Um Algoritmo para Paginação de Árvores Binárias Baseado em Empacotamento Unidimensional

Rui Alberto Ecke Tavares and Elias Procópio Duarte Jr

Universidade Federal do Paraná
Departamento de Informática
Centro Politécnico, Caixa Postal 19081, 81531-990, Curitiba, PR

Resumo

Este trabalho apresenta um novo algoritmo para paginação de árvores binárias de pesquisa que frequentemente ocorrem em biologia compoutacional. O algoritmo visa reduzir o número de páginas visitadas em pesquisas e aumentar a taxa de preenchimento das páginas utilizadas. O algoritmo constrói a paginação ótima quando possível e, utilizando empacotamento unidimensional, apresenta uma política eficiente para o preenchimento das páginas de árvores binárias desbalanceadas. A complexidade computacional do algoritmo é apresentada. O algoritmo foi implementado e resultados experimentais, comparativos com outras estratégias de paginação são apresentados. A comparação mostrou que a abordagem proposta é a única que apresenta, simultaneamente, a quantidade média de páginas acessadas em pesquisas e a taxa de preenchimento das páginas próximas do ótimo.

1 Introdução

As árvores binárias são estruturas de dados que permitem a realização de busca ou pesquisa de forma eficiente [1]. Uma árvore binária pode atingir grandes dimensões, bem como ser utilizada para armazenar dados em memória secundária ou distribuídos pelos nodos de uma rede de computadores.

Muitas aplicações em biologia computacional envolvem o processamento de *strings* utilizando árvores binárias não balanceadas [2]. Tais árvores são originadas de seqüências biológicas que não podem ser balanceadas, por isso, árvores B não podem ser aplicadas [3]. Nestes casos, é necessário definir uma estratégia eficiente para o acesso aos dados da árvore, que são organizados em páginas. Uma página é utilizada para a transferência de dados em blocos da memória secundária para a primária, além do acesso remoto em redes de computadores, por intermédio de pacotes que possuem tamanho máximo pré-fixado.

Os algoritmos para tratamento da paginação de estruturas de dados são utilizados em diversos sistemas de informação. Com isso, os critérios para alocação dos dados nas páginas são essenciais para a eficiência destes sistemas. Neste trabalho apresenta-se um algoritmo para realizar a paginação de árvores binárias baseado em empacotamento unidimensional [4]. O algoritmo visa reduzir o número de páginas visitadas em buscas e, ao mesmo tempo, aumentar a taxa de preenchimento das páginas utilizadas.

O algoritmo proposto atinge o ideal de paginação quando a árvore binária é completa e o número de nodos é múltiplo do tamanho da página, conforme ilustrado na figura 1. Além disso, estabelece uma política eficiente de preenchimento de páginas, para a hipótese das árvores degeneradas.

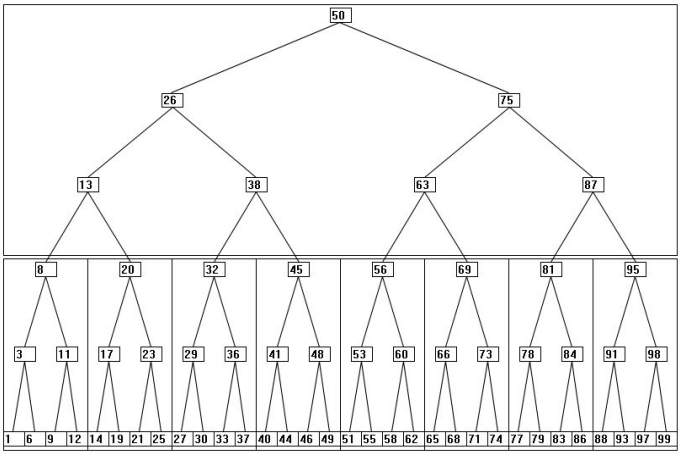

Figura 1: A alocação ideal de páginas.

O algoritmo inicia pela raiz da árvore a ser paginada alocando, sempre que possível, subárvores em páginas. Pode ser considerado guloso no sentido que procura armazenar em uma mesma página subárvores que a preencham completamente.

Quando sobram subárvores que não preenchem completamente uma página, ditas constituintes da *franja* da árvore, o algoritmo aloca essas subárvores em páginas utilizando empacotamento unidimensional. Desta forma, cada página é preenchida visando à otimização dos acessos durante uma pesquisa na árvore binária. O empacotamento unidimensional consiste em alocar um conjunto de subárvores num conjunto de páginas, conhecendo o espaço disponível nas páginas, de modo a minimizar o número de páginas utilizadas.

Uma estrutura de dados alternativa para organização de dados em memória secundária é a árvore B. Mostramos por intermédio de resultados experimentais que a estratégia proposta é equivalente às árvores B em termos do número de páginas acessadas em pesquisas. Por outro lado, a abordagem proposta produz uma taxa de preenchimento de páginas 30% superior às árvores B. Dessa forma, a quantidade de espaço necessário para armazenar e o número de bytes necessário para transferir uma árvore de um local remoto é 30% superior quando utilizamos nossa abordagem em comparação às árvores B.

O presente trabalho está dividido em seis seções, incluindo esta introdução. A seção 2 contém as definições preliminares. Na seção 3 é descrita a funcionalidade do algoritmo, bem como sua especificação formal e análise de complexidade. Na seção 4 são descritas as métricas utilizadas para mensurar o desempenho do algoritmo e apresentados resultados experimentais obtidos com a versão implementada do algoritmo. Na seção 5 são apresentados trabalhos relacionados. A seção 6 conclui o trabalho.

2 Definições Preliminares

As árvores binárias podem ser definidas recursivamente da seguinte maneira [1]:

i) A árvore binária T_0 de zero nodos é uma árvore binária.

ii) Uma árvore binária T_n de $n >= 1$ nodos é ordenada em uma tripla (T_{esq}, R, T_{dir}), onde R é o nodo simples dito raiz de T_n. T_{esq} e T_{dir} são as árvores binárias de esq e dir nodos, respectivamente ditas subárvores esquerda e direita da raiz ($esq >= 0$, $dir >= 0$ e $esq + dir = n - 1$).

Quando não é possível ou desejável manter todo o conjunto de nodos de uma árvore binária de pesquisa na memória principal, os nodos devem ser agrupados em páginas. Cada página é composta de células, cada nodo da árvore é armazenado em uma célula. Como o tempo necessário para acessar ou visitar uma página é predominantemente o tempo de transferência da página, o desempenho do algoritmo de manipulação da árvore é estritamente relacionado ao número de páginas transferidas.

Considere uma árvore binária com n elementos; uma página é definida como a unidade de armazenamento de nodos da árvore, podendo conter um máximo de p nodos. Os nodos da árvore são transferidos em páginas da memória secundária para a memória primária, ou entre duas máquinas ligadas em rede. Na pesquisa, os nodos são examinados da raiz até uma folha. O número de páginas visitadas que devem ser utilizadas para completar uma pesquisa deve ser o menor possível.

O algoritmo para paginação de árvores binárias proposto neste trabalho é dividido em duas fases. Na primeira fase são alocadas subárvores que preenchem completamente uma página. As subárvores remanescentes são ditas constituintes da *franja* da árvore. Na segunda fase é necessário que as subárvores da franja sejam alocadas utilizando empacotamento unidimensional.

O problema do empacotamento unidimensional pode ser definido como: dados uma constante C e uma lista finita de itens $L = p_1, p_2, \ldots, p_n$, onde cada item p, está associado a um valor $w(p_i)$ satisfazendo $0 < w(p_i) < C$, deseja-se encontrar o menor inteiro m tal que L possa ser particionada em m listas L_1, L_2, ... ,L_m onde cada lista L_i, satisfaz

$$w(L_i) = \sum_{p_j \in L_i} w(p_j) \leq C \text{ com } i=1..m.$$

Num típico problema de empacotamento unidimensional deseja-se particionar uma lista de itens em sublistas de maneira a minimizar o número de partições respeitando a capacidade de cada sublista.

Neste trabalho é proposto um algoritmo para alocar um conjunto de subárvores (sem restrição de precedência) em um conjunto de páginas, conhecendo o espaço disponível nas páginas, de modo a minimizar o número de páginas utilizadas. As subárvores, com tamanhos s_1, s_2 ,..., s_n devem ser alocadas em páginas de tamanho C. Um algoritmo de empacotamento unidimensional encontra a alocação que minimiza o número de páginas de tamanho C.

Por exemplo, na figura 2 a subárvore s_1 formada pelos nodos 3, 5, 7 e 12; a subárvore s_2 formada pelos nodos 36, 38 e 41; a subárvore s_3 formada pelos nodos 46, 49, 53 e 57; a subárvore s_4 formada

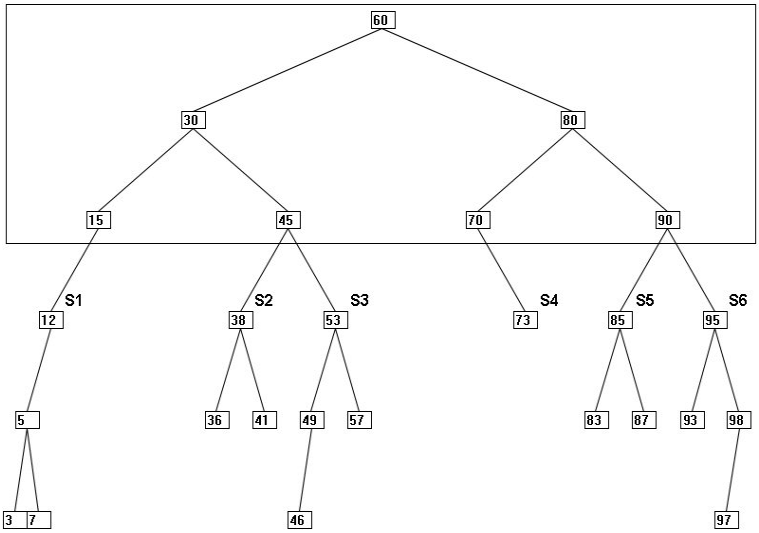

Figura 2: Exemplo de aplicação de empacotamento unidimensional.

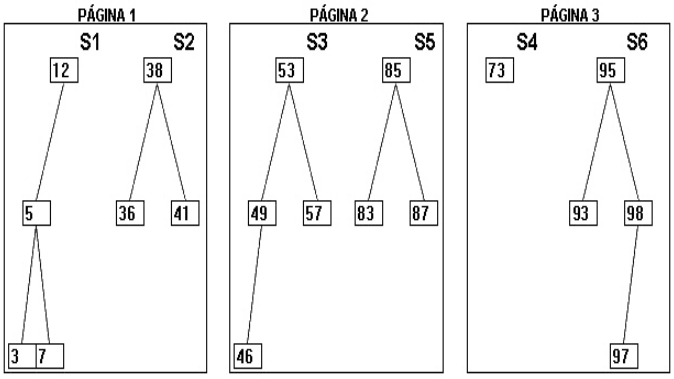

Figura 3: Paginação utilizando empacotamento unidimensional.

pelo nodo 73; a subárvore s_5 formada pelos nodos 83, 85 e 87 e a subárvore s_6 formada pelos nodos 93, 95, 97 e 98 devem ser alocadas no menor número possível de páginas. Observa-se que estamos diante de subárvores com tamanhos 4, 3, 4, 1, 3 e 4, respectivamente, e que devem ser alocadas em páginas de tamanho 7.

Uma solução ótima, que é obtida pelo algoritmo, produz uma alta taxa de preenchimento das páginas, alocando as subárvores s_1 e s_2 na página 1, as subárvores s_3 e s_5 na página 2 e as subárvores s_4 e s_6 na página 3, conforme se observa na figura 3.

O problema de empacotamento unidimensional encontra-se entre os problemas clássicos de Otimização Combinatória, sendo considerado NP-difícil no sentido forte. Entre as possíveis alternativas para a implementação do empacotamento unidimensional destaca-se a utilização de algoritmos aproximados ou heurísticos, ou seja, onde não existe a garantia de se encontrar a solução ótima, porém o tempo de execução é polinomial. Com isso, busca-se resolver o problema baseado em regras empíricas, cuja aplicação costuma ser dependente do tipo de problema. Diversos algoritmos aproximados são propostos na literatura [4]. Na implementação realizada neste trabalho utilizou-se um algoritmo guloso para solucionar o empacotamento das subárvores existentes na franja.

3 O Algoritmo Proposto

Esta seção apresenta o algoritmo proposto para paginação de árvores binárias de pesquisa. O algoritmo é aplicável quando o conjunto de informações a ser tratado é estático, as freqüências de acesso não são conhecidas e o armazenamento é remoto ou secundário.

3.1 Funcionalidade do Algoritmo

O algoritmo proposto neste trabalho é um paginador de árvores binárias que visa aumentar o grau de parentesco dos nodos que serão armazenados numa mesma página. O algoritmo atinge o ideal de paginação quando a árvore binária é completa e o número de nodos é múltiplo do tamanho da página. Além disso, estabelece uma política eficiente de preenchimento de páginas, para a hipótese das árvores degeneradas.

Antes de descrever o algoritmo é necessário definir a noção de nodo *patriarca*. Um patriarca é um nodo raiz de uma subárvore que deverá ser alocado em uma página específica, bem como seus descendentes, tantos quantos couberem na página.

Considere que uma página armazena até x níveis ou gerações com relação a um nodo de uma árvore binária, onde x é um número natural. O algoritmo inicia pela raiz da árvore a ser paginada, considerando esta raiz o patriarca da primeira página. Em seguida, são armazenadas $x-1$ gerações do patriarca nesta página. Todos os nodos da geração seguinte não podem ser armazenados aí, pois não há espaço disponível. Estes nodos serão, cada um, o patriarca de uma nova página. Em cada uma destas páginas são alocadas $x-1$ gerações do patriarca, e assim sucessivamente, até que todos os nodos tenham sido armazenados adequadamente.

Quando a árvore não é completa, isto é, quando é degenerada, ou quando o número de nodos não é múltiplo de 2^x-1, sobram espaços em uma ou mais páginas. O algoritmo adota uma política eficiente para armazenar as subárvores pertencentes à franja, aplicando empacotamento unidimensional para alocar as subárvores em páginas.

3.2 Especificação do Algoritmo

O algoritmo proposto utiliza duas estruturas de dados chamadas SQ e FL, descritas a seguir. A estrutura de dados SQ, iniciais de Stack e Queue, comporta-se ora como uma pilha, ora como uma fila. A estrutura de dados denominada *FringeList* é uma lista linear FL=(a_1, a_2,..., a_n).

Por sua vez, a estrutura de dados *StackQueue* pode ser considerada uma lista linear SQ=(a_1, a_2,..., a_n) na qual são possíveis inserções e retiradas em uma extremidade, chamada traseira ou topo e retiradas em outra extremidade, dita frontal. Sobre a estrutura SQ são definidas operações para: $criar(SQ)$, criar SQ como uma estrutura de dados vazia; $enfileirar(x, SQ)$, acrescentar o elemento x na traseira da estrutura SQ, devolvendo a estrutura SQ resultante; $desenfileirar(SQ)$, retirar o elemento frontal de SQ, devolvendo-o juntamente com a estrutura SQ resultante; $desempilhar(SQ)$, retirar o elemento do topo de SQ, devolvendo-o juntamente com a estrutura SQ resultante; $vazia(SQ)$, devolver a condição verdadeira se SQ for vazia, e em caso contrário, devolve a condição falsa.

Considerando as propriedades e operações que podem ser realizadas sobre as estruturas de dados SQ e FL, a seguir descreve-se o comportamento do algoritmo. Inicialmente, a raiz da árvore é enfileirada na SQ. Toda vez que se inicia o preenchimento de uma nova página, desenfileira-se o primeiro elemento da SQ, que se transforma no patriarca da página a ser preenchida. Todos os elementos das $x-1$ gerações do patriarca são então armazenados na página, que armazena até 2^x-1 nodos. Sobrando espaço na página, os elementos da geração seguinte que sejam patriarcas de subárvores maiores ou iguais ao espaço disponível na página são enfileirados na SQ. Os demais elementos são inseridos na FL, pois correspondem às subárvores da franja. Não sobrando espaços na página os elementos da geração seguinte que sejam patriarcas de subárvores maiores que o tamanho da página são enfileirados na SQ, os demais na FL.

Caso a página tenha espaço disponível, o algoritmo procede da seguinte maneira: o último elemento inserido na SQ é desempilhado e armazenado na página. Os seus filhos são enfileirados na SQ. Se ainda houver espaço, e a SQ não estiver vazia, mais uma vez o último elemento inserido na SQ é desempilhado e armazenado na página, e seus filhos são enfileirados na SQ. O processo se repete até que a página esteja preenchida, ou a SQ esteja vazia.

Quando a página já está completa, isto é, sem espaços sobrando, o algoritmo inicia o preenchimento de uma nova página, desenfileirando seu patriarca da SQ, como descrito acima. Quando não há mais elementos na SQ, resta ao algoritmo proceder à alocação das subárvores pertencentes à franja da árvore. Na franja da árvore existem grupos de nodos que ainda devem ser paginados e espaços sobrando em páginas, dessa forma objetiva-se alocá-los mantendo-os juntos e otimizando o uso de espaços disponíveis. Para isso propõe-se a utilização de empacotamento unidimensional, como descrito anteriormente. O algoritmo termina quando todos os nodos da árvore estão armazenados em páginas, situação refletida por FL vazia.

A especificação do algoritmo proposto para a paginação de uma árvore binária de pesquisa é descrita em alto nível na figura 4.

```
Algoritmo de Paginacao de Arvores Binarias

INICIO
  criar(SQ);
  enfileirar(raiz da arvore, SQ);
  REPETIR
    criar uma nova pagina;
    patriarca <- desenfileirar(SQ);
    alocar o patriarca e suas x-1 geracoes na pagina;
    SE (nao ha espaco disponivel na pagina)
     ENTAO
       PARA todos os patriarcas da geracao seguinte,
         cujas subarvores sejam  maiores ou iguais ao
         tamanho da pagina, enfileirar(patriarca, SQ);
       inserir na FL os demais patriarcas da geracao seguinte;
     SENAO
       PARA todos os patriarcas da geracao seguinte,
         cujas subarvores sejam maiores ou iguais ao
         espaco disponivel na pagina, enfileirar(patriarca, SQ);
       inserir na FL os demais patriarcas da geracao seguinte;

    FIM SE
    ENQUANTO (existe espaco disponivel na pagina) e
      (nao vazia(SQ)) FACA
         elemento <- desempilhar(SQ);
         alocar elemento na pagina;
         PARA cada filho de elemento
           enfileirar(filho, SQ);
    FIM ENQUANTO
  ATE que vazia(SQ);
  SE (nao vazia(FL))
    ENTAO
     aplicar algoritmo de empacotamento unidimensional
     paginando as subarvores da franja;
FIM.
```

Figura 4: O algoritmo proposto

3.3 Complexidade do Algoritmo de Paginação Proposto

A análise da complexidade do algoritmo proposto é realizada levando-se em consideração separadamente o trecho que aloca em páginas toda a árvore com exceção da franja, e o trecho que realiza o empacotamento unidimensional nas subárvores da franja. Na análise apresentada será considerada a ordem de tempo de execução e não o valor exato da função de complexidade.

Analisando-se a fase inicial do algoritmo, considera-se f uma função de complexidade tal que $f(n)$ é o número de registros acessados no arquivo, isto é, o número de vezes que um nodo da árvore é consultado. Considere que o pior caso corresponde ao maior tempo de execução sobre todas as entradas de tamanho n. Considere, ainda, que f é uma função de complexidade baseada na análise de pior caso, com isso o custo de aplicar o algoritmo nunca é maior do que $f(n)$.

Observa-se que a função de complexidade f da fase inicial do algoritmo é $f(n) = 4n$. Tal linearidade pode ser aferida analisando-se os procedimentos realizados. Para calcular o tamanho de todas as subárvores são necessários n acessos aos elementos. Para alocar todos os nodos da árvore são necessários $2n$ acessos, dos quais n acessos para a alocação do nodo em si e n acessos para se estabelecer a vinculação entre nodo pai e nodo filho. Finalmente, para processar, durante a fase inicial do algoritmo, as estruturas de dados SQ e FL são necessários no máximo n acessos a elementos.

A fase final do algoritmo, por sua vez, depende fundamentalmente do algoritmo utilizado para o empacotamento unidimensional aplicado às subárvores da franja. Considere g a função de complexidade da fase final do algoritmo tal que $g(t)$ descreve o número de acessos a nodos para alocação das t subárvores da franja. A literatura reporta algoritmos aproximados com funções de complexidade quadrática [4], ou seja, $g(t) = kt^2$, onde k é uma constante. Neste caso, apesar da complexidade ser quadrática ela se aplica ao número de subárvores da franja e não ao número de nodos do sistema.

4 Resultados Experimentais

Esta seção apresenta os resultados experimentais obtidos. São descritas as métricas utilizadas para quantificar o desempenho do algoritmo, bem como os resultados de comparações com outras estratégias de paginação, incluindo paginação seqüencial, paginação em largura, paginação em profundidade, paginação ótima teórica de árvore balanceada equivalente e árvores B.

4.1 Métricas Utilizadas

Para analisar o desempenho do algoritmo, foram definidas duas métricas: quantidade de espaços não preenchidos nas páginas e número de páginas visitadas em pesquisas.

4.2 Avaliação do Número de Páginas Visitadas em Pesquisas

A primeira comparação realizada refere-se ao número de páginas visitadas. Na tabela 1 é possível avaliar a quantidade média de páginas visitadas ao pesquisar todos os nodos de uma árvore.

Tabela 1: Quantidade média de páginas visitadas em diferentes estratégias

Estratégias	Tam. Página = 3	Tam. Página = 7	Tam. Página = 15
Seqüencial	1495,99	3313,30	7410,69
Largura	1681,04	3981,23	9309,85
Profundidade	1198,61	2500,61	5471,75
Algoritmo Proposto	969,69	1726,17	3365,55
Árvores B	841,04	1644,62	2914,78
Ótimo Teórico	771,45	1383,10	2760,65

A figura 5 apresenta a quantidade média de páginas visitadas, considerado 15 como tamanho de página. Como observado na figura, resultados são comparados com outras abordagens.

Constata-se que o algoritmo proposto é sempre melhor que as paginações seqüencial, em largura e em profundidade; sendo inferior ao valor ótimo teórico de uma árvore balanceada equivalente. Entretanto, o algoritmo proposto tem desempenho mais próximo do ótimo teórico, do que das outras abordagens.

Em comparação com as árvores B, a quantidade média de páginas visitadas é similar. Entretanto, como demonstrado na próxima seção, a abordagem proposta é muito mais eficiente que as árvores B em termos da taxa de ocupação das páginas.

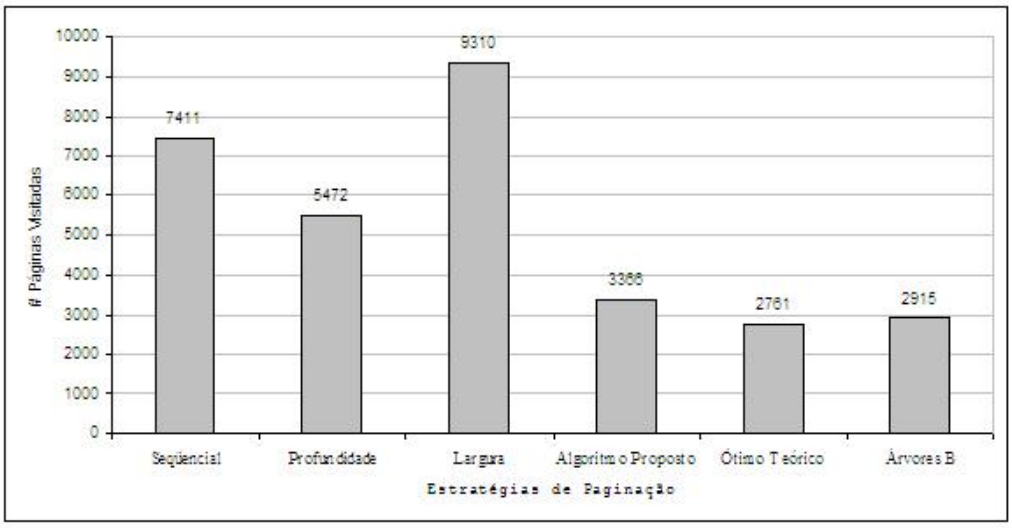

Figura 5: Média de páginas visitadas; com tamanho de página = 15

4.3 Análise da Taxa de Preenchimento das Páginas

Outro resultado experimental refere-se aos espaços nas páginas obtidas pelo processamento do algoritmo. A tabela 2 apresenta a taxa média de preenchimento das páginas obtida com as diferentes estratégias.

Tabela 2: Taxa média de preenchimento das páginas em diferentes estratégias

Estratégias	Tam. Página = 3	Tam. Página = 7	Tam. Página = 15
Seqüencial, Largura, Profundidade	98,82%	98,42%	98,69%
Algoritmo Proposto	98,77%	98,42%	98,68%
Árvores B	67,15%	67,30%	67,75%

Nos experimentos realizados observou-se que as árvores B apresentam a taxa média de preenchimento de 67,52% em árvores aleatórias. Por sua vez, o algoritmo proposto apresentou uma taxa média de preenchimento de 98,62%, próxima do ótimo que é obtido nas paginações seqüencial, em largura e em profundidade.

A figura 6 ilustra a quantidade total de espaços produzidos nas diferentes estratégias. Considerando as árvores B o total de largura de banda requerida para transferir completamente uma árvore é proporcionalmente maior do que aquele requerido para transferir uma árvore paginada pelo algoritmo proposto.

5 Trabalhos Relacionados

Intenso trabalho de investigação tem sido desenvolvido sobre algoritmos e estruturas de dados para tratamento de grandes quantidades de dados em memória externa [5][6][7], incluindo o problema da alocação de árvores de pesquisa. Uma abordagem para paginação de uma árvore binária parcialmente paginada utilizando balanceamento externo é apresentada em [8]. Algoritmos para armazenamento de árvores de sufixos em memória secundária são apresentados em [9]. Técnicas de *clustering* para minimização do mapeamento de estruturas de árvores são descritas em [10]. Um eficiente algoritmo utilizando programação dinâmica para empacotamento de árvores é proposto em [11]. Outra abordagem para empacotamento de árvores em memória hierárquica usando algoritmos aproximados é proposto em [12].

6 Conclusão

Neste trabalho foi descrito um algoritmo que realiza a paginação de árvores binárias de pesquisa não balanceadas. O algoritmo pode ser aplicado em problemas de biologia computacional que utilizam árvores binárias desbalanceadas, originadas de seqüências biológicas. O algoritmo constrói a paginação ótima

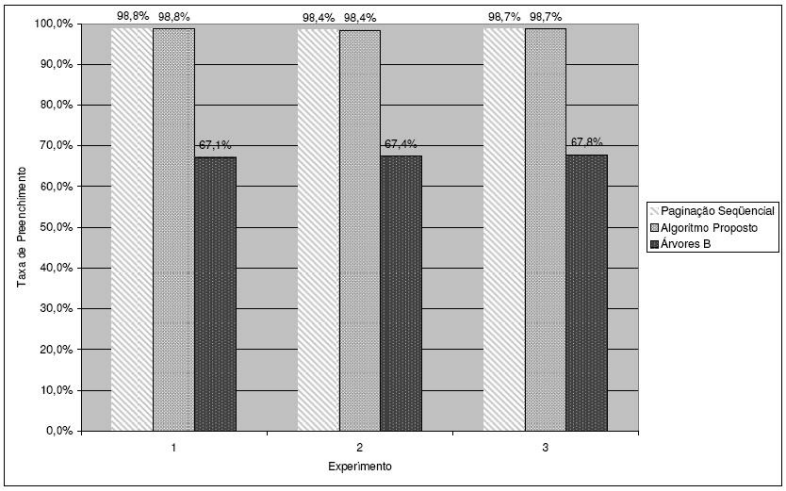

Figura 6: Taxa de preenchimento das páginas

quando isso é possível e propõe uma política eficiente para o preenchimento das páginas de uma árvore binária degenerada, baseada na aplicação de empacotamento unidimensional na franja da árvore. A complexidade do algoritmo foi apresentada e discutida. O algoritmo foi implementado e resultados experimentais foram apresentados. Considerando a média de páginas acessadas em pesquisas, o algoritmo produz uma alocação de páginas 55% superior à alocação seqüencial, 64% superior às alocações em largura e em profundidade e resultados muito próximos aos obtidos pelas árvores B. Por outro lado, considerando a quantidade de espaço não utilizado nas páginas, e o total de páginas necessárias, o algoritmo apresenta uma taxa média de preenchimento de 98,62%. A comparação mostra que a abordagem proposta é a única que resulta em quantidade média de páginas acessadas próxima da ótima, e ao mesmo tempo, uma taxa de preenchimento de páginas também próxima da ótima.

Trabalhos futuros incluem a investigação experimental do comportamento do algoritmo considerando outros algoritmos aproximados para o empacotamento da franja, e a comparação desses resultados com variações das árvores B. Outro tópico que permanece para trabalhos futuros refere-se à avaliação do algoritmo considerando dados dinâmicos, bem como o impacto do acesso concorrente aos dados.

Referências

[1] GONNET, G. H. ; BAEZA-YATES, R. *Handbook of Algorithms and Data Structures: in Pascal and C*. Addison-Wesley, 1991, 424 p.

[2] COHEN, J. Bioinformatics: An Introduction for Computer Scientists. *ACM Computing Surveys*, v. 36, n. 2, p. 122-158, 2004.

[3] PEDERSEN, C. N. S. *Algorithms in Computational Biology*. PhD Dissertation, University of Aarhus, Denmark, 2000, 210 p.

[4] GAREY, M. R. ; JOHNSON, D. S. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979, 338 p.

[5] FRAKES, W. B. ; BAEZA-YATES, R. Information Retrieval Data Structures and Algorithms. Prentice Hall, 1992, 464 p.

[6] BAEZA-YATES, R. ; RIBEIRO-NETO, B. Modern Information Retrieval. Addison-Wesley, 1999, 513 p.

[7] VITTER, J. S. External Memory Algorithms and Data Structures: Dealing with Massive Data. ACM Computing Surveys, v. 33, n. 2, p. 209-271, 2001.

[8] HENRICH, A. ; SIX, H.W. ; WIDMAYER, P. Paging Binary Trees with External Balancing. *Proceedings of the 15th International Workshop on Graph-theoretic Concepts in Computer Science*, p. 260-276, Netherlands, 1990.

[9] CLARK, D. R. ; MUNRO, J. I. Efficient Suffix Trees on Secondary Storage. *Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms*, p. 383-391, Atlanta, 1996.

[10] DIWAN, A. A. ; RANE, S. ; SESHADRI, S. ; SUDARSHAN, S. Clustering Techniques for Minimizing External Path Length. *Proceedings of the 22nd VLDB Conference*, p. 342-353, India, 1996.

[11] GIL, J. ; ITAI, A. How to Pack Trees. *Journal of Algorithms*, v. 32, n. 2, p. 108-132, 1999.

[12] BENDER, M. A. ; DEMAINE, E. D. ; FARACH-COLTON, M. Efficient Tree Layout in a Multilevel Memory Hierarchy. *Proceedings of the 10th Annual European Symposium on Algorithms*, p. 165-173, Italy, 2002.