

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Pedro Nogueira Gomes Assumpção

**Avaliação de desempenho dos métodos de
acesso R-Tree e M-Tree para consultas aos
k-vizinhos mais próximos em conjuntos de
dados de alta dimensionalidade**

Uberlândia, Brasil

2018

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Pedro Nogueira Gomes Assumpção

**Avaliação de desempenho dos métodos de acesso R-Tree
e M-Tree para consultas aos k-vizinhos mais próximos em
conjuntos de dados de alta dimensionalidade**

Trabalho de conclusão de curso apresentado
à Faculdade de Computação da Universidade
Federal de Uberlândia, Minas Gerais, como
requisito exigido parcial à obtenção do grau
de Bacharel em Sistemas de Informação.

Orientador: Professor Humberto Luiz Razente

Universidade Federal de Uberlândia – UFU

Faculdade de Ciência da Computação

Bacharelado em Sistemas de Informação

Uberlândia, Brasil

2018

Agradecimentos

Agradeço a minha família e amigos por todo apoio durante essa caminhada. A empresa em que trabalho, Xtrategie e todos meus companheiros de trabalho por terem me apoiado durante a fase final de construção desse trabalho.

Agradeço ao corpo docente e administrativo da Faculdade de Computação e da Universidade Federal de Uberlândia que proporcionaram conhecimentos que definirão minha vida profissional. Em especial ao meu orientador Professor Humberto Luiz Razente pela paciência, ajuda e dedicação durante a elaboração deste trabalho.

Agradeço ao Diretório Acadêmico Sistemas de Informação e a Associação Atlética Acadêmica Computação, que em ambos tive oportunidade de ser presidente, por terem me ajudado a adquirir conhecimentos que levarei durante toda minha vida, além da oportunidade de ter contato com estudantes de vários outros cursos da instituição, proporcionaram acesso a congressos, palestras, eventos esportivos e a construção de amizades importantes que tornaram o trajeto menos penoso.

Agradeço também ao programa Ciência sem Fronteiras, bem como à CAPES que foi a agência financiadora da bolsa e a Faculdade Siena College pela oportunidade única de estudar e conhecer a cultura de outro país além das amizades construídas nesse processo.

Resumo

Armazenamento de dados é uma atividade constante no campo da computação. A utilização de métodos de acesso para realizar buscas com eficiência é primordial. Esse trabalho apresenta duas estruturas de indexação para dados complexos (R-tree e M-tree) e o algoritmo de busca por similaridade k-vizinhos mais próximos avaliando o desempenho da busca nesses dois métodos de indexação comparando também como método sequencial, para entender casos em que a utilização do método de indexação é menos eficiente do que a não utilização de um método de acesso (causado pela maldição da dimensionalidade). Os experimentos apresentaram que uma das estruturas (R-tree) teve um desempenho pior que o método sequencial para as bases testadas e que a outra estrutura (M-tree) foi mais eficiente em todas as bases.

Palavras-chave: Métodos de acesso multidimensionais. Métodos de acesso métricos. Consultas por similaridade. K-vizinhos mais próximos. Maldição da alta dimensionalidade. Avaliação de desempenho.

Lista de ilustrações

Figura 1 – Exemplo de dados MBRs e seus MBRs (MANOLOPOULOS et al., 2006, p. 8)	16
Figura 2 – Construção da R-Tree (MANOLOPOULOS et al., 2006, p. 8)	16
Figura 3 – Um objeto O_r possui um raio de cobertura $r(O_r)$ e referencia uma árvore $T(O_r)$ (CIACCIA; PATELLA; ZEZULA, 1997)	19
Figura 4 – Construção com a base <i>Eigenfaces</i> utilizando páginas de 2 KB	26
Figura 5 – Construção com a base <i>Eigenfaces</i> utilizando páginas de 4 KB	27
Figura 6 – Consultas com a base <i>Eigenfaces</i> utilizando páginas de 2 KB	28
Figura 7 – Consultas com a base <i>Eigenfaces</i> utilizando páginas de 4 KB	29
Figura 8 – Construção com a base <i>Pendigits</i> utilizando páginas de 2 KB	30
Figura 9 – Construção com a base <i>Pendigits</i> utilizando páginas de 4 KB	31
Figura 10 – Consultas com a base <i>Pendigits</i> utilizando páginas de 2 KB	32
Figura 11 – Consultas com a base <i>Pendigits</i> utilizando páginas de 4 KB	33
Figura 12 – Construção com a base <i>Corel</i> utilizando páginas de 4 KB	34
Figura 13 – Construção com a base <i>Corel</i> utilizando páginas de 8 KB	34
Figura 14 – Consultas com a base <i>Corel</i> utilizando páginas de 4 KB	36
Figura 15 – Consultas com a base <i>Corel</i> utilizando páginas de 8 KB	37
Figura 16 – Construção com a base <i>Coverttype</i> utilizando páginas de 4 KB	38
Figura 17 – Construção com a base <i>Coverttype</i> utilizando páginas de 8 KB	39
Figura 18 – Consultas com a base <i>Coverttype</i> utilizando páginas de 4 KB	40
Figura 19 – Consultas com a base <i>Coverttype</i> utilizando páginas de 8 KB	41
Figura 20 – Construção com a base <i>Colors</i> utilizando páginas de 16 KB	42
Figura 21 – Construção com a base <i>Colors</i> utilizando páginas de 32 KB	43
Figura 22 – Consultas com a base <i>Colors</i> utilizando páginas de 16 KB	44
Figura 23 – Consultas com a base <i>Colors</i> utilizando páginas de 32 KB.	45

Lista de tabelas

Tabela 1 – Bases de dados utilizadas.	25
---	----

Lista de abreviaturas e siglas

KNN	k-vizinhos mais próximos
MA	Método de Acesso
MBR	Minimum bounding rectangle
M-Tree	Árvore Métrica (Metric Tree)
R-Tree	Árvore R
SGBDR	Sistema de Gerenciamento de Bancos de Dados Relacionais

Sumário

1	INTRODUÇÃO	8
1.1	Hipótese	9
1.2	Objetivos	9
1.3	Método	10
1.4	Organização do Trabalho	10
2	FUNDAMENTAÇÃO TEÓRICA	11
2.1	Métodos de acesso	11
2.1.1	B-Tree	12
2.2	Métodos de acesso multidimensionais e métricos	14
2.2.1	R-Tree	15
2.2.2	M-Tree	18
2.3	Maldição da dimensionalidade	21
2.4	Buscas por similaridade	21
2.4.1	Busca aos k-vizinhos mais próximos	22
3	AVALIAÇÃO DE DESEMPENHO DOS MÉTODOS DE ACESSO	
	R-TREE E M-TREE	25
3.1	Eigenfaces	26
3.2	Pendigits	30
3.3	Corel	33
3.4	Coverttype	37
3.5	Colors	41
4	CONCLUSÃO	46
	REFERÊNCIAS	47

1 Introdução

Armazenamento de dados é uma atividade muito comum e não acontece somente no campo da computação. Desde agendas, contratos com clientes e documentos até mesmo bancos de dados ou armazenamento de arquivos em um computador mostram que esta atividade é constante na vida das pessoas. Dados são armazenados a fim de que possam ser encontrados quando necessário.

Para efetuar a busca de um dado, tanto seres humanos quanto sistemas computacionais tentam utilizar métodos de indexação com a intenção de otimizar a busca. Por exemplo, uma agenda de telefones tem uma guia na lateral com as letras do alfabeto. Ao anotar um número de telefone, o proprietário da agenda anota os dados do contato na página onde a letra da guia corresponde à letra inicial do nome do contato. Quando é necessário buscar o número daquele contato, o proprietário da agenda não terá que ler todos os contatos anotados na agenda, somente irá abrir a página em que a letra da guia corresponde à letra inicial do contato e procurar o contato naquela página.

Em sistemas computacionais não é diferente. Desde o final da década de 1970 o método de indexação B-Tree ([COMER, 1979](#)) (e suas variações) é o método mais utilizado quando tratamos de dados em que uma relação de ordenação pode ser definida tal como para números (inteiros ou de ponto flutuante), datas e cadeias de caracteres. Porém o método B-Tree é incapaz de indexar dados multidimensionais onde não é possível criar uma relação de ordenação, por exemplo, pontos geográficos, dados multimídia e dados espaciais ([MANOLOPOULOS et al., 2006](#), p. 3). Os métodos de indexação R-Tree ([GUTTMAN, 1984](#)) e M-Tree ([CIACCIA; PATELLA; ZEZULA, 1997](#)) foram propostos como uma extensão do método B-Tree aplicado para dados não ordenáveis e permitem executar buscas baseadas na proximidade ao invés de se basear em ordenação.

Para muitos domínios de dados, a relação de ordem tem pouca utilidade. Quando um usuário aponta para uma localização ou um objeto na tela e deseja que o sistema encontre os 5 objetos mais próximos à ele no banco de dados, a relação de similaridade ou proximidade é a desejada. Nesse caso, deseja-se que o sistema computacional utilize um algoritmo de busca dos registros mais próximos ao ponto indicado para retornar os objetos. O algoritmo de busca de k-vizinhos foi criado com esse intuito e trabalha com dados indexados com base em proximidade ([ROUSSOPOULOS; KELLEY; VINCENT, 1995](#)). Os métodos de indexação R-Tree e M-Tree se baseiam em proximidade dos objetos e o algoritmo se aplica à ambos.

Segundo ([VOLNYANSKY; PESTOV, 2009](#)), é observado que o desempenho de vários algoritmos que lidam com conjuntos de dados multidimensionais cai exponencial-

mente com o aumento de dimensões (efeito denominado maldição da dimensionalidade). Desta forma é esperado que de acordo com o aumento no número de dimensões (colunas de uma tabela relacional) para a realização de consulta aos vizinhos mais próximos, a utilização dos métodos de acesso R-Tree e M-Tree irá se degradar.

O efeito da maldição da dimensionalidade pode variar de acordo com o domínio dos dados e da função de distância empregada, assim, não há um modo formal de provar a partir de qual número de dimensões os algoritmos passam a ter degradação a ponto de não valer a pena utilizá-los. Desse modo, neste trabalho os métodos M-Tree e R-Tree serão avaliados empiricamente com conjuntos de dados de diferentes dimensionalidades, permitindo auxiliar nas tomadas de decisão que envolvem a escolha de métodos de acesso para serem utilizados nas aplicações.

1.1 Hipótese

Realizar buscas de maneira eficiente é uma necessidade geral. Por mais que os avanços tecnológicos tenham proporcionado uma grande melhora em questões de hardware, quando tratamos grandes volumes de dados, é necessário conhecer os melhores métodos de busca. Uma boa maneira de se fazer isso é possuir um método de indexação dos dados que otimize a busca.

B-Trees, R-Trees e M-Trees são métodos de indexação que tem o propósito de otimizar buscas. O método B-Tree não funciona para os tipos de dados que serão tratados neste trabalho pois não se aplica para dados não ordenáveis e busca baseada em distância. Os métodos R-Tree e M-Tree que serão tratados neste trabalho foram criados para indexação de dados multidimensionais/métricos onde a noção de espaço ou similaridade são empregados.

Para otimizar uma busca através de uma indexação é necessário que se saiba qual tipo de dado está indexado e qual tipo de busca está sendo realizada. Este trabalho focou em observar como a busca de k-vizinhos mais próximos se comporta nos métodos de indexação R-Tree e M-Tree ao variar as dimensões dos dados (número de linhas e colunas da tabela relacional) e o tamanho da página de disco empregada pelos nós dessas estruturas. É esperado que em algumas situações (principalmente ao aumentar as dimensões dos dados) que o algoritmo de construção das estruturas se torne ineficiente, então este trabalho visa encontrar essas situações.

1.2 Objetivos

O objetivo do trabalho é avaliar empiricamente o comportamento dos métodos de acesso M-Tree e R-Tree para encontrar situações que se torna viável de utilizar a busca

dos k-vizinhos mais próximos de acordo com a variação das dimensões dos dados (número de linhas e colunas de uma tabela relacional) e variação do tamanho das páginas de disco utilizadas para armazenamento dos nós das estruturas de dados.

1.3 Método

Os resultados foram obtidos através de buscas aos k-vizinhos mais próximos nos métodos de acesso R-Tree e M-Tree em bases de dados do repositório *Machine Learning Repository* (LICHMAN, 2013). Os resultados foram analisados a partir de gráficos exibindo a tempo de execução dos algoritmos, o número de acessos ao disco e a quantidade de cálculos de distância.

1.4 Organização do Trabalho

O trabalho possui a seguinte organização. O capítulo 2 contém a fundamentação teórica, com a descrição de estruturas de indexação exemplificadas pela estrutura B-Tree, estruturas de indexação multidimensionais exemplificada pela estrutura R-Tree, estruturas de indexação métricas exemplificada pela estrutura M-Tree e busca por similaridade com o algoritmo da busca de k vizinhos mais próximos. No capítulo 3 são apresentados os experimentos realizados, as comparações de desempenho da busca dos 10 vizinhos mais próximos nas estruturas R-Tree e M-Tree também comparando com o método de busca sequencial e variando o tamanho da página de disco para cada base. O capítulo 4 apresenta as conclusões obtidas a partir dos experimentos.

2 Fundamentação Teórica

Este capítulo descreve os conceitos fundamentais empregados no desenvolvimento do projeto.

2.1 Métodos de acesso

Os Métodos de Acesso (MA) são estruturas utilizadas para que uma busca em um conjunto de dados seja feita de maneira eficiente. Fazendo uma analogia com o mundo real, imagine que existe uma sala onde estão guardados todas as pastas de todos os alunos que já passaram por uma determinada universidade de maneira desorganizada. Se todos as pastas não tiverem uma ordem estabelecida, para que se encontre a pasta de um determinado aluno ou ex-aluno com base no seu nome seria necessário verificar todas as pastas até que se encontre a pasta desejada. Caso a pasta do aluno não exista, seria necessário analisar todas as pastas para garantir que a pasta não existe. No caso descrito seria a não utilização de um método de acesso para organizar os dados e realizar a busca de maneira eficiente.

Para facilitar a busca, poderia-se ordenar as pastas em ordem alfabética do nome do aluno, dessa forma quem está procurando uma pasta específica sabe quando pode parar a busca (no caso da pasta que não esteja no arquivo). Para facilitar ainda mais, poderia-se dividir as pastas em armários que levam a primeira letra do nome dos alunos. Quando se procura por um aluno, o único armário que precisa se fazer a procura é o armário em que o nome se encontra, por exemplo, se procura um aluno cujo que se inicia com a letra C, teria que verificar todas as pastas de alunos que estão no armário C.

Caso o volume de pastas aumente, ainda poderia dividir os armários de tal forma que desmembrasse ainda mais, por exemplo, separar em cada região do armário em faixas como CA até CF, CG até CM e CP até CZ, dessa forma reduziria ainda mais o processo de procura de uma pasta. Se o usuário procura o nome Carlos, poderia ir até o armário C, gaveta CA até CF e analisar as pastas até encontrar a pasta desejada.

Nesse último exemplo seria necessário no mínimo 1 armário para cada letra do alfabeto, o que poderia desperdiçar muito espaço dentro da sala, por exemplo, pela fragmentação (espaço livre) no armário da letra Z (considerando que há poucos nomes que se iniciam com a referida letra). Além disso, esse formato de organização somente resolve o problema da busca por nomes. Caso seja necessário encontrar todos os alunos que ingressaram em um determinado ano, ou todos os alunos que ingressaram em um determinado curso, esse modo de organização das pastas não resolveria o problema.

Um sistema computacional consegue utilizar os métodos de acesso para resolver esses problemas de busca de maneira eficiente. Um método de acesso pode criar estruturas (índices) utilizando atributos dos dados. Nesse exemplo, poderia ser criada uma estrutura de indexação para cada uma desses atributos: nome, ano de ingresso, curso etc. Dessa forma, de acordo com a busca a ser efetuada, o sistema computacional utiliza a melhor estrutura para encontrar os registros. O método de acesso B-Tree ([BAYER; MCCREIGHT, 1972](#)) é um método de acesso amplamente utilizado em Sistemas de Gerenciamento e Bancos de Dados Relacionais (SGBDR) para indexação de dados com relação de ordem por possuir as características descritas a seguir.

2.1.1 B-Tree

O método de acesso B-Tree é considerado o método padrão de representação de chaves ([FOLK; ZOELLICK; RICCARDI, 1998](#)) e é um dos mais utilizados quando tratamos de dados ordenáveis como cadeia de caracteres, números e datas.

As árvores binárias de busca balanceadas são índices dinâmicos que tem custo amortizado para inserções e buscas na estrutura. Entretanto, seu propósito é memória principal. A principal motivação para a criação de uma nova estrutura foi a necessidade de utilizar armazenamento secundário com acesso aleatório (discos rígidos). Nesses dispositivos, o tempo para recuperação de uma página de disco é elevado, entretanto, a recuperação da página inteira é feita em operação única. A B-Tree foi criada como uma generalização da árvore binária mas possui algumas diferenças que a torna interessante de ser utilizada. Uma dessas diferenças é que na B-Tree cada nó pode armazenar mais de uma chave de registro e mais dois caminhos (filhos) fazendo com que a altura árvore de indexação seja menor enquanto na árvore binária cada nó possui somente uma chave de registro e apenas dois caminhos (filhos) ([COMER, 1979](#)). Na prática, para armazenamento em dispositivos secundários, a estrutura B+Tree é uma B-Tree que armazena chaves e ponteiros para páginas (que representam nós) nos nós internos (raiz e índices) e armazena chaves e localização dos registros nos nós folha. Em geral, cada nó da B+Tree de ordem d contém no máximo $2d$ chaves e $2d + 1$ ponteiros ([COMER, 1979](#)). Dessa forma, a B+Tree carrega um nó da árvore para a memória principal, efetua as comparações necessárias e depois acessa o próximo nó (se necessário). Na árvore binária, para cada chave comparada é necessário fazer um acesso ao disco para acessar o próximo nó. Como o custo de comparação na memória principal é menor que o custo de acesso ao disco, manter várias chaves em um nó se torna benefício considerável.

Assim como na árvore binária, todos os nós da B+Tree armazenam chaves de registros e os nós não folha possuem ponteiros para os nós filhos. A quantidade de chaves armazenadas em um nó da árvore é baseada no tamanho de uma página do disco uma vez que a B-Tree armazena cada um de seus nós em uma página do disco ([COMER, 1979](#)).

Desta forma, a B+Tree utiliza os recursos computacionais de maneira eficiente, pois para que um dado seja encontrado, o sistema computacional carrega um nó da árvore com apenas um acesso ao disco e efetua as operações necessárias.

Outra vantagem da B+Tree em relação a árvore binária é que os métodos de inserção e deleção de registros sempre deixam a árvore balanceada (COMER, 1979) pois a árvore é construída da base para o topo (FOLK; ZOELLICK; RICCARDI, 1998). O benefício de se manter a árvore balanceada é que a complexidade de tempo das buscas é da ordem $O(\log_d N)$ onde d é a ordem da árvore (número máximo de elementos armazenados em um nó) e N é o número de chaves armazenadas em cada nó da árvore (COMER, 1979). Dessa forma, a B+Tree resolve o problema de custo linear de inserção e deleção que acontece na árvore binária (FOLK; ZOELLICK; RICCARDI, 1998).

Em uma estrutura de indexação não balanceada como a árvore binária, pode-se acontecer a construção unilateral da árvore, fazendo com que a utilização da árvore de indexação tenha um desempenho pior que o método sequencial, uma vez que o custo da busca pode ser o mesmo da busca sequencial (linear) além desse custo se propagar para as operações de inserção e deleção, já que as duas operações utilizam a busca.

Ao inserir um novo registro na B+Tree, o algoritmo percorre a árvore do nó raiz até encontrar o nó em que a chave do registro deverá ser inserido, insere a chave e caso necessário é feito o balanceamento da árvore recursivamente até a raiz (COMER, 1979).

A busca em uma árvore binária é baseada na comparação da chave buscada em relação à chave armazenada no nó, iniciando pelo nó raiz. Caso a chave buscada seja menor que a chave armazenada no nó, é utilizado o filho da esquerda para uma nova comparação, caso a chave buscada seja maior que a chave armazenada no nó, o filho da direita é utilizado para uma nova comparação. Dessa forma sucessivamente até que se encontre a chave na árvore ou que os nós se esgotem e a chave não pertença à árvore. A busca na B+Tree é feita de maneira similar, com a diferença de carregar uma página que possui várias chaves de comparação para a memória principal (mais rápida que a memória secundária), ou seja, não é necessário acessar a memória secundária para cada comparação de chave, um conjunto de chaves de comparação é carregado para a memória principal, são feitas as comparações necessárias e assim é escolhido o novo nó a ser visitado. Vale ressaltar que se a implementação mantiver as chaves de um nó da B+Tree ordenados, uma busca binária pode ser feita para encontrar uma chave no nó em memória principal. Em páginas típicas de disco, atualmente de 4 KB ou maiores, podem caber centenas de pares chaves e localização, assim, uma busca binária no nó é desejável.

Apesar de não ser utilizada neste trabalho, a B-Tree foi base para construção dos métodos R-Tree (GUTTMAN, 1984) e M-Tree que serão descritos na sequência. Esses métodos possuem características similares à B-Tree como a construção da folha para a raiz sempre mantendo a árvore balanceada.

2.2 Métodos de acesso multidimensionais e métricos

Os métodos de acesso convencionais como a B+Tree não resolvem o problema quando a busca por similaridade ou busca por distância é necessária, pois não utiliza todos os atributos do objeto para fazer a indexação, somente utiliza a chave do objeto. Para a otimização de consultas por similaridade é necessário utilizar outros tipos de métodos de acesso, como os métodos de acesso multidimensionais e os métodos de acesso métricos. Segundo (SAMET, 2006, p. 485), esses métodos têm como objetivo resolver as seguintes consultas:

1. encontrar objetos que possuem determinados valores de atributos (*point query* ou consulta pontual);
2. encontrar objetos cujas distâncias estão dentro de uma determinada abrangência (*range query* ou busca por abrangência);
3. encontrar uma quantidade de objetos mais próximos a um objeto de consulta (*k-nearest neighbor queries* ou busca aos vizinhos mais próximos);
4. Encontrar pares de objetos que são suficientemente similares entre si (*all-closest-pairs queries*). Essa é também uma variação de uma consulta mais geral conhecida como *spatial join query* ou junção espacial.

Os métodos de acesso multidimensionais se baseiam na construção de vetores de tamanho fixo a partir dos atributos dos dados que serão utilizados nas consultas. Nesses métodos, a similaridade entre dois objetos não possui correlação com os valores dos atributos e sim por uma função distância (como a distância Euclidiana) desses vetores em um plano multidimensional (CIACCIA; PATELLA; ZEZULA, 1997).

Já os métodos de acesso métricos utilizam a distância relativa entre objetos em um espaço métrico. Segundo (CIACCIA; PATELLA; ZEZULA, 1997), um espaço métrico $M = (D, d())$, onde D é o domínio de valores dos atributos dos objetos (chaves de indexação) e $d()$ é a função de distância que deve satisfazer às seguintes propriedades:

1. **simetria:** $d(O_x, O_y) = d(O_y, O_x)$
2. **identidade:** $d(O_x, O_x) = 0$
3. **não negatividade:** $d(O_x, O_y) > 0, \forall O_x \neq O_y$
4. **desigualdade triangular:** $d(O_x, O_y) \leq d(O_x, O_z) + d(O_z, O_y)$

Uma vantagem dos métodos de acesso multidimensionais em relação aos métodos de acesso métricos é que eles podem responder diversas consultas espaciais e topológicas

(SCHNEIDER; BEHR, 2006) que estão fora do contexto desta monografia. A seguir são descritos o método de acesso multidimensional R-Tree e o método de acesso métrico M-Tree, ambas utilizadas para indexação de dados complexos.

2.2.1 R-Tree

O método de acesso R-Tree, proposto em 1984 por (GUTTMAN, 1984) tem o objetivo de indexar dados geométricos como pontos, segmentos de linha, superfícies, volumes e hipervolumes em espaços de alta dimensão (MANOLOPOULOS et al., 2006, p. 3). O método de indexação R-Tree foi proposto porque métodos clássicos de indexação para dados em uma dimensão (como B-Tree) não possuem um bom desempenho indexando dados multidimensionais (GUTTMAN, 1984).

A R-Tree é uma árvore balanceada em altura similar à B-Tree (GUTTMAN, 1984), e é utilizada para organização de um conjunto de objetos geométricos de d dimensões representados pelos seus retângulos envolventes mínimos (*minimum bounding d-dimensional rectangles* – MBR) que é o menor retângulo de d -dimensões que engloba o objeto (MANOLOPOULOS et al., 2006, p. 7). Cada nó da R-Tree corresponde ao MBR que engloba seus filhos e cada nó folha da árvore contém ponteiros para os objetos no banco de dados (MANOLOPOULOS et al., 2006, p. 7). Os objetos são representados como uma coleção de tuplas no banco de dados onde cada tupla possui um identificador único que pode ser utilizado para recuperá-los (GUTTMAN, 1984).

Segundo (MANOLOPOULOS et al., 2006, p. 7), uma R-Tree de ordem (m, M) possui as seguintes características:

- cada nó folha (exceto o nó raiz) possui M entradas. O mínimo de entradas permitido é $m \leq M/2$. Cada entrada é da forma (mbr, oid) , onde mbr é o MBR que contém o objeto e oid é o identificador do objeto;
- cada nó interno da árvore deve possuir entre $m \leq M/2$ e M entradas - assim como os nós folha. Cada entrada é da forma (mbr, p) onde p é o ponteiro para um nó filho e mbr é o MBR que contém todos os MBRs de seus filhos;
- o número mínimo de entradas no nó raiz é 2, a não ser que a árvore possua apenas 1 nó folha (apontando para 0 ou 1 objetos);
- todos os nós folha da R-Tree estão no mesmo nível.

As Figuras 1 e 2 apresentam respectivamente o exemplo de um conjunto de MBRs de alguns dados geométricos genéricos (os objetos não estão sendo mostrados, somente os retângulos) e como a R-Tree correspondente é construída (MANOLOPOULOS et al., 2006, p. 8).

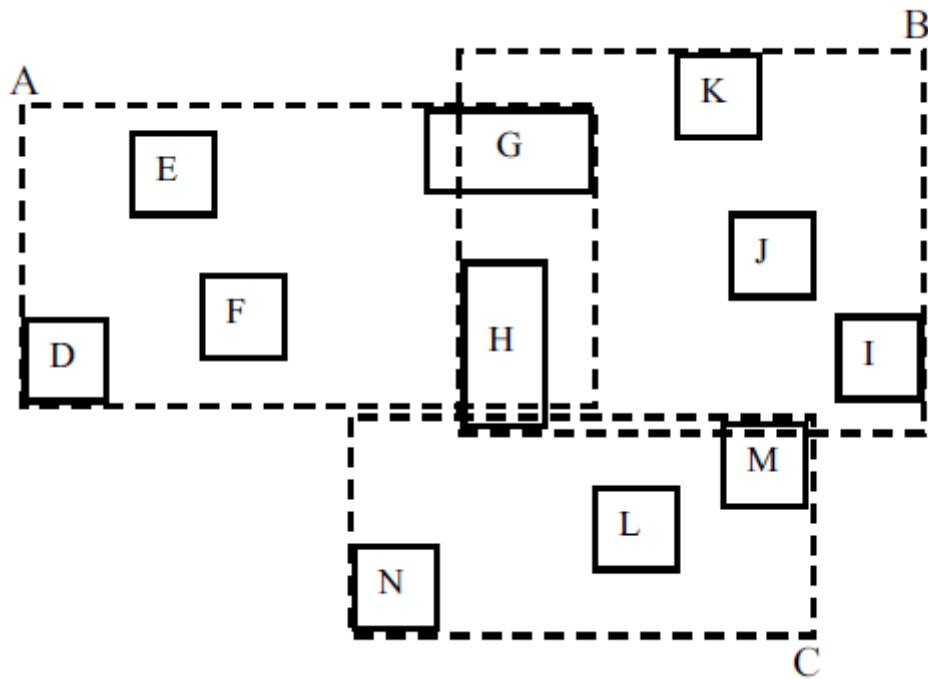


Figura 1 – Exemplo de dados MBRs e seus MBRs (MANOLOPOULOS et al., 2006, p. 8)

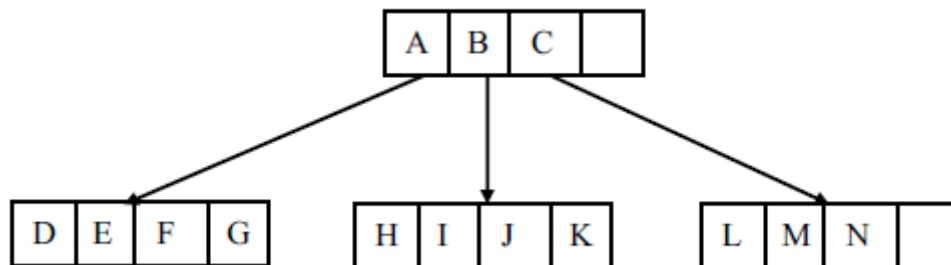


Figura 2 – Construção da R-Tree (MANOLOPOULOS et al., 2006, p. 8)

A busca na R-Tree é semelhante a busca na B-Tree, porém pode ser que o algoritmo tenha que visitar mais de um ramo ou subárvore sempre que ocorrer sobreposição de MBRs. Por exemplo, na árvore apresentada na Figura 1, os MBRs A e B se sobrepõem, ou seja, caso seja feita uma busca do objeto representado pelo MBR H, a busca de descer tanto o ramo A quanto o ramo B da árvore.

Para inserir um novo registro na R-Tree, em resumo, é necessário seguir os seguintes passos:

1. a partir da raiz, procura-se o nó folha em que a tupla será inserida;
2. adiciona a nova tupla no nó selecionado. Caso o nó tenha espaço para armazenar uma nova tupla, a nova tupla é adicionada, caso contrário, divide-se o nó selecionado

- e distribui os objetos entre os dois nós;
3. propaga-se as mudanças da folha para a raiz mantendo a árvore balanceada (dividindo nós nos níveis superiores quando necessário);
 4. caso seja necessário dividir o nó raiz, cria-se um novo nó apontando para os dois nós gerados na divisão (aumentando a altura da árvore).

O Algoritmo 2.2.1, proposto em (MANOLOPOULOS et al., 2006, p. 10) apresenta a inserção de um novo registro na R-Tree.

Algorithm 1 algoritmo inserirRegistro(TipoEntrada E, TipoNo RN)

```

1: //Insere um novo registro E em uma árvore R que possui o nó raiz RN
2: function INSERIRREGISTRO(TipoEntrada E, TipoNo RN)
3:   Percorra a árvore a partir da raiz RN até o nó folha apropriado. Em cada nível,
   seleccione o nó L cujo MBR exigirá a menor ampliação para cobrir E.mbr
4:   Em caso de empate, seleccione o nó que possui o MBR com menor área
5:   if o nó selecionado L pode acomodar E then
6:     Insere E no L
7:     Atualize todos MBRs de L até o nó raiz RN até que todos cubram E.mbr
8:   else
9:     //L não possui espaço
10:    Deixe  $\varepsilon$  ser o conjunto que consiste todos os registros de L e o novo registro E.
    Seleccione como sementes duas entradas  $e1$  e  $e2 \in \varepsilon$ , onde a distância entre  $e1$  e  $e2$  é
    a maior entre todos os outros pares de registros de  $\varepsilon$ . Crie dois nós, L1 e L2, onde o
    primeiro contém  $e1$  e o segundo  $e2$ 
11:    Analise os membros restantes de  $\varepsilon$ , um a um e direcione eles para L1 ou L2,
    dependendo em qual MBRs desses nós requer a menor expansão para cobrir essa
    entrada
12:    if Acontece um empate then
13:      Direcione a entrada para o nó que possui o MBR de menor área
14:    end if
15:    if Acontece um empate novamente then
16:      Direcione a nova entrada para o nó que contém o menor número de entradas
17:    end if
18:    if durante o direcionamento de entradas, restarem  $\lambda$  entradas a serem direcio-
    nadas e um dos nós contém  $m - \lambda$  entradas then
19:      Direcione todas as entradas restantes para esse nó sem considerar o critério
      citado acima (para que o nó contenha no mínimo  $m$  entradas)
20:    end if
21:    Atualize os MBRs dos nós que estão no caminho do nó raiz até L, para que
    cubram L1 e acomodem L2
22:    Caso necessário, faça divisões nos nós superiores
23:    Caso a raiz tenha que se dividir, cria um novo nó raiz
24:    Aumente a altura da árvore
25:  end if
26: end function

```

Sempre que se tenta inserir um registro em um nó M que está cheio, é necessário dividir os registros de M + o novo registro em dois nós da árvore de indexação (GUTTMAN, 1984)(MANOLOPOULOS et al., 2006, p. 10). Essa divisão deve ser feita de tal modo que ao realizar uma busca seja improvável ter que visitar os dois nós (GUTTMAN, 1984). Para que a divisão dos nós seja eficiente é necessário que os MBRs dos novos nós cubram a menor área possível. Foram propostos três métodos de realizar a divisão dos nós, um de complexidade linear, um de complexidade quadrático e outro de complexidade exponencial (GUTTMAN, 1984):

1. **Linear:** são escolhidos os dois registros que estão mais distantes entre si e utilizados como sementes para os novos dois nós da árvore de indexação. A partir daí percorre-se por todos os outros registros aleatoriamente adicionando-os ao nó que requer o menor aumento do seu MBR;
2. **Quadrático:** são escolhidos os dois registros que juntos gerem o MBR com maior área morta (área restante do MBR desconsiderando a área dos dois registros) como sementes dos dois novo nós da árvore de indexação. Na sequência, percorre-se todos os registros restantes inserindo cada registro no nó que gere o máximo de área morta requerindo o mínimo de aumento do seu respectivo nó;
3. **Exponencial:** é o método mais simples de ser implementado e consiste em gerar todos os agrupamentos possíveis entre os registros e identificar a combinação que possui os MBRs de menor área (ou volume). O problema desse método é que caso o nó armazene muitos registros, o número de possíveis divisões se torna muito grande e é inviável de ser utilizado.

Neste trabalho utilizou-se o método quadrático para as divisões de nós, de acordo com (GUTTMAN, 1984) é um método que apresenta uma boa relação custo/benefício para indexação e recuperação dos dados.

2.2.2 M-Tree

O método M-Tree (CIACCIA; PATELLA; ZEZULA, 1997) foi proposto para organizar e buscar grandes conjuntos de dados em espaços métricos genérico onde a similaridade entre pares de objetos é definida somente pela função de distância. O método M-Tree tem construção similar ao método R-Tree. A M-Tree somente considera a distância relativa dos objetos para organizar e particionar o espaço da busca e somente requer que a função da distância seja uma métrica, permitindo que a propriedade desigualdade triangular seja aplicada e que possa ser usada para podar ou reduzir o espaço da busca (CIACCIA; PATELLA; ZEZULA, 1997).

Uma outra diferença do método de indexação M-Tree em relação ao R-Tree é que ao invés de armazenar um retângulo de d -dimensões que engloba o objeto, um objeto e uma distância (raio) são empregados para definir uma região (bola). A Figura 3 é um exemplo de como um objeto é representado em uma M-Tree.

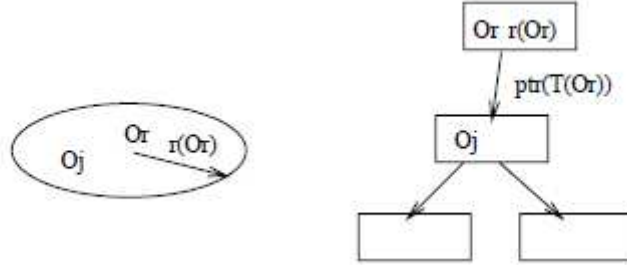


Figura 3 – Um objeto O_r possui um raio de cobertura $r(O_r)$ e referencia uma árvore $T(O_r)$ (CIACCIA; PATELLA; ZEZULA, 1997)

Os registros indexados pela M-Tree ficam armazenados nos nós folha da árvore representados por suas chaves ou atributos, enquanto os nós índice armazenam objetos chamados de pivôs (*routing objects*) (CIACCIA; PATELLA; ZEZULA, 1997).

Os nós índice são compostos de:

- O_r : objeto pivô;
- $ptr(T(O_r))$: ponteiro que referencia a raiz de uma subárvore $T(O_r)$, chamado de árvore de cobertura de O_r ;
- $r(O_r)$: raio de cobertura de O_r , sendo que todos os objetos dentro da área de cobertura de O_r estão dentro do $r(O_r)$ partindo de O_r , $r(O_r) > 0$;
- $d(O_r, P(O_r))$: distância de O_r até seu pai. O objeto pivô armazena a distância até $P(O_r)$, seu objeto pai, que é o pivô que referencia o nó onde o registro O_r é armazenado. Esse dado não existe no nó raiz da M-Tree.

Os objetos armazenados nos nós folha (O_j) são similares aos objetos armazenados nos nós índice. Em um nó não folha, não é necessário armazenar o raio de cobertura e o ponteiro referencia o identificador do registro do banco de dados (*oid*) (CIACCIA; PATELLA; ZEZULA, 1997). Em resumo, os objetos armazenados nos nós índice possuem os seguintes dados:

- O_j : atributos do registro do banco de dados;
- $oid(O_j)$: identificador do objeto para acesso ao registro completo na base de dados;

- $d(O_j, P(O_j))$: distância de O_j até seu pai.

O algoritmo de inserção de um novo registro na M-Tree é um algoritmo recursivo que tenta encontrar o nó mais apropriado para que o novo registro seja inserido (CIACCIA; PATELLA; ZEZULA, 1997). De maneira básica, a decisão em cada nível da árvore de indexação sobre qual subárvore o registro será inserido é feita de maneira que se evite o aumento do raio de cobertura. Caso haja empate, a subárvore escolhida é a que possui a menor distância entre o objeto pivô e o novo objeto inserido (CIACCIA; PATELLA; ZEZULA, 1997).

Durante a inserção de um novo registro na M-Tree, pode ser necessário a divisão do nó folha que armazena os objetos. A divisão de nós na M-Tree ideal, assim como na R-Tree, deve escolher dois registros para serem os pivôs dos novos dois nós (O_{p1} e O_{p2}), onde gere o mínimo de volume (para evitar espaços mortos) e o mínimo de sobreposição (CIACCIA; PATELLA; ZEZULA, 1997). Os seguintes algoritmos para escolha de pivôs foram propostos:

1. **m_RAD**: O algoritmo *minimum (sum of) RADii* ou mínima soma dos raios é o mais complexo em termos de cálculos de distância pois calcula as distâncias de todos os pares de registros presentes e escolhe os dois objetos onde a soma de seus raios $r(O_{p1}) + r(O_{p2})$ é a menor dentre os outros pares;
2. **mM_RAD**: Este algoritmo é similar ao m_RAD, porém ele minimiza o maior dos dois raios;
3. **M_LB_DIST**: O algoritmo *Maximum Lower Bound on DISTance* ou maior distância do limite inferior se difere das duas anteriores pois utiliza cálculos de distância armazenados. Para $O_{p1} \equiv O_p$, o algoritmo determina O_{p2} como o objeto mais distante de O_p : $d(O_{p2}, O_p) = \max_j(d(O_j, O_p))$;
4. **RANDOM**: Escolha aleatória de um par de pivôs. É uma estratégia rápida e seu desempenho pode ser utilizado como referência para outras formas de escolha;
5. **SAMPLING**: Este modo de escolha é similar ao RANDOM, porém executado em uma amostra de objetos de tamanho $s > 1$. Para cada $s(s-1)/2$ pares de objetos da amostra, os registros são distribuídos e os potenciais raios de cobertura são estabelecidos. O par de objetos onde o maior entre os dois raios de cobertura é mínimo é selecionado. Caso a promoção seja confirmada, somente s distribuições diferentes são testadas.

Após a escolha dos pivôs (O_{p1} e O_{p2}), o algoritmo deve distribuir os registros restantes em N entre os dois novos nós (N_1 e N_2). Para isso, (CIACCIA; PATELLA; ZEZULA, 1997) propõe duas estratégias:

1. **Hiperplano Generalizado:** Direcione cada objeto $O_j \in N$ para o objeto pivô mais próximo. Se $d(O_j, O_{p1}) \leq d(O_j, O_{p2})$, insira O_j em N_1 , caso contrário insira O_j em N_2 .
2. **Balanceada:** Calcule $d(O_j, O_{p1})$ e $d(O_j, O_{p2})$ para todos $O_j \in N$. Repita até que N esteja vazia.
 - Insira em N_1 o vizinho mais próximo de O_{p1} em N e o remova de N ;
 - Insira em N_2 o vizinho mais próximo de O_{p2} em N e o remova de N ;

Neste trabalho foi empregado o método de divisão mM_RAD e a estratégia de distribuição dos registros balanceada.

2.3 Maldição da dimensionalidade

Os métodos de acesso tem por objetivo indexar os dados para ganho de desempenho ao buscar os dados. Os métodos de acesso citados criam estruturas no formato de árvore hierárquica e por definição não limitam o número de dimensões utilizadas, porém é esperado que o desempenho de consultas nessas estruturas degrade de acordo com o aumento de dimensões, o que é chamado de maldição da dimensionalidade.

Com o aumento do número de dimensões, os objetos tendem a se condensar causando um aumento considerável no número de cálculos de distância. Isso significa que a distância entre o vizinho mais próximo e o vizinho mais longe tendem a convergir com o aumento da dimensionalidade. O pior caso pode ser uma situação onde $d(x,x) = 0$ e $d(y,x) = 1$ para todos $y \neq x$, isso significa que a busca por similaridade terá que comparar o objeto buscado com todos os objetos da base de dados, ou seja, a estrutura de indexação não cumpre seu papel, uma vez que não é possível desconsiderar objetos da base durante uma busca (SAMET, 2006, p.486,487).

A maldição da dimensionalidade tem uma relação com a busca por similaridade em alta dimensionalidade ao ponto de gerar a dúvida se vale a pena utilizar a busca do vizinho mais próximo nesses domínios. Experimentos mostram que a maldição da dimensionalidade pode ser notada em conjuntos que possuem objetos com 10 a 15 dimensões (SAMET, 2006, p.486).

2.4 Buscas por similaridade

Uma busca por similaridade é um tipo de busca utilizado em vários ramos da computação, desde reconhecimento de padrões até busca de textos e busca de dados multimídia. Nas últimas décadas, a utilização desse tipo de busca tem aumentado em

vários ramos como geografia, biologia molecular, busca de texto aproximado, busca de vídeos, busca de imagens e padrões de classificação. Consequentemente, a demanda por uma busca por similaridade rápida também cresceu (WEN, 2004).

Uma busca por similaridade é definida da seguinte forma (WEN, 2004): dado um conjunto de objetos e um objeto em uma consulta, encontre o objeto do conjunto mais similar ao objeto da consulta. Dependendo do critério de similaridade, pode-se utilizar múltiplos atributos dos objetos ou até mesmo todos para definir a similaridade entre objetos. Tratando cada atributo dos objetos como uma dimensão em um espaço multi-dimensional, o problema da busca por similaridade pode ser mapeado em um problema do vizinho mais próximo (que define o objeto mais próximo ou mais similar ao objeto buscado) em um espaço multidimensional e o critério de similaridade é então definido por uma função de distância. Por esse motivo, a busca por similaridade as vezes é chamada de busca do vizinho mais próximo e o resultado da busca por similaridade é chamado de vizinho mais próximo.

A busca do vizinho mais próximo pode ser modificada para a busca aos k-vizinhos mais próximos, algoritmo KNN (*k-nearest neighbour query*). A busca retorna os k vizinhos mais próximos (ou k objetos mais similares) do objeto de consulta. Esse algoritmo é definido a seguir e é utilizado para a avaliação de desempenho nas estruturas de indexação R-Tree e M-Tree de acordo com o aumento de dimensões (colunas de uma tabela em um banco de dados relacional).

2.4.1 Busca aos k-vizinhos mais próximos

A busca aos k-vizinhos mais próximos é um tipo de busca muito utilizada nos dias atuais. Uma utilização prática, por exemplo, seria quando um usuário deseja encontrar os 10 restaurantes mais próximos a um determinado ponto. A implementação eficiente da busca aos k-vizinhos é principalmente de interesse de sistemas de informação geográfica (ROUSSOPOULOS; KELLEY; VINCENT, 1995).

Diferentemente da busca baseada em relação de ordem (utilizada na B-Tree por exemplo), a busca aos vizinhos mais próximos é baseada em similaridade (SEIDL; KRIEGEL, 1998) e esta similaridade é encontrada através de funções de distância entre pontos em um espaço multidimensional ou métrico. Para uma busca dos k-vizinhos mais próximos é necessário informar um objeto q e um parâmetro k para que seja retornado os k objetos que são mais similares (ou próximos) de q (SEIDL; KRIEGEL, 1998).

Em (ROUSSOPOULOS; KELLEY; VINCENT, 1995) é proposto o Algoritmo 2.4.1 para busca dos k-vizinhos mais próximos. Trata-se de um algoritmo de busca em profundidade para a estrutura R-Tree. Este algoritmo é utilizado neste trabalho e é a base para a busca dos k-vizinhos mais próximos na estrutura R-Tree e modificado para execução da

busca dos k-vizinhos mais próximos na estrutura M-Tree.

Algorithm 2 Procedimento Recursivo - Pseudocódigo Busca dos Vizinhos Mais próximos

```

1: function BUSCAVIZINHOSMAISPROXIMOS(No, Ponto, MaisProximo)
2:   NO No //Nó atual
3:   PONTO Ponto //Ponto da consulta
4:   VIZINHOMAISSPROXIMO MaisProximo //Vizinho mais próximo
5:   //Variáveis Locais
6:   NO novoNo
7:   ARRAYRAMOS listaRamos
8:   integer dist, ult, i
9:   if No.tipo = FOLHA then
10:     for i := 1 até No.count do
11:       dist := objetoDIST(Ponto, No.ramo[i].retangulo)
12:       if dist < MaisProximo.dist then
13:         MaisProximo.dist := dist
14:         MaisProximo.retangulo := No.ramo[i].retangulo
15:       end if
16:     end for
17:   else
18:     //Gera Lista de Ramos Ativos
19:     geraListaRamos(Ponto, No, listaRamos)
20:     //Ordena Lista de Ramos Ativos baseado nos valores das métricas
21:     ordenaListaRamos(listaRamos)
22:     //Executa a poda descendente dos ramos
23:     //(talvez discarte todos os ramos)
24:     ult = podaListaRamos(No, Ponto, MaisProximo, listaRamos)
25:     //executa através da lista de ramos ativa
26:     for i := 1 até ult do
27:       novoNo := No.ramo [listaRamos[i]]
28:       //Recursivamente visita nós filhos
29:       buscaVizinhosMaisProximos(novoNo, Ponto, MaisProximo)
30:       //Executa a poda ascendente dos ramos
31:       ult := podaListaRamos(No, Ponto, MaisProximo, listaRamos)
32:     end for
33:   end if
34: end function

```

Esse pseudocódigo foi originalmente feito para utilização na R-Tree mas pode ser facilmente modificado para utilização na M-Tree. A única modificação necessária é em relação a como os cálculos de distância entre o objeto de consulta e os MBRs são feitos na M-Tree que emprega bolas (objeto e raio de cobertura) ao invés de MBRs.

Na R-Tree são utilizadas duas métricas para ordenação (sortBranchList) para a busca dos vizinhos mais próximos. Considere um objeto de consulta P e um objeto O contido em seu MBR. A primeira métrica é a menor distância entre os objetos O e P, chamada de MINDIST. A segunda métrica é a menor das maiores distâncias possíveis de P para

um lado (ou vértice) do MBR contendo O (ROUSSOPOULOS; KELLEY; VINCENT, 1995). A primeira métrica é chamada de métrica otimista, pois leva em consideração que a MINDIST irá encontrar um ponto naquela distância (o que nem sempre acontece). A MINMAXDIST é chamada de métrica pessimista, pois é uma distância maior mas que garante que dentro daquela distância existe pelo menos um ponto.

Na M-Tree, a métrica otimista é a distância entre o ponto buscado e o pivô - raio, que é a distância mais próxima entre a área de cobertura e o ponto buscado. Já a métrica pessimista é a distância entre o ponto buscado e o pivô + raio.

3 Avaliação de Desempenho dos Métodos de Acesso R-Tree e M-Tree

Os experimentos foram executados em um microcomputador com sistema operacional Linux 64 bits, i7-4770 @3.40GHz, 8 GB de RAM e disco rígido de 1 TB. Os experimentos foram implementados em C++ e utilizam a biblioteca Arboretum¹. Sabendo que os resultados dos experimentos variam de acordo com os dados utilizados, foram utilizadas bases reais retiradas do repositório *Machine Learning Repository* (LICHMAN, 2013). As bases de dados utilizadas são *Eigenfaces*, *Pendigits*, *Corel*, *Coverttype* e *Colors*. Para cada base de dados será executado testes com dois tamanhos de páginas de disco. Para as bases *Eigenfaces* e *Pendigits* serão utilizados páginas de disco de 2 KB e 4 KB, para as bases *Corel* e *Coverttype* serão utilizadas páginas de disco de 4 KB e 8 KB e para a base *Colors* serão utilizados páginas de disco de 8 KB e 16 KB. A configuração do tamanho da página de disco permite o ajuste fino do índice. Por um lado uma página maior resulta em tempo de construção maior, dado que os algoritmos de divisão de nó tem complexidade de tempo maior que linear. Por outro lado, uma página de disco maior permite que o índice tenha um menor número de níveis, que pode reduzir o número de acessos para atingir uma folha a partir da raiz. Além disso, afeta a sobreposição entre nós de um mesmo nível.

Para cada base de dados será acompanhada a construção da árvore de indexação de acordo com o aumento do volume dos dados (de 10% em 10%) comparando o número de acessos ao disco, o número de cálculos de distância e o tempo de construção (em segundos). Também serão executadas consultas dos 10 vizinhos mais próximos de acordo com aumento do tamanho da base de dados calculando o número de acessos ao disco, o número médio de cálculos de distância e o tempo de resposta para consultas (em segundos). Para todos os testes serão mostrados gráficos comparando o método de acesso R-Tree, o método de acesso M-Tree e o método sequencial. O resumo das bases de dados e suas informações é apresentado na Tabela 1.

Base de dados	Número de Elementos	Número de Dimensões	Tamanhos das páginas (KB)	Alturas R-Tree	Alturas M-Tree
Eigenfaces	11.900	16	2 e 4	4 e 3	4 e 3
Pendigits	10.992	16	2 e 4	10 e 5	4 e 3
Corel	68.040	32	4 e 8	5 e 4	4 e 4
Coverttype	581.102	54	4 e 8	14 e 11	6 e 5
Colors	112.682	112	16 e 32	12 e 7	4 e 3

Tabela 1 – Bases de dados utilizadas.

¹ Disponível em <http://gbdi.icmc.usp.br/arboretum> e <https://bitbucket.org/gbdi/arboretum>

Nos experimentos a seguir, as curvas referentes à construção do método de acesso sequencial (*Seq Scan*) representam a leitura da base de dados, serialização dos objetos e armazenamento em um arquivo sequencial com páginas de disco conforme a configuração de cada experimento. Assim, servem como base para comparação com os métodos R-Tree e M-Tree, principalmente em termos de número de páginas e tempo necessários para armazenamento em arquivo sequencial. Vale destacar que a R-Tree não realiza cálculos de distância na sua construção, assim, os cálculos dos limites utilizados para construção dos MBRs não são apresentados nos gráficos de cálculos de distância na construção dos índices. Com relação às consultas realizadas, o método de acesso sequencial permite a comparação base (*baseline*) para acessos a disco, computação de distâncias e tempo total em relação aos métodos M-Tree e R-tree.

3.1 Eigenfaces

Os dados de construção dos índices com a base de dados *Eigenfaces* que possui 11.990 objetos e 16 atributos numéricos são apresentados nas Figuras 4 e 5 que exibem o número de cálculos de distância, número de acessos ao disco e o tempo para a construção da base utilizando páginas de disco de 2 KB e 4 KB respectivamente para os métodos de acesso sequencial, R-Tree e M-Tree.

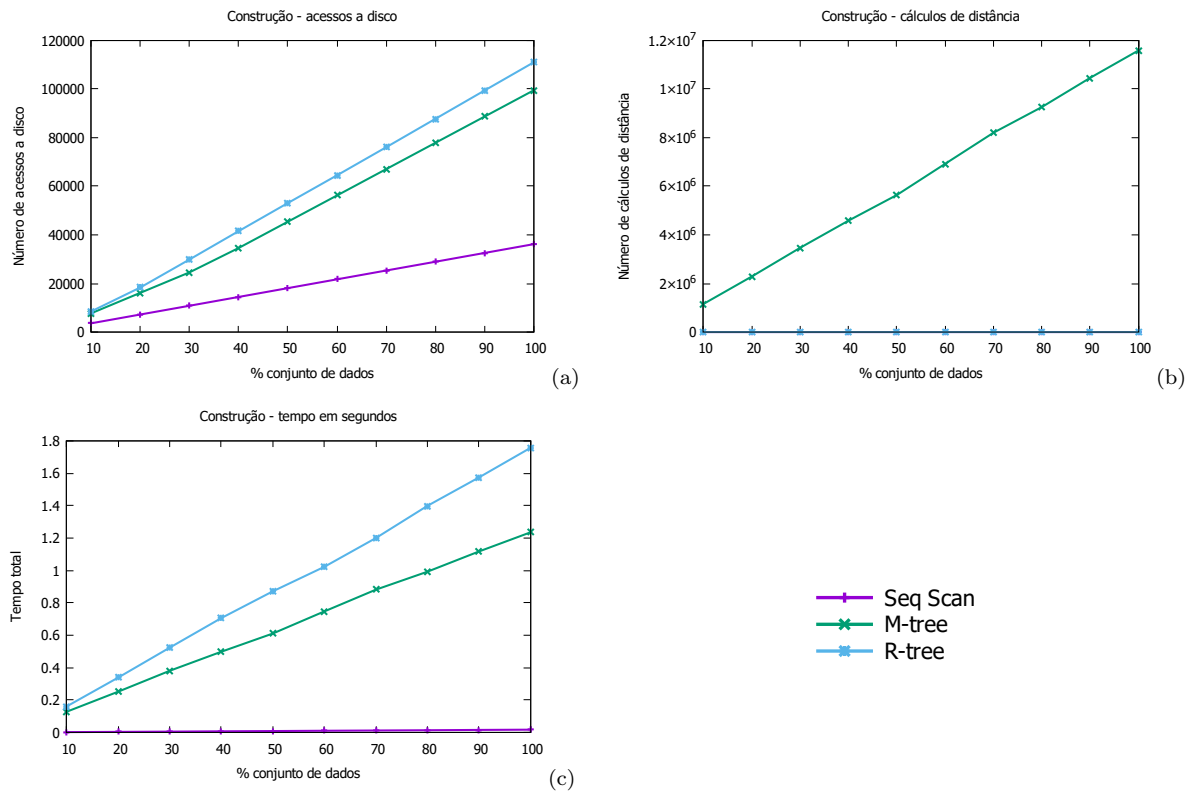


Figura 4 – Construção com a base *Eigenfaces* utilizando páginas de 2 KB

É possível observar que mesmo não fazendo cálculos de distância, a construção da R-Tree é 29,68% mais demorada que a construção da M-Tree. Também é possível perceber que ao construir a R-Tree são feitos 10,37% mais acessos ao disco em relação a M-Tree. Como esperado, por não criar uma árvore de indexação, o método sequencial tem a construção mais eficiente dos três gastando 0,98% do tempo em relação a R-Tree e 1,39% do tempo em relação a M-Tree, além de fazer 67,4% menos acessos ao disco em relação a R-Tree e 63,63% menos acessos ao disco em relação a M-Tree.

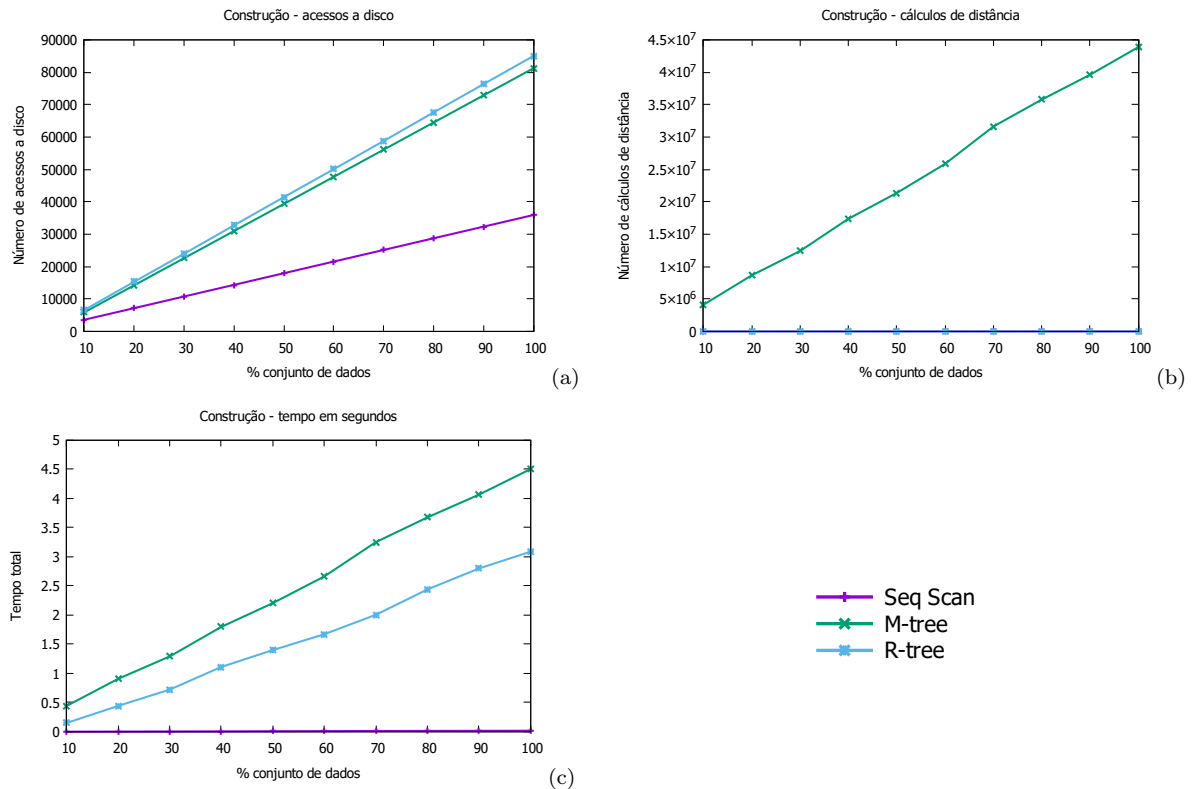


Figura 5 – Construção com a base *Eigenfaces* utilizando páginas de 4 KB

Ao utilizar páginas de disco de 4 KB percebe-se que todos os métodos se tornaram mais lentos na fase de construção quando comparamos a utilização de páginas de disco de 2 KB.

O método M-Tree foi o que teve maior aumento no tempo. Mesmo fazendo 18,32% menos acessos ao disco, o método teve um aumento de 72,58% em relação ao tempo e fez 73,59% mais cálculos de distância. O método R-Tree também reduziu o número de acessos ao disco ao utilizar páginas de disco de 2 KB, passando a fazer 23% menos acessos ao disco. Este método também apresentou um aumento no tempo de construção, utilizando 43,12% mais tempo para ser executado com páginas de 4 KB. O método sequencial levou 6,83% mais tempo para ser executado e fez 0,64% menos acessos ao disco.

Comparando os três métodos ao final da construção utilizando páginas de disco

de 4 KB, é possível visualizar que ambos métodos de indexação (R-Tree e M-Tree) levam mais tempo para serem construídos. Porém o método M-Tree teve o maior o aumento dos cálculos de distância, ao ponto que se tornou o menos eficiente utilizando 31,44% mais tempo do que o método R-Tree mesmo fazendo 4,46% menos acessos ao disco. Já o método sequencial se manteve mais eficiente, utilizando 0,60% do tempo gasto pela R-Tree e 0,41% do tempo gasto pela M-Tree. Além de fazer 57,74% menos acessos ao disco do que a R-Tree e 55,76% menos acessos ao disco do que a M-Tree.

Ao realizar consultas, o método M-Tree se mostra muito superior em relação aos outros métodos, tanto quando utilizadas páginas de 2 KB (apresentado na Figura 6) quanto quando utilizadas páginas de 4 KB (apresentado na Figura 7).

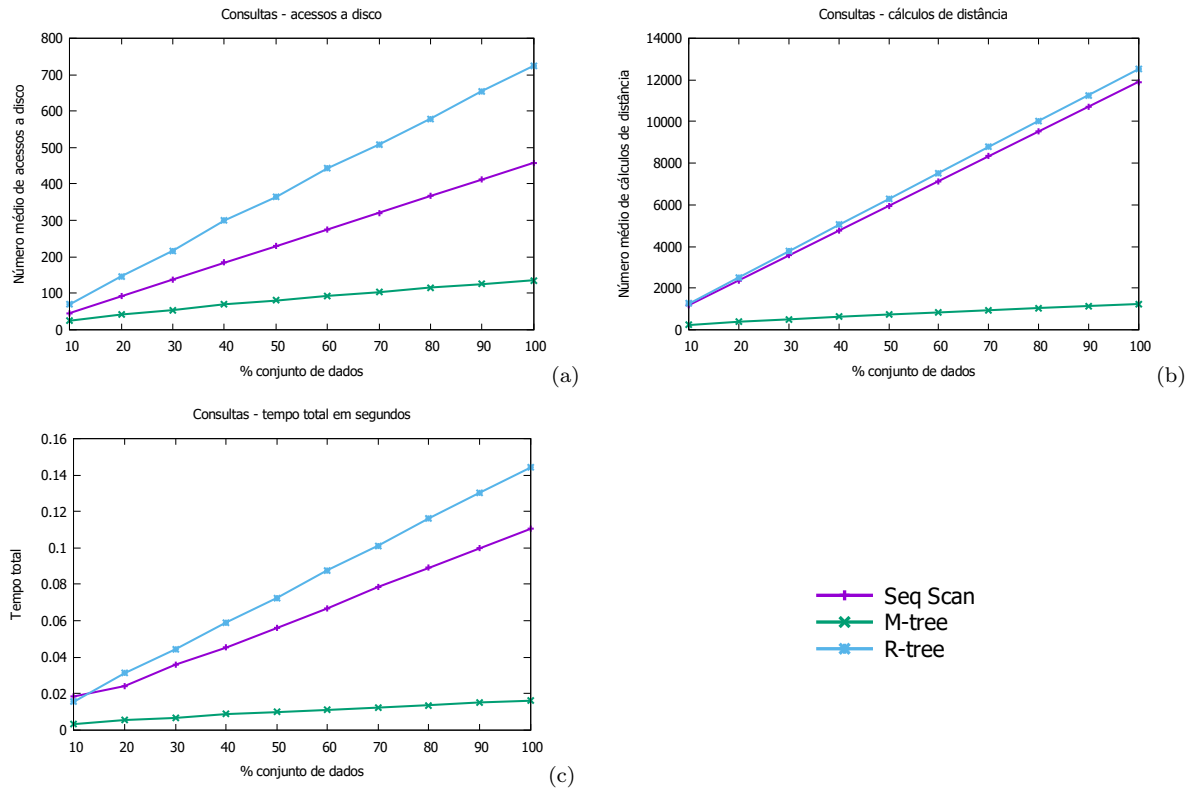


Figura 6 – Consultas com a base *Eigenfaces* utilizando páginas de 2 KB

A busca dos 10 vizinhos mais próximos na base completa utilizando o método M-Tree foi 88,80% mais rápido que o método R-Tree e 85,39% mais rápido do que o método sequencial utilizando páginas de 2 KB além de fazer 81,21% menos acessos ao disco do que o método R-Tree, 70,31% menos acessos ao disco do que o método sequencial, 90,44% menos cálculos de distância do que o método R-Tree e 89,61% menos cálculos de distância do que o método sequencial.

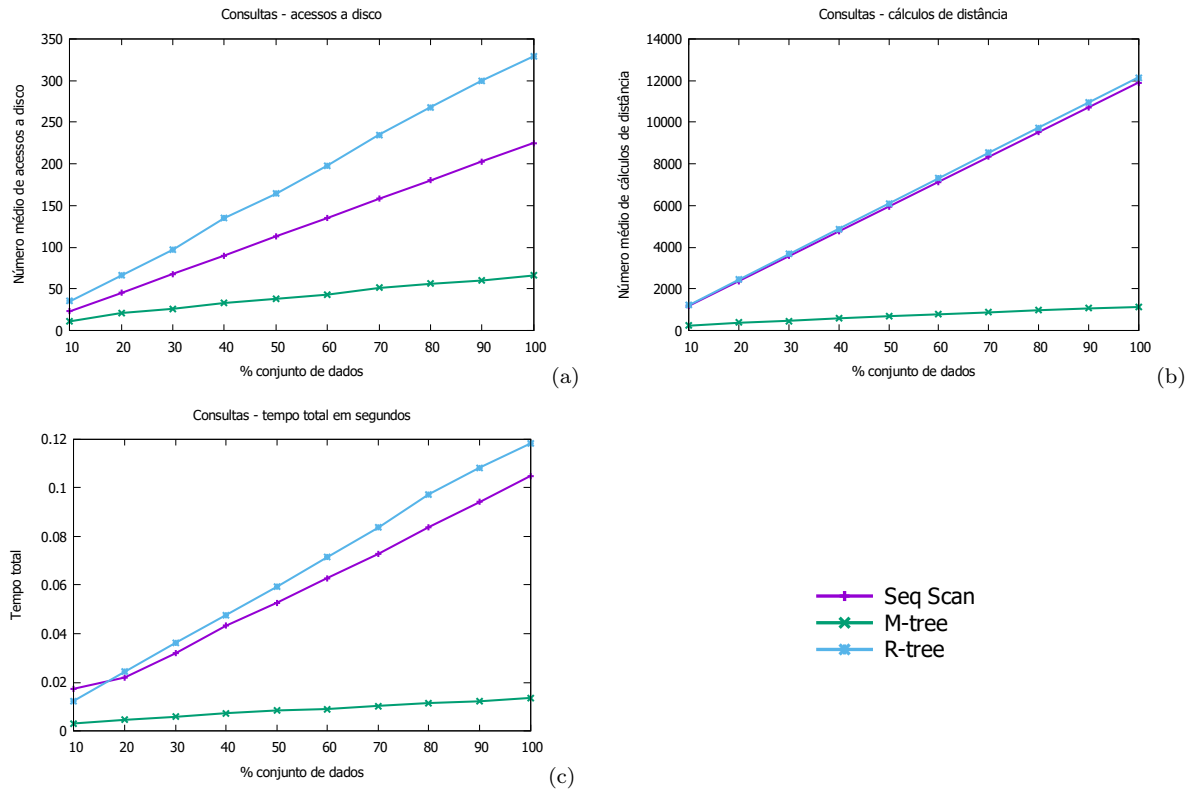


Figura 7 – Consultas com a base *Eigenfaces* utilizando páginas de 4 KB

O método R-Tree apresentou o pior desempenho dos três, ficando atrás até mesmo do método de busca sequencial, sendo 23,36% mais lento, fazendo 36,74% menos acessos ao disco e 4,88% menos cálculos de comparação.

Ao utilizar páginas de 4 KB, assim como na construção, todos os métodos diminuíram a quantidade de acessos ao disco (método sequencial em 50,87%, R-Tree em 54,56% e M-Tree em 51,47%). Em relação a quantidade de cálculo de distâncias, o método sequencial se manteve (já que percorre toda a base) e os métodos R-Tree e M-Tree tiveram uma leve redução, 2,87% e 8,97% respectivamente. Diferentemente da construção, na execução da consulta a utilização de páginas de 4 KB se mostrou mais rápida em relação às consultas utilizando páginas de 2 KB. A consulta utilizando páginas de disco de 4 KB foi 17,92% e 15,97% mais rápida para os métodos R-Tree e M-Tree respectivamente do que a consulta ao utilizar páginas de 2 KB.

Durante as consultas utilizando páginas de 4 KB, o método M-Tree se manteve com desempenho melhor. A busca dos 10 vizinhos mais próximos foi executada em 88,54% menos tempo do que a consulta utilizando o método R-Tree e 87,08% menos tempo do que a consulta pelo método sequencial. Além disso, o método M-Tree fez 79,94% menos acessos ao disco e 90,73% menos cálculos de distância do que o método R-Tree e 70,67% menos acessos ao disco e 90,54% menos cálculos de distância do que o método sequencial.

Ao aumentar o tamanho da página de disco para 4 KB o método R-Tree teve uma melhora, porém continuou com o pior desempenho dos três métodos, utilizando 11,29% mais tempo, 31,61% mais acessos ao disco e 2,07% mais cálculos de distância do que o método sequencial.

3.2 Pendigits

Nas Figuras 8 e 9 são apresentados os gráficos de construção da árvore de indexação da base de dados *Pendigits* que possui 10.990 objetos com 16 atributos numéricos, utilizando páginas de disco de 2 KB e 4 KB.

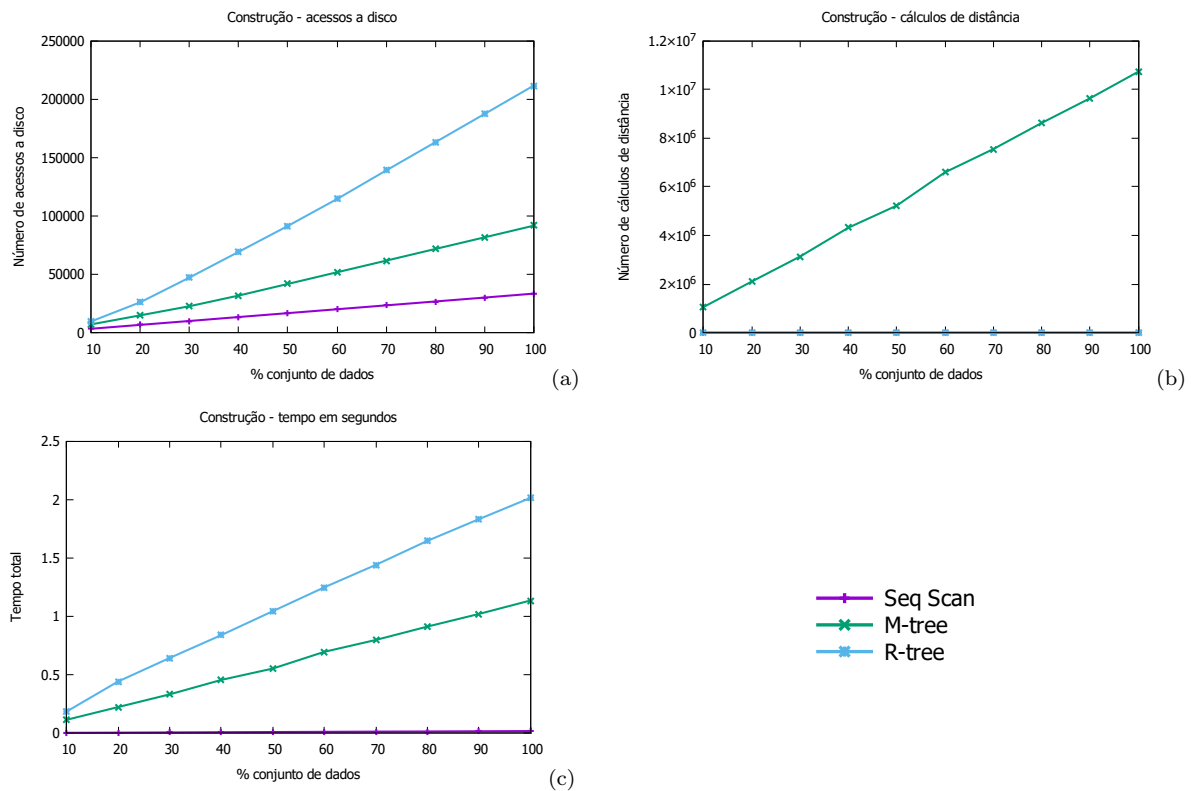


Figura 8 – Construção com a base *Pendigits* utilizando páginas de 2 KB

O método sequencial foi o mais eficiente dos três durante a construção da base, sendo 99,18% mais rápido do que o método R-Tree e 98,55% mais rápido do que o método M-Tree. Além disso, o método sequencial fez 84,22% menos acessos ao disco do que o método R-Tree e 63,56% menos acessos ao disco do que o método M-Tree. O método M-Tree se mostra mais eficiente do que o método R-Tree, sendo 43,64% mais rápido e fazendo 56,71% menos acessos ao disco.

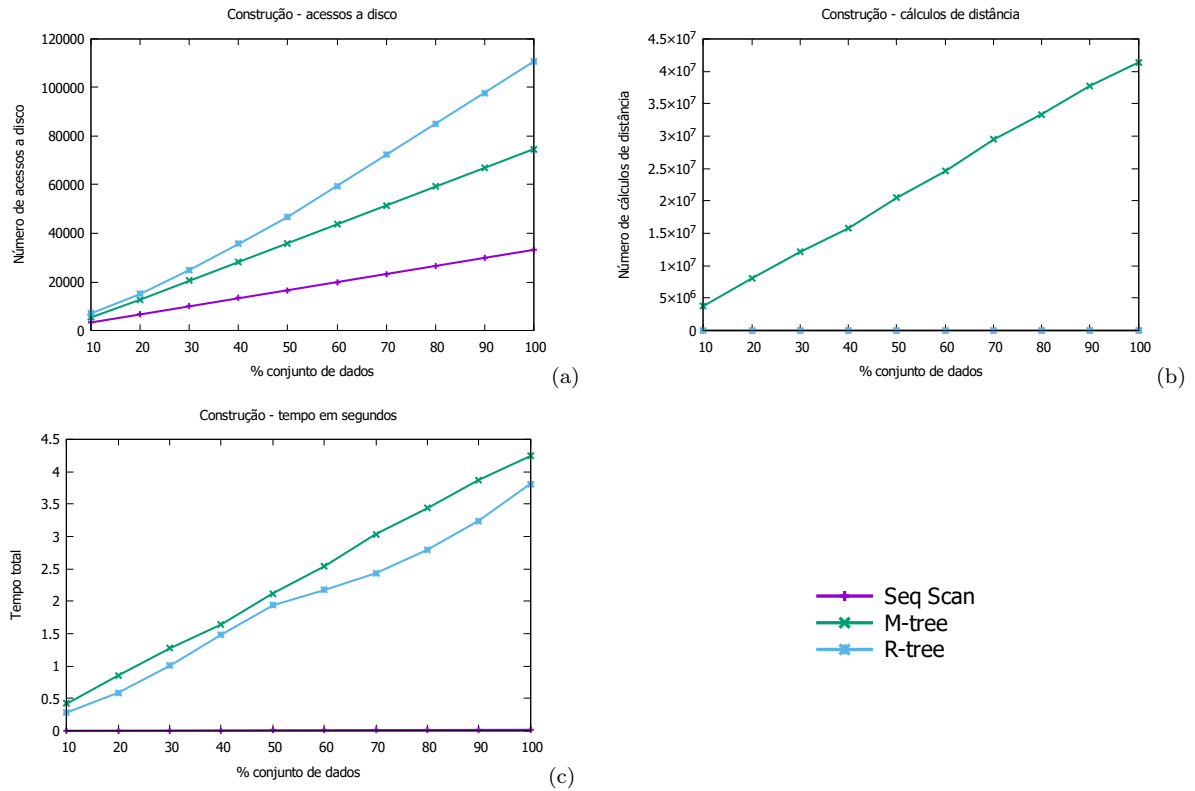


Figura 9 – Construção com a base *Pendigits* utilizando páginas de 4 KB

O aumento do tamanho das páginas do disco novamente se mostrou maléfico para o desempenho dos métodos durante a construção. O método sequencial teve uma pequena redução no número de acessos ao disco (0,64%) porém utilizando páginas de 2 KB o método era 2,03% mais rápido. O método R-Tree mesmo tendo uma redução de 47,72% no número de acessos ao disco, se mostrava 43,98% rápido ao utilizar páginas de 2 KB. O método M-Tree foi o que teve maior aumento no tempo. Mesmo fazendo 18,58% menos acessos ao disco ao utilizar páginas de 4 KB, utilizando páginas de 2 KB ele se mostrava 73,21% mais rápido e fazia 74,03% menos cálculos de distância.

Utilizando páginas de 4 KB o método sequencial se manteve como o mais eficiente, sendo 99,53% mais rápido e fazendo 70,01% menos acessos ao disco do que o método R-Tree além de ser 99,60% mais rápido e fazendo 55,52% menos acessos ao disco do que o método M-Tree. Assim como aconteceu na base *Eigenfaces*, o método M-Tree faz menos acessos ao disco (32,57%) porém devido ao grande aumento no tempo de construção do método M-Tree, o método R-Tree se mostra 15,16% mais rápido do que o método M-Tree.

Ao executar a busca de 10 vizinhos utilizando páginas de disco de 2 KB com os dados apresentados na Figura 10, o método M-Tree foi o mais eficiente dos três, sendo 46,50% mais rápido, fazendo a mesma quantidade de acessos ao disco e fazendo 20,12% menos cálculos de distância do que o método R-Tree e sendo 85,43% mais rápido, fazendo 68,79% menos acessos ao disco e 88,33% menos cálculos de distância do que o método

sequencial. Diferentemente da base anterior, o método R-Tree se mostra mais eficiente do que o método sequencial, sendo 72,77% mais rápido, fazendo 68,79% menos acessos ao disco e fazendo 88,33% menos cálculos de distância.

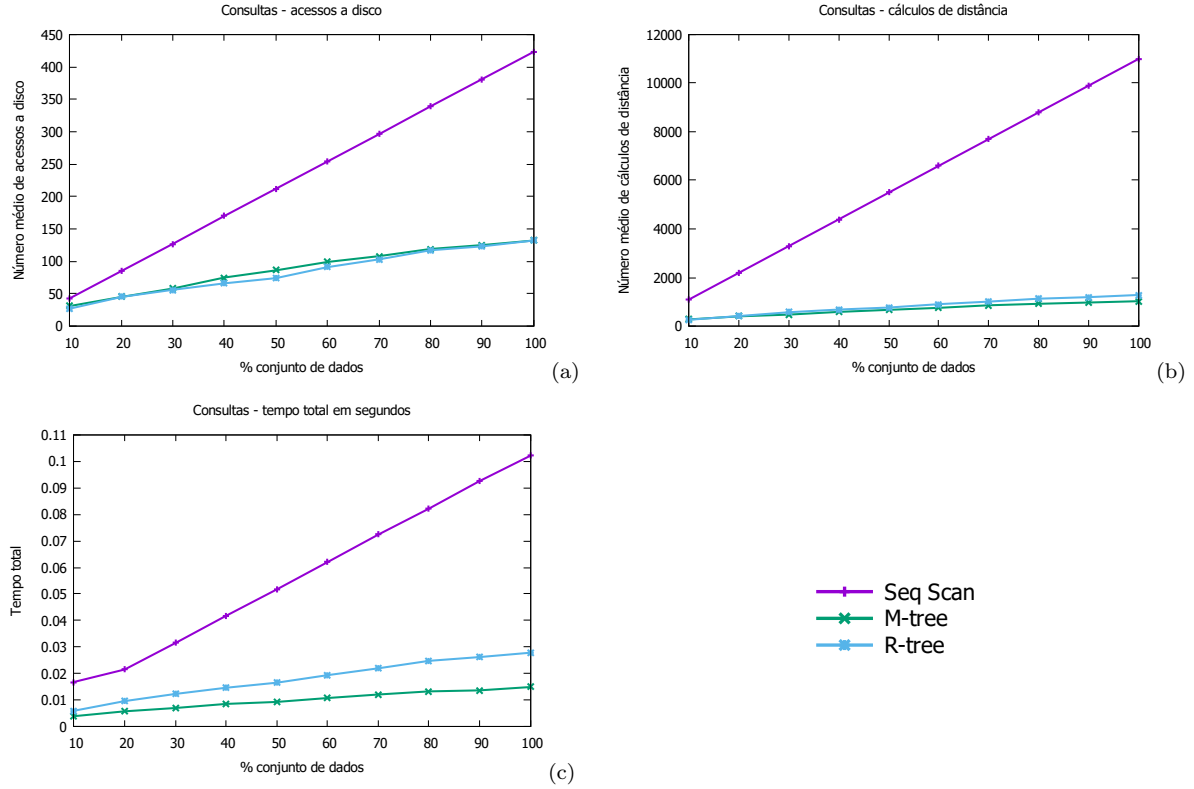


Figura 10 – Consultas com a base *Pendigits* utilizando páginas de 2 KB

Ao aumentar a página do disco para 4 KB (apresentado na Figura 11) e realizar as consultas novamente, o método M-Tree (diferentemente da consulta ao utilizar páginas de 2 KB) teve uma melhora em todos os itens avaliados, sendo 17,02% mais rápido, fazendo 47,72% menos acessos ao disco e 3,61% menos cálculos de distância. O aumento do tamanho da página do disco também melhorou o desempenho do método sequencial que fez 50,83% menos acessos ao disco e foi 6,07% mais rápido. Para o método R-Tree, o aumento no tamanho da página do disco somente reduziu o número de acessos ao disco em 28,03%, mas o método R-Tree foi 7,15% mais rápido e fez 29,68% menos cálculos de distância utilizando páginas de 2 KB.

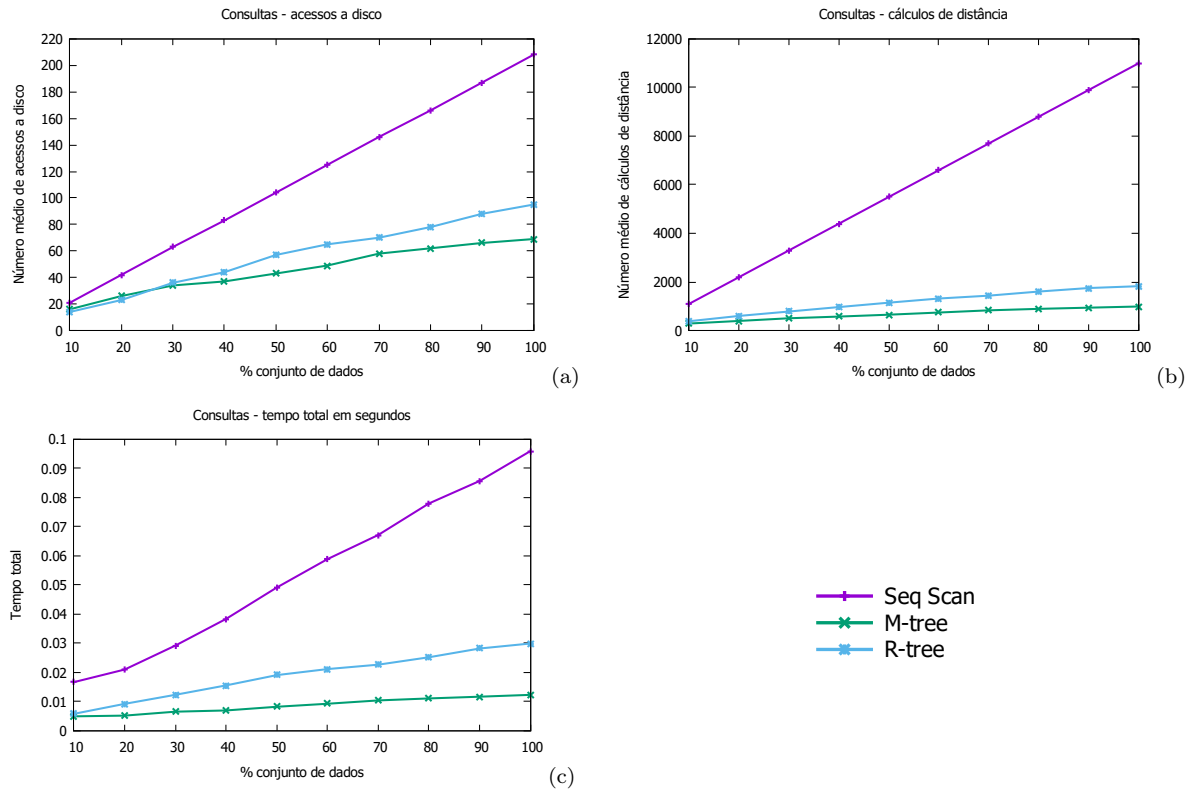
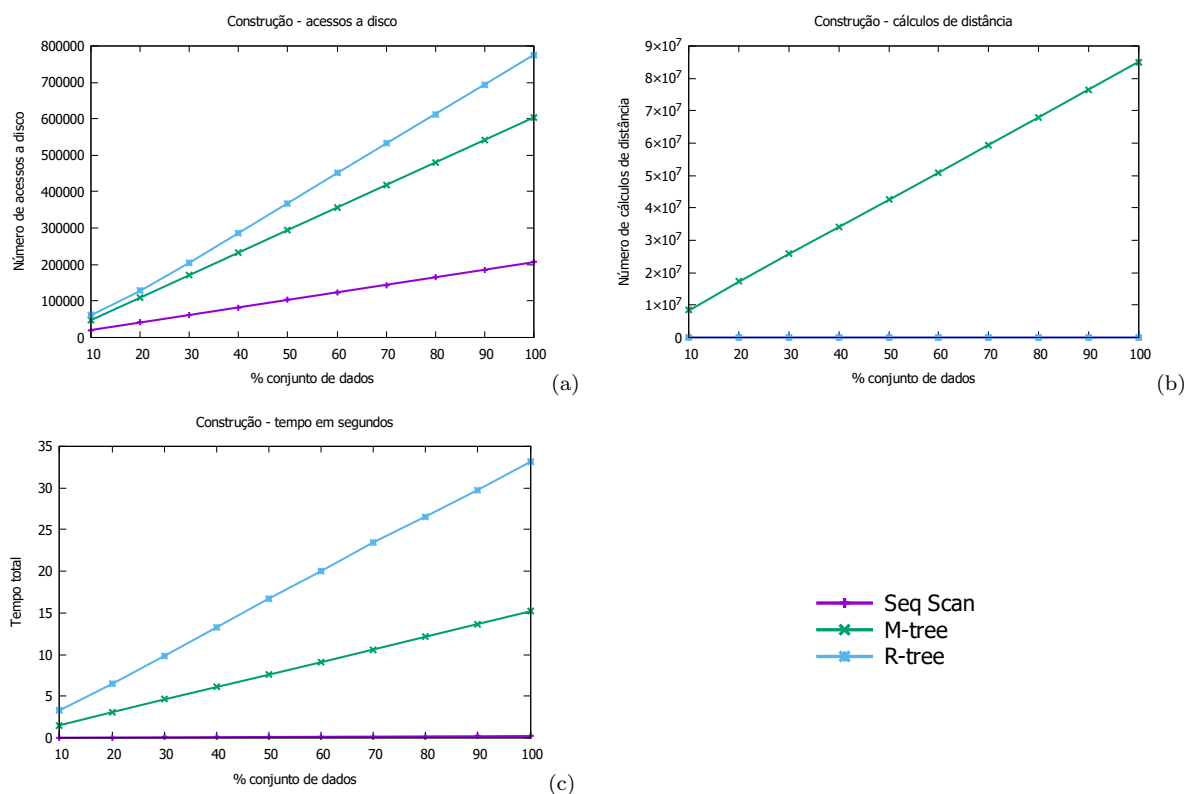
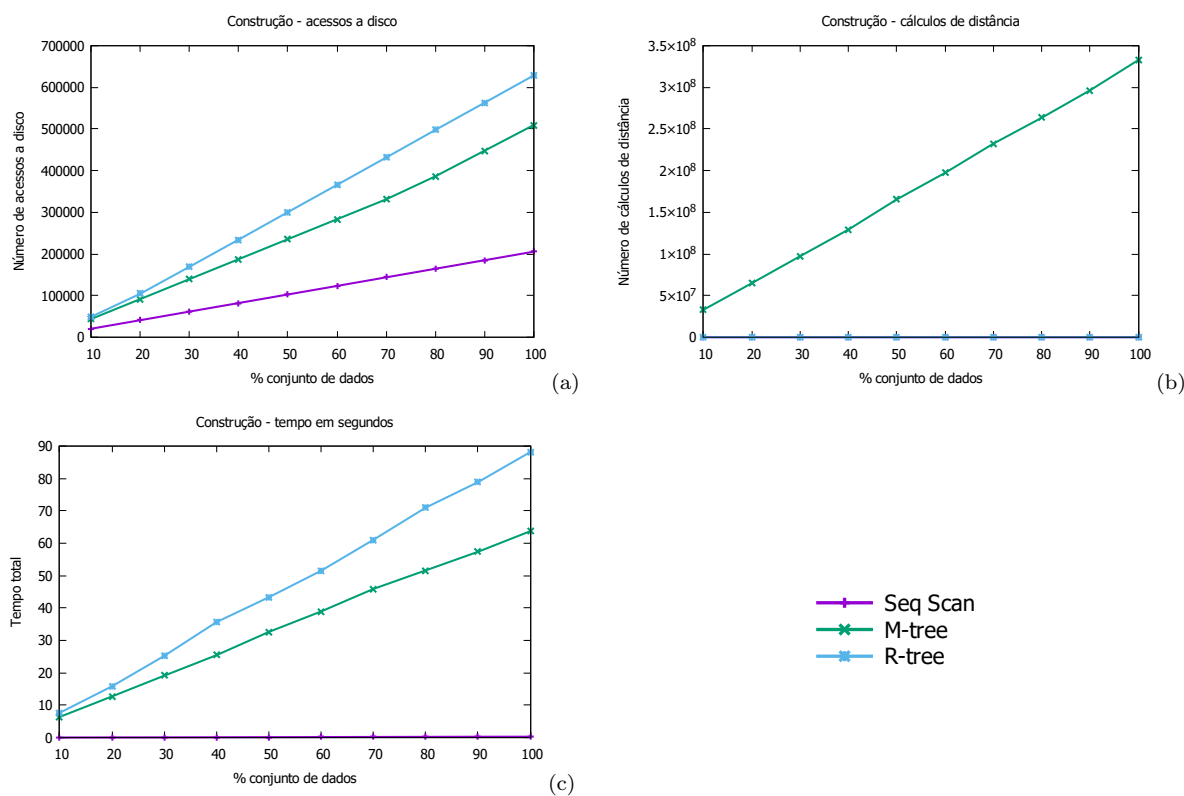


Figura 11 – Consultas com a base *Pendigits* utilizando páginas de 4 KB

Avaliando os métodos entre si, ao utilizar páginas de disco de 4 KB o método M-Tree se manteve mais eficiente, sendo 58,78% e 87,13% mais rápido do que os métodos R-Tree e sequencial respectivamente. O método M-Tree também fez 27,37% e 66,82% menos acessos ao disco do que os métodos R-Tree e sequencial respectivamente além de fazer 45,86% e 91,02% menos cálculos de distância do que os métodos R-Tree e sequencial respectivamente. O método R-Tree continuou mais eficiente do que o sequencial para a consulta, sendo 68,77% mais rápido, fazendo 54,33% menos acessos ao disco e fazendo 83,41% menos cálculos de distância.

3.3 Corel

As Figuras 12 e 13 apresentam a construção da base *Corel*, que possui 32 atributos numéricos e possui 68.040 objetos, apresentando os cálculos de distância, número de acessos ao disco e o tempo para a construção da base utilizando páginas de disco de 4 KB e 8 KB respectivamente para os métodos de acesso sequencial, R-Tree e M-Tree.

Figura 12 – Construção com a base *Corel* utilizando páginas de 4 KBFigura 13 – Construção com a base *Corel* utilizando páginas de 8 KB

Na construção da base utilizando tamanhos de disco de 4 KB é possível observar que mesmo não fazendo cálculos de distância, a construção da R-Tree é 54,16% mais demorada que a construção da M-Tree. Também é possível perceber que ao construir a R-Tree são feitos 22,14% mais acessos ao disco em relação a M-Tree. Como esperado, por não criar uma árvore de indexação, o método sequencial é o mais eficiente dos três gastando 0,64% do tempo em relação a R-Tree e 1,40% do tempo em relação a M-Tree, além de fazer 73,36% menos acessos ao disco em relação a R-Tree e 65,78% menos acessos ao disco em relação a M-Tree.

Utilizando páginas de disco de 8 KB percebe-se que todos os métodos se tornaram mais lentos na fase de construção quando comparamos a utilização de páginas de disco de 4 KB.

O método M-Tree foi o que teve maior aumento no tempo. Mesmo fazendo 15,64% menos acessos ao disco teve um aumento de 76,16% em relação ao tempo e fez 74,44% mais cálculos de distância. O método R-Tree também reduziu o número de acessos ao disco ao utilizar páginas de disco de 8 KB, passando a fazer 18,95% menos acessos ao disco. Este método também apresentou um aumento no tempo de construção, utilizando 62,41% mais tempo para ser executado. O método sequencial gastou 26,48% mais tempo para ser executado e fez 0,57% menos acessos ao disco.

Comparando os três métodos ao final da construção utilizando páginas de disco de 8 KB, é possível visualizar que ambos métodos de indexação (R-Tree e M-Tree) se degradaram. O método M-Tree foi o que teve o maior aumento de tempo com o aumento do tamanho de página de disco, mas mesmo assim se manteve mais eficiente que o método R-tree, fazendo 18,97% menos acessos ao disco e sendo 27,71% mais rápido.

Já o método sequencial se manteve mais eficiente, utilizando 0,33% do tempo gasto pela R-Tree e 0,45% do tempo gasto pela M-Tree. Além de fazer 67,31% menos acessos ao disco do que a R-Tree e 59,66% menos acessos ao disco do que a M-Tree.

Conforme apresentado na Figura 14, ao executar a busca de 10 vizinhos utilizando páginas de disco de 4 KB, o método M-Tree foi o mais eficiente. O método M-Tree foi 81,04% mais rápido do que o método sequencial e 85,59% mais rápido do que o método R-Tree além de fazer 57,94% menos acessos ao disco do que o método sequencial, 78,21% menos acessos ao disco do que o método R-Tree, 86,69% menos cálculos de distância do que o método sequencial e 84,04% menos cálculos de distância do que o método R-Tree. O método sequencial se mostrou mais eficiente do que o método R-tree sendo 23,97% mais rápido, fazendo 48,19% menos acessos ao disco e 16,60% menos cálculos de distância.

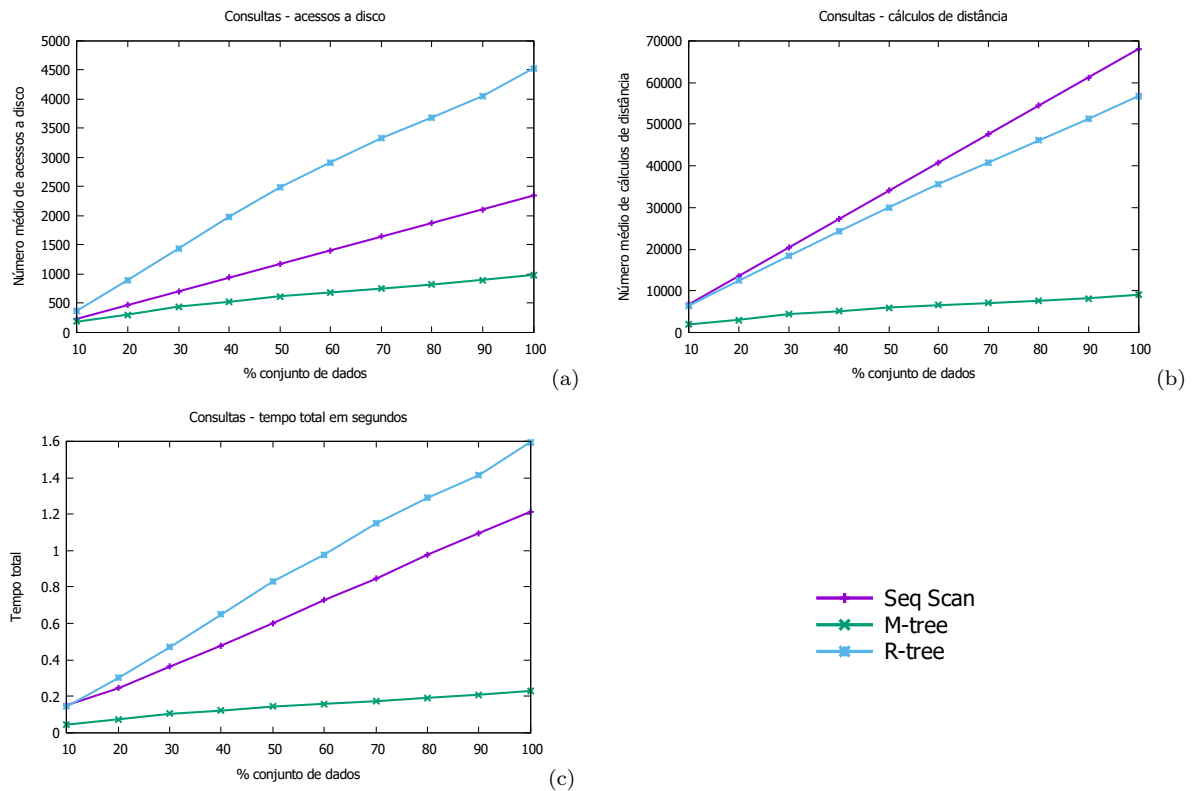


Figura 14 – Consultas com a base *Corel* utilizando páginas de 4 KB

Ao executar a consulta utilizando páginas de disco de 8 KB (Figura 15) todos os métodos apresentaram melhora em relação ao número de acessos ao disco e ao tempo. O método sequencial foi 3,37% mais rápido e fez 15,98% menos acessos ao disco. O método R-tree foi 12,37% mais rápido e fez 39,85 menos acessos ao disco. O método M-tree foi 27,96% mais rápido e fez 55,42% menos acessos ao disco. O método M-tree ainda teve 17,23% menos cálculos de distância enquanto o método R-tree aumentou o número de cálculos de distância em 1,4%.

Utilizando páginas de disco de 8 KB o método M-Tree foi 85,87% mais rápido do que o método sequencial e 88,15% mais rápido do que o método R-Tree, fez 62,52% menos acessos ao disco do que o método sequencial, 83,85% menos acessos ao disco do que o método R-Tree, 88,99% menos cálculos de distância do que o método sequencial e 86,98% menos cálculos de distância do que o método R-Tree. O método sequencial se manteve mais eficiente do que o método sequencial sendo 16,16% mais rápido e fazendo 56,91% menos acessos ao disco, porém o método sequencial fez 15,42% mais cálculos de distância.

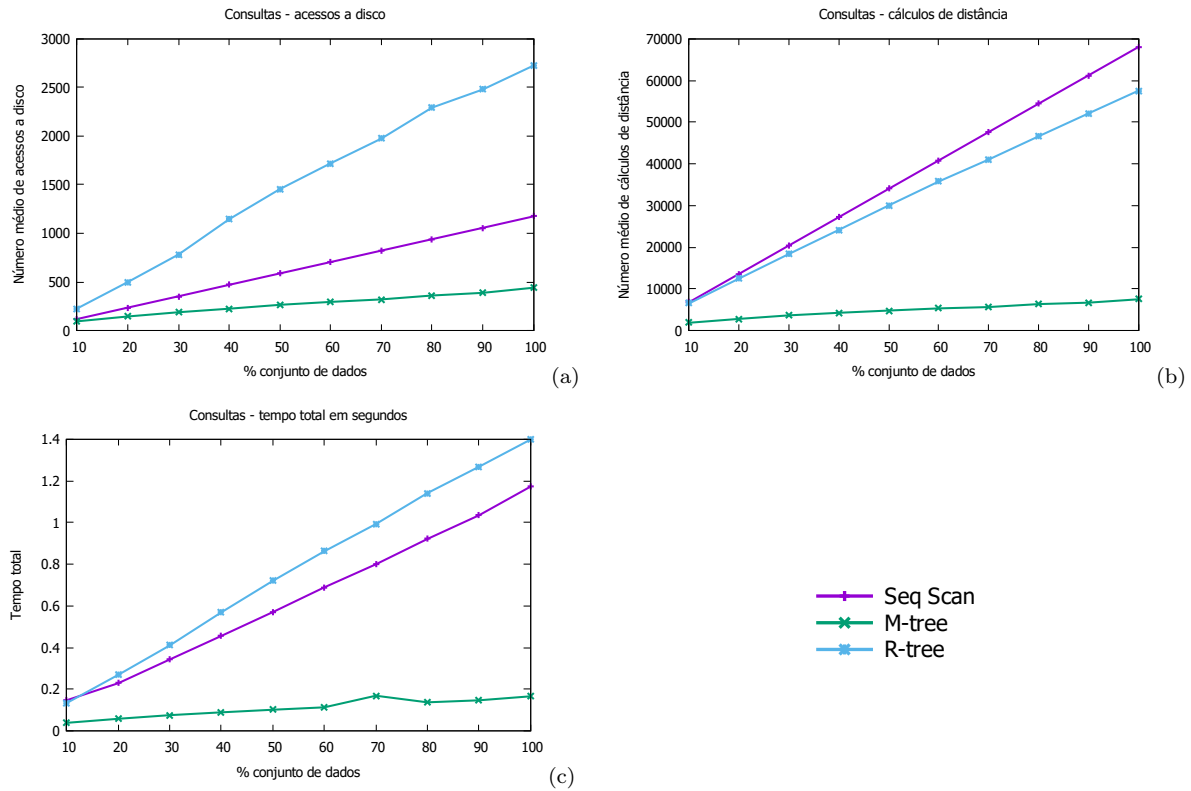


Figura 15 – Consultas com a base *Corel* utilizando páginas de 8 KB

3.4 Coverttype

As Figuras 16 e 17 apresentam os gráficos de construção da árvore de indexação da base de dados *Coverttype* que possui 581.012 objetos com 54 atributos numéricos, utilizando páginas de disco de 4 KB e 8 KB.

A construção da base utilizando o método sequencial foi a mais eficiente. Utilizando páginas de disco de 4 KB o método sequencial foi 99,65% mais rápido e fez 89,71% menos acessos ao disco do que o método R-Tree e foi 99,03% mais rápido e fez 75,48% menos acessos ao disco do que o método M-Tree. O segundo método mais eficiente foi o M-Tree já que foi 63,47% mais rápido e fez 58,04% menos acessos ao disco do que o método R-Tree.

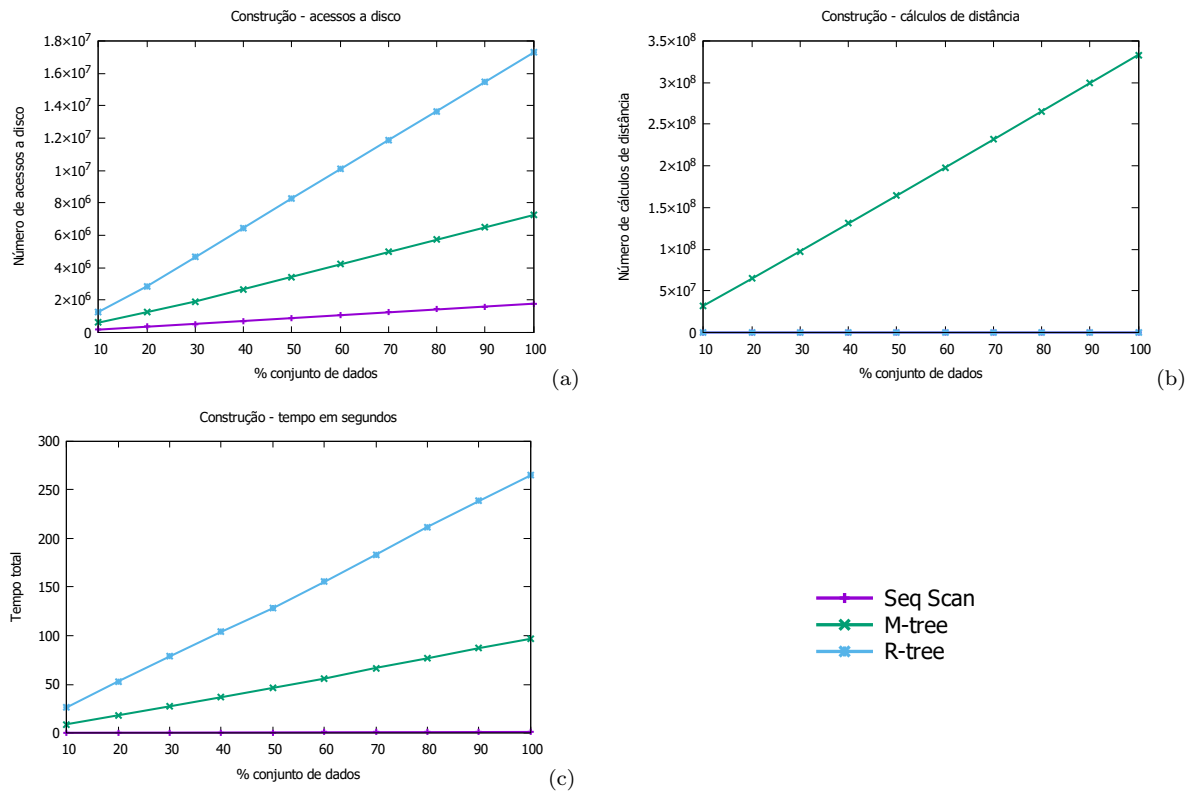


Figura 16 – Construção com a base *Covertime* utilizando páginas de 4 KB

Ao aumentar o tamanho da página do disco para 8KB, todos os métodos reduziram o número de acessos ao disco. O método sequencial fez 0,99% menos acessos ao disco, o método R-Tree fez 28,04% e o método M-Tree fez 18,63% menos acessos ao disco. Apesar de reduzir o número de acessos ao disco, todos os métodos foram mais lentos. Utilizando páginas de 4KB método sequencial foi 41,19% mais rápido, o método R-Tree foi 67,51% mais rápido e o método M-Tree foi 70,62% mais rápido. Além disso, ao utilizar páginas de 4KB o método M-Tree fez 72,21% menos cálculos de distância.

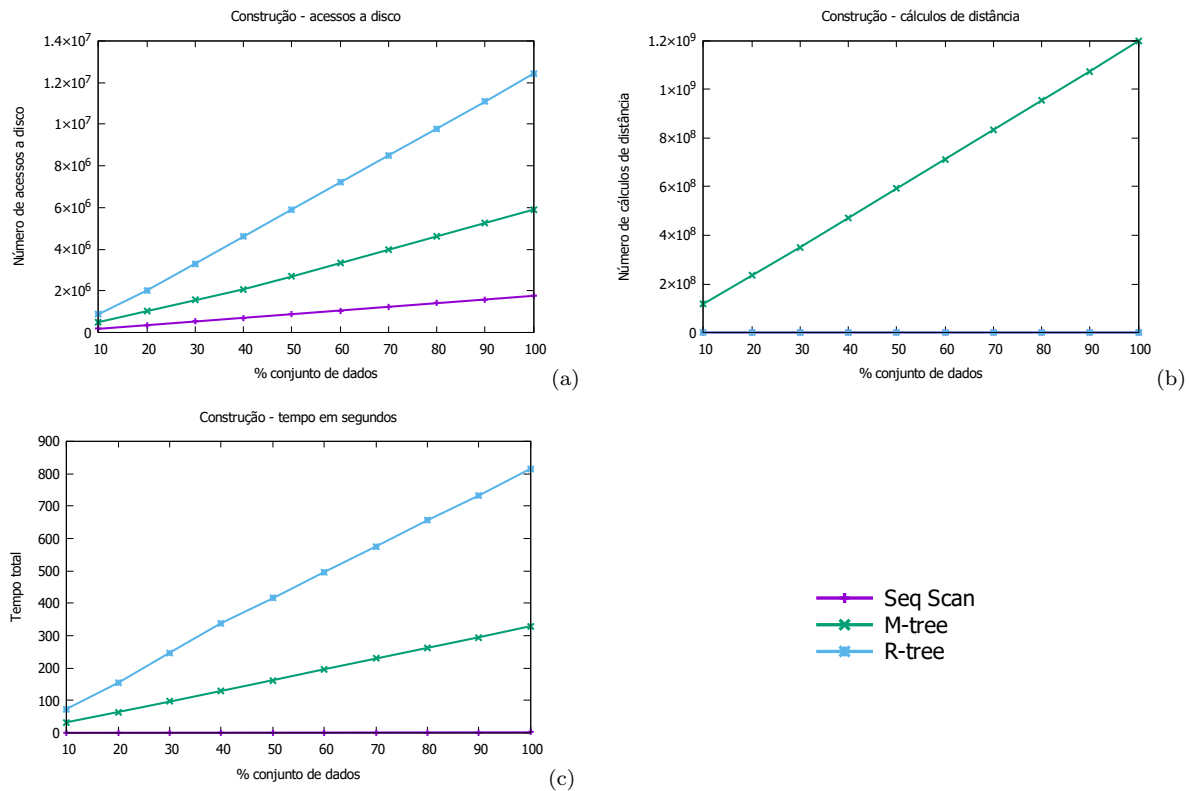


Figura 17 – Construção com a base *Covertime* utilizando páginas de 8 KB

Utilizando páginas de disco de 8 KB o método sequencial se manteve mais eficiente já que foi 99,80% mais rápido do que o método R-Tree e 99,51% mais rápido do que o método M-Tree. Além disso o método sequencial fez 85,84% menos acessos ao disco do que o método R-Tree e 70,17% menos acessos ao disco do que o método M-Tree. O método M-Tree se manteve mais eficiente do que o método R-Tree, sendo 59,60% mais rápido e fazendo 52,55% menos acessos ao disco.

Ao executar a busca de 10 vizinhos utilizando páginas de disco de 4 KB o método M-Tree foi o mais eficiente conforme dados apresentados nas Figuras 18 e 19. O método M-Tree foi 97,78% mais rápido do que o método sequencial e 95,47% mais rápido do que o método R-Tree além de fazer 93,90% menos acessos ao disco do que o método sequencial, 90,61% menos acessos ao disco do que o método R-Tree, 98,46% menos cálculos de distância do que o método sequencial e 92,41% menos cálculos de distância do que o método R-Tree. O método R-Tree se mostrou mais eficiente do que o método sequencial sendo 51,38% mais rápido, fazendo 35,03% menos acessos ao disco e 80,04% menos cálculos de distância.

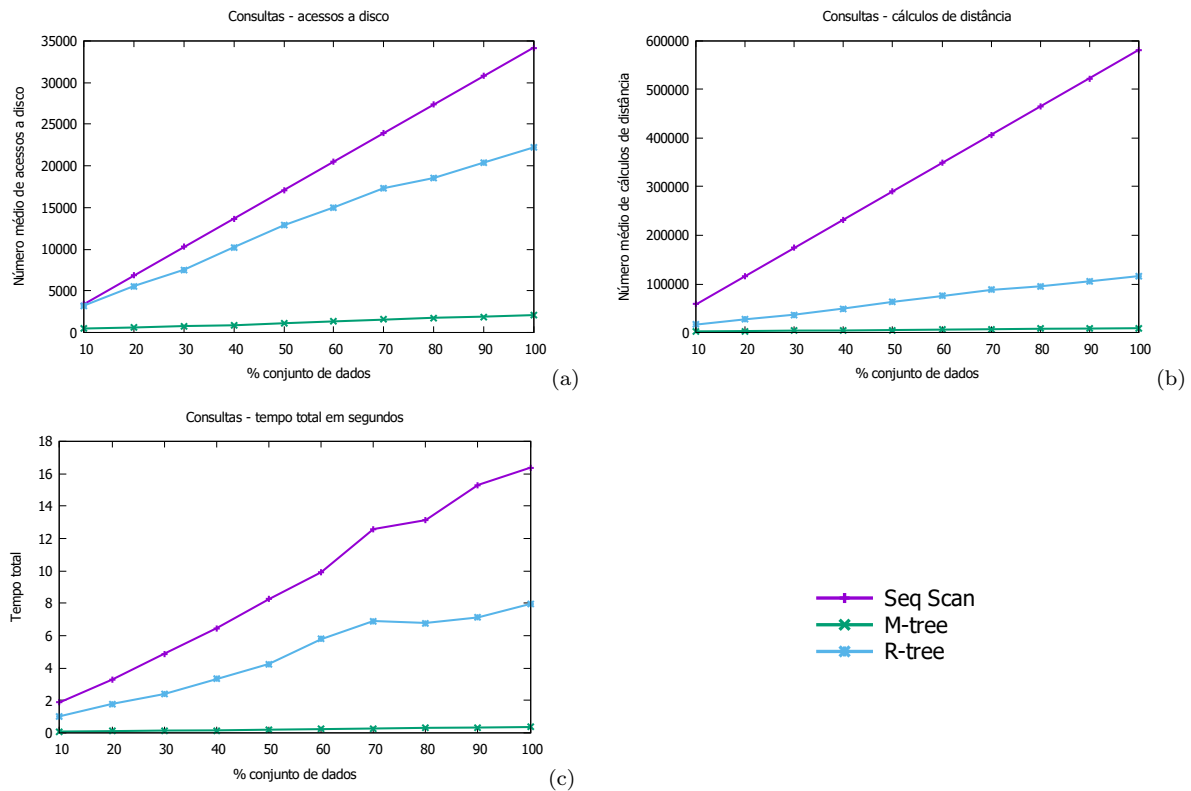


Figura 18 – Consultas com a base *Covertime* utilizando páginas de 4 KB

Ao executar a consulta utilizando páginas de disco de 8 KB o método M-Tree apresentou melhora em todos os pontos avaliados. A consulta executada no método M-Tree com páginas de disco de 8KB foi 21,28% mais rápida, foram feitos 51,92% menos acessos ao disco e 18,51% menos cálculos de distância do que quando foram utilizadas páginas de 4 KB. Para o método sequencial, o aumento do tamanho da página também se mostrou eficiente. A consulta foi 1,45% mais rápida e foram feitos 51,43% menos acessos ao disco. Para o método R-Tree o aumento do tamanho da página fez com que a consulta realizasse 25,65% menos acessos ao disco porém utilizando páginas de disco de 4 KB a consulta foi 3,44% mais rápida e foram feitos 15,63% menos cálculos de distância.

Utilizando páginas de disco de 8 KB o método M-Tree foi 98,24% mais rápido do que o método sequencial e 96,56% mais rápido do que o método R-Tree, fez 93,96% menos acessos ao disco do que o método sequencial, 93,93% menos acessos ao disco do que o método R-Tree, 98,77% menos cálculos de distância do que o método sequencial e 94,78% menos cálculos de distância do que o método R-Tree. O método R-Tree se manteve mais eficiente do que o método sequencial sendo 48,90% mais rápido, fazendo 0,54% menos acessos ao disco e 76,35% menos cálculos de distância.

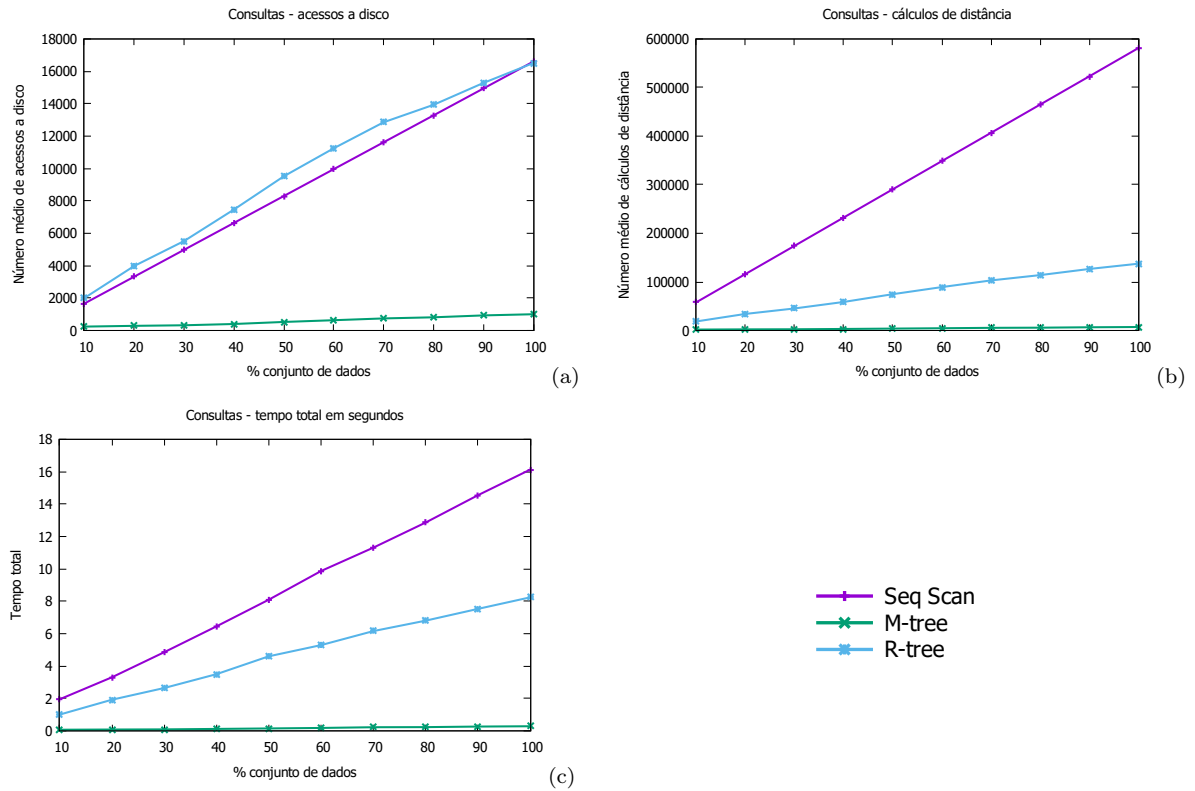


Figura 19 – Consultas com a base *Covertypes* utilizando páginas de 8 KB

3.5 Colors

Nas Figuras 20 e 21 são apresentados os gráficos de construção da árvore de indexação da base de dados *Colors* que possui 112.682 objetos com 112 atributos numéricos, utilizando páginas de disco de 16 KB e 32 KB.

Durante a construção utilizando páginas de disco de 16 KB, o método sequencial foi o mais eficiente. O método sequencial foi 99,92% mais rápido do que o método R-Tree, 99,52% mais rápido do que o método M-Tree, fez 85,69% menos acessos ao disco do que o método R-Tree e 65,52% menos acessos ao disco do que o método R-Tree. Entre os métodos de indexação M-Tree e R-Tree, o método M-Tree foi mais eficiente já que foi 82,83% mais rápido e fez 58,49% menos acessos ao disco do que o método R-Tree.

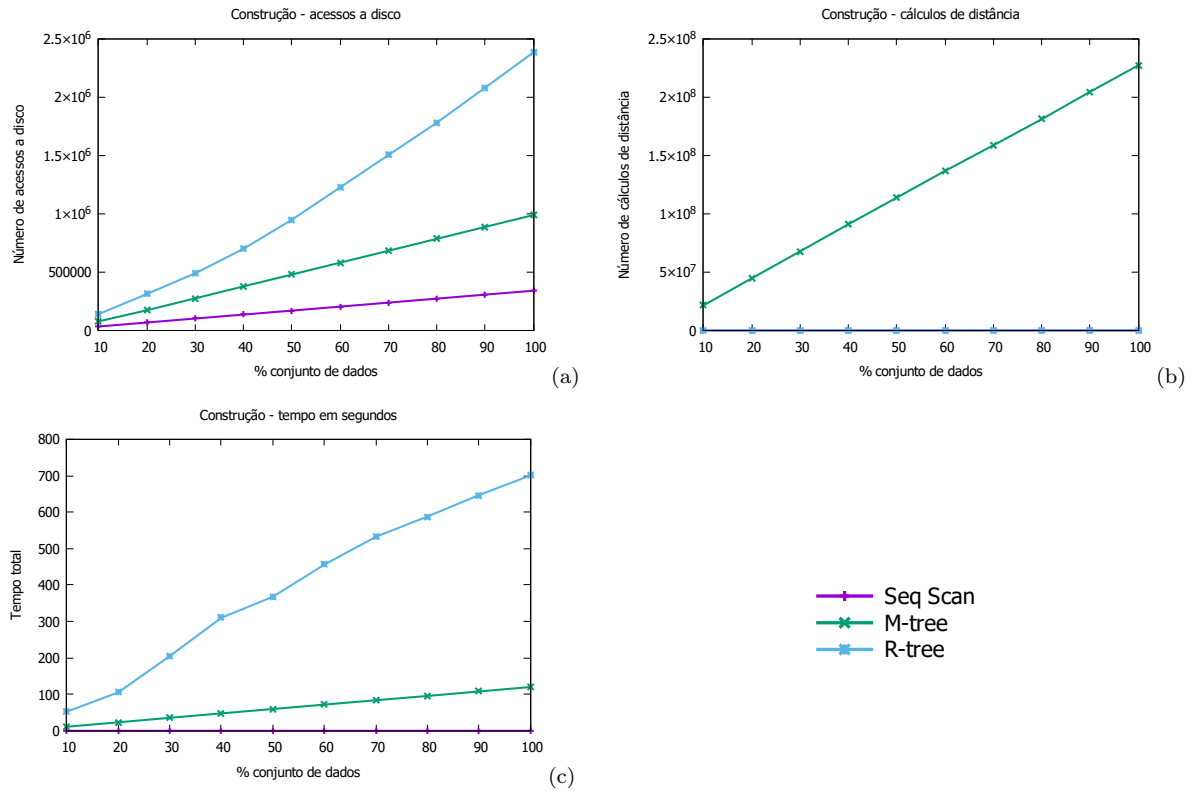


Figura 20 – Construção com a base *Colors* utilizando páginas de 16 KB

Ao aumentar o tamanho das páginas do disco para 32 KB, todos os métodos reduziram a quantidade de acessos ao disco. O método sequencial fez 0,48% menos acessos ao disco, o método R-Tree fez 32,62% menos acessos ao disco e o método M-Tree fez 20,63% menos acessos ao disco. Diferentemente do ganho em relação ao número de acessos ao disco, todos os métodos foram mais rápidos quando foram utilizadas páginas de disco de 16 KB, o método sequencial foi 45,21% mais rápido, o método R-Tree foi 66,85% mais rápido e o método M-Tree foi 73,62% mais rápido. Além disso o método M-Tree fez 73,70% menos cálculos de distância ao utilizar páginas de disco de 16 KB.

Utilizando páginas de 32 KB o método sequencial foi 99,95% mais rápido e fazendo 78,86% menos acessos ao disco do que o método R-Tree além de ser 99,76% mais rápido e fazendo 56,77% menos acessos ao disco do que o método M-Tree. O método M-Tree fez 51,11% menos acessos ao disco e foi 78,43% mais rápido do que o método R-Tree.

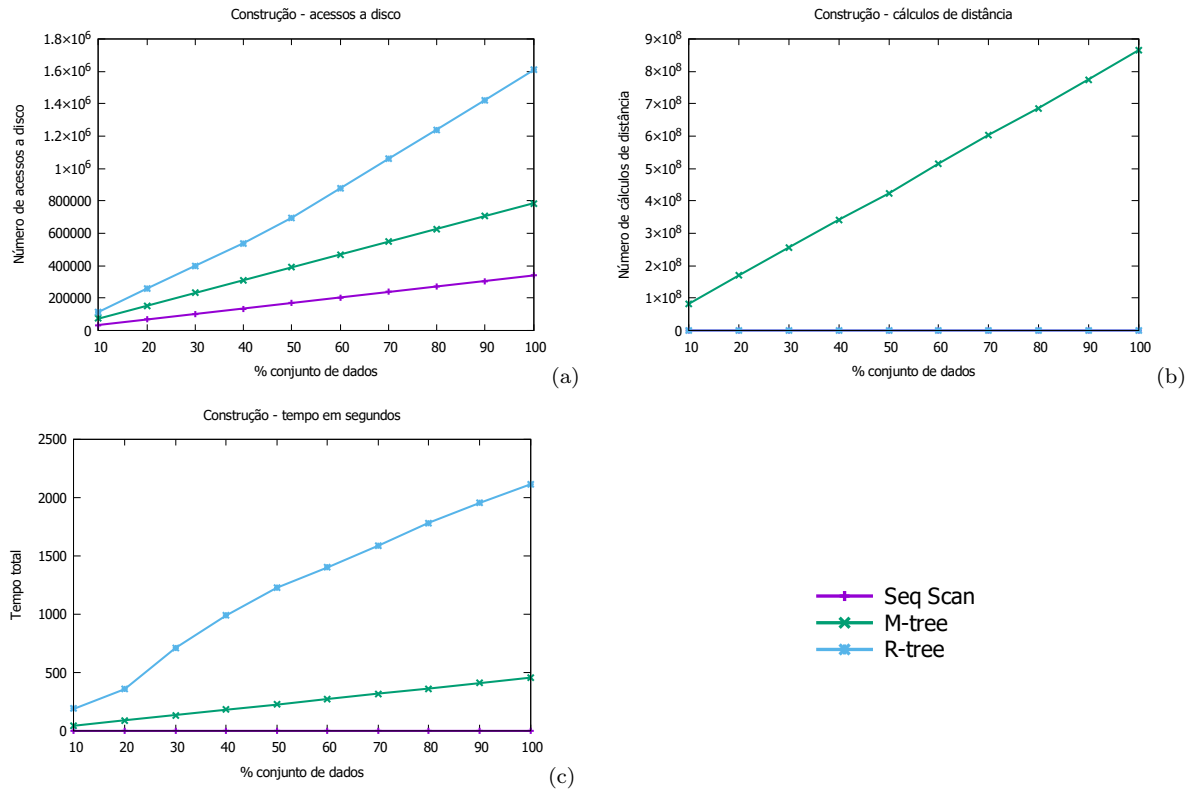


Figura 21 – Construção com a base *Colors* utilizando páginas de 32 KB

Durante a busca de 10 vizinhos na base utilizando páginas de 16 KB (apresentado na Figura 22) o método M-Tree foi 78,44% mais rápido, fez 46,89% menos acessos ao disco e fez 82,01% menos cálculos de distância do que o método sequencial. O método M-Tree também foi 89,17% mais rápido, fez 89,55% menos acessos ao disco e fez 82,72% menos cálculos de distância do que o método R-Tree. Para essa base o método sequencial foi mais eficiente do que o método R-Tree fazendo 3,95% menos cálculos de distância, 80,32% menos acessos ao disco e sendo 49,77% mais rápido.

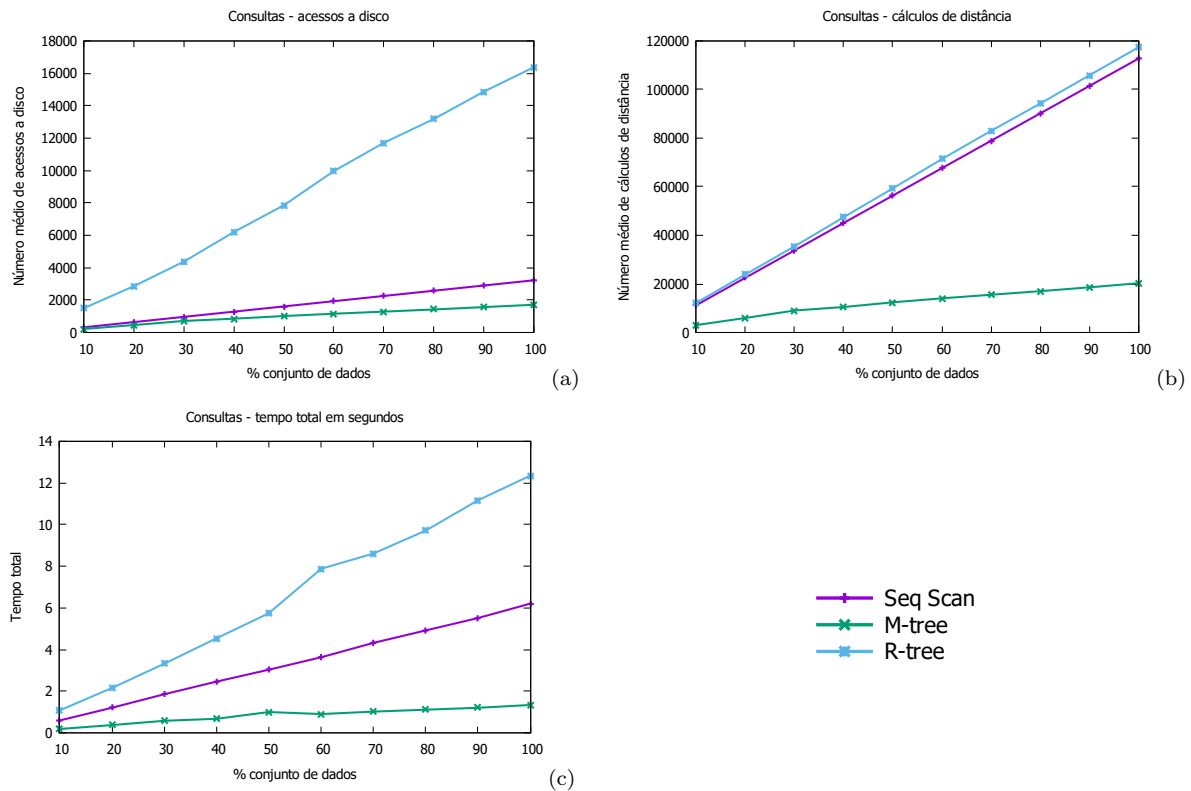


Figura 22 – Consultas com a base *Colors* utilizando páginas de 16 KB

Ao utilizar páginas de disco de 32 KB (apresentado na Figura 23), todos os métodos reduziram o número de acessos ao disco. O método sequencial fez 50,68% menos acessos ao disco, o método R-Tree fez 20,79% menos acessos ao disco e o método M-Tree fez 53,63% menos acessos ao disco. Ambos métodos sequencial e M-Tree foram mais rápidos ao utilizar páginas de 32 KB, o método sequencial foi 2,34% mais rápido e o método M-Tree foi 12,13% mais rápido, já o método R-Tree foi 6,93% mais rápido ao utilizar páginas de disco de 16KB. Os métodos R-Tree e M-Tree tiveram redução na quantidade de cálculos de distância ao utilizar páginas de disco de 32 KB. O método R-Tree fez 3,20% menos cálculos de distância e o método M-Tree fez 11,69% menos cálculos de distância.

Mais uma vez o método M-Tree se mostrou mais eficiente durante as buscas (agora utilizando páginas de disco de 32 KB). O método M-Tree foi 80,61% mais rápido, fez 50,06% menos acessos ao disco e fez 84,11% menos cálculos de distância do que o método sequencial. O método M-Tree foi 91,15% mais rápido, fez 93,88% menos acessos ao disco e fez 84,24% menos cálculos de distância do que o método R-Tree. O método sequencial continuou mais eficiente do que o método R-Tree fazendo 0,77% menos cálculos de distância, 87,75% menos acessos ao disco e sendo 54,34% mais rápido.

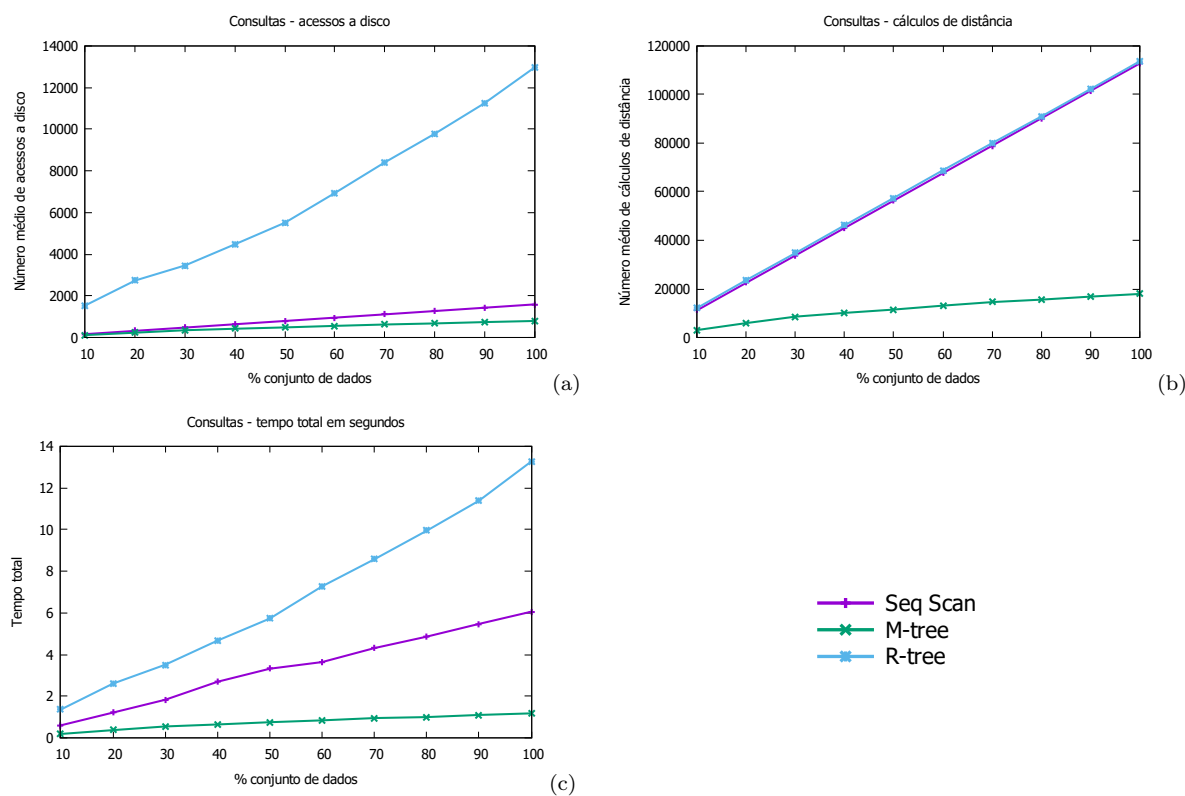


Figura 23 – Consultas com a base *Colors* utilizando páginas de 32 KB.

4 Conclusão

A indexação de dados complexos é um tema muito importante e que tem sido estudado nas últimas décadas. Cada vez mais o critério de similaridade é utilizado e indexar grandes bases com objetos que possuem várias dimensões. Mas é preciso tomar cuidado para entender qual método de acesso é mais vantajoso e se a busca é eficiente. Vários Sistemas de Gerenciamento de Bancos de Dados Relacionais (SGBDR) implementam alguma variação do método de acesso R-Tree. Este trabalho corrobora experimentalmente e empiricamente com a definição de limites na sua utilização, destacando que os efeitos da maldição da dimensionalidade podem variar com a distribuição dos dados.

Este trabalho apresentou a descrição de dois métodos de acesso utilizados para indexação de dados complexos (R-tree e M-tree) para a realização de buscas por similaridade e o algoritmo de buscas aos k-vizinhos mais próximos. Foram executados testes em 5 bases de dados calculando tempo, número de acessos ao disco e número de cálculos de distância para a construção dos índices e de consultas aos 10-vizinhos mais próximos utilizando esses índices.

Nos experimentos foi possível perceber que a maldição da dimensionalidade degrada a estrutura R-tree, que teve seu desempenho de consulta (em relação ao tempo) pior que o método sequencial em 4 das 5 bases avaliadas. Já o método M-tree se comportou muito bem, tendo melhor desempenho em relação a tempo durante as consultas em todas as bases testadas.

Durante a construção, como esperado, conforme o aumento das dimensões dos objetos das bases de dados, mais tempo foi gasto para construir as árvores de indexação (tanto na R-tree quanto na M-tree). Na maioria dos casos a M-tree foi mais rápida do que a R-tree.

Referências

- BAYER, R.; MCCREIGHT, E. M. Organization and maintenance of large ordered indices. *Acta Inf.*, v. 1, p. 173–189, 1972. Citado na página 12.
- CIACCIA, P.; PATELLA, M.; ZEZULA, P. *M-tree An Efficient Access Method for Similarity Search in Metric Spaces*. 1997. Citado 6 vezes nas páginas 4, 8, 14, 18, 19 e 20.
- COMER, D. *The Ubiquitous B-Tree*. 1979. Citado 3 vezes nas páginas 8, 12 e 13.
- FOLK, M. J.; ZOELLICK, B.; RICCARDI, G. *File Structures: An Object-Oriented Approach with C++*. United States of America: [s.n.], 1998. Citado 2 vezes nas páginas 12 e 13.
- GUTTMAN, A. *R-trees: a dynamic index structure for spatial searching*. 1984. Citado 4 vezes nas páginas 8, 13, 15 e 18.
- LICHMAN, M. *UCI Machine Learning Repository*. 2013. Disponível em: <<http://archive.ics.uci.edu/ml>>. Citado 2 vezes nas páginas 10 e 25.
- MANOLOPOULOS, Y. et al. *R-Trees - Theory and Applications*. Rio de Janeiro: Springer, 2006. Citado 6 vezes nas páginas 4, 8, 15, 16, 17 e 18.
- ROUSSOPOULOS, N.; KELLEY, S.; VINCENT, F. *Nearest neighbor queries*. 1995. Citado 3 vezes nas páginas 8, 22 e 24.
- SAMET, H. *Foundations of Multidimensional and Metric Data Structures*. [S.l.]: Morgan Kaufmann, 2006. ISBN 978-0-12-369446-1. Citado 2 vezes nas páginas 14 e 21.
- SCHNEIDER, M.; BEHR, T. Topological relationships between complex spatial objects. *ACM Trans. Database Syst.*, New York, NY, v. 31, n. 1, p. 39–81, 2006. Citado na página 15.
- SEIDL, T.; KRIEGEL, H.-P. *Optimal Multi-Step k-Nearest Neighbor Search*. 1998. Citado na página 22.
- VOLNYANSKY, I.; PESTOV, V. Curse of dimensionality in pivot based indexes. In: *International Workshop on Similarity Search and Applications (SISAP)*. [S.l.]: IEEE Computer Society, 2009. p. 39–46. ISBN 978-0-7695-3765-8. Citado na página 8.
- WEN, Y. *Similarity Search in Metric Spaces*. Waterloo, Ontario, Canada: [s.n.], 2004. Citado na página 22.