Analysis

We can use a theorem of Wilf to solve recurrence relations given in the previous class.

Theorem 1.4.1 (Wilf '91). Let a sequence $\{x_n\}$ satisfy a recurrent inequality of the form

$$x_{n+1} \leq b_0 x_n + b_1 x_{n-1} + \ldots + b_p x_{n-p} + G(n)(n \geq p)$$

where $b_i \geq 0$ for all i and $\sum b_i > 1$. Further, let c be the positive real root of the equation $c^{p+1} = b_0 c^p + \cdots + b_p$. Finally, suppose that $G(n) = o(c^n)$. Then, for every fixed $\epsilon > 0$, we have $x_n = O((c + \epsilon)^n)$.

So, it comes down to root finding!

Example

In the first algorithm for independent set, we got a recurrence relation of the form:

$$T(n) \leq T(n-1) + T(n-2) + \Theta(n^2)$$

Rewriting it to match Wilf's theorem, we get

$$T(n+1) = x_{n+1} =\leq x_n + x_{n-1} + G(n)$$

Hence, the *characteristic polynomial* of this recurrence relation is

$$c^2 = c + 1$$

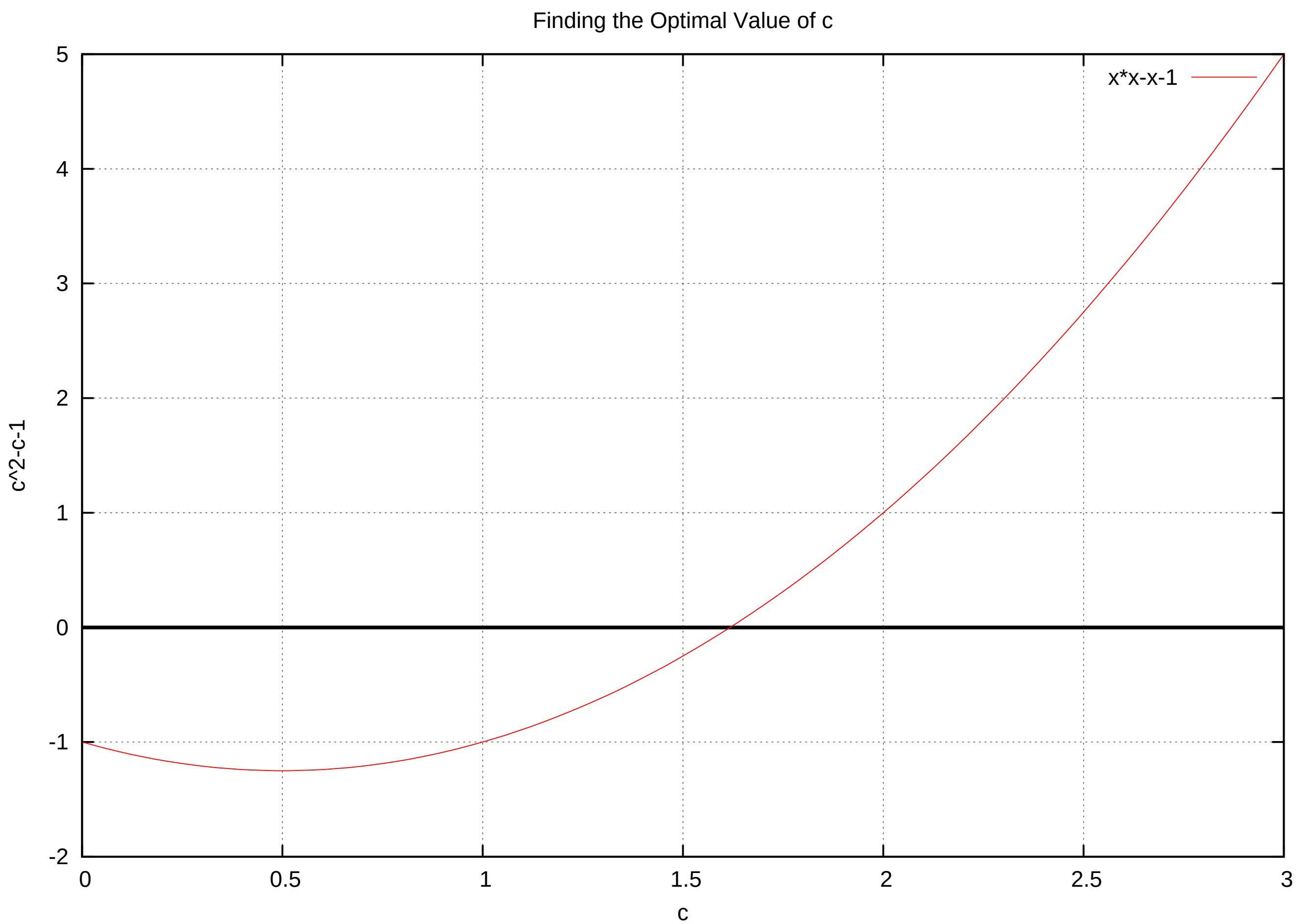
Solving for c

Another way of looking at this equation is to rewrite it as

$$c^2 - c - 1 = 0$$

and solve for c.

Graph of the Characteristic Equation



Cont'd

The positive real root of this equation is $c = 1.618...$ (this can be solved via numerical analysis). Hence,
$T(n) = O((1.618 + \epsilon)^n) = O(1.619^n)$ by Theorem 1.4.1.

Example #2

The improved algorithm had a recurrence relation of the form:

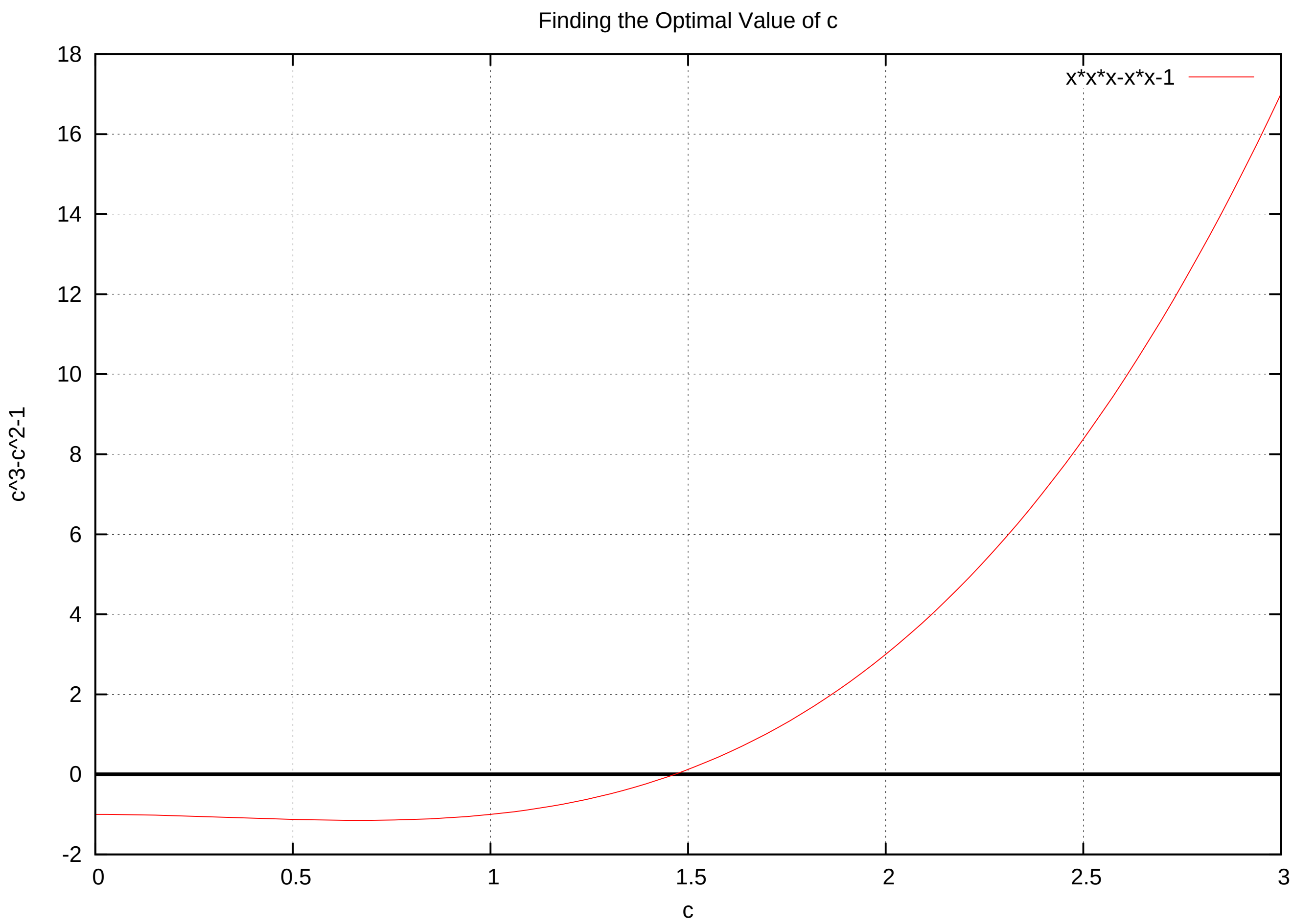
$$T(n) \leq T(n-1) + T(n-3) + \Theta(n^2)$$

Rewriting this, we get.

$$T(n+1) = x_{n+1} =\leq x_n + x_{n-2} + G(n)$$

So, the characteristic equation is $c^3 = c^2 + 1$. Hence, we need to solve the equation $c^3 - c^2 - 1 = 0$.

Graph of the Characteristic Equation



Doing Real Root Finding

It's pretty hard to see from Gnuplot, but we know that $c \leq 1.46558$. (Because $c^3 - c^2 - 1 > 0$ for this value of c.) Then, we can set $\epsilon = 0.003...$ to get an $O(1.47^n)$ run time for the previous algorithm.

A bit of Numerical Analysis

```
//
// real root finding --- This is only so accurate.
//
#include <iostream>
using namespace std;
double bisect_root() {
  // c^3 -c*c -1
  double l = 0; // c^3-c^2-1 = -1
  double u = 2; // c^3-c^2-1 = 3 (8-4-1)
  double eps = 0.0000001;
  while ((u-l)> eps) {
    double m = (u+l)/2.0;
    double p = m*m*m - m*m -1.0;
    if (p>0) {
      u=m;
    } else {
      l=m;
    }
  }
  return u; // Upper bound l<=c<=u
}
```

Cont'd

```
int main() {
  cout.precision(20);
  double br = bisect_root();
  cout << br*br*br - br*br -1 << endl;
  cout << br << endl;
}
```

Improving the run time for Independent Set

It turns out that if $deg(v) \leq 2$ for each vertex in G, we can compute the maximum independent set in polynomial time. So, we can improve the recurrence relation to $T(n) \leq T(n-1) + T(n-4) + \Theta(n^2)$. The characteristic polynomial for this recurrence relation is $c^4 = c^3 + 1$. Solving this for c gives $c = 1.380277.....$