Engineering motif search for large graphs

Andreas Björklund
Lund University

Łukasz Kowalik
Warsaw University

Petteri Kaski
Aalto University, Helsinki

Juho Lauri
Tampere University of Technology

Simons Institute for the Theory of Computing
Thursday 5 November 2015
Tight results

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

Pattern (query)

Task (search for matches to query)
Large data (large graph)

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

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

\sim 10^{10} edges,
 arbitrary topology

(edge list representation)
Motif search

Data
Vertex-colored graph $H$ (the host graph)

Query
Multiset $M$ of colors (the motif)

Task (decision):
Is there a connected 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 $M$)
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 $k$

<table>
<thead>
<tr>
<th>Authors</th>
<th>Time</th>
<th>Year</th>
<th>Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fellows et al.</td>
<td>$O^*(\sim 87^k)$</td>
<td>2007</td>
<td>Color coding</td>
</tr>
<tr>
<td>Betzler et al.</td>
<td>$O^*(4.32^k)$</td>
<td>2008</td>
<td>Color coding</td>
</tr>
<tr>
<td>Guillemot &amp; Sikora</td>
<td>$O^*(4^k)$</td>
<td>2010</td>
<td>Multilinear detection</td>
</tr>
<tr>
<td>Koutis</td>
<td>$O^*(2.54^k)$</td>
<td>2012</td>
<td>Constrained multilin.</td>
</tr>
<tr>
<td>Björklund et al.</td>
<td>$O^*(2^k)$</td>
<td>2013</td>
<td>Constrained multilin.</td>
</tr>
</tbody>
</table>

“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 $S_{i1}, S_{i2}, \ldots, S_{it}$ with $S_{i1} \cup S_{i2} \cup \cdots \cup S_{it} = \{1, 2, \ldots, n\}$?

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
Are tight algorithms useful, in practice?
Are tight algorithms useful, \textit{in practice}?

For GRAPH MOTIF, can we engineer an implementation that scales to \textit{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 $P_{H,k}(x,y)$

fast evaluation algorithm for $P_{H,k}(x,y)$

2) match colors with motif

... multilinear monomials *whose colors match motif*

randomized detection with $2^k$ evaluations of $P_{H,k}(x,y)$
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 $P_{H,k}(x,y)$

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

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

Given values to the variables $x,y$, the value $P_{H,k}(x,y)$ 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 \(u \in V(H)\)

\[ P_{1,u}(x, y) = x_u \]

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

\[ P_{l,u}(x, y) = \sum_{v \in N_H(u)} y_{l,(u,v)} \sum_{l_1+l_2=l, l_1, l_2 \geq 1} P_{l_1,u}(x, y)P_{l_2,v}(x, y) \]

Finally, take the sum over all root vertices

\[ P(x, y) = \sum_{u \in V(H)} P_{k,u}(x, y) \]
Rand. algorithm for motif search (decision)

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

• Requires $2^k$ evaluations of $P_{H,k}(x, y)$, 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

• Latency
  - hardware and software prefetching
  - hide latency with enough instructions “in flight”
Evaluating $P_{H,k}(x,y)$

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

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

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

$$P_{\ell,u}(x, y) = \sum_{v \in N_H(u)} y_{\ell,(u,v)} \sum_{\substack{l_1+l_2=\ell \\ l_1,l_2 \geq 1}} P_{l_1,u}(x, y)P_{l_2,v}(x, y)$$

Finally, take the sum over all root vertices

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

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

Multithreading over vertices $u$ (layer $l$ fixed)
Iteration, for all \( l = 2, 3, \ldots, k \) and all \( u \in V(H) \)

\[
P_{l,u}(x, y) = \sum_{v \in N_H(u)} y_{l,(u,v)} \sum_{\begin{array}{c} l_1 + l_2 = l \\ l_1, l_2 \geq 1 \end{array}} P_{l_1,u}(x, y)P_{l_2,v}(x, y)
\]

for(index_t l1 = 1; l1 < l; l1++) {
    line_t pull1, pv12;
    index_t l2 = l-l1;
    index_t i_v_l2 = ARB_LINE_IDX(b, k, l2, v);
    LINE_LOAD(pv12, d_s, i_v_l2); // data-dependent load
    index_t i_u_l1 = ARB_LINE_IDX(b, k, l1, u);
    LINE_LOAD(pull1, 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;
    LINE_MUL(p, pull1, pv12); // load (for next vertex v)
    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(%r10,%rcx), %rcx
vmovdqa %xmm1, %xmm7
vextracti128 $0x1,  %ymm1, %xmm1
vpclmulqdq $0,    %xmm6, %xmm0
vpclmulqdq $0,    %xmm7, %xmm1
vmovdqa %xmm6, %xmm0
vinsertil28 $0x1,  %xmm6, %xmm3
vpclmulqdq %17,   %xmm6, %xmm1
vpunpcklqdq %ymm0, %ymm3, %ymm1
vpunpckhqdq %ymm0, %ymm3, %ymm3
vmovdqa %xmm5, %xmm7
vpsrlq   %60,   %ymm0
vextracti128 $0x1,  %ymm2, %ymm0
vpclmulqdq %17,   %ymm2, %ymm1
vpclmulqdq $0,    %ymm3, %ymm0
vmovdqa %ymm6, %ymm0
vpsrlq   %61,   %ymm1
vextracti128 $0x1,  %ymm4, %ymm3
vpclmulqdq %17,   %ymm4, %ymm1
vpclmulqdq $0,    %ymm5, %ymm0
vmovdqa %ymm7, %ymm0
vpsrlq   %63,   %ymm1
cmpq    %rax,  %rdi
vpxor   %ymm0, %ymm2
vpsrlq  %63,   %ymm3, %ymm0
```

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]

Bit-packed $8 \times \text{GF}(2^{64})$

Small-memory node

$k = 5$
Edge-linear scaling

Bit-sliced $32 \times \text{GF}(2^8)$

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

<table>
<thead>
<tr>
<th>Vertices</th>
<th>Edges</th>
<th>Input</th>
<th>Preprocess</th>
<th>Decision</th>
<th>Total</th>
<th>Peak memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000000000</td>
<td>20000000004</td>
<td>2330 s</td>
<td>1937 s</td>
<td>3163 s</td>
<td>7452 s</td>
<td>693 GiB</td>
</tr>
<tr>
<td>1000000000</td>
<td>10000000004</td>
<td>1057 s</td>
<td>987 s</td>
<td>1545 s</td>
<td>3599 s</td>
<td>347 GiB</td>
</tr>
<tr>
<td>500000000</td>
<td>5000000004</td>
<td>492 s</td>
<td>407 s</td>
<td>770 s</td>
<td>1673 s</td>
<td>174 GiB</td>
</tr>
<tr>
<td>250000000</td>
<td>2500000004</td>
<td>237 s</td>
<td>183 s</td>
<td>376 s</td>
<td>799 s</td>
<td>87 GiB</td>
</tr>
<tr>
<td>125000000</td>
<td>1250000004</td>
<td>112 s</td>
<td>90 s</td>
<td>182 s</td>
<td>386 s</td>
<td>44 GiB</td>
</tr>
<tr>
<td>62500000</td>
<td>625000004</td>
<td>55 s</td>
<td>43 s</td>
<td>88 s</td>
<td>187 s</td>
<td>22 GiB</td>
</tr>
<tr>
<td>1000000000</td>
<td>20000000004</td>
<td>2040 s</td>
<td>1830 s</td>
<td>2915 s</td>
<td>6805 s</td>
<td>623 GiB</td>
</tr>
<tr>
<td>500000000</td>
<td>10000000004</td>
<td>939 s</td>
<td>816 s</td>
<td>1430 s</td>
<td>3196 s</td>
<td>312 GiB</td>
</tr>
<tr>
<td>250000000</td>
<td>5000000004</td>
<td>467 s</td>
<td>409 s</td>
<td>704 s</td>
<td>1586 s</td>
<td>156 GiB</td>
</tr>
<tr>
<td>125000000</td>
<td>2500000004</td>
<td>221 s</td>
<td>182 s</td>
<td>343 s</td>
<td>749 s</td>
<td>78 GiB</td>
</tr>
<tr>
<td>62500000</td>
<td>1250000004</td>
<td>109 s</td>
<td>88 s</td>
<td>165 s</td>
<td>363 s</td>
<td>39 GiB</td>
</tr>
</tbody>
</table>

- **Decision algorithm runtime**
- **Convert from edge list to adjacency list**
- **Generate random regular input** (in edge list format)
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
• 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?

  \[ O(2^k \text{poly}(k) \ m) \rightarrow O(2^k \text{poly}(k) + \text{poly}(k) \ m) \]
Thank you!

http://dx.doi.org/10.1137/1.9781611973754.10
https://github.com/pkaski/motif-search