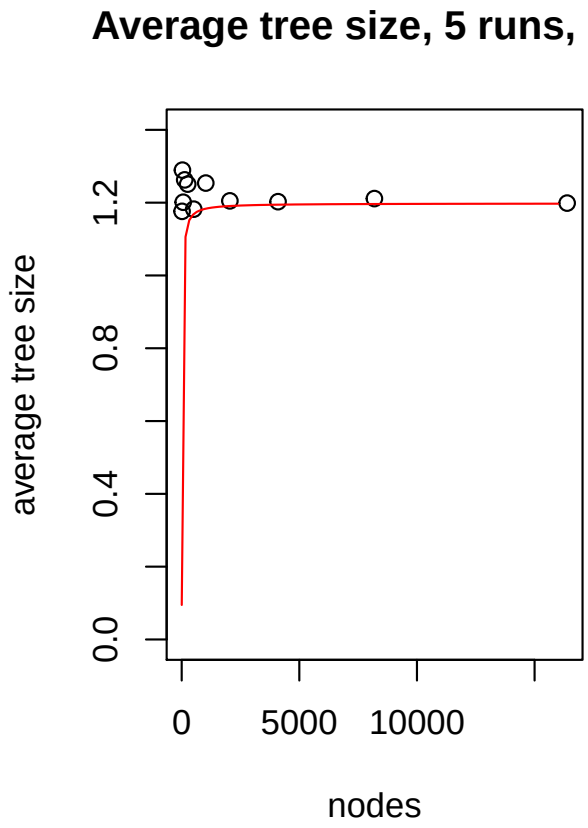
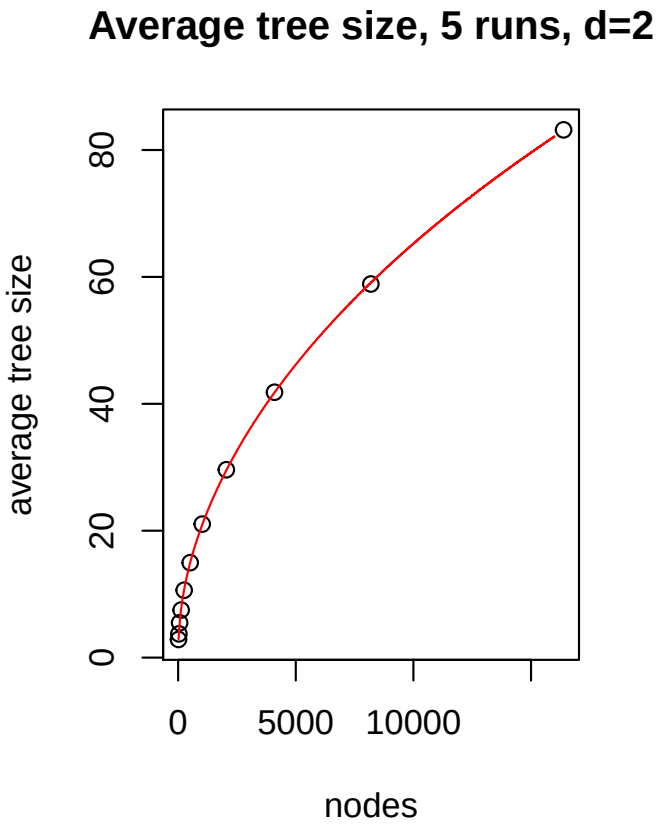
Algorithms Programming PS1

Josh Kaplan & John Hotchkiss

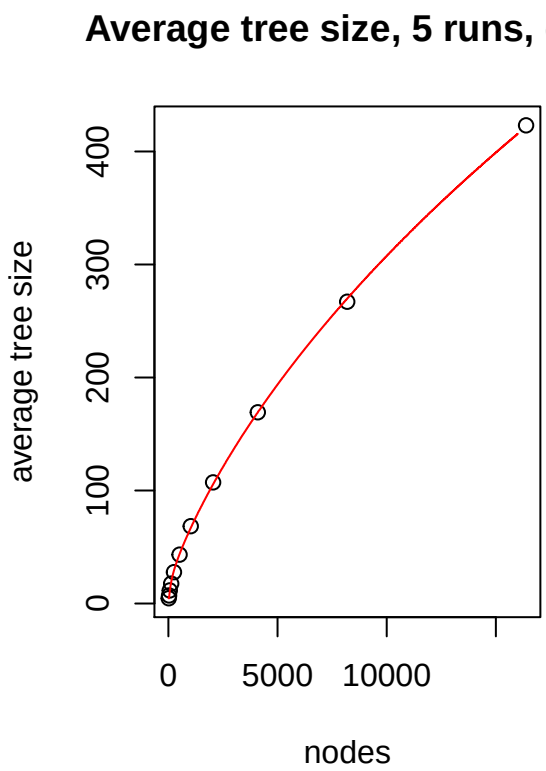
3/2/2017

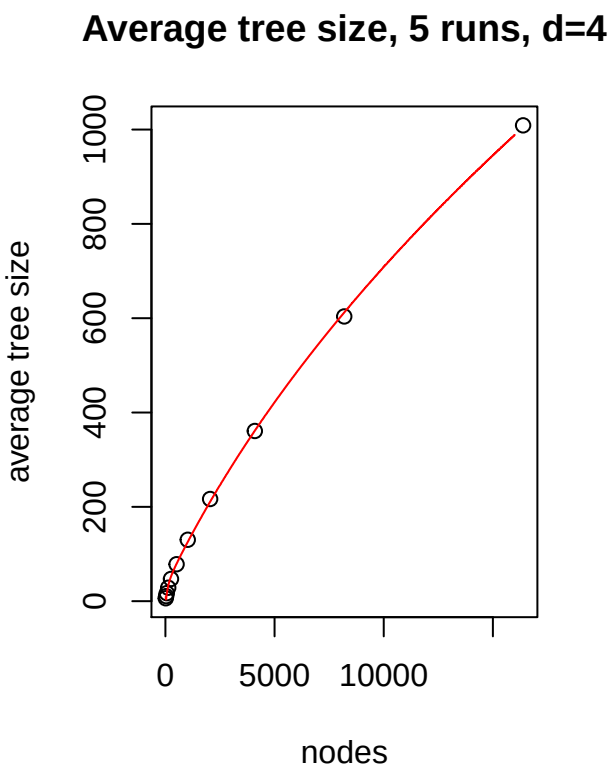
Part 1





For $d = 0$, the tree size basically stops increasing after about 32 nodes, settling down to about 1.2. We tested smaller graph sizes than 16 in order to fit the curve, and found that arctan works particularly well because it also settles down very quickly. For $d = 0$, $f(n) = .75 * \arctan(n/20) + .02$. $d = 2$ clearly exhibits different behavior - the tree size increases with the nodes at a decreasing rate, characteristic of a logarithmic function. We used regression to find a function that exactly fits our data points (The R^2 was $> .999$). The result was: f(n) = 19.9223713 +-16.6133158log(x) + 5.5573793 log(x)^2 + -0.7673005 log(x)^3 + 0.0453579 log(x)^4. The y-intercept is not very meaningful in these cases; it is set so that the model fits, but it doesn't make a lot of sense because the model is not based on any data from $n = 0 - 15$.





Again, we used regression to find functions that exactly fit our data points. The functional form for d=3,4 was basically identical to d=2, only the model coefficients changed. For d=3, f(n) = 211.2809282 +-186.5858728log(x) + 60.2758222 log(x)^2 + -8.3623451 log(x)^3 + 0.4495789 log(x)^4 and for d=4, f(n) = 667.4062802 +-588.7571167log(x) + 188.1568835 log(x)^2 + -25.8921268 log(x)^3 + 0.4495789 log(x)^4. Again, the y-intercepts are not especially meaningful in these cases because the model does not include data from $n = 0 - 15$.

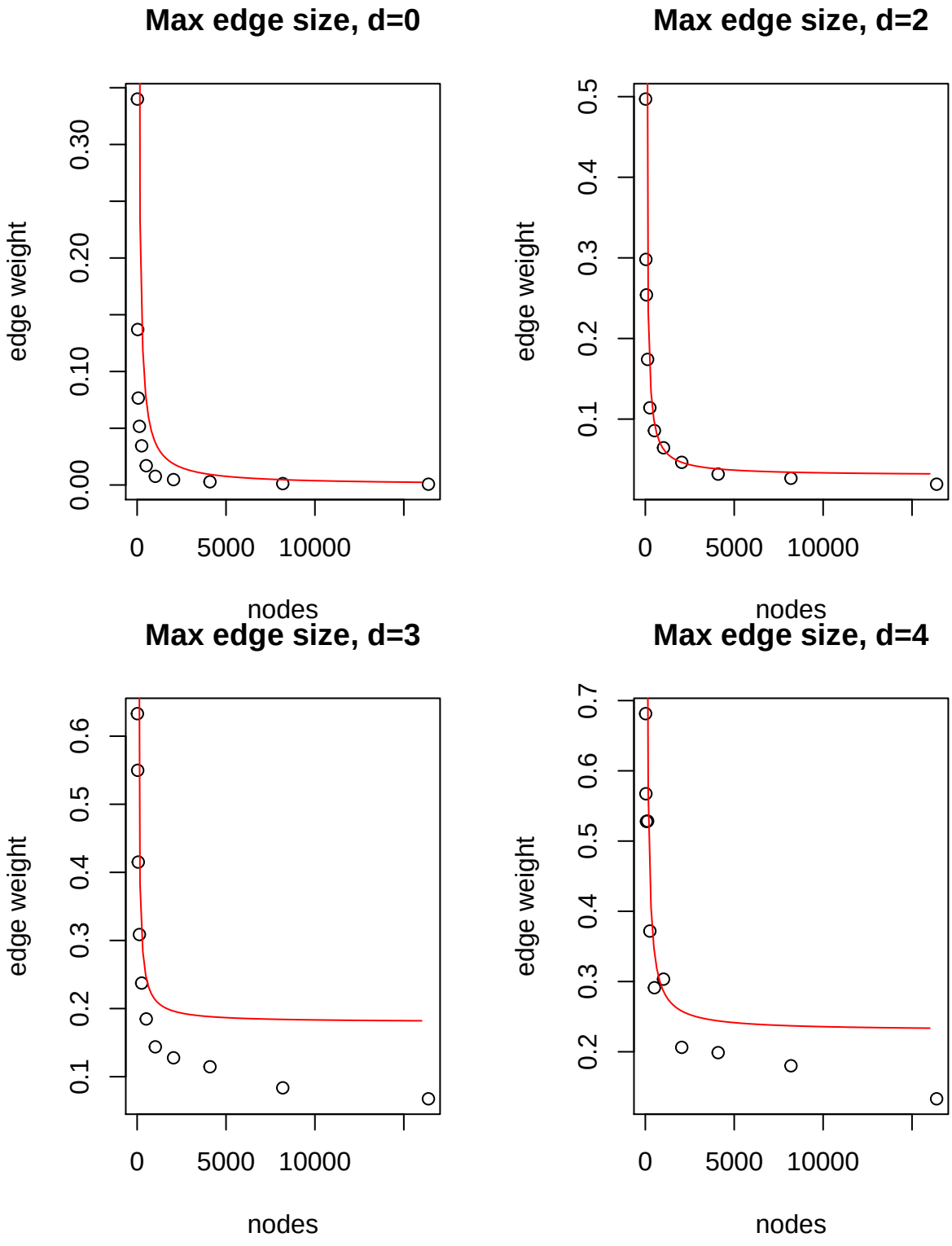
Part 2

Growth rates of f(n)

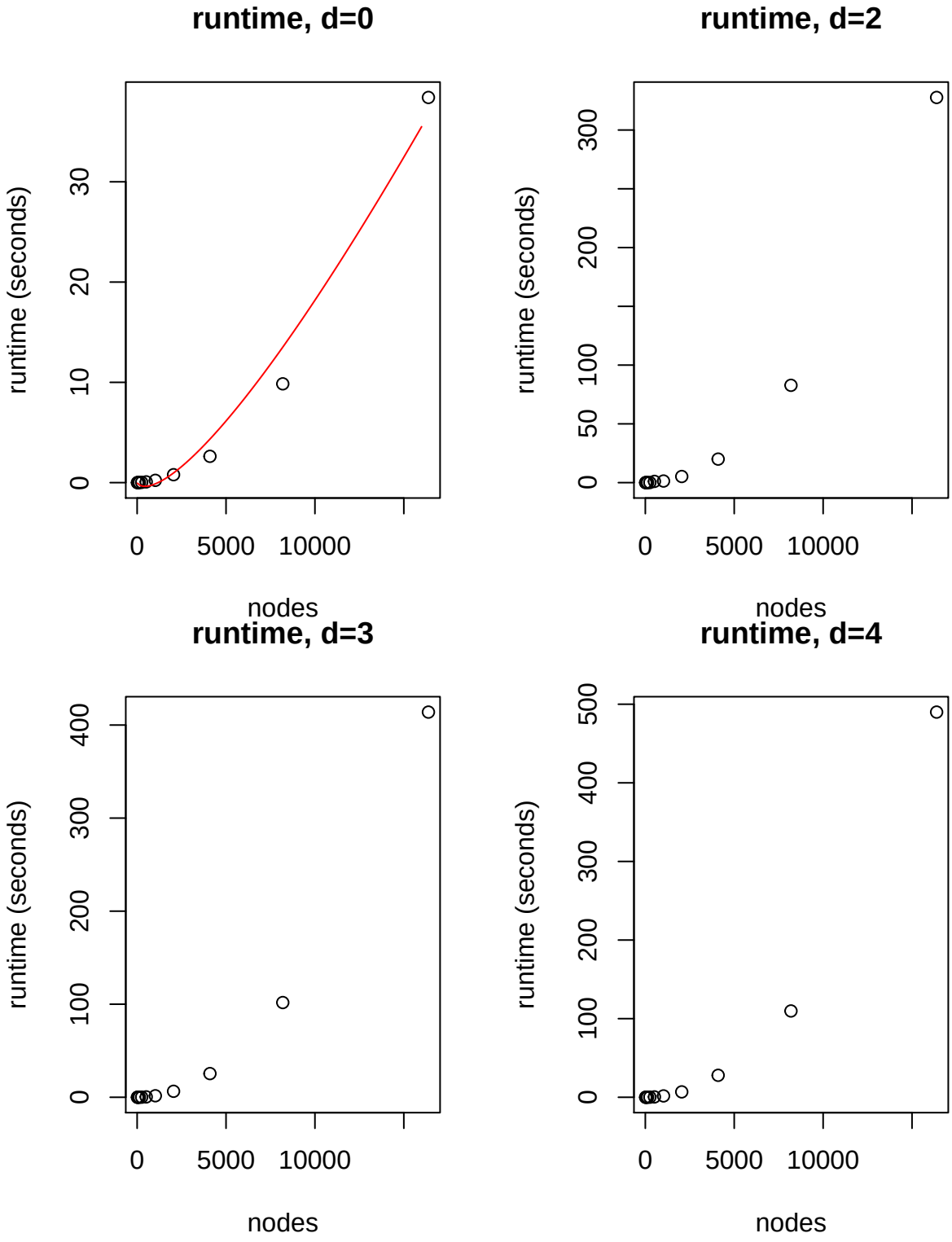
For d=2,3, and 4, the logarithmic behavior of f(n) makes sense. As n increases, each node has more edges coming off of it (since the graph is complete), so Prim's algorithm has exponentially more edges to choose from in selecting edges of minimum weight. Thus, you would expect the average edge weight to decrease as n increases. On the other hand, there are more nodes in the graph, so the MST must contain more edges in order to connect them all. So you would expect the average tree size to increase. The nodes (and the number of edges in the tree) increases linearly, while the average edge weight decreases logarithmically. Thus, we see logarithmic growth in the size of the tree. This pattern holds for graphs in any dimension >1, but higher dimensions result in overall higher edge weights, so the average tree size is a constant factor higher for all n.

We were surprised to see the quick convergence of the tree size in d=0. After about 30 nodes, the tree doesn't grow no matter how many nodes are added to the graph. After some internet research, we discovered that the limit on the weight of the spanning tree, as n goes to infinity, is equal to Riemann zeta(3) ≈ 1.2.

Determining edge weight threshold (k(n))



Runtime



For all choices of d, runtime exhibits the same functional form with respect to the number of nodes. This is the characteristic O(n log n) curve, which makes sense as, due to our optimization reducing the number of edges submitted to Prim's algorithm, |E| is O(n). The runtime increases substantially with d, as well; a line-by-line analysis indicates that much of the increased time comes in the graph generation stage - calculating the

Euclidean distance between nodes in 4 dimensions is substantially more costly than 2 dimensions. For all d and large n, nearly all of the runtime depicted above comes from generating the random graphs - for n = 16000, d = 0, Prim's algorithm takes 6 seconds to run on 305343 edges which met the threshold, while generating those edges takes 925 seconds.