Penggunaan Digital Tree Hibrida pada Aplikasi Information Retrieval untuk Dokumen Berita

Agus Zainal Arifin

Jurusan Teknik Informatika, FTIF, Institut Teknologi Sepuluh Nopember
Kampus ITS, Keputih, Sukolilo, Surabaya, 60111, Indonesia
Email : agusza@its-sby.edu

Abstrak

Ledakan jumlah dokumen elektronis dewasa ini mengakibatkan timbulnya 2 masalah besar, yakni teknologi penyimpanan dan teknologi temu kembali informasi. Salah satu aplikasi yang mulai banyak dibutuhkan dalam bidang ini adalah pendeteksian dokumen berita. Dokumen-dokumen tersebut biasa direpresentasikan dalam bentuk *array* bobot. Hal ini berarti dibutuhkan suatu metodologi yang mampu mengakomodasi kebutuhan tersebut dalam waktu singkat dan hemat memori.

Dalam penelitian ini, kami mencoba untuk mendeteksi berbagai kemungkinan dalam struktur data digital tree agar ditemukan konfigurasi yang tepat dalam menghasilkan bobot TF-IDF untuk tiap *term* secara cepat. Pertimbangan utama yang menjadi dasar pemakaian digital tree sebagai struktur data dan TF-IDF sebagai metode pembobotan adalah beberapa penelitian sebelumnya yang mengunggulkan keduanya.

Dari sejumlah uji coba, kami menyimpulkan bahwa konfigurasi yang paling tepat untuk merepresentasikan sekaligus mengitung nilai bobot *term* pada dokumen berita adalah dengan menggunakan digital tree yang digabungkan dengan algoritma sorting level trie. Pengurutan data ini sendiri akan lebih efisien bila menggunakan seluruh data yang tersedia sebagai *key* untuk pengurutan datanya.

KEYWORDS : Information Retrieval, bobot TF-IDF, Digital Tree, Quick Sort, news event detection

1. Pendahuluan

Dengan digunakannya teknologi internet, dewasa ini perkembangan jumlah informasi elektronis mengalami peningkatan yang sangat drastis, terutama jumlah dokumen berita online. Dalam hal ini, Robert R.K. [1] mengindikasikan timbulnya 2 masalah besar, yakni teknologi penyimpanan informasi (*Information Storage*) dan temu kembali informasi *(Information Retrieval)*. Penyimpanan informasi berikut pencarian dan penemuan kembalinya harus diusahakan secepat mungkin. Representasi ruang penyimpanan juga harus diharapkan mampu mengakomodasi hal ini.

Berkaitan dengan representasi sistem informasi temu kembali, Salton [2] menjelaskan bahwa terdapat 3 model yang dapat digunakan, yakni Boolean model, probabilistic model, dan *vector* space model. Model terakhir inilah yang paling sederhana dan paling produktif [2]. *Vector* model ini merepresentasikan *term* yang digunakan baik oleh dokumen maupun oleh query. Elemen *vector* tersebut merupakan bobot *term* yang dalam penelitian ini menggunakan model TF-IDF [2][5].

Dari penjelasan di atas, dapat disimpulkan bahwa perlu adanya representasi *term*, dimana pemakai dapat dengan cepat mengisikan nilai bobot *term* pada *vector* tersebut. Frakes dan Yates [3] mengatakan bahwa terdapat 3 struktur data untuk pengorganisasian data, yakni search trees, digital trees, dan hashing. Dalam penelitian ini, struktur data yang kami pilih adalah digital tree atau trie. Pertimbangannya adalah keberhasilan Horasi [4] dalam tugas akhirnya yang membuktikan bahwa struktur data digital tree merupakan metode yang paling handal untuk menyimpan kamus. Metode ini juga berhasil menghemat pemakaian memori untuk penyimpanan huruf-hurufnya, baik pada saat berada di main memori maupun pada saat disimpan di file. Oleh karena informasi yang menjadi sasaran dalam penelitian ini adalah dokumen berita, maka perlu dilakukan penyesuaian pada model trie tersebut.

Jadi permasalahan yang dihadapi adalah dibutuhkannya suatu algoritma dan struktur data yang mampu merepresentasikan *vector term* di atas. Sehingga user dapat secepat mungkin mengetahui bobot TF-IDF tiap *term*. Penelitian ini berusaha menemukan konfigurasi algoritma yang tepat berdasarkan struktur data digital tree untuk menghasilkan bobot TF-IDF tiap *term* secara cepat.

2. Metodologi

Metodologi penelitian ini adalah sebagai berikut :

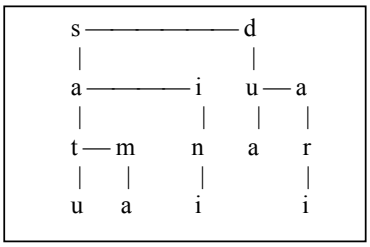
a) Pengumpulan sampel berita dari surat kabar yang online di internet. Dalam hal ini 233 dokumen berita dari segmen Berita Utama dalam surat kabar Suara Pembaruan.

b) Penganalisaan fenomena dokumen berita yang tiba secara berurutan. Analisa tersebut pada akhirnya, akan menghasilkan algoritma digital tree hibrida yang menggabungkan kemampuan *digital tree* dan q*uick sort*.
c) Dilakukan ekstraksi terhadap seluruh dokumen berita untuk memeproleh *term-term* pembentuk dokumen. Tiap *term* tersebut, diinputkan ke algoritma ini agar dapat disimpan sekaligus dihitung frekuensinya dengan cepat.
d) *Array* frekuensi dari tiap dokumen ini sekaligus menjadi input penghitungan bobot *term* dengan metode TF-IDF.
e) Dari hasil penulisan *term* ke dalam tree, dilakukan penghitungan terhadap jumlah langkah pencarian atau pemeriksaan *node* hingga berhasil ditemukannya *node* yang dicari. Penghitungan ini dilakukan pada tiap level tree, dimana tiap level maksimal terdiri dari 26 huruf yakni a sampai dengan z.
f) Dilakukan beberapa modifikasi terhadap algoritma ini untuk mencari konfigurasi terbaik agar jumlah pencariannya minimal. Evaluasi dilakukan pada setiap hasil uji coba algoritma.
g) Pengambilan kesimpulan dan pemberian saran lebih lanjut guna perbaikan algoritma.

3. Digital Tree Hibrida

3.1. Digital Tree

Digital tree [3] atau trie merupakan struktur tree recursive yang mendekomposisi string secara digital. Sampai saat ini, sama sekali belum ditemukan contoh aplikasi trie untuk selain kamus kata [4]. Misalkan kata-kata yang disimpan adalah : ‘satu’, ‘sama’, ‘sini’, ‘dua’, dan ‘dari’, maka hasil dekomposisinya seperti nampak pada Gambar 1.



Gambar 1. Bentuk Digital Tree

Pencarian kata dimulai dari *node* kiri atas atau *node* yang disebut sebagai HEAD of tree. Huruf pertama *term* dibandingkan dengan huruf pada *node* ini, dengan kemungkinan hasil :

a. Bila karakter itu sama, maka karakter berikutnya dicari pada level dibawah *node* tersebut, selanjutnya proses diteruskan untuk huruf berikutnya dengan cara yang sama. Bila pada huruf terakhir kata tersebut sama dengan huruf pada *node* yang sedang dibandingkan dan *node* tersebut menunjukkan *flag* sebagai huruf tersakhir, maka diputuskan bahwa kata tersebut ada di dalam kamus. Bila *flag* tidak menunjukkan hal itu, maka dianggap kata itu tidak terdapat di kamus, sekalipun huruf-hurufnya terakomodasi semuanya.
b. Bila karakter itu tidak sama, maka karakter itu dibandingkan dengan karakter yang selevel dengan *node* ini, demikian seterusnya hingga ditemuka *node* yang sama. Bila hingga *node* yang terakhir pada level tersebut *node* yang sedang dicari tidak ditemukan, maka diputuskan bahwa kata tersebut tidak ada dalam kamus.

Dalam hal ini jenis digital tree yang paling sesuai bentuknya adalah *De La Briandais*, dimana bentuk node tiap hurufnya seperti Gambar 2.

Key / Huruf	*Flag* akhir kata	Pointer ke *node* kanan	Pointer ke *node* bawah

Gambar 2. Bentuk node

Penyimpanan tiap node dalam bentuk file dapat dihemat dengan menjadikannya 1 byte (8 bit). Hal ini dapat diakomodasi, sebab jumlah huruf adalah 26, yang berarti dapat dialokasikan oleh 5 bit. Sedangkan 3 bit sisanya dapat dimanfaatkan oleh flag indikator akhir kata, flag untuk indikator keberadaan node di kanan dan di bawahnya

Usulan lainnya untuk aplikasi di memori adalah penggabungan *flag* akhir kata tersebut ke hurufnya. Sebab pada dasarnya jumlah huruf adalah 26 yang bila direpresentasikan dengan biner, hanya membutuhkan 5 bit. Hal ini berarti untuk 1 byte, masih tersisa 3 bit yang belum digunakan. Salah satu bit tersebut bisa digunakan sebagai *flag* akhir kata. Bahkan pada proses penyimpanan di *secondary storage*, kedua bit sisa tersebut bisa digunakan sebagai *flag* untuk menandakan ada atau tidaknya *node* berikutnya (*next*) dan *node* kanannya (*right sibling*). Dengan demikian, urutan bit yang dapat diaplikasikan untuk tiap *node* dapat diaplikasikan seperti pada tabel 1.

Tabel 1. Urutan Penggunaan Bit

Bit	Fungsi
1	Bernilai 1 bila huruf tersebut adalah huruf terakhir suatu *term*
2	Bernilai 1 bila *node* tersebut memiliki *sibling* atau *node* di kanannya
3	Bernilai 1 bila *node* tersebut memilki lanjuta huruf ke bawah
4-8	Kode huruf dengan interval 00000 hingga 11111

3.2. Vector Model

Alberto M. [5] [10] menyatakan bahwa dokumen dan *term* dapat direpresentasikan dengan

vector space. Misalkan t_1, t_2, …, t_n menyatakan *term* yang digunakan untuk mengindeks database yang terdiri dari dokumen D_1, D_2, … D_n, maka dokumen D_i dinyatakan dengan $D_i = (a_{i1}, a_{i2}, \ldots a_{in})$, dimana a_{ij} = bobot *term* t_j dalam dokumen D_i. M. Kobayashi [8] menguraikan beberapa metode pembobotan *term*, yakni Binary weighting, *Term frequency,* Log Entropy, Normal, GdIdf, Idf, Entropi, dan TF-IDF. Dalam penelitian ini, metode yang digunakan adalah TF-IDF [2][5][8][9]. Adapun rumus TF-IDF adalah sebagai berikut :

$$w(t,d) = tf(t,d) \; x \; log(N / n_t)$$

Arti *tf(t,d)* atau *term frequency* adalah jumlah *term* t dalam dokumen d. Sedangkan N adalah jumlah seluruh dokumen dan n_t adalah jumlah dokumen yang memiliki *term* t.

Lebih lanjut Salton [2] menjelaskan alasan penggunaan metode ini. Ternyata sekalipun *term frequency* banyak digunakan, namun ia hanya mendukung *Recall*. Sedangkan *Precision* akan lebih meningkat bila *vector* bobot tersebut menggunakan *term* yang jarang muncul pada koleksi dokumen. Tentunya *term* demikian akan diharapkan mampu mengelompokkan sejumlah dokumen yang memuatnya, sehingga berbeda dengan seluruh anggota koleksi dokumen lain yang tidak memilikinya. Kriteria ini dapat diakomodasi dengan menghitung *log (N / n_t)* seperti di atas. Sedangkan alasan digabungnya kedua metode tersebut adalah dimaksudkan agar mampu meningkatkan baik *Recall* maupun *Precision* sekaligus. Sehingga kriteria *term* yang paling tepat adalah *term* yang sering muncul dalam dokumen secara individu, namun jarang dijumpai pada dokumen lainnya.

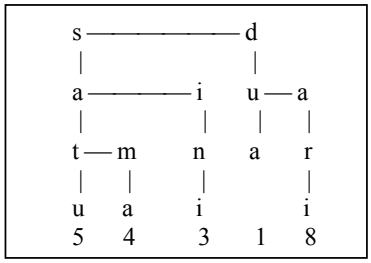
3.3. Fenomena Dokumen Berita

Pertanyaan yang banyak diajukan pembaca berita adalah peristiwa apa yang terdapat dalam dokumen berita ini. Kata ‘peristiwa’ atau *event* [7] [11] berbeda dengan kata topik dalam pengertian konvensional. Sebuah *event* menunjukkan sesuatu kejadian di suatu tempat tertentu pada saat tertentu. Pengurutan berita *event* secara temporal memiliki beberapa pola sebagai berikut :

1. Berita yang membahas *event* yang sama cenderung berdekatan waktunya.
2. Adanya gap waktu antar berita yang membahas topik yang sama, biasanya menunjukkan munculnya *event* baru.
3. Pergeseran kosa kata penting dan perubahan tajam pada distribusi frekuensi *term* biasanya menunjukkan adanya *event* baru.

Ketiga pola di atas merekomendasikan beberapa hal sebagai berikut :

1. Berdasarkan pola pertama di atas, maka dokumen yang berdampingan sangat mungkin mengandung sejumlah *term* yang sama. Sehingga suatu dokumen dapat dijadikan sebagai basis bagi pencarian bagi dokumen berikutnya, dalam artian distribusi frekuensi *term* yang terdapat pada dokumen tersebut sangat mungkin mirip dengan dokumen berikutnya. Oleh karena itu algoritma pengurutan *node* berdasarkan frekwensi tentu akan sangat membantu.
2. Berdasarkan pola kedua dan ketiga, maka pengurutan berdasarkan frekwensi *node* dari suatu dokumen tidaklah selalu membantu mempercepat pencarian *term* pada dokumen berikutnya. Sebab dimungkinkan juga sejumlah dokumen yang berdampingan, masing-masing membahas *event* yang jauh berbeda. Oleh karenanya frekwensi yang dijadikan basis pengurutan ini, haruslah diperolah dari akumulasi frekwensi *node-node* sebelumnya.
3. Berdasarkan pola kedua dan ketiga juga, maka pengurutan seharusnya bukanlah berdasarkan frekwensi *term*, melainkan frekwensi *node*. Sebab bisa saja dalam suatu *leave*, suatu *node* memiliki frekwensi yang sangat tinggi namun pada *leave* itu *term* tersebut sendirian. Sementara di lain cabang terdapat sejumlah *leave* yang memakai bersama suatu *node*, padahal seandainya frekwensi sejumlah *leave* tersebut diakumulasi, sangat mungkin lebih tinggi dari pada frekwensi *leave* pertama di atas. Hal ini dapat dicontohkan pada Gambar 2.



Gambar 2. Contoh Kasus

Pada Gambar 2, kata ‘dari’ tidak berhak diletakkan di awal meskipun frekwensinya tertinggi. Sebab *node* utamanya yakni ‘d’ hanya dilewati 9 kali, sedangkan *node* sebelumnya yakni ‘s’ dilewati 12 kali yakni 5 + 4 + 3.

3.4. Kombinasi Digital Tree dan Algoritma Pengurutan

Metode pengurutannya adalah *Quick Sort* dengan pertimbangan kecepatannya yang tinggi. Sedangkan kombinasi algoritma pengurutan dan algoritma pencarian *node* pada digital tree adalah :

1. Urutkan semua *node* pada level 1, berdasarkan frekwensi yang telah dibahas diatas. Dalam contoh di atas, *node* yang dirutkan hanya 2 yakni ‘s’ dan ‘d’ yang masing-masing frekwensi adalah 12 dan 9. Urutan ini ternyata sudah benar, sehingga proses pengurutan dapat

dilewati. Seandainya perlu penukaran posisi, maka yang ditukarkan bukan hanya posisi 's' dan 'd', melainkan juga harus diikui dengan seluruh *child node* pada masing-masing *node* tersebut. Sebab tiap *node* dengan *child*nya merupakan urutan yang tidak mungkin dipisahkan.

2. Pengurutan di level 2, dimulai dari *child* milik *node* pertama pada level 1. Pada contoh di atas, prosesnya dikenakan pada *node* 'a' dan 'i'. Demikian pula pada *child* dari *node* kedua level 1, yakni *node* 'u' dan 'a'. proses pemeriksaan dan penukaran sama seperti langkah di atas.
3. Pengurutan pada level berikutnya juga dilakukan dengan cara di atas.

4. Uji Coba dan Pembahasan

Bahan untuk ujicoba ini sebagaimana telah disampaikan adalah dokumen berita yang diambil dari surat kabar Suara Pembaruan pada segmen Berita Utama. Jumlah seluruhnya mencapai 233 dokumen yang diurutkan secara kronologis, yakni berdasarkan tanggal terbitnya. Urutan nama-nama file dokumen ini disimpan pada FileList.txt.

Terdapat 4 konfigurasi yang diuji coba, yakni :

1. Pembobotan tanpa sorting
2. Pembobotan dengan sorting berdasarkan frekwensi *term* per dokumen
3. Pembobotan dengan sorting berdasarkan frekwensi akumulasi *term*
4. Pembobotan dengan sorting yang dilakukan setiap periode tertentu

Parameter yang diukur dari tiap uji coba adalah :

1. Variabel 'Waktu' adalah lamanya proses pemetaan seluruh *term* dari file ke dalam digital tree. Hitungannya tidak dalam detik, namun dalam clock, agar lebih presisi.
2. Variabel '*Node* yg dibaca' adalah jumlah *node* yang diperiksa sistem untuk memetakan *term* ke digital tree. Pembacaan ini meliputi *node* yang benar hingga bisa diteruskan, *node* yang salah hingga perlu dilakukan pencarian ke *right sibling*nya, dan pembacaan yang tidak berhasil, sehingga perlu membuat *node-node* baru.
3. Variabel 'Jumlah *Node*' adalah banyaknya *node* yang terdapat di dalam digital tree.
4. Variabel 'Jumlah Kata Unik' adalah jumlah kata yang tidak sama dengan yang lain yang masing-masing tentu memiliki frekwensi tertentu. Kata unik ini disebut sebagai *term*.

4.1. Uji Coba

Pembentukan digital tree berdasarkan sampel di atas untuk tiap uji coba selalu menghasilkan komposisi sebagai berikut :

Jumlah *Node* = 37.304 *node*
Jumlah Kata Unik = 10.984 kata

Hal ini menunjukkan bahwa implementasi algoritma yang diusulkan ini sudah benar. Sedangkan kedua paramer lain yang diukur untuk tiap uji coba dapat dilihat pada masing-masing tahap uji coba.

Adapun tahapan uji coba tersebut adalah :

1. Uji coba pertama, yakni tanpa sorting melakukan semua prosedur di atas, namun tiap selesai memproses 1 file tidak dilakukan pengurutan tree. Hasilnya adalah :
   Waktu = 41.100 clock
   Node yg dibaca = 2.727.149 *node*
2. Uji coba di atas diulangi lagi untuk data set yang sama, namun prosedur pengurutan sekarang difungsikan. Hanya saja yang dijadikan *key* untuk pengurutan adalah frekwensi tiap *term* pada setiap dokumen. Hasilnya sebagai berikut :
   Waktu = 41.970 clock
   Node yg dibaca = 2.514.025 *node*
3. Uji coba selanjutnya diterapkan pada data yang sama namun berbeda dalam hal *key* yang diurutkan. *Key* yang digunakan adalah nilai frekwensi akumulasi *term* yakni hasil penjumlahan frekwensi *term* tiap dokumen yang sudah diproses. Hasilnya adalah :
   Waktu = 39.440 clock
   Node yg dibaca = 2.077.708 *node*
   Kemudian dicoba lagi dengan mengubah urutan dokumen berita. Pengubahan ini dengan tetap menjaga urutan kronologisnya. Pengubahan hanya dilakukan pada dokumen-dokumen dari tanggal yang sama, sebab pada tanggal yang sama jumlah beritanya cukup banyak. Pengubahan ini menghasilkan output yang agak berbeda yakni :
   Waktu = 41.340 clock
   Node yg dibaca = 2.078.694 *node*
4. Uji coba dengan sorting pada tiap periode tertentu, yakni sorting untuk digital tree hanya dilakukan setelah sistem memproses X buah dokumen. Di sini akan diambil contoh X yang bernilai 10, 25, 40, dan 100. Bila X bernilai 10 berarti sorting dilakukan setiap selesai 10 dokumen. Hasilnya seperti pada Tabel 2.

Tabel 2. Hasil Uji Coba Sorting Tiap Periode

Jumlah Dokumen	Waktu (clock)	Node yang Dibaca
10	40.470	2.604.748
25	39.670	2.324.473
40	39.850	2.204.984
100	41.880	2.138.388

4.2. Pembahasan

1. Nampak bahwa pemakaian kriteria waktu sulit untuk dipertanggung jawabkan. Dari beberapa uji coba di atas, bila kita lakukan penghitungan

korelasi antara waktu yang dibutuhkan dan *node* yang diperiksa, ternyata korelasinya sebesar 0.503. Kecilnya tingkat korelasi ini sangat mungkin disebabkan oleh perbedaan tingkat kesibukan CPU pada saat dilakukannya masing-masing uji coba tersebut. Selain itu, juga adanya karakteristik manajemen memory dari Windows sendiri yang bisa melakukan penyimpanan memorinya di Secondary Storage. Bila hal ini terjadi tentu akan memakan waktu yang lebih lama dibanding proses yang tanpa melibatkan Secondary Storage tersebut.

2. Jumlah *node* yang dibaca cukup menimbulkan kontras antara uji coba yang dilakukan dengan sorting dan tanpa sorting yakni :

   Beda Jumlah Pemeriksaan =
   2.727.149 – 2.077.708 = 649.441 *node*

3. Pembahasan berikutnya adalah perbandingan jumlah *node* yang dibaca pada ujicoba Pembobotan dengan sorting berdasarkan frekwensi *term* per dokumen dan berdasarkan frekwensi akumulasi *term*. Ternyata perbedaannya signifikan yakni :

   Beda Jumlah Pemeriksaan =
   2.727.149 – 2.514.025 = 213.124 *node*

   Hal ini dapat dipahami sebab dokumen dari tanggal yang berdekatan memang dimungkinkan membahas *event* yang sama, namun dokumen yang dibahas pada tanggal yang sama tentunya jarang sekali yang membahas berita yang sama. Hal ini tercermin dari pengurutan yang berdasarkan dokumen berita sebelumnya (masih dalam satu tanggal) ternyata kurang mengakomodasi tujuan semula.

4. Dari pengubahan nilai periode pada ujicoba subbab 5.1.4. ternyata menunjukkan korelasi yang agak tinggi yakni –0.8. Artinya Jumlah *node* yang dibaca berbanding terbalik dari jumlah periodenya. Ini menunjukkan bahwa makin banyak dokumen berita yang dijadikan bahan, maka makin mudah pula pencarian pada digital treenya, walaupun yang banyak terlibat dalam masalah ini adalah *term* dari stop list. Sayangnya hingga kini, nampaknya belum tersedia stop list untuk bahasa Indonesia.
5. Baik jumlah *node* maupun jumlah kata yang unik untuk setiap percobaan ternyata menunjukkan angka yang sama. Ini berarti algoritma kita memang sudah berjalan dengan benar.
6. Dari jumlah *node* yakni 37.304 *node*, kita dapat memperkirakan besarnya file yang akan ditempati oleh trie ini, yakni sebesar 37.204 byte. Bila trie ini sudah disimpan, maka untuk memproses dokumen selanjutnya, cukup dengan me*load* ulang file ini.
7. Menurut percobaan kami yang hasilnya tidak kami tampilkan di sini, ternyata jumlah huruf rata-rata per kata adalah 6 huruf. Jadi bila jumlah kata unik kita adalah 10.984 kata, maka tentunya dibutuhkan lebih kurang 6 x 10.984 huruf = 65.904 huruf. Jumlah (65.904 byte) ini belum *term*asuk spasi yang semestinya bisa menandai akhir dari tiap kata. Namun dengan digital tree kita hanya membutuhkan 37.304 byte atau hemat sebesar 56.6 %, belum lagi di sini sudah mengakomodiasi pencariannya yang begitu cepat dibanding dengan *linear search*.
8. Ada sebuah fenomena yang menarik, yakni 2 buah kata yang tampil selalu berdampingan tentu akan memiliki bobot yang sama. Dalam hal ini kami menemukan *term* ‘Gus’ dan ‘Dur’ memiliki akumulasi bobot yang sama yakni 170.9957428.

5. Kesimpulan dan Saran

5.1. Kesimpulan

Beberapa kesimpulan yang bisa kita tarik dari penelitian ini antara lain :

1. Konfigurasi yang paling tepat untuk merepresentasikan sekaligus mengitung nilai bobot *term* pada dokumen berita adalah dengan menggunakan digital tree yang digabungkan dengan algoritma sorting level trie. Pengurutan data ini sendiri akan lebih efisien bila menggunakan seluruh data yang tersedia.
2. Representasi data yang tepat disertai dengan algoritma yang handal ternyata sangat menentukan kecepatan waktu pembobotan. Hal ini tercermin dari perbedaan jumlah *node* yang harus diperiksa oleh algoritma asli dibanding algoritma hibrida ini, dimana algoritma asli mampu menghemat perjalanan sistem dalam mencari *term* yang dituju.
3. Dalam membuat suatu algoritma semacam ini, ada baiknya untuk mempelajari lebih dahulu karakteristik domain. Dalam hal ini, dokumen berita ternyata mempunyai karakteristik yang dapat di*explore* untuk pengembangan algoritma yang tepat.
4. Bahwa untuk mengekstrak *term* dari sebuah file ternyata tidak lepas dari kebutuhan stop list. Dengan demikian tidak semua *term* akan diproses, apalagi *term* pada stop list pada umumnya memiliki frekwensi yang sangat tinggi.

5.2. Saran

Beberapa saran yang dapat digunakan untuk mengembangkan algoritma ini lebih lanjut antara lain sebagain berikut :

1. Dari nilai bobot ini sebenarnya kita dapat melakukan beberapa hal, diantaranya adalah membuat *stop list* khusus untuk Bahasa Indonesia. Sebab kata yang bisa muncul di

berbagai dokumen, bobot IDFnya akan 0. Contoh yang paling mudah adalah 33 *term* dari output bobot ini nilainya adalah 0. Ini disebabkan *term* tersebut adalah tag-tag yang memang terdapat dalam file html.

2. Struktur data dan algoritma yang diusulkan di sini tidak menutup kemungkinan untuk dikembangkan pada metode pembobotan lainnya.
3. Aplikasi ini juga sangat mungkin untuk dijadikan sebagai bagian dari suatu topik penelitian baru yang sedang berkembang yakni Topic Detection and Tracking. Saat ini area tersebut juga dikembangkan ke arah *event*, yang kami kira aplikasi ini akan sangat mendukung.

6. Referensi

[1] Robert R. Korfhage, *Information Storage and Retrieval*, Wiley Computer Publishing, Canada, 1997.

[2] G. Salton. *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer,* Addison-Wesley, Reading, Mass., 1989

[3] William B. Frakes dan Richardo B. Yates, *Information Retrieval : Data Structures and Algorithms*, Prentice-Hall, New Jersey, 1992

[4] Ambara F. Horasi, "Rancangan Kamus Kata untuk Pemeriksaan Ejaan Elektronis", Tugas Akhir Fakultas Ilmu Komputer, Universitas Indonesia, 1997.

[5] Alberto M., "Compound Key Word Generation from Document Databases Using A Hierarchical Clustering ART Model", Intelligent Data Analysis, Vol. 1, No. 1, 1997 (http://www.elsevier.computing/locate/ida)

[6] Harry L.W., Larry D., *Data Structures & Their Algorithms*, Harper Collins Publisher.

[7] Yiming Yang, dkk., "Learning Approaches for Detecting and Tracking News *Event*s", IEEE Intelligent Systems, July/Agustus 1999, hal. 32-43.

[8] Mei Kobayashi, Koichi Takeda, " Information Retrieval on the Web: Selected Topic", Desember 1999.

[9] Yiming Yang, "An Evaluation of Statistical Approaches to Text Categorization", INRT Journal, 1998

[10] David D. Lewis, "Training Algorithms for Linear Text Classifiers", Proc. SIGIR '96: 19th Annual International ACM SIGIR Conference and Development in Information Retrieval, ACM Press, New York, Agustus 1996, hal. 298-306.

[11] Ames Allan, Ron Papka, dan Victor Lavrenko, "On-line New *Event* Detection and Tracking", Proc. SIGIR '98: 21st Annual International ACM SIGIR Conference and Development in Information Retrieval, Australia, Agustus 1998, hal. 37-45.