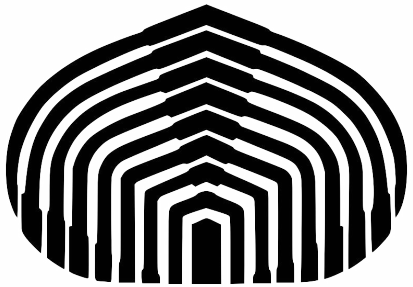


Extensión de *PostgreSQL* con Mecanismos de Optimización de Consultas Basadas en Preferencias

Fabiola Di Bartolo Francelice Sardá

Universidad Simón Bolívar

Objetivos de nuestro trabajo...

Objetivo general

Realizar la extensión de *PostgreSQL* con mecanismos de optimización de consultas basadas en preferencia.

Objetivos específicos

- Estudiar, proponer e implementar un modelo de costo para el operador *Skyline*.
- Extender y proponer algoritmos que permitan realizar de manera eficiente el proceso de optimización de las consultas *Skyline*.

Qué es una consulta *Skyline*

El *Skyline*

- Solución para las preferencias.
- Basado en un conjunto de directivas que expresan los deseos del usuario: *MAX*, *MIN* y *DIFF*.

Consulta en lenguaje natural

Deseo saber cuáles son los hoteles que se encuentran más cerca de la playa y a la vez tienen el mejor precio, agrupados por localidad.

Qué es una consulta *Skyline*

El *Skyline*

- Solución para las preferencias.
- Basado en un conjunto de directivas que expresan los deseos del usuario: *MAX*, *MIN* y *DIFF*.

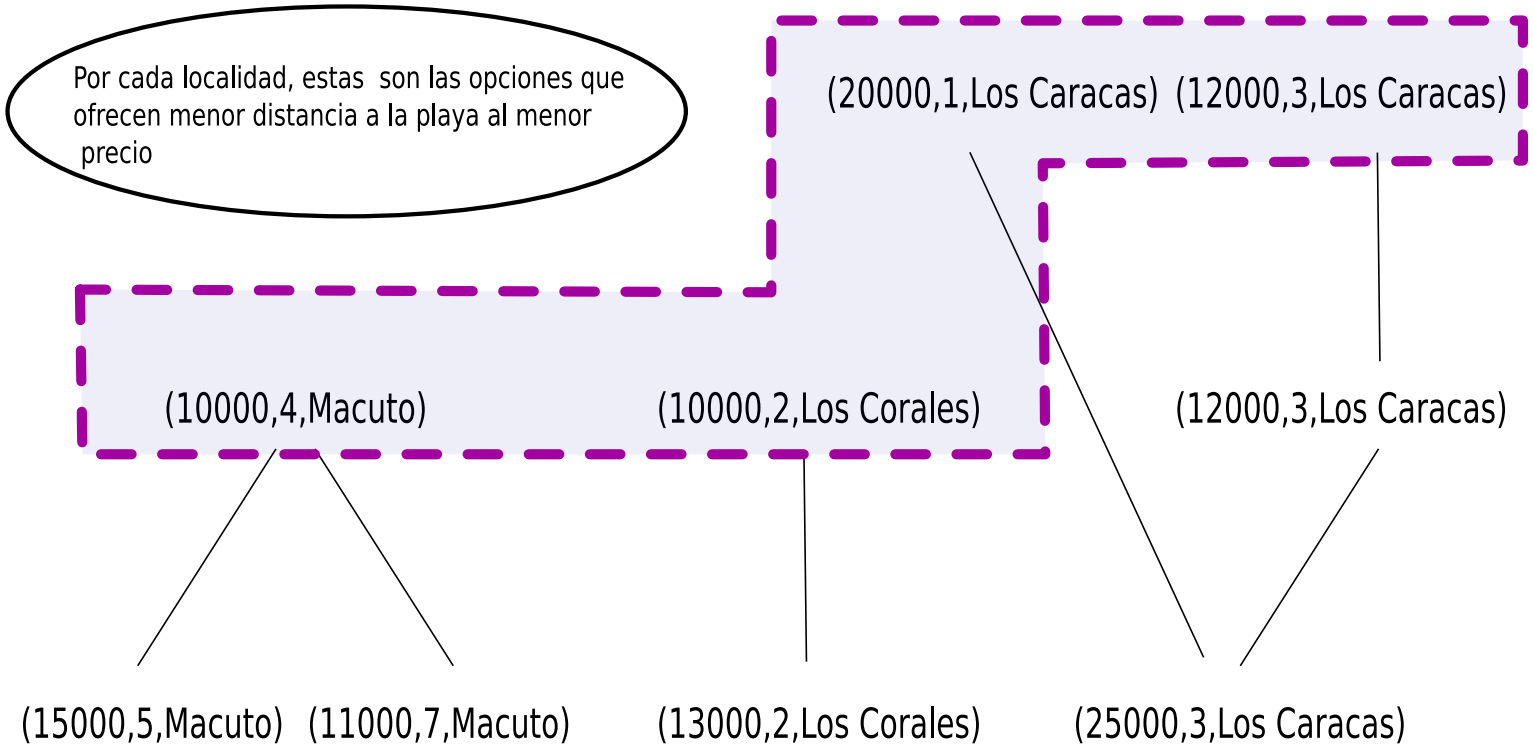
Consulta en lenguaje SQL

```
SELECT *
FROM HOTELS
SKYLINE OF dist MIN,
          price MIN,
          loc DIFF
```

Respuesta de una consulta *Skyline*

Precio(Bs)	Dist(Km)	Localidad
11000	7	Macuto
15000	5	Macuto
10000	4	Macuto
13000	2	Los Corales
10000	2	Los Corales
20000	1	Los Caracas
12000	3	Los Caracas
12000	2	Los Caracas
25000	3	Los Caracas

Respuesta de una consulta *Skyline*



Por qué optimizar las consultas *Skyline*

Por qué optimizar una consulta

- Para cada consulta pueden existir muchos planes o árboles de evaluación.
- Cada plan puede tener un costo de ejecución distinto.
- Se desea hallar el árbol de evaluación o plan que se estime sea menos costoso de ejecutar.

Por qué optimizar las consultas *Skyline*

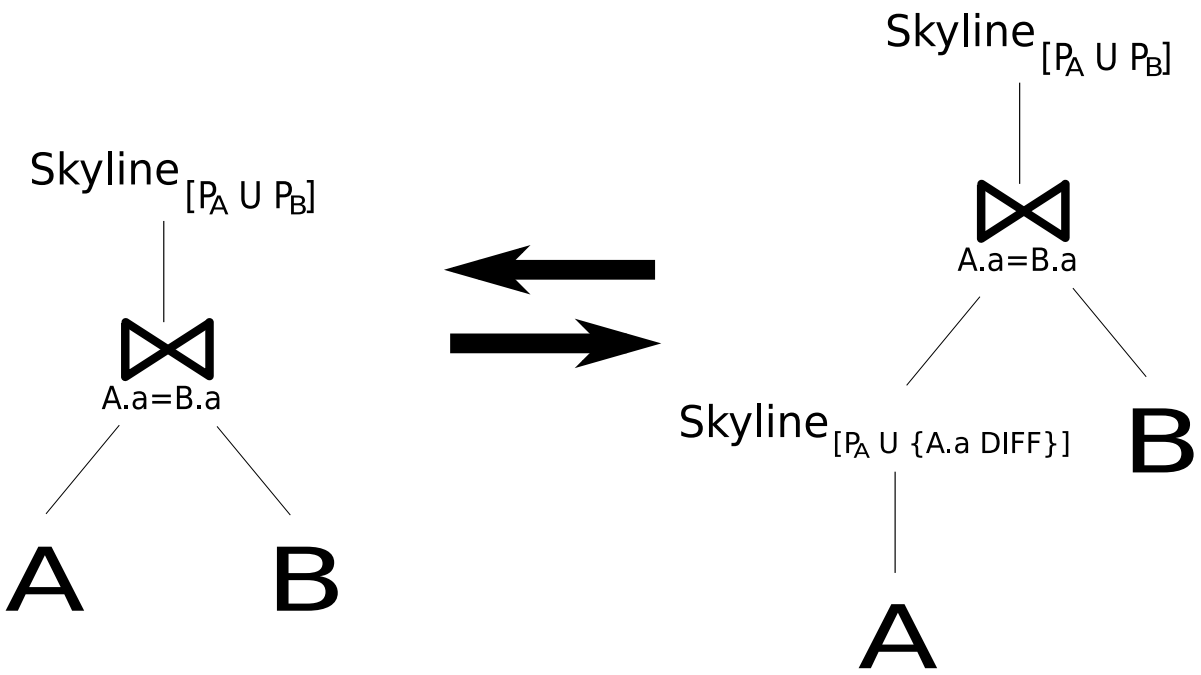
Por qué optimizar las consultas *Skyline*

- La implementación del *Skyline* como un operador adicional resulta beneficiosa.
- En algunos casos es beneficioso adelantar la ejecución de un operador *Skyline*.
- Los algoritmos de ejecución propuestos para el *Skyline* consumen altos recursos de procesamiento.

Equivalencia entre planes de una consulta *Skyline*

Regla general

$$\text{Skyline}_{[P_r \cup P_s]}(R \bowtie_c S) \equiv \text{Skyline}_{[P_r \cup P_s]}(\text{Skyline}_{[P_r \cup \{R.x\ diff\}]}(R) \bowtie_c S)$$



Estimación del costo del *Skyline*

Cardinalidad de la respuesta del *Skyline*

$$\hat{S}_{n,d} = \frac{1}{n}\hat{S}_{n,d-1} + \hat{S}_{n-1,d}$$

Costo de ejecutar un *Skyline*

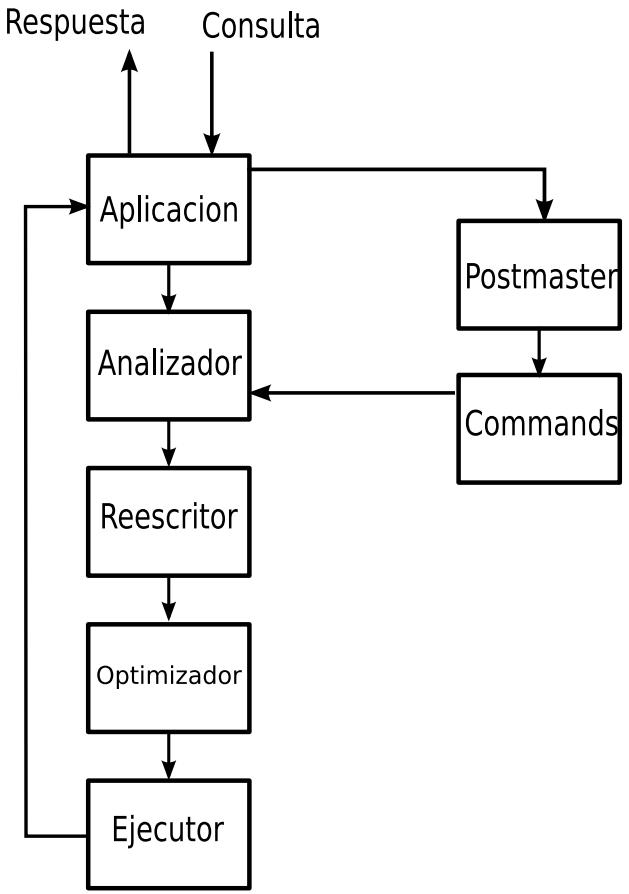
$$C_{BNL}(n,d) \approx \sum_{j=2}^{n} \frac{\hat{S}_{j-1,d}}{j-1}\hat{S}_{j-1,d+1}$$

$$C_{SFS}(n,d) \approx \sum_{j=2}^{n} \frac{\hat{S}_{j-1,d-1}}{j-1}\hat{S}_{j-1,d}$$

Procesamiento de una consulta en *PostgreSQL*

Flujo de una consulta

- Establecimiento de la conexión
- Análisis
- Reescritura
- Optimización
- Ejecución



Optimización en *PostgreSQL*

PostgreSQL realiza la optimización en dos fases:

Primera Fase: Creación del camino de evaluación

- Construcción de planes equivalentes para la evaluación del *Join* por medio de algoritmos de búsqueda.
- El producto es un árbol que indica el orden y operadores físicos con los que se evaluará el *Join* de las relaciones de la consulta.
- Se trabaja con operadores *down-level*.

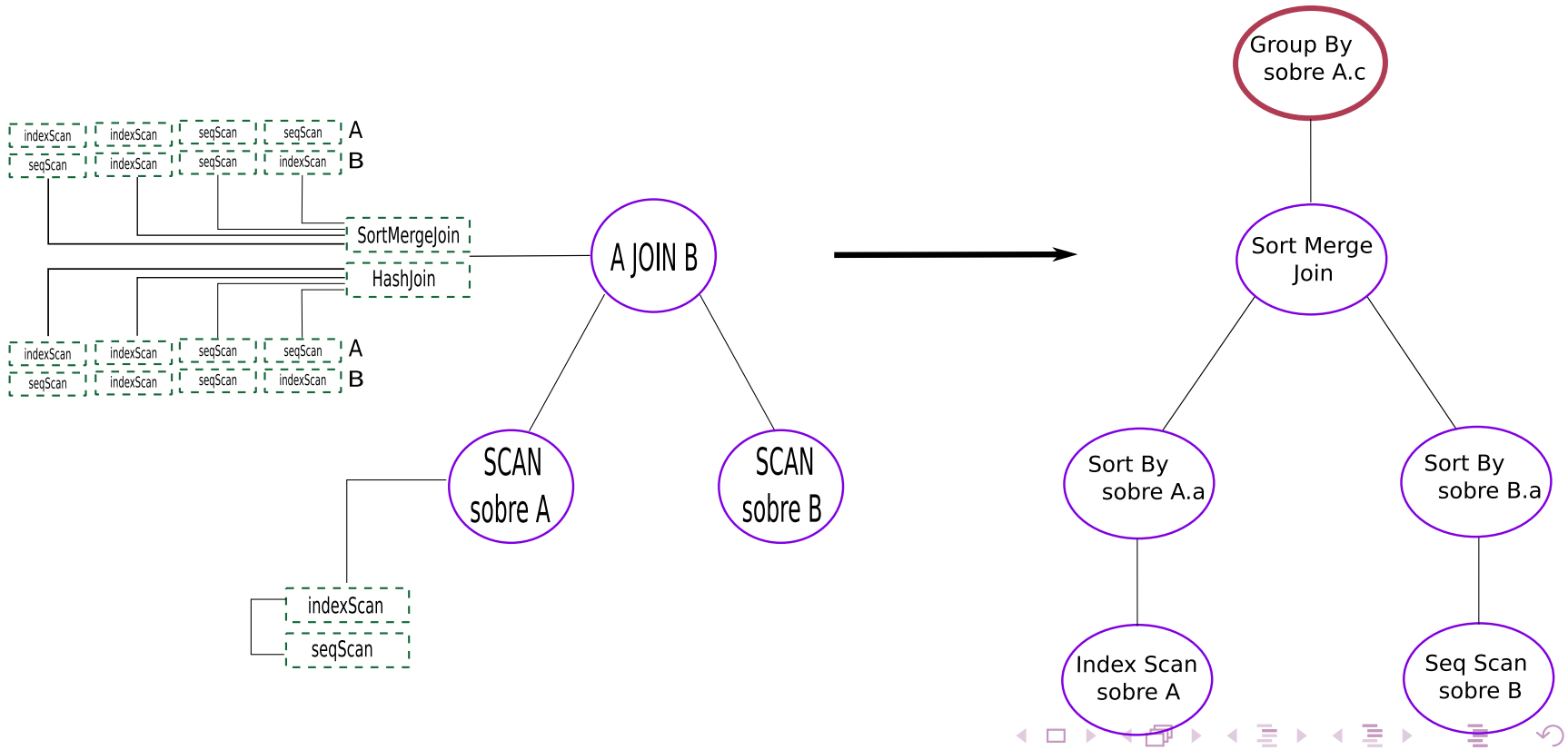
Segunda Fase: Creación del plan de evaluación

- Se realiza la transformación apropiada del camino de evaluación escogido.
- Se colocan los operadores agregados o *top-level*.

Construcción de un plan

Consulta

```
SELECT * FROM A,B WHERE A.a=B.a GROUP BY A.c
```



Algoritmo *DYQO*

Para cada nivel **i** necesario:

1. Para cada plan **p** resultante del nivel **i-1**:
   - 1.1 Para cada plan base **b**, si es factible **Join(p,b)**:
     - Caso1 Se realiza **Join(p,b)**.
   - 1.2 Si no se pudo realizar ningún **Join**, realizar Producto Cartesiano.
   - 1.3 Se eliminan los planes equivalentes.
   - 1.4 Se agregan los planes a la lista del nivel **i**.
2. Se escogen los mejores caminos para los planes del nivel **i**.

Algoritmo *GEQO*

1. Inicializar la población.
2. Mientras no se cumpla el número máximo de generaciones:
   - 2.1 Seleccionar dos individuos de la población.
   - 2.2 Reproducirlos.
   - 2.3 Si el tipo de reproducción es *cycle crossover*, entonces realizar mutación del hijo.
   - 2.4 Calcular el *fitness* del hijo.
   - 2.5 Añadir a la población.
3. Retornar el mejor individuo de la población.

Diseño de la solución

Optimización de consultas Skyline en PostgreSQL

- Esta extensión de *PostgreSQL* fue realizada a partir de la desarrollada por Brando y González denominada *PeaQock*.
- En la versión inicial de *PeaQock* se implementó el *parser* y la evaluación de consultas *Skyline*.
- El desarrollo de la nueva versión de *PeaQock* efectuado en este proyecto se enfocó en la optimización de consultas *Skyline*.

Diseño de la solución

Optimización de consultas Skyline en PostgreSQL

Se basó en la consideración del *Skyline* en la primera fase de optimización. Para esta fase se realizó:

- La inclusión del *Skyline* como nodo *down-level*.
- El modelo de costo para el *Skyline*
- Inclusión de los algoritmos *dPeaQock* y *ePeaQock* para hallar el plan de evalución.

Inclusión del *Skyline* como nodo *down level*

Primera versión de *PeaQock* (Brando y González)

El optimizador escoge el plan para evaluar la consulta sin considerar la cláusula `SKYLINE OF` y luego le agrega como nodo *top level* el *Skyline*.

Dibujo

Inclusión del *Skyline* como nodo *down level*

Versión actual de *PeaQock*

El optimizador construye los posibles planes considerando que el *Skyline* puede estar situado en cualquier lugar y escoge el plan a ser evaluado.

Dibujo

Inclusión del *Skyline* como nodo *down level*

Algunos atributos en el nodo *down level* para el *Skyline*

- Cláusula del *Skyline*.
- Conjunto de tablas involucradas en esa cláusula.
- Número estimado de tuplas a retornar.
- Caminos de evaluación: contiene los caminos de acceso *BNL* y *SFS* con sus costos respectivos y cada uno apunta a un camino de acceso del nodo anterior.

-dibujo mostrando todos los caminos de accesos para scan, skyline y join -mostrar el plan trasnformado para ser evaluado (sort.. etc)

Modelo de costo para el Skyline

El optimizador realiza estimaciones de costo para escoger un plan entre las demás alternativas. El modelo de costo que fue implementado para el *Skyline* incluye:

1. Número estimado de tuplas a retornar.
2. Número estimado de comparaciones para *BNL* y *SFS*.
3. Costo estimado para el *BNL* y *SFS*.

Modelo de costo para el Skyline

Las fórmulas originales del modelo de costo del Skyline se modificaron para:

1. Ofrecer un mejor desempeño en cuanto a tiempo empleado para realizar la estimación.
2. Considerar el criterio *diff* como otra dimensión más para proporcionar una estimación más completa.

Modelo de costo para el Skyline

grafica de estimaciones de la cardinalidad del skyline

Modelo de costo para el Skyline

Modelo de costo implementado	
Cardinalidad del Skyline	$\hat{S}_{n,d} \approx a(d) * log(n)^{b(d)}$
Comparaciones	$Comp(n,d) \approx \sum_{j=i}^{n} \frac{\hat{S}_{j-1,d}}{j-1} \hat{S}_{j-1,d+1} + Comp'(i,d)$
Costo BNL	$C_{BNL}(n,d) \approx Comp(n,d) + n * k$
Costo SFS	$C_{SFS}(n,d) \approx C_{Sort}(n,C) + Comp(n,d-1) + n * k$

Donde:

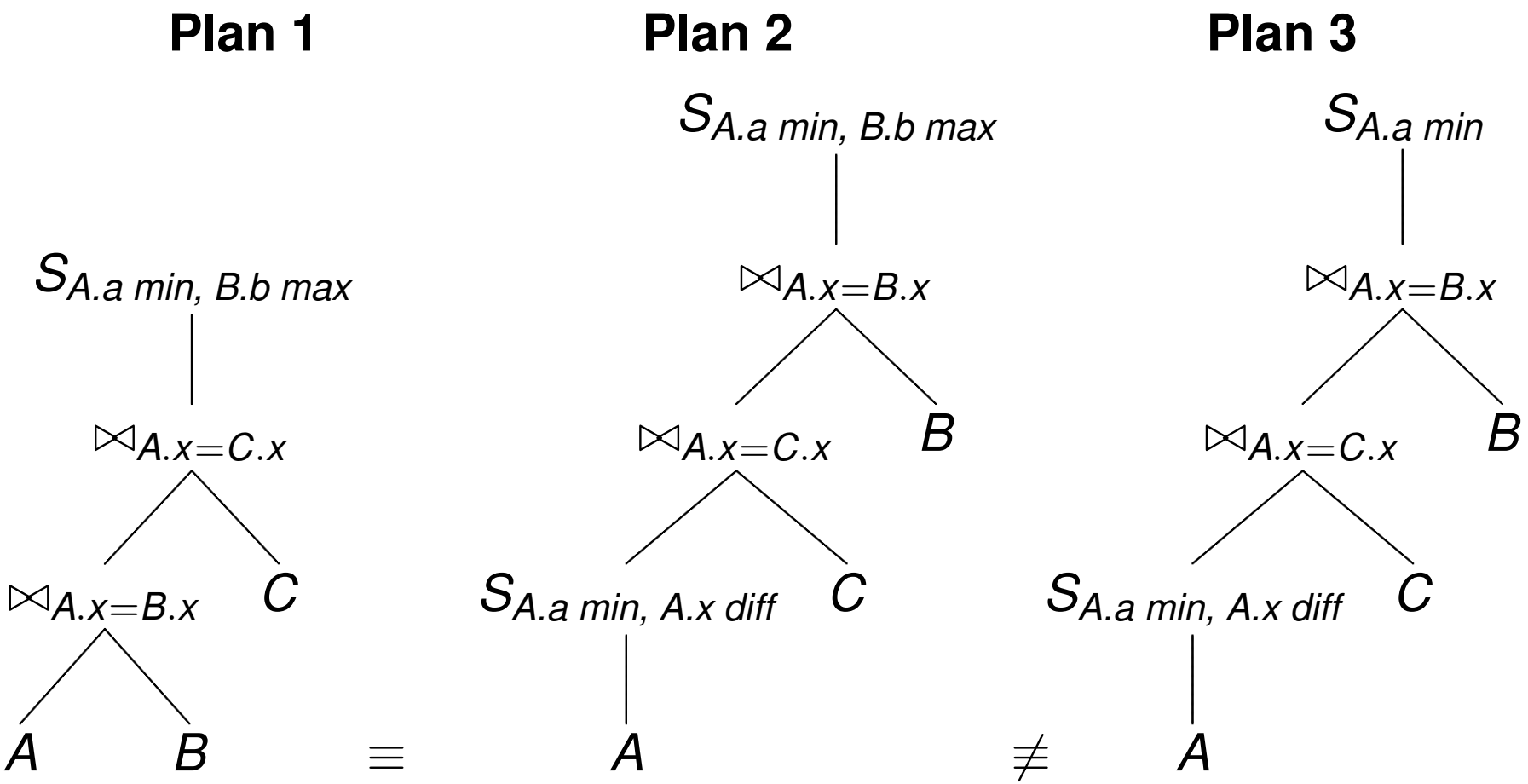
- n es el número estimado de tuplas de la relación entrante.
- d es el número de dimensiones *min*, *max* y *diff* del *Skyline*.
- i es el número de tuplas más cercano por debajo a n al cual le corresponde una entrada en la matriz de comparaciones $Comp'$.
- k es el costo de procesar una tupla.
- C es el conjunto formado por los criterios del *Skyline*.

Algoritmos para la optimización de consultas Skyline

Características generales

- El espacio de búsqueda es mayor. El *Skyline* se puede colocar si es posible en cualquier lugar del plan.
- El nodo raíz de un plan para una consulta *Skyline* debe ser un *Skyline*.
- Es posible elegir si se desea incluir el *Skyline* en la primera fase de optimización o no y el algoritmo a utilizar.
- La equivalencia entre planes fue modificada.

Equivalencia entre planes



dPeaQock: variante del DYQO

Para cada nivel **i** necesario:

1. Para cada plan **p** resultante del nivel **i-1**:
   - 1.1 Para cada plan base **b**, si es factible **Join(p,b)**:
     - Caso1 Se realiza **Join(p,b)**.
     - Caso2 Si **p** contiene una o varias relaciones del *Skyline* crea **s=Skyline(p)**, escoge los mejores caminos y luego realiza **Join(s,b)**.
     - Caso3 Si **b** pertenece al *Skyline* crea **s=Skyline(b)**, escoge los mejores caminos y luego realiza **Join(p,s)**.
     - Caso4 Si **p** contiene una o varias relaciones del *Skyline* y **b** también pertenece crea **s1=Skyline(b)** y **s2=Skyline(b)**, escoge los mejores caminos y luego realiza **Join(s1,s2)**.
   - 1.2 Se eliminan los planes equivalentes.
   - 1.3 Se agregan los planes a la lista del nivel **i**.
2. Se escogen los mejores caminos para los planes del nivel **i**.

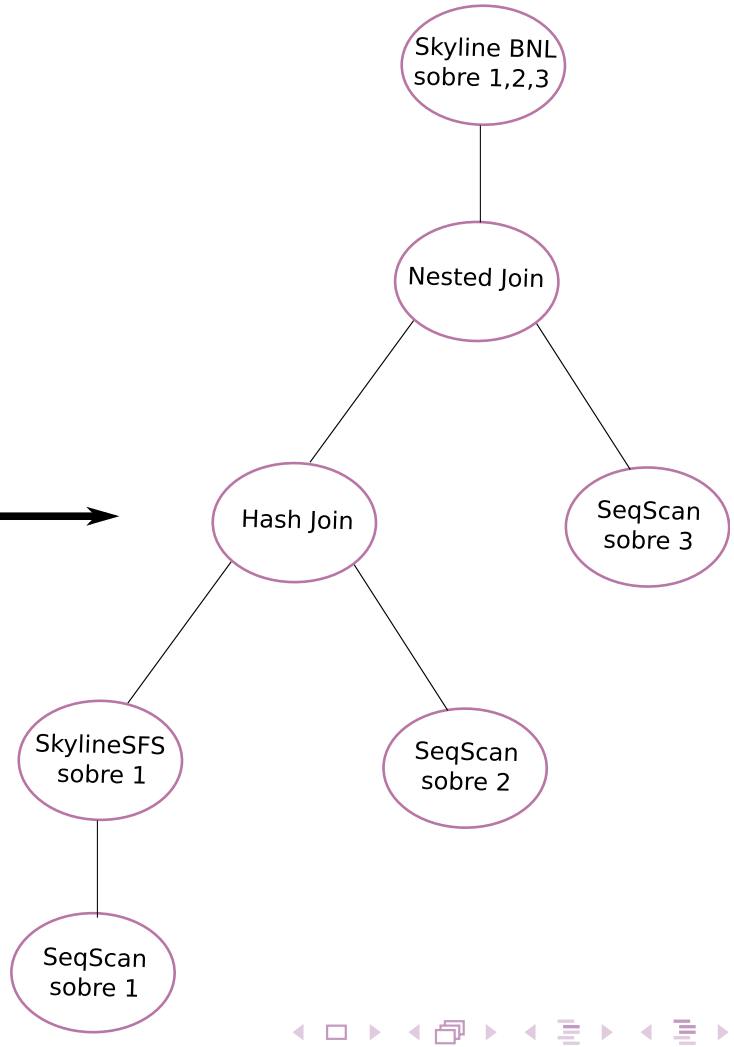
ePeaQock

Características de *ePeaQock*

- Generación de planes válidos.
- Ausencia de operador de reproducción.
- Tasa de mutación alta.
- Múltiples operadores de mutación: `mutarJoin` y `mutarSkyline`.
- Condición de parada compuesta.
- Selección elitista.

Representación del individuo

$1\text{-}S_1\text{-}2\text{-}3\text{-}S_{1,2,3}$ ⟶



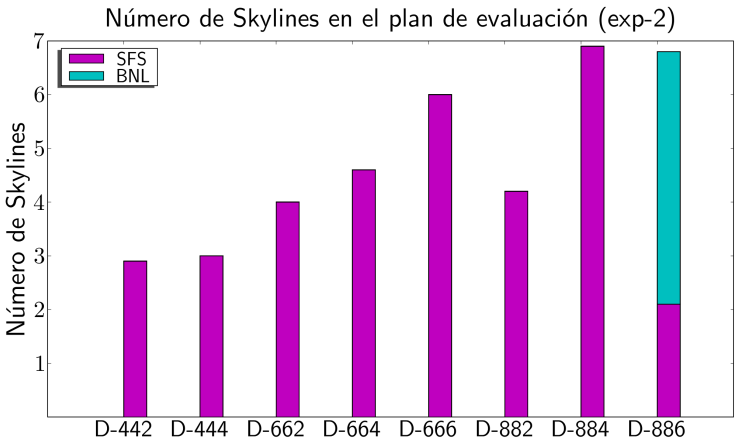
Algoritmo *ePeaQock*

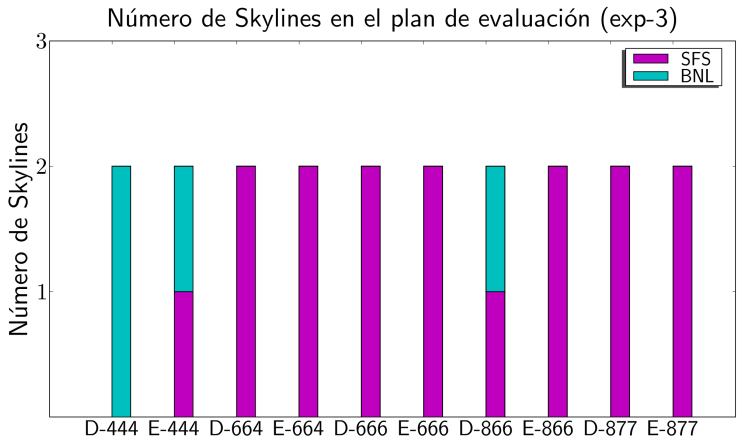
1. Inicializar la población.
2. Mientras no se cumpla el número máximo de generaciones y no se estabilice el mejor individuo:
   - 2.1 Mientras no se mute el porcentaje de individuos indicado.
     - 2.1.1 Escoger individuo manteniendo el elitismo.
     - 2.1.2 Escoger el método de mutación a aplicar: `mutarJoin` o `mutarSkyline`.
     - 2.1.3 Mutar al individuo escogido.
     - 2.1.4 Reemplazar al individuo mutado.
     - 2.1.5 Calcular su nuevo *fitness*.
3. Retornar el mejor individuo de la población.

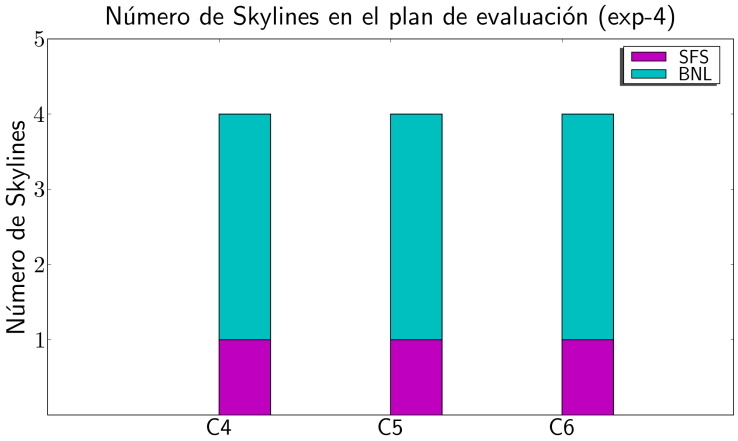
Experimentos y resultados

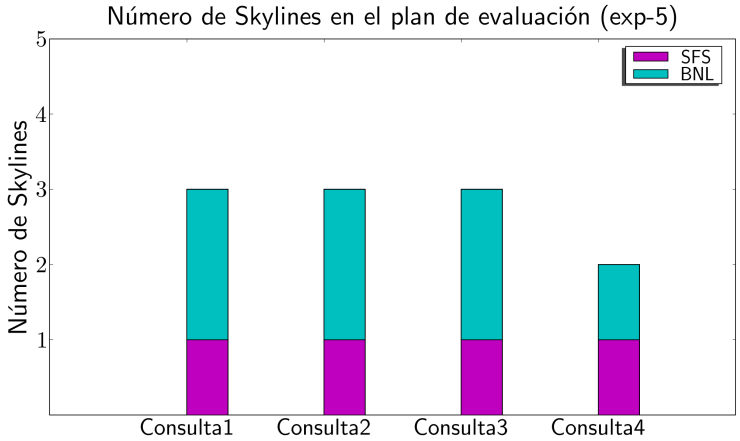
- 5 experimentos realizados: 4 con datos uniformes y uno con datos reales.
- Consultas aleatorias para datos uniformes.
- Consultas a una base de datos de *Baseball* para datos reales.
- Total de corridas: 400
- Variables consideradas:
  - Selectividad: 0.001, 0.0006, 0.0002
  - Cardinalidad: 1.000 - 100.000
  - Tablas: 3 - 8
  - Dimensiones del *Skyline*: 3 - 8
  - Tablas involucradas en el *Skyline*: 2 - 7

Características de los planes escogidos





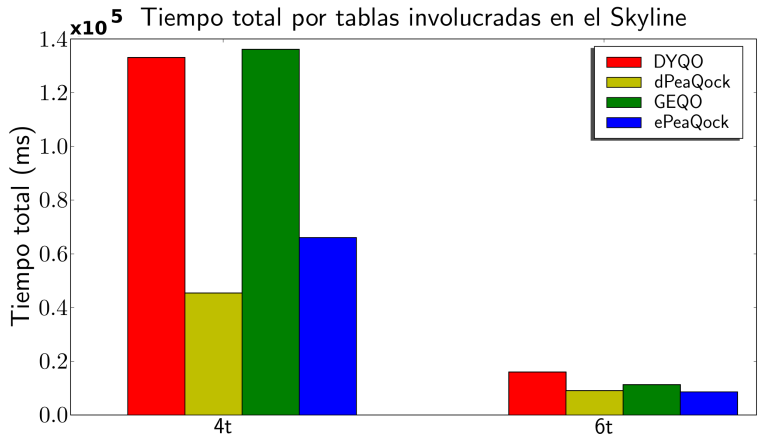


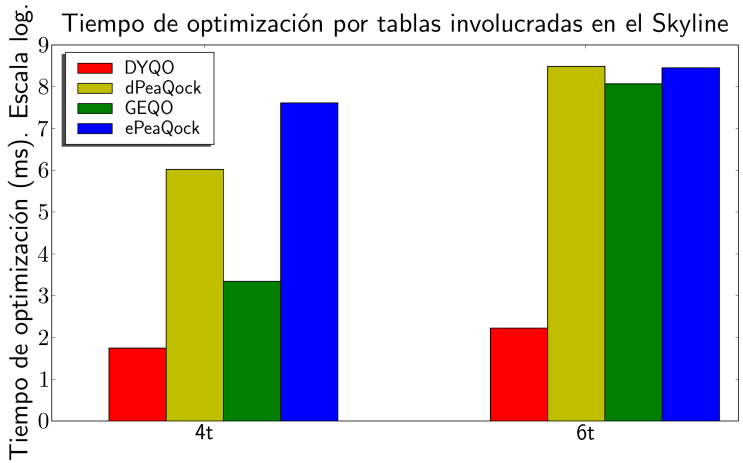


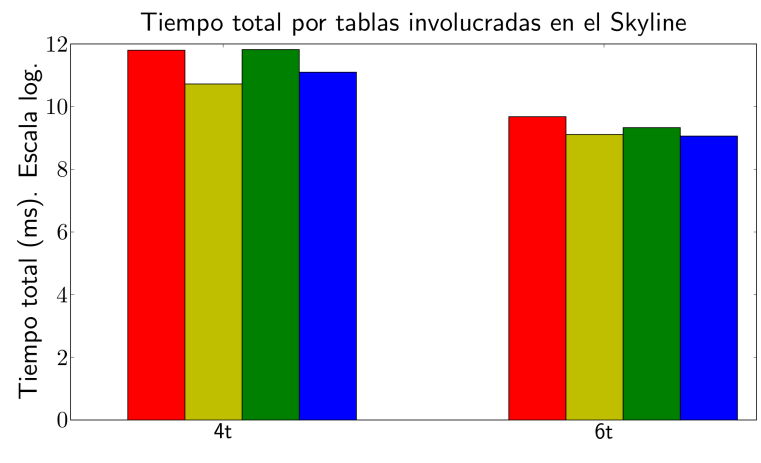
Estudio de los algoritmos:

Tablas involucradas en el Skyline

Tablas	$\frac{T_{DYQO}}{T_{dPeaQock}}$	$\frac{T_{GEQO}}{T_{ePeaQock}}$
4t	2.93	2.06
6t	1.76	1.30



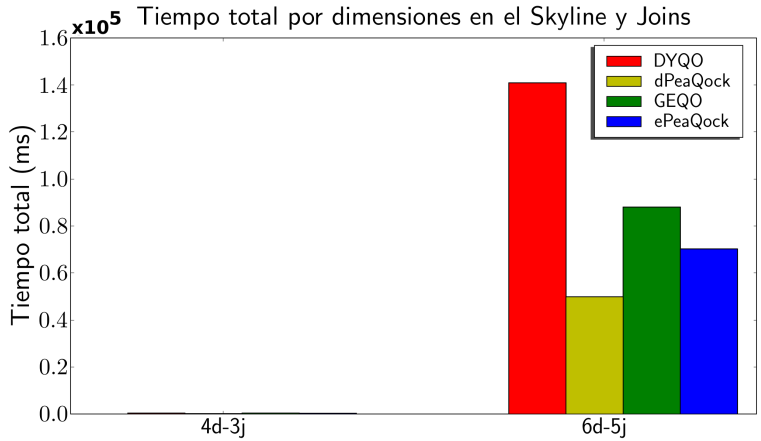


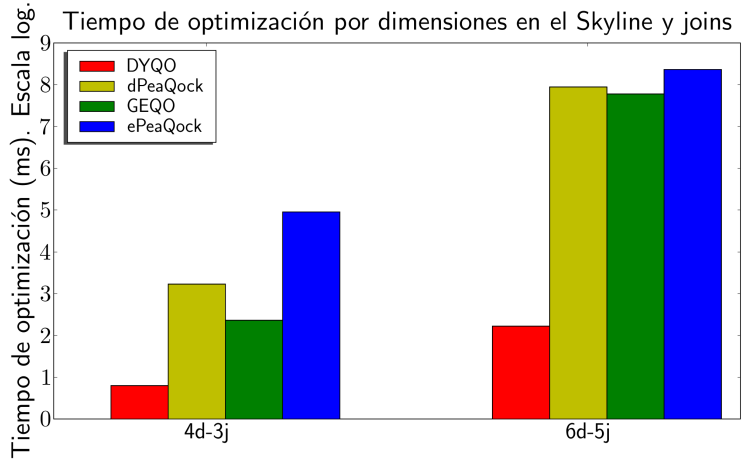


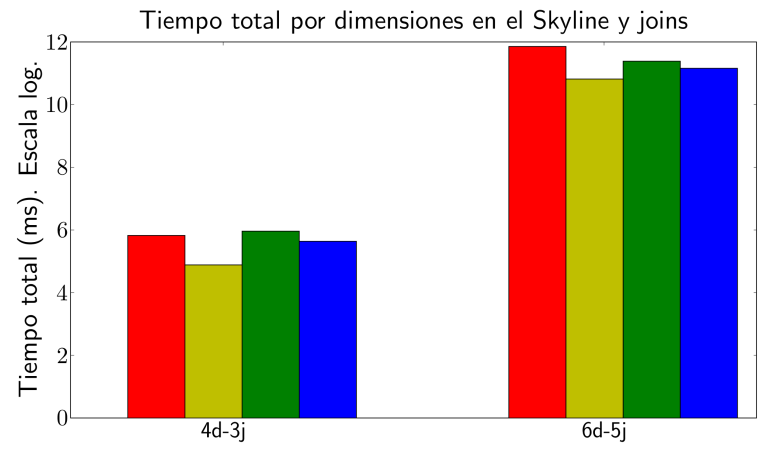
Estudio de los algoritmos:

Número de Joins y Dimensiones del Skyline

Joins-dim	$\frac{T_{DYQO}}{T_{dPeaQock}}$	$\frac{T_{GEQO}}{T_{ePeaQock}}$
3j-4d	2.55	1.38
5j-6d	2.82	1.25

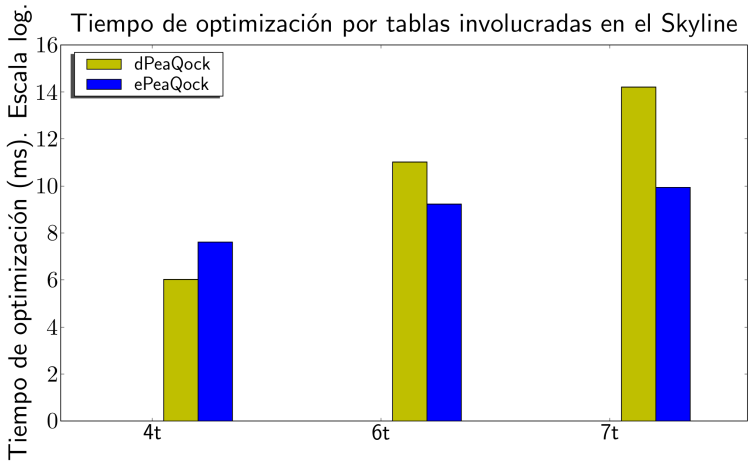


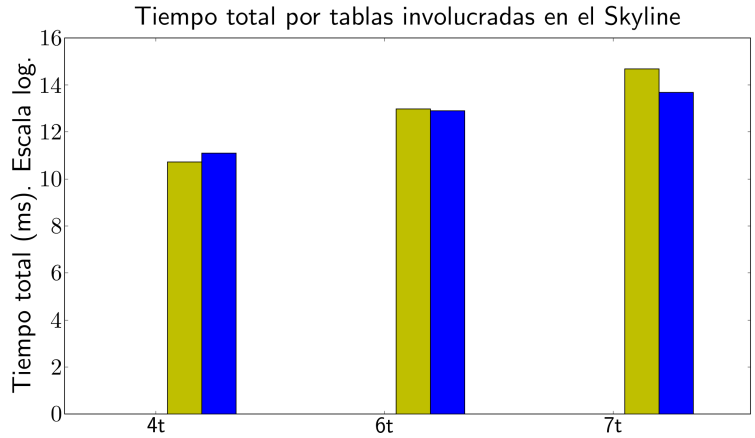


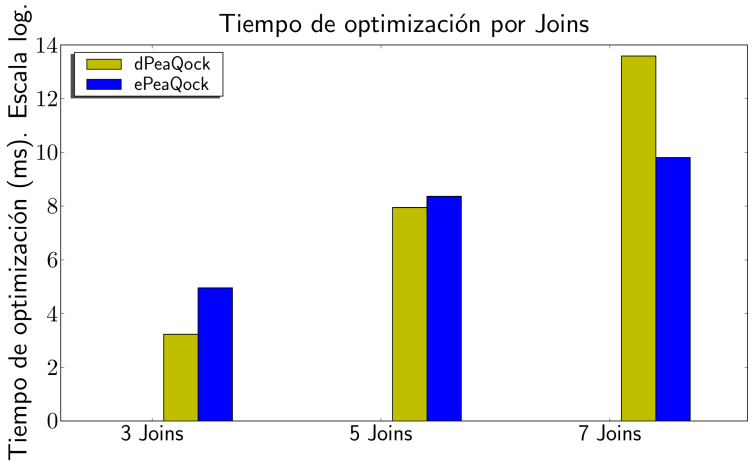


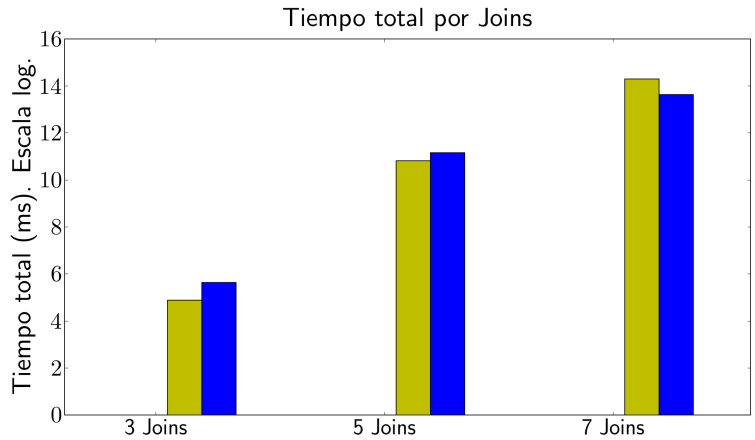
Estudio de los algoritmos:

dPeaQock y ePeaQock

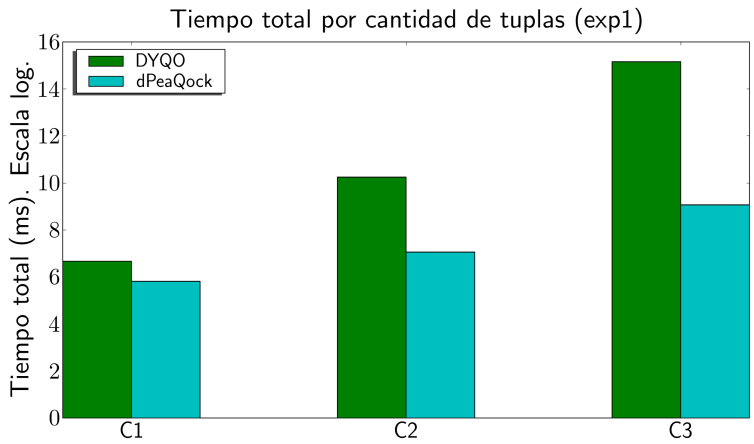


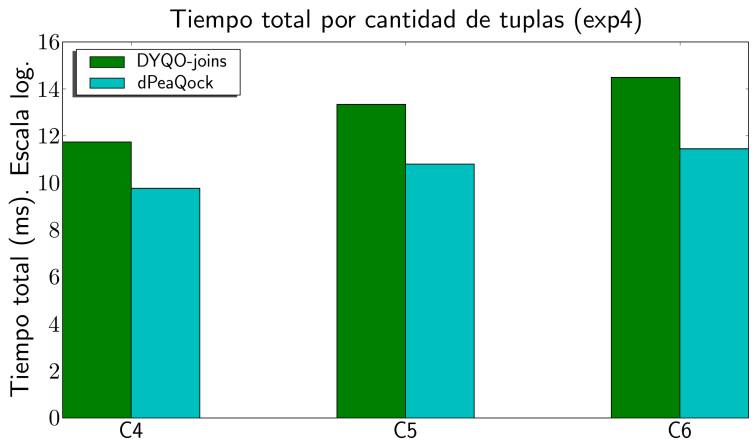






Observaciones sobre la cardinalidad

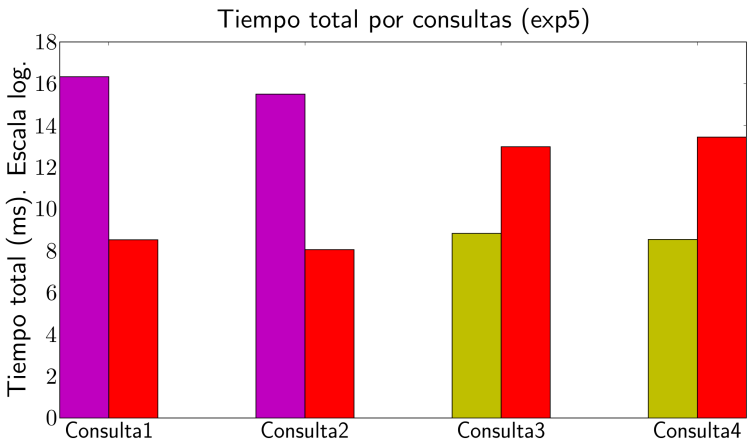




Cardinalidad	$\frac{T_{DYQO}}{T_{dPeaQock}}$
C1 (1.000 tuplas)	2.36
C2 (2.000 tuplas)	24.28
C3 (5.000 tuplas)	442.12

Cardinalidad	$\frac{T_{DYQO\text{-}Joins}}{T_{dPeaQock}}$
C4 (10.000 tuplas)	7.21
C5 (15.000 tuplas)	12.63
C6 (20.000 tuplas)	20.86

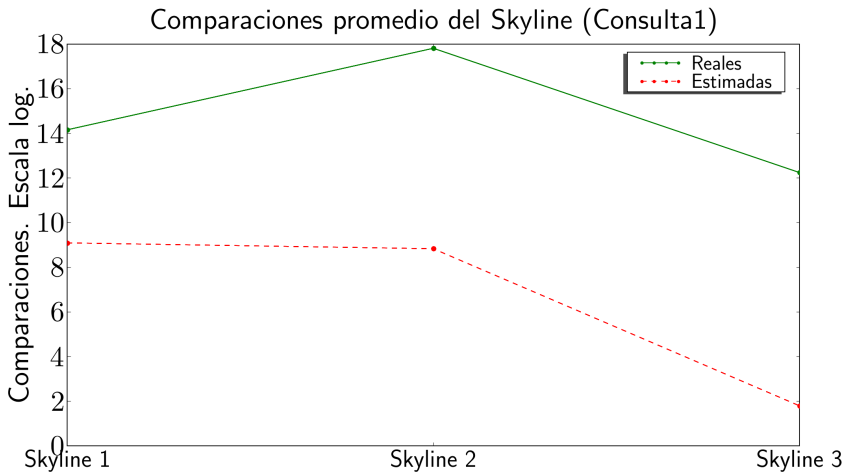
Observaciones sobre datos reales

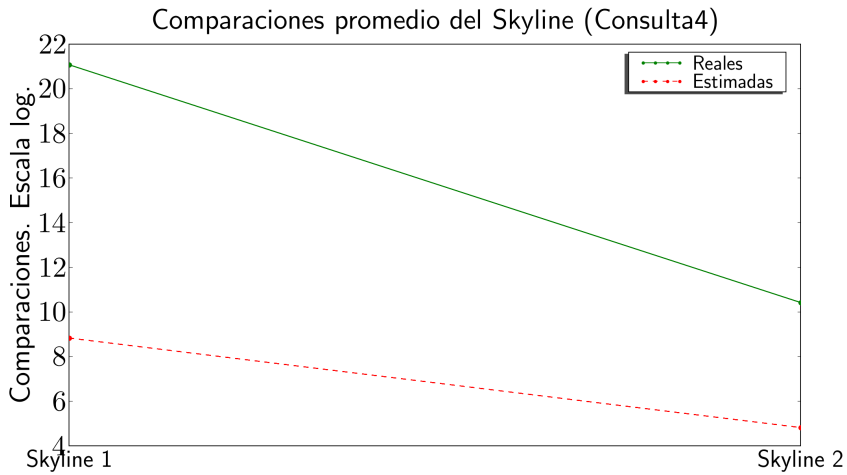


Consulta	$\frac{T_{DYQO}}{T_{dPeaQock}}$
Consulta1	2455.424
Consulta2	1707.884
Consulta3	0.015
Consulta4	0.007

Observaciones sobre datos reales:

Número de comparaciones reales y estimadas





- Nótese la diferencia en la estimación de comparaciones para el primer *Skyline* en las consultas 1 y 4.

Conclusiones

- Se logró extender el mecanismo de optimización de *PostgreSQL* para que permitiera la optimización de consultas *Skyline*.
- Se logró implementar un modelo que se ajusta adecuadamente en la mayoría de los casos de prueba utilizados.
- En las consultas donde resultó favorable realizar la optimización, la mejora en el tiempo de ejecución total es evidente: la mejora mínima es 1,76 veces y la máxima es de 2455,42 veces de *dPeaQock* respecto a *DYQO*.
- Los planes escogidos siempre distribuian al menos una vez el *Skyline*.

Conclusiones

- Se pudo adaptar y transformar la base teórica de trabajos anteriores para este proyecto.
- Se produjeron documentos que permitieran comprender el optimizador de *PostgresSql* y que sirvieran de guía para su extensión.
- Adicionalmente, se planteó e implementó el algoritmo evolutivo *ePeaQock* que permite manejar espacios de búsqueda grandes.

Recomendaciones y trabajos futuros

- Implementación de la primera propuesta del modelo de costo.
- Aproximar las funciones de estimación de costo del *Skyline*.
- Proponer modelos de costo de otros algoritmos para calcular el *Skyline*.
- Realizar un estudio completo del algoritmo *ePeaQock* orientado a su mejora.

Muchas gracias ...

Preguntas...