The tree-width of C

Philipp Klaus Krause, Lukas Larisch

June 3, 2015
# Table of Contents

1. Tree-Decomposition
2. Thorup’s Algorithm
3. Bound
4. Experimental Results
5. Conclusion
Table of Contents

1. Tree-Decomposition
2. Thorup’s Algorithm
3. Bound
4. Experimental Results
5. Conclusion
```c
int binomial_coefficient(int n, int k) {
    int i, delta, max, c;
    if(n < k)
        return(0);
    if(n == k)
        return(1);
    if(k < n - k) {
        delta = n - k;
        max = k;
    } else {
        delta = k;
        max = n - k;
    }
    c = delta + 1;
    for(i = 2; i <= max; i++)
        c = (c * (delta + i)) / i;
    return(c);
}
```
int binomial_coefficient(int n, int k)
{
    int i, delta, max, c;
    if(n < k)
        return(0);
    if(n == k)
        return(1);
    if(k < n - k)
    {
        delta = n - k;
        max = k;
    }
    else
    {
        delta = k;
        max = n - k;
    }
    c = delta + 1;
    for(i = 2; i <= max; i++)
    {
        c = (c * (delta + i)) / i;
    }
    return(c);
}

int binomial_coefficient(int n, int k)
{
    int i, delta, max, c;
    if(n < k)
        return(0);
    if(n == k)
        return(1);
    if(k < n - k)
    {
        delta = n - k;
        max = k;
    }
    else
    {
        delta = k;
        max = n - k;
    }
    c = delta + 1;
    for(i = 2; i <= max; i++)
    {
        c = (c * (delta + i)) / i;
    }
    return(c);
}
Given a graph $G$ a \textit{tree-decomposition} of $G$ is a pair $(T, \chi)$ of a tree $T$ and a mapping $\chi: V(T) \to 2^{V(G)}$:

- $\bigcup_{i \text{ node of } T} \chi(i) = V(G)$,
- $\forall \{x, y\} \text{ edge in } V(G): \exists i \text{ node of } T: x, y \in \chi(i)$,
- $\forall x \in V(G): \{i \text{ node of } T | x \in \chi(i)\}$ is connected in $T$.

\footnote{Halin 1967, Robertson and Seymour 1983}
The width of a tree-decomposition \((T, \chi)\) is
\[
\max\{|\chi(i)| \mid i \text{ node of } T\} - 1.
\]
The tree-width \(tw(G)\) of a graph \(G\) is the minimum width over all tree decompositions of \(G\).

Many problems that are NP-hard can be solved efficiently when restricted to graphs of bounded tree-width\(^2\).

This includes important problems in compiler construction.

\(^2\)Courcelle, 1990
For some important problems in compiler construction, there are algorithms based on tree-decompositions:

- Register allocation\(^3\).
- Flow analysis\(^4\).
- Bank selection instruction placement\(^5\).
- Redundancy elimination.

Some of them are implemented in SDCC, a free C compiler for embedded systems. Lower tree-width results in lower runtime or better results.

---

\(^4\) Alstrup et alii 2000
\(^5\) K. 2013
Let $\ell \in \mathbb{N}$ be fixed. A program is called $\ell$-structured if its control-flow graph has tree-width at most $\ell$.

- Algol and Pascal without goto: 2-structured \textsuperscript{6}.
- Modula-2: 5-structured \textsuperscript{6}.
- Ada without goto and labeled loops: 6-structured \textsuperscript{7}.
- Java without labeled break and continue: 6-structured \textsuperscript{8}.
- C without goto: Previously considered to be 6-structured \textsuperscript{6}.

\textsuperscript{6}Thorup 1998
\textsuperscript{7}Burgstaller et alii 2004
\textsuperscript{8}Gustedt et alii 2002
Let $n \in \mathbb{N}$ and $I \subseteq [n]^2$. An *I-chain* from $i$ to $j$ is a sequence of pairs $(i_1, j_1), \ldots, (i_l, j_l) \in I$ such that for all $k < l$ holds: $i_k < i_{k+1} < j_k < j_{k+1}$. An I-chain from $i$ to $j$ is *maximal*, if there is neither an I-chain from some $i' < i$ to $j$ nor an I-chain from $i$ to some $j' > j$. 
Let $I$ be the iCodes of a program in the order they occur. Let $S$ be the symmetrical closure of $I$. Thorup’s algorithm uses maximal $I$-chains and maximal $S$-chains. However, C code can be written such that there is only one maximal $I$-chain and $S$-chain, so the chains yield no useful information. This results in Thorup’s algorithm giving tree-decompositions of arbitrarily large width even when the tree-width is small.
Counterexample

```c
int a1, a2, a3, a4;
int x, b1, b2, b3, b4;

void c(void)
{
    if (a1)
    {
        if (a2)
        {
            if (a3)
            {
                x++;
            }
            else
                b3++;
        }
        else
            b2++;
    }
    else
        b1++;
}
```
Table of Contents

1 Tree-Decomposition

2 Thorup’s Algorithm

3 Bound

4 Experimental Results

5 Conclusion
For every C function, that contains at most \( g \) labels targeted by `goto` we can find a tree-decomposition of width at most \( k = 7 + g \) in linear time.

Proof: Constructive algorithm that recurses on the blocks
For every $k \geq 7$, there is a C function that has tree-width at least $k$ and at most $g = k - 7$ labels targeted by goto.

Proof: C function that contains a model of $K_8$, tree-width can be increased by 1 by adding another node with a label, which we can reach from all existing nodes via goto.
Tightness of Bound

int x0, x1, x2, x3, x4, x5, x6, x7, x8, x9, x10, x11, x12, x13, x14, x15, x16, x17;

void g(void);

void f(void)
{
    switch(x0)
    {
    case 0:
        if(x1)
            return;
        while(x2)
            {
                case 1:
                    if(x3)
                        break;
                    else if(x4)
                        return;
                    if(x5)
                        {
                            if(x6 || x7)
                                {
                                    case 2:
                                        if(x8)
                                            break;
                                        else if(x9)
                                            return;
                                        else if(x10)
                                            continue;
                                }
                            else
                                {
                                    case 3:
                                        if(x11)
                                            break;
                                        else if(x12)
                                            return;
                                        else if(x13)
                                            continue;
                                }
                    }
                case 4:
                    if(x14)
                        break;
                    else if(x15)
                        return;
                    else if(x16)
                        continue;
            }
        case 5:
            if(x17)
                break;
    g();
    default:
        ;
    }
}
Table of Contents

1. Tree-Decomposition
2. Thorup’s Algorithm
3. Bound
4. Experimental Results
5. Conclusion
Tried various heuristic and preprocessing approaches
Preprocessing followed by the fill-in heuristic followed by triangulation minimization turned out to perform best

<table>
<thead>
<tr>
<th></th>
<th>#CFGs</th>
<th>avg. width</th>
<th>max. width</th>
<th>avg. time[ms]</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Thorup</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>stdlib</td>
<td>142</td>
<td>2.430</td>
<td>11</td>
<td>0.0872</td>
</tr>
<tr>
<td>Contiki</td>
<td>1082</td>
<td>2.296</td>
<td>22</td>
<td>0.1105</td>
</tr>
<tr>
<td>Fuzix</td>
<td>529</td>
<td>2.767</td>
<td>29</td>
<td>0.1113</td>
</tr>
<tr>
<td>Whetstone</td>
<td>7</td>
<td>1.571</td>
<td>3</td>
<td>0.2326</td>
</tr>
<tr>
<td>Dhrystone</td>
<td>15</td>
<td>2.200</td>
<td>7</td>
<td>0.1272</td>
</tr>
<tr>
<td>Coremark</td>
<td>42</td>
<td>2.310</td>
<td>8</td>
<td>0.1293</td>
</tr>
<tr>
<td><strong>PP+FI+TM</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>stdlib</td>
<td>142</td>
<td>1.852</td>
<td>4</td>
<td>0.0666</td>
</tr>
<tr>
<td>Contiki</td>
<td>1082</td>
<td>1.712</td>
<td>7</td>
<td>0.0929</td>
</tr>
<tr>
<td>Fuzix</td>
<td>529</td>
<td>1.966</td>
<td>6</td>
<td>0.0742</td>
</tr>
<tr>
<td>Whetstone</td>
<td>7</td>
<td>1.429</td>
<td>2</td>
<td>0.3166</td>
</tr>
<tr>
<td>Dhrystone</td>
<td>15</td>
<td>1.667</td>
<td>2</td>
<td>0.2061</td>
</tr>
<tr>
<td>Coremark</td>
<td>42</td>
<td>1.643</td>
<td>3</td>
<td>0.1507</td>
</tr>
</tbody>
</table>
Benchmark scores

![Graph showing relative score changes for STM8 benchmarks Whetstone, Dhrystone, and Coremark.]
Control-flow graphs of C programs have bounded tree-width
Our bound is tight
Thorup’s approach is flawed
There are well-working approaches for obtaining tree-decompositions
In SDCC, Thorup’s approach will be replaced after the 3.5.0 release