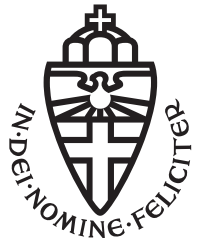
Radboud University Nijmegen



Faculty of Science

Probabilistic Model of Traversing Trees

A simplified view on Elliptic Curve Prime Proving

Name:	Vincent Koppen
Student number:	4076028
Study:	Bachelor Mathematics
Supervisors:	Dr. W. Bosma
	Prof. dr. E.A. Cator

25 August 2014

Contents

1 Preface

Ever since elliptic curves were briefly mentioned during the Cryptography course I was interested in what they exactly were and what they could be used for. Therefore when one of the proposal subjects for the bachelor thesis included elliptic curves, I was immediately interested. Elliptic curves have all kind of applications of which cryptography is one. For instance, by using elliptic curves it is possible to offer the same security as RSA but with a much smaller key.
In this thesis we will take a look at prime proving using elliptic curves, Elliptic Curve Primality Proving (ECPP), which currently is the fastest known algorithm to test the primality of general numbers. Gyöngyvér Kiss, a Ph.D. student under supervision of Wieb Bosma and Antal Járai, has made her own implementation of this algorithm. This thesis tries to find a possible optimization for her implementation.

At first we looked at the implementation of ECPP. Unfortunately we came to the conclusion that it was probably too hard to make an actual model for this implementation and we switched to a somewhat simplified model. This model still had ties with the actual implementation, since in this model certain costs are given to actions in the implementation of ECPP. We want to minimize the total cost. To accomplish this, we made a script in R that calculated this cost for different starting parameters and thereby gave us insight in the behavior of the cost. But since probability was involved in calculating the cost, the results could vary quite a bit for the same starting parameters. We therefore wanted to determine the expected cost instead, which would lead to even more insight into the behavior of the cost. With help from Henk Don, Post-doc in the Applied Stochastics group, we managed to get a very promising expected cost and can thereby, for given starting parameters, give an clear answer to what choice should be made.

This thesis will first take a look at what elliptic curves are and what they are used for in Elliptic Curve Primality Proving. However, since we switched to a simplified model in which elliptic curves aren't involved anymore, this part will be somewhat superficial. After that we will take a look at our simplified model, first by actually calculating the cost and the results this gives. Lastly we will describe how to calculate the expected cost.

2 Elliptic Curve Prime Proving

In this section some insight is given into Elliptic Curve Prime Proving (ECPP). However some proofs will be left for the reader, because we switched to a different model and the proofs would be too much of a detour.

2.1 Elliptic Curves

This subsection makes use of [1].

2.1.1 Definition

The analyzed program is an application of ECPP. Therefore some knowledge about elliptic curves is needed. An **elliptic curve** E over a field K is given by a formula of the form:

$$y^2 = x^3 + ax + b$$

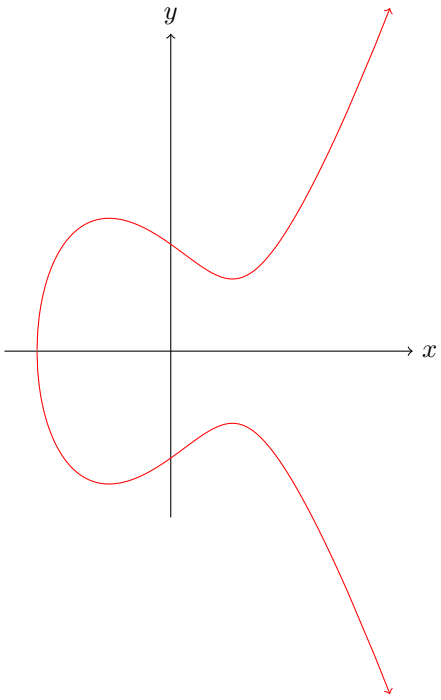
with $a, b \in K$. Its **discriminant**, Δ, is given by:

$$\Delta = 4a^3 + 27b^2$$

and should not be equal to zero. The final requirement is that E contains a point $\mathcal{O}$ "at infinity". It's now possible to define the set of points on E over K:

$$E = \{(x, y) : x, y \in K, y^2 = x^3 + ax + b\} \cup \{\mathcal{O}\}.$$

Example 2.1.1. $E : y^2 = x^3 - 2x + 2$ over $\mathbb{R}$



Notation 2.1.1. $E : y^2 = x^3 + ax + b$ is the elliptic curve represented by $y^2 = x^3 + ax + b$, this means that $E = \{(x, y) : y^2 = x^3 + ax + b\} \cup \{\mathcal{O}\}$. Therefore $E = \{(x, y) : y^2 = x^3 - 2x + 2\} \cup \{\mathcal{O}\}$ in example 2.1.1.

2.1.2 Addition on elliptic curves

The point $\mathcal{O}$ is needed to define addition on elliptic curves. Let $P = (x_1, y_1), Q = (x_2, y_2)$, $P, Q \neq \mathcal{O}$, be points on an elliptic curve E. Then $P \oplus Q = (x_3, y_3)$ can be determined algebraically:

$$x_3 = \lambda^2 - x_1 - x_2$$

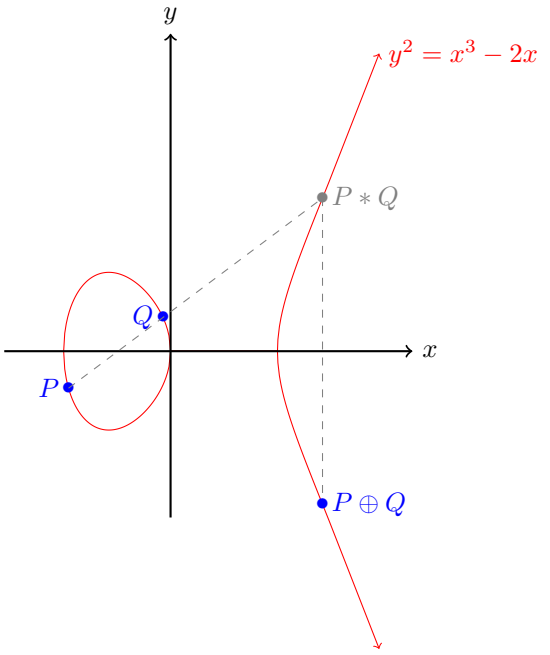
$$y_3 = \lambda(x_1 - x_3) - y_1$$

where λ is given by:

$$\lambda = \frac{y_2 - y_1}{x_2 - x_1}$$

At first this doesn't look very straightforward, but drawing gives a much better impression. $P \oplus Q$ can be determined by drawing the line through P and Q, that will (almost) always intersect with E in a third point, which we denote by $P * Q$. Then $P \oplus Q$ is the reflection of $P * Q$ in the x-axis.

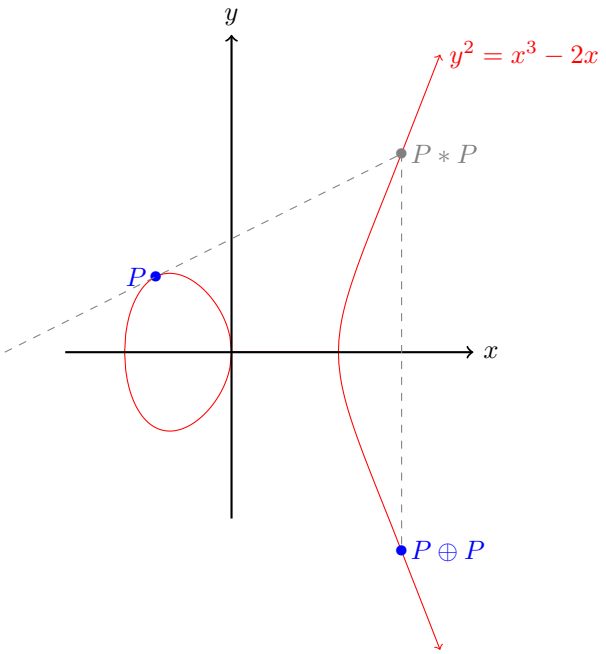
Example 2.1.2. Addition of $P, Q \in E$, with $E : y^2 = x^3 - 2x$ over $\mathbb{R}$. (Notice that the shape is different from example 2.1.1, but the method will work for both.)



As said before, the line through P and Q will almost always intersect a third time. But sometimes it won't, namely when the line through P and Q is parallel to the y-axis. Take for example the points $P*Q, P\oplus Q$ from example 2.1.2. It's clear that the line through $P*Q$ and $P\oplus Q$ won't intersect with E for a third time (algebraically we see that λ isn't defined in this case, because we divide by zero). So we say that this line intersects with E in the point $\mathcal{O}$,which can be seen as a point "at infinity".

Another visually strange case is $P\oplus P$, because a line through P isn't uniquely defined. It is however possible to determine $P\oplus Q$ for points Q with $d(P,Q)<\varepsilon$ for all $\varepsilon>0$. Now let ε go to zero, then the line through P and Q will converge to the tangent to E at P. This line will cross E in exactly one point (remember that $P\oplus P=\mathcal{O}$ if the line is parallel to the y-axis). The construction of $P\oplus P$ is now done the same as before.

Example 2.1.3. $P\oplus P$ with $E: y^2=x^3-2x$ over $\mathbb{R}$.



In the next section we make use of the fact that the points on an elliptic curves form a group together with the addition defined above. However we first need to check if the group properties are indeed true for the addition on elliptic curves.

Theorem 2.1.2. *The following equations hold for the addition on elliptic curves:*

1. $P\oplus\mathcal{O}=\mathcal{O}\oplus P=P$ *(Identity)*
2. $P\oplus(-P)=\mathcal{O}$ *(Inverse)*

3. $P \oplus (Q \oplus R) = (P \oplus Q) \oplus R$ *(Associativity)*

4. $P \oplus Q = Q \oplus P$ *(Commutativity)*

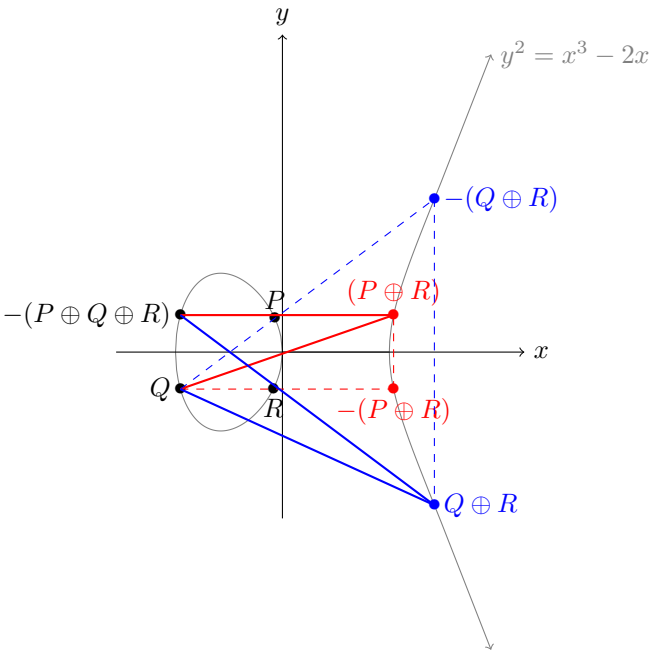
where $-P = (x, -y)$ *if* $P = (x, y)$.

Proof. I will not really prove those equalities, but make them plausible using the visual interpretation of addition on elliptic curves over $\mathbb{R}$.

1. (Identity)
   Think of $\mathcal{O}$ as the point vertically above P at $y = \infty$. Then the line through those two points will intersect with E in $-P$. So reflecting this in the x-axis gives us P, hence $\mathcal{O} \oplus P = P$. Later we will see that the addition is commutative which gives: $P \oplus \mathcal{O} = \mathcal{O} \oplus P = P$.

2. (Inverse)
   This is precisely how $-P$ is defined, because $-P$ is the reflection of P in the x-axis. So the line through P and $-P$ will we parallel to the y-axis, hence $P \oplus (-P) = \mathcal{O}$.

3. (Associativity)
   Let's use E from example 2.1.2. Choosing P, Q, R as below gives the following result:



   Because of the choice of $P, Q, R \in E$, it follows that $Q = P \oplus Q \oplus R$. Clearly $P \oplus (Q \oplus R)$, the intersection of the blue line with E, is equal

to $(P \oplus Q) \oplus R$, the intersection of the red line with E. They both deliver the same $-(P \oplus Q \oplus R)$, thus also give the same $P \oplus Q \oplus R$. So $P \oplus (Q \oplus R) = (P \oplus Q) \oplus R$.

4. (Commutativity)
   The line through P and Q stays the same in both cases, thus $P*Q = Q*P$. Hence it immediately follows that $P \oplus Q = Q \oplus P$.

□

2.1.3 Elliptic curves and (finite) groups

From the results above, it's clear that the points of an elliptic curve together with the addition form an abelian group with unit element $\mathcal{O}$.

Theorem 2.1.3. *Let $L \subset K$ be a field and suppose that an elliptic curve E over K is given by an equation of the form $E : y^2 = x^3 + ax + b$ with $a, b \in L$. Let $E(L)$ denote the set of points of E with coordinates in L, i.e.*

$$E(L) = \{(x, y) \in E : x, y \in L\} \cup \{\mathcal{O}\}.$$

Then $E(L)$ is a subgroup of E over K, thus $E(L) \leq E(K)$.

Corollary 2.1.4. *Suppose p is prime. Then $\mathbb{F}_p$ is a finite field with p elements. Suppose $E : y^2 = x^3 + ax + b$ over K, then $E(\mathbb{F}_p)$ is equal to $E : y^2 = x^3 + ax + b \mod p$, which makes $E(\mathbb{F}_p)$ a finite subgroup of $E(K)$.*

Example 2.1.4. This corollary tells us that if

$$E_1 : y^2 = x^3 + x + 1 \mod 37$$

then $E_1 = E(\mathbb{F}_{37})$ is a subgroup of E, where $E : y^2 + x + 1$, over $\mathbb{R}$.

2.2 Application of ECPP

This subsection is based on results from [2].

2.2.1 Prerequisites

Before the specific application of the ECPP can be explained, we need some theorems.
Remark: Suppose we have $n \in \mathbb{N}$ and $E : y^2 = x^3 + ax + b \mod n$. If n is prime, then $E = E(\mathbb{F}_n)$ is a group. However if n is not prime, then we are looking at an elliptic curve over a ring (instead of a field). Luckily it's possible to define a "pseudo-addition" over such a elliptic curve that behaves the same as the normal addition, but the points now won't form a group. Therefore $E(\mathbb{Z}_n)$ doesn't have to be a group if n not prime.

Theorem 2.2.1. *Let $n_0 \in \mathbb{N}$ with $gcd(n_0, 6) = 1$. Let E be an elliptic curve over $\mathbb{Z}_{n_0}$, where $gcd(n_0, \Delta(E)) = 1$, and let $m, n_1 \in \mathbb{N}$ with $n_1 | m$. Suppose that for every prime factor q of n_1 there exists $P \in E$ such that $mP = 0_E$ and $\frac{m}{q}P \neq 0_E$. Then for all prime factors p of n_0 we have $\#E[\mathbb{Z}/p\mathbb{Z}] \equiv 0 \mod n_1$.*

Corollary 2.2.2. *Suppose that the hypotheses of Theorem 2.2.1 are satisfied. Then:*

$$n_1 > (n_0^{\frac{1}{4}} + 1)^2 \Rightarrow n_0 \text{ is prime.}$$

This means that n_1 has to be slightly bigger then $\sqrt{n_0}$. The importance of this corollary will become clear in the next section.

Miller-Rabin primality test In this test a certain a is chosen, which has to meet some requirements. It is proven that for an odd composite number n at least $\frac{3}{4}$ of all the possible a prove that n is indeed composite. So in other words if you run the Miller-Rabin primality test once for an odd n and it doesn't prove that n is composite, there is a probability of at most $\frac{1}{4}$ that n is composite. Running this test again for a different a will decrease this probability to 4^{-2}. Running the test k times means the probability that n is composite is at most 4^{-k}. So n will become "more probably" prime every time the Miller-Rabin primality test is done for different a.

2.2.2 Proof of primality

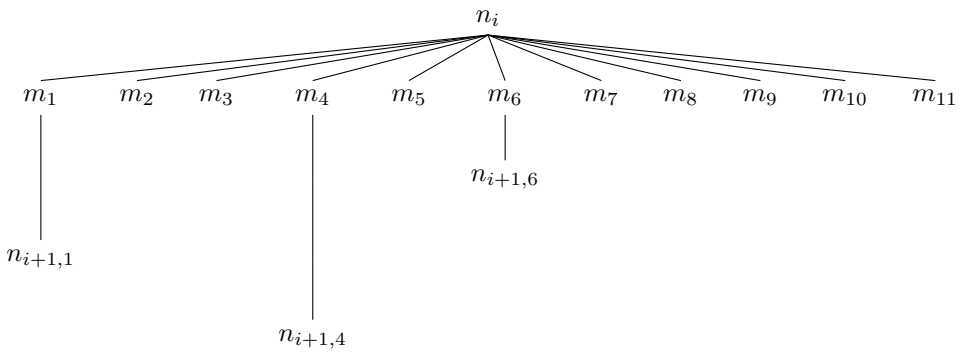
Suppose we want to proof that a probable prime N is indeed prime. This can be done using elliptic curves. The method that is used works recursively. The starting point is a probably prime $N = n_0$ and the endpoint is a n_k that's certain to be prime, for example because it's in a list of known primes. The method used gives a sequence $n_0, n_1, \ldots, n_k$ such that n_i, n_{i+1} fulfill corollary 2.2.2 (replace n_0 and n_1 with respectively n_i and n_{i+1}), which proves that $N = n_0$ is indeed prime (this still needs some work, but won't be discussed here).

Recursive step The recursive step is the most important to understand and will be described below. Note however that a lot of details are left out.

Suppose that we are at level i, this means that we already have found $n_0, n_1, \ldots, n_i$, such that every pair of consecutive numbers fulfill the conditions of corollary 2.2.2 and are all probably prime. We now want to find n_{i+1}, such that n_i, n_{i+1} fulfill corollary 2.2.2 and n_{i+1} is probable prime.
First we need to make a list of primes up to some upper bound $s = s(i)$, next make a list of elliptic curves modulo n_i with a discriminant up to a certain bound $d = d(i)$, where all the discriminants must factor into a product of primes from the prime list. All those elliptic curves represent finite subgroups of order m_j. So for different elliptic curves, we get different m. All these m are however bounded by: $n_0 - 2\sqrt{n_0} \leq m \leq n_0 + 2\sqrt{n_0}$. Let's say there is a list of these m: $m_1, m_2, \ldots, m_l$. We now hope that at least one of those m_j factors in a special way, namely $b \cdot n_{i+1,j} = m_j$, where b will factor into the product of primes

from the prime list and $n_{i+1,j}$ a probable prime according to the Miller-Rabin primality test.

Example 2.2.1. Visual view of the process. In this case m_1, m_4, m_6 appeared to factor as wanted. All the other m_j did not.



The next step is to choose one of the $n_{i+1,j}$ and add it to the list as n_{i+1}. We now have a list $n_0, n_1, \ldots, n_i, n_{i+1}$ and are at level $i + 1$.

2.2.3 Optimizing

Even when describing the process without too much detail, a lot of variables / choices appear. To name a few:

$\mathbf{s = s(i)}$
: This is one of the clearest variables. Much depends on this upper bound for the primes. Not only is it important for the discriminants we use, but it also plays a big role in determining the next n_{i+1}. If s is very big, there will (probably) be a lot more $n_{i+1,j}$ to chose from. Yet it will take much longer to find the factor b. Choosing s small will be quicker to check for a factor b, however it will also become unlikelier to find a $n_{i+1,j}$.

$\mathbf{d = d(i)}$
: For d it's almost the same as with s. Choosing d big would mean checking a lot of elliptic curves, choosing d small would result in just a few m and a small chance of finding a $n_{i+1,j}$ that fits the conditions.

Choice of $\mathbf{n_{i+1}}$
: As seen in example 2.2.1 there can be more than one candidate. So we need to determine which of the possibilities is the best. The most straightforward choice would be to pick the smallest $n_{i+1,j}$. However it's not clear if maybe some numbers make better choice than others. It is also possible that all the $n_{i+1,j}$ are almost as big as n_i. In this case we should ask ourselves whether we choose such a $n_{i+1,j}$ or decide to keep searching to find a better candidate.

No $n_{i+1,j}$

It's also possible that no suitable $n_{i+1,j}$ are found. In this case there is an important choice to make. There are two possibilities:

More effort One of the possibilities is to make more effort to find a suitable $n_{i+1,j}$. This can be done by make s bigger or trying different elliptic curves, thus making d bigger. This will however result in some sort of new upper bound, because it isn't possible to keep changing s and d. This new bound should be the real upper bound. If this bound is reached s and d won't change and we have to backtrack, as explained below.

Backtrack Instead of trying it for this n_i, it's also possible to go back. Suppose that originally there were three $n_{i,j}$ and we chose one of those as n_i. It's possible to go back and select one of the $n_{i,j}$ that hasn't been tried yet. For the chosen $n_{i,j}$ we may find a suitable $n_{i+1,j}$, without the need of changing s, d.

This choice is very important in the complete process and not straightforward. It could vary a lot depending on the situation, but it's hard to get data that would give insight in this decision.

Probable prime

Every n_i for $0 \leq i \leq k$ is a probable prime. However we need to decide how certain we want to be that n_i is indeed probable prime. Is running the Miller-Rabin primality test a few times enough or do we have to run is atleast a thousand times?

Data

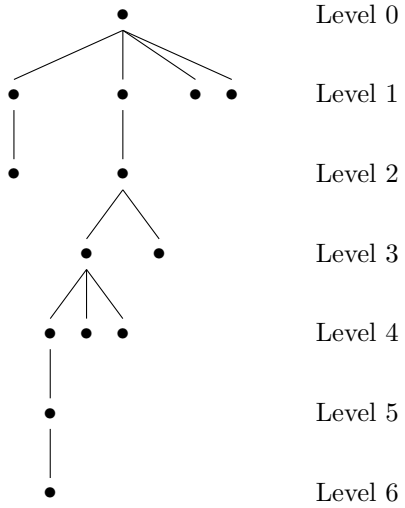
One of the biggest problems was the lack of data. In the best case scenario, we would know the complexity order of specific parts in the process. For instance the order of the Miller-Rabin test is known, but it appeared that this order was the biggest of them all (while in reality it's a (fairly) quick test). So while order does say something about the time it takes to complete a certain task, it's not the best way to determine what part of the process will actually take the longest.

3 Simplified model

3.1 The model

This model will, like the application of ECPP viewed before, work recursively. The end result of this model will be a tree that reaches a certain level.

Example 3.1.1. This is a possible output of the model. This tree reaches level 6.



In the following there will be referred to **(the number of) children on level** i; this is the same as the number of nodes on a certain level in the tree. So in the example above level 0 has 1 child, level 1 has 4 children, level 2 has 2 children, etc.

Starting point The starting step of this model is one child at level 0, the model will stop if level n is reached. This n has to be set at the beginning and in our model will almost always be $n = 300$.

Recursive step Suppose level i has k children, with $k > 0$. Let x_i denote the number of *unused* children, thus $x_i = k$. To reach level $i + 1$, we need to pick one of the children on level i. This child can be used to reach level $i + 1$. A child on level i can create x_{i+1} children on level $i + 1$, where $x_{i+1} \sim \text{Poiss}(\mu)$ and μ given. So now $x_i = k - 1$, because one child is used to determine x_{i+1}. Suppose $x_{i+1} > 0$, then level $i+1$ is reached and we are done with the recursive step. If $x_{i+1} = 0$ we will need to backtrack.

Backtracking Suppose that we are on level i with $x_i \geq 1$ and we choose one of the children. This child will now create x_{i+1} children on level $i+1$. However if $x_{i+1} = 0$ we can't continue to level $i+1$, because there aren't any children there. Now that $x_i \geq 0$ there are two possibilities:

$x_i > 0$
: While $x_i > 0$, there are still unused children on level i. So we can pick one of those unused children and determine x_{i+1} for this child. If $x_{i+1} > 0$, then we reached level $i+1$. If $x_{i+1} = 0$ again, then try again if $x_i > 0$, else $x_i = 0$ as below.

$x_i = 0$
: Since all the children are used on level i, there is no other choice than to go back to level $i-1$. Since $x_{i-1} \geq 0$, there are again the same possibilities. So if $x_{i-1} > 0$, then try to reach level $i+1$ again by determining x_i and x_{i+1}, if $x_i > 0$. If however also $x_{i-1} = 0$, then go back to level $i-2$. Repeat this process until some $x_j > 0$, with $1 \leq j \leq i$, or until level 0 is reached. In the last case, the tree is exhausted: $x_i = 0$ for all $0 \leq i \leq n$. So we will have to start over by setting $x_0 = 1$ and try again to reach level n.

Optimizing Instead of optimizing the run-time of this model, there are certain costs assigned to different parts of the model and the goal is to get some insight in how this cost behaves for different μ.

Definition 3.1.1.

- K is the cost for making a child. So every child, used or not, will cost K.
- C is the cost that has to be paid to go to the next level.

So in example 3.1.1 the step from level 2 to level 3 will cost: $C + 2 \cdot K$. Note that during backtracking it could happen that the tree was completely exhausted and we had to start over. The cost made until then will however be remembered. If $\mu \approx 1$ then there will be a lot of backtracking, so in this case C will play an important role in the total cost. If $\mu > 3$, almost no backtracking will be involved, since for $\mu = 3$:

$$P(x_i = 0) = \frac{e^{-\mu} \cdot \mu^0}{0!} = e^{-\mu} \approx 0.05.$$

On the other hand, there will be many unused children that all cost K, so in this case K will be a big influence for the total cost.

Remarks

- Since there are only two cost variables, the actual value of C, K aren't that important. Instead we can look at the ratio $C : K$. Therefore from now on we set $C = 1$ and K will vary to see the influence of different ratios.

- We will only look at $1 \leq \mu \leq 5$ because $\mu < 1$ will almost never reach level 300, so the cost will be very high except for some really absurd ratios. The same goes for $\mu > 5$, there will be too much unused children, so except if $K \approx 0$ it will not be useful to look at.
- If there are multiple children on a certain level, we will have to choose one of those. We will always pick from left to right.
- The code for the model is written in R and can be found in appendix 5.1.

3.2 Links between the model and ECPP

This new model is of course not chosen at random, there still are a lot of links between the two. The most obvious is the level structure, both work in a recursive way. It is however good to notice that in the model the levels are all of the same size, while in the application of ECPP this isn't true, as already explained. The children on level i of course correspond to the $n_{i,j}$ found using ECPP, we use the Poisson distribution to determine x_i. We actually don't know for sure if this really is the case, but since there was no real data we had to choose something. The cost we chose represents different parts of the recursive step in the application of ECPP. Globally speaking K symbolizes the time it takes to find the new $n_{i,j}$ from the m_j, and C the time it takes to find the right elliptic curves to test.

3.3 Results

It's good to mention that the actual cost for one specific μ isn't important, but the difference between the costs for different values of μ is. Thus the results below will always be for multiple μ at the same time.

3.3.1 Everything fixed except for μ

The simplest case is where we simulate the cost with parameters $C = 1, K = 15, n = 300$ and $1 \leq \mu \leq 5$.

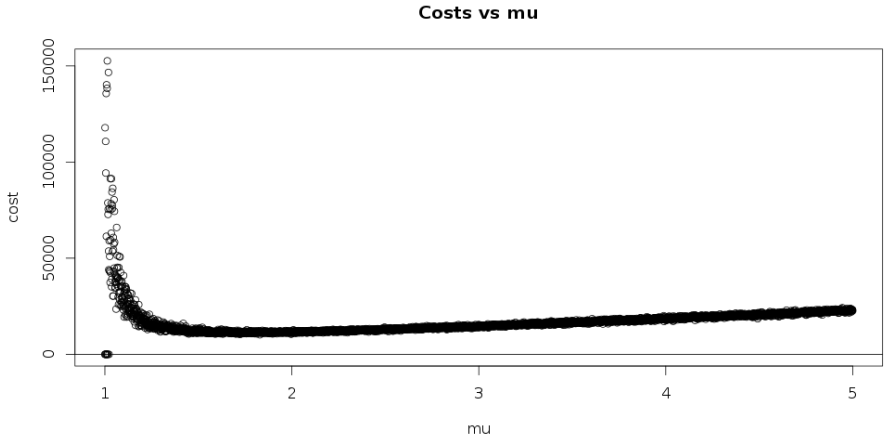


Figure 1: Simulated cost for $C = 1, K = 15, n = 300$ and $1 < \mu < 5$.

Some remarks about this plot

1. There are μ with cost zero. This of course isn't true, because the cost is atleast $n \cdot (C + K)$. The μ with cost zero are actually the most expensive of them all. Their value is changed to zero to make the scale of the plot better. If they weren't set to zero, the *cost*-axis would have a much bigger scale and a lot of detail would be lost as can be seen below:

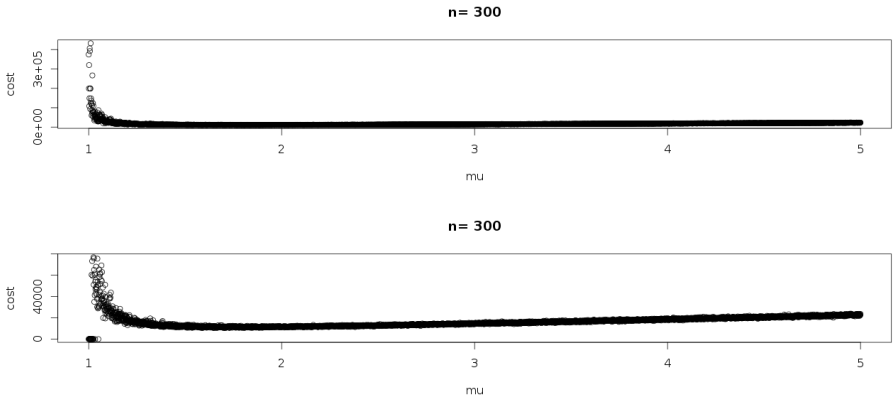


Figure 2: The upper plots show the results without editing the expensive points, while the lower plot is made after editing.

Without editing the plot looks almost like a constant line. Rescaling the

figure through editing actually shows that this is wrong, since the cost at $\mu = 5$ appears to be almost twice the cost at $\mu = 1.5$. It is of course possible to remove those points completely from the plot, but it is still of some use to know how many points are too expensive and for what μ they appear.

2. It is clear that the cheapest μ is somewhere between $1 < \mu < 2$. If $\mu \approx 1$ the cost varies too much because of all the backtracking, while $\mu \approx 2$ is too expensive because of all the unused children.

3. For $\mu > 2$ the plot looks like a straight line, which is somewhat to be expected. As said before there won't be much backtracking if $\mu > 2$, so C will be paid n times. Every level has an expected value of μ children which all cost K. So this results in:

$$cost(\mu) = n \cdot (C + K \cdot \mu).$$

Plotting this line confirms that this line is approached as μ increases.

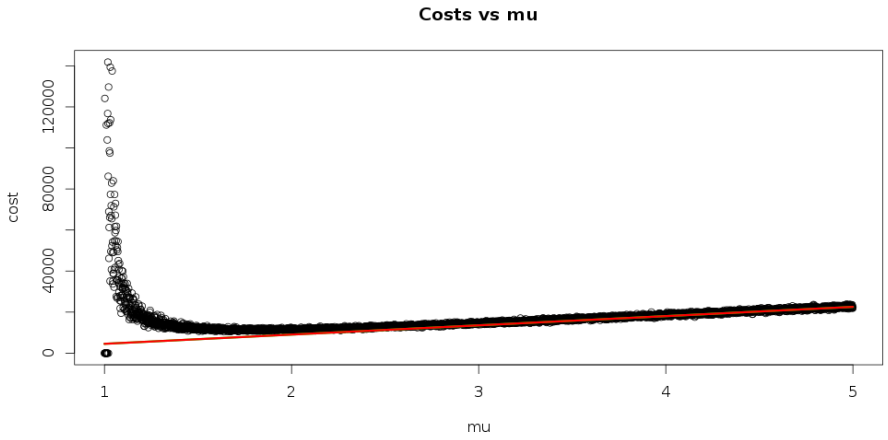


Figure 3: Same plot as figure 2, in which the red line, given by $cost(\mu) = n(C + K\mu)$, is added.

3.3.2 Importance of n

Now that the base worked I was interested in what kind of difference n makes. Usually we set $n = 300$ in our model, because this seemed like a good estimate of levels in the real program. However it's good to know what the consequences are of changing n.

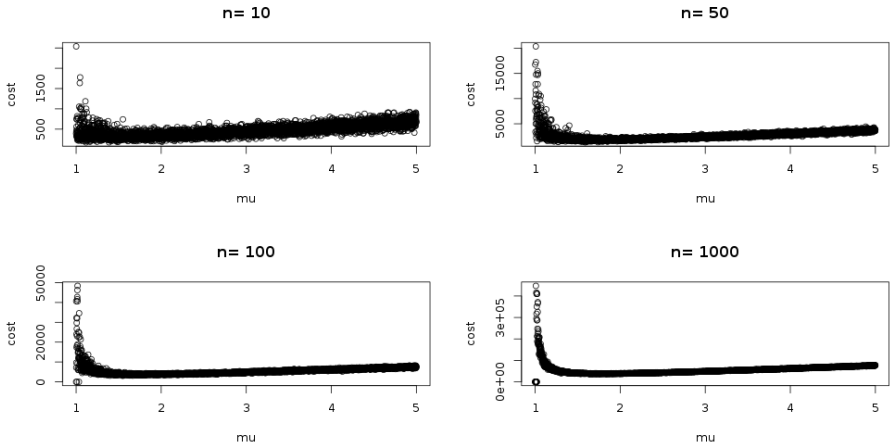


Figure 4: $C = 1, K = 15$, $1 \leq \mu \leq 5$.

For small n we see a lot of scattering and it's not really obvious what the best μ is, however for $n = 100, n = 300$ and $n = 1000$ the differences are smaller. It is nevertheless clear that for bigger n the plot becomes less scattered, especially for $1 \leq \mu \leq 2$. But since ECPP works for big primes and situations in which the conditions of corollary 2.2.2 are satisfied, n will probably never be under 100. If the actual level to be reached appears to be bigger than $n = 300$, then it will only become clearer what value of μ is perfect. However if the actual level is smaller then $n = 300$, it will become harder to give a clear answer to what the best μ is. All in all the shape of the plot doesn't really change, so while it might become less accurate it's still possible to make some educated guess.

3.3.3 Cost vector and the perfect μ

As said before, it isn't really clear what the influence is of certain choices in the application of ECPP. So until now the ratio between the costs C and K were fixed, yet there is no real reason why this should be a ratio of $1 : 15$. So instead of looking at a fixed ratio, we will look at a vector $\mathbf{K}$ which entries are different values of K, where we have chosen $0.1 \leq K \leq 20$. This will result in different ratios. Of course, this can still be plotted, but the image won't give us that much information.

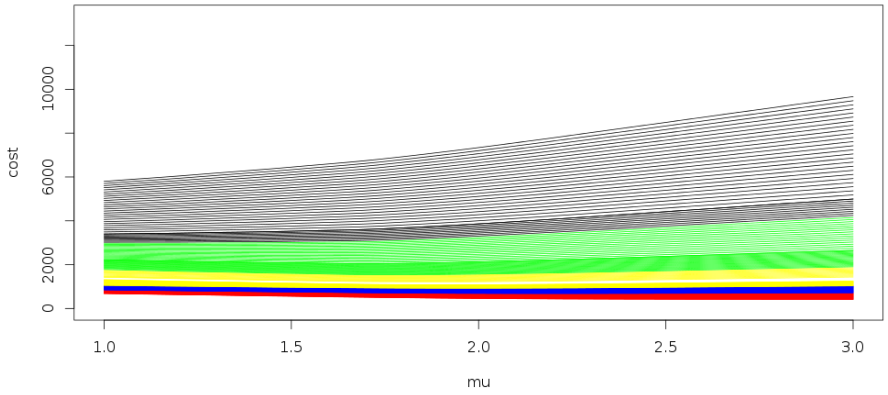


Figure 5: $C = 1$, $0.1 \leq K \leq 20$, $1 \leq \mu \leq 3$.

The colors are added to make it somewhat more visually attractive, but they still don't show what the best μ is for different K.

Best μ Luckily, the program now has a matrix with the costs for all μ and K. Therefore it's possible to select the cheapest μ for all elements of $\mathbf{K}$. Plotting those μ gives a much better understanding how μ changes for different K.

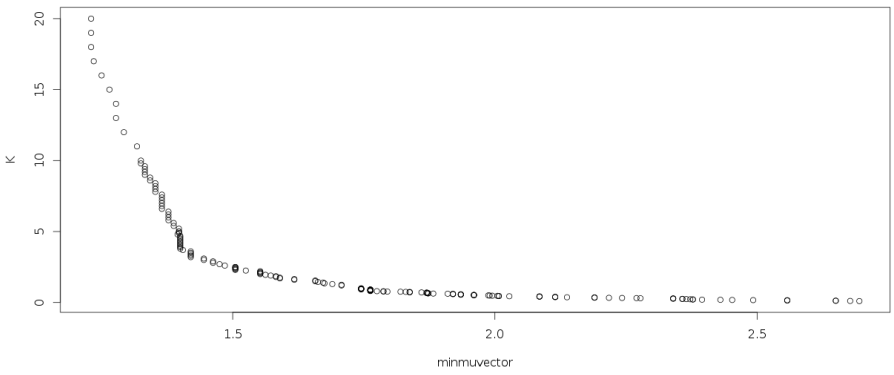


Figure 6: Plot of the cheapest μ for every value that can appear in $\mathbf{K}$.

It's now clear that K has a big influence on the best μ. Therefore it may be possible to optimize the application of ECPP a bit, if we obtain some more insight about the ratio between C and K. But since the model and ECPP vary quite a bit, we don't know this for sure. Before we can actually say something,

it would be much better to have an estimate of the cost instead of actually having to calculate the cost every time. Not only is this faster, but it gives a good estimate. Because sometimes one μ is "lucky" and is very cheap to reach level 300, while normally it's more expensive. So this could influence finding the best μ quite a bit, which could result in picking a μ that normally takes very long to reach level 300. This will be discussed in the next section.

4 Estimated cost

4.1 Some definitions

Before we can determine the estimated cost, we need some new definitions.

Definition 4.1.1. A tree is said to *reach level* n, if there is only one root needed to get to level n. So the tree will never be completely exhausted before level n is reached.

Now that we have this definition, it's possible to define two type of trees. These type of trees will play a very important role in estimating the cost.

Definition 4.1.2. A tree is said to be an X_i*-tree* if it reaches level i. If it does *not* reach level i, it's called a Y_i*-tree*.

Notation 4.1.1. From now on $\mathbb{E}(Y_i)$ will denote the expected cost for a Y_i-tree, likewise $\mathbb{E}(X_i)$ the expected cost for an X_i-tree.

Notation 4.1.2. We will denote the expected number of children on level i that are X_i-trees with x_i. Likewise y_i will denote the expected number of children on level i that are Y_i-trees.

Both x_{i-1} and y_{i-1} will depend on whether they are in an X_i-tree or a Y_i-tree. We assume that being on the root of a tree that reaches level n is the same as saying that that root will get at least one child that reaches level n, therefore $x_1 \geq 1$ in that case.

Definition 4.1.3. The probability for picking an X_i-tree the first time is denoted by P_i. The probability for picking a Y_i-tree the first time is denoted by Q_i. Note that $Q_i = 1 - P_i$.

Suppose that P_i can be calculated and there are good estimates for the cost of an X_i- and Y_i-tree, then it's possible to give a good estimate for the costs of reaching level i. So it's important to say something about all those values.

4.2 Expected values

4.2.1 P_i

It is easier to determine Q_i and calculate P_i with $P_i = 1 - Q_i$.

Theorem 4.2.1. *Let μ be the expected value for the number of children, then:*

$$Q_i = \begin{cases} 0 & \text{if } i = 0 \\ e^{-\mu} e^{\mu \cdot Q_{i-1}} & else \end{cases}$$

Proof. We distinguish two cases:

$i = 0$
: Q_0 is the probability of picking a tree that doesn't reach level 0. However, since every tree starts at level 0 this probability is equal to 0. So $Q_0 = 0$.

$i > 0$

If $i > 0$, then $i = 1$ or $i > 1$.

$i = 1$

If $i = 1$, then a tree is a Y_1-tree if it doesn't have any children on level 1, because otherwise it would reach level 1. The probability of not getting any children is $e^{-\mu}$, so $Q_1 = e^{-\mu}$, since there is always exactly one child on level 0. And this gives the same Q_1 as the formula does:

$$Q_1 = e^{-\mu} \cdot e^{\mu \cdot Q_0} = e^{-\mu}.$$

So the formula is true for $i = 1$

$i > 1$

For $i > 1$ we also look at the starting point of a Y_i-tree. The child on level 0 is now allowed to get children on level $i - 1$, however all these children should be Y_{i-1}-trees. The probability of getting j children that are all Y_{i-1}-trees is equal to $\frac{e^{-\mu} \cdot \mu^j}{j!} \cdot Q_{i-1}^j$. Summing over all possible values j will give us Q_i:

$$\begin{aligned} Q_i &= \sum_{j=0}^{\infty} \frac{e^{-\mu}\mu^j}{j!} \cdot Q_{i-1}^j \\ &= e^{-\mu} \cdot \sum_{j=0}^{\infty} \frac{(\mu \cdot Q_{i-1})^j}{j!} \\ &= e^{-\mu} \cdot e^{\mu \cdot Q_{i-1}}. \end{aligned}$$

The formula for Q_i also holds for $i > 1$, which means that it's true for all $i \in \mathbb{N}$.

□

Using this theorem it's possible to calculate both Q_i and P_i, which will be often used later on.

4.2.2 $\mathbb{E}(Y_i)$

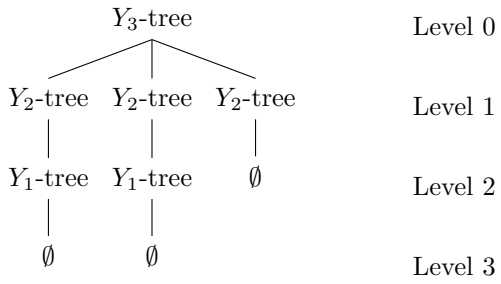
To determine the estimated cost for a Y_i-tree, it's important to say something about x_i, y_i. Since a Y_i-tree will never reach level i, the root of such a $Y - i$-tree will never get a X_{i-1}-tree as child, so $x_{i-1} = 0$. It can however get a Y_{i-1}-tree as child, if $i > 1$, which means y_{i-1} doesn't have to equal zero. It's easily seen that $x_j = 0$ for $0 \leq j < i$, however y_j can be bigger than zero as long as $j > 1$. So it's important to determine y_j given the fact that we are in a Y_i-tree.

Theorem 4.2.2. *It holds that* $y_i = Q_i \cdot \mu$.

Proof. There are μ children to be expected on level i and the probability of such a child being a Y_i-tree is Q_i. Therefore $y_i = Q_i \cdot \mu$. □

Let's look at an example of a Y_3-tree and determine its cost.

Example 4.2.1. Possible Y_3-tree



The cost can be determined by looking at the cost per level:
Level 0 $\rightarrow$ Level 1: $C + 3 \cdot K$
Level 1 $\rightarrow$ Level 2: $3 \cdot C + 2 \cdot K$
Level 2 $\rightarrow$ Level 3: $2 \cdot C$
Total cost: $6 \cdot C + 5 \cdot K$.

It's possible to generalize this example to estimate the cost of a random Y_i-tree. The cost can be determined recursively: the total cost is the cost to reach level $i + 1$ from level i, C, plus K times the number of children on level $i + 1$ plus the cost of all those individual children on level $i + 1$. Generalizing this with the expected cost results in the following theorem:

Theorem 4.2.3.

$$\mathbb{E}(Y_i) = \begin{cases} C & if\ i = 1 \\ C + y_{i-1} \cdot K + y_{i-1} \cdot \mathbb{E}(Y_{i-1}) & else \end{cases}$$

Proof. There are two cases:

1. $i = 1$
   A Y_1-tree doesn't get any children, since $Q_0 = 0$. So C will have to be paid to check if there are indeed no children, but that's all.

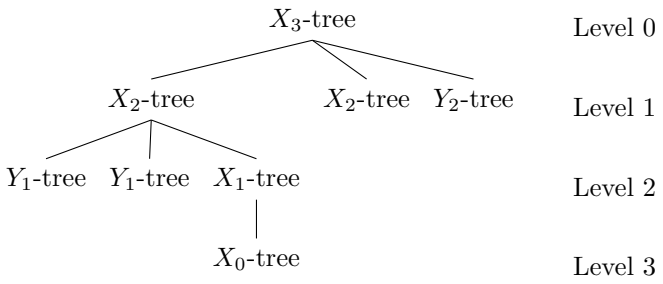
2. $i \neq 1$
   It's possible for the child on level 0 to get children on level 1 if $i \neq 1$, however all those children have to be Y_{i-1}-trees. So the cost made are C to move one level down, K times the expected children, y_{i-1}, and y_{i-1} times the expected cost of an Y_{i-1}-tree, $\mathbb{E}(Y_{i-1})$. Combining these costs we obtain
   $$\mathbb{E}(Y_i) = C + K \cdot y_{i-1} + y_{i-1} \cdot \mathbb{E}(Y_{i-1})$$
   as desired.

□

4.2.3 $\mathbb{E}(X_i)$

The starting point of an X_i-tree can get both tree-types as children, so there is some more to be done in comparison with the Y_i-tree. Notice that one of the children of the starting point in an X_i-tree, should be an X_{i-1}-tree, which means $x_{i-1} \geq 1$. Theorem 4.2.2 still holds for y_i.

Example 4.2.2. Possible X_3-tree.



Note that an Y_i-tree will only add to the cost, except for the K that every child costs, if it's on the "left" from the first X_i-tree on level i. So the Y_2-tree on level 1 in example 4.2.2 is "free", because the program reached level 3 before that child was needed. However the two Y_1-trees on level 2 aren't free, because they both are on the left side of the first X_1-tree.

Y_i-trees inside an X_n-tree Suppose that we are on a level i and of all the children x are X_i-trees and y are Y_i-trees. The possible places for a Y_i-tree are: left of all X_i-trees, between two X_i-trees or right of all the X_i-trees. So in total there are $x + 1$ possible places of which one is to the left of all the X_i-trees. So we are interested in

$$\mathbb{E}\left(\frac{1}{1+x_i} : x_i \geq 1\right).$$

Theorem 4.2.4.

$$\mathbb{E}\left(\frac{1}{1+x_i} : x_i \geq 1\right) = \frac{1}{\lambda_{i-1}} - \frac{1}{e^{\lambda_{i-1}} - 1}$$

where $\lambda_i = P_i \cdot \mu$.

Proof. Without the requirement that $x_i \geq 1$, we would expect $x_i = P_i \cdot \mu$ (similar to Theorem 4.2.2. So the expected value for x_i, without the condition that $x_i \geq 1$, is λ. Therefore:

$$\begin{aligned}
\mathbb{E}\left(\frac{1}{1+x_i} : x_i \geq 1\right) &= \frac{\mathbb{E}\left(\frac{1}{1+x_i} \cap x_i \geq 1\right)}{\mathbb{E}(x_i \geq 1)} \\
&= \frac{\sum_{j=1}^{\infty} \frac{1}{1+j} \cdot \frac{e^{-\lambda}\lambda^j}{j!}}{1 - e^{-\lambda}} \\
&= \frac{e^{-\lambda}}{1 - e^{-\lambda}} \cdot \frac{1}{\lambda} \cdot \sum_{j=1}^{\infty} \frac{\lambda^{j+1}}{(j+1)!} \\
&= \frac{e^{-\lambda}}{1 - e^{-\lambda}} \cdot \frac{1}{\lambda} \cdot \sum_{k=2}^{\infty} \frac{\lambda^k}{k!} \\
&= \frac{e^{-\lambda}}{1 - e^{-\lambda}} \cdot \frac{1}{\lambda} \cdot (e^{\lambda} - \lambda - 1) \\
&= \frac{e^{-\lambda}(e^{\lambda} - 1)}{1 - e^{-\lambda}} \cdot \frac{1}{\lambda} - \frac{e^{-\lambda}}{1 - e^{-\lambda}} \\
&= \frac{1}{\lambda} - \frac{1}{e^{\lambda} - 1}.
\end{aligned}$$

□

Now that we have this estimation it's possible to estimate the cost of all the trees that don't make it to the desired level inside a tree that is certain to reach this level. This cost is

$$y_{i-1} \cdot \left(\frac{1}{P_{i-1} \cdot \mu} - \frac{1}{e^{P_{i-1} \cdot \mu} - 1}\right) \cdot \mathbb{E}(Y_{i-1})$$

since we expect $y_{i-1}(\frac{1}{P_{i-1} \cdot \mu} - \frac{1}{e^{P_{i-1} \cdot \mu} - 1})$ Y_i-trees before the first X_i-tree.

Expected number of X_i-trees per level We now know everything that we need about the Y_i-trees, but we still need to determine the expected number of X_i-trees per level. Normally this would be equal to $P_i \cdot \mu$. Since $x_i \geq 1$ we will have to look at the conditional probability.

Theorem 4.2.5. *Suppose* $x_i \sim Poiss(\mu)$, $\lambda = P_i \cdot \mu$, *then*

$$\mathbb{E}(x_i : x_i \geq 1) = \frac{\lambda}{1 - e^{-\lambda}}.$$

Proof.

$$\begin{aligned}
\mathbb{E}(x_i : x_i \geq 1) &= \frac{\mathbb{E}(x_i \cap x_i \geq 1)}{\mathbb{E}(x_i \geq 1)} \\
&= \frac{1}{1 - e^{-\lambda}} \cdot \sum_{i=1}^{\infty} i \cdot \frac{e^{-\lambda}\lambda^i}{i!} \\
&= \frac{e^{-\lambda}}{1 - e^{-\lambda}} \cdot \lambda \cdot \sum_{i=1}^{\infty} \frac{\lambda^{i-1}}{(i-1)!} \\
&= \frac{e^{-\lambda}}{1 - e^{-\lambda}} \cdot \lambda \cdot e^{\lambda} \\
&= \frac{\lambda}{1 - e^{-\lambda}}.
\end{aligned}$$

□

This gives us all the information we need to make an estimate for the cost of an X_i-tree.

Expected cost X_i-tree Combining everything described above gives the following theorem:

Theorem 4.2.6.

$$\mathbb{E}(X_i) = \begin{cases} 0 & if\ i = 0 \\ C + K \cdot \left(\dfrac{\lambda_{i-1}}{1 - e^{-\lambda_{i-1}}} + y_{i-1} \right) & \\ \quad + y_{i-1} \cdot \left(\dfrac{1}{\lambda_{i-1}} - \dfrac{1}{e^{\lambda_{i-1}} - 1} \right) \cdot \mathbb{E}(Y_{i-1}) & else \\ \quad + \mathbb{E}(X_{i-1}) & \end{cases}$$

where $\lambda_i = P_i \cdot \mu$.

Proof. We actually don't have to prove anything, since this theorem combines the facts already proven above. The key idea is that we need to pay C to go one level down, K times the number of expected children, pay all the Y_{i-1}-trees at the left of the first X_{i-1}-tree and pay the X_i-tree. □

4.3 Expected cost

Number of Y_n-trees before first X_n-tree The number of Y_i-trees before the first X_i-tree can be determined using the geometric distribution. This will tell us the number of failures before the first success. In this case failure is picking a Y_n-tree, while success is picking an X_n-tree. The probability for picking an X_n-tree is P_n, so using the geometric distribution we expect to succeed in $\frac{1}{P_n}$ times. So there are $\frac{1}{P_n} - 1$ Y_n-trees to be expected before the first X_n-tree.

The estimated cost Combining everything results in the following formula for the expected cost:[1]

$$cost = \left(\frac{1}{P_n} - 1\right) \cdot \mathbb{E}(Y_n) + \mathbb{E}(X_n).$$

4.4 Results

4.4.1 First test

So now that we can calculate the expected cost, it is much easier to pinpoint a cheap μ for different K. However first we wanted to see if the estimate is actually any good. Therefore we did the following:[2]

- Determine the expected cost for a given μ.
- Actually calculate the cost a thousand times for this μ.
- Compare the mean of the thousand calculated costs with the expected cost.

Some of the results are printed below. This gives a good first impression whether the estimate is of any use.

$\mu = 1.1$	
Mean	5330.8
Expected	5326.506

$\mu = 2.6$	
Mean	2055.302
Expected	2055.296

$\mu = 1.6$	
Mean	1961.437
Expected	1959.728

$\mu = 3.1$	
Mean	2279.561
Expected	2281.08

$\mu = 2.1$	
Mean	1896.248
Expected	1897.026

$\mu = 3.6$	
Mean	2537.95
Expected	2537.344

4.4.2 Z-score

Now we would like to look a bit more precisely at how good the estimation actually is. Therefore we use the Z-score, which can be calculated as below:

$$Z = \sqrt{r} \cdot \frac{\overline{cost} - \mathbb{E}(X_n)}{\sigma(cost)}.$$

Herein r denotes how many times the cost is calculated for this μ (thus $r = 1000$ in the results above), $\overline{cost}$ is the average cost of those r calculations and $\sigma(cost)$ the standard deviation of all the calculated costs.

Calculating multiple Z-scores for the same μ should show if the Z-scores behave as one then expects. We tested this by calculating a thousand Z-scores

[1] For the actual implementation of the code, see Appendix 5.2.
[2] For the actual implementation of the code, see Appendix 5.3.

for $\mu = 1.7$, where $r = 1000$ as above. It's possible to test if our hypothesis, i.e. that $E(X_n)$ is indeed the expected cost, holds for these Z-scores.

Our hypotheses are:

- H_0: $\mathbb{E}(X_n) = \overline{cost}$.
- H_1: $\mathbb{E}(X_n) \neq \overline{cost}$.

If H_0 is true, then the Z-scores calculated as above would approximate the standard normal distribution very closely. This follows from the fact that all the costs are calculated independently of each other and we can therefore apply the Central Limit Theorem. Making a histogram of all the Z-scores and comparing with the standard normal distribution shows that is indeed the case:

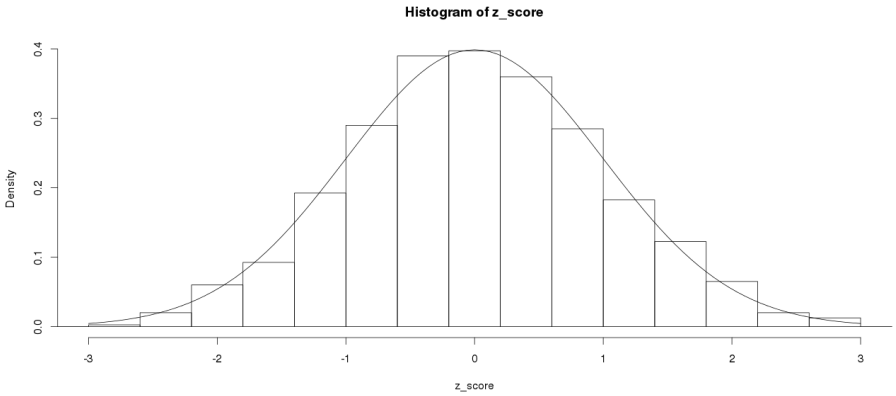


Figure 7: Histogram of all the Z-scores and the standard normal distribution

Figure 7 gives us no reason to reject H_0. Thus we're confident enough to continue with $E(X_n)$ as the expected cost.

4.4.3 Perfect μ

Since the expected cost $\mathbb{E}(X_n)$ seems to be a very good estimate for the cost, we can now determine the cheapest μ for given C, K, n. If we take $n = 300$, we can improve figure 4 quite a bit.
Remark: Cheapest μ is defined as the μ with the lowest expected cost for given parameters C, K, n.

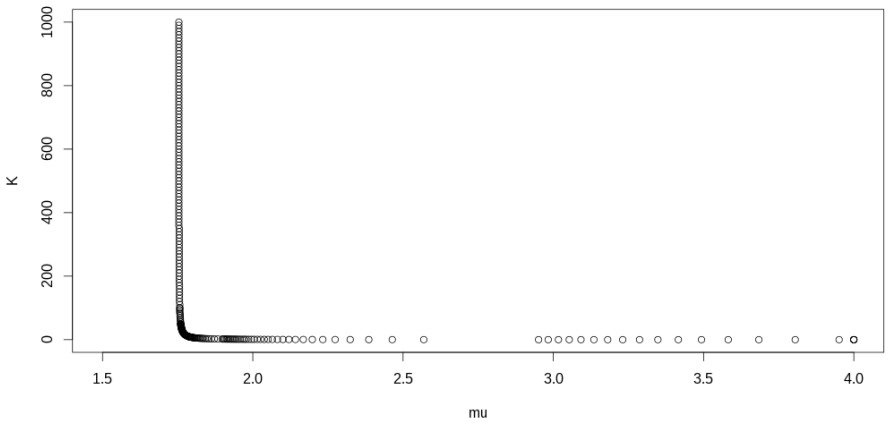


Figure 8: Plot of the cheapest μ for different K, $1 < \mu \leq 4$

There is quite some difference between figure 4 and figure 8, most notably the minimal value μ takes on. In figure 4 we see that μ goes as low as $\mu = 1.2$, however in figure 8 this isn't true and we have $\mu > 1.75$ for all K. This difference is not that hard to explain, although the "limit" of $\mu = 1.75$ is bigger than we expected at first.
Figure 4 is created by simulating actual costs to reach a level, so it's possible that one $\mu \approx 1.2$ is very lucky and reaches level 300 very cheaply, thus this μ becomes the cheapest μ (This was somewhat covered by repeating the simulation and taking an average over the cheapest μ, but this apparently wasn't enough). There are of course also $\mu \approx 1.2$, that are unlucky and reach level 300 at a rather high cost. Figure 8, on the other hand, shows the estimated cost to reach level 300. So while indeed there can be a cheap $\mu \approx 1.2$, there are also very expensive $\mu \approx 1.2$. It's therefore in general better to take $\mu > 1.75$, even for really large K.

4.4.4 Limits of K

K to infinity

As can be seen in figure 8, the cheapest μ doesn't really change that much for $K > 200$; it looks like there is some kind of asymptote. Apparently for $\mu < 1.75$, there are too much expected Y_i-trees that cost too much together.

K to zero

If $K \to 0$, then it seems that $\mu \to \infty$. This can be seen in figure 8, as for $K < 1$ we see that $\mu \to 4$. It's even more obvious in the figure below:

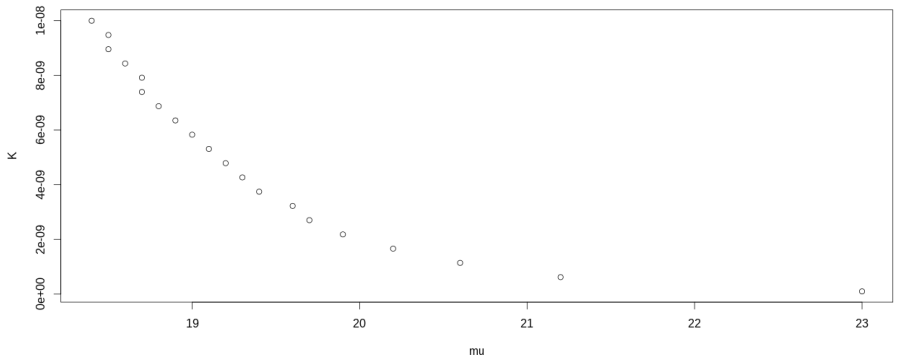


Figure 9: Cheapest μ for $K \to 0$.

However for K really small, $K \approx 10^{-10}$, the difference becomes really small for $\mu > 30$. Thus while in theory the cheapest μ will (probably) go to infinity, in practice the difference between $\mu = 35$ and $\mu = 100$ is hardly noticeable (even default precision in R isn't high enough to notice the difference).

4.5 Influence of n

Lastly we will take a look at the influence of n on the cheapest μ. Below is a table of the cheapest μ for different n and K.

	$K = 0.01$	$K = 1$	$K = 100$
$n = 10$	4.7313	1.146	1
$n = 100$	4.7383	2.01	1.751
$n = 300$	4.7388	2.015	1.757
$n = 1000$	4.7389	2.016	1.759

Notice that we put an extra decimal in the column of $K = 0.01$, to emphasize the small changes happening if $K \to 0$. As we have seen before, the difference between $n = 10$ and $n = 100$ is big, but the differences between $n = 100, n = 300$ and $n = 1000$ are quite small. So it seems that we took a good default value with $n = 300$, since we expect $100 < n < 5000$ in the real case. So even though $n = 300$ is a guess, the values for the cheapest μ probably won't change that much in a real situation.

References

[1] Joseph H. Silverman, *An Introduction to the Theory of Elliptic Curves*, Brown University and NTRU Cryptosystems, Inc., 2006 <http://www.math.brown.edu/~jhs/Presentations/WyomingEllipticCurve.pdf>

[2] Wieb Bosma, Antal Járai, Gyöngyvér Kiss, *Better paths for elliptic curve primality proofs*, Radboud University Nijmegen, July 2009 <http://www.math.ru.nl/~bosma/pubs/reportfinal.pdf>

5 Appendix

5.1 Appendix 1: Code for the model

```
#Parameters
C=1
K=15 #Can be a vector
n=300 #level to be reached
mu=2 #Can be a vector
active_level=1


#Needed variables

#Matrix to count all the number of children
#Set to 1 on the first row, the rest equals -1
tree=matrix(
  c(rep(N_start,length(mu)),rep(-1,(n-1)*length(mu))),
  nrow=n,
  ncol=length(mu),
  byrow=TRUE)

#Matrix to remember the cost for all K and mu
cost=matrix(
  c(rep(0,length(mu)*length(K))),
  nrow=length(K),
  ncol=length(mu),
  byrow=TRUE)


###############
##THE  MODEL##
###############

#Continue until level n is reached
while (active_level < n){

  #Pick a child and continue to the next level
  tree[active_level,]=tree[active_level,]-1
  active_level=active_level + 1

  #Determine the number of children on the next level
  for (j in 1:length(mu)){
      tree[active_level,j]=rpois(1,mu[j])

      #Determine the cost made per mu for every K
      for (k in 1:length(K)){
        cost[k,j]=cost[k,j]+C #Determine the cost
        cost[k,j]=cost[k,j]+K[k]*tree[active_level,j]
      }
  }

 #Check if backtracking is needed
  for (j in 1:length(mu)){

    #Continue until there are children on the active level for this mu
    while (tree[active_level,j] == 0){

      #Set new termperary "active level" i
      i=active_level

      #Go a level up as long as there are no unused children available
      while (tree[i,j]==0 && i>1) {i=i-1}

      #Start again if the tree is exhausted
      if  (tree[i,j]==0) {tree[1,j]=1}
```

```
      #A unused child is found, try to reach the active level again
      else{

        #Used child and determine its children (and the costs)
        tree[i,j]=tree[i,j]-1
        i=i+1
        tree[i,j]=rpois(1,mu[j])
        for (k in 1:length(K)){
          cost[k,j]=cost[k,j]+C #Determine the cost
          cost[k,j]=cost[k,j]+K[k]*tree[active_level,j]
        }

        #Continue to the next level as long there are children and active level isn't reached yet
        while (tree[i,j]>0 && i<active_level){

          tree[i,j]=tree[i,j]-1
          i=i+1
          tree[i,j]=rpois(1,mu[j])

          for (k in 1:length(K)){
            cost[k,j]=cost[k,j]+C
            cost[k,j]=cost[k,j]+K[k]*tree[active_level,j]
          }
        }
      }
    }
  }
}

#Make empty plot
plot(x=NULL,xlim=c(mu[1],mu[length(mu)]),ylim=c(0,10*median(cost)),xlab="mu",ylab="cost")

#Plot lines in different colors.
for (i in 1:length(K)){
  if(i < .2*length(K)){
    lines(loess.smooth(mu,cost[i,]),col="red")
  }
  else if ( i < .4*length(K)){
    lines(loess.smooth(mu,cost[i,]),col="blue")
  }
  else if ( i < .6*length(K)){
    lines(loess.smooth(mu,cost[i,]),,col="yellow")
  }
  else if ( i < .8*length(K)){
    lines(loess.smooth(mu,cost[i,]),col="green")
  }
  else{
    lines(loess.smooth(mu,cost[i,]))
  }
}
```

5.2 Appendix 2: Code for the expected cost

```
#Parameters
C=1
K_vector=c(seq(0,0.1,0.005),seq(0.2,2,.05),seq(2,10,0.1),seq(10,50,.5),seq
    (50,100,1),seq(100,1000,10)) #Vector of values for K
mu=seq(1.1,4,.01) #Can be a vector
n=300  #end level of the tree
minmu=vector()
mincost=vector()


#Needed variables
Q_matrix <<-matrix(rep(0,length(mu)*(n+1)),nrow=n+1,ncol=length(mu),byrow=
    TRUE)

##############################
##Functions for the estimate##
##############################

#Function that calculates Q_i for given mu and end level n
#Input:
#   mu  (expected children for every level)
#   n   (the desired end level)
#Output:
#   Matrix Q_matrix with all the Q_i for 0 <= i <= n

Q <- function(mu,n){
  if(n == 1){
    Qn=exp(-mu)
  }
  else{
    Qn=exp(-mu*(1-Q(mu,n-1)))
  }
  Q_matrix[n+1,]<<-Qn
  return(Qn)
}

#Function to estimate the cost of a Y-tree
#Input:
#   C   (cost for going one level down)
#   K   (cost per child)
#   mu  (expected amount of children)
#   n   (the desired end level for this Y-tree)
#Output:
#   Cost of an Y_n-tree

Y <- function(C,K,mu,n){
  if(n ==0)
    Yn=0
  else
    Yn=C+K*mu*Q_matrix[n,] + Q_matrix[n,]*mu*Y(C,K,mu,n-1)
  return(Yn)
}
```

```
#Function to estimate the cost of a X-tree
#Input:
#    C    (cost for going one level down)
#    K    (cost per child)
#    mu   (expected amount of children)
#    n    (the desired end level for this X-tree)
#Output:
#    Cost of an X_n-tree

X <- function(C,K,mu,n){
  if (n == 0)
    Xn=0
  else{
    Pn=1-Q_matrix[n,]
    Xk=(Pn*mu)/(1-exp(-Pn*mu))
    Yk=Q_matrix[n,]*mu
    Xn=C+K*(Xk+Yk)+Yk*(1/(Pn*mu)-1/(exp(Pn*mu)-1))*Y(C,K,mu,n-1) + X(C,K,mu,n
        -1)
  }
  return(Xn)
}


######################
##Estimate the cost##
######################

#Calculate the expected cost for different K
for (i in 1:length(K_vector)){

        #Calculate the cost for all mu at once
        estimate=(1/(1-Q(mu,n))-1)*Y(C,K_vector[i],mu,n)+X(C,K_vector[i],mu,n
            )

        #Determine which mu was the cheapest
        minmu[i]=which(estimate==min(estimate),arr.ind=TRUE)
}
```

5.3 Z-score test

```
#Parameters
C=1
K=2
mu_vector=seq(1.1,4,.05) #Can be a vector
n=300  #end level of the tree
repetitions=50000 #Number of actual cost calculations


#Needed variables
Q_matrix <<-matrix(rep(0,(n+1)),nrow=n+1,ncol=1,byrow=TRUE)
cost=matrix(rep(0,repetitions*length(mu_vector)),nrow=length(mu_vector),byrow
    =TRUE)
estimate<-vector()
average<-vector()
z_score<-vector()

##############################
##Functions for the estimate##
##############################

#Function that calculates Q_i for given mu and end level n
#Input:
#   mu  (expected children for every level)
#   n   (the desired end level)
#Output:
#   Matrix Q_matrix with all the Q_i for 0 <= i <= n

Q <- function(mu,n){
  if(n == 1){
    Qn=exp(-mu)
  }
  else{
    Qn=exp(-mu*(1-Q(mu,n-1)))
  }
  Q_matrix[n+1,]<<-Qn
  return(Qn)
}

#Function to estimate the cost of a Y-tree
#Input:
#   C   (cost for going one level down)
#   K   (cost per child)
#   mu  (expected amount of children)
#   n   (the desired end level for this Y-tree)
#Output:
#   Cost of an Y_n-tree

Y <- function(C,K,mu,n){
  if(n ==0)
    Yn=0
  else
    Yn=C+Q_matrix[n,]*K*mu + Q_matrix[n,]*mu*Y(C,K,mu,n-1)
  return(Yn)
}
```

```
#Function to estimate the cost of a X-tree
#Input:
#    C    (cost for going one level down)
#    K    (cost per child)
#    mu   (expected amount of children)
#    n    (the desired end level for this X-tree)
#Output:
#    Cost of an X_n-tree

X <- function(C,k,mu,n){
  if (n == 0)
    Xn=0
  else{
    Pn=1-Q_matrix[n,]
    Xk=(Pn*mu)/(1-exp(-Pn*mu))
    Yk=Q_matrix[n,]*mu
    Xn=C+K*(Xk+Yk)+Yk*(1/(Pn*mu)-1/(exp(Pn*mu)-1))*Y(C,K,mu,n-1) + X(C,K,mu,n
        -1)
  }
  return(Xn)
}


#Calculate Z-score for (different) mu
for (i in 1:length(mu_vector)){
  mu=mu_vector[i]

  ######################
  ##Estimate the cost##
  ######################
  estimate[i]=(1/(1-Q(mu,n))-1)*Y(C,K,mu,n)+X(C,K,mu,n)


  ###################
  ##Calculate cost##
  ###################

  #Since the index starts at 1, starting point is level 1 instead of level 0
  #So the end level should also be increased
  n=n+1
  for (j in 1:repetitions){

    level=1 #because of index
    tree=c(1,rep(0,n)) #tree with 1 child at level 1, 0 on other levels

    #Continue until end level is reached
    while (level < n){

      #Start again if tree is exhausted
      if(level==1 && tree[level]==0){
        tree[level]=1
      }

      #Choose a child, determine it's children and calculate the cost
      tree[level]=tree[level]-1
      tree[level+1]=rpois(1,mu)
      cost[i,j]=cost[i,j]+K*tree[level+1]
      cost[i,j]=cost[i,j]+C

      #See if the next level has any children.
      #Else go back to the highest level that still has unused children
      level=level+1
      while(tree[level]==0 && level!=1){
        level=level-1
      }
    }
  }
```

```
  ##########################
  ##Determine the z-score##
  ##########################

  #Determine the averages and the z-score
  average[i]=mean(cost[i,])
  z_score[i]=sqrt(repetitions)*(average[i]-estimate[i])/sd(cost[i,])

  #Print out the result
  cat("Repeat:␣␣␣",repetitions)
  cat("\nMu:␣␣␣␣␣␣␣",mu)
  cat("\nMean:␣␣␣␣␣",mean(cost[i,]))
  cat("\nVerwacht:␣",estimate[i])
  cat("\nSd:␣␣␣␣␣␣␣",sd(cost[i,]))
  cat("\nZ:␣␣␣␣␣␣␣␣",z_score[i])
  cat("\n\n")

  #Decrease n again, needed if mu_vector indeed a vector
  n=n-1
}
```