Optimizing the Critical Path of Distributed Dataflow Graph Algorithms

Dante Durrman
Dept. Mathematics
UNC Charlotte
Charlotte, NC
ddurrman@uncc.edu

Erik Saule
Dept. Computer Science
UNC Charlotte
Charlotte, NC
esaule@uncc.edu

***Abstract*—Executing graph algorithms in a parallel or distributed context is a challenging problem. Solving race conditions with locks is usually prohibitively expensive and some algorithms opt for a strategy that ignores the race condition altogether and corrects later the derived solution if it is invalid. Alternatively, dataflow algorithms solve the synchronization problem by executing the algorithm by following a partial order on the graph. While removing the cost of locks or avoiding a checking phase improves performance, it is possible that the algorithm picks a partial order with long chains, which limit parallelism.**

In this paper, we investigate how distributed dataflow graph algorithm obtain a partial order and how one could favor orders with shorter long chains. Most dataflow algorithms obtain their order by having each vertex of the graph pick a uniformly random number in $[0; 1)$ and order the vertices based on that number. We believe that this type of order could lead to long chains in graphs with dense regions such as small world graph. We design two alternative ways of generating the order to make it similar to a largest degree first order.

We study the behavior of these different algorithms on a wide range of randomly generated RMAT graphs and on a set of real world graphs. And we show that our ordering methods can significantly reduce the length of the longest chain.

***Index Terms*—graph analysis; distributed computing; partial order; interval coloring; randomized algorithms**

I. Introduction

Graphs are a key mathematical object of modern science as they are used to model a variety of objects of studies including physical objects, roads, computer networks, and social interactions. With the increase in complexity of studies that are performed, the size of graphs that are used has increased. Consequently, computational costs of analyses have increased and the machines we use to perform these calculations have grown more parallel and more distributed.

Executing graphs on parallel and distributed machines is complicated because of the irregularity of the memory access patterns. Simple solutions involve using some form of a locking mechanism. However, the locking overhead usually dominates the calculations. Some problems admit optimistic algorithms where race conditions are ignored at first and the solution is later examined and fixed. It is possible that a race condition happens in a way that leads to an incorrect solution [1], [2]. These optimistic algorithms fundamentally require a later reconciliation phase, which can be as costly as the algorithm itself.

A third category of dataflow graph algorithms rely on a partial order to the graph, where vertices are processed in an order compatible with that partial order.

The most classic dataflow algorithms are Luby's algorithm for Maximum Independent Set [3] and the Jones-Plassmann algorithm for graph coloring [4]. However, many problems admit dataflow algorithms, such as maximum cardinality matching [5]. These algorithms are particularly suitable for the setting where each vertex is its own independent computational node. They are also easily written in think-like-a-vertex programming models [6]. They have also been used on shared memory systems, including the Cray XMT [1], [5].

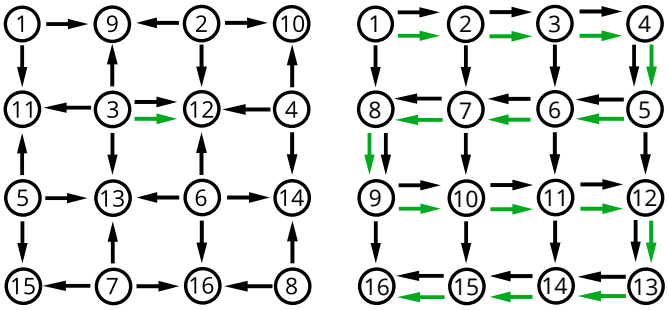
A bottleneck to the execution of these algorithms is the longest chain of vertices set by the partial order. Since the precise partial order often does not matter for correctness, dataflow algorithms often use a random order. Random orders can be generated in a distributed way and have been shown to yield desirable properties in several types of graphs [3], [5]. In this paper, we investigate alternative ways to generate random orders that minimize the length of the longest chain of vertices in the algorithm, minimizing its runtime.

The paper is organized as follows. Section II explains how dataflow algorithms work and provides a model of the problem as a graph coloring problem. Section III discusses related papers and results. Section IV presents new ways to generate orderings for dataflow algorithms and gives a theoretical argument for why they are sound. Section V studies the behavior of the algorithms on random RMAT graphs and shows that our methods perform usually better. Section VI studies the behavior of these algorithms on real world graphs and shows that our methods perform usually better. Section VII provides some concluding remarks.

II. Problem Statement

A. Dataflow Algorithms

All dataflow graph algorithms share a similar structure. We explain in detail how Luby's algorithm computes an Independent Set [3] as a distributed algorithm. Let $G = (V, E)$ be a graph. We denote the neighbors of vertex v by $\Gamma(v)$, the degree of v by $\delta(v) = |\Gamma(v)|$, and the maximum degree in the graph by $\Delta = \max \delta(v)$.



(a) Lucky draw (b) Unlucky draw

Fig. 1: The execution time of dataflow algorithm depends on the random number generation. Black edges highlight the order of the dataflow, while green edges show the critical path. On a lucky draw, the critical path of the algorithm contains only 2 vertices; while an unlucky draw can have 16 vertices in its critical path.

In Luby's algorithm, each vertex v starts by picking a random number $r(v)$ uniformly in $[0; 1)$, which is assumed to be unique, and sends it to each of its neighbors. Vertex v notes which of its neighbors u have the property that $r(u) < r(v)$.

Vertex v marks its own state as `unknown`. It awaits a message from each of its neighbors u if $r(u) < r(v)$ which contains the state of neighbor u. If u's state is `marked`, v changes its state to `unmarked`.

After receiving messages from all neighbors u, if vertex v state is `unknown`, it changes its state to `marked`. And finally v sends its state to all its neighbors u such that $r(u) > r(v)$. At the end of this process, all vertices in the `marked` state are a maximal independent set (by inclusion).

Other dataflow algorithms are similar in structure. A random number is assigned to each vertex. And each vertex executes an algorithm only after all of its neighbors with a lower value have been executed.

In Luby's algorithm, each vertex v executes an algorithm of complexity $\Theta(\delta(v))$, linear in its number of neighbors. However, since each vertex has to wait for some of its neighbors to complete, the entire process might not unfold in $\Theta(\Delta)$, proportionally to the maximum degree of the graph. The time it takes for the algorithm to unfold depends on the longest chain of communication induced by the algorithm, which depends on the random numbers generated.

A grid graph is used as an example in Figure 1. In a lucky draw (Figure 1a), random numbers are generated in a way that leads to lots of parallelism. The black arrows show the direction of the communication in the graph. The green arrows show the longest chain in the oriented graph. In this case, all chains are of two vertices and an arbitrary one was highlighted.

In an unlucky draw (Figure 1b), the random numbers are generated in a way that leads to no parallelism as the longest chain in the oriented graph visits every vertex in the graph. Of course, both of these draws are extremely unlikely to happen.

B. Combinatorial Optimization Model

Ultimately, dataflow algorithms rely on the underlying coloring of a graph with intervals. Let $G = (V, E)$ be the graph that gets processed by the dataflow algorithm. Let w be a weight function on vertices such that $w(v)$ is the processing time of an algorithm for vertex v. The dataflow algorithm is dependent upon two neighboring vertices $(x, y) \in E$ to not be executed simultaneously. It assumes they run at times $[start(x); start(x) + w(x))$ and $[start(y); start(y) + w(y))$ such that these two intervals are disjoint. This is the definition of a coloring of the graph with intervals of length given by the weights of the vertices. The objective is to minimize the total number of used colors, which is equivalent to minimizing the total runtime of the application.

How good the model is will largely be determined by how the weight function w is set. On a computing machine with substantially larger latency than execution time at each node, a good model will be achieved by setting all w to 1 and solving the standard coloring problem. If the processing at each vertex is the primary cost of the dataflow algorithm, then setting $w(v)$ to the complexity of the vertex algorithm for each vertex is the right call. For most algorithms, vertices will need to gather partial information from their neighbors, do some processing, which is usually proportional to the number of neighbors, and finally, communicate the partial information to the neighbors. All operational costs will be proportional to the number of neighbors of a vertex. Setting the weight to the degree of the vertex $w(v) = \delta(v)$ often makes the most sense. We will make this assumption going forward even though the analysis can be adapted to other weight functions.

The distributed dataflow algorithm solves that problem using a particular ordering algorithm. But fundamentally, any coloring of the graph with intervals would be sufficient to derive a correct execution of the dataflow algorithm. And better colorings would lead to better runtimes for the execution.

In the context of a distributed graph, it makes sense to run a distributed coloring algorithm: aggregating the graph to a single computing node to execute a one-node algorithm would likely be prohibitively expensive compared to the rest of the dataflow algorithm execution time.

III. Related Works

Scheduling, edge orientation, and coloring problems are fundamentally related. In a typical task graph scheduling problem [7], the order of tasks is known in advance and the problem is to decide when and where the tasks will run given a list of dependency constraints. An interesting result is that list scheduling [8] always guarantees to get an application executed on P processors quicker than $\frac{\sum_{v \in V} w(v)}{P} + max_{c \in allchains(G)} \sum_{v \in c} w(v)$. The second term is the length of the longest chain in the graph, which is optimized by minimizing the number of colors.

Another problem is the edge orientation problem. The Gallai-Hasse-Roy-Vitaver theorem proved that the maximum path length in an oriented graph is always greater than one plus the chromatic number of its unoriented counterpart

(and equal for the optimal orientation) [9]. On weighted graphs, most edge-orientation problems attempt to minimize maximum weighted outdegree [10] as opposed to maximum path length of an acyclic orientation. There are distributed algorithms to minimize the number of colors for graphs with particular structures. For instance, if the edges can be oriented so that every vertex has an outdegree less than or equal to 1, then the Cole-Vishkin algorithm can be applied [11].

Even though the classic problem of coloring graph is polynomial for particular categories for graphs [12], it is NP-Complete for arbitrary graphs [13]. On arbitrary graphs, the problem is even not polynomially approximable [14] and often the best guarantee that can be made is that greedy algorithms can always achieve a coloring using fewer than $\Delta+1$ colors [15]. In practice, it has been reported that selecting orderings of vertices can generate better colorings [16]–[18]. We leverage that intuition in this particular work where we generate orderings using some graph property to reduce the total number of colors.

Coloring graphs with intervals has received little consideration in the past. The general NP-Completeness result holds on arbitrary graphs and the problem is often studied from a radio spectrum allocation perspective [19]. Recently, we studied the problem of coloring stencil graphs with interval and provided NP-completeness and approximation results [20].

The complexity of dataflow algorithms for maximal independent set and matching was shown to be polylogarithmic with high probability when processing random graphs on PRAM machines [5]. However, they did not attempt to reduce the critical path length by adjusting the randomized algorithm.

IV. Deriving better partial orders

A. Methods

Luby's algorithm and most dataflow algorithms generate random numbers uniformly in $[0; 1)$ to derive the order of vertices. This yields a partial order in a distributed setting so each vertex v performs only $\Theta(\delta(v))$ calculations and $\Theta(\delta(v))$ communications. The communication term is required for a vertex to know its place in the order relative to its neighbors. At the level of the system, there are only $\Theta(E)$ calculations and communications. We call `Uniform` this particular ordering algorithm.

While that algorithm has optimal cost to derive a partial order, it may not derive the best order. In particular, this ordering does not leverage any properties of the graph and its vertices. We know from literature that coloring heuristics can benefit from considering vertex properties and local structure. In particular the Smallest Last [16] and Largest First [18] orderings are know to be good for the classic coloring problem [21]. Since Smallest Last is a dynamic ordering, it is costly to replicate in a distributed setting. Instead, we focus on emulating Largest First.

Instead of generating random numbers uniformly in $[0; 1)$, we propose adjusting random number generation based on the property of the vertex drawing the number. In particular, we call `Linear` the algorithm where vertex v draws a number uniformly in $[0; \delta(v))$. This algorithm has the same complexity as `Uniform` but it generates an ordering that will tend to put vertices with high degree towards the end of the ordering.

However, a vertex v is guaranteed to be after all vertices u such that $\delta(u) = \delta(v) - 1$ only with probability $\frac{1}{\delta(v)}$ (for an infinite number of such vertices u). `Linear` is a good approximation of Largest First for vertices with dramatic difference in degrees. But it is a poor approximation of that ordering for graph with very large maximum degrees and many vertices of large degrees.

We suggest a third generation algorithm called `Exponential` where vertex v draws a random number in $[0; 2^{\delta(v)})$. This algorithm still has the same communication and computational cost with an additional benefit: the probability that a vertex v has a random number $r(v)$ greater than all vertices u such that $\delta(u) = \delta(v) - 1$ is greater than $\frac{1}{2}$. Therefore, it is a better approximation of the Largest First ordering.

B. Basic analysis

Regular graphs have the property that all vertices have the same degree. This category of graphs encompasses many typical structures. Cliques, cycles, torus (2d, 3d, or arbitrary dimension) are all regular graphs. Because all vertices have the same degree, all three algorithms behave in exactly the same way. Although each method generates random numbers in different intervals, all vertices in that method generate numbers in the same interval. Consequently, all three algorithms behave in the same way.

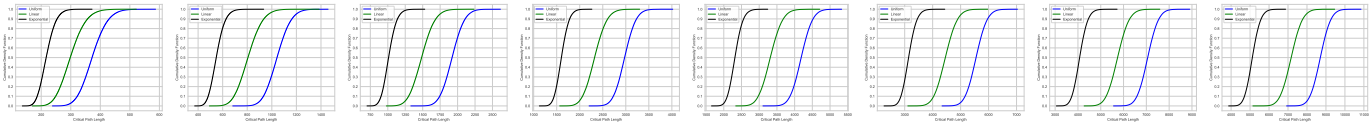
Star graphs show why the methods work differently. Consider a star graph of V vertices. The center vertex (hub) has a degree of $V - 1$, while all other vertices (spokes) have a degree of 1. There are only two possible solutions (excluding symmetries) that can be generated by the distributed algorithm. Either the hub vertex is in between two of the spokes, or it is not. It does not matter if the hub is before all spokes or after all spokes. If the hub vertex is between the two spokes, the longest chain has 3 vertices; otherwise, it has 2 vertices.

`Uniform` will put the spoke vertex first with probability $\frac{1}{V}$, and will put it last will probability $\frac{1}{V}$. As $V \to \infty$, the algorithm will generate a path of 2 vertices with probability $\frac{2}{V}$ and a path of 3 vertices with probability $\frac{V-2}{V}$.

On the other hand, `Exponential` will force all spoke vertices have random numbers in $[0; 2)$, while the hub vertex will be a random number in $[0; 2^{V-1})$. The hub will have a random number greater than 2 with probability $\frac{2^{V-1}-2}{2^{V-1}}$. In all cases, the longest chain will be of 2 vertices with a probability greater than $\frac{2^{V-1}-2}{2^{V-1}}$, which goes to 1 as V goes to infinity.

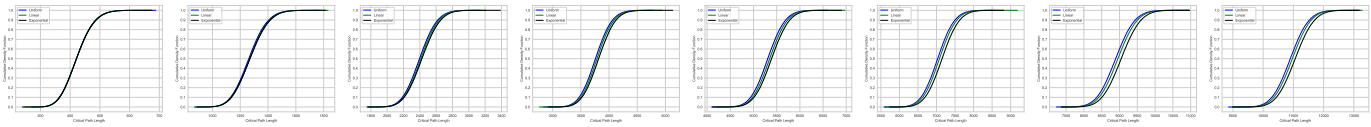
In the case of the `Linear` algorithm, the spokes have random numbers in $[0; 1)$, while the hub has a number taken in $[0; V - 1)$. The hub has a random number greater than 1 with probability $\frac{V-2}{V-1}$. And so, the probability of having a path of 2 vertices tends to 1 as V approaches infinity.

Although star graph are not commonly found in real world applications, many graphs, such as social networks, are similar to star graphs: they are structured like onions with dense center



(a) $ef = 4$ (b) $ef = 8$ (c) $ef = 12$ (d) $ef = 16$ (e) $ef = 20$ (f) $ef = 24$ (g) $ef = 28$ (h) $ef = 32$

Fig. 2: Cumulative Density Function of the longest chain induced by `Uniform`, `Expnential` and `Linear` on RMAT Graph with a = 0.10, b = 0.20, c = 0.50, d = 0.20 for different values of edge factor ef. The different values of edge factor show almost identical patterns for the length of the critical path.



(a) $ef = 4$ (b) $ef = 8$ (c) $ef = 12$ (d) $ef = 16$ (e) $ef = 20$ (f) $ef = 24$ (g) $ef = 28$ (h) $ef = 32$

Fig. 3: Cumulative Density Function of the longest chain induced by `Uniform`, `Expnential` and `Linear` on RMAT Graph with a = 0.42, b = 0.19, c = 0.19, d = 0.02 for different values of edge factor ef. The different values of edge factor show almost identical patterns for the length of the critical path.

regions and layers of ever lesser dense regions. We believe that an algorithm like `Exponential` favors shorter paths in social networks because once a path enters a denser region is entered it tends not to exit it.

V. Study on Recursive Graph Model (RMAT)

A. Methodology

RMAT graphs have 2^n nodes. They are constructed by recursively splitting a square matrix into 4 quadrants: a, b, c, d. Each quadrant has an associated probability that a given edge will fall into that quadrant, so $a + b + c + d = 1$. Edges are generated one at a time and placed in a quadrant following the given probabilities, and recursively until the edge is placed in a 1×1 sub matrix. The number of edges in a graph is usually controlled by setting an edge factor ef which will generate $ef * 2^n$ edges.

RMAT graphs have several desired properties of many real graphs. They have a power-law degree distribution that resembles real graphs in several applications. They also exhibit a community structure and have a small diameter [22]. RMAT graphs are the 2×2 special case of Kronecker graph [23]. Note that RMAT is a directed graph model, however we need an undirected graph, so the matrix is symmetrized after generation.

All the studies that we conduct on RMAT graphs make the same assumption. We assume that our graph is derived from an underlying RMAT distribution, and we try to ascertain whether `Uniform`, `Linear`, or `Exponential` would obtain a shorter longest chain. Since RMAT is a probabilistic model and these methods themselves are randomized, statistical evidence is required. For an RMAT parameter set and an algorithm, we estimate the probability density function by sampling 1000 RMAT graphs generated by these parameters, and for each graph that was generated, by sampling 100 executions of each ordering algorithm. This gives us $100k$ values of longest path length for each ordering for a particular set of parameters.

In general, we present the sampled Cumulative Density Function (CDF) of the longest chain and the confidence intervals for the expected length of the longest chain. We validated the significance of the difference in the expected length of longest chains between two orderings with a pairwise two-population z-test. The p-value was always substantially lower than 0.05 which indicates that all results presented on RMAT graphs are statistically significant.

B. Initial Investigation

As there seem to be no real consensus on what RMAT parameters to use to benchmark algorithms, we started our exploration by considering two sets of RMAT parameters that appear frequently in the literature. We have four initial questions. Does the ordering method make a difference? Does one of the methods lead to shorter path? How does edge factor impact the results? Do the parameters a, b, c, and d make a difference?

We used parameters $(0.10, 0.20, 0.50, 0.20)$ and $(0.42, 0.19, 0.19, 0.02)$. We varied the edge factor between 4 and 32. We present the Cumulative Density Function of the length of the longest path in Figures 2 and 3.

The statistic tests showed that the distribution are statistically significantly different. The difference on the RMAT parameters $(0.1, 0.2, 0.5, 0.2)$ was fairly important. It seems that for all edge factors, `Linear` leads to shorter longest paths than `Uniform` and `Exponential` leads to shorter longest paths than both of them. However, the difference on the RMAT parameters $(0.42, 0.19, 0.19, 0.02)$ was very small. (Even though it was statistically significant.)

The edge factor seems to have little impact on the relative performance of each method. The difference between orderings appears to be more pronounced for larger edge factor; however, the trends remained the same.

a	b	c	d	Uniform CI	Exponential CI	Linear CI	U/E	U/L	L/E
0.30	0.28	0.28	0.14	[2944; 2947]	[2623; 2625]	[2850; 2853]	1.123	1.033	1.087
0.40	0.24	0.24	0.12	[4950; 4953]	[4680; 4683]	[4859; 4862]	1.058	1.019	1.038
0.50	0.20	0.20	0.10	[6968; 6971]	[6643; 6647]	[6845; 6848]	1.049	1.018	1.030
0.60	0.16	0.16	0.08	[8353; 8357]	[7949; 7952]	[8175; 8178]	1.051	1.022	1.028
0.70	0.12	0.12	0.06	[9211; 9214]	[8738; 8740]	[8987; 8990]	1.054	1.025	1.029
0.30	0.28	0.17	0.25	[2111; 2113]	[2064; 2066]	[2103; 2105]	1.023	1.004	1.019
0.30	0.28	0.25	0.17	[2633; 2635]	[2437; 2439]	[2583; 2585]	1.080	1.019	1.060
0.30	0.28	0.34	0.08	[3782; 3785]	[3112; 3115]	[3510; 3513]	**1.215**	1.077	1.128
0.30	0.35	0.14	0.21	[2625; 2627]	[2047; 2049]	[2402; 2404]	**1.282**	1.093	**1.173**
0.30	0.35	0.21	0.14	[3064; 3067]	[2464; 2467]	[2825; 2827]	**1.243**	1.085	1.146
0.30	0.35	0.28	0.07	[3959; 3962]	[3221; 3225]	[3644; 3647]	**1.229**	1.086	1.131
0.30	0.42	0.11	0.17	[3632; 3635]	[2181; 2183]	[2880; 2883]	**1.665**	**1.261**	**1.321**
0.30	0.42	0.17	0.11	[3821; 3824]	[2433; 2436]	[3061; 3064]	**1.570**	**1.248**	**1.258**
0.30	0.42	0.22	0.06	[4343; 4346]	[3110; 3113]	[3685; 3688]	**1.396**	**1.178**	**1.185**
0.30	0.49	0.08	0.13	[4852; 4855]	[2245; 2249]	[3437; 3441]	**2.160**	**1.411**	**1.530**
0.30	0.49	0.13	0.08	[4775; 4778]	[2505; 2508]	[3325; 3330]	**1.906**	**1.435**	**1.328**
0.30	0.49	0.17	0.04	[5052; 5056]	[3151; 3155]	[3883; 3887]	**1.603**	**1.301**	**1.232**
0.40	0.24	0.14	0.22	[3246; 3249]	[3185; 3188]	[3242; 3245]	1.019	1.001	1.018
0.40	0.24	0.22	0.14	[4571; 4574]	[4377; 4380]	[4509; 4512]	1.044	1.014	1.030
0.40	0.24	0.29	0.07	[5946; 5950]	[5500; 5504]	[5759; 5763]	1.081	1.032	1.047
0.40	0.30	0.12	0.18	[4026; 4029]	[3457; 3460]	[3811; 3814]	**1.165**	1.056	1.102
0.40	0.30	0.18	0.12	[5003; 5006]	[4551; 4555]	[4821; 4825]	1.099	1.038	1.059
0.40	0.30	0.24	0.06	[6141; 6144]	[5652; 5656]	[5932; 5935]	1.086	1.035	1.049
0.40	0.36	0.10	0.14	[4903; 4906]	[3767; 3771]	[4333; 4336]	**1.301**	1.132	1.150
0.40	0.36	0.14	0.10	[5506; 5509]	[4637; 4641]	[5071; 5075]	**1.187**	1.086	1.094
0.40	0.36	0.19	0.05	[6383; 6387]	[5684; 5688]	[6045; 6049]	1.123	1.056	1.063
0.40	0.42	0.07	0.11	[5667; 5670]	[3901; 3906]	[4639; 4643]	**1.452**	**1.221**	**1.189**
0.40	0.42	0.11	0.07	[6196; 6199]	[4927; 4932]	[5478; 5483]	**1.257**	1.131	1.112
0.40	0.42	0.14	0.04	[6670; 6674]	[5681; 5685]	[6132; 6136]	**1.174**	1.088	1.079
0.50	0.20	0.12	0.18	[5009; 5012]	[4881; 4884]	[4981; 4984]	1.026	1.006	1.020
0.50	0.20	0.18	0.12	[6489; 6492]	[6213; 6216]	[6399; 6402]	1.044	1.014	1.030
0.50	0.20	0.24	0.06	[7813; 7816]	[7365; 7368]	[7616; 7620]	1.061	1.026	1.034
0.50	0.25	0.10	0.15	[5687; 5690]	[5230; 5234]	[5515; 5519]	1.087	1.031	1.054
0.50	0.25	0.15	0.10	[6920; 6924]	[6500; 6504]	[6748; 6751]	1.065	1.026	1.038
0.50	0.25	0.20	0.05	[7984; 7987]	[7503; 7507]	[7766; 7770]	1.064	1.028	1.035
0.50	0.30	0.08	0.12	[6294; 6297]	[5524; 5528]	[5943; 5946]	1.139	1.059	1.076
0.50	0.30	0.12	0.08	[7263; 7267]	[6644; 6648]	[6969; 6972]	1.093	1.042	1.049
0.50	0.30	0.16	0.04	[8073; 8076]	[7495; 7499]	[7786; 7789]	1.077	1.037	1.039
0.50	0.35	0.06	0.09	[6812; 6815]	[5778; 5781]	[6278; 6281]	**1.179**	1.085	1.087
0.50	0.35	0.09	0.06	[7537; 7540]	[6729; 6732]	[7108; 7111]	1.120	1.060	1.056
0.50	0.35	0.12	0.03	[8135; 8138]	[7420; 7423]	[7754; 7757]	1.096	1.049	1.045
0.60	0.16	0.10	0.14	[6491; 6495]	[6204; 6208]	[6406; 6409]	1.046	1.013	1.032
0.60	0.16	0.14	0.10	[7774; 7777]	[7418; 7422]	[7631; 7635]	1.048	1.019	1.029
0.60	0.16	0.19	0.05	[9077; 9080]	[8613; 8616]	[8843; 8846]	1.054	1.026	1.027
0.60	0.20	0.08	0.12	[6942; 6945]	[6427; 6431]	[6741; 6745]	1.080	1.030	1.049
0.60	0.20	0.12	0.08	[8257; 8260]	[7803; 7806]	[8044; 8048]	1.058	1.026	1.031
0.60	0.20	0.16	0.04	[9253; 9256]	[8754; 8757]	[8997; 9000]	1.057	1.028	1.028
0.60	0.24	0.06	0.10	[7261; 7264]	[6516; 6519]	[6926; 6929]	1.114	1.048	1.063
0.60	0.24	0.10	0.06	[8597; 8600]	[8041; 8044]	[8308; 8311]	1.069	1.035	1.033
0.60	0.24	0.13	0.03	[9283; 9286]	[8731; 8734]	[8987; 8990]	1.063	1.033	1.029
0.60	0.28	0.05	0.07	[7829; 7832]	[6958; 6962]	[7380; 7383]	1.125	1.061	1.061
0.60	0.28	0.07	0.05	[8537; 8540]	[7838; 7841]	[8157; 8160]	1.089	1.047	1.041
0.60	0.28	0.10	0.02	[9249; 9252]	[8631; 8634]	[8900; 8903]	1.072	1.039	1.031
0.70	0.12	0.07	0.11	[7229; 7232]	[6852; 6856]	[7101; 7105]	1.055	1.018	1.036
0.70	0.12	0.11	0.07	[8854; 8857]	[8424; 8427]	[8659; 8662]	1.051	1.023	1.028
0.70	0.12	0.14	0.04	[9813; 9816]	[9197; 9200]	[9514; 9517]	1.067	1.031	1.034
0.70	0.15	0.06	0.09	[7797; 7800]	[7252; 7255]	[7573; 7576]	1.075	1.030	1.044
0.70	0.15	0.09	0.06	[9093; 9096]	[8589; 8592]	[8850; 8853]	1.059	1.027	1.030
0.70	0.15	0.12	0.03	[10032; 10035]	[9347; 9350]	[9694; 9696]	1.073	1.035	1.037
0.70	0.18	0.05	0.07	[8267; 8270]	[7587; 7591]	[7941; 7945]	1.090	1.041	1.047
0.70	0.18	0.07	0.05	[9183; 9186]	[8599; 8602]	[8876; 8879]	1.068	1.035	1.032
0.70	0.18	0.10	0.02	[10074; 10076]	[9370; 9372]	[9700; 9702]	1.075	1.039	1.035
0.70	0.21	0.04	0.05	[8667; 8669]	[7899; 7902]	[8263; 8266]	1.097	1.049	1.046
0.70	0.21	0.05	0.04	[9168; 9171]	[8503; 8506]	[8798; 8800]	1.078	1.042	1.035
0.70	0.21	0.07	0.02	[9844; 9846]	[9198; 9200]	[9470; 9472]	1.070	1.039	1.030

TABLE I: Critical path length (95% confidence intervals) and ratios of average critical path lengths across methods for different RMAT parameters. (Bolded numbers highlight critical path length ratios greater than 1.15.)

C. Exploring the RMAT Parameter Space

Our initial investigation revealed that the a, b, c, and d parameters are important, while edge factor did not appear to be important. `Exponential` seems to lead to shorter path than `Linear`; and `Uniform` seems leads to longer path. The question is whether the parameters we used were odd cases or whether the trends hold for any RMAT graph.

In this section, we explore the RMAT parameter space in a systematic fashion. We generated a set of 65 different parameters on graphs of with 2^9 vertices and fixed the edge factor to 16. We selected values of a in the range $[.3, .8]$ regularly. Subsequently, other parameters were selected as a fraction of the remaining number of edges. The remaining cases were seperated into $b > c$ and $b = c$. The precise values of a, b, c, d that we used are included in Table I.

Table I also provides the confidence interval of the average length of the longest chain for all three orderings. It also provides the ratio of the average longest chain path between `Uniform` and `Exponential`, between `Uniform` and `Linear`, and between `Linear` and `Exponential`. We highlighted in bold the ratios that are greater than 1.15.

Since none of the ratios were smaller than 1 and all results were statistically significant, it does seem that on RMAT graphs, `Exponential` is better than `Linear`, which is itself better than `Uniform`. `Exponential` leads to path less than half the length of `Uniform` path on average with RMAT parameters $(0.30, 0.49, 0.08, 0.13)$. Overall, `Exponential` obtained a path at least 15% better than `Uniform` on 17 different RMAT parameters.

While it is unclear how the different parameters control for the difference in path length, it appears that smaller values of the a parameter seem to favor the `Exponential` ordering.

VI. Real Graph Study

While it is encouraging that the `Exponential` ordering leads to shorter longest path on RMAT graphs, it does not necessarily hold that the result will be the same given graph extracted from real world applications. We tested several real world graphs from the Stanford Network Analysis Project (SNAP): CA-HepPh, Email-Enron, p2p-Gnutella04, roadNet-PA, soc-Epinions1, soc-pokec-relationships, web-Google, and WikiTalk. A summary of the properties of these graphs are given in Table II. All these graphs have small world properties, except the graph of the roads of Pennsylvania, which is almost a regular graph. We included that graph as a control.

We present the summary statistics of the orderings in Table II and the Cumulative Density Function of the length of the longest path in Figure 4.

All the results are statistically significant. On two graphs, `Exponential` does not lead to the smallest longest chain in average: ca-HepTh and roadNet-PA. Alhough the difference in distribution is fairly small and the average length only differs by less than 2%.

On the other graphs, `Exponential` leads to longest chains shorter than `Uniform` by more than 7% in average, and by more than 15% on 4 of the graphs. Surprisingly, the average longest chain generated by `Uniform` is almost 4.5 times longer than the average longest chain generated by `Exponential`. `Linear` overall, sits in between `Uniform` and `Exponential`.

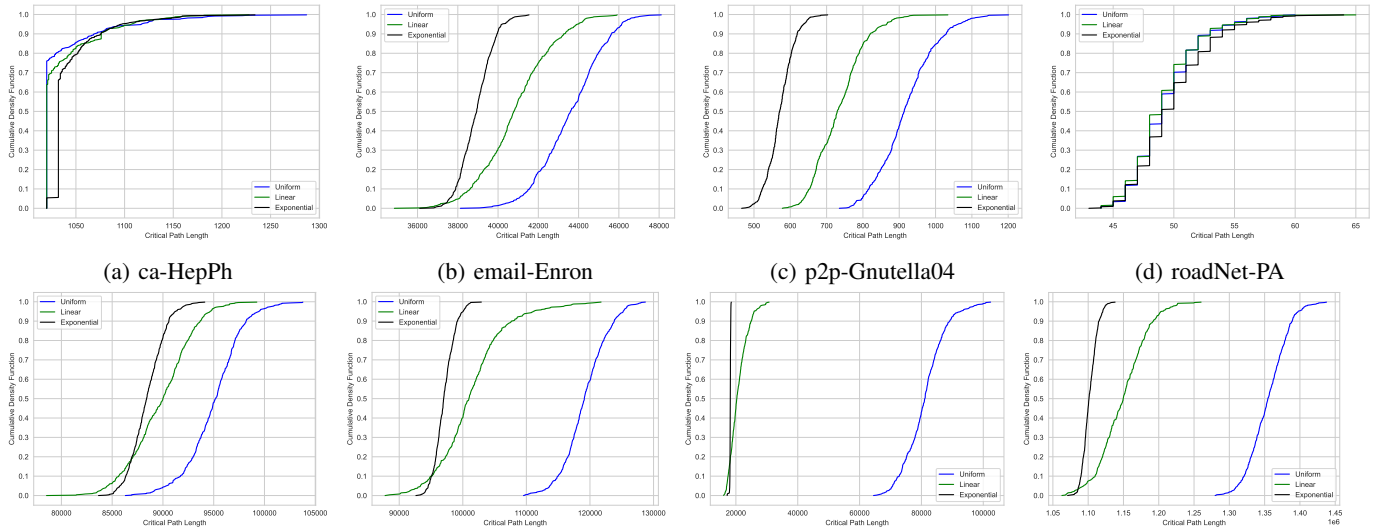
While we expected to see `Exponential` lead to much shorter longest chains than `Uniform`, it is not clear yet to why ca-HepPh does not follow the same trend. We hypothesize that ca-HepPh has one large cluster of vertices which is mostly completely connected. And as such, behaves in practice similarly to a clique.

VII. Conclusion

In this paper we investigated the performance of distributed dataflow graph algorithms. We modeled the problem of optimizing the critical path of the partial order used by the algorithm using a formulation as a coloring problem with

Name	Vertices	Edges	Max Degree	Clustering Coefficient	Diameter	Uniform CI	Exponential CI	Linear CI	U/E	U/L	L/E
CA-HepPh	89,209	118,521	491	0.6115	13	[1030; 1036]	[1040; 1045]	[1032; 1037]	0.991	0.999	0.992
Email-Enron	36,692	183,831	1,383	0.4970	11	[43437; 43720]	[38836; 38982]	[40688; 41002]	1.120	1.067	1.050
p2p-Gnutella04	10,879	39,994	103	0.0062	9	[911; 925]	[568; 575]	[728; 740]	**1.606**	**1.251**	**1.284**
roadNet-PA	1,090,920	1,541,898	9	0.0465	786	[49; 49]	[49; 50]	[48; 49]	0.990	1.010	0.980
soc-Epinions1	75,888	405,740	3,044	0.1378	14	[94793; 95270]	[88297; 88593]	[89488; 90034]	1.074	1.059	1.015
soc-pokec-relationships	1,632,804	22,301,964	14,854	0.1094	11	[118924; 119528]	[96958; 97239]	[100836; 101775]	**1.228**	**1.177**	1.043
web-Google	916,428	4,322,051	6,332	0.5143	21	[80466; 81618]	[18166; 18192]	[20577; 21084]	**4.458**	**3.891**	1.146
WikiTalk	2,394,385	4,659,565	100,029	0.0526	9	[1352414; 1357165]	[1101248; 1103043]	[1145942; 1151894]	**1.229**	**1.179**	1.042

TABLE II: Graph basic statistics, critical path length (95% confidence intervals), and ratios of average critical path lengths across methods for several real world graphs. (Bolded numbers highlight critical path length ratios greater than 1.15.)



(a) ca-HepPh (b) email-Enron (c) p2p-Gnutella04 (d) roadNet-PA

(e) soc-Epinions1 (f) soc-pokec (g) web-Google (h) wiki-Talk

Fig. 4: Cumulative Density Functions of longest chain on Real World Graphs. All graphs (except ca-HepTh and roadNet-PA) show a major difference in critical path length across methods: `Exponential` and `Linear` have much shorter critical paths than `Uniform`.

intervals of colors. We proposed two alternative ways to derive a partial order, `Exponential` and `Linear`. These methods rely on local properties of the vertices, which enable these orderings to run with no additional cost.

We investigated the efficacy of these algorithms on a large number of RMAT graphs. We showed that `Exponential` outperforms the state of the art on all tested RMAT parameters. We also tested the `Exponential` algorithm on 8 real world graphs and showed it never loses more than 2% to state of the art and reduce the longest chain by more than 20% on 4 of these graphs.

Two important questions remain. The first is to understand more precisely why `Exponential` is better than state of the art; We believe that investigating the behavior of the algorithm relatively to the k-core decomposition of the graph might yield more. insight. Finally, we also want to experimentally measure how the reduction in longest chain decrease the practical runtime of these dataflow algorithms.

ACKNOWLEDGMENT

This work is supported by a grant from the National Science Foundation number CCF-1652442.

REFERENCES

[1] U. V. Çatalyürek, J. Feo, A. H. Gebremedhin, M. Halappanavar, and A. Pothen, "Graph coloring algorithms for multi-core and massively multithreaded architectures," *Parallel Comput.*, vol. 38, no. 10–11, p. 576–594, oct 2012.

[2] A. E. Sarıyüce, E. Saule, and U. V. Catalyurek, "Scalable hybrid implementation of graph coloring using MPI and OpenMP," in *26th International Symposium on Parallel and Distributed Processing, Workshops and PhD Forum (IPDPSW), Workshop on Parallel Computing and Optimization (PCO)*, May 2012.

[3] M. Luby, "A simple parallel algorithm for the maximal independent set problem," *SIAM Journal on Computing*, vol. 15, no. 4, pp. 1036–1053, 1986.

[4] M. T. Jones and P. E. Plassmann, "A parallel graph coloring heuristic," *SIAM Journal on Scientific Computing*, vol. 14, no. 3, pp. 654–669, 1993.

[5] G. E. Blelloch, J. T. Fineman, and J. Shun, "Greedy sequential maximal independent set and matching are parallel on average," in *Proceedings of the Twenty-Fourth Annual ACM Symposium on Parallelism in Algorithms and Architectures*, ser. SPAA '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 308–317.

[6] R. R. McCune, T. Weninger, and G. Madey, "Thinking like a vertex: A survey of vertex-centric frameworks for large-scale distributed graph processing," *ACM Comput. Surv.*, vol. 48, no. 2, oct 2015.

[7] Y.-K. Kwok and I. Ahmad, "Static scheduling algorithms for allocating directed task graphs to multiprocessors," *ACM Comput. Surv.*, vol. 31, no. 4, pp. 406–471, Dec. 1999.

[8] R. L. Graham, "Bounds on multiprocessing timing anomalies," *SIAM*

Journal on Applied Mathematics, vol. 17, no. 2, pp. 416–429, Mar. 1969.

[9] D. B. West, *Introduction to Graph Theory*. Prentice-Hall, 1996.

[10] Y. Asahiro, J. Jansson, E. Miyano, and et al., "Approximation algorithms for the graph orientation minimizing the maximum weighted outdegree," *J Comb Optim*, vol. 22, pp. 78–96, 2011.

[11] R. Cole and U. Vishkin, "Deterministic coin tossing with applications to optimal parallel list ranking," *Information and Control*, vol. 70, no. 1, pp. 32–53, 1986. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0019995886800237

[12] K. Appel and W. Haken, "Every planar map is four-colorable, ii: Reducibility," *Illinois J. Math.*, no. 21, pp. 491–567, 1977.

[13] M. R. Garey and D. S. Johnson, *Computers and Intractability*. Freeman, San Francisco, 1979.

[14] D. Zuckerman, "Linear degree extractors and the inapproximability of max clique and chromatic number," *Theory of Computing*, vol. 3, pp. 103–128, 2007.

[15] A. H. Gebremedhin, F. Manne, and A. Pothen, "What color is your jacobian? Graph coloring for computing derivatives," *SIAM Review*, vol. 47, no. 4, pp. 629–705, 2005.

[16] D. W. Matula and L. L. Beck, "Smallest-last ordering and clustering and graph coloring algorithms," *J. ACM*, vol. 30, pp. 417–427, July 1983.

[17] D. Brélaz, "New methods to color the vertices of a graph," *Commun. ACM*, vol. 22, pp. 251–256, April 1979.

[18] D. J. A. Welsh and M. B. Powell, "An upper bound for the chromatic number of a graph and its application to timetabling problems," *The Computer Journal*, vol. 10, no. 1, pp. 85–86, 1967.

[19] D. Orden, J. M. Gimenez-Guzman, I. Marsa-Maestre, and E. De la Hoz, "Spectrum graph coloring and applications to wi-fi channel assignment," *Symmetry*, vol. 10, no. 3, 2018.

[20] D. Durrman and E. Saule, "Coloring the vertices of 9-pt and 27-pt stencils with intervals," in *Proc. Of IPDPS*, May 2022.

[21] A. E. Sarıyüce, E. Saule, and U. V. Catalyurek, "Improving graph coloring on distributed memory parallel computers," in *18th Annual International Conference on High Performance Computing*, 2011.

[22] C. F. Deepayan Chakrabarti, Yiping Zhany, "R-mat: A recursive model for graph mining," in *Proc. of SIAM International Conference on Data Mining (SDM)*, 2004.

[23] J. Leskovec, D. Chakrabarti, J. Kleinberg, C. Faloutsos, and Z. Ghahramani, "Kronecker graphs: an approach to modeling networks," *Journal of Machine Learning Research*, vol. 11, pp. 985–1042, 2010.