Disjoint Set Union Using Relaxed Scheduling

Martin Hafskjold Thoresen

December 4, 2018

1 Introduction

1.1 Relaxed Scheduling

By Claim 1 in [1] we know that for a k-relaxed queue the top element is always removed within $\mathcal{O}(k)$ in expectation.

1.2 Our Problem

The problem we are considering is as follows: we have a relaxed queue Q of tasks, which are operations to the Disjoint-Set problem. The tasks are ordered by some ordering π. We want to look at the probability of dequeueing a task t from Q that "depends" on another task t' in Q with a lower rank, meaning t' is before t in π. When we get such a task we have to insert it back into Q, as we want to respect the ordering that π imposes on us, if there is a dependency between the tasks. By finding this we want to bound the number of times we have to re-insert something into Q.

The dependencies between tasks form a graph G, in which the vertices are tasks, and there is an edge between two tasks if they depend on each other.

2 Independent Collisions

One way to go about this problem is to find the probability that any two uniformly sampled tasks collide: $\Pr_{u,v\sim E}[u \otimes v]$. If we have this we can bound the number of expected collisions. We begin with defining what we mean by a collision.

Definition 1 (Edge-collision)**.** *Two edges collide if they touch the same component in exactly one of their endpoints, and this component is smaller or equal in size to the two others. When u and v collides we say $u \otimes v$.*

2.1 The Structure of Random Graphs

Recall that $G = (V, E)$ is a graph in which the vertices are tasks and the edges are dependencies between them. We assume that this graph is random. Erdős–Rényi showed[2] the structure of random graphs asymptotically in the number of nodes. In their model each edge is randomly inserted with a probability of p, *or* each graph on m edges is uniformly selected; these two models are equivalent for the following results[1]. Let $n = |V|$ and $m = |E|$. We look at the structure of G given the size of m:

1. $m \in [0,\ (\frac{1}{2} - \epsilon)n]$ No component in G are of size more than $O(\log n)$.
2. $m \in [(\frac{1}{2} - \epsilon)n,\ (\frac{1}{2} + \epsilon)n]$ One large component is of increasing size. All other components are of size $O(\log n)$.

[1] TODO: I think?

3. $m \in [(\frac{1}{2} + \epsilon)n,\ n^2)$ The largest component is a giant, consisting of $n^{2/3}$ vertices. No other component is of size larger than $O(\log n)$

Now we want the probability that two randomly selected edges, that are not yet in G, collide. We note that the second largest component in all cases is $O(\log n)$, and by Definition 1 we cannot have a collision in the largest component; this simplifies the analysis since we can get the same bound on the number of collisions in all three cases. Fix the first edge u. If u is internal to some component we may discard it, so we assume that it is not. If u touches the largest component of G we must have the collision in the remaining endpoint, which reduces the probability of a collision by a factor of 2. We therefore assume that u connectes two components neither of which are the largest. In order to have $u \otimes v$, exactly one of the endpoints of v must be in either of the components that u touch. Since the sizes of all components except the giant is $\mathcal{O}(\log n)$ the probability that either endpoint of v is "incident" to u is:

$$\Pr_{u,v\sim E}[u \otimes v] \leq \mathcal{O}(2 \log n/n) \tag{1}$$

2.2 The Number of Collisions

Fix the time at $t = \ell$. By Claim 1 we expect to have to pop $\mathcal{O}(k)$ elements from the queue before getting the top element. All of these $\mathcal{O}(k)$ elements have rank $\mathcal{O}(k)^2$, so there are $\mathcal{O}(k^2)$ pairs of tasks we have to check. Each pair has a probability $\mathcal{O}(2 \log n/n)$ by eq. (1), so the expected number of collisions at time $t = \ell$ is

$$\mathbb{E}[\text{collisions at } t = \ell] \leq \mathcal{O}(2k^2 \log n/n). \tag{2}$$

Summing over all time steps ℓ we get

$$\mathbb{E}[\text{collisions}] \leq \sum_{\ell=1}^{m} \mathcal{O}(2k^2 \log n/n) = \mathcal{O}\left(\frac{m}{n} k^2 \log n\right). \tag{3}$$

2.2.1 Successful Pops

If we do get a taks that has no outstanding dependencies we can process it straight away, making the limit of the sum in eq. (3) decrease by 1. What is the probability of this happening? Let t be the popped task. t cannot collide with any of the $\mathcal{O}(k)$ tasks before it in Q:

$$\Pr[\text{no-collisions}] \leq (1 - \mathcal{O}(2k \log n/n))^{\mathcal{O}(k)}$$

To get a rough idea of how this looks, we discard the $\mathcal{O}$s, and let $k = \log n$:

$$\Pr[\text{no-collisions}] \approx (1 - 2\log^2 n/n)^{\log n}$$

(According to Mathematica, this goes to 1, which means for large n we are almost guaranteed to have no collisions)

Comments

This is slightly worse than Theorem 3.1 in [1], in which the expected number of collisions in the general framework is $\mathcal{O}\left(\frac{m}{n}\right)$ poly(k). It is not clear why this is, except for the fact that the above analysis is probably too pessimistic?

[2]TODO: double check

3 Constraints on Sampled Edges

In Section 2 we assumed that the structure of the connectivity graph was totally random. It is not clear however that this is the upper bounds for other types of structures on the Union operations. In this section we look at other possible *Sampling Graphs*, from which the edges are sampled.

Let F be the connectivity forest on which we are solving the Union-Find problem, and let S be a graph on the same vertex set, but with an edge set E that is different from the complete set. We still have an element of randomness, namely the ordering of the operations. We look at different families of Es in order to try to find the worst case.

Metric of Fitness

First we must define our metric of fitness. Intuitively we are trying to minimize the number of failed Pops from the task queue, but we are also not willing to have few tasks (eg. let $Q = \emptyset$, and we have no collisions!). Let the *fitness* of a graph $\mathcal{F}(G)$ be the ratio the number of tasks performed before no more collisions can take place to the number of expected collisions. The numerator is usually the number of sampled edges before the graph is connected. In Section 2 we considered $G = K_n$. Then the fitness of K_n is

$$\mathcal{F}(K_n) = \mathcal{O}\left(\frac{m}{\frac{m}{n}k^2 \log n}\right) = \mathcal{O}\left(\frac{n}{k^2 \log n}\right)$$

Intuition

The general idea is that the sampling graph is the target for all Union operations: each operation brings the connectivity forest one step closer to becomming S. Unless S itself is a forest, the connectivity forest will not really become S, since we ignore Unions that are internal to an already connected component (Eg. with $S = K_n$, F ends up as a spanning tree, not K_n).

Maybe some clever choice of S can force a random selection of edges into maximizing the number of collisions, while still having a sizable set of tasks.

3.1 The Cycle Graph

For each fixed edge u there are exactly two edges in the graph that are indicent. By relaxing the definition of a collision to include these edges the probability of sampling an edge v such that $u \otimes v$ is $\Pr_{u,v\sim E}[u \otimes v] \le 2/(n-\ell)$, where ℓ is the iteration time. Since we require exactly $n-1$ edges for the cycle to become connected, we have $m = n-1$ Ignoring that edges may be succesfully out-of-ordered processed the number of expected collisions is

$$\sum_{\ell=1}^{n-1} 2k/(n-\ell) = 2kH_{n-1} = \mathcal{O}(k \log n)$$

where H_n is the nth harmonic number. The the fitness is

$$\mathcal{F}(C_n) = \mathcal{O}\left(\frac{n-1}{k \log n}\right)$$

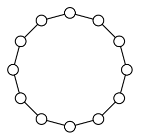

Figure 1: The Cycle Graph C_{12}

3.2 The Star Graph

The Star graph S_n is the complete bipartite graph $K_{1,n}$, and it is an example of a graph with (almost) no collisions. With the exception of the first edge sampled all edges connects a single vertex to the largest component in the graph, so any pair of edges shares the largest component.

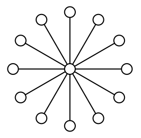

Figure 2: The Star Graph S_{12}

Since the probability of a collision is zero after the first edge is sampled the expected number of collisions *in total* is the same as the expected number of dequeues from the task queue before getting the first element, which is $\leq 1/(1 - e^{-1/k}) = \mathcal{O}(k)$. Since S_n is a tree we need $n - 1$ edges to connect it. Thus the fitness is

$$\mathcal{F}(S_n) = \mathcal{O}\left(\frac{n-1}{k}\right)$$

3.3 The Barbell Graph

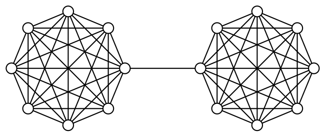

Figure 3: The Barbell graph B_8

The barbell graph $B_n = (V, E)$ consists of two complete graphs K_n connected by a single edge, so $|V| = 2n$ and $|E| = 2\binom{n}{2} + 1 = n(n-1) + 1$. Instead of considering the number of edges we must sample in expectation before connecting B_n we split the analysis into two parts. Let e_c be the edge that connects the two complete graphs in B_n, and look at the graph after sampling $\mathcal{O}(n \log n)$ edges, and show that no collisions take place after this point:

Case 1: e_c is not sampled. We expect both complete graphs to have $\mathcal{O}(n \log n)$ edges, which makes them connected in expectation by Erdős–Rényi. Since we only have two components, no more collisions in the graph are possible.

Case 2: e_c is sampled. We again expect that both of the complete graphs are connected, which makes the entire B_n connected.

Next we must count the number of expected collisions within the first $\mathcal{O}(n \log n)$ sampled edges. If e_c is not sampled we expect to have twice the collisions compared to having a single K_n, since we have two independent K_n. If e_c is sampled the graph is depicted in Figure 4.

We wish to find the probability that two randomly sampled edges collide: $\Pr_{u,v \sim E}[u \otimes v]$. Name the components that e_c connects A and B, and let the largest component in each complete graph be M_i. The probability of a collision not including $A \cup B$ is the same as when $G = K_n$, so we only need to consider collisions that include $A \cup B$. There are two cases to consider: (1) $A \cup B$ is the component the colliding edges share, and (2) $A \cup B$ is larger than some other component in the collision.

For the first case we note that the size of $A \cup B$ is $\mathcal{O}(\log n)$, since neither A nor B was not the largest component in their

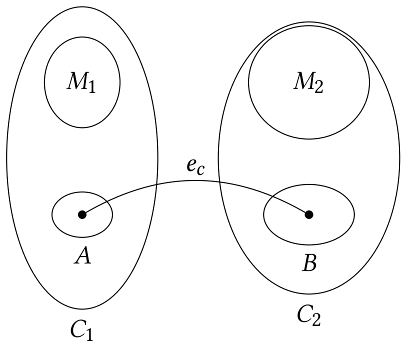

Figure 4: The state of B_n after e_c has been sampled.

respective graphs prior to the join, so the probability of a collision is bounded by the probability that both edges hit $A \cup B$:

$$\begin{aligned}\Pr_{u,v\sim E}[u \otimes v \wedge u \sim A \cup B \wedge v \sim A \cup B] &\leq \Pr_{u,v\sim E}[u \sim A \cup B \wedge v \sim A \cup B] \\ &= \left(\frac{2\mathcal{O}(\log n)}{n}\right)\left(\frac{2\mathcal{O}(\log n)}{n}\right) \\ &= \mathcal{O}\left(\frac{\log^2 n}{n^2}\right)\end{aligned}$$

which is less probable than a random collision.

For the second case we have $A \cup B$ being the largest component in the collision. Without loss of generality let C_1 be the subgraph in which the collisions happens. The addition of e_c matters in the analysis compared to the case without it because there might be a component S_1 such that $|S_1| > |A|$ but $|S_1| \leq |A \cup B|$: now edges (m_1, s_1) and (a, s_1) might collide, whereas before adding e_c they could not. Note that if A is the largest component in C_1 it does not matter if we have sampled e_c or not, so we may assume that A is not the largest component, so that its size is $\mathcal{O}(\log n)$. In the worst case B is the largest component in C_2 and $A \cup B$ is larger than M_1. Now any edge $u \in C_1$ have $\mathcal{O}(\log^2 n)$ potential edges with which it can collide, namely all edges from its smaller component to A. The probability that any of these edges are sampled next, and thus that $u \otimes v$ is

$$\Pr_{u,v\sim E}[u \in C_i \wedge u \otimes v] \leq \mathcal{O}\left(\frac{\log^2 n}{|E|}\right) \leq \mathcal{O}\left(\frac{\log^2 n}{n^2}\right)$$

This shows that all cases which makes the barbell graph different from the complete graph decreases the probability of a collision, and so the expected number of collisions in B_n is less than that of K_n.

3.4 Summary

Table 1 summarizes the fitness of the graphs considered in this section.

Table 1: The fitness $\mathcal{F}$ of the graphs discussed in this section.

Graph	$\mathcal{F}(G)$
K_n	$n/(k^2 \log n)$
C_n	$(n-1)/(k \log n)$
S_n	$(n-1)/k$
B_n	$\geq n/(k^2 \log n)$

We can then order the graphs by their fitness:

$$\mathcal{F}(K_n) < \mathcal{F}(B_n) < \mathcal{F}(C_n) < \mathcal{F}(S_n)$$

3.4.1 What about other graphs?

As we have seen, sampling from B_n or S_n instead of K_n does not increase the expected number of collisions. It is also difficult to come up with a graph that is worse than K_n, so it is a reasonable guess that K_n does maximize the number of collisions in expectation.

Attempt 1

Perhaps we are able to show that for any graph G the addition of an edge does not decrease the number of expected collisions.

However, we are able to come up with a counterexample to this, as shown in Figure 5: prior to adding e_c we are guaranteed to have a collision since there is only two edges that remain, and these may collide. Afterwards we may get e_c instead, which does not collide, since the shared component is bigger than the single node.

There are a few things that feel off about this, though: 1. this only shows that given this exact state we decrease the number of expected collisions, *at this step*. It is not clear that the number is decreased overall. 2. this graph is very sparse, which somehow should make it a nice graph? 3. in the setting of the task queue we only have two tasks left, so the probability of getting the first element should be at least 1/2.

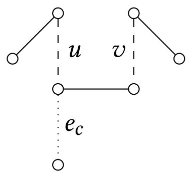

Figure 5

Attempt 2

Let $a \sim b$ be that a and b are in the same component.

Fix one edge a. Let $C(a)$ denote the number of edges b such that $a \otimes b$, and look at how $C(a)$ is affected by adding an edge e into the sampling graph. Let L and A be the largest and smallest component that a are connected to (ties), and let E be one the components connected to e.

1. $a \sim e$ and $e \sim L$: Since e cannot collide with a but the number of edges increases, the probability that a random edge collides with a decreases.
2. $a \sim e$ and $e \sim A$: There are two cases to consider:
   (a) $|A| \leq |E|$: e is now subject to collision with a. Since $\frac{C(a)}{n} \leq \frac{C(a)+1}{n+1}$ the probability increases, unless $C(a) = n$, where n is the number of remaining edges in the sample graph.
   (b) $|A| > |E|$: e cannot collide with a, so the probability decreases.
3. $a \nsim e$: a and e can not collide.

We try to imagine the worst possible graph — that is the graph that maximizes the expected number of collisions — in hope of ending up with a counter example to K_n, or K_n itself. This is weird however, since changing the sampling graph also changes the expected state of the forest.

Let $Z(v)$ be the size of the component in which v is, and let $L(v)$ be the number of nodes u such that $Z(v) \leq Z(u)$. That is, $L(u)$ is a bound on the number of edges containing v that may collide with some other edge.

4 Random Notes

Loose ends and other ideas.

4.1 Neighborhoods in the dependency graph

There is a few interesting things to be said about the dependency graph. If we relax the notion of a collision to just sharing exactly one endpoint, and assume that all tasks in the queue form a tree (that is, there are never any Union operations that try to join two nodes that are already in the same component) we can form a new graph F^* in which the nodes are the components of the connectivity forest, and the edges are the Union in Q. Now the dependency graph G is the line graph $L(F^*)$. Furthermore, performing a task t in the connectivity forest correspond to contracting the edge corresponding to t in F^*; this in turn correspond to adding the following edges to the dependency graph: $\{(u, v) | (u, t) \in E \wedge (t, v) \in E\}$: that is, all vertices (tasks) that have the performed task as a common neighbor become connected. In other words, the neighborhood of t becomes fully connected.

`Martin: [probably need a picture here!]`

This is interesting because line graphs have a curious property, namely that they cannot have S_3 as an induced subgraph, which means that all subset of three vertices of a neighborhood of any node cannot be an independent set (equivalently, $\alpha(N(t)) \leq 2$ for all t). This puts a requirement on the number of edges that the neighborhood of any element must contain, which in turn gives us a bound on the number of edges that we introduce by fully connecting the neighborhood when a task is performed.

What is this bound? First, for any three nodes to contain at least one edge, the neighborhood of any two nodes must contain all nodes, so $2\delta(G) \geq n-2 \implies \delta(G) \geq n/2 - 1$. Consider the induced subgraph of the neighbourhood of some node u. Since $m \geq 1/2 \sum^n \delta(G)$ we have

$$\frac{n(n-2)}{4} \leq m \leq \frac{n(n-1)}{2}$$

Here m is the number of edges in the neighbourhood, and n is the size of the neighbourhood, which is $d(u)$ in F^*. `Martin: [poorly written]`

A series of Union operations correspond to a series of edge contractions in F^*, so we have a sequence of F^*s:

$$F_0^* \longrightarrow F_1^* \longrightarrow F_2^* \longrightarrow \cdots \longrightarrow F_m^*$$

each of which have a corresponding line graph $L(F_i^*)$ in which the neighborhoods of each node becomes more and more connected.

The bottom line of this is that the degree of the vertices in the dependency graph may tell us a lot about the degrees of future dependency graphs after Union operations have been performed.

References

[1] Alistarh, D., Brown, T., Kopinsky, J., and Nadiradze, G. Relaxed schedulers can efficiently parallelize iterative algorithms. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing* (2018), ACM, pp. 377–386.

[2] Erdos, P., and Rényi, A. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci 5*, 1 (1960), 17–60.