ALGORITMA TPDA DAN TPDA Π SEBAGAI ALTERNATIF STRUKTUR BAYESIAN NETWORK

Ivan Michael Siregar, Mewati Ayub, Hendry Handaka

Departemen Teknik Informatika, Institut Teknologi Harapan Bangsa
Jl. Dipatiukur 80-84,Bandung
ivan@ithb.ac.id
hhandaka@alumni.ithb.ac.id
Universitas Maranatha
mewati.ayub@eng.maranatha.edu

Absrak— Knowledge discovery in databases (KDD) merupakan proses pencarian pengetahuan bermanfaat dari data menggunakan teknik komputasi. Salah satu langkah khusus dalam KDD adalah data mining, yaitu aplikasi algoritma spesifik untuk mengekstrak pola/model dari data. Salah satu representasi model data mining adalah Bayesian Network (BN). BN digunakan untuk merepresentasikan pengetahuan tentang hubungan kebebasan/kebergantungan diantara variabel. BN terdiri dari struktur yang merepresentasikan pengetahuan secara kualitatif dan parameter yang merepresentasikan pengetahuan secara kuantitatif. Ada dua poendekatan untuk mengkonstruksi struktur BN dari data, yaitu metode analisis dependensi dan metode search & scoring. Tujuan utama tugas akhir ini adalah melakukan studi dan implementasi dari Algoritma TPDA dan TPDA-π untuk mengkonstruksi struktur BN dari data.

Kata kunci— data mining, Bayesian Network, analisis dependensi.

I. PENGANTAR

Bayesian Belief Network atau disingkat dengan *Bayesian Network* (BN) adalah suatu formalisasi graf untuk merepresentasikan suatu kondisi yang tidak pasti. Komponen BN itu sendiri meliputi *Direct Acyclic Graf* (DAG) dan *Conditional Probability Table* (CPT). Parameter-parameter pada CPT merupakan probabilitas kondisional dari nilai-nilai *node* diberikan setiap kombinasi nilai parent *node*nya, kecuali pada *node* yang tidak mempunyai *parent*, merupakan probabilitas awal dari nilai-nilai *node*. Pengkonstruksian BN dari data terdiri dari dua tahap, yaitu konstruksi struktur atau membangun DAG dan estimasi parameter atau menentukan nilai-nilai probabilitas pada CPT.

Ada dua pendekatan umum untuk mengkonstruksi struktur BN dari data, yaitu metode *search & scoring* dan metode analisis dependensi. Pada pendekatan metode *search & scoring*, algoritma-algoritmanya memandang masalah konstruksi struktur sebagai pencarian sebuah struktur yang paling cocok dengan data.

Proses konstruksi dimulai dari sebuah graf tanpa *edge*, dan kemudian menggunakan metode pencarian untuk menambahkan sebuah *edge* pada graf. Setelah itu digunakan metode *scoring* untuk melihat apakah struktur yang baru lebih baik daripada struktur sebelumnya. Jika lebih baik, maka *edge* tetap ditambahkan dan berusaha menambahkan sebuah *edge* yang lain. Proses ini berlanjut sampai tidak ada struktur baru yang lebih baik daripada struktur sebelumnya. Pada pendekatan metode analisis dependensi, masalah konstruksi dipandang secara berbeda. Sebuah struktur merepresentasikan banyak kebergantungan, oleh karena itu algoritma-algoritma dengan pendekatan ini berusaha menemukan kebergantungan dari data, dan kemudian menggunakan kebergantungan ini untuk menyimpulkan struktur. Hubungan kebergantungan diukur menggunakan CI (*conditional independence*) *test*.

Meskipun algoritma tersebut pada umumnya memberi hasil yang baik, namun ada beberapa masalah yang sering dihadapi yang mengakibatkan performa algoritma yang menurun. Masalah tersebut diantaranya [CHE01[a]]:

1. Dibutuhkan *node ordering*. Banyak algoritma yang memiliki kompleksitas yang besar dan akan sangat berpengaruh pada performa proses data dalam volume yang besar.
2. Kompleksitas saat komputasi. Secara teori dan praktik, algoritma konstruksi BN bekerja relatif lambat. Hal ini terjadi karena kebanyakan algoritmanya bersifat *dependency-analysis based* yang membutuhkan jumlah pengujian *conditional independence* (CI *test*) yang eksponensial.

Three Phase Dependency Analysis (TPDA) dan TPDA-π (TPDA dengan *node ordering*) yang dikembangkan oleh Jie Cheng adalah algoritma yang bisa mengatasi kelemahan di atas, dan merupakan algoritma yang bekerja dengan baik (mampu menemukan model yang sempurna) pada *training data* dengan kuantitas yang cukup. Model yang dihasilkan oleh TPDA dan TPDA-π adalah sebuah DAG *(Direct Acyclic Graf)* yaitu graf berarah tanpa siklus yang merepresentasikan keberhubungan antara variabel. Yang membedakan TPDA dan TPDA-π adalah bahwa pada TPDA tidak diketahui *node ordering* sedangkan pada TPDA-π *node ordering* merupakan sebuah prasyarat mutlak.

II. LEARNING BAYESIAN NETWORK MENGGUNAKAN INFORMASI TEORI

A. Konsep Dasar

Pada saat mengkonstruksi struktur BN, perlu terlebih dahulu dipahami beberapa konsep dasar yang akan digunakan. Konsep dasar tersebut akan digunakan dalam menganalisis struktur BN, agar struktur yang dikonstruksi memberikan model yang benar. Konsep dasar tersebut mengacu pada [NEA04] dan [CHE98] yang mencakup beberapa definisi yang diberikan berikut.

Berikut akan dijelaskan tentang struktur graf BN, yang merupakan graf berarah tanpa siklus berarah atau disebut *directed acyclic graf* (DAG). Penjelasan mengacu pada [NEA04].

Definisi 1. Sebuah *directed* graf G adalah pasangan (V,E) bisa didefinisikan sebagai sebuah pasangan berurut yang terdiri dari himpunan terbatas V dari beberapa *node*, dan sebuah hubungan ketetanggaan yang tidak refleksif E pada V. Dengan hubungan ketetanggaan yang tidak refleksif tersebut, maka *node* $x \in E$ akan memenuhi $(x,y) \notin E$, artinya suatu *node* tidak bisa sebagai titik awal dan sekaligus juga sebagai titik akhir. Graf G dinotasikan sebagai (V,E). Untuk setiap $(x,y) \in E$ maka dikatakan bahwa ada sebuah ***arc*** (sisi berarah) dari *node* x ke *node* y. Pada graf tersebut, *arc* direpresentasikan dengan panah dari x ke y, dengan x sebagai titik awal dan y sebagai titik akhir. Dikatakan juga bahwa *node* x dan *node* y adalah ***adjacent*** (bertetangga). X disebut sebagai ***parent*** dari y, dan y disebut sebagai ***child*** dari x. Dengan menggunakan konsep *parent* dan *child* secara rekursif, maka dapat dibuat juga konsep ***ancestor*** dan ***descendent***. Sebuah *node* yang tidak memiliki *parent* disebut dengan ***root***.

Definisi 2. Pada *learning Bayesian Network*, sering perlu untuk menemukan sebuah *path* yang menghubungkan dua *node* tanpa mempertimbangkan arah dari sisi-sisi pada *path*. Untuk membedakannya dengan *path* berarah *(directed path)* yang menghubungan dua *node* dengan *arc-arc* satu arah, maka *path* tersebut dikatakan sebagai ***adjacency path*** atau ***chain***. Definisi ini bisa digunakan juga untuk graf berarah, graf tidak berarah, dan graf campuran. *Chain* dinyatakan dengan memperlihatkan garis tak berarah diantara *node-node* dalam *chain*. Sebagai contoh misalkan *chain* [G,A,B,C] dinotasikan dengan G—A—B—C. Jika memperhatikan arah dari edge, maka harus digunakan panah. Misalnya untuk memperlihatkan arah dari *chain* tersebut, dinyatakan dengan $G \leftarrow A \rightarrow B \rightarrow C$.

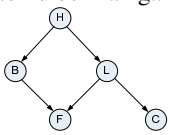
Definisi 3. Untuk setiap *node* pada *adjacency path*, jika dua *arc* pada suatu *path* dengan titik akhirnya bertemu pada sebuah *node* v, maka v disebut sebagai *node* yang ***collider*** karena kedua panah 'bertemu' bertemu pada *node* v. *Node* yang bukan merupakan ***collider*** disebut dengan ***non-collider***. Konsep *collider* dan *non-collider* hanya mengacu pada suatu *path* tertentu saja, sehingga sebuah *node* bisa saja sebagai *collider* pada suatu *path* dan sekaligus sebagai *non-collider* pada *path* lainnya.

Node ordering adalah domain pengetahuan yang digunakan oleh algoritma learning Bayesian network yang menspesifikasikan urutan *causal* atau *temporal* dari *node-node* yang terdapat pada graf, sehingga tidak ada *node* yang menjadi sebab atau terjadi lebih awal, daripada *node* yang kemunculannya lebih dahulu dari *node* tersebut pada suatu urutan.

B. Kondisi Markov

Kondisi Markov merupakan hubungan antara DAG dengan distribusi probabilitas [NEA04]. BN memanfaatkan kondisi Markov untuk merepresentasikan JPD secara efisien. Misalkan P adalah JPD dari variabel acak himpunan V, dan G=(V,E) adalah sebuah DAG. (G,P) disebut sebuah BN jika (G,P) memenuhi kondisi Markov [NEA04].

Misalkan diberikan JPD P dari variabel acak dalam himpunan V dan sebuah DAG G=(V,E). (G,P) dikatakan memenuhi kondisi Markov jika untuk setiap variabel $X \in V$, {X} bebas kondisional terhadap himpunan semua *nondescendant* X diberikan himpunan semua *parent* dari X. Jika himpunan *parent* dan himpunan *nondescendant* dari X dinotasikan dengan PA_X dan ND_X maka $I_P(\{X\},ND_X|PA_X)$ [NEA04]. Sebagai contoh diberikan gambar berikut:



Gambar 1. Sebuah DAG untuk mengilustrasikan kondisi Markov [NEA04]

Jika (G,P) memenuhi kondisi Markov untuk JPD P, maka akan dimiliki kebebasan kondisional seperti pada tabel berikut.

TABEL 1. KEBEBASAN KONDISIONAL DARI DAG PADA GAMBAR 3

PA	PA	Kebebasan Kondisional
C	*{L}*	$I_P(\{C\},\{H,B,F\}\|\{L\})$
B	*{H}*	$I_P(\{B\},\{L,C\}\|\{H\})$
F	*{B,L}*	$I_P(\{F\},\{H,C\}\|\{B,L\})$
L	*{H}*	$I_P(\{L\},\{B\}\|\{H\})$

C. Direction Dependent Separation (d- Separation)

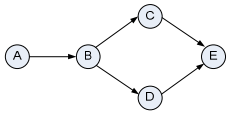
Semua kebebasan kondisional yang diperoleh dari kondisi Markov diidentifikasikan dari DAG dengan *d-separation*. Sebelum diberikan definisi *d-separation*, perlu diketahui definisi berikut:

Definisi 4. Misalkan G(V,E) adalah sebuah DAG, $A \subseteq V$, X dan Y adalah *node* berbeda dalam V-A, dan p adalah sebuah *chain* antara X dan Y. Maka p **diblok** oleh A jika dipenuhi salah satu kondisi berikut [NEA04]:

- Ada sebuah *node* $Z \in A$ pada *chain* p, dan Z adalah *head-to-tail node* pada *chain* p.
- Ada sebuah *node* $Z \in A$ pada *chain* p, dan Z adalah *tail-to-tail node* pada *chain* p.

- Ada sebuah *node* Z ∈ A pada *chain* p, yang dalam hal ini semua *descendant* dari Z bukan anggota A, dan Z adalah *head-to-head* (*collider/v-structure*) pada *chain* p.

Definisi 5. Untuk sebuah DAG G=(V,E), A ⊆ V, X dan Y adalah *node* berbeda dalam V-A. X dan Y dikatakan *d-separated* oleh A dalam G jika setiap *chain* antara X dan Y diblok oleh A. Sebagai contoh diberikan gambar berikut:



Gambar 2. DAG yang digunakan untuk mengilustrasikan d-separation

Dari DAG pada gambar tersebut dalam dikatakan bahwa:

- A dan E *d-separated* oleh {B}, karena *chain* A-B-C-E terblok pada B dan *chain* A-B-D-E terblok pada B
- B dan E *d-separated* oleh {C,D} karena *chain* B-C-E terblok pada C dan *chain* B-D-E terblok pada D

C dan D *d-separated* oleh {B} karena *chain* C-B-D terblok pada B dan *chain* C-E-D terblok tanpa *node* apapun

D. Kondisi Faithfulness dan Monotone Faithfulness

Dari kondisi Markov, hanya dapat diperoleh kebebasan tetapi tidak dapat diperoleh kebergantungan. Kondisi Markov hanya menunjukkan bahwa tidak adanya *edge* berarti ada kebebasan kondisional, namun adanya *edge* tidak berarti bahwa tidak ada kebebasan kondisional. Agar tidak adanya *edge* mempunya arti tidak ada kebebasan kondisional atau ada kebergantungan langsung, maka harus dipenuhi kondisi *faithfulness* [NEA04].

Misalkan dimiliki sebuah JPD dari variabel acak dalam himpunan V dan sebuah DAG G=(V,E). (G,P) dikatakan memenuhi kondisi *faithfulness* jika berdasarkan kondisi Markov, G mengandung semua dan hanya kebebasan kondisional pada P. Yaitu jika dipenuhi kedua kondisi berikut [NEA04]:

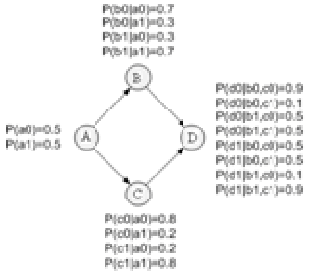
1. (G,P) memenuhi kondisi Markov (artinya G mengandung hanya kebebasan kondisional pada P).
2. Semua kebebasan kondisional pada P ada pada G, berdasarkan kondisi Markov.

Jika (G,P) memenuhi kondisi faithfulness, P dan G disebut saling *faithful*, dan G disebut *perfect map* dari P.

Teorema 1. Misalkan dimiliki sebuah JPD P dari variabel acak dalam himpunan V dan sebuah DAG G=(V,E). (G,P) memenuhi kondisi *faithfulness* jika dan hanya jika semua dan hanya kebebasan kondisional pada P yang teridentifikasi oleh *d-separation* pada G.

Teorema 2. Jika (G,P) memenuhi kondisi *faithfulness*, maka P memenuhi kondisi ini dengan semua dan hanya DAG yang ekuivalen Markov terhadap G. Selanjutnya, jika gp adalah DAG *pattern* yang sesuai dengan kelas ekuivalen Markov ini, *d-separation* dalam gp mengidentifikasi semua dan hanya kebebasan kondisional pada P. Gp dan P dikatakan saling faithful, dan gp adalah ***perfect map*** dari P.

Beberapa model DAG dapat berupa DAG *faithul* atau berupa *monotone* DAG *faithful*. Untuk menjelaskan hubungan antara DAG *faithful* dan *monotone* DAG *faithful*, berikut ini diberikan sebuah model probabilistik [CHE98].



Gambar 3. Distribusi probabilitas yang hanya memiliki IP({X},{Z}|{Y}) faithful terhadap DAG pattern

Pada graf tersebut, diberikan struktur BN yang lengkap, yang tersusun oleh struktur DAG dan empat tabel CP untuk empat *node*. Dengan menggunakan persamaan (2.2) maka dapat ditentukan *mutual information* dari pasangan *node* dari tabel CP. Dengan menggunakan nilai pada CP tabel dapat ditentukan *mutual information* antara *node* B dan *node* C, sehingga didapat I(B,C|A,D) = 0,018 dan I(B,C|D) = 0. Hal ini jelas sebuah kontradiksi dengan keadaan yang seharusnya, sebab dengan menggunakan {A,D} sebagai *condition-set*, akan berakibat pada terbentuknya sebuah *open path* B-D-C. Sedangkan jika menggunakan {D} sebagai *condition-set*, akan berakibat pada terbentuknya dua *open path* yaitu B-D-C dan B-A-C.

Jika model tersebut termasuk pada jenis *monotone DAG-faitful*, *I*(B,C|D) seharusnya bernilai lebih besar dari *I*(B,C|A,D). Sebagai catatan, model tersebut bukanlah sebuah DAG-*faithful* karena kebebasan antara B dan C jika diketahui {D} tidak bisa diekspesikan oleh struktur DAG. Namun jika dilakukan sedikit perubahan pada parameter CP tabel, sebagai contohnya, nilai CP tabel untuk *node* C diubah menjadi sama dengan nilai CP tabel untuk *node* B, maka akan dihasilkan nilai *I*((B,C|D) lebih besar dari 0 tetapi masih tetap bernilai lebih kecil dari *I*(B,C|A,D). Sekarang akan didapatkan model yang merupakan DAG-*faithful*, karena B dan C tidak bebas jika diketahui {D}, tetap bukan model yang *monontone* DAG-*faithful*.

III. PENDEKATAN KONSTRUKSI STRUKTUR BN DARI DATA

Ada dua cara memandang BN, masing-masing mengusulkan pendekatan untuk mengkonstruksi struktur BN dari data [CHE01[a]]. Pertama, BN adalah sebuah struktur yang merepresentasikan JPD dari variabel-variabel. Pandangan ini mengusulkan pendekatan konstruksi struktur dengan metode *search & scoring*. Kedua, struktur BN merepresentasikan sekumpulan hubungan kebebasan kondisional diantara *node-node*, menurut konsep *d-separation*. Pandangan ini mengusulkan pendekatan konstruksi dengan metode analisis dependensi. Kedua pendekatan umum untuk mengkonstruksi struktur BN tersebut :

1. Metode *search & scoring*

2. Metode analisis dependensi

IV. PENDEKATAN KONSTRUKSI STRUKTUR BN DARI DATA

Algoritma konstruksi struktur BN dengan pendekatan *dependency analysis* menggunakan CI *test* untuk menentukan apakah ada kebebasan kondisional diantara dua variabel acak diskrit (variabel dengan nilainya terbatas). Salah satu jenis CI *test* yang bisa digunakan untuk menghitung kebebasan kondisional pada variabel acak diskrit adalah *conditional mutual information test* [CHE01].

Untuk mengukur aliran informasi, digunakan teknik pengukuran dengan *mutual information* dan *conditional mutual information*. Pada *information theory*, *mutual information* antara *node* A dan B digunakan untuk merepresentasikan informasi harapan yang diperoleh tentang *node* B, setelah diketahui nilai variabel *node* A. Dalam BN, jika dua *node* saling bergantung, dengan telah diketahuinya nilai dari salah satu *node* maka akan memberikan informasi tentang nilai dari *node* yang lain. Informasi tersebut bisa diperoleh dengan menggunakan pengukuran *mutual information*. Oleh karena itu, *mutual information* diantara dua *node* dapat memberitahukan jika dua *node* adalah saling bergantung, dan juga memberitahukan bagaiman kedekatan hubungan *node-node* tersebut [CHE01[a]].

Mutual information antara dua *node* A dan B didefinisikan sebagai berikut [CHE98]:

$$I(X_i, X_j) = \sum_{x_i, x_j} P(x_i, x_j) \log \frac{P(x_i, x_j)}{P(x_i)P(x_j)} \quad \text{.....(2.1)}$$

dan *conditional mutual information* didefinisikan sebagai berikut [CHE98]:

$$I(X_i, X_j \mid c) = \sum_{x_i, x_j, c} P(x_i, x_j, c) \log \frac{P(x_i, x_j \mid c)}{P(x_i \mid c)P(x_j \mid c)} \quad \text{..(2.2)}$$

yang dalam hal ini *c* adalah himpunan *node*.

V. ANALISIS KONSTRUKSI STRUKTUR BAYESIAN NETWORK

Algoritma *Three Phase Dependency Analysis Phi* (TPDA-π) dan *Three Phase Dependency Analysis* (TPDA) merupakan dua algoritma yang digunakan dalam mengkonstruksi struktur BN dengan menggunakan pendekatan analisis dependensi. Pada TPDA-π urutan *node* (*node ordering*) diberikan, namun pada TPDA urutan *node* (*node ordering*) tidak diberikan.

A. *Diketahui Node Ordering*

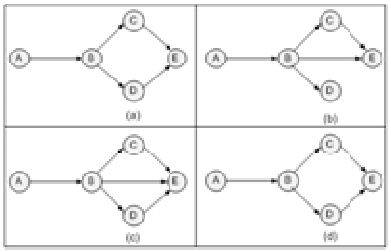
1) Tiga Tahap dari Algoritma TPDA-π

Tiga tahap dari algoritma ini adalah *drafting*, *thickening* dan *thinning*. Pada tahap pertama, algoritma ini menghitung *mutual information* dari setiap pasang *node* untuk mengukur kedekatannya dan membuat draft berdasarkan pada informasi ini. Pada tahap kedua, algoritma akan menambahkan *arc* ketika tiap pasang *node* tidak *conditional independence* pada suatu set kondisi tertentu. Hasil dari tahap kedua ini adalah suatu *I-map* yang mendasari model kebergantungan. Pada tahap ketiga setiap *arc* dari *I-map* diperiksa menggunakan CI (*conditional independence*) *test* dan setiap dua *node* yang *conditional independent* akan dihilangkan. Hasil dari tahap ketiga ini adalah suatu *map* sempurna yang mendasari model ketika model tersebut memenuhi kondisi DAG-*Faithful*.

Tahap I: (*Drafting*)

Langkah-langkah yang dilakukan pada tahap ini yaitu :

1. Inisiasi graf G(V,E) dengan kondisi V = {semua atribut dari dataset} dan E = { }. Inisiasi suatu list kosong L.
2. Untuk setiap pasang *node* (v_i,v_j) dimana $v_i,v_j \in V$ dan $i \neq j$, hitung *mutual information* $I(v_i,v_j)$ menggunakan persamaan (2.1). Untuk semua pasang dari *node* yang memiliki *mutual information* lebih besar dari nilai tetapan ε, urutkan berdasarkan nilai *mutual information*-nya dan simpan pasangan *node* tersebut kedalam list L dari besar ke kecil. Buat pointer *p* yang menunjuk pada pasangan *node* pertama dalam L.
3. Ambil dua pasang *node* pertama dari list L dan hapus dari list tersebut. Tambahkan *arc* yang sesuai kedalam E. Pindahkan pointer *p* ke pasangan *node* selanjutnya. (Pada algoritma ini, arah dari *arc* ditetapkan oleh *node ordering* yang diberikan).
4. Ambil pasangan *node* dari L yang ditunjuk oleh pointer *p*. Jika tidak ada *open path* antara dua *node*, tambahkan *arc* tersebut kedalam E dan hapus pasangan *node* tersebut dari L.
5. Pindahkan pointer *p* ke pasangan *node* selanjutnya dan kembali lakukan langkah 4 kecuali p menunjuk pada akhir list L. Sebagai ilustrasi mekanisme algoritma tersebut, berikut ini sebuah contoh sederhana sebuah jaringan *multi-connected*. Andaikan diberikan sebuah basis data yang direpresentasi dengan BN seperti pada gambar berikut, maka akan ditemukan struktur BN dari data tersebut.



Gambar 4. Multi connected network sederhana dan hasil fase I, II, III dari Algoritma TPDA-π

Setelah melalui langkah 2, akan dihasilkan *mutual information* dari 10 pasang *node*. Asumsikan bahwa setelah diurutkan maka seluruh pasang *node* tersebut akan memiliki urutan sebagai berikut $I(B,D) \geq I(C,E) \geq I(B,E) \geq I(A,B) \geq I(B,C) \geq I(C,D) \geq I(D,E) \geq I(A,D) \geq I(A,E) \geq I(A,C)$, dan seluruh *mutual information* yang dimiliki masing-masing pasangan *node* bernilai lebih besar dari ε. Kemudian seluruh pasangan *node* tersebut akan disimpan ke list L sehingga L berisikan *{(B,D),(C,E),(B,E),(A,B),(B,C),(C,D),(D,E),(A,D),(A,E),(A,C)}*. Pada kondisi nyata banyaknya pasangan *node* pada list L akan lebih sedikit dari jumlah

seluruh kemungkinan pasangan *node* yang bisa dibentuk tanpa memperdulikan nilai *mutual information*. Hal ini dapat terjadi karena ada beberapa pasang *node* yang kemungkinan memiliki *mutual information* lebih kecil dari dari ε [CHE98]. Pada langkah 3, 4 dan 5 algoritma akan memperoleh pasangan *node* secara iteratif pada urutan yang terdapat pada list L, dan selanjutnya menghubungkan kedua *node* tersebut dengan sebuah *arc* dan membuang pasangan *node* pada list L dengan syarat tidak ada *open path* diantara kedua pasangan *node* tersebut. Karena setiap pasang *node* yang terhubung oleh *arc* telah dibuang dari list L, maka pada bagian akhir dari fase ini, list L akan berisikan pasangan *node* berikut *{(C,D),(D,E),(A,D),(A,E),(A,C)*], yaitu yang merupakan pasangan *node* yang tidak terhubung langsung pada fase I namun memiliki *mutual information* lebih besar dari ε. Graf yang dihasilkan oleh fase I ditunjukkan pada gambar (b). Pada graf terebut terlihat bahwa graf yang dihasilkan hampir seperti graf yang sesungguhnya. Pada graf tersebut *arc (B,E)* adalah *arc* yang salah ditambahkan *(wrongly added arc)* dan *arc (D,E)* adalah *arc* yang hilang *(missing arc)*.

Pada langkah pertama perlu dilakukan pengurutan *mutual information* dari masing-masing pasangan *node* dari yang terbesar sampai yang terkecil pada list L. Hal ini dilakukan karena nilai *mutual information* yang lebih besar pada dasarnya merepresentasikan hubungan langsung (pada graf direpresentasikan dengan adanya *arc*) dibandingkan dengan nilai *mutual information* yang lebih kecil, yang merepresentasikan hubungan tidak langsung. Pada dasarnya, prinsip fase I dari Algoritma TPDA adalah juga prinsip yang digunakan pada algoritma Chow-Liu, dan pada Algoritma Chow-Liu telah dijamin bahwa graf yang dihasilkan akan sama dengan graf aslinya, dan fase II serta fase III tidak akan mengubah apapun. Itulah sebabnya, Algoritma Chow-Liu bisa dipandang sebagai kasus dari Algoritma TPDA untuk konstruksi BN [CHE98].

Draft model yang dihasilkan pada fase I menjadi basis untuk melakukan fase II.

Tahap II: (*Thickening*)

Langkah-langkah pada tahap II :

6. Pindahkan pointer *p* ke pasangan *node* pertama dalam L.
7. Ambil pasangan *node* (*node* 1, *node* 2) dari L pada posisi pointer *p*. Panggil prosedur *find_cut_set* (*current graf, node1,node2*) untuk menemukan *cut-set* yang dapat menghasilkan d-separate *node1* dan *node2* pada graf terkini. Gunakan CI test untuk melihat jika *node1* dan *node2* conditional independent dikarenakan *cut-set*. Jika benar, lanjutkan ke langkah selanjutnya, sebaliknya jika tidak, hubungkan pasangan *node* tersebut dengan menambahkan sebuah *arc* kedalam E.
8. Pindahkan pointer *p* ke pasangan *node* selanjutnya dan kembali ke langkah 7 kecuali *p* menunjuk ke akhir list L.

Pada fase I, masih ada beberapa pasang *node* pada L hanya karena sudah terdapat *open path* diantara pasangan *node* tersebut. Pada fase II, digunakan CI *test* dan *d-separation* untuk memeriksa apakah pasangan *node* tersebut seharusnya terhubung atau tidak. Karena hanya ada satu CI test yang digunakan untuk memeriksa keterhubungan tersebut, maka tidak bisa dipastikan apakah keterhubungan tersebut adalah benar-benar perlu, tetapi bisa dipastikan bahwa tidak akan ada lagi *missing arc* pada akhir fase II tersebut.

Pada fase II algoritma memeriksa seluruh pasangan *node* yang tersisa pada L, yaitu pasangan *node* yang memiliki *mutual information* lebih besar dari ε dan tidak terhubung langsung. Dengan setiap pasangan *node* pada L, algoritma pertama sekali akan menggunakan prosedur *find_cut_set* untuk mendapatkan sebuah *cut-set* yang dapat membuat dua *node d-separated* lalu menggunakan CI *test* untuk memeriksa apakah dua *node* adalah bebas kondisional diberikan *cut-set*. Prosedur *find_cut_set* mencoba untuk menemukan minimum *cut-set* (*cut-set* dengan jumlah *node* paling sedikit) dengan menggunakan metode heuristik. Hal ini dilakukan karena *condition-set* dengan ukuran yang kecil dapat mengakibatkan CI test lebih handal dan efisien. Setelah dilakukan CI *test*, sebuah *arc* ditambahkan jika dua *node* tidak bebas kondisional. Pada fase ini juga ada kemungkinan terjadinya *wrongly added arc*, karena bebarapa *arc* yang seharusnya ada namun kemungkinan masih missing sampai berakhirnya fase ini, dan dengan adanya missing *arc* tersebut dapat mencegah ditemukannya *cut-set* yang tepat. Karena satu-satunya penyebab gagalnya menambahkan sebuah *arc* adalah karena *node* tersebut bebas, pada fase II tidak terjadi kehilangan *arc* dari model yang sesungguhnya, jika model tersebut adalah DAG-*faithful*.

Pada contoh, graf setelah fase II ditunjukkan pada gambar (c). *Arc* (D,E) ditambahkan sebab D dan E tidak bebas kondisional pada {B}, yang merupakan *cut-set* terkecil antara *node* D dan E pada graf terkini. *Arc* (A,C) tidak ditambahkan karena hasil CI *test* mengindikasikan bahwa A dan C adalah bebas diberikan *cut-set* {B}. *Arc* (A,D), (C,D), dan (A,E) tidak ditambahkan karena alasan yang sama. Pada gambar (c) terlihat bahwa graf yang dihasilkan setelah fase II berisikan semua *arc* dari model yang sesungguhnya, yaitu merupakan sebuah *I-map* dari model yang sesungguhnya.

Tahap III (*Thinning*)

Langkah selanjutnya yang akan dilakukan :

9. Untuk setiap *arc* (*node1, node2*) pada E, jika terdapat *path-path* lain disamping *arc* antara dua *node*, hapus sementara *arc* dari E dan panggil prosedur *find_cut_set* (*current graf, node1,node2*) untuk menemukan *cut-set* yang dapat menghasilkan *d-separate node1* dan *node2* pada graf. Gunakan CI *test* untuk melihat jika *node1* dan *node2 conditional independence* disebabkan oleh *cut-set*. Jika ya, hapus *arc* secara permanen, sebaliknya jika tidak, tambahkan kembali *arc* tersebut kedalam E.

Pada fase ini, algoritma pertama sekali mencoba menemukan *arc* yang menghubungkan satu pasangan *node* yang juga dihubungkan oleh *path* lainnya. Hal ini juga

berarti bahwa tidak mungkin ada kebebasan antara pasangan *node* tersebut karena adanya *arc* berarah diantara kedua *node* tersebut. Selanjutnya algoritma membuang *arc* tersebut dan menggunakan prosedur *find_cut_set* untuk menemukan sebuah *cut-set*. Setelah itu, CI test digunakan untuk memeriksa apakah kedua *node* tersebut bebas kondisional pada *cut-set*. Jika benar, *arc* tersebut akan dibuang secara permanen, dan sebaliknya jika tidak *arc* tersebut akan ditambahkan kembali karena ternyata *cut-set* tidak bisa memblok aliran informasi yang mengalir diantara kedua *node* tersebut. Prosedur ini terus dikerjakan hingga semua pasangan *node* selesai diperiksa.

B. *Tanpa Diketahui Node Ordering*

Sebagai perluasan dari Algoritma TPDA-π, algoritma ini mengambil tabel database sebagai input dan konstruksi struktur *Bayesian Network* sebagai output. Sejak tidak diberikannya input *node ordering*, Algoritma TPDA harus menghadapi dua kendala yaitu bagaimana untuk menentukan jika dua *node conditionally independent* dan bagaimana untuk mengorientasikan *edge* pada graf.

1} Tiga Tahap dari Algoritma TPDA

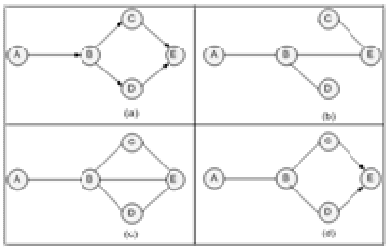
Tiga tahap dari algoritma ini sama dengan tahap pada Algoritma TPDA-π yaitu *drafting*, *thickening* dan *thinning*. Pada tahap pertama, algoritma akan menghitung *mutual information* dari setiap pasang *node* untuk mengukur kedekatannya dan membuat draft berdasarkan informasi tersebut. Draft berisi graf yang saling terhubung satu demi satu (graf tanpa loop). Pada tahap kedua, algoritma menambahkan *edge* pada graf ketika pasangan *node* tidak dapat dipisahkan menggunakan grup dari CI test. Hasil dari tahap 2 berisi semua *edge* dari model dasar kebergantungan. Pada tahap ketiga, setiap edge diperiksa menggunakan grup dari CI test dan akan dihapus jika dua *node* dari edge bersifat *conditionally independent*. Hasil dari tahap tiga mengandung edge yang tepat sama dengan model dasar model pada model *DAG-faithful monotone*. Pada akhir dari tahap ini, algoritma juga melakukan prosedur untuk menentukan arah *edge* pada graf, tapi prosedur ini tidak mampu untuk menentukan arah dari semua *edge*.

Tahap I (*Drafting*)

Langkah-langkah yang dilakukan pada tahap ini yaitu :

1. Inisiasi graf G(V,E) dengan kondisi V = {semua atribut dari dataset} dan E = { }. Inisiasi suatu list kosong L.
2. Untuk setiap pasang *node* (v_i,v_j) dimana $v_i,v_j \in V$ dan $i \neq j$, hitung *mutual information* $I(v_i,v_j)$ menggunakan persamaan (2.1). Untuk semua pasang dari *node* yang memiliki *mutual information* lebih besar dari nilai tetapan ε, urutkan berdasarkan nilai *mutual information*-nya dan simpan pasangan *node* tersebut kedalam list L dari besar ke kecil. Buat pointer *p* yang menunjuk pada pasangan *node* pertama dalam L.
3. Ambil dua pasang *node* pertama dari list L dan hapus dari list tersebut. Tambahkan edge yang sesuai kedalam E. Pindahkan pointer *p* ke pasangan *node* selanjutnya.
4. Ambil pasangan *node* dari L yang ditunjuk oleh pointer *p*. Jika tidak ada *adjacency* antara dua *node*, tambahkan edge tersebut kedalam E dan hapus pasangan *node* tersebut dari L.
5. Pindahkan pointer *p* ke pasangan *node* selanjutnya dan kembali lakukan langkah 4 kecuali p menunjuk pada akhir list L atau G bernilai N - 1 edge. (N adalah jumlah *node* dalam G. Ketika G bernilai N – 1 *edge*, tidak ada lagi *edge* yang dapat ditambahkan tanpa membentuk loop.)

Sebagai ilustrasi mekanisme algoritma, maka digunakan contoh yang sama seperti pada TPDA-π. Andaikan diberikan sebuah basis data yang direpresentasi dengan BN seperti pada gambar (a), maka akan ditemukan struktur BN dari data tersebut.



Gambar 5. Multi connected network sederhana dan hasil fase I,II, III dari algoritma TPDA

Setelah melalui langkah 2, akan dihasilkan *mutual information* dari 10 pasang *node*. Asumsikan seperti sebelumnya bahwa setelah diurutkan maka seluruh pasang *node* tersebut akan memiliki urutan sebagai berikut $I(B,D) \geq I(C,E) \geq I(B,E) \geq I(A,B) \geq I(B,C) \geq I(C,D) \geq I(D,E) \geq I(A,D) \geq I(A,E) \geq I(A,C)$, dan seluruh *mutual information* yang dimiliki masing-masing pasangan *node* bernilai lebih besar dari ε. Sesudah melewati tahap ke 5, akan didapatkan draft seperti yang ditunjukkan pada gambar (b). Sekarang list berisikan {(*B,D*),(*C,D*),(*D,E*),(*A,D*),(*A,E*),(*A,C*)}. Pada contoh ini, (*B,E*) adalah *arc* yang salah ditambahkan *(wrongly added arc*) dan (*D,E*) dan (*B,C*) adalah *arc* hilang *(missing arc)* karena adanya *adjacency path* (*D-B-E*) dan (B-E-C).

Seperti pada Algoritma TPDA-π, tujuan merancang fase ini adalah untuk menemukan model *draft* yang cukup dekat dan akurat dengan model yang sesungguhnya hanya dengan melakukan pengujian *mutual information* pada pasangan *node*.

Perbedaan algoritma ini yaitu tidak perlunya menyatakan setiap pasang *dependency open-path* karena tidak mungkin untuk menentukan apakah suatu path open atau tidak tanpa diketahui *direction* dari tiap *edge*. Tahap ini berhenti ketika setiap pasang dependensi dinyatakan oleh *adjacency path*. Hasilnya kebanyakan adalah satu *adjacency path* antara semua pasang *node*. Oleh karena itu draft ini dinamakan *polytree* atau grup dari *polytree*.

Tahap II (*Thickening*)

Langah-langkah pada tahap II :

6. Pindahkan pointer *p* ke pasangan *node* pertama dalam L.
7. Ambil pasangan *node* dari L pada posisi pointer *p*. Panggil prosedur *try_to_separate*(*current graf, node1,node2*) untuk melihat jika pasangan *node* dapat di-*separated* pada graf. Jika benar, lanjutkan ke langkah selanjutnya, sebaliknya jika tidak, hubungkan pasangan *node* tersebut dengan menambahkan *arc* kedalam E.
8. Pindahkan pointer *p* ke pasangan *node* selanjutnya dan kembali ke langkah 7 kecuali *p* menunjuk ke akhir list L.

Pada tahap I, masih ada beberapa pasang *node* pada L hanya karena sudah terdapat *open path* diantara pasangan *node* tersebut. Pada fase II, digunakan prosedur *try_to_separate* untuk melihat apakah pasangan *node* haru dihubungkan atau tidak. Seperti pada Algoritma TPDA-π, hubungan antar *node* tidak seluruhnya benar, tapi yang pasti tidak ada *edge* yang hilang sesudah tahap ini.

Pada fase II, algoritma memeriksa seluruh pasangan *node* yang memiliki *mutual information* lebih besar dari ε dan tidak terhubung langsung. Perbedaan dengan Algoritma TPDA-π terletak pada penggunaan *return* satu *cut set* dari prosedur *find_cut_set* untuk memeriksa jika dua *node conditionally independent*, algoritma ini menggunakan grup dari CI test pada prosedur *try_to_separate*. Yang perlu diketahui yaitu bahwa dua *node* dapat di-*separated* hanya jika berada pada keadaan *independent conditionally* pada *cut-set*. Beberapa pasang *node* mungkin saja tidak dapat di-*separated* karena berada dalam keadaan *conditionally independent*. Artinya ada beberapa *edge* yang salah dimasukkan pada tahap ini. Alasan yang perlu disampaikan :

1. Beberapa *edge* yang benar mungkin saja tetap hilang (*missing*) sampai pada akhir tahap ini dan *edge* yang hilang itu dapat menghalangi prosedure *try_to_separate* untuk menemukan *cut-set* yang tepat.
2. Karena prosedur *try_to_separate* menggunakan metode heuristik maka tidak akan dapat menemukan *cut-set* yang benar untuk suatu struktur grup yang spesial.

Pada contoh, graf setelah fase II ditunjukkan pada gambar (c). *Edge* (*B,C*) dan (*D,E*) ditambahkan karena prosedur *try_to_separate* tidak dapat melakukan *separate* pada pasangan *node* dengan menggunakan CI test. *Edge* (*A,C*) tidak ditambahkan karena CI test menghasilkan bahwa A dan C adalah *conditionally independent* diberikan oleh *cut-set* {B}. *Edge* (*A,D*), (*C,D*) dan (*A,E*) tidak dimasukkan juga karena alasan yang sama.

Tahap III (*Thinning*)

Langkah selanjutnya yang akan dilakukan :

9. Untuk setiap edge dalam E, jika terdapat path lain disamping *edge* antara dua *node*, hapus sementara *edge* dari E dan panggil prosedur *try_to_separated* (*current graf, node1,node2*). Jika dua *node* tidak dapat di-*separated*, tambahkan kembali *edge* ini kedalam E, bila sebaliknya hapus *edge* secara permanen.
10. Panggil prosedur orient_edges (*current graf*).

Seperti telah dijelaskan bahwa pada fase I dan fase II ada kemungkinan ditemukan *wrongly added arc*, maka pada fase III ini yang dilakukan adalah memastikan adanya *arc* yang *wrongly added* dan selanjutnya membuang *arc* tersebut. Pada tahap ini, algoritma akan mencari setiap *edge* yang menghubungkan pasangan *node* yang juga terhubung oleh path lain. Sejak dependency antara pasangan *node* tidak berhubungan langsung dengan *edge*, algoritma menghilangkan *edge* ini sementara dan mengguanakan prosedur *try_to_separate* untuk memeriksa jika *edge* tersebut dapat dipisahkan. Jika berhasil, maka *edge* akan dihapus permanen, sebaliknya *edge* dikembalikan. Proses ini berulang hingga semua *edge* diperiksa. Sesudah itu, prosedur *try_to_separate* digunakan untuk mencek kembali semua *edge*.

Graf yang dihasilkan pada akhir tahap III ditunjukkan pada gambar (d), yang merupakan struktur sesungguhnya dari graf. *Edge* (*B,E*) dibuang karena B dan E adalah bebas kondisional diberikan {*C,D*}. Pada akhir fase ini, telah berhasil dibuat suatu model *monotone DAG-faithful* yang strukturnya sama dengan struktur model BN yang benar.

Tahap ini juga dapat menentukan dua dari lima arah edge dengan benar, yaitu *edge* (*C,E*) dan *edge* (*D,E*). Hasil ini tidak terlalu sempurna. Bagaimanapun melihat pada keterbatasan identifikasi *collider* yang dilakukan, hasil yang diperoleh sangat baik. Tidak ada metode lain yang lebih baik untuk struktur ini. Biasanya hasil akan lebih baik jika stuktur *network* mengandung lebih banyak *edge* dan *collider*.

VI. Analisis Kompleksitas

Pada bagian ini akan dijelaskan mengenai kompleksitas waktu pada Algoritma TPDA-π dan Algoritma TPDA. Kompleksitas dilihat berdasarkan pada penggunaan *mutual information* dan *conditional mutual information* sebagai CI *test*.

A. *Analisis Kompleksitas Algoritma TPDA-π*

Misalkan suatu dataset memiliki *N* atribut, jumlah maksimum dari nilai yang mungkin dari setiap atribut adalah *r* dan setiap atribut memiliki paling banyak *k* parent. Kompleksitas yang muncul pada tiap tahap:

Tahap I: Sejak tahap I menghitung *mutual information* antara dua *node*, diperlukan $N(N-1)/2$ kali atau $O(N^2)$ penghitungan *mutual information*. Pada persamaan (2.1), setiap penghitungan *mutual information* memerlukan $O(r^2)$ kali dari operasi dasar seperti logaritma, perkalian dan pembagian. Mengurutkan pasangan *node* diselesaikan dalam $O(N^2)$ langkah dengan menggunakan algoritma *buble sort*. Kompleksitas waktu dari tahap I ini pada operasi dasar adalah $O(N^2r^2)$.

Tahap II: Tahap ini mencoba untuk menambahkan setiap *arc* ke dalam graf dan memerlukan CI test paling banyak $O(N^2)$ kali. Pada persamaan (2.2), setiap CI *test* memerlukan paling

banyak $O(r^{k+2})$ operasi dasar. Kompleksitas dari tahap ini pada operasi dasar adalah $O(N^2 r^{k+2})$.

Tahap III: Tahap ini memiliki kompleksitas algoritma $O(N^2)$.

B. *Analisis Kompleksitas Algoritma TPDA*

Tahap I: Tahap ini memiliki komplesitas yang sama seperti algoritma sebelumnya yaitu $O(N^2)$ penghitungan CI *test*.

Tahap II: Tahap ini mencoba untuk menambahkan edge pada graf yang didapat dari tahap I. Karena paling banyak sisa pasangan *node* yang harus dimasukkan adalah *N(N-1)/2–(N-1)*, maka tahap ini akan memanggil prosedur *try_to_separate* paling banyak *N(N-1)/2–(N-1)* kali atau $O(N^2)$. Eksekusi dari prosedur *try_to_separate* memiliki kompleksitas $O(N^2)$. Jadi, tahap II membutuhkan $O(N^4)$ CI *test*.

Tahap III: Tahap ini mencoba untuk menghilangkan setiap *edge* dari graf yang dihasilkan tahap II. Paling banyak *edge* yang terdapat pada graf adalah *N(N-1)/2* sehingga fase ini akan menggunakan prosedur *try_to_separate* sebanyak *N(N-1)/2* kali atau $O(N^2)$. Jadi, tahap III membutuhkan $O(N^4)$ CI *test* sama seperti pada tahap II.

VII. KESIMPULAN

Algoritma TPDA-π dan Algoritma TPDA sebagai algoritma konstruksi struktur BN dengan pendekatan analisis dependensi menghasilkan DAG *pattern*, dengan asumsi yang harus dipenuhi yaitu : atribut yang ada pada tabel harus memiliki nilai-nilai yang diskrit dan tidak ada missing value, seluruh record terjadi secara bebas diberikan model probabilistik dari data, dan volume data yang dimiliki harus cukup besar.

Kompleksitas dari Algoritma TPDA lebih besar dibandingkan dengan Algoritma TPDA-π. Hal ini terlihat dari jumlah CI test yang dilakukan pada Algoritma TPDA lebih banyak daripada Algoritma TPDA-π dengan nilai batas yang sama.

REFERENSI

[1] [NEA04] Neapolitan, Richard E. (2004). Learning Bayesian Networks, Prentice Hall

[2] [CHE98] Cheng, J., Bell, D. A., Liu, W. (1998) Learning Bayesian Network from Data: an Efficient Approach Based on Information Theory. Faculty of Informatics, University of Ulster, U.K.

[3] [CHE01[a]] Cheng, J., Bell, D. A., Liu, W. (2001) Learning Bayesian Network from Data: An Information-Theory Based Approach. Faculty of Informatics, University of Ulster, U.K.

[4] [CHE01[b]] Cheng, J. (2001) Belief Network PowerConstrutor (Evaluation Copy) Version 2.2 Beta.

[5] http://www.cs.ualberta.ca/~jcheng/bnpc.html