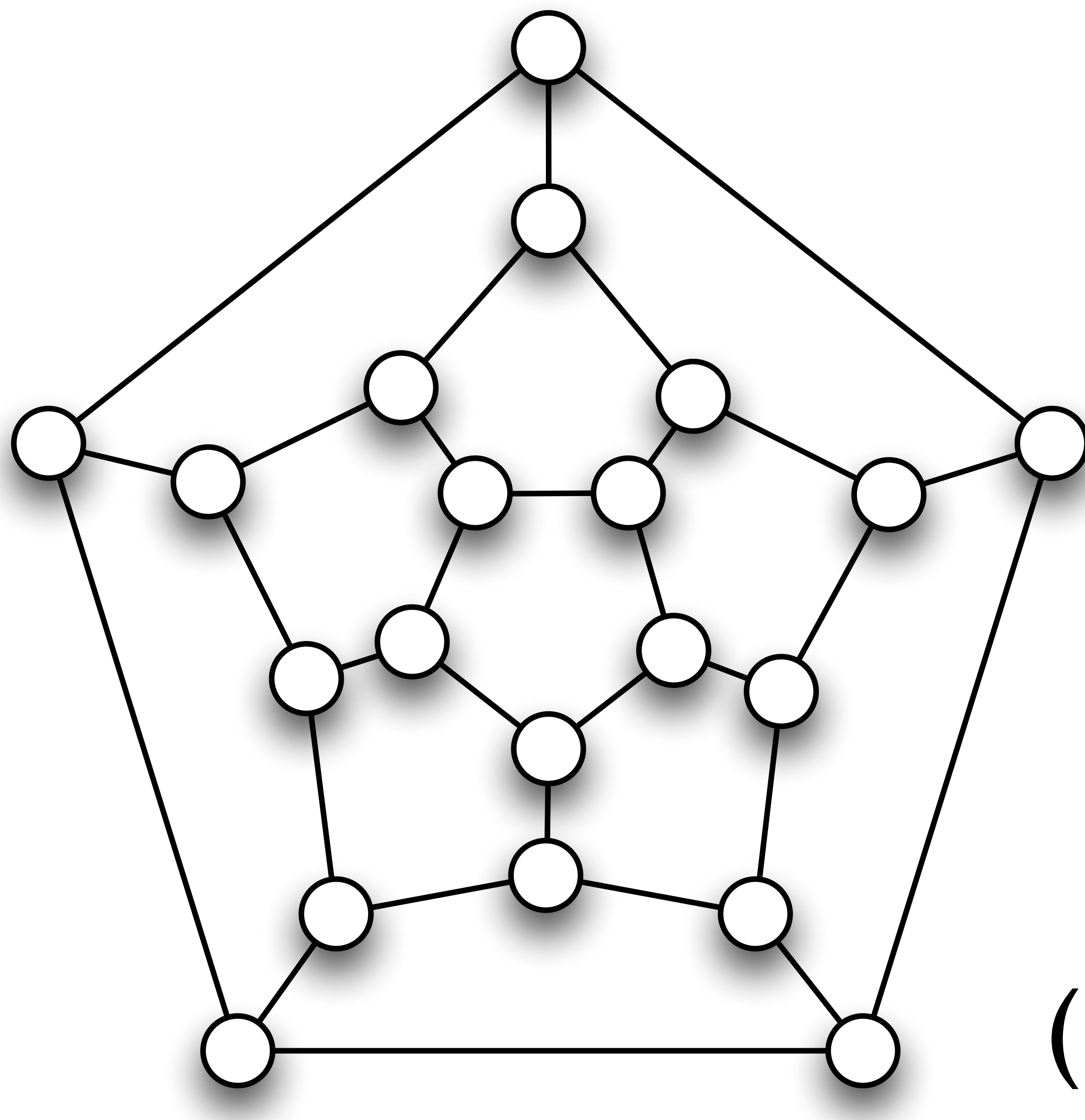
A coarse-grained view

- **Data**
  — "large" (e.g. large database)
- **Task**
  — "small" (e.g. search for a small *pattern* in data)
  — all too often NP-hard

We need a more *fine-grained* perspective

Graph search

Data

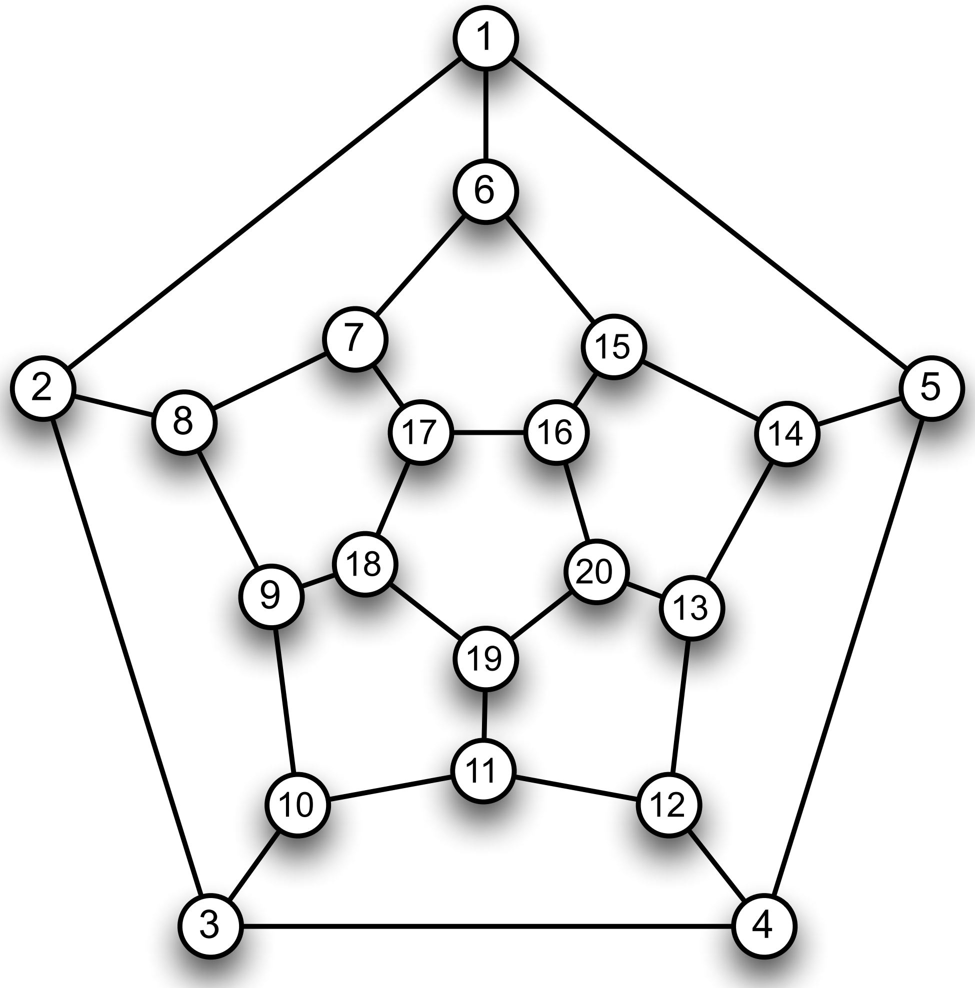


(+ annotation)

Pattern (query)

Task (search for *matches* to query)

Large data (large graph)



1,2	2,8	8,9	14,15	15,16
2,3	3,10	9,10	6,15	16,17
3,4	4,12	10,11	7,17	17,18
4,5	5,14	11,12	9,18	18,19
1,5	6,7	12,13	11,19	19,20
1,6	7,8	13,14	13,20	16,20

(edge list representation)

One edge
= two 64-bit integers
(2 x 8 = 16 bytes)

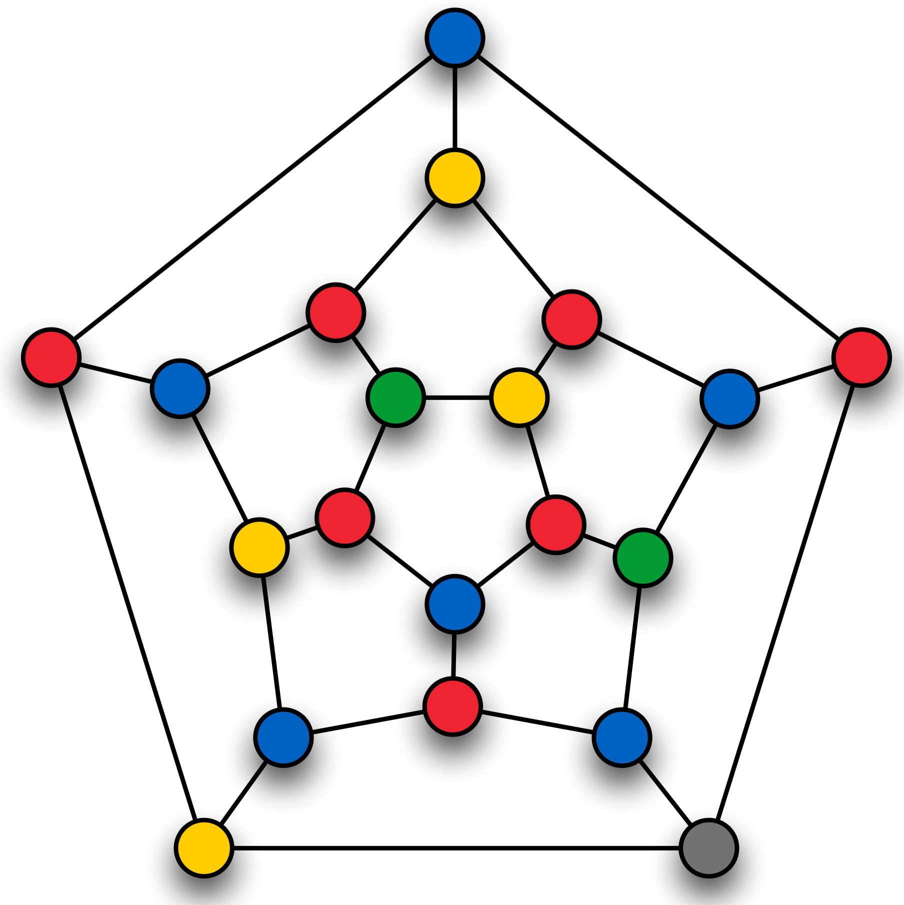
One terabyte
(=10^{12} bytes)
stores about
60 billion edges

~10^{10} edges,
arbitrary topology

Motif search

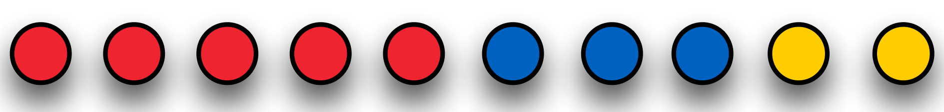
Data

Vertex-colored graph H
(the **host graph**)



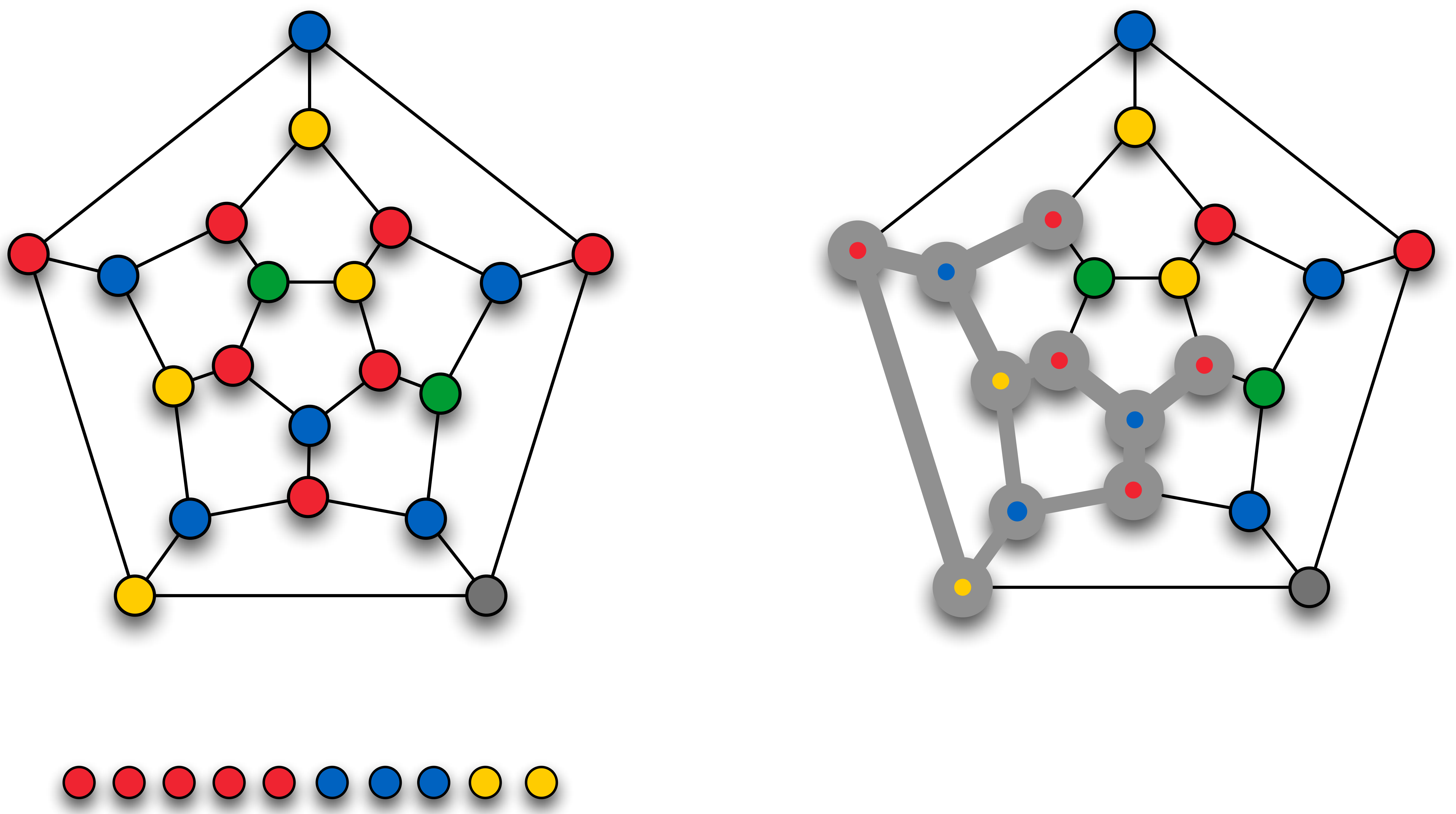
Query

Multiset M
of colors (the **motif**)



Task (decision):
Is there a <u>connected</u> subgraph whose colors agree with M ?

Data, query, and one match



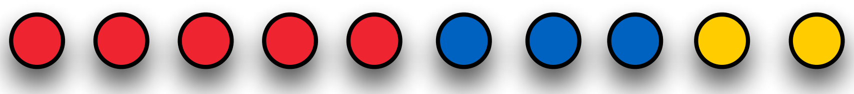
Limited background on motif search

- Extension of *jumbled pattern matching* on strings (=paths) and trees
- This variant introduced by Lacroix *et al.*
  (IEEE/ACM Trans. Comput. Biology Bioinform. 2006)
- Many variants and extensions
  - Exact match
    (Lacroix *et al.* 2006)
  - Match (large enough) multisubset
    (Dondi *et al.* 2009)
  - Multiple color constraints, weights on edges, scoring by weight
    (Bruckner *et al.* 2009)
  - Minimum-add / minimum-substitution distance
    (Dondi *et al.* 2011)
  - Minimum weighted edit distance
    (Björklund *et al.* 2013)

  ⋮

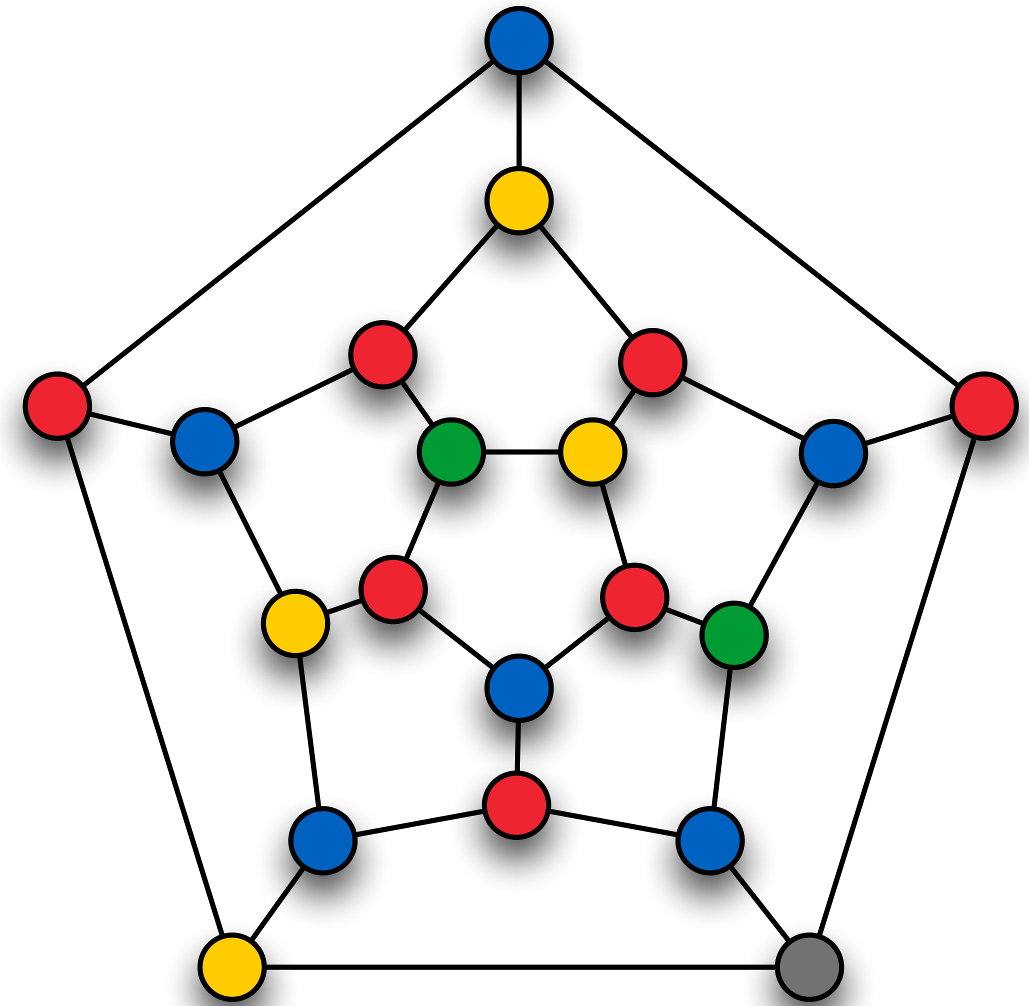
Complexity of motif search

NP-complete
if M has at least
two colors



(easy reduction from Steiner tree)

NP-complete on
trees with max. degree 3,
M has distinct colors
(Fellows *et al.* 2007)



Solvable
in linear time
in the size of H

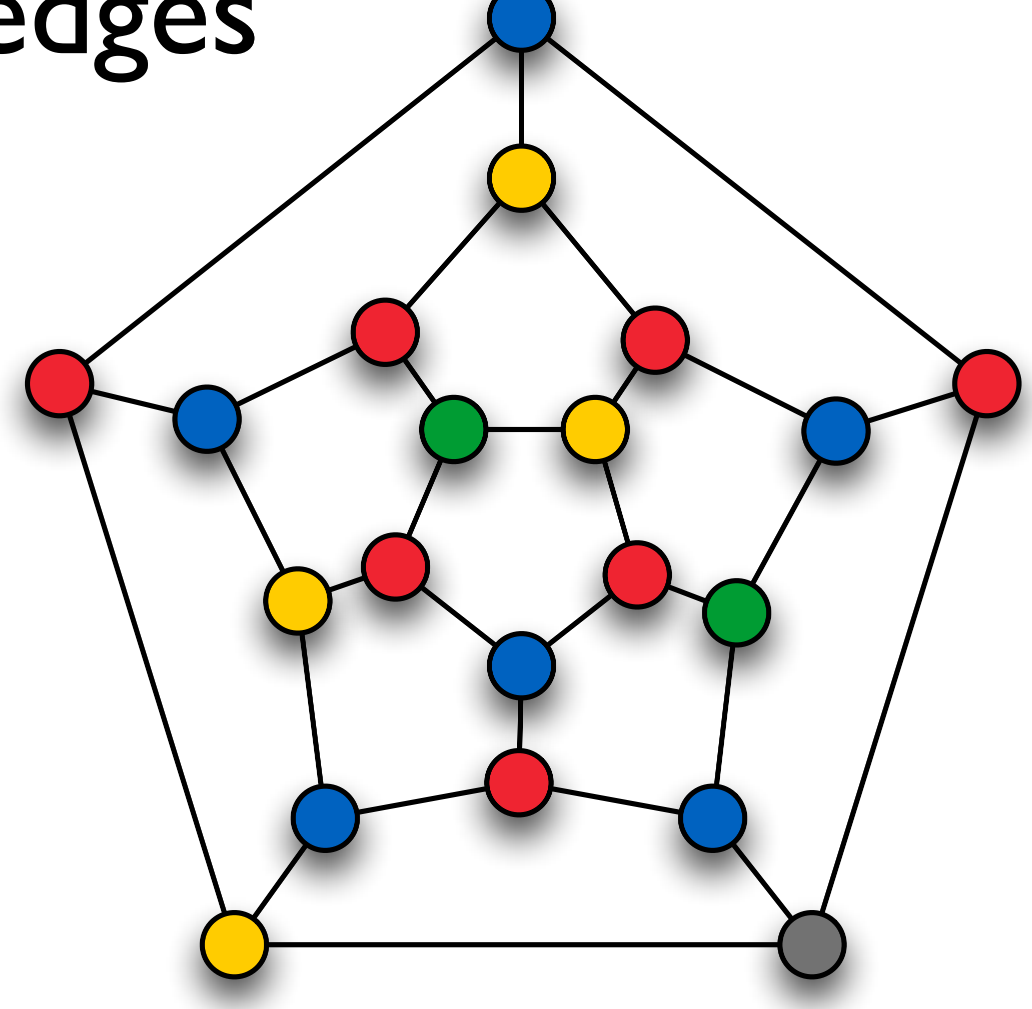
(and exponential in the size of M)

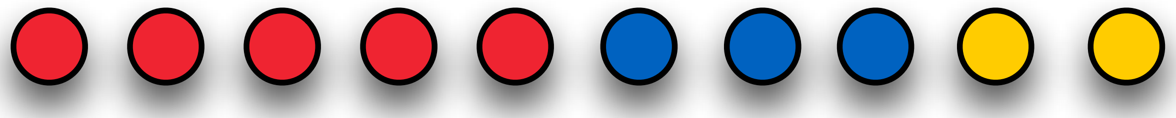
Parameterization

Let H have *n* vertices and *m* edges

Let M have size *k*

Worst-case running time
as a function of n, m, k ?





Dependence on k

Authors	Time		Approach
Fellows *et al.*	$O^*(\sim 87^k)$	2007	Color coding
Betzler *et al.*	$O^*(4.32^k)$	2008	Color coding
Guillemot & Sikora	$O^*(4^k)$	2010	Multilinear detection
Koutis	$O^*(2.54^k)$	2012	Constrained multilin.
Björklund *et al.*	$O^*(2^k)$	2013	Constrained multilin.

"FPT race"

tight
(unless there is
a breakthrough for
SET COVER)

Tightness (conditional)

SET COVER
Input: Sets $S_1, S_2, \ldots, S_m \subseteq \{1,2,\ldots,n\}$
Budget $t \in \mathbb{Z}$
Question:
Do there exist sets $S_{i_1}, S_{i_2}, \ldots, S_{i_t}$ with $S_{i_1} \cup S_{i_2} \cup \cdots \cup S_{i_t} = \{1,2,\ldots,n\}$?

Theorem [Björklund, K., Kowalik 2013]
If GRAPH MOTIF can be solved in $O^*((2-\varepsilon)^k)$ time,
then SET COVER can be solved in $O^*((2-\varepsilon')^n)$ time

Key lemma [implicit in Cygan et. al 2012]:
If SET COVER can be solved in $O^*((2-\varepsilon)^{n+t})$ time,
then it can also be solved in $O^*((2-\varepsilon')^n)$ time

Tight results

Satisfiability Lower Bounds and Tight Results for Parameterized and Exponential-Time Algorithms

Nov. 2 – Nov. 6, 2015

Program: Fine-Grained Complexity and Algorithm Design

Are tight algorithms useful,
***in practice*?**

Are tight algorithms useful, *in practice* ?

For GRAPH MOTIF, can we engineer an implementation that scales to *large* graphs?

(as long as the motif size *k* is small)

Starting point (theory): $\tilde{O}(2^k k^2 m)$-time randomized algorithm (decides *existence* of match)

Theory background for tight algorithm

- Key idea: **algebrize** the combinatorial problem
  — here: use *constrained multilinear detection*

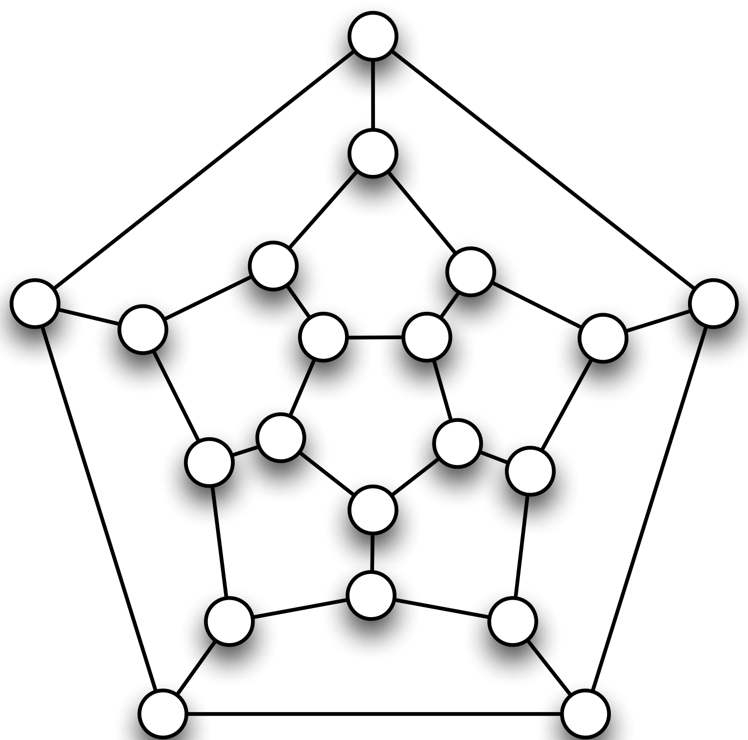
- Pioneered in the context of group algebras

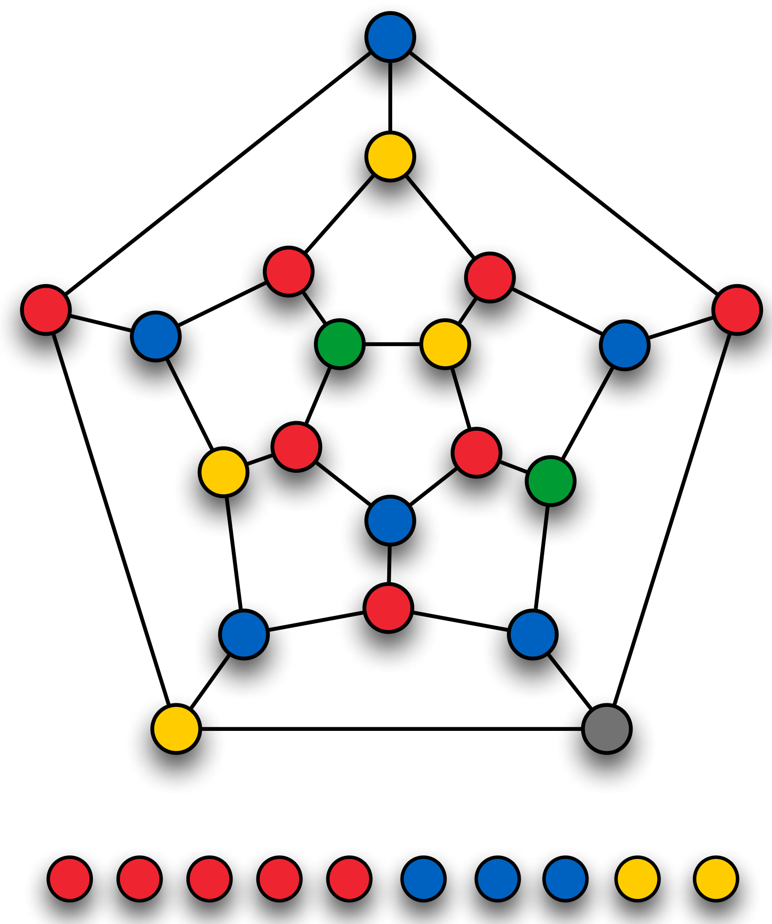
  Koutis (2008), Williams (2009),
  Koutis and Williams (2009),
  Koutis (2010), Koutis (2012)

- Here we use generating polynomials
  and substitution sieving in characteristic 2

  Björklund (2010),
  Björklund *et al.* (2010, 2013)

The algebraic view





1) connected subgraphs

... are witnessed by *multilinear* monomials in a generating polynomial $P_{H,k}(\mathbf{x},\mathbf{y})$

fast <u>evaluation</u> algorithm for $P_{H,k}(\mathbf{x},\mathbf{y})$

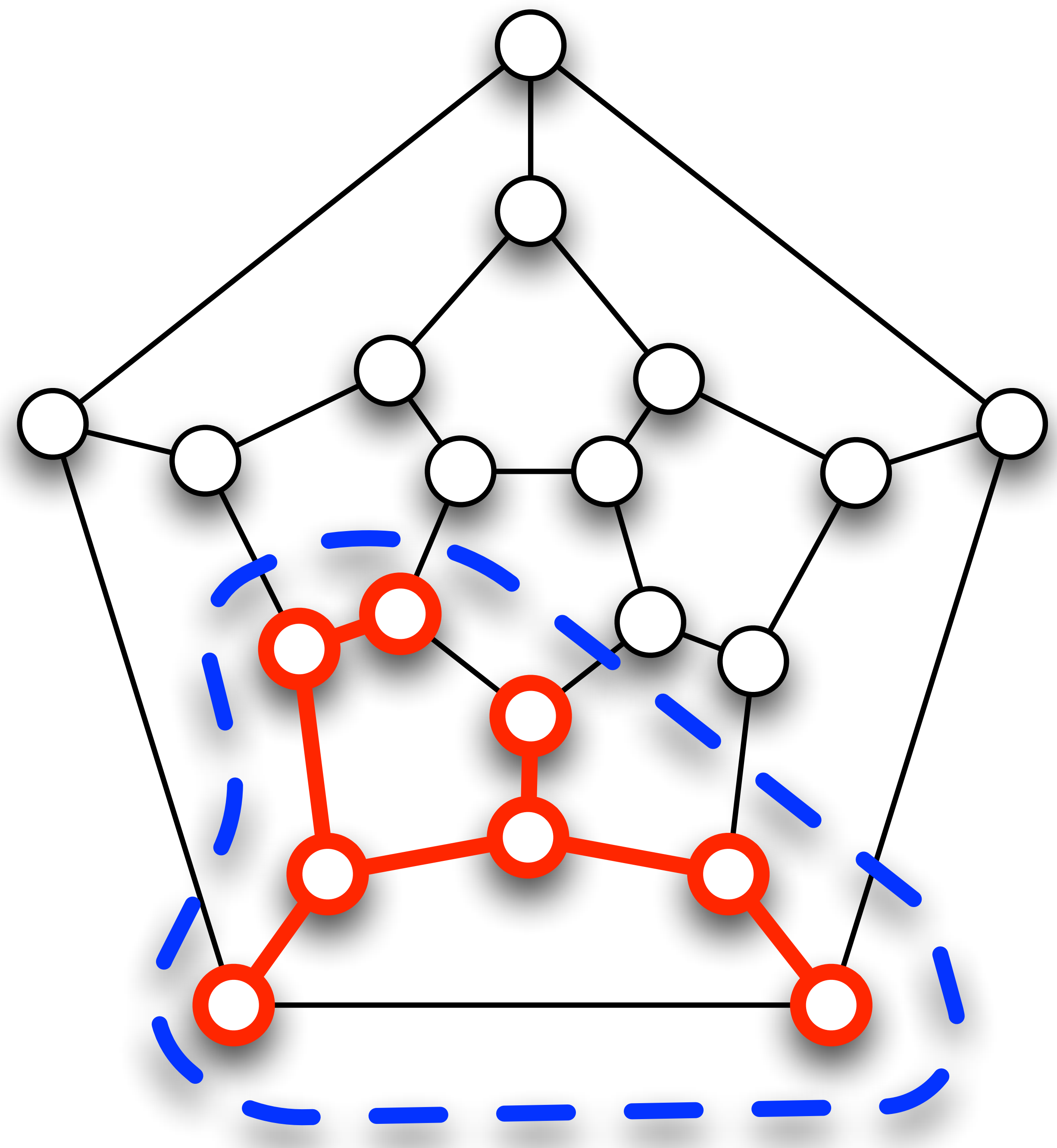
2) match colors with motif

... multilinear monomials *whose colors match motif*

randomized detection with 2^k <u>evaluations</u> of $P_{H,k}(\mathbf{x},\mathbf{y})$

Connected sets to multilinearity

Intuition:
Use spanning trees to
witness connected sets



Every connected
set of vertices
has at least one
spanning tree

Connected sets to multilinearity

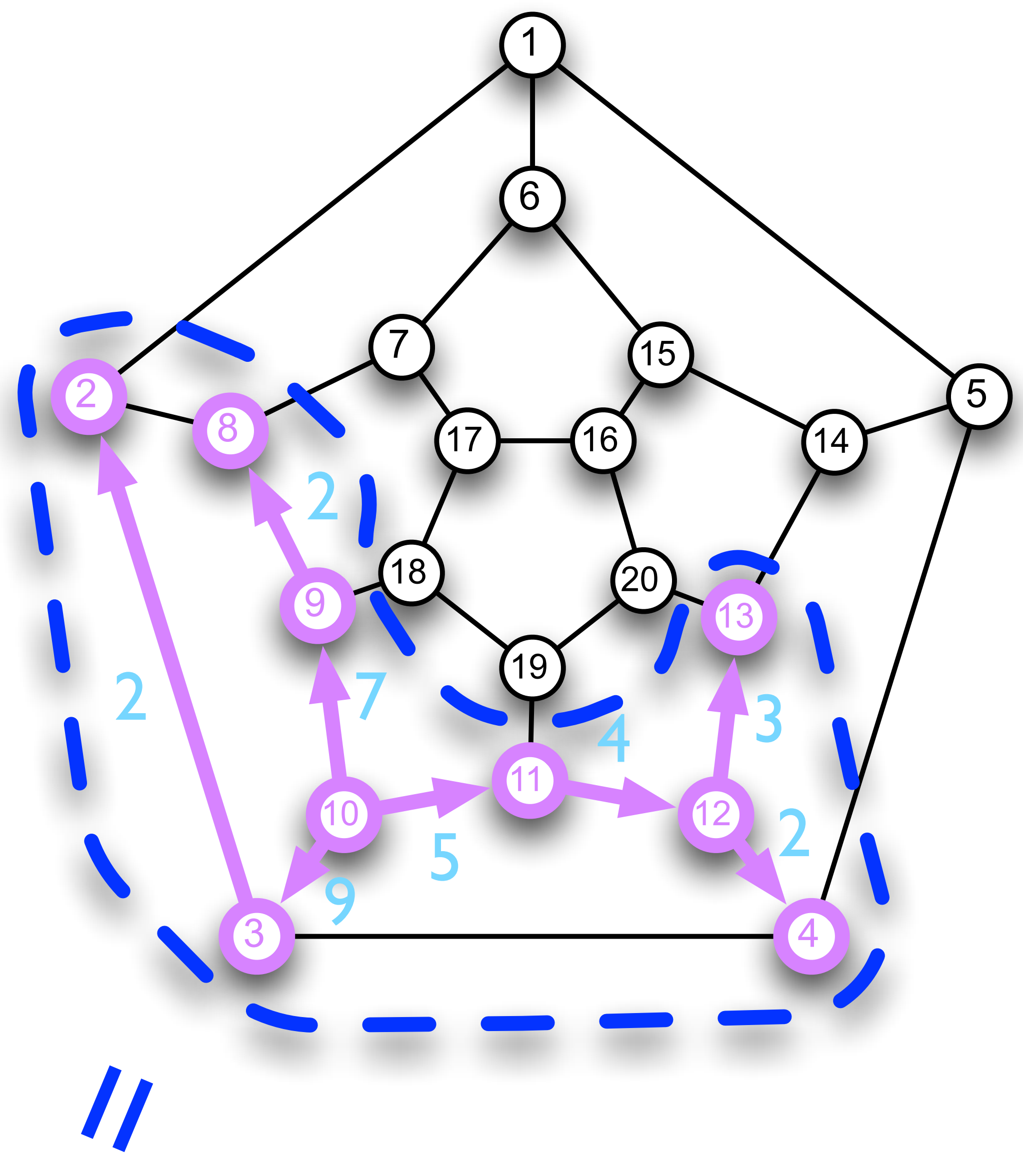
- Key idea:
  Branching walks (Nederlof 2008)
  [introduced in the context of inclusion-exclusion algorithms for Steiner tree]
- Transported to multivariate polynomial algebrizations of connected sets
  (Guillemot and Sikora 2010)
- A multivariate polynomial with edge-linear time, vertex-linear working memory evaluation algorithm
  (Björklund, K., Kowalik 2013 & 2015)

The polynomial $P_{H,k}(\mathbf{x},\mathbf{y})$

Each *"rooted spanning tree"* of size k in H occurs as a unique *multilinear* monomial in $P_{H,k}(\mathbf{x},\mathbf{y})$

There are no other *multilinear* monomials in $P_{H,k}(\mathbf{x},\mathbf{y})$

Given values to the variables **x**,**y**, the value $P_{H,k}(\mathbf{x},\mathbf{y})$ can be computed fast



$x_2\, x_3\, x_4\, x_8\, x_9\, x_{10}\, x_{11}\, x_{12}\, x_{13}\, y_{2,(3,2)}\, y_{2,(9,8)}\, y_{9,(10,3)}\, y_{7,(10,9)}\, y_{5,(10,11)}\, y_{4,(11,12)}\, y_{2,(12,4)}\, y_{3,(12,13)}$

Evaluation algorithm at point (x,y)

Base case, for all $u \in V(H)$

$$P_{1,u}(\mathbf{x},\mathbf{y}) = x_u$$

Dynamic programming

- <u>edge-linear</u> $\tilde{O}(k^2 m)$ time
- <u>vertex-linear</u> $\tilde{O}(kn)$ working memory

Iteration, for all $\ell = 2, 3, \ldots, k$ and all $u \in V(H)$

$$P_{\ell,u}(\mathbf{x},\mathbf{y}) = \sum_{v \in N_H(u)} y_{\ell,(u,v)} \sum_{\substack{\ell_1+\ell_2=\ell \\ \ell_1,\ell_2 \geq 1}} P_{\ell_1,u}(\mathbf{x},\mathbf{y}) P_{\ell_2,v}(\mathbf{x},\mathbf{y})$$

Finally, take the sum over all root vertices

$$P(\mathbf{x},\mathbf{y}) = \sum_{u \in V(H)} P_{k,u}(\mathbf{x},\mathbf{y})$$

Rand. algorithm for motif search (decision)

- Ideas: 1) polynomial $P_{H,k}(\mathbf{x}, \mathbf{y})$
  2) constrained multilinearity sieve
  3) DeMillo–Lipton–Schwartz–Zippel lemma

- Requires 2^k evaluations of $P_{H,k}(\mathbf{x}, \mathbf{y})$, which leads to running time $\tilde{O}(2^k k^2 m)$ and working memory $\tilde{O}(kn)$

- Algorithm is (essentially) just a big sum: The 2^k evaluations can be executed *in parallel*

No false positives
False negatives with probability at most $k \cdot 2^{-b+1}$
(arithmetic over $GF(2^b)$, $b = O(\log k)$)

Are tight algorithms useful, *in practice* ?

Starting point (theory): $\tilde{O}(2^k k^2 m)$-time randomized algorithm for graph motif (decides *existence* of match)

Engineering aspects

- Here focus on:
  Shared-memory multiprocessors (CPU-based)

- Two key subsystems
  - Memory (DDR3/DDR4-SDRAM)
  - CPUs (Intel x86–64 with ISA extensions)

    (e.g. Haswell/Broadwell microarchitecture with AVX2, PCLMULQDQ)

Engineering an implementation

the new generating polynomial $P_{H,k}(x,y)$
and parallel evaluation algorithm

- **Capacity**
  - O(*kn*) working memory
  - use ISA extensions (AVX2 + PCLMULQDQ), if available, for arithmetic in $GF(2^b)$
- **Bandwidth**
  - use memory one 512-bit cache line at a time
  - use all CPUs, all cores, all (vector) ports

    multithreading vectorization
- **Latency**
  - hardware *and software* prefetching
  - hide latency with enough instructions "in flight"

Evaluating $P_{H,k}(x,y)$

Base case, for all $u \in V(H)$

$$P_{1,u}(\mathbf{x},\mathbf{y}) = x_u$$

Vectorization over several independent points $(\mathbf{x}^{(j)},\mathbf{y}^{(j)})$ at once

Iteration, for all $\ell = 2, 3, \ldots, k$ and all $u \in V(H)$

$$P_{\ell,u}(\mathbf{x},\mathbf{y}) = \sum_{v \in N_H(u)} y_{\ell,(u,v)} \sum_{\substack{\ell_1+\ell_2=\ell \\ \ell_1,\ell_2 \geq 1}} P_{\ell_1,u}(\mathbf{x},\mathbf{y}) P_{\ell_2,v}(\mathbf{x},\mathbf{y})$$

Finally, take the sum over all root vertices

$$P(\mathbf{x},\mathbf{y}) = \sum_{u \in V(H)} P_{k,u}(\mathbf{x},\mathbf{y})$$

Multithreading over vertices u (layer l fixed)

Inner loop in C

Iteration, for all $\ell = 2, 3, \ldots, k$ and all $u \in V(H)$

$$P_{\ell,u}(\mathbf{x},\mathbf{y}) = \sum_{v \in N_H(u)} y_{\ell,(u,v)} \sum_{\substack{\ell_1+\ell_2=\ell \\ \ell_1,\ell_2 \geq 1}} P_{\ell_1,u}(\mathbf{x},\mathbf{y}) P_{\ell_2,v}(\mathbf{x},\mathbf{y})$$

```
for(index_t l1 = 1; l1 < l; l1++) {
    line_t pul1, pvl2;
    index_t l2 = l-l1;
    index_t i_v_l2 = ARB_LINE_IDX(b, k, l2, v);
    LINE_LOAD(pvl2, d_s, i_v_l2);                // data-dependent load
    index_t i_u_l1 = ARB_LINE_IDX(b, k, l1, u);
    LINE_LOAD(pul1, d_s, i_u_l1);
    index_t i_nv_l2 = ARB_LINE_IDX(b, k, l2, nv);
    LINE_PREFETCH(d_s, i_nv_l2);   // user prefetch data-dependent
    line_t p;                      // load (for next vertex v)
    LINE_MUL(p, pul1, pvl2);
    LINE_ADD(s, s, p);
}
```

Compiled inner loop (w/ AVX2 +PCLMULQDQ)

```
.L610:
        movq            %r9,    %rcx
        movq            %rdi,   %rsi
        imulq           %r8,    %rcx
        subq            %rax,   %rsi
        leaq            -1(%rsi,%rcx), %rcx
        salq            $6,     %rcx
        vmovdqu         (%rdx,%rcx), %ymm6
        vmovdqu         32(%rdx,%rcx), %ymm5
        movq            %rbx,   %rcx
        imulq           (%r15), %rcx
        vmovdqa         %xmm6, %xmm0
        vextracti128    $0x1,   %ymm6, %xmm6
        leaq            -1(%rax,%rcx), %rcx
        addq            $1,     %rax
        salq            $6,     %rcx
        vmovdqu         (%rdx,%rcx),   %ymm1
        vmovdqu         32(%rdx,%rcx), %ymm4
        leaq            -1(%rsi,%r10), %rcx
        vmovdqa         %xmm1, %xmm7
        vextracti128    $0x1,   %ymm1, %xmm1
        vpclmulqdq      $0,     %xmm6, %xmm1, %xmm2
        vpclmulqdq      $0,     %xmm0, %xmm7, %xmm3
        vpclmulqdq      $17,    %xmm6, %xmm1, %xmm1
        vmovdqa         %xmm4, %xmm6
        vinserti128     $0x1,   %xmm2, %ymm3, %ymm3
        vpclmulqdq      $17,    %xmm0, %xmm7, %xmm0
        vinserti128     $0x1,   %xmm1, %ymm0, %ymm0
        vpunpcklqdq     %ymm0, %ymm3, %ymm1
        vpunpckhqdq     %ymm0, %ymm3, %ymm3
        vmovdqa         %xmm5, %xmm7
        vpsrlq          $60,    %ymm3, %ymm0
        vextracti128    $0x1,   %ymm4, %xmm4
        vextracti128    $0x1,   %ymm5, %xmm5
        vpsrlq          $61,    %ymm3, %ymm2
        salq            $6,     %rcx
        cmpq            %rax,   %rdi
        vpxor           %ymm0, %ymm2, %ymm2
        vpsrlq          $63,    %ymm3, %ymm0
```

```
        prefetcht0      (%rdx,%rcx)
        vpxor           %ymm2, %ymm0, %ymm2
        vpxor           %ymm2, %ymm3, %ymm2
        vpsllq          $1,     %ymm2, %ymm0
        vpxor           %ymm1, %ymm0, %ymm0
        vpsllq          $3,     %ymm2, %ymm1
        vpclmulqdq      $0,     %xmm7, %xmm6, %xmm3
        vpxor           %ymm0, %ymm1, %ymm0
        vpsllq          $4,     %ymm2, %ymm1
        vpxor           %ymm0, %ymm1, %ymm0
        vpclmulqdq      $17,    %xmm7, %xmm6, %xmm1
        vpxor           %ymm0, %ymm2, %ymm2
        vpclmulqdq      $0,     %xmm5, %xmm4, %xmm0
        vpclmulqdq      $17,    %xmm5, %xmm4, %xmm4
        vinserti128     $0x1,   %xmm0, %ymm3, %ymm3
        vinserti128     $0x1,   %xmm4, %ymm1, %ymm1
        vpunpcklqdq     %ymm1, %ymm3, %ymm4
        vpunpckhqdq     %ymm1, %ymm3, %ymm1
        vpsrlq          $61,    %ymm1, %ymm3
        vpxor           %ymm2, %ymm8, %ymm8
        vmovdqa         %ymm8, 80(%rsp)
        vpsrlq          $60,    %ymm1, %ymm0
        vpxor           %ymm0, %ymm3, %ymm0
        vpsrlq          $63,    %ymm1, %ymm3
        vpxor           %ymm0, %ymm3, %ymm0
        vpxor           %ymm0, %ymm1, %ymm0
        vpsllq          $3,     %ymm0, %ymm3
        vpsllq          $1,     %ymm0, %ymm1
        vpxor           %ymm4, %ymm1, %ymm1
        vpxor           %ymm1, %ymm3, %ymm1
        vpsllq          $4,     %ymm0, %ymm3
        vpxor           %ymm1, %ymm3, %ymm1
        vpxor           %ymm1, %ymm0, %ymm0
        vpxor           %ymm0, %ymm9, %ymm9
        vmovdqa         %ymm9, 112(%rsp)
        jg              .L610
```

4 x GF(2^{64}) vectorization (4 independent points)

Open source

https://github.com/pkaski/motif-search

Experiments

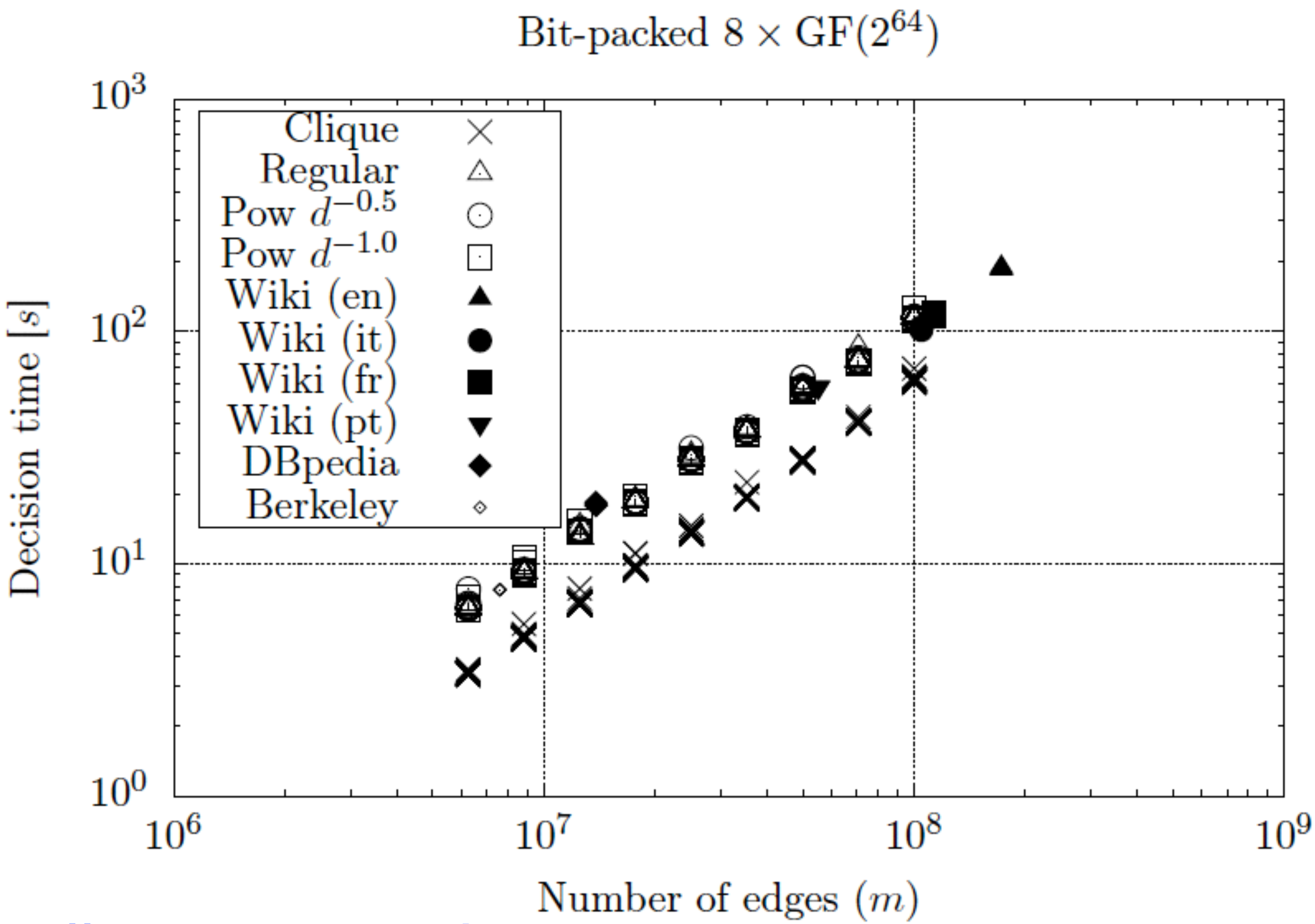
For GRAPH MOTIF,
can we engineer an implementation
that scales to *large* graphs?
(as long as the motif size *k* is small)

Hardware configurations

- **Small-memory node (1 CPU, total 4 cores)**
  — 1 x 3.20-GHz Intel Core i5-4570 CPU
  (Haswell muarch, 4 cores, 6 MiB LLC, 2 channels to main mem.)
  — 16 GiB main memory (4 x 4 GiB DDR3-1600)
- **Large-memory node (2 CPU, total 20 cores)**
  — 2 x 2.80-GHz Intel Xeon E5-2680 v2 CPU
  (Ivy Bridge muarch, 10 cores, 25 MiB LLC, 4 channels to main mem.)
  — 256 GiB main memory (16 x 16 GiB DDR3-1866)
- **Fat-memory node (4 CPU, total 24 cores)**
  — 4 x 2.67-GHz Intel Xeon X7542 CPU
  (Nehalem muarch, 6 cores, 18 MiB LLC, 1 channel to main mem.)
  — 1 TiB main memory (64 x 16 GiB DDR3-1066)

Edge-linear scaling

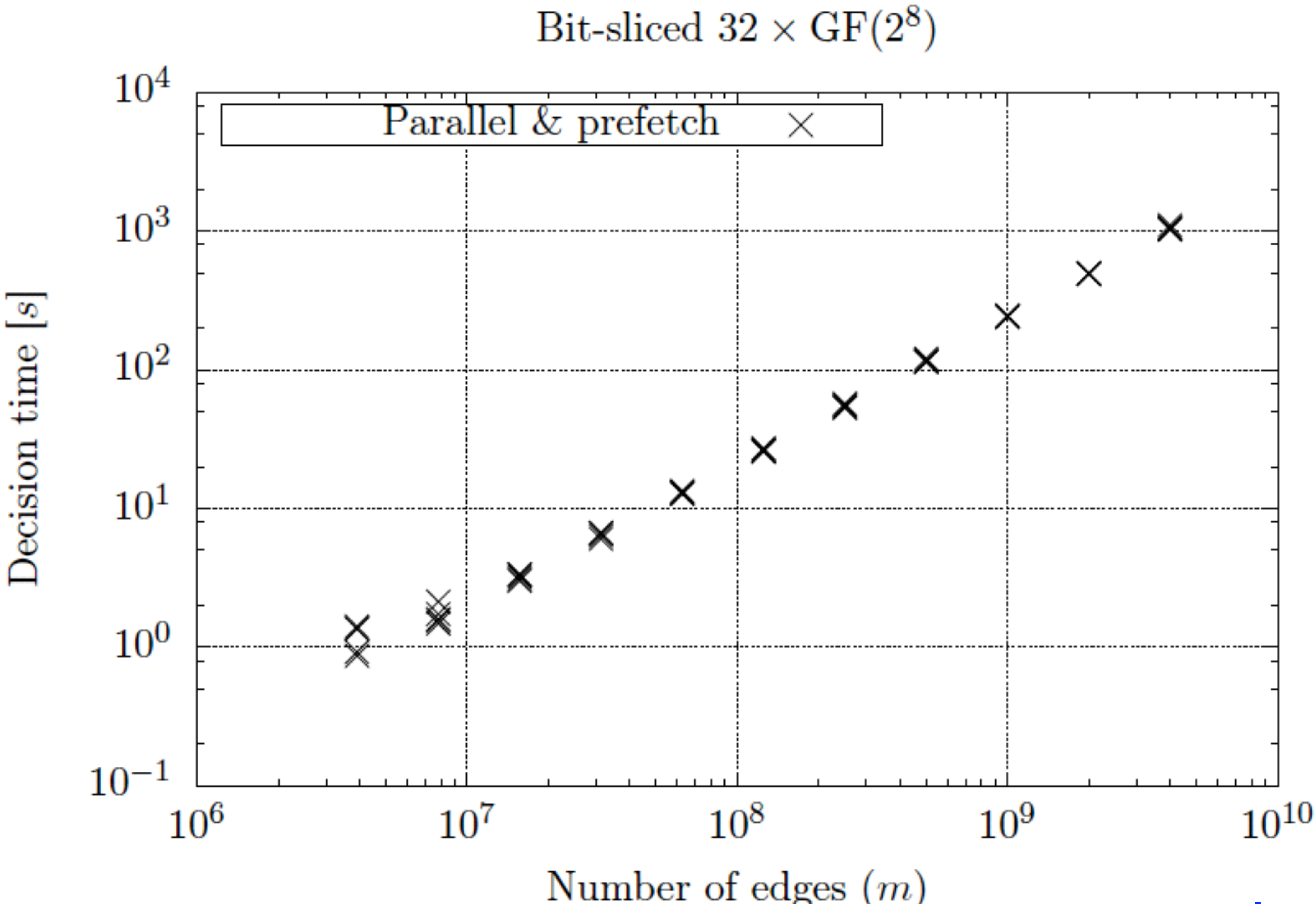
[Natural graphs from the Koblenz network collection]



Small-memory node

k = 5

Edge-linear scaling

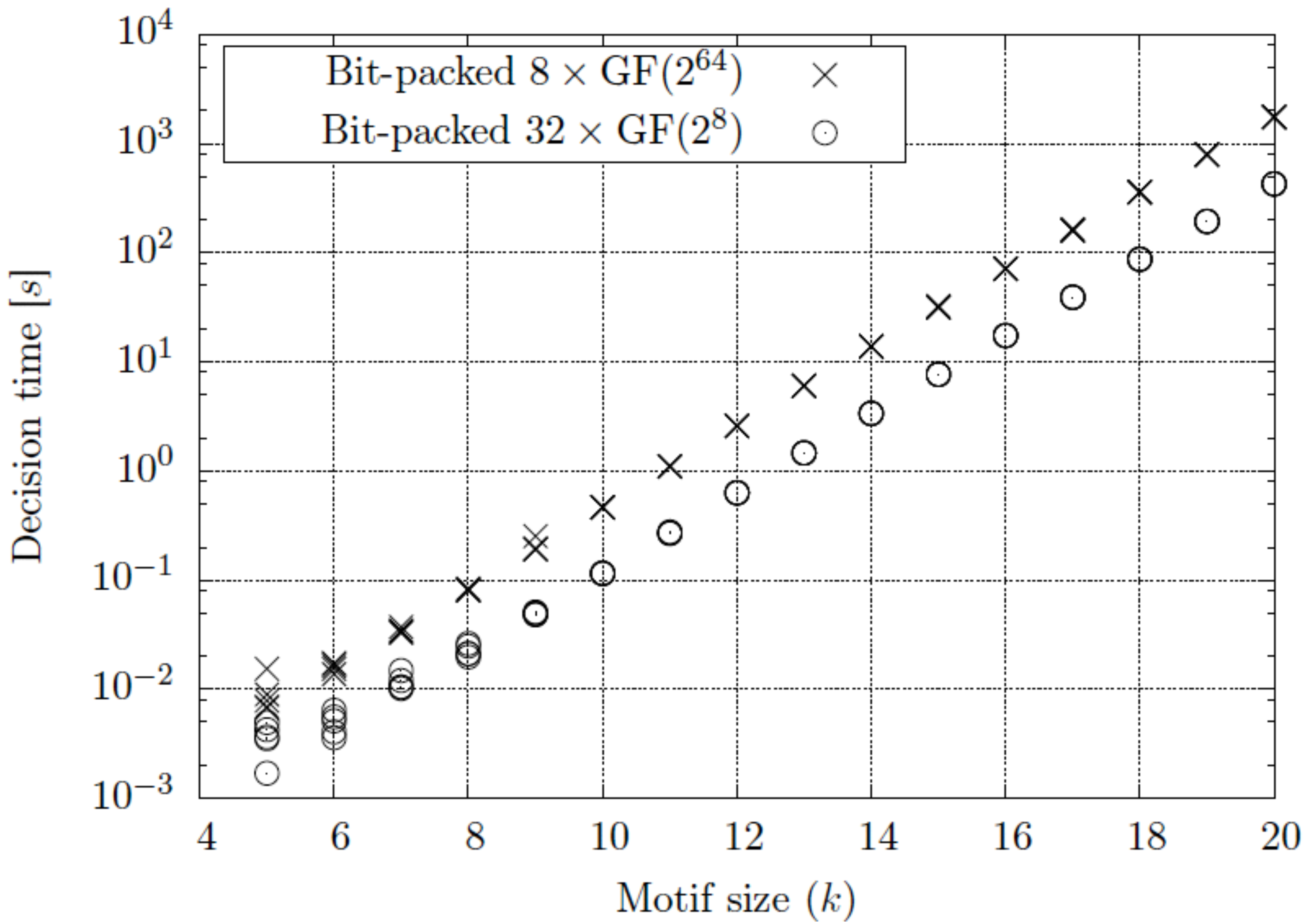


Large-memory node

k = 5 fixed

5 independent random 20-regular graphs for each value of m

Exponential scaling in k

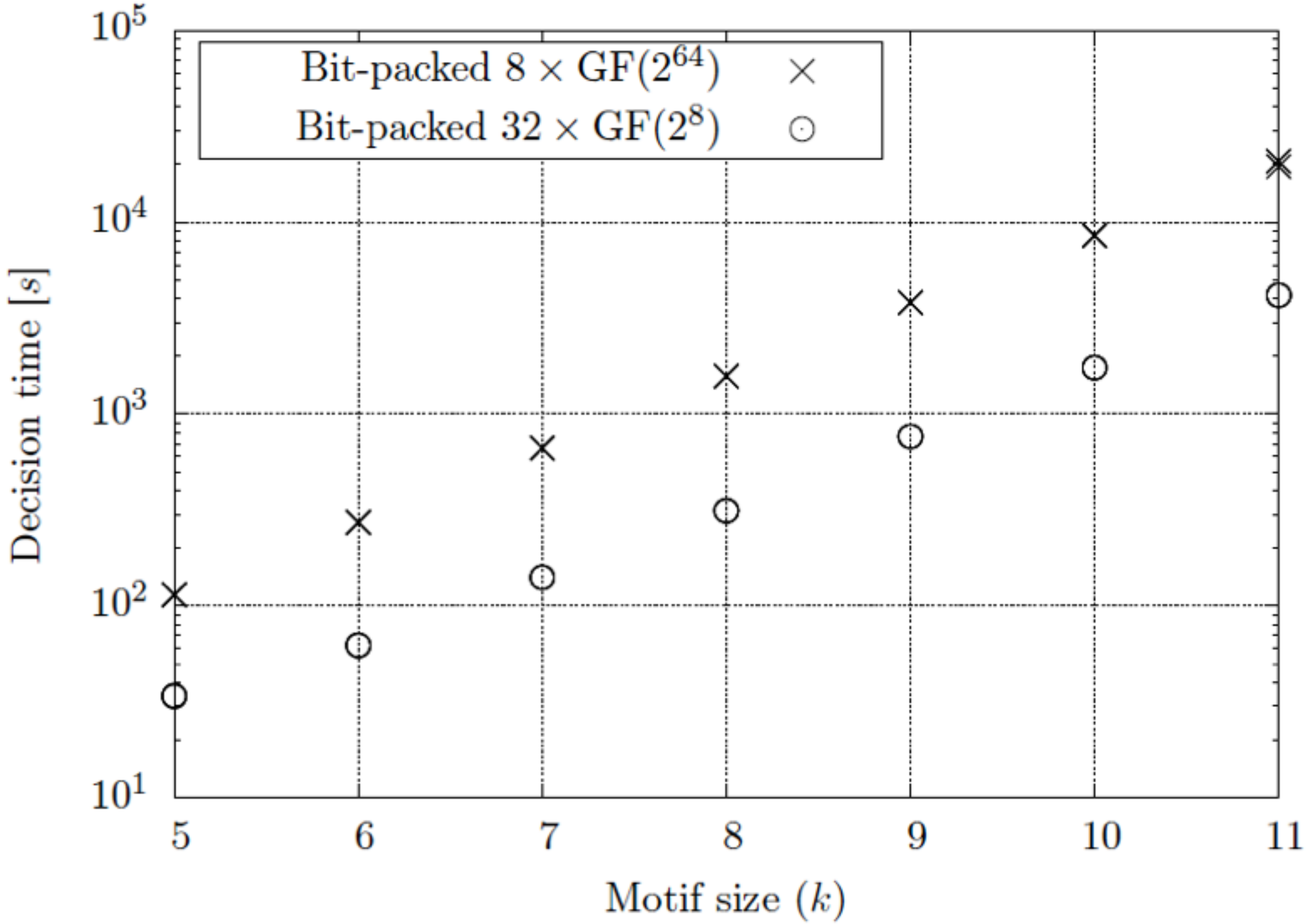


Small-memory node

n = 1000, m = 10000

5 independent random 20-regular graphs for each value of k

Exponential scaling in k



Small-memory node

n = 10 million, m = 100 million

5 independent random 20-regular graphs for each value of k

Large graphs

Vertices	Edges	Input	Preprocess	Decision	Total	Peak memory
2 000 000 000	20 000 000 004	2330 s	1937 s	3163 s	7452 s	693 GiB
1 000 000 000	10 000 000 004	1057 s	987 s	1545 s	3599 s	347 GiB
500 000 000	5 000 000 004	492 s	407 s	770 s	1673 s	174 GiB
250 000 000	2 500 000 004	237 s	183 s	376 s	799 s	87 GiB
125 000 000	1 250 000 004	112 s	90 s	182 s	386 s	44 GiB
62 500 000	625 000 004	55 s	43 s	88 s	187 s	22 GiB
1 000 000 000	20 000 000 004	2040 s	1830 s	2915 s	6805 s	623 GiB
500 000 000	10 000 000 004	939 s	816 s	1430 s	3196 s	312 GiB
250 000 000	5 000 000 004	467 s	409 s	704 s	1586 s	156 GiB
125 000 000	2 500 000 004	221 s	182 s	343 s	749 s	78 GiB
62 500 000	1 250 000 004	109 s	88 s	165 s	363 s	39 GiB

decision algorithm runtime

convert from edge list to adjacency list

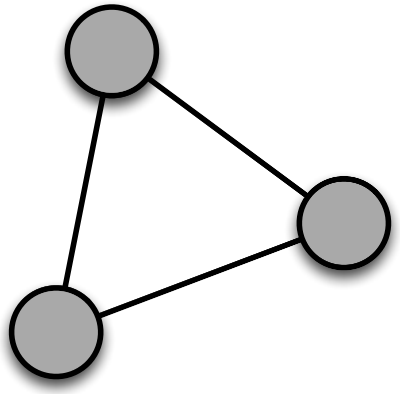
generate random regular input
(in edge list format)

Fat-memory node

$k = 5$

Summary (engineering)

- A proof-of-concept practical algorithm for small k, large m
- NP-hard problem, yet in practice (for small k) can process inputs with hundreds of millions of edges — many polynomial-time algorithms do worse than this!



- Algorithm is "just a big sum" — the same polynomial evaluated at different points — easy SIMD parallelization

Summary (engineering)

- Some implementation details to get performance:
  - Vectorized finite-field arithmetic (low-level implementation)
  - Using memory one 512-bit cache line at a time
  - Coping with latency: memory layout to enable *hardware prefetching*, *software-prefetch* indirect reads ahead of time
- Not covered in this presentation: how to upgrade decision algorithm to list all solutions
- See paper (ALENEX'15) and source code (~6000 lines of C):

http://dx.doi.org/10.1137/1.9781611973754.10

https://github.com/pkaski/motif-search

Summary (theory)

- **Theory work supports engineering**
  (here: generating polynomial, multilinear sieves,
  polynomial identity testing, …)
- Derandomization?
  Indexing (preprocessing) the data to enable fast search?
- Coping with increasing latencies?
- *Yet tighter* (yet more *fine-grained*) algorithms?
  - E.g. from *multiplicative* to *additive* dependency
    in the size of the data?

$$O(2^k \text{ poly}(k)\ m) \quad \rightarrow \quad O(2^k \text{ poly}(k) + \text{poly}(k)\ m)$$

Thank you!

http://dx.doi.org/10.1137/1.9781611973754.10

https://github.com/pkaski/motif-search