Caderno de Pesquisa

Disciplina Integradora II

Orientador:
Duncan Dubugras Alcoba Ruiz (duncan.ruiz@pucrs.br)

Alunos:
Eduardo Andrade (eduardo.a@edu.pucrs.br),
Marcelo Heredia (marcelo.heredia@edu.pucrs.br),
Michael Rosa (michael.rosa@edu.pucrs.br)

Projeto:
GADIS: A Genetic Algorithm for Database Index Selection

Repositório:
https://github.com/tecnicasilegais/Integradora2

19/03 - Sexta-Feira

Feito o primeiro contato com o professor Duncan por email solicitando informações sobre como proceder na escolha do artigo para reprodução de pesquisa, onde neste mesmo dia o professor retornou o email nos oferecendo as opções de artigos que poderíamos escolher e fazendo um breve comentário sobre cada uma delas.

23/03 - Terça-Feira

Retornamos o contato do professor com nossa decisão sobre o artigo, onde o grupo através de uma análise das opções optou por escolher o artigo , *GADIS: A Genetic Algorithm for Database Index Selection (S)*, que pode ser encontrado através do repositório da PUCRS no seguinte link:

https://repositorio.pucrs.br/dspace/bitstream/10923/15395/2/GADIS_A_Genetic_Algorithm_for_Database_Index_Selection.pdf

Foi solicitado um um agendamento para futuras reuniões com o professor e também o acesso a dissertação de mestrado que apoia o artigo.

O professor Duncan retornou o email no mesmo dia, anexando a dissertação e agendando nossas reuniões para as Quartas-Feiras das 18h30 às 19h.

24/03 - Quarta-Feira

Tivemos nossa primeira reunião com nosso orientador, onde o mesmo nos apresentou uma prévia do artigo escolhido com algumas informações importantes para que possamos nos ambientar com a proposta do artigo e também reunir informações suficientes para dar início a reprodução da pesquisa.

31/03 - Quarta-Feira

Nesta reunião com o professor Duncan, apresentamos dúvidas sobre o funcionamento dos algoritmos genéticos e como poderia ser aplicado para a reprodução do artigo. Após algumas explicações e dicas, o professor nos indicou os dois sites abaixo:

https://www.algoritmosgeneticos.com.br
http://computacaointeligente.com.br/algoritmos/o-algoritmo-genetico/

05/04 - Segunda-Feira

Nosso grupo se reuniu para decidir o SGBD e a linguagem de programação que utilizaremos para fazer a reprodução. Decidimos utilizar a linguagem C# para implementar o algoritmo genético e a conexão com o banco de dados, a principal razão foi nossa experiência com a linguagem e a facilidade de conexão com o SGBD escolhido, o MySQL.

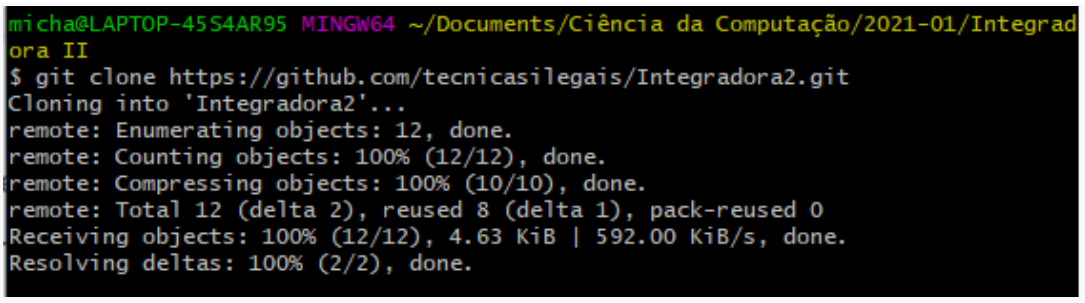
Inicializamos um repositório no GitHub para iniciarmos o desenvolvimento do projeto.

Link do repositório: https://github.com/tecnicasilegais/Integradora2

Para baixar o projeto e rodar ele localmente você precisará:

- Git Bash instalado em sua máquina.
- Executar no terminal do Git Bash o comando seguinte:

```
git clone https://github.com/tecnicasilegais/Integradora2.git
```



06/04 - Terça-Feira

Compilamos o programa de benchmark utilizado no artigo, TPC-H, por meio do arquivo de projeto para Visual Studio (.sln). Após a compilação geramos os dados para teste utilizado o comando "dbgen -vf -s 1", que gera 8 arquivos .tbl.

Com os dados gerados, iniciamos o MySQL e criamos uma base para armazenar as tabelas:

```
mysql -u root -p --local-infile
mysql> CREATE DATABASE tpch;
mysql> USE tpch;
```

Foi necessário alterar o tamanho do buffer do servidor, pois o tamanho padrão não é o suficiente para trabalhar com o tamanho das tabelas:

```
mysql> SET GLOBAL innodb_buffer_pool_size=402653184;
```

Após configurado o tamanho do buffer, as tabelas são criadas,

```
CREATE TABLE NATION  ( N_NATIONKEY  INTEGER NOT NULL,
                       N_NAME       CHAR(25) NOT NULL,
                       N_REGIONKEY  INTEGER NOT NULL,
                       N_COMMENT    VARCHAR(152));

CREATE TABLE REGION  ( R_REGIONKEY  INTEGER NOT NULL,
                       R_NAME       CHAR(25) NOT NULL,
                       R_COMMENT    VARCHAR(152));

CREATE TABLE PART  ( P_PARTKEY     INTEGER NOT NULL,
                     P_NAME        VARCHAR(55) NOT NULL,
                     P_MFGR        CHAR(25) NOT NULL,
                     P_BRAND       CHAR(10) NOT NULL,
                     P_TYPE        VARCHAR(25) NOT NULL,
                     P_SIZE        INTEGER NOT NULL,
                     P_CONTAINER   CHAR(10) NOT NULL,
                     P_RETAILPRICE DECIMAL(15,2) NOT NULL,
                     P_COMMENT     VARCHAR(23) NOT NULL );

CREATE TABLE SUPPLIER ( S_SUPPKEY     INTEGER NOT NULL,
                        S_NAME        CHAR(25) NOT NULL,
                        S_ADDRESS     VARCHAR(40) NOT NULL,
                        S_NATIONKEY   INTEGER NOT NULL,
                        S_PHONE       CHAR(15) NOT NULL,
                        S_ACCTBAL     DECIMAL(15,2) NOT NULL,
                        S_COMMENT     VARCHAR(101) NOT NULL);

CREATE TABLE PARTSUPP ( PS_PARTKEY     INTEGER NOT NULL,
                        PS_SUPPKEY     INTEGER NOT NULL,
                        PS_AVAILQTY    INTEGER NOT NULL,
                        PS_SUPPLYCOST  DECIMAL(15,2)  NOT NULL,
```

```
                    PS_COMMENT     VARCHAR(199) NOT NULL );

CREATE TABLE CUSTOMER ( C_CUSTKEY     INTEGER NOT NULL,
                    C_NAME       VARCHAR(25) NOT NULL,
                    C_ADDRESS    VARCHAR(40) NOT NULL,
                    C_NATIONKEY  INTEGER NOT NULL,
                    C_PHONE      CHAR(15) NOT NULL,
                    C_ACCTBAL    DECIMAL(15,2)   NOT NULL,
                    C_MKTSEGMENT  CHAR(10) NOT NULL,
                    C_COMMENT    VARCHAR(117) NOT NULL);

CREATE TABLE ORDERS  ( O_ORDERKEY      INTEGER NOT NULL,
                   O_CUSTKEY       INTEGER NOT NULL,
                   O_ORDERSTATUS    CHAR(1) NOT NULL,
                   O_TOTALPRICE     DECIMAL(15,2) NOT NULL,
                   O_ORDERDATE      DATE NOT NULL,
                   O_ORDERPRIORITY  CHAR(15) NOT NULL,
                   O_CLERK         CHAR(15) NOT NULL,
                   O_SHIPPRIORITY   INTEGER NOT NULL,
                   O_COMMENT       VARCHAR(79) NOT NULL);

CREATE TABLE LINEITEM ( L_ORDERKEY    INTEGER NOT NULL,
                    L_PARTKEY     INTEGER NOT NULL,
                    L_SUPPKEY     INTEGER NOT NULL,
                    L_LINENUMBER  INTEGER NOT NULL,
                    L_QUANTITY    DECIMAL(15,2) NOT NULL,
                    L_EXTENDEDPRICE  DECIMAL(15,2) NOT NULL,
                    L_DISCOUNT    DECIMAL(15,2) NOT NULL,
                    L_TAX        DECIMAL(15,2) NOT NULL,
                    L_RETURNFLAG  CHAR(1) NOT NULL,
                    L_LINESTATUS  CHAR(1) NOT NULL,
                    L_SHIPDATE    DATE NOT NULL,
                    L_COMMITDATE  DATE NOT NULL,
                    L_RECEIPTDATE DATE NOT NULL,
                    L_SHIPINSTRUCT CHAR(25) NOT NULL,
                    L_SHIPMODE     CHAR(10) NOT NULL,
                    L_COMMENT      VARCHAR(44) NOT NULL);
```

populadas com os dados gerados

```
LOAD DATA LOCAL INFILE 'region.tbl' INTO TABLE REGION FIELDS TERMINATED BY '|';
LOAD DATA LOCAL INFILE 'part.tbl' INTO TABLE PART FIELDS TERMINATED BY '|';
LOAD DATA LOCAL INFILE 'nation.tbl' INTO TABLE NATION FIELDS TERMINATED BY '|';
LOAD DATA LOCAL INFILE 'customer.tbl' INTO TABLE CUSTOMER FIELDS TERMINATED BY '|';
LOAD DATA LOCAL INFILE 'supplier.tbl' INTO TABLE SUPPLIER FIELDS TERMINATED BY '|';
LOAD DATA LOCAL INFILE 'orders.tbl' INTO TABLE ORDERS FIELDS TERMINATED BY '|';
LOAD DATA LOCAL INFILE 'partsupp.tbl' INTO TABLE PARTSUPP FIELDS TERMINATED BY '|';
LOAD DATA LOCAL INFILE 'lineitem.tbl' INTO TABLE LINEITEM FIELDS TERMINATED BY '|';
```

e têm as dependências alteradas.

```
ALTER TABLE REGION
ADD PRIMARY KEY (R_REGIONKEY);
ALTER TABLE NATION
ADD PRIMARY KEY (N_NATIONKEY);
ALTER TABLE NATION
ADD FOREIGN KEY NATION_FK1 (N_REGIONKEY) references REGION(R_REGIONKEY);
ALTER TABLE PART
ADD PRIMARY KEY (P_PARTKEY);
ALTER TABLE SUPPLIER
ADD PRIMARY KEY (S_SUPPKEY);
ALTER TABLE SUPPLIER
ADD FOREIGN KEY SUPPLIER_FK1 (S_NATIONKEY) references NATION(N_NATIONKEY);
ALTER TABLE PARTSUPP
ADD PRIMARY KEY (PS_PARTKEY,PS_SUPPKEY);
ALTER TABLE CUSTOMER
ADD PRIMARY KEY (C_CUSTKEY);
ALTER TABLE CUSTOMER
ADD FOREIGN KEY CUSTOMER_FK1 (C_NATIONKEY) references NATION(N_NATIONKEY);
ALTER TABLE LINEITEM
ADD PRIMARY KEY (L_ORDERKEY,L_LINENUMBER);
ALTER TABLE PARTSUPP
ADD FOREIGN KEY PARTSUPP_FK1 (PS_SUPPKEY) references SUPPLIER(S_SUPPKEY);
ALTER TABLE PARTSUPP
ADD FOREIGN KEY PARTSUPP_FK2 (PS_PARTKEY) references PART(P_PARTKEY);
ALTER TABLE ORDERS
ADD PRIMARY KEY (O_ORDERKEY);
ALTER TABLE ORDERS
ADD FOREIGN KEY ORDERS_FK1 (O_CUSTKEY) references CUSTOMER(C_CUSTKEY);
ALTER TABLE LINEITEM
ADD FOREIGN KEY LINEITEM_FK1 (L_ORDERKEY)  references ORDERS(O_ORDERKEY);
ALTER TABLE LINEITEM
ADD FOREIGN KEY LINEITEM_FK2 (L_PARTKEY,L_SUPPKEY) references PARTSUPP(PS_PARTKEY,
PS_SUPPKEY);
```

07/04 - Quarta-Feira

Tivemos reunião com o professor Duncan, onde foi apresentado a estrutura do projeto, como as definições da linguagem e banco de dados que serão utilizados pelo time, também foi apresentado um problema encontrado pelo grupo onde a biblioteca escolhida pelo time para apoiar o desenvolvimento do algoritmo genético possui algumas limitações, nesta mesma reunião o professor exemplificou como é feito o processo de crossover e mutação.

Um dos problemas encontrados foi que a biblioteca escolhida (GeneticSharp) não possui suporte ao NSGA-II que foi implementado na pesquisa, o problema está descrito no FAQ da Wiki da biblioteca que pode ser encontrado no repositório abaixo:
https://github.com/giacomelli/GeneticSharp/wiki/FAQ.

Nesta mesma reunião, o professor comentou que, devido ao tempo que ainda temos, devíamos tomar essa decisão de linguagem e bibliotecas e avançar com a mesma, para que não perdêssemos muito tempo nisso. No mesmo dia fizemos uma busca de outras bibliotecas de C# que suportam o método de avaliação NSGA-II, mas sem sucesso.

08/04 - Quinta-Feira

Devido aos problemas encontrados e também discutidos com o professor, o grupo optou por trocar de linguagem, fizemos uma migração de C#, que seria apoiada pela biblioteca GeneticSharp, para usar o Python, que terá apoio da biblioteca DEAP. Foi atualizado o repositório, e todos os arquivos da versão anterior do projeto podem ser encontrados na pasta “old” conforme o link
https://github.com/tecnicasilegais/Integradora2/tree/main/old.

A principal razão pela escolha de Python como nova linguagem do projeto foi por esta ser a linguagem na vanguarda das pesquisas de IA, além de possuir as principais bibliotecas da área, pois será nela que você encontrará a maioria das estruturas de machine learning e deep learning. Já a biblioteca DEAP, que é a mesma que foi escolhida no artigo que estamos replicando, possui um portfólio grande de ferramentas que podem ser usadas para auxiliar no desenvolvimento de algoritmos genéticos. A documentação da biblioteca pode ser encontrada no link abaixo:
https://deap.readthedocs.io/en/master/tutorials/advanced/gp.html

Alguns exemplos que usamos para o desenvolvimento inicial:
https://github.com/DEAP/deap/blob/d328fe6b68e7528b2d2d990bb2ab1ad1786e6f58/examples/ga/onemax.py
https://deap.readthedocs.io/en/master/overview.html

Para geração de população aleatória e demais usos de aleatórios usamos a biblioteca NumPy, consultando este link:
https://numpy.org/doc/stable/reference/random/generator.html

12/04 - Segunda-Feira

Fizemos como primeiro exemplo usando a biblioteca DEAP um código que já tínhamos de Algoritmo Genético, que resolvia o problema de distribuição de cargas. Esse algoritmo, que fizemos na aula de IA desse mesmo semestre com a professora Sílvia nos foi muito útil para entender o funcionamento do DEAP.

Com a conclusão do algoritmo de distribuição de cargas usando DEAP, já enfrentamos os primeiros problemas, que foi a impossibilidade de usar os algoritmos automáticos do DEAP para resolver o Algoritmo Genético. O motivo disto é que, como nosso critério de seleção usa tanto elitismo quanto torneio, tivemos que criar um método novo de seleção que chamava os dois de forma adaptada, pois o DEAP em si não implementa os dois juntos. Para conseguir resolver esse problema e gerar a parte principal do algoritmo de forma manual, achamos uma resposta nesses links:

https://github.com/DEAP/deap/blob/master/deap/tools/selection.py

https://stackoverflow.com/questions/42724381/python-deap-how-to-stop-the-evolution-when-the-fitness-doesnt-increase-after-x

O link acima, além de ter nos ajudado com a questão do método principal feito de forma manual, nos ajudou com outro problema que já tínhamos encontrado, a implementação do critério de parada.

Código resultante:

```
    def sel_elite_tournament(individuals, k_elitist, k_tournament, tournsize):
  return tools.selBest(individuals, k_elitist) + tools.selTournament(individuals, k_tournament,
tournsize=2)

toolbox.register('select', sel_elite_tournament, k_elitist=1, k_tournament=POP_SIZE - 1, tournsize=2)
```

O motivo do problema é que o algoritmo simples de execução de GAs chama o método de seleção mandando como parâmetro a população e o tamanho dela. Mas nosso método necessita de outros parâmetros e não recebe o tamanho da população, que é facilmente acessível pelo vetor contendo a população em si.

13/04 - Terça-Feira

Com o conhecimento adquirido durante a implementação do problema de Distribuição de Cargas, começamos a implementar o GA para o artigo em si, muitas coisas foram reaproveitadas, como o problema do critério de parada e o método composto de seleção. Fizemos mais as seguintes leituras:

https://medium.com/@rossleecooloh/optimization-algorithm-nsga-ii-and-python-package-deap-fca0be6b2ffc

https://stackoverflow.com/questions/56777610/deap-implementing-nsga-ii-for-flight-ticket-problem

https://github.com/DEAP/deap/blob/master/examples/ga/nsga2.py

https://github.com/DEAP/deap/blob/master/deap/tools/emo.py

Após as leituras, onde buscamos respostas sobre o uso do algoritmo de avaliação NSGA-II usado no artigo, encontramos respostas que nos confundiram. Nos exemplos acontece uma substituição dos métodos de seleção pelo do NSGA-II, ou seja, teríamos que tirar a seleção por elitismo e de torneio e colocar o método NSGA-II no lugar.

Enquanto tentamos criar o algoritmo genético, fizemos as conexões ao banco de dados através do Python, segundo as instruções do link:

https://dev.mysql.com/doc/connector-python/en/connector-python-example-connecting.html.

14/04 - Quarta-Feira

Tivemos a reunião com o professor Duncan, onde foram apresentados os problemas encontrados durante a semana.

Um dos problemas encontrados foi na implementação do método de seleção da biblioteca DEAP. Os métodos de Elitismo, Torneio e o NSGA-II são considerados métodos de seleção e, de forma nativa da biblioteca, só se poderia chamar um deles. Apresentamos este questionamento com o professor, pois até então fomos capazes de modificar o método de seleção para que suportasse Elitismo e Torneio juntos. Porém não sabíamos como fazer para executar o método NSGA-II junto com os outros dois. A primeira informação que o professor nos forneceu é que o método NSGA-II implementa elitismo dentro de si, só ficando a questão do Torneio. Discutimos um pouco sobre como resolver o torneio para que pudéssemos replicar a pesquisa de forma fiel.

Os outros problemas foram relacionados ao banco de dados.

O artigo detalha que 24 índices foram utilizados, totalizando 24 genes. Já na dissertação que estamos usando como apoio para criação do código e base de dados, é mencionado que foram usados 23 índices. Ainda na dissertação é mencionado que isso possibilita 2^{22} possibilidades, e demonstrada uma tabela de índices contendo 22 índices e genes correspondentes. O professor nos orientou a utilizar 22 índices, como dito no número de possibilidades e tabela.

Com relação à medição de desempenho, a dissertação considera que o indivíduo base de cálculo é o que tem todas as colunas indexadas, e fala:

“Quanto maior a razão entre o indivíduo utilizado como base de cálculo e o indivíduo atual, mais ganho na otimização do tempo de resposta será alcançado.” Discutindo com o professor, concluímos que o ganho na otimização do tempo de resposta aumenta com a **diminuição** da razão, diferente do que diz a dissertação.

Discutimos também como medir o tamanho ocupado pelos índices, e o professor nos forneceu um script SQL para testar.

15/04 - Quinta-Feira

Com mais um pouco de leitura e revisão dos códigos exemplos do DEAP, achamos a resposta sobre o Torneio junto com NSGA-II nesse exemplo:

https://github.com/DEAP/deap/blob/master/examples/ga/nsga2.py

Neste exemplo é declarado o método de seleção apenas como o NSGA-II:

```
toolbox.register("select", tools.selNSGA2)
```

Porém, durante a iteração das gerações, não é utilizado apenas esse método, mas sim um outro de torneio que não é o método de Torneio Binário.

```
for gen in range(1, NGEN):
    offspring = tools.selTournamentDCD(pop, len(pop)) # Aqui está o método. <=
    offspring = [toolbox.clone(ind) for ind in offspring]

    for ind1, ind2 in zip(offspring[::2], offspring[1::2]):
        if random.random() <= CXPB:
            toolbox.mate(ind1, ind2)

        toolbox.mutate(ind1)
        toolbox.mutate(ind2)
        del ind1.fitness.values, ind2.fitness.values

    invalid_ind = [ind for ind in offspring if not ind.fitness.valid]
    fitnesses = toolbox.map(toolbox.evaluate, invalid_ind)
    for ind, fit in zip(invalid_ind, fitnesses):
        ind.fitness.values = fit

    pop = toolbox.select(pop + offspring, MU)
    record = stats.compile(pop)
    logbook.record(gen=gen, evals=len(invalid_ind), **record)
    print(logbook.stream)

print("Final population hypervolume is %f" % hypervolume(pop, [11.0, 11.0]))

return pop, logbook
```

Como podemos ver na segunda linha de código, é utilizado um método chamado `selTournamentDCD`, este utiliza o conceito de Dominância para realizar o torneio, ou seja, é feito para funcionar com métodos como NSGA-II.

21/04 - Quarta-Feira

Foi criado a estrutura do projeto onde será armazenado as queries, para isso foi criado uma classe auxiliar contendo todas as queries necessárias para reprodução de pesquisa, seja para o controle de índices (criação e exclusão) ou consultas.

O grupo precisou desenvolver algumas queries, pois nem o artigo, nem a dissertação tinham a query usada para determinados cálculos, um exemplo de query criada foi a de cálculo do tamanho de índice da base, segue o código abaixo:

```
select sum(round(index_length/1024/1024,2)) as index_size
from information_schema.tables
where table_type = 'BASE TABLE'
  and table_schema = 'tpch';
```

Outro exemplo, foi a correção de uma query de consulta que estava junto às demais queries no “Apêndice A” da dissertação, pois a query apresentava um erro, após a correção da query, a mesma ficou assim:

```
create view REVENUE0 (supplier_no, total_revenue) as
  select l_suppkey, sum(l_extendedprice * (1 - l_discount))
  from LINEITEM
  where l_shipdate >= date '1997-07-01'
     and l_shipdate < date '1997-07-01' + interval '3' month
  group by l_suppkey;
  select s_suppkey, s_name, s_address, s_phone, total_revenue
  from SUPPLIER, REVENUE0
  where s_suppkey = supplier_no
     and total_revenue = (
        select max(total_revenue)
        from REVENUE0
     )
  order by s_suppkey;
drop view REVENUE0;
```

22/04 - Quinta-Feira

A query corrigida no dia anterior apresentou problemas no algoritmo por ter várias seleções. Corrigimos o algoritmo e inserimos os códigos de alteração dos índices.

Com a finalização dos códigos de acessos e coletas do banco de dados, conseguimos testar efetivamente o algoritmo genético. Como desenvolvemos as cegas o tempo todo, encontramos alguns erros, mas todos foram corrigidos. Adicionamos o critério de parada por convergência utilizando a fitness das 10 últimas gerações

23/04 - Sexta-Feira

Em reunião com o professor orientador, resolvemos dúvidas sobre as consultas mencionadas na dissertação, que continham erros, e o uso dos índices. Discutimos com ele a diminuição do tempo de execução do algoritmo, que de acordo com os cálculos levaria em torno de 27 dias, e concordamos em gerar bases de tamanho menor, 100, 500 e 700 MiB.

Após alguns testes no Algoritmo Genético usando as novas bases geradas do TPC-H, vimos que a população estava convergindo rapidamente, o que é um tanto estranho. Analisando melhor os cálculos de aptidão contidos na dissertação referente ao artigo, constatamos que, embora fale-se muito de minimização, tanto no tempo de consulta quanto no tamanho dos índices do banco de dados, o problema da aptidão era de maximização. Isso se deve ao fato de que foi feita uma normalização no cálculo do objetivo onde o número a ser gerado teria como limite superior o número 1, que seria o caso ideal do objetivo a ser testado.

Implementamos alguns Logs de sistema no algoritmo tanto para testar quanto para armazenar os resultados de gerações para que possamos ver a evolução do algoritmo a cada geração.

Todas as novas bases menores executaram em alguns minutos, porém a original de 1 GiB continuava não executando nem a primeira geração, então mudamos as configurações do SGBD para melhor aproveitar a máquina:

Configuração	Valor inicial	Valor utilizado
innodb_buffer_pool_size	550M	2048M
read_buffer_size	64K	256K
read_rnd_buffer_size	256K	1024K
innodb_log_buffer_size	1M	4M
sort_buffer_size	256K	1024K
join_buffer_size	256K	1024K
innodb_log_file_size	48M	256M
max_allowed_packet	4M	16M

24/04 - Sábado

O tamanho dos índices não estava variando, então decidimos analisar as bases de dados. Descobrimos que a tabela de informações do MySQL não é atualizada, portanto os índices tinham sempre o mesmo tamanho. Alteramos a fonte utilizada para obtenção do tamanho dos índices:

```
select sum(round(stat_value * @@innodb_page_size / 1024 / 1024, 2)) size_in_mb
from mysql.innodb_index_stats
where stat_name = 'size' and database_name = 'tpch';
```

Links consultados:
python - Comparing two NumPy arrays for equality, element-wise - Stack Overflow
https://docs.python.org/pt-br/3/howto/logging.html
https://www.programiz.com/python-programming/datetime/strftime

25/04 - Domingo

Contatamos o professor Duncan sobre a mudança das configurações do SGBD, ele disse que não tem problema desde que o experimento seja todo feito na mesma configuração. Iniciamos as execuções do algoritmo genético para coletar resultados.

Pedimos ao professor uma sugestão de contribuição para o trabalho, o mesmo recomendou que fizéssemos uma otimização na troca de índices do algoritmo genético, de forma que os indivíduos fossem ordenados para que houvesse menos trocas de índice por execução.

26/04 - Segunda-Feira

Agora que tudo está funcionando, seguimos colhendo resultados para o artigo e apresentação.

05/05 - Quarta-Feira

Foi feito o último encontro com nosso orientador antes da entrega da primeira etapa de avaliação, neste encontro foram mostrados alguns resultados coletados durante a semana, neste mesmo dia o grupo aproveitou para finalizar o powerpoint de apresentação.

07/05 - Sexta-Feira

Apresentamos a nossa reprodução de pesquisa para os professores e para os demais alunos da disciplina.

12/05 - Quarta-Feira

O grupo se encontrou com o orientador e debateu estratégias para implementar o algoritmo de Levenshtein, que foi a sugestão de melhoria do grupo para segunda etapa da reprodução, o algoritmo de Levenshtein se propõe a otimizar o algoritmo da pesquisa e torná-lo mais rápido em sua execução, esta é uma etapa muito importante da reprodução, pois a sua implementação terá um impacto muito positivo e irá diminuir a quantidade de criação e exclusão de índices, consequentemente diminuindo o custo da reprodução.

15/05 - Sábado

Começamos a testar diferentes bibliotecas de cálculo de distância de edição para ordenar os indivíduos, incluindo a alternativa de Damerau-Levenshtein, porém ao analisar melhor os casos comparativos que temos constatamos que esta não era necessária. Tivemos a ideia de utilizar uma distância de Levenshtein com pesos maiores para criação de índices, porém a biblioteca que disponibiliza isto funciona apenas com versões mais antigas de Python onde o conector MySql não funciona. Testamos a ordenação utilizando a distância entre cada indivíduo e um indivíduo sem índices criados.

17/05 - Segunda-Feira

Descobrimos um erro no código usado na reprodução que impacta o desempenho do algoritmo genético. Estávamos gerando a população da geração seguinte usando a chamada de torneio com dominância da biblioteca DEAP porém, a chamada de torneio não preenche o restante da população automaticamente. Com isso, a população da geração seguinte iniciava em 8 (k=5 gera 8 indivíduos na biblioteca), os operadores de cruzamento e mutação eram usados nesses 8 indivíduos e só após isso era preenchida a população com os 50 indivíduos. Corrigimos o erro preenchendo a população assim que é feita a chamada do torneio, de modo que possam acontecer mais mutações e cruzamentos.

19/05 - Quarta-Feira

Tivemos o encontro semanal com nosso orientador onde foi apresentado a ele o erro encontrado durante a semana no algoritmo, tendo em vista que o erro impacta diretamente o desempenho do algoritmo, o grupo buscou saber do orientador os próximos passos para avançarmos na reprodução e contribuição da pesquisa, e fomos

orientados a seguir com o desenvolvimento da etapa de melhoria e deixar a correção do erro para segundo plano.

23/05 - Domingo

Encontramos uma nova biblioteca de distâncias que implementa pesos em Levenshtein, “strsimpy”, e utilizamos ela para nossos testes. Alteramos o código de cálculo de distância para que ele inicie com o indivíduo mais parecido com o sem indícios e calcule a partir deste resultado o próximo mais parecido.

https://pypi.org/project/strsimpy/

26/05 - Quarta-Feira

Tivemos nosso encontro semanal com o orientador do grupo, onde foi mostrada toda a evolução da etapa de melhoria já com a nova biblioteca funcionando. Combinamos de coletar resultados para comparar a execução com e sem a ordenação pela distância de Levenshtein.

28/05 - Sábado

Iniciamos o processo de coleta de resultados de execuções para analisar, gerar gráficos e colocar no artigo. Percebemos que não houveram grandes diferenças entre os tempos de execução do algoritmo com e sem a ordenação, separamos os resultados para discuti-los com o professor na quarta-feira.

02/06 - Quarta-Feira

Neste dia tivemos a reunião com o nosso orientador onde foram apresentados os resultados obtidos na coleta realizada durante a semana, resultados estes onde aparentemente não se obteve muitos ganhos com a implementação do algoritmo de Levenshtein, discutimos o problema e levantamos algumas possibilidades que podem ter impactado no resultado, sendo uma delas, a possibilidade do SGBD estar otimizando a criação dos índices por conta própria, pelo fato dele já ter criado anteriormente o índice, outra a possibilidade do método ter pouco impacto em função de tratar todas colunas igualmente, quando algumas colunas detém a maior parte do espaço em disco do banco de dados inteiro.

05/06 - Sábado

Formatamos os resultados da comparação antes e depois de ordenar os indivíduos, e calculamos a média e desvio padrão da distância de Levenshtein entre cada indivíduo e o anterior. As distâncias estão diminuindo como esperado, é provável que o SGBD esteja realizando otimizações, fazendo com que não seja tão efetivo como gostaríamos.

09/06 - Quarta-Feira

Em reunião com o orientador, concordamos que os resultados eram para ser mais promissores do que aparentam. Decidimos considerar apenas as tabelas com maior espaço em disco no método de Levenshtein para ver se há um impacto maior na velocidade do programa. Caso contrário, a possibilidade é que o SGBD esteja guardando os índices descartados e recuperando-os, fazendo com que a solução pareça menos relevante do que é.

12/06 - Sábado

Modificamos o código que ordena os indivíduos usando o método de Levenshtein para que sejam consideradas apenas as colunas das tabelas com mais dados. Em função do pouco tempo que nos resta, fizemos de forma hard-coded escolhendo as colunas que verificamos serem as maiores. Iniciamos a coleta de resultados para discutir com o professor.

- **Projetos futuros**: Decidir quais colunas usar na ordenação de forma automática em código, calculando na execução do algoritmo o tamanho de cada tabela e fazendo a decisão usando os dados.

14/06 - Segunda-Feira

Fizemos coleta dos tempos de execução na base tpch de 100MB, constatamos que não houve uma diferença significativa, na verdade ambas execuções (código normal e com uso de Levenshtein) os tempos são parelhos na maior parte das gerações.

16/06 - Quarta-Feira

Mostramos os resultados para o orientador. Foi constatado que o SGBD pode estar sim influenciando os resultados, mas percebemos que os dados de tempo que coletamos até então eram as durações de geração, não apenas o tempo gasto criando e excluindo índices. Como já temos métodos que fazem o cálculo do tempo apenas

fazendo operações de índices no banco de dados, decidimos coletá-los e usá-los para os resultados, de forma que tenhamos precisão nos resultados a serem apresentados.

18/06 - Sexta-Feira

Coletamos os resultados mais precisos tanto na base tpch de 100MB quanto na de 500MB, constatamos que realmente há uma melhora considerável no tempo de execução. Iniciamos a escrita do artigo usando alguns trechos do antigo como base. Estamos criando as seções que são necessárias para o artigo novo e gerando gráficos para colocar na seção de resultados.