

BAB II

LANDASAN TEORI

II.1 Sistem Berbasis Pengetahuan

Pada subbab ini akan dibahas teori tentang definisi dan karakteristik *Expert System*(ES), arsitektur ES, representasi pengetahuan dan *problem solving*.

II.1.1 Definisi dan Karakteristik SBP

Sistem Berbasis Pengetahuan (SBP) atau *Knowledge-based System* adalah sistem yang melakukan sebuah tugas dengan menggunakan aturan-aturan yang diubah ke sebuah representasi simbolik pengetahuan, dan juga memakai banyak metode algoritmik atau statistik. *Expert System* adalah program komputer yang merepresentasikan dan melakukan penalaran pengetahuan suatu subjek tertentu untuk memecahkan persoalan atau memberikan saran berdasar oleh pengetahuan yang diambil dari pakar di bidang tersebut. Contohnya sebuah program yang bisa menerjemahkan cuaca termasuk SBP karena tidak membutuhkan tenaga ahli khusus dalam menentukan cuaca sesuai ilmu meteorologi, tetapi program yang dapat memprediksi cuaca adalah ES karena membutuhkan keahlian khusus oleh pakar dalam bidang meteorologi. Oleh karena itu, dapat dikatakan ES merupakan varian dari SBP.

ES dapat berbeda dari aplikasi yang lebih konvensional karena ES memiliki karakteristik sebagai berikut [PEJ99] :

- ES memiliki performansi yang tinggi, yaitu memiliki kualitas solusi yang minimal sama dengan kualitas solusi yang diberikan oleh pakar, dengan respon yang *reasonable*.
- *Understandable*, ada penjelasan solusi bagi user ataupun pengembang sehingga dapat meningkatkan kepercayaan.
- ES mensimulasikan *human reasoning* tentang suatu *domain* masalah, daripada mensimulasikan *domain* itu sendiri. Basis pengetahuannya mudah dimodifikasi.

ES menyelesaikan masalah dengan metode heuristic atau perkiraan, yang tidak seperti solusi algoritmik, tidak ada jaminan untuk sukses.

ES berbeda dari program Intelegensia Buatan yang lain karena :

- ES mengurus masalah kompleksitas yang nyata yang biasanya membutuhkan banyak keahlian dari manusia.
- ES harus mempunyai kecepatan dan *reliability* yang tinggi untuk menjadi kakas yang berguna.
- ES harus bisa menjelaskan solusi atau saran untuk meyakinkan pengguna bahwa *reasoning*-nya benar dengan fakta-fakta yang mendukung.

ES dibutuhkan oleh manusia sebab ketersediaan pakar terbatas. Sulitnya menemukan pakar menyebabkan biaya menggunakan pakar tersebut cukup mahal. Pemakaian ES untuk menggantikan pakar dalam pemecahan suatu kasus yang memiliki *domain* masalah dunia nyata merupakan salah satu jalan keluar yang praktis.

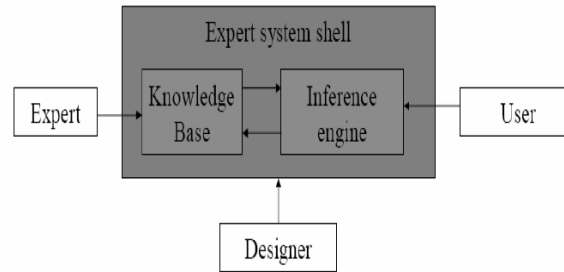
Beberapa alasan dipakainya ES adalah adanya hal sebagai berikut :

- Adanya pakar, biaya yang digunakan lebih mahal, mungkin bisa menjadi tidak produktif karena faktor usia, dan sangat langka untuk menemukan seorang pakar di bidangnya
- Adanya pengetahuan yang terstruktur dengan baik
- Adanya *domain* masalah yang terbatas
- Masalah yang tidak bisa diselesaikan dengan program konvensional

ES saat ini telah dipakai dunia untuk membantu mengambil keputusan, contohnya adalah *Subsystem Treatment Selection* pada *Hopper : DSS for Rangeland Grasshopper Management* di mana sistem ini bertugas untuk mengendalikan populasi belalang pada lokasi produksi pakan ternak di Amerika Utara bagian barat [MLK07]. Selain membantu pengambilan keputusan, ES juga dapat digunakan untuk memberikan saran di mana pengguna dapat memberikan *input* secara interaktif, contohnya adalah *math tutor system* [MLK07].

II.1.2 Arsitektur SBP

Berikut akan dijabarkan tentang arsitektur *Expert System* (lihat gambar II-1 diambil dari [MLK07]).



Gambar II-1 Arsitektur Dasar *Expert System*

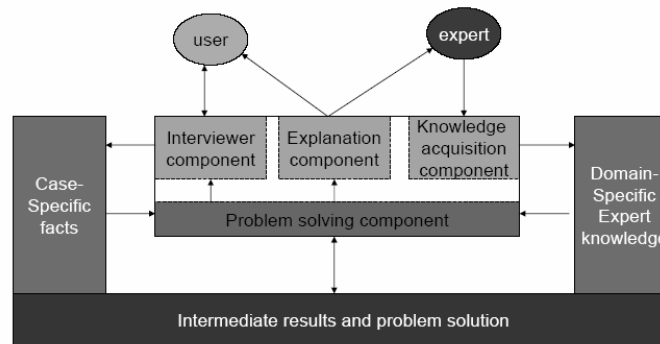
Pada arsitektur dasar ES di atas terdapat beberapa komponen, yaitu :

1. Sebuah *shell* ES di mana ada basis pengetahuan dan mesin inferensi yang mengelola basis pengetahuan tersebut.
 - i. Basis pengetahuan memuat berbagai *rules* dan *facts*
 - ii. Mesin inferensi memproses *rules* dan *facts* tersebut setelah ada interaksi dari *user*.
2. Seorang pakar (*expert*) menuangkan pengetahuannya pada basis pengetahuan.
3. Seorang *designer* merancang keseluruhan antarmuka ES agar dapat dipakai oleh pengguna (*user*).
4. *User* memberi masukan berupa *facts* dan melakukan interaksi terhadap mesin inferensi.

Seiring berkembangnya zaman, ditemukan beberapa masalah pada ES yaitu [MLK07]:

- Pakar sulit mengekspresikan kepakarannya menjadi suatu pengetahuan yang abstrak.
- Penggunaan knowledge engineer tergolong mahal dan adanya kemungkinan kesalahan interpretasi.
- Asumsi pengetahuan sudah lengkap di awal penggunaan.
- Perlunya pemeriksaan pengetahuan karena pengetahuan terus berkembang.

Oleh karena masalah-masalah di atas, para peneliti mengembangkan arsitektur *Expert System* seperti gambar II-2 di bawah ini [MLK07].



Gambar II-2 Arsitektur *Expert System*

Komponen-komponen baru yang ada di atas yaitu :

1. *Interviewer component* yaitu komponen yang berperan sebagai *interface*, dengan kata lain komponen yang memberi kemudahan kepada user untuk memasukkan fakta yang diinginkannya.
2. *Knowledge acquisition component* yaitu komponen sebagai *interface* antar basis pengetahuan dan *expert* dapat memodifikasi sendiri basis pengetahuan melalui komponen ini.
3. *Case-specific facts* yaitu komponen yang berisi fakta-fakta masukan dari user.
4. *Domain-specific expert knowledge* yaitu komponen yang berisi pengetahuan dari pakar.
5. *Problem solving component* yaitu komponen yang berperan sebagai mesin inferensi dalam arsitektur ini. Mengelola fakta dari *user* dan *rule* yang diberikan *expert*
6. *Intermediate result and problem solution* yaitu komponen yang menerima hasil sementara dari pengelolaan yang dilakukan *problem solving component*. Hasil atau solusi sementara di-*generate* dari fakta dan rule yang diproses di *problem solving component*. Hasil sementara ini akan dikirimkan kembali ke *user* dan *expert* untuk di-*review*.
7. *Explanation component* yaitu komponen yang memberikan penjelasan tentang hasil sementara kepada *user* dan *expert*.

Kelebihan dari arsitektur ES yang digambarkan di atas adalah:

- Sistem dikontrol oleh *user*, berdasarkan fakta yang ada
- Terdapat sebuah distribusi dan menjaga kepakaran itu sendiri

- Adanya penjelasan solusi
- Dapat memecahkan masalah dengan data dan pengetahuan yang tidak lengkap dan tidak tentu

Akan tetapi terdapat juga beberapa kekurangan yaitu:

- Hasil sistem tergantung kemampuan pakar sebagai sumber pengetahuan
- *Domain* terbatas
- Proses akuisisi dan *maintenance* sulit, seperti yang terjadi pada *Complex Control Flow : Rule Spaghetti*.

II.1.3 Representasi Pengetahuan

Dalam bidang ES, representasi pengetahuan menekankan adanya beberapa cara sistematis untuk mengkode apa yang diketahui oleh seorang ahli tentang beberapa *domain* masalah. Representasi mengimplikasikan organisasi. Representasi pengetahuan harus dapat memastikan bahwa pengetahuan dapat diakses dan mudah digunakan melalui mekanisme alami. Basis pengetahuan harus diindeks secara luas dan *content-addressable*, jadi program apa saja yang menggunakannya dapat mengatur bagaimana pengetahuan yang berbeda dapat diaktifkan tanpa mengetahui bagaimana pengetahuan itu disimpan.

Sebuah representasi didefinisikan sebagai sebuah set dari konvensi sintatik dan semantik yang membuatnya mungkin untuk menjelaskan sesuatu [PEJ99]. Dalam intelegensia buatan, sesuatu tersebut mengandung arti kondisi dari sebuah *domain* masalah, contohnya, objek di domain tersebut, propertinya, dan keterhubungan yang ada antarobjek.

Sintaks dari sebuah representasi berupa sebuah kumpulan *rules* untuk mengkombinasikan simbol dalam bentuk ekspresi dalam bahasa representasi. Hanya ekspresi yang dibentuk dengan baik mempunyai arti.

Sintaks yang biasa digunakan dalam intelegensia buatan adalah konstruksi *predicate-argument* dalam bentuk :

<sentence> ::= <predicate>(<argument>, ..., <argument>)

Misalnya predikat *beri* mungkin berupa relasi tiga argumen dimana argumen pertama adalah nama seseorang, argumen kedua adalah seseorang yang lain dan argumen yang ketiga adalah nama sesuatu benda, contoh :

`beri(orang1, orang2,sesuatu)`

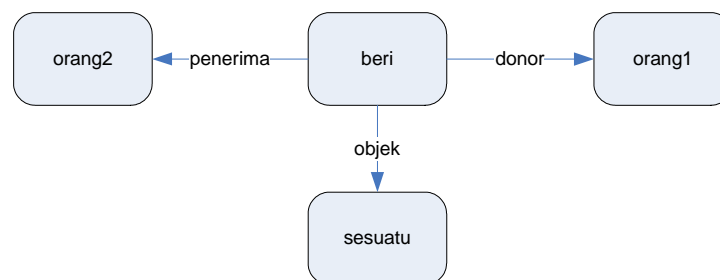
Simbol di atas menyatakan bahwa *orang1* memberikan *sesuatu* kepada *orang2*.

Beberapa bentuk representasi pengetahuan akan dijelaskan dengan singkat di bawah ini, untuk penjelasan lebih lanjut tentang representasi pengetahuan dan jenis-jenisnya terdapat dalam buku *Introduction to Expert Systems* [PEJ99].

II.1.3.1 Semantic Networks

Jenis yang paling sederhana dari objek terstruktur adalah *semantic network* yang dikembangkan pada awal tahun 1960-an untuk merepresentasikan arti dari kata-kata berbahasa Inggris. Hal ini sangat penting dalam hal sejarah dan juga dalam memperkenalkan ide dasar dari *class hierarchies* dan *inheritance*.

Semantic nets adalah sebuah jaringan yang biasanya digunakan untuk menstrukturkan jenis informasi yang lebih umum. Ini adalah kumpulan dari simpul dan penghubung di mana simpul menandakan konsep dan penghubung menandakan hubungan antarsimpul. *Semantic nets* sangat baik dalam merepresentasikan hubungan antardua objek, tetapi agak sulit jika mau merepresentasikan hubungan antara tiga atau lebih objek.



Gambar II-3 Contoh Fragmen-Fragmen *Semantic Network*

Gambar II-3 yang diambil dari [PEJ99] memperlihatkan fragmen-fragmen dari sebuah *semantic net*. Fragmen tersebut merepresentasikan kata kerja "beri", di mana terdapat tiga peran: seorang donor, seorang penerima, dan sebuah objek untuk diberi. Label

simpul pada akhir penghubung ini menyatakan apa jenis entitas yang bisa mengisi peran dalam suatu pertanyaan. Oleh karena itu donor dan penerimanya adalah orang, sementara yang diberikan adalah sebuah benda.

Kesimpulannya, *semantic nets* bisa menyederhanakan representasi pengetahuan tentang sebuah objek yang dapat diekspresikan sebagai hubungan biner. Relasi *subclass* dan *instance* memperbolehkan penggunaan *inheritance* untuk mendapatkan fakta/relasi baru dari yang telah direpresentasikan secara eksplisit. Bagaimanapun juga, *nets* tidak mempunyai sebuah semantik yang jelas.

II.1.3.2 Frame

Frame adalah jenis dari *semantic nets* yang merupakan cara paling populer untuk merepresentasikan pengetahuan non-prosedural dalam sebuah ES. Dalam sebuah *frame*, seluruh informasi relevan untuk sebuah konsep khusus disimpan dalam entitas kompleks yang tunggal, disebut *frame*. *Frame* sangat mirip dengan struktur data dan mendukung *inheritance*. *Frame* sering digunakan untuk merekam pengetahuan tentang objek atau *event* yang khas, seperti burung yang khas, atau restoran yang khas.

Beberapa pengetahuan mendasar tentang orang dapat direpresentasikan dalam *frame* sebagai berikut:

```

Orang
  subclass:      Manusia
  breath:       yes
  moving:       yes

Sesuatu
  subclass:      Benda
  breath:       no
  moving:       no
  genre:        maskulin

Orang1
  instance:     Orang
  gender:       male
  artis:       tidak

Orang2
  instance:     Orang
  gender:       female
  artis:       ya

```

Sebuah *frame* (seperti Orang) mempunyai sebuah nomor dari atribut atau *slot* seperti gender di mana slot ini mungkin diisi dengan nilai yang khusus, seperti male. Dapat digunakan sebuah "*" untuk mengindikasikan atribut tersebut bahwa itu adalah

true milik dari sebuah anggota khas dari kelas, dan tidak perlu dimiliki oleh semua anggota. Banyak sistem *frame* dapat membedakan antara nilai atribut khas dan nilai *definite* yang harus *true*.

Sistem *frame* adalah sistem yang cukup rumit dan pintar. *Frame* juga mendukung *inheritance reasoning* yang fleksibel. Berbeda dari *nets*, dalam sistem *frame* penyelesaian perbedaan antara *slot* dan objek dilakukan dengan membedakan antara nilai *default* dan *definite*, dan dengan memperbolehkan *pengguna* untuk membuat slot *first class citizens*, memberikan properti khusus pada *slot* dengan menulis definisi berbasis *frame*.

II.1.3.3 Rule Based System

Peraturan (*rules*) mempunyai peran langsung dalam produksi sifat dalam sebuah intelegensia buatan. Sebuah *agent* dapat melakukan aksi tertentu karena dia mempunyai sebuah representasi peraturan yang relevan dengan pembentukan sifat dalam pertanyaan yang diberikan.

Production rules adalah sebuah ketentuan yang digunakan dalam teori otomata, tata bahasa resmi, dan desain bahasa pemrograman, sebelum digunakan dalam layanan model psikologi dan ES. *Production rules* berfungsi untuk menimbulkan sejumlah aturan dari sifat-sifat. Ketika diberikan sejumlah *input*, aturannya menentukan bagaimana output yang harus dibuat. Dalam aplikasi ES, aturan tersebut biasanya menentukan bagaimana struktur simbol yang merepresentasikan masalah yang ada harus dimanipulasi sehingga bisa memberikan representasi itu lebih dekat ke solusi.

Sebuah *production system* terdiri atas sebuah *rule set*, kadang-kadang disebut *production memory*, sebuah *rule interpreter* yang menentukan kapan untuk memakai *rule* tertentu, dan sebuah *working memory* yang menyimpan data, pernyataan akhir dan hasil lanjutan yang mengatasi kondisi masalah yang ada. *Working memory* adalah struktur data pusat di mana diperiksa dan dimodifikasi oleh produksi. Aturan-aturan dipicu oleh data ini, dan *rule interpreter* mengatur aktivasi dan seleksi aturan dari setiap siklus.

Aturan dalam sebuah *production system* punya bentuk umum :

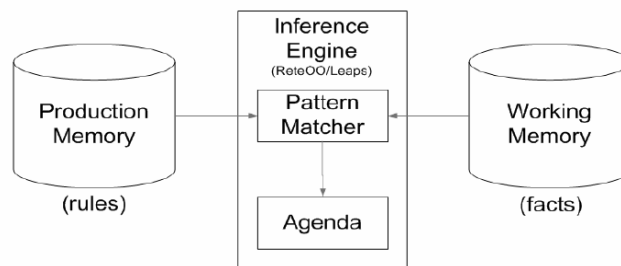
$$P_1, \dots, P_m \rightarrow Q_1, \dots, Q_n$$

Keterangan:

if *premises* P_1 and ... and P_m are true,
then perform *actions* Q_1 and ... and Q_n .

Premises kadang-kadang disebut juga kondisi, dan *actions* disebut kesimpulan. Sebuah proposisi benar atau mungkin jika kondisi tertentu ditemukan lalu satu jenis aksi akan disimpulkan untuk dilakukan.

Sebuah *Rule-Based System* (RBS) dapat dibentuk dengan menggunakan sebuah *assertions set*, yang secara kolektif membentuk *working memory*, dan sebuah *rule set* yang menentukan aksi pada *assertion set*. RBS secara relatif adalah model sederhana yang bisa diadaptasi ke banyak masalah. Secara keseluruhan, RBS dapat dikerjakan dengan mudah untuk masalah yang pengetahuannya dalam area masalah tersebut dapat ditulis dalam bentuk peraturan *if-then* dan untuk area masalah yang tidak besar. Jika ada terlalu banyak peraturan, pemeliharaan sistem akan rumit dan terdapat banyak *failure* dalam kerjanya.



Gambar II-4 Arsitektur *Rule Based Systems*

Untuk membuat sebuah RBS dalam suatu masalah, harus dipunyai sebagai berikut (lihat gambar II-4 yang diambil dari [MLK07]) :

1. Sebuah set dari fakta untuk merepresentasikan *initial working memory*. Ini seharusnya apa saja yang relevan pada state awal sistem.
2. Sebuah set dari peraturan. Ini harus meliputi seluruh aksi yang harus dilalui dalam jangkauan masalah, tapi tidak ada yang tidak relevan. Jumlah peraturan dari sistem dapat mempengaruhi kinerjanya, jadi tidak ada pemasukan peraturan yang tidak dibutuhkan.
3. Sebuah kondisi yang menentukan bahwa sebuah solusi telah ditemukan atau tidak ditemukan. Ini penting untuk menghentikan beberapa RBS yang bekerja dalam *infinite loop*.

Interpreter pada kumpulan *production rules* dapat dijelaskan dalam siklus *recognize-act*, yang terdiri dari langkah-langkah sebagai berikut:

- i) Cocokkan pola kondisi aturan dengan elemen-elemen di *working memory*.
- ii) Jika ada lebih dari satu aturan yang cocok, maka pilih satu untuk digunakan, langkah ini disebut *conflict resolution*.
- iii) Gunakan aturannya, mungkin tambahkan sebuah *item* baru ke *working memory* atau hapus *item* yang lama dan lanjut ke langkah i.

Normalnya *conflict resolution* akan memilih sebuah *single rule* untuk dipakai dari kumpulan konflik. Strategi *conflict resolution* biasanya adalah kombinasi dari beberapa mekanisme dasar yang tiap-tiapnya mempunyai properti yang berbeda. Kinerja yang baik dari sebuah ES tergantung oleh properti kunci tertentu dari kuasa kontrol, seperti sensitivitas dan stabilitas. Sensitivitas artinya merespon cepat pada perubahan di lingkungan yang direfleksikan pada *working memory*, sedangkan stabilitas berarti menunjukkan suatu jenis kesinambungan dalam garis *reasoning*.

Tanpa *conflict resolution*, sebuah *production system* tidak mempunyai metode sederhana untuk menyelesaikan hal-hal yang berhubungan dengan nondeterminisme, mengelola *exceptions* atau memfokuskan perhatian pada garis *reasoning* tertentu. Dengan kata lain, representasinya akan kekurangan *heuristic power* sehingga programnya akan susah untuk dikontrol meskipun pengetahuan yang direpresentasikannya benar adanya.

Untuk mengelola *rules* itu sendiri, terdapat dua pendekatan yaitu :

- a. **Forward Chaining**, di mana *rules* diproses berdasarkan sejumlah fakta yang ada, dan didapatkan konklusi sesuai dengan fakta-fakta tersebut. Pendekatan *forward chaining* disebut juga *data driven*.
- b. **Backward Chaining**, di mana diberikan target/*goal*, kemudian *rules* yang aksinya mengandung *goal* di-trigger. *Backward chaining* ini cocok untuk menelusuri fakta yang masih belum lengkap, disebut juga *goal driven*.

Contoh penggunaan RBS ini adalah tentang apa aksi yang harus dilakukan untuk pergi ke kampus (lihat tabel II-1) :

Tabel II-1 Tabel Contoh *Rule Based System*

R	IF	THEN
1	Orang2 = "male"	Seharusnya "beri benda maskulin"
2	Orang2 = "female"	Seharusnya "beri benda feminin"
3	Seharusnya "beri benda maskulin", artis = "ya"	Aksinya "beri jam tangan bermerk"
4	Seharusnya "beri benda maskulin", artis = "tidak"	Aksinya "beri kemeja"
5	Seharusnya "beri benda feminin", artis = "ya"	Aksinya "beri kalung berlian"
6	Seharusnya "beri benda feminin", artis = "tidak"	Aksinya "beri bunga"

Lalu ketika user memberikan input berupa : Orang2 = male dan Orang2 adalah artis, maka keluarannya berupa "beri jam tangan bermerk".

II.1.4 Problem Solving

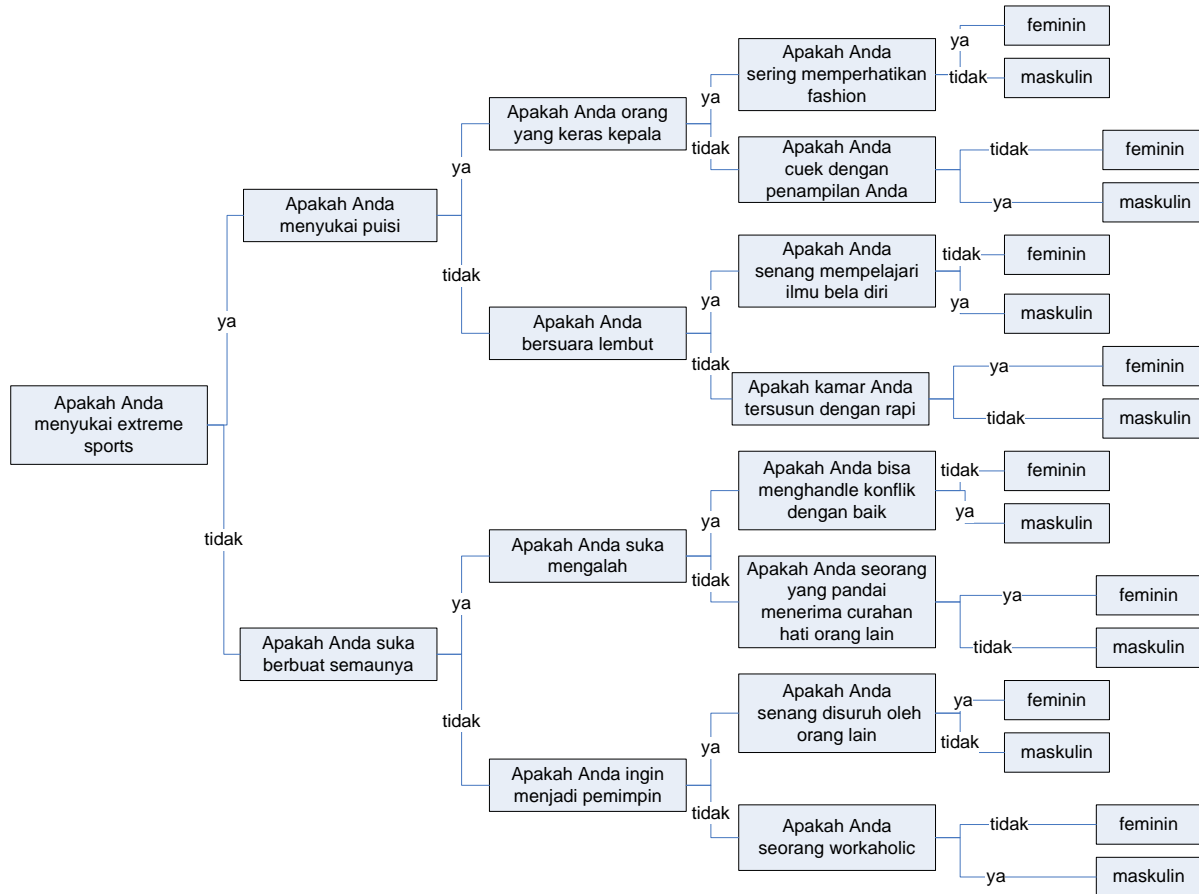
Sistem pakar memiliki kemampuan untuk memberikan saran kepada pengguna agar pengguna dapat membuat solusi yang lebih optimal. Dalam tugas akhir ini akan dibahas tentang kemampuan ES dalam hal *problem solving* yaitu *classification*, *constructive*, dan *simulation*.

II.1.4.1 Classification

Klasifikasi adalah masalah yang biasa ditemukan di berbagai bidang. Contohnya dalam bidang flora dan fauna, para pakar tertarik untuk menempatkan spesies baru dalam sebuah taksonomi jenis tumbuhan dan hewan. Istilah kelas biasanya terdapat dalam struktur organisasi di mana subkelas memiliki fungsi yang berbeda dengan superkelasnya, dan kelas yang tergolong *siblings* dalam hirarkinya biasanya memiliki beberapa fungsi yang sama.

Karakteristik klasifikasi yang penting adalah pakar memilih sebuah kategori dari kumpulan solusi yang mungkin dan telah dienumerasi. Pada kasus sederhana, fungsi yang menonjol dari sebuah objek bisa diklasifikasikan dengan mudah, jadi penjodohan antara data dan kategori bisa dilakukan dengan cepat. Pada kasus yang lebih rumit, fungsi yang menonjol tidak cukup untuk mengidentifikasi cabang dan *level* yang benar dari suatu hirarki. Untuk kasus ini, digunakan *heuristic classification* sebuah asosiasi nonhirarkis antara data dan kategori yang membutuhkan *intermediate inference*, mungkin melibatkan konsep pada taksonomi yang lain.

Pada *classification*, pengguna memilih salah satu solusi dari beberapa kumpulan solusi yang sudah ditetapkan oleh sistem berdasarkan fakta yang diberikan. Salah satu contoh sederhana *classification* adalah sebagai berikut :



Gambar II-5 Contoh *Classification*

Dalam sistem tersebut, ditanyakan kepada *user* pertanyaan di bawah ini.

Untuk menentukan apakah sifat Anda termasuk feminin atau maskulin, jawab pertanyaan di bawah ini (Ya-Tidak):

1. Apakah Anda menyukai *extreme sports* seperti *skateboard* dan *bungee jumping*? Ya
2. Apakah Anda menyukai puisi ? Tidak
3. Apakah Anda bersuara lembut ? Tidak
4. Apakah kamar Anda tersusun dengan rapi ? Tidak

Maka sistem tersebut mengeluarkan *output* : Anda termasuk orang yang berkepribadian maskulin.

Pada contoh di atas (gambar II-5) hanya terdapat dua solusi, berkepribadian maskulin atau feminin. Dari jawaban pertanyaan-pertanyaan tersebut didapat bahwa orang tersebut termasuk yang bersifat maskulin sesuai dengan *tree* yang telah dibuat sebelumnya.

II.1.4.2 Constructive

Dalam *problem solving* di ES ada strategi *classification problem solving* dan *constructive problem solving*. Perbedaan mendasar dari keduanya adalah pada *classification problem solving*, solusi dapat dipilih dari beberapa kumpulan solusi yang sudah ditetapkan, sedangkan pada *constructive problem solving*, solusi dibuat baru sesuai dengan syarat dan *constraints* yang ada.

Beberapa *task* yang membutuhkan *constructive problem solving* adalah perencanaan, desain, konfigurasi dan diagnosa. Konsep *task* tersebut adalah sebagai berikut :

- Dalam perencanaan, elemen solusinya adalah aksi, dan solusinya adalah urutan aksi yang menuju goal. *Constraint*-nya dibuat sesuai dengan hukum ruang dan waktu, contohnya, seseorang tidak dapat berada di dua tempat secara bersamaan.
- Dalam desain, elemen solusinya adalah komponen, dan solusinya adalah kombinasi komponen yang membentuk sebuah objek kompleks yang memenuhi *constraint* fisik tertentu pada materi.
- Pada diagnosa penyakit, elemen solusinya adalah penyakit, dan solusinya adalah kumpulan penyakit yang memenuhi gejala-gejala secara keseluruhan.

Bayangkan ketika seseorang ada di dalam sebuah ruangan dan harus menata perabotan ruangan tersebut. *Goal*-nya adalah untuk memikirkan penataan yang memenuhi *constraint* fisik yang dibuat oleh dimensi ruangan, dimensi perabotan, dan sekumpulan aturan umum tata ruang. Umumnya strategi yang digunakan untuk masalah ini adalah sebagai berikut :

- (1) Jika mungkin, mulai dengan sebuah penataan sebagian yang memenuhi *constraints*. Jika tidak, mulai dari awal.

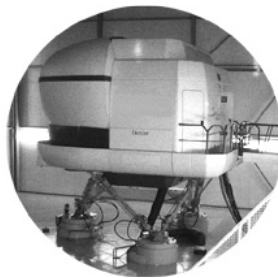
- (2) Jika penataan telah selesai, maka berhenti. Jika tidak lakukan sebuah penataan yang lebih baik dari penataan yang sekarang.
- (3) Jika penataan yang baru melanggar *constraint*, maka rencanakan perbaikan yang membuat sebuah penataan baru dengan meng-*undo* langkah sebelumnya sesedikit mungkin.
- (4) Lanjut ke langkah (2).

Strategi ini menjelaskan bahwa pemilihan aksi berikutnya adalah aksi yang punya *constraint* dan *timing* untuk aksi berikutnya yang lebih sedikit. Strategi ini dinamakan *least commitment strategy*.

Dalam *constructive problem solving* terdapat ketidaklengkapan pengetahuan dan terdapat banyak solusi yang dekat dengan solusi optimal. Hal ini menyebabkan *problem solver* harus mengikuti *least commitment strategy*. Program harus mampu mendeteksi konflik antar-*constraint* dan bisa *backtrack* dan mulai lagi atau merencanakan sebuah perbaikan terhadap konfliknya. Untuk penjelasan lebih lanjut, dapat dibaca di buku *Introduction to Expert Systems* oleh Peter Jackson [PEJ99].

II.1.4.3 Simulation

Simulasi adalah teknik untuk meniru sifat dari suatu situasi atau sistem dengan membuat model atau situasi serupa dengan tujuan memperoleh informasi dengan lebih baik dari situasi atau sistem tersebut. Salah satu metode pemecahan masalah dalam sistem pakar adalah metode simulasi.



Gambar II-6 Contoh Penerapan Simulasi dalam Bidang Penerbangan

Untuk mereduksi biaya pengembangan dan melakukan evaluasi dalam lingkungan yang realistis, simulasi digunakan sebagai pendekatan yang paling efektif, terutama untuk sistem berbasis pengetahuan yang menggunakan perhitungan yang detail

terhadap berbagai macam situasi yang mempunyai resiko tinggi, contohnya *domain navigasi pesawat terbang* (lihat gambar II-6 diambil dari [CYA99]). Ada dua pendekatan untuk melakukan simulasi, yaitu *fast-time simulation* dan *real time simulation* [CYA99].

Fast-time simulation adalah pendekatan berbasis skenario untuk menguji sistem berbasis pengetahuan. Pendekatan ini digunakan untuk menguji kemampuan dasar pemecahan masalah dari sistem berbasis pengetahuan dan untuk menyempurnakan basis pengetahuan. *Real-time simulation* menggunakan lingkungan realistik yang dapat memberikan manusia perasaan yang nyata dan pakar untuk mengevaluasi kemampuan pemecahan masalah dari sistem pakar secara langsung.

Selain di bidang penerbangan, simulasi mulai digunakan juga dalam bidang kesehatan, keuangan, olahraga, dll. Penggunaan simulasi membantu user mendapatkan proyeksi hasil yang diinginkan dan menurunkan tingkat kegagalan dalam bereksperimen.

II.2 Prediksi Kebangkrutan Perusahaan

Pada subbab ini dijelaskan mengenai kebangkrutan dan model prediksi kebangkrutan perusahaan.

II.2.1 Kebangkrutan

Sebelum melihat lebih lanjut bagaimana prediksi kebangkrutan perusahaan sangat diperlukan, perlu dipahami terlebih dahulu definisi bangkrut dan definisi kebangkrutan itu sendiri.

II.2.1.1 Definisi Bangkrut

Pengertian bangkrut yang dipakai di berbagai negara ada berbagai macam. Bangkrut yang dalam hal ini diidentikkan dengan kesulitan keuangan dapat dikategorikan menjadi beberapa macam, yaitu kegagalan ekonomi (*economic failure*), kegagalan bisnis (*business failure*), insolvabilitas teknis (*technical insolvency*), insolvabilitas dalam kepailitan (*insolvency in bankruptcy*) dan kepailitan legal (*legal bankruptcy*). Meskipun demikian, kegagalan perusahaan mempunyai arti yang luas, dan tidak harus diartikan bahwa kegagalan tersebut harus selalu diikuti dengan solusi kepailitan [AVI00].

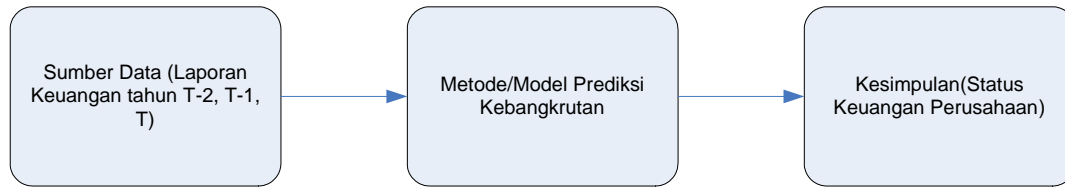
Kesulitan keuangan adalah suatu keadaan apabila arus kas operasi tidak cukup untuk memenuhi kewajiban-kewajiban lancarnya seperti hutang dagang atau biaya bunga [AVI00]. Pengertian tersebut dapat diperluas dengan mengaitkannya dengan *insolvency*. *Insolvency* sendiri didefinisikan dalam *Blacklaw's Dictionary* sebagai "ketidakmampuan debitur untuk membayar hutang, debitur kekurangan alat untuk membayar hutang sedemikian rupa sehingga aset tidak dapat memenuhi kewajiban.". Ketidakmampuan perusahaan untuk membayar kewajibannya (*insolvency*) adalah gejala awal yang menyebabkan kegagalan perusahaan [BRG94]. Kesimpulannya adalah bahwa kesulitan keuangan merupakan akibat dari semua faktor kegagalan pengelolaan perusahaan, maupun kegagalan karena faktor lain di luar perusahaan.

II.2.1.2 Definisi Prediksi Kebangkrutan

Prediksi kebangkrutan sangat banyak dibahas dalam studi tentang kegagalan bisnis perusahaan. Dalam perumusan dari model prediksi kebangkrutan, banyak jenis model telah dirancang. Perancangannya melibatkan sejumlah studi kasus perusahaan-perusahaan yang telah bangkrut atau dapat mempertahankan status keuangan perusahaannya dengan menghitung variabel akuntansi dan keuangan perusahaan-perusahaan tersebut kemudian mengklasifikasikannya menjadi perusahaan yang bangkrut atau non-bangkrut.

Ada empat jenis dari model prediksi kebangkrutan berdasarkan rasio laporan keuangan, *cash flow*, *stock returns*, dan *return standard deviations* [HWK04]. Dengan menggunakan rasio laporan keuangan, pendekatan yang berbeda dikembangkan untuk memprediksikan kebangkrutan. Prediksi kebangkrutan (informasi lebih lengkap tersedia di [BAN07]) menyatakan bahwa di akhir 1960-an, adanya usaha untuk mengembangkan model prediksi kebangkrutan dimulai. Ada tiga jenis model prediksi kebangkrutan yang berbeda yaitu:

1. model statistik (*multivariate discriminate analysis* [MDA], model analisis *logit*, dan analisis *probit*),
2. model matematik/statistik *gambler's ruin*,
3. model jaringan saraf buatan (*artificial neural network model*).



Gambar II-7 Komponen Prediksi Kebangkrutan Perusahaan

Jadi, dalam menentukan status keuangan perusahaan, setiap akuntan menggunakan langkah-langkah sebagai seperti yang ditunjukkan gambar II-7 di bawah ini yang akan dilanjutkan di subbab selanjutnya.

II.2.2 Komponen Prediksi Kebangkrutan

Dalam menentukan status keuangan perusahaan, diperlukan komponen-komponen prediksi kebangkrutan perusahaan (lihat gambar II-7) yang akan dijelaskan lanjut dalam subbab ini.

II.2.2.1 Sumber Data

Sumber data yang dipakai dalam model-model prediksi kebangkrutan perusahaan adalah berupa laporan keuangan (*financial statement*) dan rasio keuangan (*financial ratios*).

II.2.2.1.1 Laporan Keuangan

Sebelum menjelaskan tentang laporan keuangan, ada baiknya diketahui apa arti akuntansi sebenarnya. Akuntansi dapat didefinisikan sebagai seni mencatat, menyajikan dan menafsirkan secara sistematis transaksi-transaksi keuangan sebuah perusahaan [SDY91].

Laporan keuangan yang pada dasarnya merupakan ikhtisar dari data keuangan perusahaan yang pencatatannya dilakukan melalui fungsi kedua dari kegiatan akuntansi, dapat berupa:

- A. Laporan Keuangan Umum atau *general purpose financial statements*, pada umumnya terdiri dari :
 - Neraca Keuangan (gambar II-8 dari [SDY91])
 - Laporan Perhitungan Rugi-Laba

- Laporan Arus Kas
- Catatan atas Laporan Keuangan

B. Laporan Keuangan Khusus atau *special purpose financial statements*, terdiri dari :

- *Increase-or-decrease statement*
- *Trend index statement*
- *Common-size statement*
- *Statement of sources and uses of fund*
- *Statement of changes in working capital*
- *Schedule of cost of goods sold*
- *Cost of production report*

Tentunya tidak semua jenis dari laporan keuangan ini yang akan dipakai dalam menentukan parameter-parameter yang akan dipakai sistem nantinya. Jenis laporan keuangan yang dipakai akan dijelaskan kemudian sesuai dengan metoda prediksi kebangkrutan perusahaan yang bersangkutan.

AKTIVA		KEWAJIBAN	
Aktiva lancar		Kewajiban lancar	
Kas	\$2 0 6 5 0 0 0	Utang usaha	\$ 9 0 0 0 0 0
Piutang Usaha	2 7 2 0 0 0 0	Utang upah	2 5 0 0 0 0
Penyediaan	7 6 0 0 0 0 0	Sewa diterima di muka	2 4 0 0 0 0
Asuransi dibayar di muka	2 3 0 0 0 0 0	Total kewajiban	\$1 3 9 0 0 0 0
Total	\$7 8 4 5 0 0 0		
Properti, pabrik & perl		Ekuitas pemilik	
Tanah	\$10 0 0 0 0 0 0	Modal Pat King	18 2 0 5 0 0 0
Peralatan kantor \$1.800		Total kewajiban dan ekuitas pemilik	<u>19 5 9 5 0 0 0</u>
Dikurangi akumul. peny. 50	1 7 5 0 0 0 0		
Total properti, pabrik, dan peralatan	<u>11 7 5 0 0 0 0</u>		
Total aktiva	<u>\$19 5 8 5 0 0 0</u>		

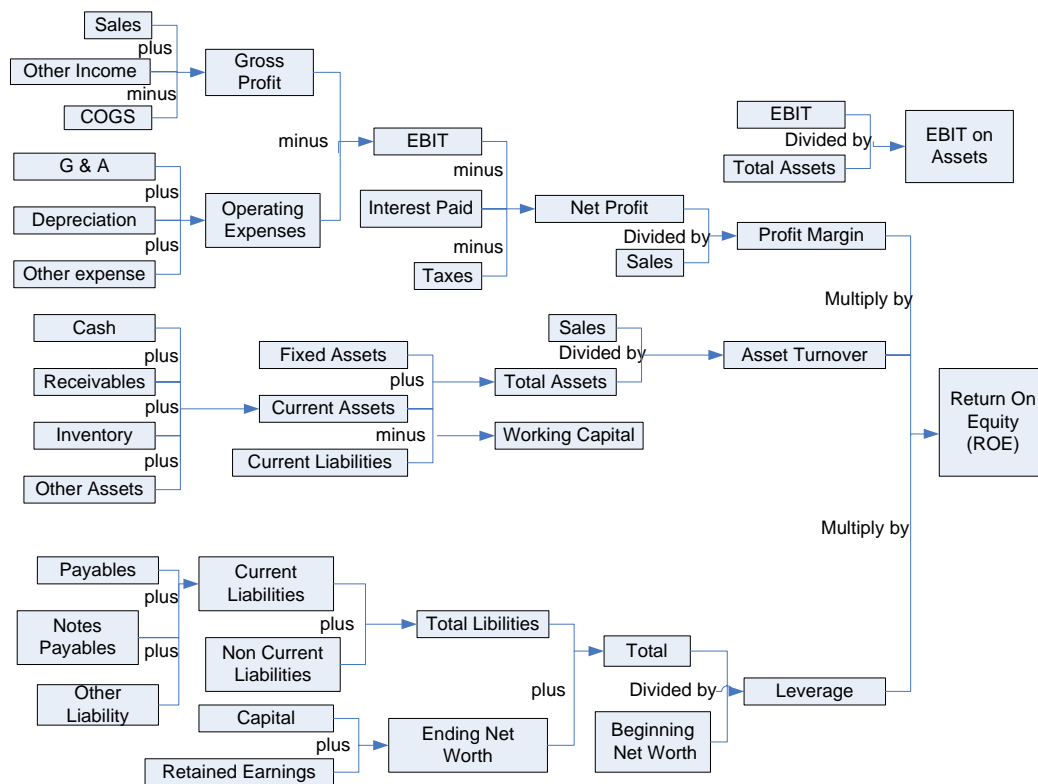
Gambar II-8 Contoh Neraca Keuangan

II.2.2.1.2 Analisis Rasio Keuangan

Data sumber akuntansi utama dapat dimanfaatkan untuk membentuk rasio-rasio keuangan. Di antaranya adalah laporan keuangan umum yang terdiri dari neraca, laporan rugi-laba, dan laporan perubahan modal kerja. Rasio-rasio keuangan yang semua unsur data akuntansinya hanya bersumber dari sebuah neraca saja disebut rasio keuangan neraca atau *balance sheet ratios*, yang semua unsur data akuntansinya bersumber hanya dari laporan rugi-laba saja disebut rasio keuangan rugi-laba atau

income statement ratios dan yang unsur data akuntansinya bersumber sebagian dari neraca atau laporan perubahan modal disebut rasio keuangan antarlaporan atau *interstatement ratios* [SDY91].

Seperti yang telah dijelaskan sebelumnya, laporan keuangan berguna bagi bankir, kreditor, pemilik, dan pihak-pihak yang berkepentingan lainnya dalam menganalisis serta menginterpretasikan kinerja keuangan dan kondisi perusahaan. Terdapat berbagai macam alat yang biasa digunakan untuk menganalisis serta menginterpretasikan kinerja keuangan dan kondisi perusahaan. Salah satunya terutama berguna dalam menganalisis kemampuan perusahaan untuk membayar utang-utangnya.



Gambar II-9 Penggabungan Neraca dan Laporan Laba Rugi Analisa Sistem Du Pont

Hubungan antara kewajiban dengan ekuitas pemilik dinyatakan dalam rasio sebagai berikut:

Rasio kewajiban terhadap ekuitas pemilik = total kewajiban / total ekuitas pemilik

Rasio satu dan rasio yang lain memiliki keterhubungan yang dibahas pada analisis sistem Du Pont. Analisis Sistem Du Pont yang telah di-*breakdown* lebih detil dibuat dengan menggabungkan laporan laba-rugi dan neraca untuk membedah secara terstruktur laporan keuangan dan menilai kondisi umum keuangan perusahaan (lihat gambar II-9). Dari gambar tersebut dapat disimpulkan jika salah satu rasio diubah nilainya maka nilai rasio yang lain dapat berubah. Ketergantungan antarrasio ini dapat digunakan untuk melihat pengaruh perubahan satu variabel keuangan terhadap yang lain.

Sebagai contoh, rasio kewajiban terhadap ekuitas pemilik Computer King pada akhir tahun 1999 adalah sebesar 0,025, dengan laporan sebagai berikut :

$$\text{Rasio kewajiban terhadap ekuitas pemilik} = \$400 / \$16.050 = 0,025$$

Menurut contoh di atas, rasio bernilai 1 menyatakan bahwa kewajiban dan ekuitas pemilik bernilai sama. Dengan kata lain, jika perusahaan mengalami kerugian sebesar jumlah kewajibannya, maka total aktiva perusahaan yang masih tersisa untuk kreditor akan persis sama dengan jumlah klaim atas aktiva tersebut. Apabila kejadian ini terjadi, kreditor masih dapat menerima pembayaran atas pinjaman yang diberikan kepada perusahaan dan pemilik tidak akan mendapatkan apa-apa. Sebaliknya, apabila rasio tersebut bernilai 3, maka kerugian yang jumlahnya lebih besar dari sepertiga kewajiban akan menyebabkan total aktiva perusahaan yang tersisa tidak mencukupi untuk menutup klaim kreditor [WAR99].

II.2.2.2 Model Prediksi Kebangkrutan Perusahaan

Model-model prediksi kebangkrutan perusahaan menggunakan pendekatan yang berbeda untuk memprediksi kepailitan. Beaver (1967) adalah peneliti model prediksi kebangkrutan yang paling awal. Dari variabel-variabel yang dianalisa Beaver, disimpulkan bahwa *cash flow*/total hutang adalah faktor yang paling penting untuk dipertimbangkan dalam memprediksi kebangkrutan. Pekerjaan Beaver disempurnakan dengan *multivariate analysis* yang dikembangkan oleh Altman dan peneliti yang lain.

II.2.2.2.1 Multiple Discriminant Analysis

Altman Z-score Model adalah model prediksi kebangkrutan perusahaan yang paling terkenal [BAN07]. Berdasarkan atas *multiple discriminate analysis*(MDA), model ini

memprediksi kesehatan keuangan perusahaan berdasarkan fungsi diskriminan dari rumus berikut (dari [ZSC07]):

(II-1)

$Z = 1.2X_1 + 1.4X_2 + 3.3X_3 + 0.6X_4 + 0.999X_5$
X_1 : <i>working capital/total assets</i>
X_2 : <i>retained earnings/total assets</i>
X_3 : <i>earnings before interest and taxes/total assets</i>
X_4 : <i>market value of equity / book value of total liabilities</i>
X_5 : <i>sales/total assets</i>
Jika $Z < 1.81$ maka perusahaan masuk dalam zona bangkrut
Jika $Z > 2.99$ maka perusahaan masuk dalam zona aman.

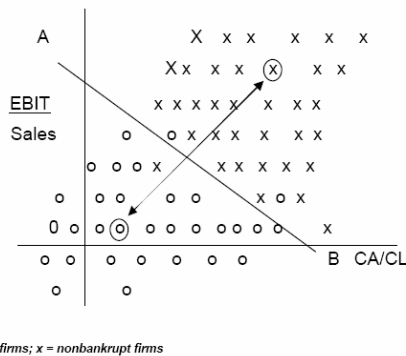
Model Z-Score yang dikembangkan di tahun 1968 dibuat berdasarkan sebuah contoh dari 66 perusahaan manufaktur dengan 33 pasangan perusahaan yang dikelompokkan. Grup bangkrut terdiri dari perusahaan yang telah memenuhi kondisi bangkrut sesuai *Chapter X* dari *United States bankruptcy* dari 1946 sampai 1965. Berdasarkan contohnya, seluruh perusahaan mempunyai Z-Score yang lebih tinggi dari 2.99 berada di dalam area non-bangkrut, sementara yang berada di bawah 1.81 adalah perusahaan yang bangkrut.

Altman mendiskusikan keuntungan utama dari MDA dalam hal mengelola masalah klasifikasi dengan potensinya untuk menganalisa seluruh profil variabel dari objek secara sekaligus daripada secara sekuensial memeriksa karakter individualnya. Dan dia menyatakan bahwa sesuai dengan berkembangnya *programming linear* dan *integer* dalam teknik tradisional di *capital budgeting*, pendekatan MDA ke analisa ratio secara tradisional mempunyai potensi untuk merumuskan masalah kembali dengan baik. Secara khusus kombinasi dari rasio dapat dianalisa secara bersama untuk menghilangkan kemungkinan keambiguan dan misklasifikasi yang diamati dalam pembelajaran rasio secara tradisional yang sebelumnya.

Berikutnya Altman mengembangkan model Z-Score yang telah direvisi (dengan koefisien yang telah direvisi dan jalan pintas Z-Score) yang membuang variabel X_4 dan X_5 di atas dan menggantikannya dengan sebuah variabel baru $X_4 = \text{net worth/total}$

liabilites. Variabel X_5 diduga dibuang untuk meminimisasi efek potensial industri yang berhubungan dengan *asset turnover*.

Model *Z-Score* adalah sebuah analisis linear yang membobotkan lima ukuran dan menambahkannya untuk menghasilkan nilai keseluruhan lalu menjadi dasar untuk mengklasifikasikan perusahaan dalam satu pengelompokan. Gambar II-10 (dari [HWK04]) menunjukkan sebuah analisis dua variabel di mana ukuran dari *profitability* dan *liquidity* ditempatkan untuk contoh perusahaan yang sehat dan tidak sehat. Model diskriminan memilih bobot yang tepat yang akan memisahkan nilai rata-rata dari tiap kelompok ketika dalam waktu yang sama meminimalkan jarak statistik dari tiap pengamatan(individu x dan o) dan *mean* dari kelompoknya. Tiap pengamatan diproyeksikan dalam garis AB yang membedakan kedua kelompok dengan baik.



Gambar II-10 *Linear Discriminant Analysis*

Sekarang model Altman telah diadaptasi ke dalam situasi perusahaan [ZSC07]. Formula Altman dibuat menjadi dua, yaitu formula untuk *Public Companies*(*Original Z-Score*) dan *Private Companies* (*Model A Z'-Score*). Untuk perusahaan *private*, digunakan rumus berikut ini (dari [ZSC07]) :

(II-2)

$Z = 0.717X_1 + 0.847X_2 + 3.107X_3 + 0.420X_4 + 0.998X_5$
X_1 : <i>working capital/total assets</i> X_2 : <i>retained earnings/total assets</i> X_3 : <i>earnings before interest and taxes/total assets</i> X_4 : <i>market value of equity / book value of total liabilities</i> X_5 : <i>sales/total assets</i>

Jika $Z < 1.23$ maka perusahaan masuk dalam zona bangkrut
 Jika $Z > 2.90$ maka perusahaan masuk dalam zona aman.

II.2.2.2.2 Model Lain

Ada beberapa model lain yang cukup terkenal seperti *Gambler's ruin mathematical/statistical model* dan model jaringan saraf tiruan. Akan tetapi dalam Tugas Akhir ini, model ini tidak akan digunakan, sehingga hanya dijabarkan sedikit mengenai model-model prediksi kebangkrutan perusahaan selain Altman *Z-Score*.

A. Model Springate

Model ini dikembangkan pada tahun 1978 oleh Gordon L.V. Springate. Dengan mengikuti prosedur yang dikembangkan oleh Altman, Springate menggunakan *step-wise multiple discriminate analysis* untuk memilih empat dari 19 rasio keuangan yang populer sehingga dapat membedakan perusahaan yang berada dalam zona bangkrut atau zona aman. Model Springate merumuskan sebagai berikut [HWK04]:

(II-3)

$Z = 1.03X_1 + 3.07X_2 + 0.66X_3 + 0.4X_4$
X_1 : <i>working capital/total assets</i> X_2 : <i>Net Profit before Interest and Taxes / total assets</i> X_3 : <i>Net Profit before Taxes / current liabilities</i> X_4 : <i>sales / total assets</i> Jika $Z < 0.862$ maka perusahaan masuk dalam zona bangkrut

Model ini mempunyai akurasi rata-rata 92.5% dengan menggunakan 40 perusahaan yang dicoba oleh Springate. Sementara ilmuwan yang lain yaitu Botheras(1979) mencoba menggunakan model Springate dengan 50 perusahaan yang memiliki aset rata-rata \$2,5 juta dan mendapatkan keakuratan 88% .

B. *Gambler's Ruin Mathematical/Statistical Model*

Gambler's ruin model mengasumsikan bahwa perusahaan telah diberikan sejumlah kapital, K , dan perubahan dalam K adalah Z , yang acak. Perubahan positif dalam K dihasilkan dari *cash flow* yang positif dari operasi. Dalam asumsi ini, perusahaan akan jadi bangkrut jika $K+Z < 0$.

Kapital K dapat diukur oleh pasar atau nilai akuntansi yang mengarah ke spesifikasi yang berbeda. Ada beberapa peneliti yang mengambil pendekatan ini, salah satunya yaitu Deakin (1977).

Contoh yang diambil Deakin terdiri dari 80 perusahaan yang dipilih acak dari *Moody's Industrial Manual* dan cocok dilihat dari tahun data, dan 63 perusahaan yang bangkrut, 32 perusahaan dari pembelajarannya di tahun 1972 dan 31 perusahaan yang bangkrut di tahun 1970 dan 1971 dari pembelajaran oleh Altman dan McGough pada tahun 1974. Kumpulan lima rasio yang diturunkan dari Libby dihitung untuk 143 perusahaan, dengan menggunakan data dua tahun sebelum kebangkrutan.

Deakin menganalisa 47 perusahaan yang bangkrut dari tahun 1972 sampai 1974, sebagai uji coba modelnya. Model lima variabel mengidentifikasi dengan tepat 39 perusahaan yang bangkrut, dua tahun sebelum bangkrut. Ada misklasifikasi dari satu perusahaan dan tujuh perusahaan perlu diinvestigasi lebih lanjut. Rumus model Deakin adalah sebagai berikut [HWK04] :

(II-4)

$I = -1.369 + 13.855X_1 + 0.060X_2 - 0.601X_3 + 0.396X_4 + 0.194X_5$
I: <i>Overall Index</i>
X_1 : <i>Net income/ total assets</i>
X_2 : <i>Current assets/ total assets</i>
X_3 : <i>Cash/ total assets</i>
X_4 : <i>Current assets / current liabilities</i>
X_5 : <i>Sales / current assets</i>

C. Model Jaringan Saraf Tiruan

Mulai di akhir 1980an, jaringan saraf tiruan menjadi metodologi penelitian yang dominan dalam intelegensia buatan. Peneliti menerapkan jaringan saraf dalam mengklasifikasikan masalah termasuk prediksi kebangkrutan. Banyak penelitian jaringan saraf dalam prediksi kebangkrutan mengetengahkan perbandingan dari kinerja (akurasi prediksi) jaringan saraf dengan metodologi lain seperti *discriminant analysis*, *logit analysis*, algoritma genetik, *decision*

tree, dan lain-lain. Sejumlah penelitian melaporkan bahwa kinerja jaringan saraf lebih baik sedikit daripada teknik yang lain, tapi pada umumnya hasilnya dibantah.

Jaringan saraf tiruan adalah komputer yang dibuat untuk mengelola informasi, secara paralel, sama seperti otak manusia. Jaringan saraf tiruan menyimpan informasi dalam bentuk pola dan bisa dipelajari dari pengalaman pengelolaannya. Tidak seperti MDA dan *logit analysis*, jaringan saraf tiruan membutuhkan syarat data yang kurang tegas, contohnya syarat untuk kelinearan. Jaringan saraf tiruan tidak mengemukakan bagaimana caranya membobotkan variabel. Maka dari itu, kesimpulan tidak dapat diturunkan dari beberapa variabel.