Estudio empírico de la estructura `UnionFind`

En el presente trabajo se implementa una estructura `UnionFind` con diversas estrategias de unión y compresión de caminos, se evalúan y analizan sus respectivas performance desde un aspecto teórico y práctico.

Se analizan:

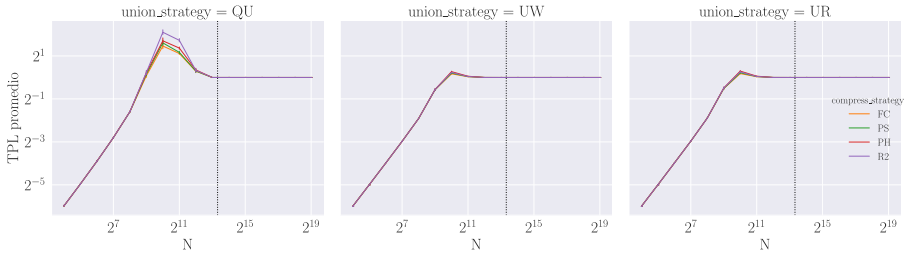
- el *total path length* (TPL) en función de N
- el *total pointer updates* (TPU) en función de N
- la cantidad de *roots' children* y sets disjuntos en función de N
- el tiempo de ejecución empírico en función de N

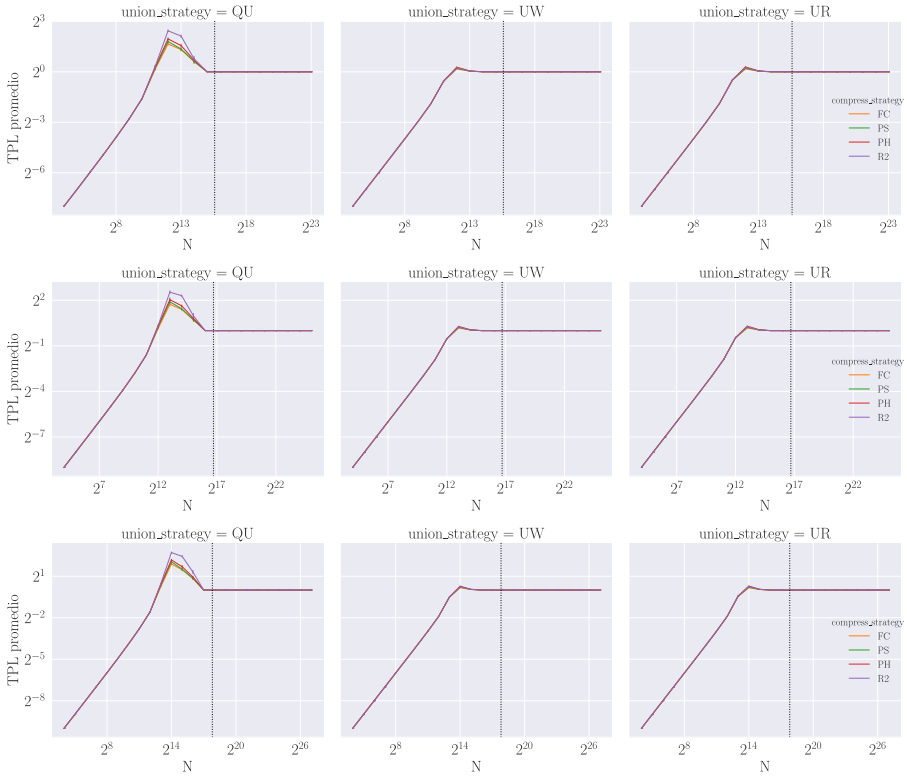
En adición a todas las estrategias de unión y compresión propuestas originalmente por el enunciado se agregó al análisis la estrategia de compresión de caminos *reversal k=2* (R2). Durante un `Find(i)`, esta estrategia hace que todos los nodos recorridos salvo la raíz y sus hijos directos apunten al nodo i y sea este quien apunte al nodo raíz. Es casi tan similar a *full compression* (FC) con la ventaja de requerir un solo recorrido.

En donde tenga sentido todas los gráficos tienen una línea vertical punteada con coordenada $N = n \log n$ ya que teóricamente para ese valor de N y superiores los n sets disjuntos iniciales formaran un único set y las siguientes llamadas a `Union(i,j)` solo afectaran a la estructura solo a través de una estrategia de compresión de caminos.

Análisis del *total path length* en función de N (salvo para la estrategia NC)

Las siguientes figuras muestran la evolución del *total path length* (TPL) en función de la cantidad de llamadas a `Union(i,j)` (N) para cada combinación de estrategia de unión y de compresión con excepción de NC (estrategia sin compresión). Cada fila corresponde a un número inicial de sets disjuntos (n): $n = 2^{10}$, $n = 2^{12}$, $n = 2^{13}$ y $n = 2^{14}$.

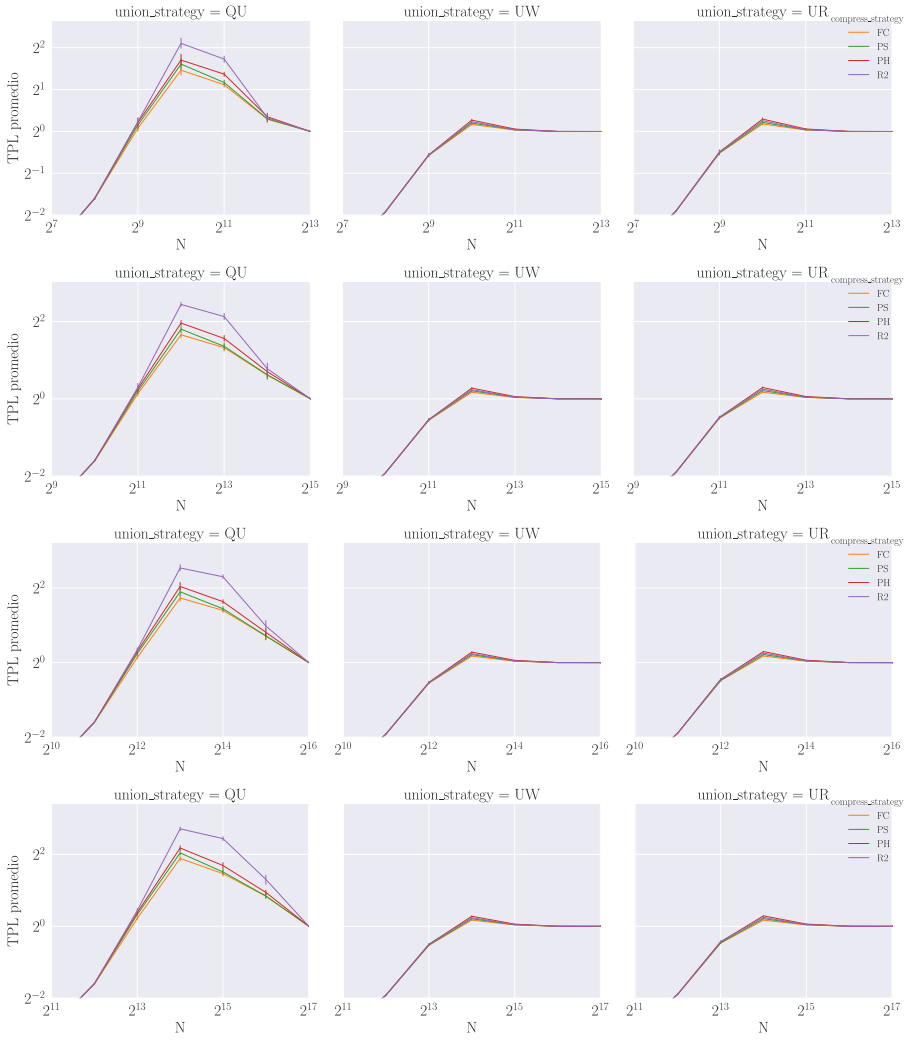




En todos los casos para $N \ll n$ el TPL crece linealmente con N. Inicialmente hay n sets con un total de TLP 0 (cada set tiene un único elemento). Con cada llamada a `Union(i,j)`, dos sets se unen para formar uno con TPL de 1. Esto se debe a que es más probable hacer una unión entre 2 sets de 1 elemento cada uno que entre sets más grandes.

Para cuando $N \gg n$ el TPL se vuelve constante igual a 1. Para $N \gg n$ habrá un solo set de n elementos y cada llamada a `Union(i,j)` no hara sino otra cosa comprimir la estructura (debido a las llamadas a `Find(.)`). Eventualmente el único set tendrá una altura de 1 y por ende un TPL promedio de 1.

El análisis es más interesante para $N \sim n$ donde hay una transición entre el comportamiento lineal y el constante del TPL. Las siguientes figuras son un acercamiento a dicha transición.



De los gráficos puede verse que para la estrategia *quick union* (QU), las estrategias de compresión se diferencian entre si con *full compression* (FC) siendo la mejor y *reversal k=2* (R2) siendo la peor.

Para el resto de las estrategias de unión, las estrategias de compresión no muestran diferencias significativas, al menos para los valores de n usados.

Futura investigación

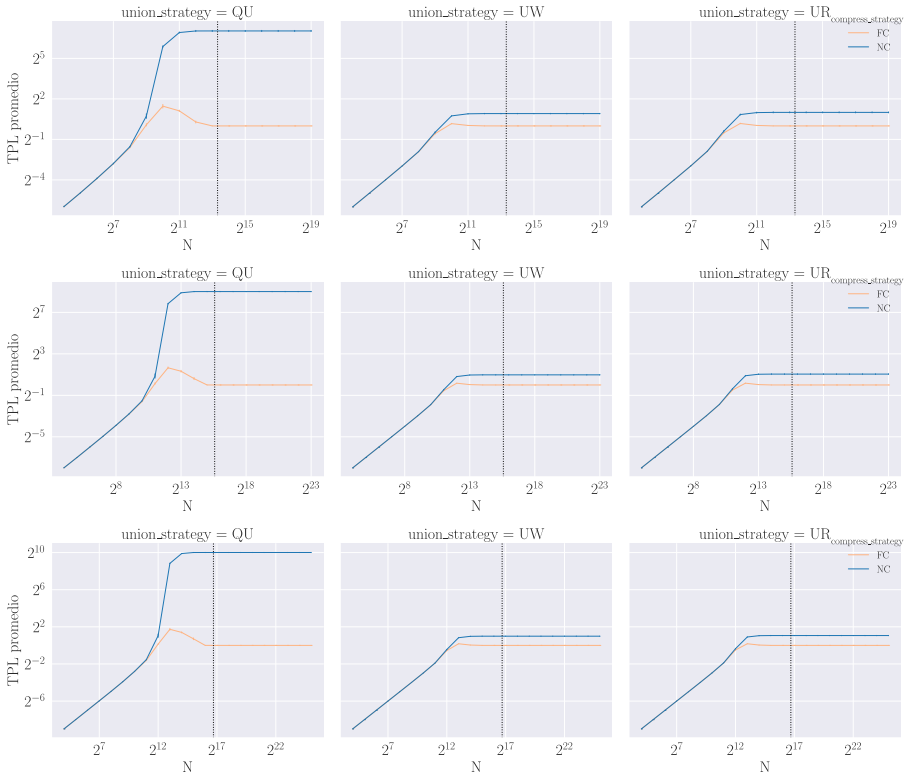
Para $N \sim n$ seria interesante usar más puntos de muestreo. De los gráficos puede tener una idea de la forma de las curvas de TPL pero lo correcto seria tener más muestras y no especular. También seria interesante ver para que valor de N se alcanza un máximo.

Análisis del *total path length* en función de N entre las estrategias NC y FC

La estrategia de compresión nula (NC) requiere una análisis especial puesto que como no comprime ningún camino es de esperarse que el TPL promedio sea monótonamente creciente.

Notar como los valores de TPL para las estrategias anteriores alcanzan un máximo entre 2^0 y 2^3 (1 y 8). En cambio para NC el máximo puede llegar a 2^{10} haciendo que las pequeñas diferencias entre las otras estrategias y NC se vean más pequeñas en comparación.

Aun así, junto con NC se dibujo el TLP para FC como punto de referencia. Cada fila corresponde a un número inicial de sets disjuntos (n): $n = 2^{10}$, $n = 2^{12}$ y $n = 2^{13}$.



Para $N \ll n$ el TPL crece linealmente con N, tal como sucede con el resto de las estrategias. Es para $N \gg n$ donde hay un cambio.

Con la estrategia QU, `Union(i,j)` une el set i con j sin tomar en consideración si el set resultante seria más alto de lo necesario – y por ende, más lento. Teóricamente el peor caso es $O(n)$ donde `UnionFind` consiste es un solo set cuya estructura es una lista.

Con pares (i, j) elegidos al azar este peor caso es improbable. Aun así el TLP es más de 2^5 veces más grande que para FC y el resto de las estrategias de compresión.

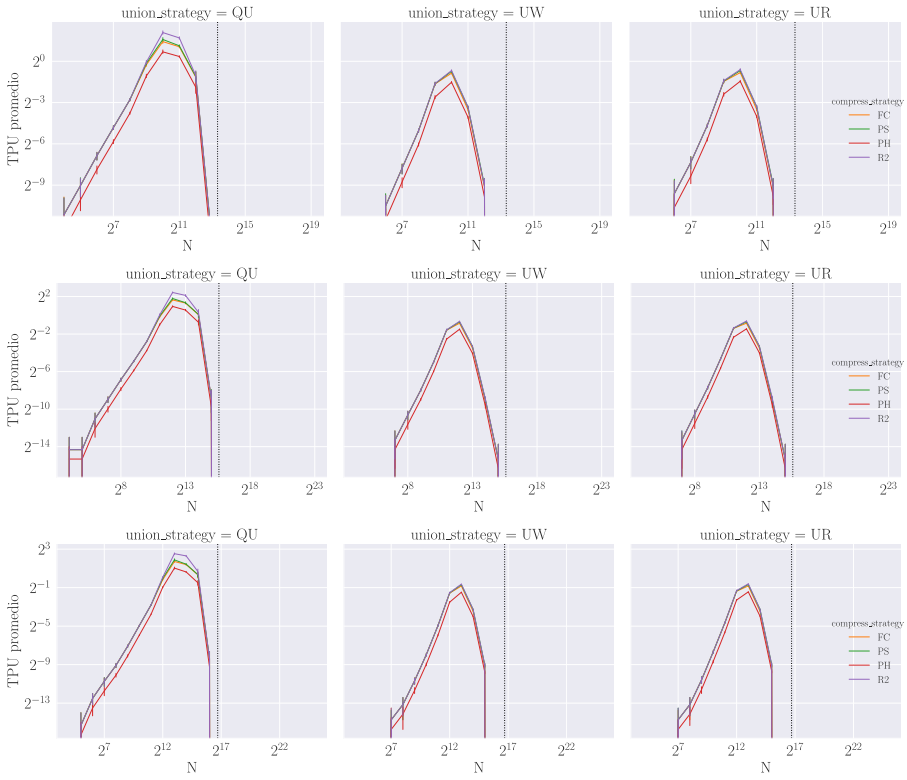
La situación cambia cuando la unión es *by weight* (UW) o *by rank* (UR). `Union(i,j)` evita crear estructuras altas lo que termina en TLP promedios mucho más bajos aun sin comprimir.

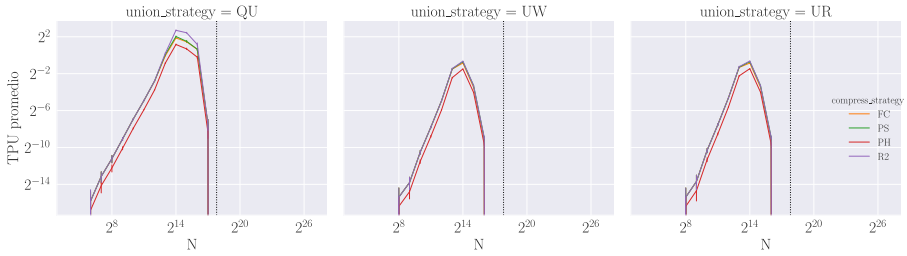
Futura investigación

Para las estrategias UW y UR tiene sentido comparar NC con todas las demás estrategias de compresión y para n incluso más altos y ver cuan grande se vuelve la diferencia entre los TPL.

Análisis del *total pointer updates* en función de N

Total pointer updates (TPU) es la cantidad de updates a los punteros que se harían si se hicieran una compresión de caminos. Es un estimador del costo de escritura en memoria de cada `Find(i)`. Las siguientes figuras muestran el TPU promedio en función de N; cada fila corresponde a un número inicial de sets disjuntos (n): $n = 2^{10}$, $n = 2^{12}$, $n = 2^{13}$ y $n = 2^{14}$.





Como puede verse el TPU de la compresión *path halving* (PH) es siempre la mitad que sus alternativas. Esta estrategia actualiza el puntero *parent* de solo la mitad de los nodos en el camino.

Al igual de lo que pasa con TPL, el TPU tiene un comportamiento lineal para $N \ll n$ con el detalle que las mediciones muestran una mayor dispersion de valores (en la gráfica se ven las *error bars* más alargadas).

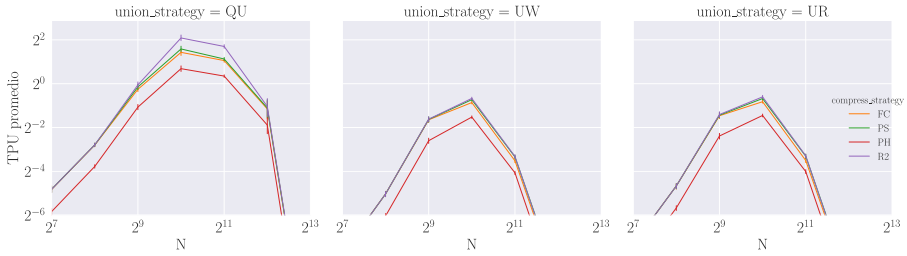
Para $N \sim n \log n$, el TPU promedio se desploma. Esto puede entenderse mejor viendo al TPU como una función que depende del TPL:

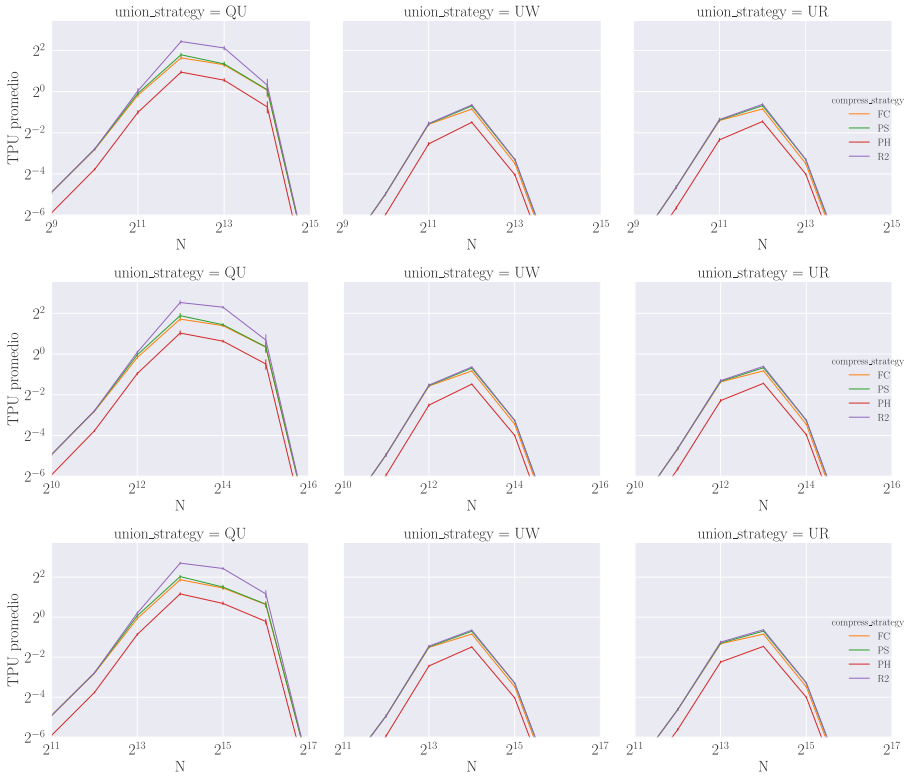
$$\mathrm{TPU}_{\mathrm{FC}} = \mathrm{TPL}_{\mathrm{FC}} - \sum_{\forall r \,\mathrm{root}} r\text{'s children count}$$

Para el resto de las estrategias de compresión están las siguientes equivalencias: $\mathrm{TPU}_{\mathrm{FC}} = \mathrm{TPU}_{\mathrm{PS}} = \mathrm{TPU}_{\mathrm{R2}}$, $\mathrm{TPU}_{\mathrm{FC}} = 2\,\mathrm{TPU}_{\mathrm{PH}}$ y $\mathrm{TPU}_{\mathrm{NC}} = 0$.

Como se vió anteriormente el TPL promedio llega a 1 para $N \sim n \log n$ lo que implica que los sets tienen altura 1 y por lo tanto muchos más nodos como hijos directo del nodo raíz lo que cancela el TPL y da un TPU promedio cercano a cero.

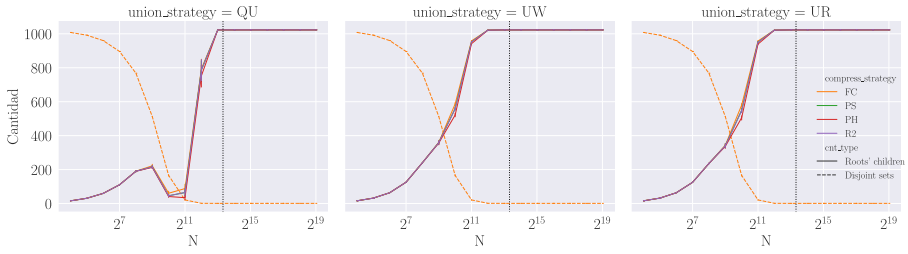
Las siguientes figuras muestran un acercamiento para $N < n \log n$:

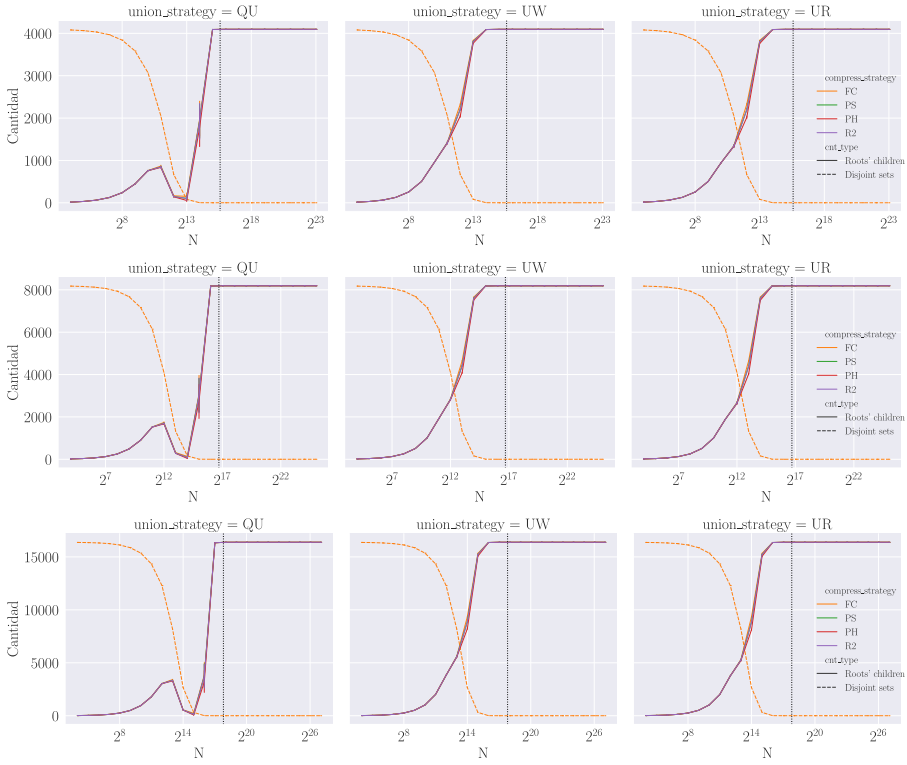




Análisis de la cantidad de *roots' children* y sets disjuntos en función de N

Las siguientes figuras muestran la cantidad de nodos que son hijos directo de un nodo raíz, *roots' children*, en líneas contiguas y la cantidad de nodos raíz o sets disjuntos que hay en función de N, en una línea de guiones curva. Cada fila corresponde a un número inicial de sets disjuntos (n): $n = 2^{10}$, $n = 2^{12}$, $n = 2^{13}$ y $n = 2^{14}$.

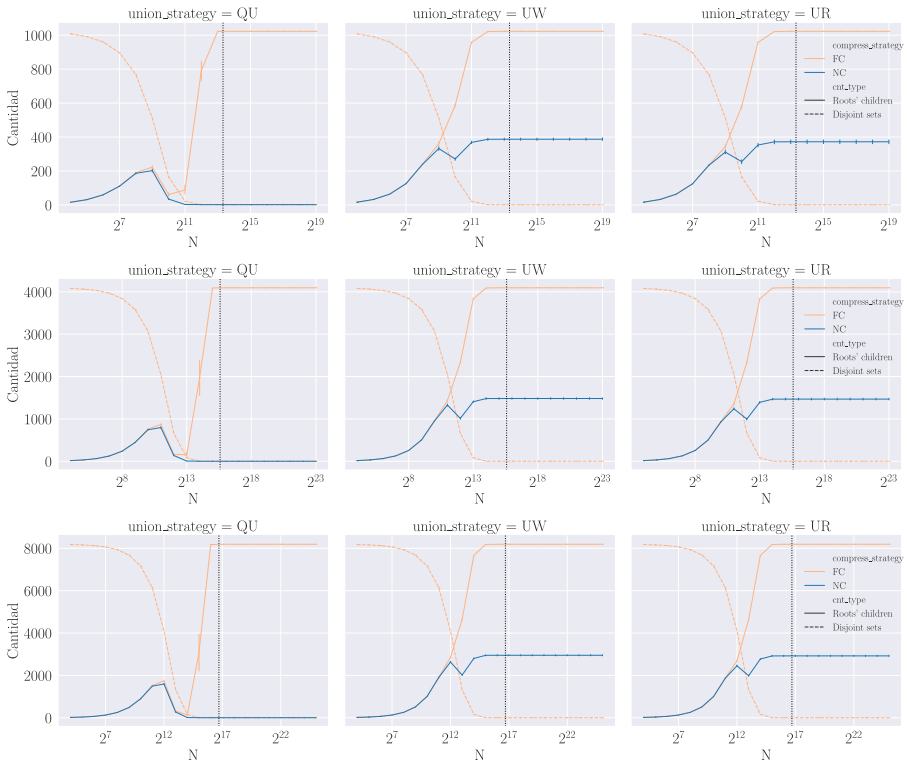




La línea de guiones curva es fácil de interpretar: comienza en n, pues al principio hay inicialmente n sets disjuntos, y decae a 1 para $N \sim n \log n$ pues para ese momento todos los nodos pertenecen a un mismo y único set. Es un comportamiento independiente de las estrategias usadas.

Las curvas con líneas sólidas son la cantidad de hijos directos: comienzan en 0, como es de esperarse, y crecen rápidamente ya que en cada llamada a `Union(i,j)` es muy probable que se traten de sets disjuntos i y j y la unión establezca un nuevo hijo directo a la vez que se reduce en uno la cantidad de set disjuntos. Por eso las curvas son un reflejo de la curva de guiones, tanto para N chicos como para $N \gg n$.

Para QU se muestra un ascenso lento, un quiebre, un descenso y luego una subida repentina. Este comportamiento se entiende mejor comparando una estrategia de compresión como FC con la compresión nula NC como lo muestran las siguientes figuras (para $n = 2^{10}$, $n = 2^{12}$ y $n = 2^{13}$):



Para UW y UR, la cantidad de hijos directos se estanca muy por debajo de n. Para N grandes, `Union(i,j)` une 2 sets que ya forman parte de un mismo set y debido a que no hay compresión, la estructura no cambia y la cantidad de hijos directos queda fija. En cambio, para otras estrategias de compresión como FC, la llamada a `Union(i,j)` tiene como efecto secundario 2 llamadas a `Find(.)` que permiten la compresión de los caminos y el agregado de múltiples nodos como nuevos hijo directos.

Para QU la cantidad de hijos directos se desploma a valores muy chicos. Esto se explica si se tienen sets con pocos hijos directos que se unen entre si: el set resultante tendrá pocos hijos directos pero con una mayor altura.

Es interesante ver para QU las curvas de NC y FC van a la par: para N chicos ambas coinciden indicando que quien domina la función es la estrategia de unión QU. Pero luego hay un quiebre y las curvas se separan: mientras la curva de NC se va a 0 la de FC sube rápidamente lo que indica que es la estrategia de compresión la que tracciona y domina la cantidad de hijos directos.

Futura investigación

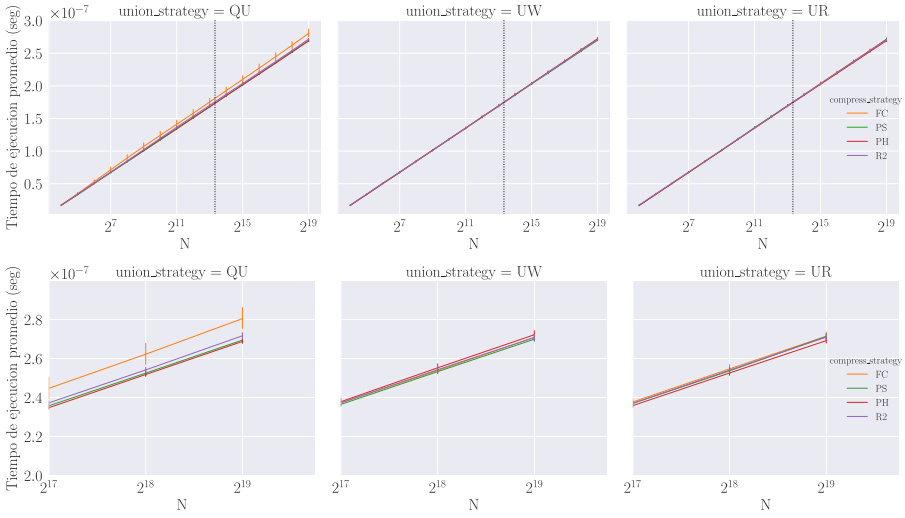
Con solo 20 experimentos (20 *seeds*) no queda tan claro que tan típico es el comportamiento visto en QU. Ademas, se requieren más puntos de medición

para ver con exactitud la curva durante el quiebre, el descenso y el posterior ascenso.

Análisis del tiempo de ejecución empírico en función de N

El enunciado propone modelar el costo de una estrategia como $2\,\mathrm{TPL} + \epsilon\,\mathrm{TPU}$ pero se prefirió tener una medición empírica para $n = 2^{10}$ y para todas las estrategias salvo NC. La razón es que TLP y TPU son abstracciones que no permiten ver la interacción con el hardware, en especial con la memoria y la cache.

La primer figura es el gráfico completo, la segunda es un acercamiento para $N \gg n$.



En QU puede verse como FC tiene un costo de performance superior al del resto de las estrategias debido a que FC recorre 2 veces el camino al nodo raíz. Para UW y UR este costo es despreciable por que los caminos en si son cortos ya que tanto UW como UR fomentan la construcción de arboles de baja altura.

Futura investigación

Salvo para QU, no es posible ver que estrategia tiene una mejora visible con respecto a la otra. Hay varios puntos a mejorar:

- analizar n más grandes: ver si hay o no una tendencia
- incrementar el número de rondas: para mayor precision
- reducir el impacto / *overhead* de la instrumentación

El ultimó punto es importante: se midió el tiempo de ejecución de cada llamada a `Union(i,j)` lo que también tiene un mayor overhead. Se debería poder medir

m llamadas seguidas a `Union(i,j)` con un overhead por la medición amortizado de $1/m$.

Escenario de experimentación

Definiremos como configuración del experimento los siguientes parámetros que han de quedar fijos para toda la ejecución del mismo:

- n es la cantidad inicial de sets en la estructura `UnionFind`
- *seed* es la semilla que inicializa el generador de números aleatorios y por ende define deterministicamente la lista de pares (ver más adelante).
- U y C las estrategias de unión y compresión de camino respectivamente.

Para una configuración dada se construye una estructura `UnionFind` de n elementos y se ejecuta `Union(i,j)` para todo par no ordenado (i, j) con $i \neq j$ elegidos al azar y sin repetir de los $\binom{n}{2}$ posibles pares. Con un *seed* fijo, los pares son generados deterministicamente con el algoritmo `shuffle` y el generador `MT19937` de `g++ 10.2.1 C++17`. Los *seeds* usados son los primeros dígitos del número PI.

En cada llamada a `Union(i,j)` se midió el tiempo de ejecución tomando recaudo que el compilador no reordenara las instrucciones y pueda distorsionar la medición. Para tener menos ruido en la medición debido a factores externos como otros procesos compitiendo por la CPU, el OS manejando una interrupción, etc, se configuro un ambiente lo más tranquilo posible:

- se deshabilitó hyperthreading
- se desactivó el CPU throttling
- se desactivó la interfaz gráfica y la red (wifi)
- se removió del scheduler el CPU número 3 para su uso exclusivo en los tests.

Aun así no todo el ruido puede ser removido. Para minimizarlo se ejecuto el mismo test R veces. Dado que en cada *ronda* la configuración del experimento es la misma (mismo n, misma lista de pares, mismas estrategias), la performance de `Union(i,j)` se espera determinística salvo por un ruido aditivo positivo.

Cada Δ_i pares procesados, se tomó el tiempo de ejecución acumulado hasta el momento y se lo comparó con el tiempo de la ronda anterior para el mismo Δ_i quedándose con el mínimo de ambos. Así tras R rondas se obtuvo el mínimo de ejecución para cada momento Δ_i, reduciendo así el ruido.

Ademas, cada Δ_i pares procesados se midió el *total path length* (TPL), *total pointer updates* (TPU), la cantidad de sets disjuntos y la cantidad de nodos que son hijos directo de un nodo raíz. Para una misma configuración del experimento, todos estos valores son determinísticos independiente de en que ronda se midan. Por simplicidad se midieron en la última.

En el enunciado se propone usar una frecuencia de medición equiespaciada, múltiplos de Δ_0 (Δ_0, $2\Delta_0$, $3\Delta_0$, $4\Delta_0$). Sin embargo se optó por una frecuencia

que se duplica en cada iteración con un Δ_0 inicial de 16 ($\Delta_0 = 16$, $2\Delta_0 = 32$, $4\Delta_0 = 64$, $8\Delta_0 = 128$).

El experimento se repitió para S *seeds distintos* de los que se obtuvieron S resultados para cada medición Δ_i. Es a partir de estos S valores que se calculan los valores medios y variancia de TPL, TPU, tiempo de ejecución y demás.

A su vez, todo lo mencionado se ejecutó para los distintas combinaciones de n, U y C. Sin embargo, debido a que la cantidad de pares (i, j) a procesar por experimento crece como $\binom{n}{2}$ algunas configuraciones no fueron ejecutadas.

- Todos los valores salvo el tiempo de ejecución fueron medidos con 1 sola ronda:
  - para $n = 2^{10}$, $n = 2^{12}$ y $n = 2^{13}$, todas las combinaciones de U y C.
  - para $n = 2^{14}$, todas las combinaciones de U y C salvo NC.
- Los tiempos de ejecución fueron medidos con 35 rondas solo para $n = 2^{10}$ y todas las combinaciones de U y C salvo NC.

La decisión de descartar NC con $n = 14$ se debió a que en dichos escenarios los sets no son comprimidos y los tiempos eran más largos que los que teníamos asignado. A su vez, la performance teórica de dicha estrategia es considerablemente peor que la del resto y creemos que en el resto de los escenarios, con n chicos, esto ya sera visible. Lo mismo sucedió con la medición del tiempo de ejecución donde no hay ningún escenario con NC. Creemos que con el análisis de TPL sera suficiente para llegar a conclusiones razonables.

El código fue compilado con `g++ (Debian 10.2.1-6) 10.2.1 20210110` y ejecutado en un hardware físico AMD Ryzen 5000 con 8 GB de RAM con L1i/L1d de 32kb, L2 de 512kb y un único L3 de 16mb compartido por los cores.

Documentación consultada

- Shuffle: link:(random_shuffle - C++17)
- MT19937: link:(mersenne_twister_engine - C++17)
- `DoNotOptimize`: link:(StackOverflow post), link:(source code - Google:benchmark)
- `taskset`: link:(manpage taskset)
- Quiescent environment: link:(blog post)
- UnionFind: link:(slides - Princeton)

Apéndice - Código fuente

El código fuente se puede encontrar en Github:

link:(https://github.com/eldipa/disjoint_sets_performance_eci_advanced_data_structures)

La estructura del repositorio es la siguiente:

- `data_collected`: tiene las mediciones recolectadas durante la experimentación y las que son reflejadas por las figuras del presente trabajo.

- `plots`: las figuras y los scripts necesarios para recrearlas
- `submission`: este informe
- `disjoint_set.h`: la implementación de `UnionFind` incluyendo todas las estrategias de unión y compresión.
- `demo.cpp`: una mini aplicación para probar la estructura y renderizar el árbol resultando con `dot`
- `perf.cpp`: la implementación del setup de prueba y ejecución de experimentos.
- `Makefile`: instrucciones de compilación