1 Computing the Chromatic Polynomial in Small Space

A *graph* is a set of two distinct abstract units called *vertices* and *edges*; an edge being a connection between two vertices. The vertices are contained in a vertex set V, and we say that the graph has *order* n if $|V| = n$. Similarly, we have an edge set E, and say that the graph has *size* $m = |E|$; the maximum size of a graph of order n is $\binom{n}{2} = n(n-1)/2$. We write a graph G as $G = (V, E)$. An edge $vw \in E$ implies $v \in V$ and $w \in V$, where v and w are the vertices connected by vw; v and w are said to be *adjacent*. In this report, edges are undirected, have multiplicity *leq*1 and are never loops. An example graph is found in figure 1 [file ms-w-caption.svg].

A *proper q-colouring* of a graph $G = (V, E)$ is a mapping $\sigma : V \to [q]$ where $[q] = \{1, 2, \ldots, q\}$ such that $\sigma(v) \neq \sigma(w)$ for each $vw \in E$. In other words, σ is an assignment of a *colour* to each vertex v such that no two adjacent vertices get the same colour. This problem is known as *graph colouring*, and is one of the canonical NP-hard problems, with practical applications for register allocation, scheduling and pattern matching.

The number of ways to q-colour G is $P(G, q)$, and the *chromatic polynomial* $\chi_G(t)$ of a graph G passes through each point $(q, P(G, q))$. In other words, it *counts* the number of ways to colour G for any amount of colours. In particular, the *chromatic number* is the smallest c for which $\chi_G(c) > 0$. The polynomial $\chi_G(t)$ is of high interest in the field of algebraic graph theory, as one of the main graph invariants. An application for the chromatic polynomial exists in statistical physics, where it occurs as the zero-temperature limit of the antiferromagnetic Potts model.

In 2011, Björklund, Husfeldt, Kaski and Koivisto presented an algorithm to compute the chromatic polynomial in time $O^*(2^n)$ and space $O^*(1.2916^n)$, referred to here as the **BHKK** algorithm. In this paper, we explore the practical performance of this algorithm, as compared to an existing deletion-contraction algorithm by Haggard, Pearce and Royle from 2010 (called HPR).

2 Results

In general, it can be said that for orders below 20, HPR will nearly always outperform BHKK, but on average, we show in figure 2 [file bhkk-hpr-rt.svg] that BHKK is faster for graphs of order > 21. In particular, the memory consumption, see figure 3 [file bhkk-hpr-rss.svg], is *considerably* lower; from about 11% at $n = 19$ to 0.4% at $n = 23$.

Graph size also matters, as we see from figures 4 and 5 [files bhkk-size-rt.svg and bhkk-size-rss.svg]. In fact, even for slightly smaller graphs than order 22, BHKK can still be a better choice depending on the graph size. This should however not be contributed to the fact that BHKK scales very well with increasing size (even though it does), but to the fact that HPR scales terribly bad for mid-range sizes.

3 Parallelization

The BHKK algorithm is designed in a way that makes it a very good candidate for a parallelized implementation. On the testing machine we have access to 12 parallel threads, and the previously discussed results were all the parallelized version run on that machine. But how much did the parallelization improve our results? This is shown in figure 6 [file pll-rt.svg], where we can see that parallelization allows us to increase the order of about 2 and terminate in the same amount of time. The `0.3` version is parallelized while `0.1` is not. We do pay some memory for this, see figure 7 [file pll-rss.svg], but as the time gain is so significant, it would almost always be a given to choose the parallelized version.