Particionamento Dinâmico Espaço-Temporal para Séries de Dados Matriciais

Geovani P. Santos[1], **Daniel S. Kaster**[1]

[1] Departamento de Computação — Universidade Estadual de Londrina (UEL)
Caixa Postal 10.011 – 86.057-970 – Londrina – PR – Brasil

{geovani.pereira,dskaster}@uel.br

Abstract. *Spatio-temporal matrix data series typically involve huge volumes, making partitioning essential for efficient querying. Current methods use static partitioning either by space or by time. This paper introduces a dynamic partitioning system that splits data spatially and groups it temporally based on query context over regions of interest. It maintains fixed-size spatio-temporal blocks to minimize secondary-storage block accesses. Experiments on geospatial datasets show up to a 75% reduction in query time, demonstrating the approach's effectiveness.*

Resumo. *Séries temporais de dados matriciais normalmente demandam um grande volume de dados, tornando o particionamento crucial para consultas eficientes. As abordagens atuais particionam os dados de forma estática, no espaço ou no tempo. Este trabalho propõe um sistema de particionamento dinâmico que realiza a divisão dos dados no espaço e agrupamento no tempo conforme o contexto das consultas sobre regiões de interesse, gerenciando o particionamento espaço-temporal em blocos de mesmo tamanho para reduzir os acessos a blocos em memória secundária. Experimentos em conjuntos de dados geoespaciais indicam redução de até 75% no tempo de consulta, evidenciando o potencial da proposta.*

1. Introdução

A constante evolução das tecnologias de registro de sensoriamento remoto resulta em um aumento considerável no volume de dados matriciais gerados e consumidos, o que permite a realização de análises cada vez mais diversificadas e precisas [Zhao et al. 2022, yun Luo et al. 2023]. Estes tipos de dados são persistidos como séries temporais matriciais[1], por serem registros bidimensionais de vários espectros eletromagnéticos ao decorrer do tempo [yun Luo et al. 2023]. Dados *raster* geoespaciais são uma categoria particular deste tipo de dados, pois são caracterizados pelo armazenamento e análise de informações que abrangem extensas regiões e incluem componentes temporais [Gonçalves et al. 2009, do Valle Goncalves et al. 2017, Khaki et al. 2021, Sakamoto 2020, Kemmer et al. 2023].

Os conjuntos de dados utilizados são volumosos e cobrem extensas áreas, tipicamente muito maiores do que as regiões de interesse (ROIs, do inglês *Regions of*

[1]Matematicamente, uma matriz é um objeto bidimensional. Contudo, é comum que tensores de ordem superior sejam denominados "matrizes N-dimensionais", o que pode gerar ambiguidades. Para evitar conflitos, é adotada a convenção matemática: o termo "matriz" se restringe a objetos bidimensionais, enquanto para dimensões superiores é utilizado o termo "tensor N-dimensional".

Interest) envolvidas em operações espaço-temporais, normalmente aceleradas pelo uso de índices [Tian et al. 2022]. Assim, o particionamento dos dados no espaço (*tiling* ou *chunking*) é uma estratégia muito adotada para ajustar o armazenamento dos dados à demanda das consultas. Porém, as estratégias de particionamento descritas na literatura trabalham de forma estática apenas no espaço, considerando isoladamente cada instante temporal de uma série, ou de forma localizada no espaço-tempo, fixando uma região espacial e agrupando suas instâncias no tempo. As estratégias de particionamento sobre um instante temporal [Vu and Eldawy 2020, Vu et al. 2021, Hojati et al. 2024] tratam como o dado vai ser dividido independente dos demais dados da série temporal. Podem existir casos onde métodos *in situ* [Zalipynis 2018] são aplicados, possibilitando que os dados matriciais possam ter configurações de particionamento estáticos diferentes em uma mesma série temporal. Contudo, esta estratégia não é dinâmica, pois depende de como o arquivo foi codificado e o método de particionamento só pode ser alterado se todo o dado for codificado novamente. Já estratégias que consideram o espaço-tempo [Hu et al. 2020, GDAL/OGR 2024], tratam uma região em um período pré-fixado como um bloco, ou seja, um tensor tridimensional. Embora tais abordagens sejam eficientes para executar operações sobre a região e o intervalo escolhidos, são limitadas a este caso e penalizam operações realizadas em intervalos ou regiões diferentes.

Diante deste cenário, o presente trabalho propõe uma nova abordagem para particionamento de séries temporais de dados matriciais de forma dinâmica. A abordagem particiona os dados e indexa de forma a possibilitar reorganizações espaço-temporais dos blocos sob demanda, visando otimizar o tempo de consulta. Estas alterações ocorrem conforme o contexto das consultas, podendo modificar blocos para armazenar informações de mais de um período temporal, reduzindo o acesso a disco independente da configuração dos blocos de regiões adjacentes. A abordagem proposta é materializada em um método de acesso que particiona os dados de forma dinâmica, implementando uma estratégia de indexação em três níveis. O primeiro nível é para busca temporal dos dados por meio de uma B^+-tree. O segundo nível é responsável pela indexação espacial dos dados matriciais, com uso de uma estrutura KD-tree com restrições específicas para esta proposta para cada instante de tempo. Por fim, o terceiro nível é uma indexação *in loco*, responsável pela localização do período temporal correto de cada bloco.

A proposta foi avaliada sobre um conjunto de dados composto por uma série temporal de dados geoespaciais, utilizando consultas de janelas aleatórias e de janelas deslizantes e otimização espaço-temporal em níveis crescentes de particionamento. Os resultados mostram que a proposta reduz drasticamente o custo das operações, chegando a consumir apenas 25% do tempo da consulta sobre o particionamento inicial, ressaltando o seu potencial de avanço do estado da arte.

2. Trabalhos Relacionados

O particionamento de dados é essencial para a eficiência em sistemas de análise de séries de dados matriciais, especialmente diante dos desafios impostos pelo grande volume de informações. Estratégias que envolvem a divisão dos dados em blocos podem melhorar significativamente o desempenho em aplicações, pois permitem recuperar somente as partições que intersectam a área de interesse. Exemplos de usos incluem aprendizagem de máquina [Villarroya and Baumann 2023], identificação de padrões e monitoramento de regiões [Kemmer et al. 2023]. No entanto, como os blocos são recuperados por com-

pleto, pode ocorrer a recuperação de dados que não são necessários para o resultado. A redução do tamanho dos blocos pode ser uma solução até certo ponto, porém é necessário manter equilíbrio no tamanho do bloco para maximizar o tempo de leitura dos dados de uma região de interesse, tornando insuficiente a simples redução do seu tamanho. Assim, a reorganização estratégica dos blocos é fundamental para minimizar o desperdício e otimizar o tempo de consulta.

Na literatura, são descritos métodos para particionamento de séries de dados matriciais, como os dados *raster*, destacando duas vertentes: (i) o particionamento espacial [Vu and Eldawy 2020, Vu et al. 2021, Hojati et al. 2024, Baumann et al. 1998, PostGIS 2024, Zalipynis 2018] e (ii) espaço-temporal [Hu et al. 2020]. No particionamento espacial, cada matriz de uma série temporal (instante temporal) é particionada de forma independente. Ou seja, são métodos de particionamento para instantes espaciais, pois os dados particionados não têm relação direta com os dados temporais adjacentes na série temporal. Por exemplo, a R*-Grove [Vu and Eldawy 2020] é apresentada como uma estratégia de particionamento para instantes espaciais baseada na R-tree, restringindo as divisões com dimensões voltadas para a redução de fragmentação. Deste modo, reduz-se o número de recuperações de blocos da memória secundária, promovendo consultas mais eficientes. De forma semelhante, a DSTree [Hojati et al. 2024], que é uma estrutura para consultas espaço-temporais que combina uma árvore de intervalos temporais com uma Quad-tree para cada instante de tempo, efetua o particionamento para instantes espaciais. A primeira camada gerencia intervalos temporais e a segunda indexa dados espaciais, mas os blocos armazenam dados de um único instante de tempo.

Já o particionamento espaço-temporal usa tensores N-dimensionais, ou seja, em um único bloco vão existir dados de uma região em um período temporal pré-fixado. Por exemplo, o trabalho de [Hu et al. 2020] divide os dados em subconjuntos de tensores N-dimensionais e os armazena em memória secundária seguindo um particionamento uniforme. O estudo aproveita as propriedades da arquitetura Hadoop, que mantém grandes blocos (e.g. 64 MB/128 MB), assim conseguindo armazenar regiões em longos períodos de tempo em um mesmo bloco. Deste modo, permite melhorar consultas específicas, mas ao custo de reduzir a eficiência de outras, como uma consulta que exige apenas um instante temporal desses blocos é forçada a recuperar todo o bloco. Nota-se, portanto, que são necessárias estratégias que façam um melhor balanço do tamanho do bloco e o consequente agrupamento de dados de instantes subsequentes.

Além desses métodos, existe a abordagem *in situ* que pode trabalhar com as duas vertentes, adotando uma delas dependendo do tipo de arquivo e codificação, mas de forma estática. Isto é, para modificar o método de particionamento, é preciso recodificar o dado completamente. Por exemplo, a biblioteca `GDAL` [GDAL/OGR 2024] pode trabalhar com arquivos no formato `TIF`, `JP2`, `NetCDF` e `HDF5`. Nos dois primeiros tipos, cada instante temporal é processado isoladamente. Nos formatos `NetCDF` e `HDF5`, os arquivos podem ser codificados para trabalharem usando a metodologia espaço-temporal. De forma similar, o `ChronosDB` [Zalipynis 2018] também opera *in-situ*, abstraindo formatos via tensores N-dimensionais e delegando processamento à biblioteca `GDAL`. Embora o `ChronosDB` possa trabalhar com o particionamento espaço-temporal oferecido pela `GDAL`, seu foco é o particionamento de instantes temporais. Esta abordagem particiona os dados adicionando uma margem com informações provenientes dos particionamentos

vizinhos, criando uma margem de sobreposição. Assim, ele visa reduzir o número de acessos ao disco quando somente poucos dados de interesse se encontram na margem de sobreposição.

Percebe-se, portanto, que há uma carência de abordagens de particionamento espaço-temporal que sejam adaptáveis a diferentes tipos de contextos de consultas. O armazenamento dos dados demanda flexibilidade para otimizar consultas espaço-temporais. Assim, este trabalho propõe uma abordagem dinâmica, oferecendo flexibilidade para operações espaciais e espaço-temporais e adaptabilidade dinâmica conforme a evolução temporal dos dados, mitigando desperdícios e otimizando a recuperação.

3. *Dynamic Matrix Partitioning System*

Esta seção descreve a abordagem Dympas (***Dy**namic **M**atrix **Pa**rtitioning **S**ystem*), que é uma nova proposta de particionamento de dados matriciais visando minimizar o tempo de recuperação de dados de ROIs em um dado intervalo de tempo. A proposta organiza o particionamento de séries de dados matriciais em blocos de tamanho fixo simultaneamente no espaço e no tempo, a fim de equilibrar o tempo de recuperação dos dados e a redução do desperdício conforme a demanda de consultas espaço-temporais. Isso é alcançado por meio da reorganização dos dados de forma dinâmica com as operações de *split* e *unsplit*, o que a torna uma abordagem flexível, superando limitações dos métodos de particionamento estáticos identificados na literatura. Neste contexto, ”dinâmico”refere-se à capacidade de alternar o tipo de particionamento (espacial ou espaço-temporal) de uma região, independentemente das adjacentes, sem implicar a automatização do processo.

3.1. Operações de *Split* e *Unsplit*

Dado um limiar configurável que determina o tamanho da região que corresponde a um bloco, a Dympas insere cada matriz da série temporal individualmente, fazendo *splits* sucessivos no espaço até alcançar o limiar. Adota-se a estratégia de particionamento no estilo da KD-tree, dividindo o espaço ao meio recursivamente, alternando divisões horizontais e verticais no espaço. Para cada região, aplica-se um algoritmo de compressão de dados, armazenando-se em um bloco de dados a versão comprimida da região.

O limiar não poder ser muito pequeno para poder tirar proveito da largura de banda na transferência de blocos da memória secundária. Assim, operações espaço-temporais frequentemente consideram ROIs menores do que a região de um bloco em dados matriciais de um conjunto de instantes de tempo subsequentes, demandando acessar, ao menos, um bloco por instante de tempo. Para otimizar as operações nesses casos, a proposta subdivide espacialmente a região de um número de instantes de tempo subsequentes ($2^{\text{nível}}$) e reorganiza os dados para acomodar dados de cada sub-região dos instantes em um único bloco. Assim, o *split* espaço-temporal consiste em operação de divisão de área e aumento temporal de regiões persistidas em um bloco. Inversamente, o *unsplit* espaço-temporal consiste em aumento de região e redução temporal de regiões persistidas em um bloco.

A Figura 1 ilustra o processo de *split*, onde, inicialmente, os dados de uma mesma região em instantes subsequentes T_1 e T_2 são armazenados cada um em um bloco (N e M, respectivamente). Ao detectar-se uma oportunidade de otimização espaço-temporal, o *split* divide a região em duas partes, A e B, reorganiza os blocos para armazenar os dados de A nos instantes T_1 e T_2 em um mesmo bloco e faz o mesmo para B. Desta maneira,

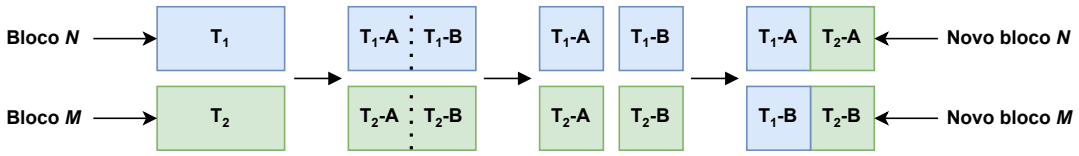

Figura 1. Funcionamento do *split* espaço-temporal em blocos de dados.

uma consulta com um ROI circunscrito em uma das sub-regiões, por exemplo, em A, que utilize os dois instantes de tempo, passa a acessar apenas um bloco (novo bloco N).

Este processo é repetido recursivamente conforme a demanda da carga de trabalho, permitindo agrupar em blocos dados de vários instantes de tempo de regiões de área reduzida, mas que ainda cubram as ROIs das operações espaço-temporais executadas. A organização parte do nível de particionamento inicial, o nível 0, contém blocos com área na norma do limiar e com somente um instante temporal. Quando um *split* é aplicado, é necessário aplicar a operação a $2^{\text{nível}}$ instantes temporais consecutivos cronologicamente. Toda a série deve estar em um nível k de particionamento, para ser possível gerar um novo nível de particionamento. Já a operação de *unsplit* reverte um *split*. Dado um estado onde existe um *split* nível X, onde $X \geq 1$, em uma região A. Ao realizar o *unsplit* sobre A, ele retorna para o estado anterior com nível $X - 1$.

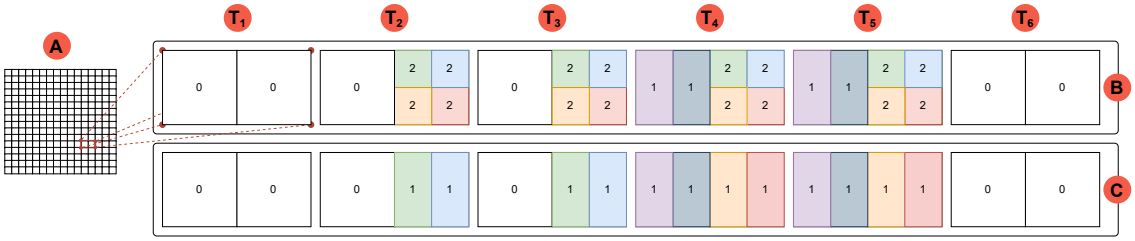

Figura 2. Funcionamento do particionamento proposto em diferentes níveis.

A Figura 2 apresenta um exemplo de particionamento espaço-temporal com aplicação dos operadores *split* e *unsplit*. A Figura 2A ilustra o dado matricial em um instante de tempo. Na Figura 2B, são apresentados os blocos correspondentes a duas regiões adjacentes da matriz, mas considerando um intervalo de instantes de tempo $[T_1, T_6]$. Os blocos indicam que foi aplicado um *split* de nível 1 em $[T_4, T_5]$ (regiões demarcadas com 1 na figura) e outro *split* de nível 2, no intervalo $[T_2, T_5]$ (regiões demarcadas com 2). Os blocos brancos, representam o particionamento de nível 0, enquanto os particionamentos de mesma cor representam dados que estão em um mesmo bloco. Para alcançar o particionamento de nível 2, foram realizados três *splits* em sequência, sendo dois de nível 1, em $[T_2, T_3]$ e $[T_4, T_5]$, seguidos de um de nível 2, no intervalo $[T_2, T_5]$.

Caso a carga de trabalho indique a necessidade de um *unsplit*, i.e., as consultas estão utilizando dados de um número de instantes de tempo significativamente menor do que o número de instantes agrupados em um bloco, causando acessos a dados de períodos não utilizados, o *unsplit* é invocado, retrocedendo o particionamento ao nível acima. E.g., um *unsplit* sobre o particionamento de nível 2 no intervalo $[T2, T5]$ no exemplo, resulta em particionamentos de nível 1, em $[T_2, T_3]$ e $[T_4, T_5]$, conforme ilustrado na Figura 2C.

3.2. Estrutura

A Dympas é composta por uma estrutura hierárquica de 3 níveis, conforme a Figura 3:

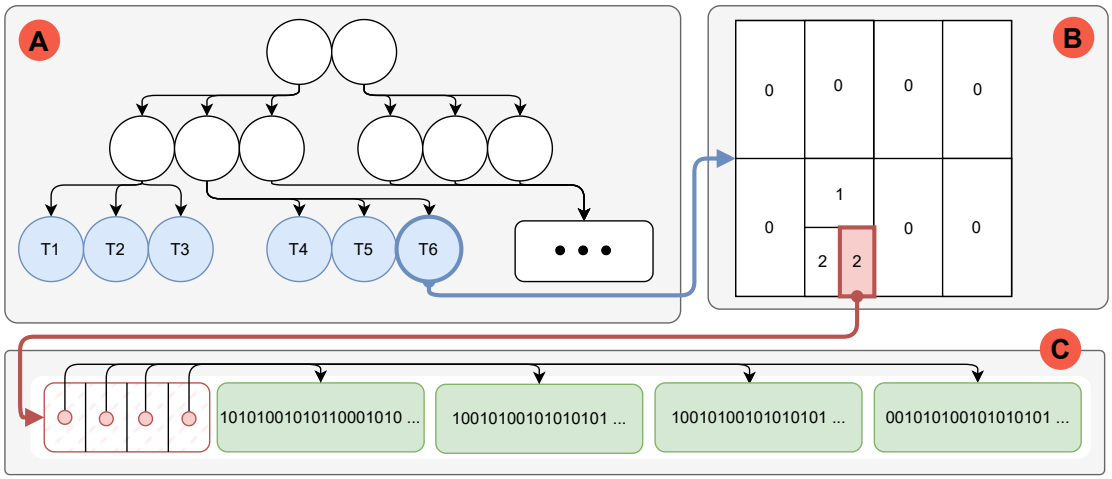

Figura 3. Estrutura da abordagem proposta.

1. B^+-tree (Figura 3A) para indexação temporal de cada instante temporal das matrizes inseridas na estrutura, onde cada nó folha tem um *timestamp* T_i;
2. KD-tree (Figura 3B), para cada nó folha da B^+-tree existe uma KD-tree correspondente ao particionamento atual referente à matriz no instante temporal T_i, onde cada nó tem um UID (*unique identifier*) referente a um bloco persistido na memória secundária com os dados referentes àquela região;
3. Hash Index (Figura 3C), como um índice *in loco* no bloco, onde, para cada UID, existe um bloco que tem um ou mais instantes temporais.

A escolha da B^+-tree é devida à sua eficiência para buscar dados ordenáveis, como instantes temporais, bastando percorrer a árvore somente uma vez até um nó folha em tempo $\mathcal{O}(\log N_1)$ [de Berg et al. 2008], onde N_1 é a altura da árvore, e percorrer os nós folhas até a busca cobrir todo o intervalo. Assim, para localizar um período temporal de tamanho K_1 tem-se um tempo de busca de $\mathcal{O}(\log N_1 + K_1)$ [Knuth 1998].

O uso da estrutura KD-tree justifica-se por sua capacidade de particionar o espaço bidimensional com um padrão rigorosamente definido, onde a orientação de cada divisão se inverte na subsequente. A versão aplicada impõe ainda uma restrição adicional, onde cada divisão é realizada precisamente na metade do tamanho da dimensão correspondente. Essa abordagem resulta em uma complexidade de busca dada por $\mathcal{O}(N_2^{1-\frac{1}{d}} + K_2)$ [de Berg et al. 2008], na qual d representa a dimensionalidade (neste caso, 2), N_2 indica a altura da árvore (ou número de particionamentos) e K_2 corresponde à quantidade de regiões que intersectam a área de busca.

A terceira camada de indexação utiliza *hash in loco* (Figura 3C). Cada bloco reúne metadados para o mapeamento *hash* e dados binários de uma região da KD-tree, abrangendo um ou mais instantes temporais. Ao efetuar o *hash* de um *timestamp*, identifica-se o trecho de dados correspondente ao ROI e ao *timestamp*. Se esse bloco contiver vários instantes, ele tende a ser reusado e, por isso, deve ser mantido em memória principal. Se o

bloco estiver na memória secundária, a busca tem complexidade $\mathcal{O}(B)$ (B representa o tamanho do bloco); se estiver em memória principal, a busca ocorre em $\mathcal{O}(1)$ [Knuth 1998].

Com isso, a complexidade total da estrutura para busca é $\mathcal{O}\left((\log N_1 + K_1) + K_1 \cdot [(N_2^{1-\frac{1}{d}} + K_2) + (K_2 \cdot B)]\right)$, pois, para cada um dos K_2 nós folha da KD-tree que intersectam com a ROI, é feito um acesso a bloco via *hash index in loco*, e o intervalo de tempo da consulta espaço-temporal requer o acesso a cada uma das K_1 KD-trees retornadas pela busca temporal na B^+-tree. Ressalte-se que a implementação da Dympas reduz o custo médio significativamente com o uso de um mecanismo de *cache* de blocos. A redução do custo é particularmente relevante para blocos que armazenam dados de diversos instantes de tempo, demandando um único acesso à memória secundária para todos os instantes agrupados no bloco, tornando o termo $\mathcal{O}(K_2 \cdot B)$ desprezível nesses casos. Isso porque apenas o primeiro instante acessa a memória secundária e os demais têm o retorno do bloco diretamente da *cache*.

3.3. Operações

Algoritmo 1: Algoritmo de busca espaço-temporal.

```
Input: ROI, start, end
Output: results
results ← [];
times ← B+Tree.findPeriod(start, end);
foreach T_i ∈ times do
    spaceIndex ← getKDTree(T_i);
    UIDs ← spaceIndex.search(ROI);
    ROIFrame ← emptyMatrix(ROI);
    foreach UID ∈ UIDs do
        block ← DiskManager.load(UID); // Verifica a cache
        chunk ← block.hashIndex.get(T_i);
        ROIFrame.add(chunk);
    end
    results.append(ROIFrame, T_i);
end
return results;
```

Esta seção apresenta um algoritmo genérico para a proposta que realiza uma busca temporal inicial com uma subsequente varredura espacial, maximizando a eficiência e garantindo a precisão na recuperação dos dados (algoritmo 1). Inicialmente, o algoritmo utiliza a B^+-tree para realizar uma consulta no período especificado, identificando os *timestamps* para os quais os dados matriciais estão disponíveis. Cada *timestamp* extraído serve de chave para a seleção da respectiva KD-tree, responsável por executar a busca espacial, determinando, por meio do ROI fornecido, quais particionamentos e seus respectivos UIDs intersectam a região de interesse.

Em sequência, para cada instante de tempo, o algoritmo verifica se o bloco de dados associado ao UID já se encontra em memória principal. Se não estiver, é recuperado da memória secundária. Uma busca baseada em *hash* é realizada para localizar com exatidão a posição e o tamanho dos dados no bloco, e esses dados são inseridos na matriz de saída correspondente ao ROI. O processo se repete para todos os instantes de tempo,

aproveitando a reutilização dos blocos que contêm mais de um período temporal e já estão carregados em memória principal. Assim, é evitada a redundância e aprimora-se o desempenho geral da operação.

4. Experimentos

4.1. Carga de Trabalho

A carga de trabalho utilizada nos testes é composta por operações espaço-temporais típicas em tarefas de análise e extração de características para aprendizado de máquina. Fundamentalmente, as operações recuperam dados de uma dada região de interesse em uma série de instantes de tempo consecutivos. As operações são subdividas em dois tipos: (*I*) janela aleatória e (*II*) janela deslizante.

As operações de janela aleatória recuperam dados de um ROI fornecido de intervalos variados de instantes de tempo, representando consultas isoladas para análise espaço-temporal. Já as consultas de janela deslizante recuperam dados de um mesmo ROI em instantes subsequentes de tempo, de forma iterativa, por exemplo, para calcular estatísticas móveis, como o acumulado de precipitação dos últimos 15 dias para cada dia de um intervalo de interesse. Uma aplicação prática dessa operação é o acompanhamento meteorológico em regiões agronômicas, para orientar o manejo de patologias. O cálculo da precipitação acumulada em 15 dias consiste em uma operação de janela deslizante sobre os dados. A cada dia, a janela de 15 dias avança um instante, de modo que o cálculo para o dia 16, que opera sobre $[T_1, T_{15}]$, enquanto para o dia 17, usa o intervalo $[T_2, T_{16}]$. As consultas por janela deslizante permitem um uso agressivo da *cache* de blocos, pois a mesma ROI é examinada à medida que o intervalo temporal é varrido sequencialmente do início ao fim, com acesso a um único bloco novo da memória secundária a cada deslizamento da janela. Em contrapartida, as consultas por janela aleatória até podem beneficiar-se da *cache* de blocos, mas isso ocorre de forma menos frequente.

A carga de trabalho é baseada um conjunto de dados de sensoriamento remoto do Sentinel-2 [Drusch et al. 2012] e em geometrias de talhões agrícolas registrados na Agência Nacional de Águas (ANA), para simular operações em um ambiente real. Foi utilizada a região de código `T22KDV` do Sentinel-2, que cobre parte do norte do estado do Paraná, de maio de 2017 até agosto de 2024. Foi usada a banda espectral de 490nm com resolução espacial de 10 metros, sobre uma área de aproximadamente 110km$\times$110km. Os dados foram filtrados com a restrição de terem menos de 10% de nuvem, o que é uma decisão muito comum em situações reais, resultando em uma série temporal com 68 arquivos do tipo *raster* no intervalo. Cada *raster* tem dimensão de 10980×10980 pixeis e precisão de 2 bytes por pixel. Para as ROI, foram utilizados os dados da geometria de talhões cadastrados na ANA contidos na região `T22KDV` do Sentinel-2, totalizando 78 talhões. As ROIs foram definidas como o retângulo envolvente mínimo dos talhões (*bounding boxes*). O tamanho médio das ROIs é 90×90 pixeis, sendo que a maior ROI tem as dimensões de 141×37 pixeis e a menor 30×37 pixeis.

As operações, tanto de janela aleatória quanto deslizante, foram definidas para períodos de 7 meses. Contudo, o número de *rasters* da série contida em diferentes intervalos de 7 meses varia, pois há períodos com maior ou menor ocorrência de nuvens, variando o número de *rasters* com 10% ou mais de nuvem descartados. A quantidade de *rasters* dentro dessas janelas varia de 2 (muita ocorrência de nuvens) a 7 (sem descarte

por cobertura de nuvens), com um tamanho médio de 5,24 *rasters*. A cada nova janela, há um acréscimo de 1 mês no início, em relação à janela anterior. Assim, para cada ROI, os dados permitiram produzir 82 janelas de maio de 2017 até agosto de 2024.

Nas consultas de janela aleatória, foram geradas combinações exaustivas de ROI e período temporal da janela, resultando em 6396 consultas por bateria de teste. Já para as operações com janelas deslizantes, a consulta é realizada sobre um ROI sequencialmente sobre todas as janelas até cobrir toda a abrangência temporal dos dados, resultando em 82 consultas, uma para cada ROI.

4.2. Configuração dos Experimentos

O objetivo principal dos testes é avaliar a variação do tempo das consultas usando o método proposto em comparação ao particionamento exclusivamente espacial. Não há correlatos diretos e o particionamento exclusivamente espacial representa a categoria dominante de abordagens existentes, inclusive em implementações comerciais como o PostGIS e outros, o que justifica a escolha. Por particionamento exclusivamente espacial, entenda-se o particionamento inicial adotado na proposta desse trabalho, em que o dado matricial de cada instante de tempo é subdividido iterativamente usando a estratégia da KD-tree (particionamentos verticais e horizontais alternados) até alcançar um limiar predefinido, que estabelece o tamanho da região a ser depositada de forma comprimida em um bloco (cf. Subseção 3.1). Nos resultados, o particionamento exclusivamente espacial é denotado como *split nível 0*.

Os testes consideraram variações no tamanho de bloco, com base no limiar de particionamento, para analisar o comportamento das estratégias mediante de diferentes tamanhos de bloco. Os limiares avaliados foram 22500, 90000 e 360000, que correspondem, respectivamente, às regiões de 150×150, 300×300 e 600×600 pixeis.

A carga de trabalho foi submetida para as diferentes configurações de limiar, considerando níveis crescentes de otimização espaço-temporal. Ressalte-se que a proposta desse trabalho permite o ajuste dinâmico do particionamento espaço-temporal, contudo, neste trabalho, as operações de *split* foram invocadas manualmente e de forma sucessiva, como tarefas de otimização de bancos de dados executadas por um administrador com base em sua análise e expertise a respeito da carga de trabalho. Isto é, a primeira execução da carga de trabalho considerou o particionamento exclusivamente espacial (*split* nível 0). Em seguida, aplicou-se o *split* espaço-temporal nos blocos interceptados por ROIs para os níveis 1,2 e 3 (que agrupam, respectivamente, 2, 4 e 8 instantes de tempo), executando novamente a carga de trabalho após cada nível.

Para cada contexto, o teste foi repetido 3 vezes com a mesma configuração, totalizando 76752 consultas para medir possíveis variações de consistência e garantir que não ocorreram anomalias. Para avaliar o benefício da *cache* de blocos, foram considerados casos típicos em dois extremos: casos sem qualquer aproveitamento de *cache* e casos com ótimo aproveitamento de *cache*. Para avaliar os casos sem aproveitamento de *cache*, foram utilizadas operações de janela aleatória, limpando a *cache* ao término de cada operação. Esse caso é típico quando se deseja executar uma operação espaço-temporal para realizar uma análise isolada, seja para uma região ou um intervalo não consultados anteriormente, portanto, os blocos necessários não deverão estar em *cache*. Um exemplo de operações no outro extremo são as operações de janela deslizante, que são altamente

beneficiadas pela *cache*. Para avaliar este caso, nos experimentos com operações de janela deslizante não foi feita limpeza da *cache* de blocos. Eventuais retiradas de blocos da *cache* foram realizadas exclusivamente pelo mecanismo de liberação de *frames*, que foi implementado no Dympas utilizando a política LRU (*Least Recently Used*).

Todos os testes foram realizados em um mesmo ambiente de desenvolvimento, implementados na mesma base de código em C++ e com execução em um mesmo equipamento. O computador utilizado é equipado com processador Intel Core i7-9700, com 32GB de RAM DDR4, HDD SATA 2TB de 7.200RPM e sistema operacional Ubuntu GNU/Linux 24.04 64 bits. O ambiente foi exclusivamente utilizado para estes testes para evitar interferências no gerenciamento dos dados do sistema. Além disso, a *cache* do sistema de arquivos utilizado pelo sistema operacional foi esvaziada antes de cada execução.

5. Resultados

Esta seção apresenta os resultados obtidos conforme os parâmetros descritos na Seção 4, visando realizar uma análise quantitativa do tempo de recuperação dos dados provenientes de regiões espaço-temporais, com base na variação de configurações do método proposto. Com objetivo de contrapor a abordagem de particionamento espacial (*split* nível 0), normalmente empregada em sistemas de gerenciamento de dados de sensoriamento remoto, à de particionamento espaço-temporal dinâmico (nível 1 à 3). Os testes realizados no contexto da análise geoespacial demonstraram uma melhoria significativa do desempenho de execução com a aplicação da abordagem proposta neste trabalho, com redução de até 75% nos tempos de consulta.

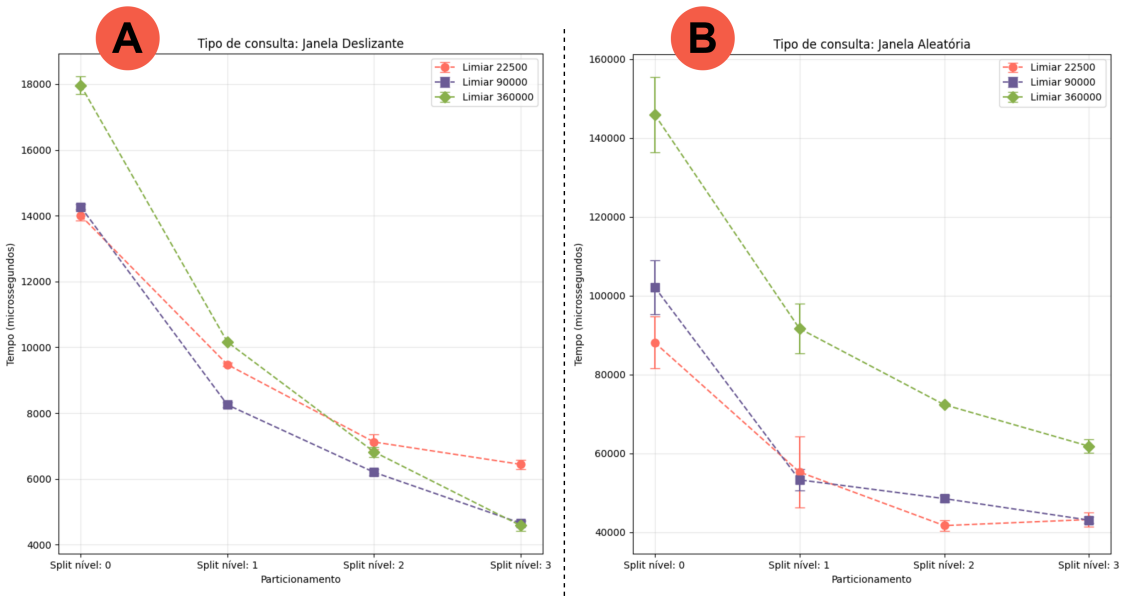

Figura 4. Média de tempo das consultas realizadas para cada nível de particionamento realizado via operador *split*.

A Figura 4 apresenta os resultados médios das três baterias de testes realizadas para cada configuração. Cada teste foi conduzido com uma configuração composta pelo nível de particionamento (realizado pelo operador *split*), um limiar e um tipo de consulta. A Figura 4A apresenta os resultados das consultas por janelas deslizantes, correspondentes a análises mais prolongadas que permitem um ótimo aproveitamento de *cache*.

Observou-se que, para níveis de *split* acima de 0, há um maior reaproveitamento, o que reduz consideravelmente os acessos à memória secundária à medida que o nível de particionamento aumenta. Dessa forma, o tempo de acesso a uma mesma região é reduzido em até 75%, enquanto o particionamento de nível 0 demandou, em média, 17,9 ms, a otimização com particionamento de nível 3 resultou em um tempo médio de apenas 4,5 ms. As consultas por janelas aleatórias, embora mais restritas temporalmente do que as realizadas com janelas deslizantes e, portanto, com menor aproveitamento de *cache*, também apresentaram resultados expressivos. A Figura 4B exibe esses resultados, nos quais se observou uma redução média de 56%, considerando todos os limiares. O melhor caso foi observado para o limiar 90000, em que a redução obtida com o particionamento de nível 3 foi de 58% em relação ao particionamento exclusivamente espacial (nível 0). Ressalte-se que o tamanho médio de janela aleatória na carga de trabalho (5,24) não é perfeitamente alinhado ao número de blocos agrupados em *splits* espaço temporais (2, 4 ou 8 instantes de tempo, respectivamente, para os níveis de *split* 1, 2 e 3), o que causa um certo desperdício em algumas consultas, o que não acontece nas consultas com janelas deslizantes, em que todos os instantes acessados são úteis. Contudo, a proposta possibilita amortizar parcialmente esse desperdício pelo uso da largura de banda do acesso à memória secundária. A Figura 4B também mostra que existe uma relação ótima entre o tamanho de bloco e o nível de *split*, visível para o limiar 22500, em que o tempo de execução do nível de *split* 3 é superior ao do nível 2.

O custo computacional da operação de *split* mostra uma correlação direta com o limiar e o nível de particionamento. Para limiares de 360000, 90000 e 22500, os tempos médios de transição entre os níveis de particionamento ($0 \rightarrow 1$, $1 \rightarrow 2$, $2 \rightarrow 3$) foram, respectivamente: 37, 49 e 54 ms; 70, 77 e 87 ms; e 162, 149 e 197 ms. Fica evidenciado que limiares menores têm um custo de processamento maior por operar mais blocos.

A Tabela 1 apresenta o compilado dos resultados médios das 3 baterias de testes realizados, com os dados para janelas deslizantes (JD) e para janelas aleatórias (JA) para os três limiares. As linhas contêm os dados, para cada nível de particionamento, do número médio de blocos recuperados da memória secundária ou o tempo médio das consultas em microssegundos.

Tabela 1. Tabela das médias dos resultados das 3 baterias de testes.

Unidade	Nível	JD 22500	JD 90000	JD 360000	JA 22500	JA 90000	JA 360000
Blocos lidos	0	1,99	1,41	1,08	12,57	8,94	6,86
Tempo	0	14004,36	14268,75	17962,02	88131,52	102201,84	145943,28
Blocos lidos	1	1,48	0,90	0,66	11,10	6,89	5,00
Tempo	1	9473,28	8250,30	10159,90	55298,23	53310,96	91680,70
Blocos lidos	2	1,16	0.60	0,42	11,05	5,72	4,00
Tempo	2	7119,26	6199,67	6819,31	41717,12	48559,90	72367,36
Blocos lidos	3	1,01	0,44	0,25	12,51	5,55	3,34
Tempo	3	6439,15	4652,75	4578,32	43247,54	43098,76	61838,48

A Figura 5A apresenta uma visualização agregada dos resultados. Nota-se, claramente, uma convergência dos melhores desempenhos em direção a níveis maiores de *split* e menores tamanhos de bloco, para a carga de trabalho avaliada. Porém, que a abordagem proposta direciona os ganhos para o melhor caso, independentemente do tamanho do bloco. Assim, é esperado que o ajuste do particionamento espaço-temporal à carga de

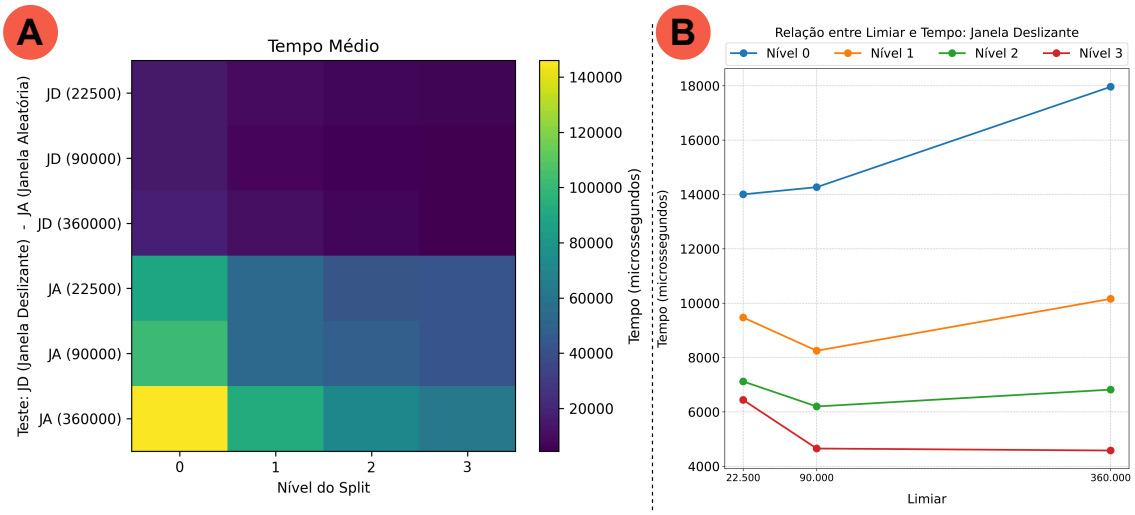

Figura 5. Distribuição dos resultado e tendências das configurações.

trabalho seja ainda maior ao usar-se níveis de *split* de forma dinâmica e não homogênea.

A Figura 5B apresenta as tendências dos testes das janelas deslizantes realizados através da relação de tempo e limiar para operações com janelas deslizantes. Para esse caso, fica evidenciado que o uso de blocos menores para o particionamento exclusivamente espacial (nível 0) é vantajoso. Porém, essa relação não se mantém ao utilizar-se o particionamento espaço-temporal, onde os melhores casos encontram-se em blocos de tamanhos intermediários. Considerando-se simultaneamente consultas de janela aleatória e de janela deslizante, o melhor caso para a carga de trabalho avaliada é a configuração de tamanho de bloco intermediário (90000) com o maior nível de *split* (nível 3). Estes resultados evidenciam a superioridade da proposta.

6. Conclusão

Este trabalho apresentou uma nova abordagem para o particionamento dinâmico de séries de dados matriciais, visando suprir uma lacuna identificada na literatura, na qual os particionamentos existentes atuam exclusivamente em instantes ou séries temporais. Para contornar essa limitação, a abordagem proposta, Dympas, realiza operações de *split* e *unsplit* espaço-temporal, permitindo particionar de forma dinâmica os dados no espaço e agrupar no tempo, organizando os dados em blocos para ajustar o particionamento à demanda de operações espaço-temporais. Resultados experimentais mostraram uma melhoria significativa no desempenho, com redução de até 75% no tempo de consulta em análises temporais sobre determinadas regiões, evidenciando o potencial da estrutura para aplicação em soluções temporais. Trabalhos futuros incluem avaliar a proposta em outras situações e com outros conjuntos de dados e desenvolver métodos para automatizar a otimização possibilitada pela Dympas, realizada neste trabalho de forma manual.

Agradecimentos

Os autores agradecem ao CNPq e à Espectro Ltda pelo apoio financeiro, por meio do projeto PreCISIA (processo n^o 424490/2021-8), com bolsa RHAe (processo n^o 350921/2022-9), e à Fundação Araucária, com bolsa vinculada ao NAPI CIA-Agro.

Referências

Baumann, P., Dehmel, A., Furtado, P., Ritsch, R., and Widmann, N. (1998). The multidimensional database system rasdaman. In *Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, pages 575–577. ACM.

de Berg, M., Cheong, O., van Kreveld, M., and Overmars, M. (2008). *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin, 3 edition.

do Valle Goncalves, R. R., Zullo, J., Romani, L. A. S., do Amaral, B. F., and Sousa, E. P. M. (2017). Agricultural monitoring using clustering techniques on satellite image time series of low spatial resolution. In *2017 9th International Workshop on the Analysis of Multitemporal Remote Sensing Images (MultiTemp)*, pages 1–4. IEEE.

Drusch, M., Del Bello, U., Carlier, S., Colin, O., Fernandez, V., Gascon, F., Hoersch, B., Isola, C., Laberinti, P., Martimort, P., Meygret, A., Spoto, F., Sy, O., Marchese, F., and Bargellini, P. (2012). Sentinel-2: Esa's optical high-resolution mission for gmes operational services. *Remote Sensing of Environment*, 120:25–36. The Sentinel Missions - New Opportunities for Science.

GDAL/OGR (2024). *GDAL/OGR Geospatial Data Abstraction software Library*. Open Source Geospatial Foundation.

Gonçalves, R., Nascimento, C., Zullo Jr, J., and Romani, L. (2009). Relationship between the spectral response of sugar cane, based on avhrr/noaa satellite images, and the climate condition, in the state of sao paulo (brazil), from 2001 to 2008. In *International Workshop on the Analysis of Multi-temporal Remote Sensing images (MultiTemp)*, volume 5, pages 315–322.

Hojati, M., Roberts, S., and Robertson, C. (2024). Dstree: A spatio-temporal indexing data structure for distributed networks. *Mathematical and Computational Applications*, 29:42. search about DHT.

Hu, F., Yang, C., Jiang, Y., Li, Y., Song, W., Duffy, D. Q., Schnase, J. L., and Lee, T. (2020). A hierarchical indexing strategy for optimizing apache spark with hdfs to efficiently query big geospatial raster data. *International Journal of Digital Earth*, 13:410–428.

Kemmer, C., Reis, L., Rodrigues-Silva, J., Scolin, L., Canteri, M., and Kaster, D. (2023). Proposta de uma plataforma para o desenvolvimento e análise visual de modelos preditivos para doenças da soja. In *Anais do XIV Congresso Brasileiro de Agroinformática*, pages 326–333, Porto Alegre, RS, Brasil. SBC.

Khaki, S., Pham, H., and Wang, L. (2021). Simultaneous corn and soybean yield prediction from remote sensing data using deep transfer learning. *Scientific Reports*, 11:11132.

Knuth, D. E. (1998). *The Art of Computer Programming, Volume 3: Sorting and Searching*. Addison-Wesley, Reading, Massachusetts, 2nd edition.

PostGIS (2024). *PostGIS*. Open Source Geospatial Foundation. PostGIS is a spatial database extender for PostgreSQL.

Sakamoto, T. (2020). Incorporating environmental variables into a modis-based crop yield estimation method for united states corn and soybeans through the use of a random

forest regression algorithm. *ISPRS Journal of Photogrammetry and Remote Sensing*, 160:208–228.

Tian, R., Zhai, H., Zhang, W., Wang, F., and Guan, Y. (2022). A survey of spatio-temporal big data indexing methods in distributed environment. *IEEE J. Sel. Top. Appl. Earth Obs. Remote. Sens.*, 15:4132–4155.

Villarroya, S. and Baumann, P. (2023). A survey on machine learning in array databases. *Applied Intelligence*, 53:9799–9822.

Vu, T. and Eldawy, A. (2020). R*-grove: Balanced spatial partitioning for large-scale datasets. *Frontiers in Big Data*, 3.

Vu, T., Eldawy, A., Hristidis, V., and Tsotras, V. (2021). Incremental partitioning for efficient spatial data analytics. In *Proceedings of the VLDB Endowment*, volume 15, pages 713–726. VLDB Endowment.

yun Luo, T., cheng Zou, B., hui Li, S., cheng Pang, S., and rui Zhang, C. (2023). High stability of opto-mechanical structure design and validation of Beijing No.3 camera. In Jiang, Y., Wang, X., Liu, D., Xue, B., Wang, Y., Cao, L., Wang, Q.-H., and Lu, C.-Y., editors, *AOPC 2023: Optical Sensing, Imaging, and Display Technology and Applications; and Biomedical Optics*, volume 12963, page 129630V. International Society for Optics and Photonics, SPIE.

Zalipynis, R. A. R. (2018). Chronosdb. *Proceedings of the VLDB Endowment*, 11:1247–1261.

Zhao, Q., Yu, L., Du, Z., Peng, D., Hao, P., Zhang, Y., and Gong, P. (2022). An overview of the applications of earth observation satellite data: Impacts and future trends. *Remote Sensing*, 14(8).