INTEGRASI ALGORITMA POHON KEPUTUSAN C4.5 YANG DIKEMBANGKAN KE DALAM *OBJECT-RELATIONAL* DBMS

Veronica S. Moertini[1,2]**, Benhard Sitohang**[1]**, Oerip S. Santosa**[1]

[1]Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung
[2]Jurusan Ilmu Komputer, Universitas Katolik Parahyangan, Bandung
moertini@home.unpar.ac.id, benhard@if.itb.ac.id, oerip@informatika.org

ABSTRAK

Integrasi teknik-teknik data mining ke dalam DBMS, khususnya Object-Relational DBMS (ORDBMS), masih merupakan bidang penelitian yang aktif. Isu utama pada integrasi ini adalah: peleburan algoritma data mining ke dalam ORDBMS dengan memanfaatkan fitur-fiturnya untuk memperbaiki kualitas teknik tersebut. Pada penelitian ini, algoritma klasifikasi C4.5 dikembangkan dengan pendekatan aljabar relasional dan diintegrasikan ke dalam ORDBMS sebagai prosedur-prosedur tersimpan Java dan berbasis SQL, dengan tujuan untuk meningkatkan skalabilitas dan efisiensinya. Hasil eksperimen menunjukkan bahwa algoritma yang sudah diintegrasikan berhasil memperbaiki skalabilitas dan pada kasus khusus juga memperbaiki efisiensi.

***Kata kunci**: pengembangan algoritma C.5, integrasi C4.5 ke dalam Object-Relational DBMS, algoritma C4.5, perbaikan skalabilitas C4.5.*

1. PENDAHULUAN

Data yang tersimpan di *Database Management Systems* (DBMS) dapat terus bertambah ukurannya dan makin lama menjadi makin besar. Beberapa tahun terakhir ini, para pengguna DBMS sudah mulai merasakan adanya kebutuhan untuk menambang data yang tersimpan di DBMS untuk mengekstraksi pengetahuan yang berharga dari data yang menumpuk ini. Akan tetapi, algoritma-algoritma data mining yang diperlukan untuk mengekstraksi pengetahuan dari data, pada saat ini belum sepenuhnya dikembangkan untuk memanfaatkan fitur-fitur DBMS yang dapat meningkatkan kualitas dan diintegrasikan ke dalam DBMS, khususnya *Object-Relational* DBMS (ORDBMS) yang merupakan teknologi DBMS terbaru [4,11].

Pada [11], Saleem *et al*. mengajukan konsep awal tentang skema integrasi teknik-teknik data mining ke dalam ORDBMS, yaitu dengan "peleburan" teknik-teknik data mining pada skema ORDBMS atau menempatkan teknik-teknik ini pada "host language" ORDBMS. Saleem belum membahas penerapan konsep tersebut pada setiap teknik data mining secara rinci. Sebagaimana diketahui, sampai saat ini sudah cukup banyak algoritma data mining yang dikembangkan para peneliti. Algoritma tersebut dapat dikelompokkan menurut fungsinya, misalnya klasifikasi dan prediksi, pengelompokan, asosiasi, *outlier detection*, analisis tren, dll. Karena setiap algoritma memiliki fungsi dan kelebihan/kekurangannya sendiri, penelitian yang mendalam untuk mengintegrasikan setiap algoritma ke dalam ORDBMS dengan memanfaatkan fitur-fitur ORDBMS masih diperlukan. Penelitian ini berkontribusi pada integrasi algoritma C4.5 ke dalam ORDBMS.

Algoritma C4.5 adalah algoritma klasifikasi pohon keputusan yang sudah banyak digunakan. Dalam mengkonstruksi pohon, C4.5 memuat seluruh kasus ke dalam memori sehingga dibatasi oleh ketersediaan memori dan skalabilitas C4.5 dinilai kurang baik [4,9,10]. Penelitian ini ditujukan untuk meningkatkan skalabilitas algoritma C4.5 dengan memperhatikan efisiensi dari algoritma ini dan sekaligus mengintegrasikannya ke dalam ORDBMS. Pendekatan yang akan digunakan adalah aljabar relasional, khususnya operator proyeksi dan seleksi. Operator ini akan dimanfaatkan dalam merancang algoritma sehingga pembacaan data dapat dilakukan langsung ke tabel basisdata yang berukuran sangat besar maupun memuat partisi data himpunan data (yang berukuran lebih kecil dibandingkan terhadap data asli) ke dalam memori untuk diproses lebih lanjut secara efisien. Dengan pendekatan tersebut, algoritma akan memanfaatkan fungsi-fungsi yang tersedia di DBMS, sehingga berbentuk sederhana atau mudah dipahami. Karena efisiensi merupakan kriteria penting untuk mengukur kualitas teknik klasifikasi, efisiensi juga diperhatikan pada tahap implementasi algoritma di lingkungan ORDBMS.

2. ALGORITMA C4.5

Algoritma C4.5 mengkonstruksi pohon keputusan dari data pelatihan, yang berupa kasus-kasus atau rekord-rekord (tupel) dalam basisdata. Setiap kasus berisikan nilai dari atribut-atribut untuk sebuah kelas. Setiap atribut dapat berisi data diskret

atau kontinyu (numerik). C4.5 juga menangani kasus yang tidak memiliki nilai untuk sebuah atau lebih atribut. Akan tetapi, atribut kelas hanya bertipe diskret dan tidak boleh kosong [4,9,10].

Tiga prinsip kerja algoritma C4.5 adalah:

- Pertama, konstruksi pohon keputusan. Tujuan dari algoritma konstruksi pohon keputusan adalah mengkonstruksi struktur data pohon (dinamakan pohon keputusan) yang dapat digunakan memprediksi kelas dari sebuah kasus atau rekord.
- Kedua, pemangkasan pohon keputusan dan evaluasi. Karena pohon yang dikonstruksi dapat berukuran besar dan tidak mudah "dibaca", C4.5 dapat menyederhanakan pohon dengan melakukan pemangkasan berdasarkan nilai tingkat kepercayaan. Pemangkasan juga bertujuan untuk mengurangi tingkat kesalahan prediksi pada kasus (rekord) baru.
- Ketiga, pembuatan aturan-aturan dari pohon keputusan. Aturan-aturan dalam bentuk *if-then* diturunkan dari pohon keputusan dengan melakukan penelusuran dari akar sampai ke daun.

Penjelasan rinci tentang algoritma C4.5 dapat ditemukan di [4,9,10].

2.1 Algoritma Konstruksi Pohon

Algoritma dasar untuk induksi pohon keputusan pada C4.5 adalah algoritma *greedy* yang membangun pohon keputusan dari atas ke bawah (*top-down*) secara rekursif dengan cara *divide* dan *conquer*. Masukan dari algoritma ini adalah set data yang berisi sampel-sampel data dan kandidat atribut yang harus ditelaah, terdiri dari minimal sebuah atribut prediktor dan sebuah atribut kelas. Atribut prediktor dapat bertipe diskret atau numerik, sedangkan atribut kelas harus bertipe diskret. Dalam terminologi basisdata, set data ini berupa tabel, sedangkan sampel adalah rekord. Set data ini dapat memiliki atribut (kolom tabel) bertipe diskret maupun kontinyu. Adapun langkah-langkah konstruksi pohon ditunjukkan pada Algoritma 1 [4,9].

Algoritma: Generate_decision_tree
Narasi: Membuat pohon keputusan dari data pelatihan yang diberikan.
Masukan: Sampel data pelatihan, *samples*, yang direpresentasikan dengan atribut bernilai diskret, kandidat himpunan atribut, *attribute-list.*
Keluaran: Sebuah pohon keputusan.
Metoda:
(1) buat sebuah simpul *N*,
(2) **if** *samples* memiliki kelas yang sama, *C*, **then**
(3) **return** *N* sebagai simpul daun dengan label kelas *C*;
(4) **if** *attribute-list* kosong **then**
(5) **return** *N* sebagai simpul daun dengan label kelas terbanyak di *samples*
(6) pilih *test-attribute*, yaitu salah satu atribut dari *attribute-list* dengan *gain ratio* terbesar;
(7) beri label pada simpul N dengan *test-attribute*;
(8) **for** setiap nilai a_i pada *test-attribute;*
(9) tambahkan cabang pada simpul N untuk kondisi *test-attribute* = a_i*;*
(10) buat partisi sampel s_i dari *samples* dimana *test-attribute* = a_i*;*
(11) **if** s_i kosong **then**
(12) tempelkan daun yang diberi label dengan kelas terbanyak di *samples*;
(13) **else** tempelkan simpul yang dibuat oleh Generate_decision_tree (s_i, *attribute-list- test-attribute);*

Algoritma 1. Algoritma konstruksi pohon keputusan [4,9].

2.2 Komputasi *Gain Ratio* pada Kontruksi Pohon C4.5

Pada konstruksi pohon C4.5, di setiap simpul pohon, atribut dengan nilai *gain ratio* yang tertinggi dipilih sebagai atribut test atau split untuk simpul. Rumus dari *gain ratio* adalah *gain ratio(a) = gain(a) / split info((a),* dimana *gain(a)* adalah *information gain* dari atribut *a* untuk himpunan sampel *X* dan *split info(a)* menyatakan informasi potensial yang didapat pada pembagian *X* menjadi *n* sub himpunan berdasarkan telaahan pada atribut *a*.

Sedangkan *gain(a)* didefinisikan sebagai [9]:

$$gain(a) = info(X) - info_a(X) \quad (1)$$

dimana

$$info(X) = -\sum_{j=1}^{k} \frac{freq(C_j, X)}{|X|} \times \log_2\left(\frac{freq(C_j, X)}{|X|}\right)$$

dengan *k* adalah jumlah kelas pada himpunan rekord *X*. $freq(C_j, X)$ menyatakan jumlah sampel pada *X* yang memiliki nilai kelas C_j . |*X*| menyatakan kardinalitas (jumlah anggota) himpunan data *X*. $info_a(X) = \sum_{i=1}^{n} \frac{|X_i|}{|X|} \times info(X_i)$ menyatakan *info(X)* dengan *a* adalah atribut yang ditelaah dan *n*

adalah jumlah sub himpunan yang dibentuk dari *X* (pada atribut diskret, *n* adalah jumlah nilai disting pada *a*, sedangkan pada atribut kontinyu, *n* = 2).

Sedangkan rumus *split info(a)* adalah [9]:

$$split\ info(a) = -\sum_{i=1}^{n} \frac{|X_i|}{|X|} \times \log_2\left(\frac{|X_i|}{|X|}\right) \qquad (2)$$

dimana X_i menyatakan sub himpunan ke-*i* pada sampel *X*. Bahasan detil dari komputasi gain ratio ini dapat ditemukan di [4,9].

2.3 Partisi Himpunan Data dan Masalah Skalabilitas

Pada eksekusi Algoritma 1 pada langkah 10 lalu, sampel data pada simpul dipartisi menjadi sub-sub set data yang juga disimpan di memori. Pada pemanggilan algoritma secara rekursif (langkah 13), setiap sub set data lalu menjadi parameter masukan pada pemanggilan algoritma ini. Pemuatan seluruh dan partisi-partisi himpunan data pada memori ini menyebabkan algoritma C4.5 bergantung kepada ketersediaan memori pada sistem. Jika jumlah sampel pada himpunan data sangat besar, maka hal ini dapat menjadi masalah.

2.4 Hasil Penelitian untuk Mengatasi Masalah Skalabilitas

Algoritma GAC (Grouping And Counting). Pada [5], Lu merancang algoritma GAC. Masukan dari GAC adalah sebuah tabel dengan atribut-atribut diskret, yang berisi rekord-rekord yang sudah digabungkan dan tidak ada nilai yang hilang. Keluaran dari algoritma ini adalah sebuah tabel keputusan, yang memiliki atribut prediktor, kelas, support dan *confidence*. Pembuatan tabel keputusan ini dilaporkan cepat dan tidak dibatasi ketersediaan memori, karena menggunakan SQL. Kekurangan yang ditemui pada GAC adalah tidak dihasilkannya pohon atau aturan-aturan untuk dikonversi ke SQL. Pada GAC, untuk mengklasifikasi rekord-rekord yang baru perlu digunakan algoritma lain, yaitu *k*-Nearest Neighboor atau Naive Bayes. Berdasarkan kriteria penilaian algoritma klasifikasi (lihat [4]), kelemahan algoritma ini adalah tidak dihasilkannya model atau aturan yang nilai interpretabilitasnya tinggi.

Algoritma RainForest. Pada [3], Gehrke merancang framework generik untuk mengatasi masalah skalabilitas pada algoritma pohon keputusan. Framework ini direalisasi dengan 3 buah algoritma, yaitu RF-Write, RF-Read dan RF-Hybrid. Dengan menggunakan RF-Write dan RF-Read, RF-Hybrid mengkonstruksi pohon sampai level tertentu dengan mengakses disk secara langsung (dengan RF-Read) dan meneruskan mengakses statistik dari data yang sudah tersimpan di memori (dengan RF-Write). Framework ini dilaporkan dapat mengatasi skalabilitas dan memiliki tingkat efisiensi yang baik. Dalam kaitannya dengan integrasi algoritma klasifikasi ke dalam DBMS, masalah yang ditemui pada framework ini adalah belum memanfaatkan fitur yang disediakan DBMS (pendekatan yang digunakan adalah logika algoritma). Framework ini juga relatif kompleks atau sulit untuk dipahami.

3. PENGEMBANGAN ALGORITMA

Tujuan utama dari integrasi ini adalah untuk "melebur" algoritma data mining agar menjadi fungsi internal ORDBMS yang berkualitas, sehingga pengguna dapat menggunakannya sesuai dengan kebutuhan. Karena DBMS merupakan teknologi yang sudah matang, digunakan secara luas, dapat mengelola data dalam ukuran yang sangat besar, memfasilitasi kueri tabel dengan cepat dan mudah dengan SQL, mendukung arsitektur *client-server*, dan khususnya pada ORDBMS, mendukung prosedur tersimpan berorientasi obyek, maka pengembangan algoritma C4.5 dan integrasinya ke dalam ORDBMS akan memanfaatkan fitur-fitur ini.

3.1 Aljabar Relasional Seleksi dan Proyeksi

Dimisalkan *X* adalah sebuah tabel atau relasi yang memiliki atribut-atribut a_1, a_2, a_3 dan a_4 yang dinyatakan sebagai $X = (a_1, a_2, a_3, a_4)$. Domain atribut-atribut tersebut adalah: $a_1, a_2 \in Z$, $a_1, a_2 > 0$ dan $a_1, a_2 < 1000$, $a_3 \in \{v_1, v_2, ..., v_m\}$ dan $a_4 \in \{v_1, v_2, ..., v_n\}$. Operator seleksi dan proyeksi didefinisikan seperti di bawah ini [2].

Operator Seleksi (atau Restriksi), σ. Operasi seleksi diterapkan pada sebuah tabel dan menghasilkan tupel-tupel atau rekord-rekord yang memenuhi kondisi atau predikat tertentu. Representasi umum dari operasi seleksi adalah $\sigma_{predikat}(X)$, dengan *X* adalah nama tabel dan *predikat* adalah kondisi pada seleksi. Predikat dapat mengandung operator $\wedge$ (AND) dan $\vee$ (OR).

Operator Proyeksi, $\prod$. Operasi seleksi diterapkan pada sebuah tabel dan menghasilkan tabel yang merupakan "sub-himpunan vertikal" (memiliki sebuah atribut atau sebagian atribut) dari tabel asli. Representasi umum dari operasi proyeksi adalah $\prod_{a_1,a_2,...a_s}(X)$, dengan *X* adalah nama tabel dan $a_1, a_2, \ldots, a_s$ adalah atribut-atribut yang dipilih.

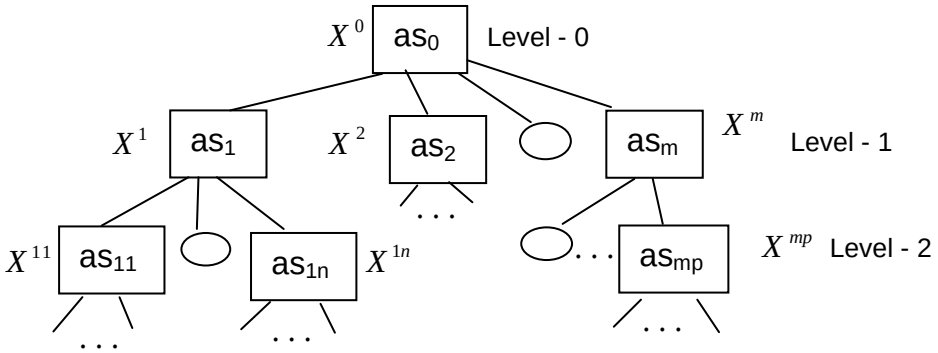
Dalam kaitannya dengan partisi himpunan data yang diperlukan pada konstruksi pohon C4.5, operator seleksi dan proyeksi ini dapat dimanfaatkan untuk membentuk partisi-partisi pada simpul-simpul pohon yang lalu digunakan untuk membangun cabang pada simpul tersebut. Di bawah ini diberikan diskusi mengenai partisi himpunan set data pada pohon keputusan C4.5.

3.2 Partisi Himpunan Data dan Estimasi Ruang Memori

Dalam representasi model, pada Gambar 1 ditunjukkan sebuah pohon keputusan beserta partisi-partisi himpunan datanya sampai kedalaman 2 level. Kotak menyatakan simpul pohon dan ellip menyatakan daun, dimana daun tidak memiliki partisi himpunan data karena tidak memiliki cabang-cabang di bawahnya. Isi kotak, as_i, menyatakan atribut split pada simpul ini, dengan indeks subskript *i* menyatakan nomor simpul. X^i menyatakan himpunan data yang ditelaah untuk membangun cabang di bawah simpul-*i*. (as_i dipilih berdasarkan nilai *gain ratio* pada X^i.) Pada pohon, himpunan data pada simpul-simpul di level-*i* merupakan partisi atau subset dari himpunan data simpul level di atasnya (induknya), pada level-(*i*-1). Atau, dapat pula dinyatakan bahwa himpunan data pada simpul-simpul di level-*i* diperoleh dengan menerapkan sebuah fungsi filter tertentu, H^i, pada himpunan data pada simpul induknya, di level-(*i*-1). Dengan demikian, secara umum X^i dapat dituliskan sebagai

$$X^i = H^i(X^{i-1}) \qquad (3)$$

dimana H^i adalah fungsi yang berisi operasi seleksi (memilih rekord-rekord yang memiliki atribut split pada simpul di induknya dengan nilai tertentu) dan proyeksi (mengeluarkan atribut split yang terdapat pada simpul induknya). Partisi himpunan data dan fungsi filter dibahas lebih lanjut di bawah ini.



Gambar 1. Pohon keputusan dan partisi himpunan data.

Jika diketahui himpunan data $X^0 = (a_1, a_2, a_3, ..., a_k)$, maka dapat didefinisikan himpunan atribut untuk X^0, yaitu $D = \{a_1, a_2, a_3, ..., a_k\}$. Berdasarkan himpunan data dan atribut ini, maka himpunan partisi data dan fungsi-fungsi filter yang digunakan untuk membentuk partisi pada simpul-simpul anak dapat dirumuskan. Selain itu, kebutuhan ruang memori untuk meyimpan partisi-partisi himpunan data juga dapat diestimasi. Untuk ini, diasumsikan bahwa ruang memori yang diperlukan untuk menyimpan sebuah nilai atribut yang dapat bertipe integer, float atau diskret, adalah sebesar 1 *word* (untuk keperluan penyimpanan, nilai atribut diskret dapat diganti dengan sebuah nilai integer yang merepresentasikan nilai atribut diskret). Dengan demikian, estimasi kebutuhan ruang memori pada Level-0 adalah $M_{X^0} = |X^0| \times |D|$ *word* dimana |.| menyatakan kardinalitas himpunan.

Pada Level -1:

Didefinisikan $D^0 = D - \{as_0\}$, dimana $as_0 \in D$.(Sebagai contoh, jika $as_0 = a_3$, maka $D^0 = \{a_1, a_2, a_4, a_5, ..., a_k\}$).

Partisi himpunan, X^i, pada simpul-simpul di level-1 dapat dituliskan sebagai:

- Simpul-1: $X^1 = \sigma_{pred\_1(as_0)}(\prod_{D^0}(X^0))$ atau

$$X^1 = H^1(X^0) \quad \text{dengan}$$

$$H^1(.) = \sigma_{pred\_1(as_0)} \prod_{D^0} \qquad (4)$$

- Simpul-*m*: $X^m = \sigma_{pred\_m(as_0)}(\prod_{D^0}(X^0))$

atau $X^m = H^m(X^0)$ dengan

$$H^m(.) = \sigma_{pred\_m(as_0)} \prod_{D^0} \qquad (5)$$

Predikat yang terdapat pada operator σ, yang dinyatakan sebagai *pred_i*(as_0), adalah predikat yang dikenakan pada atribut split pada simpul induknya, yaitu simpul-0.

Estimasi kebutuhan ruang memori pada setiap simpul di Level-1 adalah $M_{X^i} = |X^i| \times |D^0|$ *word*. Berdasarkan sifat dari operasi seleksi dan proyeksi yang mengurangi jumlah rekord pada tabel, maka $|X^i| < |X^0|$. Karena $|D^0| < |D|$, maka disimpulkan bahwa $M_{X^i} < M_{X^0}$.

Pada Level-2:

Untuk Simpul-*m1 s/d* Simpul –*mp*, didefinisikan $D^m = D^0 - \{as_m\}$, dimana $as_m \in D^0$. Partisi himpunan, X^{ij}, pada simpul-simpul ini dapat dituliskan sebagai:

Pada simpul-*mp*:

$$X^{mp} = \sigma_{pred\_mp(as_m)}(\prod_{D^m}(X^m)) \quad \text{atau}$$

$$X^{mp} = H^{mp}(X^0) \quad \text{dengan}$$

$$H^{mp}(.) = \sigma_{pred\_mp(as_m)} \prod_{D^m} \sigma_{pred\_1(as_0)} \prod_{D^0} \qquad (6)$$

Estimasi kebutuhan ruang memori pada setiap simpul di Level-2 adalah $M_{X^{ij}} = | X^{ij} | \times | D^m |$ *word*. Karena $|X^{ij}| < |X^i|$ dan $|D^m| < |D^0|$, maka $M_{X^{ij}} < M_{X^i} < M_{X^0}$.

Pada Level 3:
Analogi dengan level-1 dan level-2 di atas, maka partisi himpunan data pada Simpul-*mpq* dapat didefinisikan sebagai:

$$X^{mpq} = \sigma_{pred\_mpq(as_p)}(\prod_{D^p}(X^{mp})) \text{ atau}$$

$$X^{mpq} = H^{mpq}(X^0), \text{ dengan}$$

$$H^{mpq}(.) = \sigma_{pred\_mpq(as_p)} \prod_{D^p} \sigma_{pred\_mp(as_m)} \prod_{D^m} \sigma_{pred\_1(as_0)} \prod_{D^0} \quad (7)$$

Estimasi kebutuhan ruang memori pada setiap simpul di Level-2 adalah $M_{X^{ijk}} = | X^{ijk} | \times | D^{mp} |$ *word*. Analogi pada Level-2, maka $M_{X^{ijk}} < M_{X^{ij}} < M_{X^i} < M_{X^0}$.

Pada level-level selanjutnya, partisi himpunan data dan filter pada simpul-simpul dapat dicari dengan melakukan analogi pada level 3 di atas. Juga dapat ditunjukkan bahwa estimasi kebutuhan ruang memori untuk sebuah partisi data pada level yang makin dalam, akan makin kecil.

3.3 Perintah SQL untuk Partisi Himpunan Data

Perintah SQL mendukung implementasi fungsi-fungsi filter di atas dalam bentuk yang sederhana. Sebagai ilustrasi, tinjauan diberikan terhadap fungsi-fungsi filter berikut ini. Pada level-1: $H^m(.) = \sigma_{pred\_m(as_0)} \prod_{D^0}$.

Pada level-2:

$$H^{mp}(.) = \sigma_{pred\_mp(as_m)} \prod_{D^m} \sigma_{pred\_1(as_0)} \prod_{D^0}$$

Pada level-3:

$$H^{mpq}(.) = \sigma_{pred\_mpq(as_p)} \prod_{D^p} \sigma_{pred\_mp(as_m)} \prod_{D^m} \sigma_{pred\_1(as_0)} \prod_{D^0}$$

Pada fungsi-fungsi tersebut, terlihat bahwa filter pada level yang lebih bawah (anak) memberikan tambahan sepasang operator seleksi dan proyeksi pada filter induknya. Pada perintah SQL, penambahan operator berimplikasi pada pengurangan atribut pada perintah SELECT dan penambahan string *operand* pada klausul WHERE.

Konsep partisi set data dan fungsi filter yang telah dibahas dapat dimanfaatkan untuk mengembangkan skalabilitas Algoritma 1. Karena masalah utama yang terkait dengan skalabilitas pada algoritma ini adalah pemuatan seluruh sampel-sampel set data ke dalam memori, maka hal ini akan dihindari. Pada DBMS, nilai-nilai atribut dan frekuensi sampel dengan nilai atribut yang sama pada set data (untuk pemilihan *test-attribute* pada langkah 7) dapat langsung dibaca dari basisdata melalui mekanisme yang diatur di DBMS. Tetapi pembacaan langsung ke basisdata dapat berakibat tingginya akses I/O dan memperlambat eksekusi (hal ini disimpulkan melalui eksperimen-eksperimen), sehingga pemuatan partisi set data masih akan diperlukan, tapi dibatasi untuk sub set data yang berukuran lebih kecil. Maka, pada algoritma yagn dikembangkan, pemanggilan algoritma secara rekursif akan diberi parameter yang berupa string filter (yang diperbarui pada setiap pemanggilan secara rekursif).

Dengan memperhatikan hal di atas, maka ide dasar pengembangan algoritma untuk konstruksi pohon, yang memiliki tingkat skalabilitas yang lebih baik di DBMS adalah:

a. Sampai dengan kedalaman tertentu (diset sebagai parameter pengguna), konstruksi pohon dilakukan dengan cara membaca nilai-nilai atribut dan frekuensi sampel langsung dari basisdata (langkah 1 s/d 18).
b. Selanjutnya, tiap cabang pohon dilanjutkan dengan menggunakan Algoritma 1 (langkah 19), dimana seluruh sampel pada set data (yang menjadi parameter pada pemanggilan algoritma) akan dimuat di dalam memori.
c. Pada butir (a), digunakan filter yang berisi klausul untuk mengekstraksi isi basisdata yang memenuhi kondisi tertentu (pada implementasinya akan berupa klausul WHERE seperti yang dibahas). Pada pemanggilan yang pertama (eksekusi rekur ke-1), filter tidak ada isinya (kosong). Lalu, pada setiap akhir rekur, filter akan diperbarui dan dijadikan parameter masukan pada pemanggilan algoritma ini secara rekursif (langkah ke 17 dan 18).

Pada Algoritma 2 terlihat bahwa logika algoritma sederhana dan mudah diimplementasikan. Algoritma ini sederhana karena dirancang dengan pemanfaatan SQL untuk filter yang digunakan pada parameter pemanggilan algoritma secara rekursif.

Algoritma: Generate_decision_tree_ext
Narasi: Buat pohon keputusan dari data pelatihan yang diberikan.
Masukan: Data pelatihan (*samples)*, kedalaman pohon maksimum yang dibangun dengan mengakses basisdata secara langsung (*dSQL*), string klausul WHERE *(Filter)* yang digunakan untuk menyaring rekord-rekord yang tidak dibutuhkan di *samples*, daftar atribut pada *samples* (*attribute-list*).
Keluaran: Sebuah pohon keputusan
Metoda:

```
(1)  buat sebuah simpul N;
(2)  if dSQL < Max_depthDb
(3)     if attribute-list kosong then
(4)        return N sebagai simpul daun dengan label kelas terbanyak di samples;
(5)     baca label dan jumlah kelas di samples dengan memakai Filter
(6)     if samples memiliki kelas yang sama, C, then
(7)        return N sebagai simpul daun dengan label kelas C;
(8)     dengan Filter, pilih test-attribute dari attribute-list yang memiliki gain ratio terbesar;
(9)     beri label simpul N dengan test-attribute;
(10)    for each nilai atribut a_i pada test-attribute;
(11)       tambahkan cabang pada simpul N untuk kondisi test-attribute = a_i;
(12)       dengan Filter, lakukan uji daun pada himpunan sampel dengan kondisi test-attribute = a_i;
(13)       if cabang adalah kandidat untuk daun then
(14)             tempelkan daun dengan label kelas terbanyak pada himpunan sampel;
(15)       else
(16)             perbarui Filter untuk partisi himpunan data pada simpul ini (lihat III.3.3), simpan di NewFilter
(17)             perbarui attribute-list untuk simpul ini, simpan di new_ attribute-list
(18)             tempelkan simpul yang dikembalikan Generate_decision_tree_ext (samples, (dSQL +1),
                    NewFilter );
(19) else
(20)    new_samples = samples yang disaring dengan Filter
(21)    tempelkan simpul yang dikembalikan oleh Generate_decision_tree (new_samples, attribute-list);
```

Algoritma 2. Algoritma konstruksi pohon keputusan yang dikembangkan.

4. IMPLEMENTASI

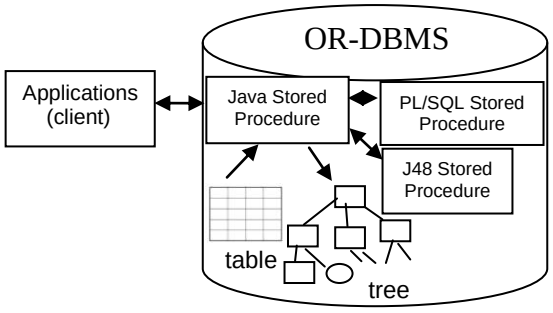
4.1 Arsitektur Sistem

Pada penelitian ini, Algoritma 2 (bagian dari algoritma C4.5) diimplementasikan sebagai prosedur-prosedur tersimpan berbasis SQL dan Java pada Oracle 10*g* (bahasan tentang prosedur tersebut dapat ditemukan pada [6,7,8]). Pemilihan kedua jenis prosedur ini dimaksudkan untuk mengambil keuntungan dari keduanya. Prosedur berbasis SQL dapat berisi perintah-perintah SQL yang memanfaatkan pengoptimasi kueri dan indeks tabel, karena itu dapat mengakses tabel dengan cepat. Prosedur Java berbasis memori, sehingga efisien dalam menangani data yang sudah dimuat ke dalam memori. Program utama adalah Java yang memanggil prosedur tersimpan PL/SQL (berisi kode SQL untuk mengakses partisi data, fungsi agregat COUNT dan mengembalikan hasilnya ke prosedur pemanggil) dan *J48* (implementasi Algoritma 1 dengan Java). *J48* diambil dari Weka system [12] yang dimodifikasi untuk dijadikan prosedur tersimpan. Pada arsitektur *client-server* 2-tingkat, program-program aplikasi *client* dapat memanggil program utama (lihat Gambar 2).

Pada implemententasi Algoritma 2, profil tabel yang dipilih untuk ditangani adalah:

- Atribut diskret. Ini dimaksudkan untuk mengambil keuntungan dari indeks bitmap yang mendukung akses tabel pada kolom bertipe diskret yang cepat.
- Tidak ada nilai yang hilang, supaya fungsi agregat COUNT dapat digunakan secara langsung.

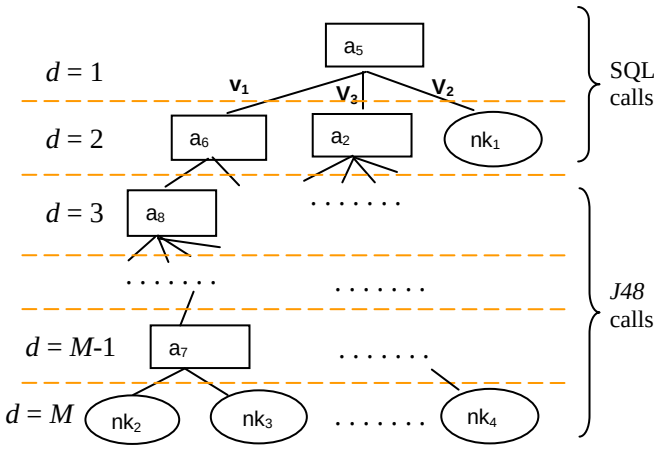
Dengan profil yang dipilih di atas, implementasi ini dapat berguna untuk set data berukuran besar yang sudah digeneralisasi (dimana nilai atribut kontinyu sudah dijadikan atribut diskret) dan data warehouse (yang saat ini juga sudah terintegrasi di dalam banyak ORDBMS), dimana data yang tersimpan sudah melalui tahap pembersihan dan transformasi.



Gambar 2. Arsitektur sistem yang mengimplementasikan Algoritma 2.

4.2 Struktur Data untuk Pohon Keputusan

Ilustrasi pembentukan pohon dengan Algoritma 2 diberikan pada Gambar 3, dimana pohon akan dikonstruksi oleh dua tipe prosedur tersimpan, yaitu prosedur berbasis SQL yang membaca nilai dan frekuensi rekord pada set data langsung dari basisdata dan *J48* yang memuat sampel-sampel ke memori lalu menelaahnya untuk melanjutkan konstruksi cabang pohon.



Gambar 3. Sebuah pohon dikontruksi dengan memanggil prosedur tersimpan berbasis SQL (di sini sampai level 2, atau *dSQL* = 2) dan *J48* untuk melanjutkan konstruksi pohon di tiap cabang.

Adapun struktur data pohon keputusan dideklarasikan sebagai kelas Java, `Tree`. Atribut-atribut utama kelas `Tree` adalah:

- `m_isLeaf` yang bertipe boolean (*true* jika daun, *false* jika node bukan daun),
- `m_ClassAtrValue` yang berisi nilai kelas daun dan bertipe string (*null* jika node ini bukan daun),
- `m_attrName` yang berisi nama atribut split (pemenang) untuk node ini dan bertipe string (*null* jika node ini daun),
- `m_sons` yang berisi array dari obyek `Tree` dan node anak-anak (*null* jika node ini daun),
- `m_sonJ48` yang berisi obyek dari kelas *J48* (*null* jika node ini daun atau tidak punya anak obyek *J48*).

Sedangkan metoda-metoda (*methods*) utamanya adalah:

- `traverseTree(Tree)` yang berfungsi untuk melakukan traversal sturktur pohon dan menampilkan setiap nilai node dan
- `classifyACase()` yang berfungsi untuk mengklasifikasi sebuah rekord baru (mengembalikan nilai kelas dalam bentuk string).

5. EKSPERIMEN

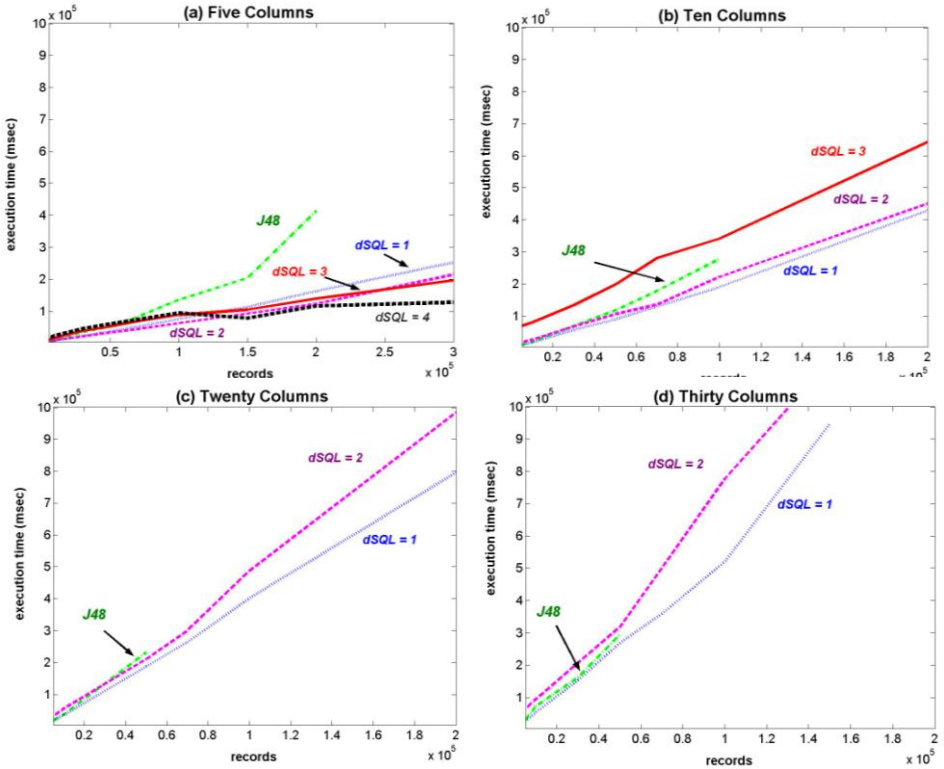
Pada penelitian ini dilakukan dua kelompok eksperimen untuk mengobservasi skalabilitas dan efisiensi Algoritma 2, yaitu dengan set data sintetik acak dan hasil eksekusi fungsi. (Pemilihan data sintetik ini berdasarkan alasan bahwa sampai saat ini penulis tidak dapat menemukan data nyata dalam ukuran yang sangat besar.) Set data untuk eksperimen-eksperimen ini berupa tabel-tabel basisdata. Untuk tiap tabel, program utama (prosedur tersimpan) Java dieksekusi beberapa kali, waktu eksekusi dicatat, lalu nilai rata-rata waktu eksekusi dihitung. Pada eksperimen ini juga dilakukan perbandingan hasil eksekusi antara *J48*, yang merupakan implementasi dari Algoritma 1 dan Algoritma 2. Seluruh eksperimen dilakukan pada komputer personal dengan sistem operasi Windows XP, CPU Pentium 2.4 GHz dan memori 1 Gbyte. ORDBMS yang digunakan adalah Oracle 10*g*.

Pada kelompok eksperimen pertama, dibuat sejumlah tabel sintetik acak. Tabel-tabel itu memiliki 5 sampai 30 kolom dengan 5000 sampai 30.000 rekord. Hasil eksperimen diberikan pada Gambar 4. Pada kelompok eksperimen kedua, set data sintetik dibuat dengan teknik yang dirancang oleh Agrawal pada [1], yang banyak digunakan di eksperimen-eksperimen pada literatur. Pada set data ini, atribut kelas dibuat dengan *Function-1* (untuk memproduksi pohon yang berukuran kecil) dan *Function-7* (untuk memproduksi pohon yang berukuran besar). Nilai atribut numerik kemudian diubah ke nilai diskret. Waktu eksekusi yang ditunjukkan pada Gambar 5 diperoleh dengan *dSQL* = 3.

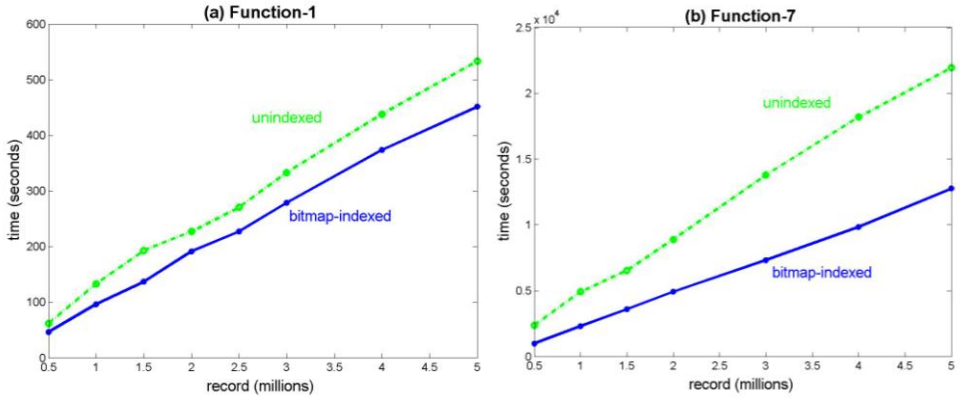
Selain eksperimen di atas, pada penelitian ini juga dilakukan eksperimen untuk menguji akurasi Algoritma 2 dengan lima set data yang diperoleh dari Internet. Mengingat keterbatasan tempat, hasil eksperimen rinci tidak dapat diberikan di sini. Tapi secara umum dapat disimpulkan bahwa akurasi Algoritma 2 baik atau tidak lebih buruk daripada Algoritma 1.

Bahasan Skalabilitas. Pada Gambar 4 ditunjukkan bahwa *J48* (Algoritma 1) hanya dapat menangani tabel sampai dengan berukuran 200.000 rekord untuk tabel dengan 5 kolom, 100.000 untuk 10 kolom, 50.000 untuk 20 dan 30 kolom. Sedangkan Algoritma 2 (dengan *dSQL* = 1, 2 dan 3) dapat menangani tabel dengan ukuran yang lebih besar. Pada Gambar 5, ditunjukkan bahwa Algoritma 2 dapat menangani tabel yang berukuran sangat besar, yaitu 1 s/d 5 juta rekord. Dengan demikian, melalui eksperimen, dapat disimpulkan bahwa Algoritma 2 dapat meningkatkan skalabilitas dalam mengkonstruksi pohon.

Bahasan Efisiensi. Pada kelompok eksperimen pertama (Gambar 4), hasil yang menarik didapatkan untuk eksperimen dengan tabel 5 dan kolom. Konstruksi pohon dengan Algoritma 2 (*dSQL* = 1 dan 2) memerlukan waktu yang lebih sedikit dibandingkan dengan waktu yang diperlukan Algoritma 1 (*J48*). Hal ini dapat terjadi karena indeks bitmap, yang diakses oleh perintah SQL, memiliki ukuran yang jauh lebih kecil dibandingkan dengan tabel asli (lihat Tabel 1) atau penggunaan I/O (masukan / keluaran) banyak dikurangi. Pada kelompok eksperimen kedua (Gambar 5), terlihat bahwa waktu untuk konstruksi pohon pada tabel yang berisi set data *Function-1* (baik yang diindeks maupun tidak) lebih cepat dibandingkan dengan set data *Function-7*. Pada juga ditunjukkan bahwa perbandingan waktu eksekusi antara tabel yang diindeks dengan bitmap lebih baik dibandingkan dengan yang tidak diindeks.



Gambar 4. Perbandingan waktu konstruksi pohon antara Algoritma 1 (*J48*) dengan Algoritma 2 (dengan *dSQL* = 1, 2, 3) untuk tabel dengan (a) 5, (b) 10, (d) 20, (e) 30 kolom.



Gambar 5. Waktu eksekusi Algoritma 2 untuk tabel berukuran sangat besar.

Tabel 1. Jumlah blok tabel dan indeks bitmap pada tabel dengan 5 kolom.

Col	#Dis-tincts	Bitmap index blocks of 5-300 thousand records tables								
		5,000 **(40)**	10,000 **(80)**	30,000 **(239)**	50,000 **(397)**	70,000 **(556)**	100,000 **(794)**	150,000 **(1191)**	200,000 **(1588)**	300,000 **(2378)**
1	4	1	1	4	5	6	10	13	17	25
2	9	1	2	7	9	13	17	26	30	47
3	5	1	2	5	6	8	11	16	22	31
4	7	1	2	7	7	11	14	21	28	41
5	10	1	2	5	10	15	20	25	35	50
Total of bitmap index blocks		**5**	**9**	**28**	**37**	**53**	**72**	**101**	**132**	**194**

Keterangan: 5,000 **(40)** berarti bahwa tabel yang berisi 5000 rekord menempati 40 blok pada disk.

6. KESIMPULAN DAN PENELITIAN LANJUTAN

Algoritma konstruksi pohon yang dikembangkan, yaitu Algoritma 2, dapat meningkatkan skalabilitas Algoritma 1 (C4.5). Algoritma 2 juga dapat meningkatkan efisiensi C4.5 khusus pada tabel yang diindeks dengan bitmap dan memiliki jumlah kolom dan nilai disting sedikit. SQL berperan dalam memperbaiki skalabilitas dan efisiensi pada tabel dengan profile tertentu. (Tabel berukuran besar dengan jumlah kolom dan nilai disting kecil dapat ditangani dengan baik oleh prosedur berbasis-SQL, khususnya fungsi agregat COUNT).

Untuk penelitian lanjutan, kedalaman pohon yang dikonstruksi dengan akses basisdata secara langsung perlu dioptimasi dengan algoritma. Perumusan kedalaman ini harus memperhitungkan memori yang tersedia dan efisiensi konstruksi program. Prosedur berbasis-SQL perlu untuk dikembangkan agar dapat menangani nilai atribut yang hilang dan atribut kontinyu. Penanganan nilai atribut yang hilang dan atribut kontinyu ini memperlukan komputasi yang kompleks, karena mengandung proses pengurutan, komputasi bobot di tiap kolom, dll. DBMS menyimpan statistik tabel di kamus data. Ini mungkin berguna dalam mengurangi kompleksitas komputasi tersebut. Penelitian untuk membandingkan Algoritma 2 dengan algoritma yang lain maupun teknik pemrograman yang lain (misalnya pemrograman paralel dan terdistribusi) dan algoritma klasifikasi yang lain juga perlu dilakukan. Penelitian lanjutan juga dapat ditujukan untuk merumuskan profil-profil tabel yang dapat ditangani teknik atau algoritma tertentu dengan baik. Jika ini sudah memberikan hasil, penelitian selanjutnya dapat dilakukan untuk merumuskan teknik pengoptimasi algoritma klasifikasi, yang cara kerjanya mirip dengan pengoptimasi kueri SQL yang sudah ada di DBMS. Dengan demikian, kelak, pengguna DBMS dapat mempercayakan pemilihan algoritma atau teknik klasifikasi yang paling sesuai untuk set data mereka ke DBMS.

7. DAFTAR PUSTAKA

1. **Agrawal, R., Imielinski, T., Swami, A.,** "Database mining: A performance Perspective", IEEE TKDE, Dec., 1993.
2. **Conolly, T.; Begg, C.,** "Database Systems A Practical Approach to Design, Implementation and Management", 3rd ed., Addison Wesley Pub., USA, 2002.
3. **Gehrke, J., Ramakrishnan, R., Ganti, V.,** RainForest – "A Framework for Fast Decision Tree Construction of Large Datasets", *Proc. of the 24th VLDB Conf.*, New York, USA, 1998.
4. **Han, J., Kamber, M.,** "Data Mining Concepts and Techniques", Morgan Kaufmann Pub., USA, 2001.
5. **Lu, H., Liu, H., "**Decision Tables: Scalable Classification, Exploring RDBMS Capabilities", *Proc. of the 26th VLDB Conf.*, Cairo, Egypt, 2000.
6. **Oracle Co.,** "Java Stored Procedures Developer's Guide Release 2 (9.2)", USA, Maret, 2002.
7. **Oracle Co.,** "PL/SQL User's Guide and Reference 10g Release 2", USA, Juni, 2005.
8. **Oracle Co.,** "Oracle 9i Application Developer's Guide: Object-Relational Features Release 2 (9.2)", Maret, 2002.
9. **Quinlan, J.R.,** *C4.5:* "Programs for Machine Learning", Morgan Kaufmann Pub., USA, 1993.
10. **Ruggieri, S., "**Technical Report TR-00-01: Efficient C4.5", Dipartimento di Informatica, Universita di Pisa, Itali, 2001.
11. **Saleem, M.A., Hameed, J**., "Integration of Data Mining and Object-Relational Database Systems", Ghulam Ishaq Khan Institute of Engineering Science and Technology, Swabi, NWFP, Pakistan, 2003.
12. **The Univ. of Waikato,** WEKA Machine Learning Software ver.3.4, Dept. of Computer Science, The Univ. of Waikato, Hamilton, New Zealand, www.cs.waikato.ac.nz/ml/weka, 2003.