Otimização de índices com distanciamento de Levenshtein - GADIS

Marcelo P. R. Heredia, Eduardo C. Andrade, Michael L. S. Rosa, Duncan D. A. Ruiz

Escola Politécnica – Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
Porto Alegre – RS – Brazil

{marcelo.heredia, eduardo.a, michael.rosa}@edu.pucrs.br,
duncan.ruiz@pucrs.br

Abstract. *This article aims to describe the process of coding and studies done in order to contribute to the article 'GADIS: A Genetic Algorithm for Database Index Selection' (Neuhaus et al., 2019). One of the biggest concerns of the Data Science area is database performance. The reproduced article proposes the use of genetic algorithms, using them to discover combinations of indexes that offer the best balance between used space and gained query operations performance. As a contribution to the research, the 'Levenshtein distance' algorithm was added in the reproduction in order to find the best sequence that minimizes the differences between each pair of individuals tested, reducing the creation and exclusion of indexes and, consequently, the algorithm's execution time and the DBMS cost to perform the operations with the indexes.*

Resumo. *Este artigo tem como objetivo descrever o processo de codificação e estudos realizados com a finalidade de fazer uma contribuição ao artigo 'GADIS: A Genetic Algorithm for Database Index Selection' (Neuhaus et al., 2019). Uma das maiores preocupações do ramo da ciência de dados é o desempenho do banco de dados. O artigo reproduzido propõe o uso de algoritmos genéticos, usando-os para descobrir combinações de índices que ofereçam o melhor balanço entre espaço utilizado e desempenho de operações de consulta ganho. Como contribuição para a pesquisa, foi adicionado na reprodução o algoritmo de 'distância de Levenshtein' com o objetivo de encontrar a melhor ordem que minimize a diferença entre cada dois indivíduos testados, diminuindo a criação e exclusão de índices e, consequentemente, o tempo de execução do algoritmo e o custo do SGBD para realizar as operações com os índices.*

Palavras-Chave

Indexação de BD, Algoritmos Genéticos, IA, Levenshtein, Otimização

1. Introdução

O crescimento no volume de dados nos últimos anos vem se dando de forma exponencial. Muitos negócios se viram obrigados a buscar soluções para garantir um melhor desempenho de suas bases de dados, com isso surgiram diferentes estratégias, com objetivo de melhorar o desempenho dos sistemas. O baixo desempenho de um banco de dados pode acarretar não somente um desperdício de tempo para o negócio, mas também um alto custo computacional, fazendo com que sejam destinados mais recursos para a aplicação. Uma técnica usada para melhorar este desempenho da base de dados é a inclusão de índices, que tem como objetivo tornar uma consulta a um banco de dados mais rápida. Seu funcionamento se dá semelhante ao sumário de um livro, em

que nele você encontra a página correta de acordo com o assunto e não precisa ficar folheando o livro até encontrar o assunto desejado.

Ao mesmo tempo que a criação de índices é capaz de aprimorar o desempenho de rotinas de consultas a bancos de dados, existem consequências. Criar índices no banco de dados aumenta o espaço em disco utilizado e, além disso, diminui o desempenho de rotinas de inclusão/exclusão do banco de dados, já que cada uma dessas alterações deve ser refletida nos índices.

Tendo em vista o problema descrito, o objetivo do artigo reproduzido é otimizar o tempo de consulta a um banco de dados relacional, utilizando a técnica de indexação descrita anteriormente. Para que isso seja possível é usado o algoritmo genético, que tem como objetivo explorar as soluções de indexação buscando um conjunto de índices que reduza o tempo de respostas de conjuntos de consultas submetidas de forma frequente por um determinado período de tempo e, ao mesmo tempo, considerando a quantidade de memória secundária alocada. Adicionalmente, espera-se obter uma boa relação de custo-benefício entre a configuração de índices gerada e a sua utilização pelo otimizador do sistema no plano de execução das consultas.

Este artigo se propõe a contribuir com a pesquisa desenvolvida no artigo GADIS: A Genetic Algorithm for Database Index Selection (Neuhaus et al., 2019). Para isto, será usado o algoritmo de distância de Levenshtein com o objetivo de diminuir o custo computacional do algoritmo, fazendo assim com que o tempo de execução diminua à medida que os indivíduos são ordenados pela distância de seus índices e, consequentemente, menos alterações serão feitas nos índices das tabelas a cada geração.

2. Fundamentação teórica

2.1. Algoritmo Genético

O algoritmo genético é uma técnica de busca inspirada na teoria da evolução natural de Charles Darwin. Esta técnica é muito utilizada na computação para encontrar soluções aproximadas em problemas de busca e de otimização. Este algoritmo simula o processo de seleção natural, onde os indivíduos mais aptos de uma população são selecionados para reprodução a fim de produzir descendentes para a próxima geração.

O processo do algoritmo genético começa com a seleção dos indivíduos mais aptos de uma população, o cálculo da aptidão de um indivíduo é uma função definida de acordo com o objetivo do problema. Eles produzem descendentes que herdam as características dos pais e serão adicionados à próxima geração. Este processo continua até se obter a solução desejada, o critério de parada seja atendido ou em caso de convergência (onde não há melhora na aptidão dos indivíduos em um número n de gerações). No final da execução temos como resultado uma geração com os indivíduos mais aptos.

2.2. Distância de Levenshtein

Hoje em dia, quando se navega em um dispositivo, seja ele computador ou celular, o software utiliza diversas técnicas de processamento de texto. O corretor ortográfico de um celular, por exemplo, verifica as palavras sendo digitadas para indicar se estão corretas e sugere as melhores opções. Para isso, métodos que trabalham com

processamento e manipulação de texto são usados, um exemplo é o método da distância de Levenshtein.

O método foi criado pelo cientista russo Vladimir Levenshtein com o objetivo de calcular o número mínimo de operações necessárias para transformar uma palavra em outra, quanto mais operações maior a diferença. (Gilleland & Merriam Park Software, 2006) Entende-se como operações a inserção, remoção e substituição de um caractere.

Para o cálculo da distância de substituição, podem ser especificados pesos para a troca de cada caractere para outro específico, ou de forma genérica, um peso fixo para qualquer troca entre dois caracteres diferentes.

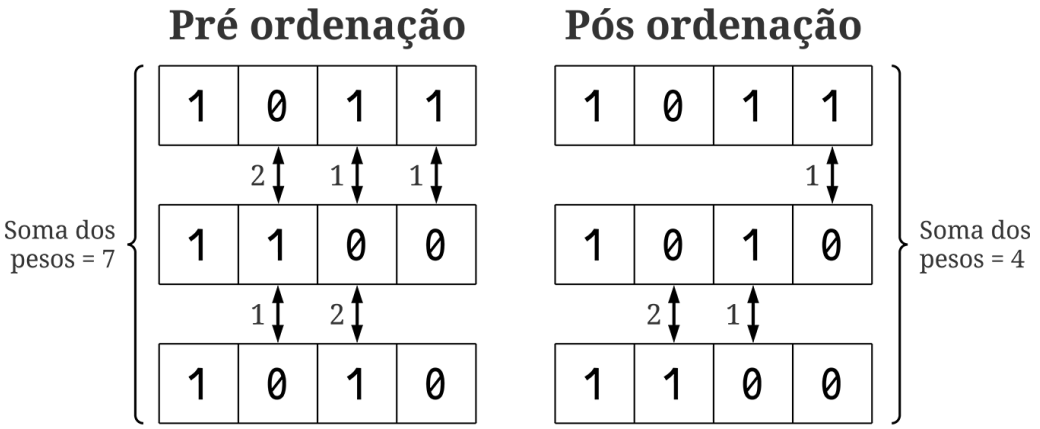

Figura 1. Ordenação dos indivíduos usando Levenshtein.
Fonte: Elaborada pelos autores.

Conforme a Figura 1, o problema abordado tem palavras de tamanho fixo e com apenas duas opções: ‘1’ ou ‘0’. Esse é um uso bastante simples do método de Levenshtein, onde foram definidos pesos para a troca entre esses dois valores, sendo peso 1 para trocas que vão do ‘1’ para o ‘0’ (remoção de índice) e peso 2 para trocas de ‘0’ para ‘1’ (criação de índice).

Para fazer o cálculo da distância entre duas palavras nessa codificação é feito o somatório dos pesos, onde os pesos são essas trocas a cada letra das palavras. Tendo esse dado pode-se iniciar a ordenação de uma série de palavras usando esses pesos.

Para fazer a ordenação em função dos pesos, é necessária uma palavra inicial e, a partir desta, decide-se a palavra seguinte conferindo a distância entre ela com as demais. Repete-se o processo até que a lista de palavras esteja ordenada.

3. Desenvolvimento

Para o desenvolvimento da contribuição, foi estudado o código da pesquisa reproduzida com a finalidade de destacar os pontos onde os indivíduos eram avaliados. Ao encontrar esses pontos, deu-se início a busca por bibliotecas que lidam com o distanciamento de Levenshtein. A biblioteca *strsimpy* foi a que melhor atendia aos requisitos para a realização do cálculo de diferença entre indivíduos.

No início da etapa de testes, percebeu-se que havia um erro na geração de novos indivíduos após a seleção por torneio, fazendo com que ficassem menos indivíduos na população parcial. O motivo é o fato da seleção por torneio apenas entregar um número *n* de indivíduos e após isso não preencher a população com os da geração anterior.

Como o tamanho do torneio estabelecido no artigo reproduzido era *5*, apenas *8* indivíduos eram selecionados e os métodos de *offspring* (crossover e mutação) aconteciam apenas nesses 8 indivíduos. Após os métodos de *offspring*, os indivíduos restantes da população eram preenchidos. Esse erro foi despercebido majoritariamente por não ter exercido uma influência considerável nos resultados, mas diminuiu a quantidade de cruzamentos e mutações que poderiam ocorrer na população. Após a constatação do erro, foi feita a correção e iniciado o processo de desenvolvimento do cálculo das distâncias de Levenshtein.

O uso do método de Levenshtein se deu após a execução dos métodos de seleção do algoritmo genético, antes de cada avaliação. A população foi ordenada usando como parâmetro inicial o último estado do banco de dados que é salvo nos atributos do código que faz a conexão. A partir deste, é coletado o indivíduo mais próximo e, a cada seleção que segue, é coletado o indivíduo mais próximo do anterior. Os pesos definidos foram apenas para substituição, pois o tamanho do indivíduo não varia, escolhemos o peso *2* para operações que criam um índice e peso *1* para operações que desfazem um índice. Essa definição de pesos se deve ao fato de que é mais demorado criar um índice do que excluí-lo.

A ordenação leva em consideração apenas as tabelas com maior volume de dados do esquema, o motivo é que o tempo de criação de índices depende diretamente da quantidade de dados na tabela, fazendo com que colunas que têm grande volume de dados tenham maior impacto na otimização. As consultas do volume das tabelas no esquema foram feitas de forma manual e as colunas a serem consideradas no algoritmo de Levenshtein foram fixadas no código.

4. Resultados

A coleta dos resultados foi iniciada com execuções do algoritmo genético com e sem a otimização de Levenshtein em uma base TPC-H de 100MB. Após algumas execuções, foram comparadas as durações de gerações do algoritmo genético sem muitas melhorias. Com a execução dos algoritmos em uma base TPC-H de 500MB e uma melhoria no cálculo do tempo, onde passou a ser coletado apenas o tempo construindo e apagando índices nas tabelas, foi construído um gráfico com esse tempo a cada geração. (Fig. 2)

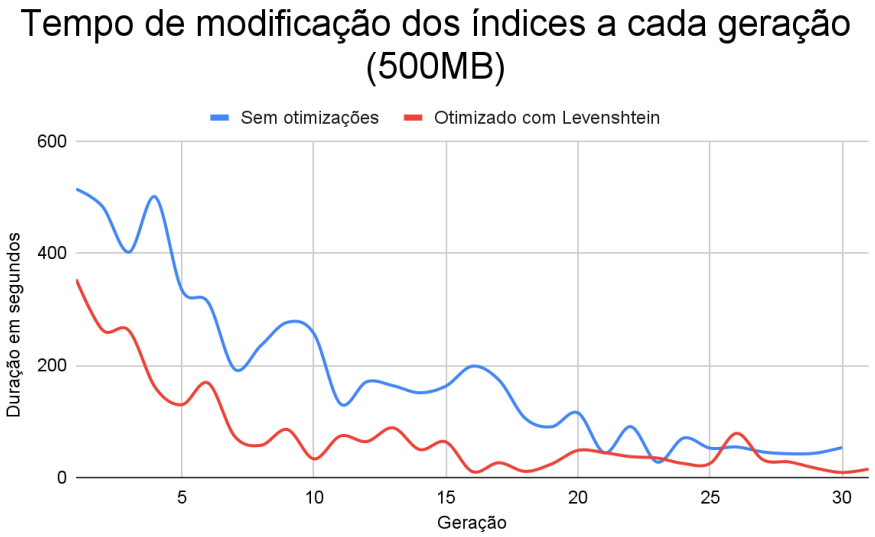

Figura 2. Gráfico do tempo levado para modificar os índices em cada geração. Fonte: Elaborada pelos autores.

Analisando o gráfico do tempo de modificação de índices por geração (Fig. 2), é notável a melhora na performance do banco de dados, principalmente nas primeiras gerações. A parte mais ao final do gráfico mostra a população começando a convergir, fazendo com que não haja muitas mudanças entre os indivíduos e o algoritmo gaste menos tempo modificando os índices, o que pode explicar a diminuição na diferença de tempo. Ainda assim, é uma diferença bem significativa, tendo em vista que o gráfico é em segundos.

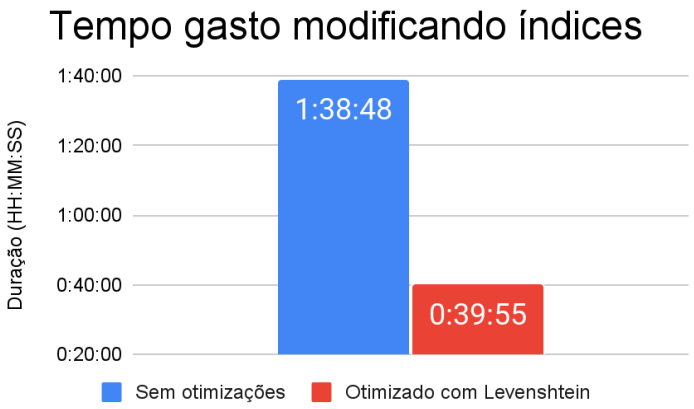

Figura 3. Gráfico do tempo total levado para modificar os índices. Fonte: Elaborada pelos autores.

No gráfico do tempo total gasto modificando os índices (Fig. 3), fica ainda mais evidente a diferença entre o tempo das duas abordagens, onde a execução sem a otimização de Levenshtein demora quase uma hora a mais modificando os índices, o que é um grande impacto considerando a execução completa do algoritmo genético.

5. Conclusão

A indexação de banco de dados é uma otimização bastante eficaz e com alta complexidade de decisão, tornando-se um problema muito interessante de ser abordado com técnicas de Inteligência Artificial, como Algoritmos Genéticos ou Machine Learning, com objetivo de busca de soluções o mais otimizadas possíveis, ou dentro de critérios de aceitação.

A contribuição realizada conseguiu diminuir em 60% o tempo de modificação de índices, comparado com a versão anterior à introdução da ordenação por distância. Levando em consideração que a otimização foi aplicada no SGBD sem desativar as otimizações já feitas por ele, e em uma amostra de dados menor, a utilização do algoritmo de distância de Levenshtein diminuir em quase uma hora o tempo do algoritmo indica que nas aplicações reais onde se têm bases de dados de vários GBs, a otimização pode acabar sendo algo indispensável.

6. Trabalhos Relacionados

Nos últimos anos, muitas pesquisas buscaram explorar maneiras de otimizar uma base de dados com as mais diversas estratégias, que vão do uso de técnicas mais simples, até o uso de IA com implementação de algoritmos genéticos e machine learning. Abaixo algumas pesquisas que exploram o mesmo campo de pesquisa da reprodução realizada.

6.1. SmartIX: A database indexing agent based on reinforcement learning

Esta pesquisa traz uma proposta semelhante à pesquisa explorada neste artigo, porém utilizando uma técnica diferente para a seleção dos índices. Ao invés de usar um algoritmo genético para a seleção, a pesquisa cria o SmartIX, que é uma arquitetura com objetivo de resolver os problemas de seleção de índices de forma automática usando um outro conceito de IA, que é o aprendizado por reforço. (Paludo Licks et al., 2020)

6.2. Genetic Algorithm for Database Indexing

Esta pesquisa possui a mesma proposta da pesquisa reproduzida e usa a mesma técnica, que é o uso de algoritmo genético para seleção de índices, porém com uma estratégia diferente. Nesta pesquisa, o esquema de dados e consultas foi implementado pelos próprios autores, diferente da pesquisa reproduzida que usa o esquema de dados TPC-H. Nesta abordagem temos outras diferenças, pois o gene corresponde ao número de colunas, o indivíduo corresponde aos índices criados em uma única tabela, e sua função de avaliação se baseia não somente no tempo de resposta das operações de consultas a uma base de dados, mas também no tempo de inserção. (Korytkowski et al., 2004)

7. Trabalhos Futuros

Uma melhoria futura a ser aplicada é a criação de um esquema de dados e consultas específicos, que simulem a base de dados onde o GADIS será aplicado, pois atualmente a pesquisa usa o esquema de dados benchmark TPC-H para a replicação da pesquisa.

Outra possibilidade é uma melhor generalização na decisão dos índices a serem usados na ordenação por Levenshtein e seus respectivos pesos. Atualmente a melhoria feita está codificada usando as tabelas que contém mais dados no esquema todo, porém essa

consulta foi feita de forma manual, sendo assim as colunas usadas no método de Levenshtein estão fixadas no código.

É também necessário realizar um estudo sobre o SGBD utilizado, pois o mesmo realiza algumas otimizações que podem incluir guardar índices que são constantemente criados e apagados em cache. Caso o SGBD esteja fazendo esse "backup", existe a possibilidade de que esses dados estejam influenciando não só no tempo criando e apagando índices, mas também no algoritmo genético em si, pois os indivíduos são testados sob uma configuração de índices ativos e inativos e caso o SGBD use índices que não devia, o resultado é inadequado.

8. Referências

Andrade, E., Herédia, M., & Rosa, M. (2021). *Reprodução de Pesquisa*. GitHub. https://github.com/tecnicasilegais/Integradora2

Fortin, F.-A., De Rainville, F.-M., Gardner, M.-A., Parizeau, M., & Gagné, C. (2012, jul). Evolutionary Algorithms Made Easy. *Journal of Machine Learning Research*, *13*, 2171-2175. https://github.com/DEAP/deap

Gilleland, M., & Merriam Park Software. (2006). *Levenshtein Distance, in Three Flavors*. http://people.cs.pitt.edu/~kirk/cs1501/Pruhs/Spring2006/assignments/editdistance/Levenshtein%20Distance.htm

Korytkowski, M., Gabryel, M., Nowicki, R., & Scherer, R. (2004). Genetic Algorithm for Database Indexing. *Artificial Intelligence and Soft Computing - ICAISC 2004*, *3070*(ICAISC 2004), 1142--1147. https://doi.org/10.1007/978-3-540-24844-6_179

Linden, R. (2006). *Algoritmos Genéticos* (3rd ed.). Ciência Moderna. https://www.algoritmosgeneticos.com.br/

Miehe, P. (2019). Indexação Autônoma de Dados Utilizando Algoritmos Genéticos.

Neuhaus, P., Couto, J., Wehrmann, J., Ruiz, D., & Meneguzzi, F. (2019). GADIS: A Genetic Algorithm for Database Index Selection. *The 31st International Conference on Software Engineering and Knowledge Engineering*.

Paludo Licks, G., Colleoni Couto, J., de Fátima Miehe, P., de Paris, R., Dubugras Ruiz, D., & Meneguzzi, F. (2020, 08 01). SmartIX: A database indexing agent based on reinforcement learning. *Applied Intelligence*. https://doi.org/10.1007/s10489-020-01674-8

Ribeiro, C. (2016). *TPC-H Benchmark, specific for MYSQL*. GitHub. https://github.com/catarinaribeir0/queries-tpch-dbgen-mysql