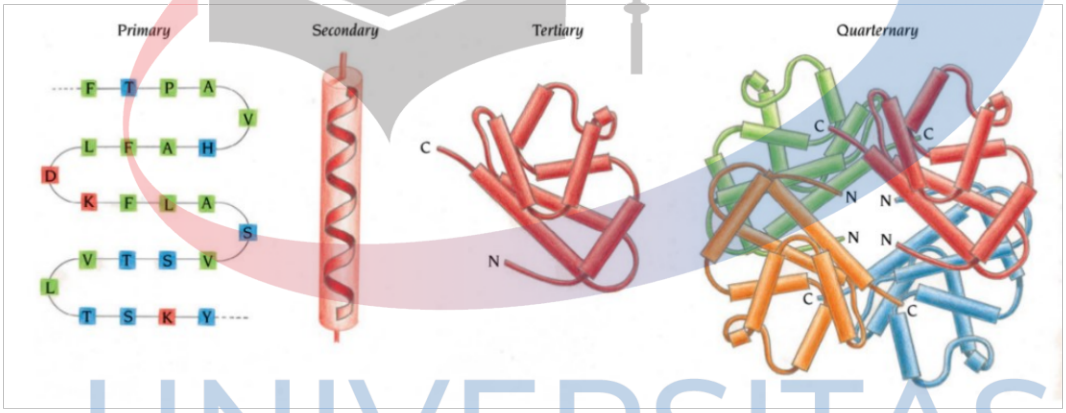
BAB II
TINJAUAN PUSTAKA

2.1. Protein Folding

Protein adalah zat yang sangat kompleks dan terdapat pada semua organ manusia (Jafari and Javidi, 2020). Protein memiliki fungsi yang sangat penting dalam organisme, seperti fungsi struktural pada otot dan tulang, fungsi katalitik untuk semua reaksi biokimia yang membentuk metabolisme dan fungsi untuk mengkoordinasikan gerakan dan transduksi sinyal. Protein terbentuk dari asam amino yang tersusun secara linier (Czibula et al., 2011) dan pada umumnya protein selalu melipat ke bentuk struktur tiga dimensi yang stabil atau dikenal dengan struktur *native*. Struktur *native* dari sebuah protein hanya dapat ditentukan melalui deret asam aminonya dan terbentuk melalui sebuah proses yang disebut pelipatan protein (Li et al., 2018b). Gambar 2.1 merupakan ilustrasi tahapan pelipatan sebuah protein.



Gambar 2. 1 Struktur Protein

Deret linier asam amino dari rantai poplipeptida protein dinamakan struktur *primary*. Hal yang membedakan struktur *secondary* dengan struktur sebelumnya adalah untaian alpha (α) dan beta (β). Struktur *tertiary* terbentuk dari penggabungan elemen struktural menjadi satu atau beberapa kesatuan yang disebut dengan *domains*. Strukur *quarternary* merupakan struktur protein yang terakhir, dimana struktur ini tersusun dari beberapa rantai polipeptida. Asam amino yang berjauhan pada struktur *tertiary* dan *quarternary* digabungkan agar dapat melakukan fungsinya (Branden and Tooze, 2012).

Salah satu model matematika yang dipelajari secara luas untuk pelipatan protein adalah model *Hydrophobi-Polar*. Pada model ini, 20 tipe asam amino diklasifikasi menjadi *Hydrophobic* (H) dan *Polar* (P). Penyederhanaan deret asam amino ini dilakukan karena interaksi *Hydrophobic* pada protein adalah faktor yang penting dalam proses pelipatan protein (Li et al., 2018b). Dapat diasumsikan bahwa informasi dalam proses pelipatan terkandung

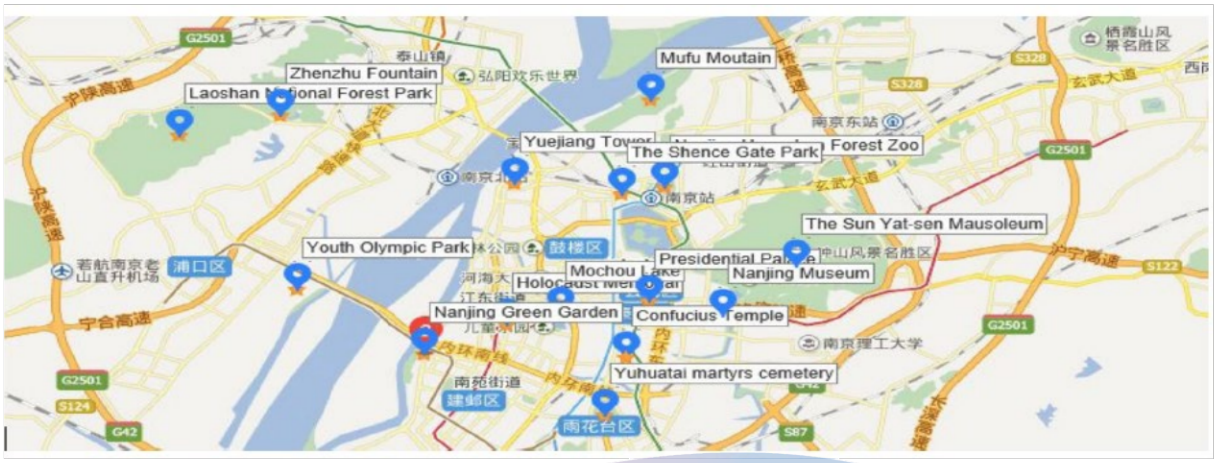
secara eksklusif pada urutan linier asam amino. Setelah dalam keadaan tiga dimensi yang stabil, protein dapat melakukan fungsinya yaitu interaksi tiga dimensi dengan protein lain dan interaksi yang memediasi fungsi organisme. Oleh karena itu, protein dapat dianggap sebagai unit dasar kehidupan, sehingga dengan memahami struktur dan fungsi protein, akan memberikan pamahaman yang lebih baik akan proses yang terjadi pada hidup suatu organisme (Czibula et al., 2011).

2.1.1. Protein Folding Problem

Memprediksi struktur *native* protein berdasarkan deret asam aminonya dan memahami mekanisme bagaimana protein terlipat dikenal sebagai *protein folding problem*. Beberapa teori yang dikembangkan telah memberikan pemahaman tentang mekanisme pelipatan protein. Namun simulasi pelipatan protein secara komputasi masih sangat sulit (Li et al., 2018a). Masalah ini adalah salah satu tantangan terbesar yang dihadapi oleh peneliti pada bidang bioinformatika karena memiliki tujuan penelitian yang penting terhadap pembuatan obat, memprediksi penyakit dan peningkatan fungsi dari suatu protein tertentu (Czibula et al., 2011).

Protein folding problem juga termasuk masalah optimasi kombinatorial yang sulit dipecahkan secara optimal. Oleh karena itu, segala upaya untuk memperbaiki masalah ini begitu berharga (Jafari and Javidi, 2020). Masalah ini juga termasuk dalam masalah *NP-complete* yang mangacu pada konteks untuk memprediksi struktur dari sebuah protein berdasarkan deret asam aminonya (Czibula et al., 2011.). Solusi dari masalah optimasi kombinatorial umumnya mengandung pertimbangan dari banyaknya desain konfigurasi yang memungkinkan. Kompleksitas masalah ini terus meningkat dengan faktorial dari n buah *nodes* (Halim and Ismail, 2017).

Selain pelipatan protein, masalah lain yang juga termasuk sebagai masalah optimasi kombinatorial adalah adalah *traveling salesman problem* (TSP) (Bello et al., 2017). TSP adalah permasalahan untuk menentukan rute *salesman* agar dapat mengunjungi semua kota dan tiap kota hanya dapat dikunjungi sekali. *Salesman* akan memulai dari satu kota dan kembali ke kota asalnya dengan rute yang memiliki jarak dan biaya paling minimum. (Amin, Ikhsan and Wibisono, 2005). Gambar 2.2 merupakan salah satu contoh penerapan algoritma *two phase heuristic algorithm* (TPHA) terhadap masalah *multiple-travelling salesman problem* (MTSP) yang kemudian dievaluasi pada peta elektronik *Baidu* (Xu et al., 2018).



Gambar 2. 2 Peta Elektronik Baidu

Sumber : (Xu et al., 2018)

2.1.2. Model *Hydrophobic-Polar*

Model *lattice* adalah salah satu model yang dapat digunakan untuk memprediksi struktur protein (Bechini, 2013). Walaupun model ini hanya memberikan gambaran kasar dari proses pelipatan sebuah protein, namun model ini telah memberikan pengetahuan yang penting atas mekanisme pelipatan dari struktur protein (Perdomo-Ortiz et al., 2012). Model *Hydrophobic-Polar* (HP) adalah salah satu model berbasis *lattice* yang dipelajari secara luas untuk pelipatan protein. Pada model ini, 20 jenis asam amino yang berbeda diklasifikasikan sebagai hidrofobik (H) atau polar (P) (Li et al., 2018b). Dalam proses pelipatan protein, *hydrophobicity* merupakan faktor penting yang membedakan antar asam amino. Berdasarkan sifatnya terhadap air, asam amino diklasifikasikan menjadi dua jenis, yaitu:

1. *Hydrophobic* atau non-polar (H), merupakan asam amino yang memiliki sifat menolak terhadap air.
2. *Hydrophilic* atau polar, merupakan asam amino yang memiliki sifat tidak menolak terhadap air.

Model HP merupakan model yang didasarkan dari hasil observasi bahwa *hydrophobic* merupakan faktor penting yang mengarahkan protein ke stuktur *native* tiga dimensi (Czibula et al., 2011.). Tujuan dari model HP adalah untuk menemukan *global minimum* dari fungsi energi. Protein dalam keadaan *native* diasumsi memiliki *free energy* yang paling minimum, dan oleh karena itu proses pelipatan dimaksudkan untuk meminimalisasi energi ini (Anfinsen, 1973).

Struktur primer protein dilihat sebagai deret linier dari *n* buah asam amino dan setiap asam amino diklasifikasi menjadi dua kategori: *hydrophobic H* dan hydrophilic *P*, didefinisikan sebagai :

$$\mathcal{P} = p_1 p_2 \dots p_n \; where \; p_i \in \{H, P\}, \forall 1 \leq i \leq n \quad (1)$$

Keterangan:

$\mathcal{P}$ = Protein $\mathcal{P}$ tersusun dari deret asam amino $p_1 p_2 \dots p_n$.

p_i = Elemen dari himpunan $\{H, P\}$

H = Amino *Hydrophobic*

P = Amino *Hydrophillic*

n = Panjang deret asam amino

Konformasi dari protein $\mathcal{P}$ adalah fungsi C, dimana fungsi ini memetakan deret linier protein $\mathcal{P}$ ke dalam kisi(*lattice*) kartesian dua dimensi.

Jika dinyatakan:

$$B = \{\mathcal{P} = p_1 p_2 \dots p_n | \; p_i \in \{H, P\}, \forall 1 \leq i \leq n, n \in N\} \quad (2)$$

$$\mathcal{G} = \{G = (x_i, y_i) | x_i, y_i \in \Re, 1 \leq i \leq n\} \quad (3)$$

Keterangan:

B = Himpunan protein $\mathcal{P}$.

$\mathcal{G}$ = *lattice* dua dimensi

G = koordinat asam amino ke- i (x_i, y_i)

$\Re$ = Ruang dua dimensi *lattice* $\mathcal{G}$

Maka konformasi C juga dapat didefinisikan sebagai berikut:

$$C: B \rightarrow \mathcal{G} \quad (4)$$

$$\mathcal{P} = p1p2 \dots p_n \mapsto \{(x_1, y_1), (x_2, y_2) \dots . (x_n, y_n)\} \quad (5)$$

(x_i, y_i) menyatakan posisi di dalam *lattice*, dimana asam amino p_i dipetakan oleh fungsi $C, \forall 1 \leq i \leq n$

Pemetaan C disebut *path* jika:

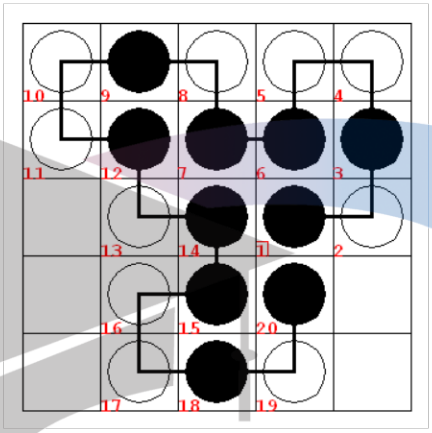
$$C, \forall 1 \leq i, j \leq n, \text{with } |i - j| = 1 \Longrightarrow |x_i - x_j| + |y_i - y_j| = 1 \quad (6)$$

Definisi (6) menyatakan bahwa fungsi C adalah *path* jika terdapat dua asam amino yang berurutan dalam struktur primer protein i, j merupakan tetangga (secara horizontal atau vertikal) di dalam *lattice*. Dimana setiap amino satu dengan yang sebelumnya selalu memiliki selisih urutan koordinat sebesar 1. Hal tersebut diasumsikan bahwa asam amino yang terdapat di *lattice* dalam posisi apapun mungkin memiliki jumlah maksimum dari 4 tetangga, yaitu: *up*, *down*, *left*, *right*.

Path C disebut *self-avoiding* jika fungsi C merupakan injeksi:

$$\forall 1 \leq i, j \leq n, \text{with } i \neq j \Longrightarrow (x_i, y_i) \neq (x_j, y_j) \quad (7)$$

Definisi pada persamaan (7) menegaskan bahwa posisi yang dipetakan dari dua asam amino yang berbeda tidak dapat bertumpang tindih di dalam *lattice*. Konfigurasi C dianggap valid jika *path*-nya *self-avoiding*. Contoh konfigurasi protein berdasarkan model HP dapat dilihat pada gambar 2.3, dimana protein $\mathcal{P}$ tersebut memiliki panjang 20. Lingkaran hitam menyatakan asam amino *hydrophobic*, sedangkan lingkaran putih menyatakan asam amino *hydrophilic*..



Gambar 2. 3 Konfigurasi Protein Untuk Deret Linear $\mathcal{P} = HPHPPHHPHPPHPHHPPHPH$

Sumber : (Czibula et al., 2011)

Fungsi energi pada model HP merepresentasikan fakta bahwa asam amino *hydrophobic* memiliki kecenderungan untuk membentuk inti *hydrophobic*. Akibatnya fungsi energi menambahkan nilai -1 untuk setiap asam amino yang dipetakan oleh C pada posisi yang bertetanggaan di dalam *lattice*, tetapi tidak berurutan dalam struktur primer protein $\mathcal{P}$. Dua asam amino yang dijelaskan sebelumnya disebut dengan topologi bertetangga.

Setiap asam amino *hydrophobic* dalam konformasi C yang valid dapat memiliki paling banyak 2 tetangga (kecuali asam amino yang pertama dan terakhir, yang dapat memiliki paling banyak 3 tetangga topologi).

Jika ditetapkan fungsi *I* sebagai :

$$I: \{1, \dots, n\} \times \{1, \dots, n\} \rightarrow \{-1,0\} \text{ where } \forall 1 \leq i,j \leq n, \text{with } |i = j| \geq 2 \quad (8)$$

Dimana fungsi I merupakan 2 asam amino yang bertetangga didalam lattice dengan selisih koordinat lebih besar dari 2 .

$$I(i,j) = \begin{cases} -1 & \text{if } p_i = p_j = H \text{ and } |x_i - x_j| + |y_i - y_j| = 1 \\ 0 & \text{otherwise} \end{cases} \quad (9)$$

Fungsi $I(i,j)$ menyatakan bahwa, jika asam amino p_i dan asam amino p_j merupakan asam amino *Hydrophobic* H, maka akan mendapatkan *free energy* sebesar -1 dan selainnya akan mendapatkan *free energy* sebesar 0.

Fungsi energi E untuk konformasi C yang valid ditetapkan sebagai berikut:

$$E(C) = \sum_{1 \le i \le j-2 \le n} I(i,j) \quad (10)$$

Dimana fungsi energi $E(C)$ adalah akumulasi dari fungsi $I(i,j)$ untuk setiap koordinat asam amino i lebih besar dari 1 dan lebih kecil dari asam amino $j-2$. Dimana $j-2$ lebih kecil dari panjang deret asam amino n.

Untuk setiap pasangan asam amino *hydrophobic* yang bertetangga di dalam *lattice*, namun tidak didalam struktur primer protein akan ditambahkan nilai -1. Fungsi energi untuk contoh konfigurasi yang ditunjukkan pada gambar 2.3 adalah -9, dimana pasangan yang menghasilkan fungsi energi adalah: (1,6), (1,14), (1,20), (3,6), (7,12), (7,14), (9,12), (15,18), (15,20).

Sebuah solusi untuk permasalahan pelipatan protein pada *bidimensional* HP dinyatakan dengan notasi π, dengan definisi sebagai berikut:

$$\pi = \pi_1 \pi_2 \dots \pi_{n-1},\ \pi_i \in \{L, R, U, D\}, \forall 1 \le i \le n-1 \quad (11)$$

Keterangan:

π = Solusi untuk deret linier protein $\mathcal{P}$

π_i = Elemen dari himpunan $\{L, R, U, D\}$

L = Left (Kiri)

R = Right (Kanan)

U = Up (Atas)

D = Down (Bawah)

Dimana untuk setiap π_i menginformasikan arah asam amino saat ini relatif terhadap asam amino sebelumnya (L-left, R-right, U-up, D-down) dan solusi π untuk panjang n deret linear $\mathcal{P} \in B$ dapat dinyatakan dengan $n-1$. Sebagai contoh, solusi dari konfigurasi deret linier protein $\mathcal{P}$ pada gambar 2.3 adalah $\pi = RUULDLULLDRDRDLDRRU$ (Czibula et al., 2011.).

2.1.3. *Protein Folding* Pada Model *Bidimensional* HP Adalah *NP-complete*

Bidimensional protein folding dalam model HP termasuk dalam jenis permasalahan NP-*complete*. Sebelumnya sudah terdapat beberapa literatur yang membahas tentang NP-*complete* terhadap pelipatan protein. Beberapa peneliti menunjukkan bahwa pada umumnya penyajian kembali dari permasalahan, dimana monomer saling menarik atau saling menolak dengan cara umum yang dapat digunakan dalam penyandian, merupakan NP-*complete* (Fraenkel, 1993);

(Ngo et al, 1994); (Unger and Moult, 1993). Hal tersebut juga dibuktikan oleh Paterson dan Przytycka bahwa generalisasi kombinasi model HP ke Alfabet tak terbatas, dimana satu simbolnya netral seperti HP's 0 simbol, dan skor menghitung jumlah kedekatan elemen dengan simbol yang sama, merupakan NP-*complete* (Paterson and Przytycka, 1996). (Nayak et al, 1997) meningkatkan pembuktian ini menjadi alfabet terbatas, meskipun merupakan alfabet yang sangat besar. Ini merupakan hasil pertama yang menyelesaikan kompleksitas model HP dua dimensi dengan lebih sederhana. Masalah model HP telah disinggung dari sudut pandang algoritma *approximation* (Hart and Istrail, 1995); hasil yang diperoleh memberikan sedikit pencerahan terhadap aspek masalah ini (Crescenzi et al., 1998).

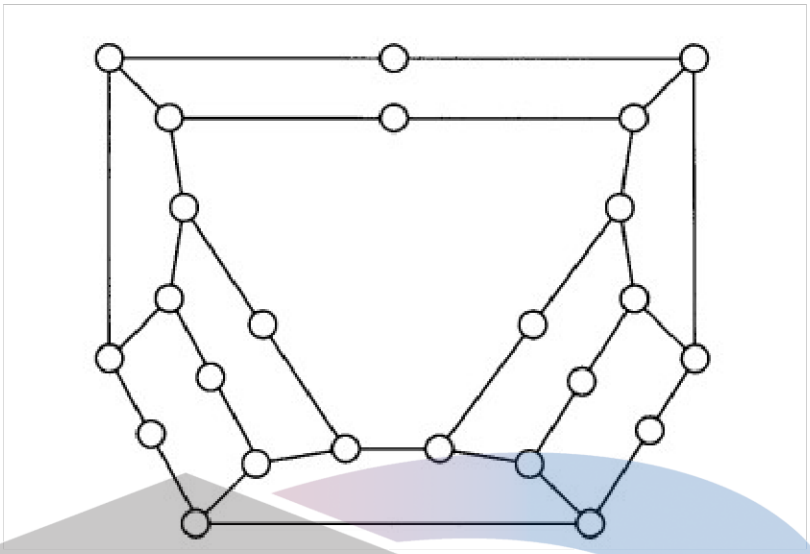
2.1.3.1. The MULTISTRING Folding Problem

Lattice dua dimensi merupakan sebuah grafik, (Z^2, L), dengan set node Z^2(semua titik pada bidang *Euclidean* dengan koordinat berupa bilangan bulat integer), dan set tepi $L = \{((x,y),(x',y')) : |x - x'| + |y - y'| = 1\}$. Sebuah pelipatan tetap f adalah sebuah titik $f(i,j)$, dan $f(i,j+1)$ disebut sebagai *f-neighbours*. Skor dari sebuah pelipatan f merupakan jumlah dari tepi $\{(x,y),(x',y') \in L\}$. Sebuah tepi pada *lattice* $\{(x,y),(x',y') \in L\}$ dikatakan sebagai sebuah *loss* (kerugian), jika titik tersebut bukan termasuk *f-neighbours* dan tepat satu dari dua titik tersebut merupakan sebuah gambar di bawah *f* dari sepasang (i,j).

MultiString Folding Problem telah dibuktikan sebagai NP-*complete* (cresenzi et all). *MultiString Folding Problem* adalah dengan diberikannya sebuah set strings $s_1, \ldots, s_m \in \{0,1\}$ dan integer *E*, apakah akan terjadi sebuah pelipatan(*folding*) dengan *E* atau nilai kerugian (*looses*) paling sedikit?

2.1.3.2. The String Folding Problem

Bagian ini menunjukkan bahwa *string folding problem* (kasus spesial dari *multistring problem* dengan |S| = 1, yang mencakup *protein folding problem* di dalam model HP 2-dimensi) juga termasuk *NP-complete*. Disebut grafik *planar special* jika terdiri bagian yang terputus-putus dengan node tingkat tiga, terhubung bersama oleh *path* yang panjangnya dua, dan menjadi *triply* terhubung jika semua node tingkat dua *collapsed*. Contoh dapat dilihat pada gambar 2.4.

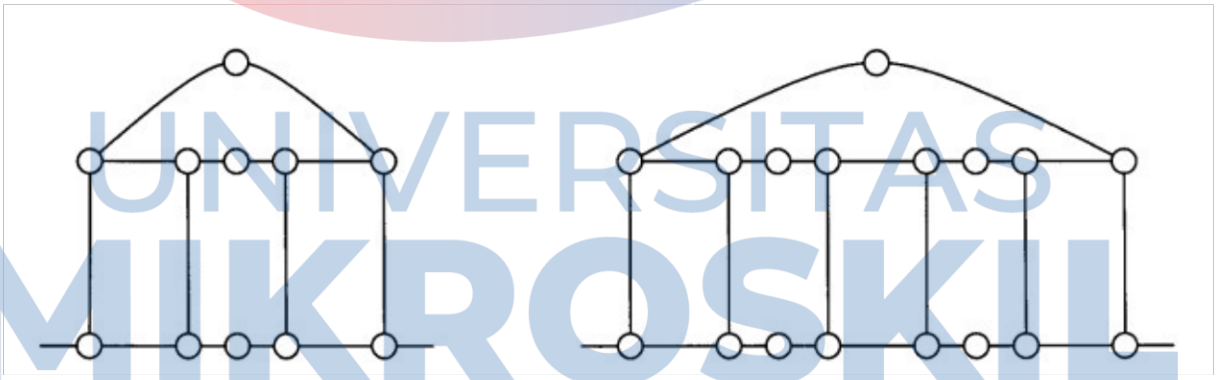


Gambar 2. 4 Grafik *Special Planar*

Sumber : (Crescenzi et al., 1998)

Teorema 1: *Hamilton cycle problem* tetap *NP-complete* bahkan jika terbatas pada grafik *planar*.

Bukti: reduksi dari kover yang tepat ke *planar Hamilton cycle* (Johnson and Papadimitrion, 1985) menghasilkan grafik *special planar*, jika 2-*input* dan 3-*input* "*or*" *gadgets* diganti dengan yang ditunjukkan pada gambar 2.5.



Gambar 2. 5 *2-input* Dan *3-input Gadgets* Baru

Sumber : (Crescenzi et al., 1998)

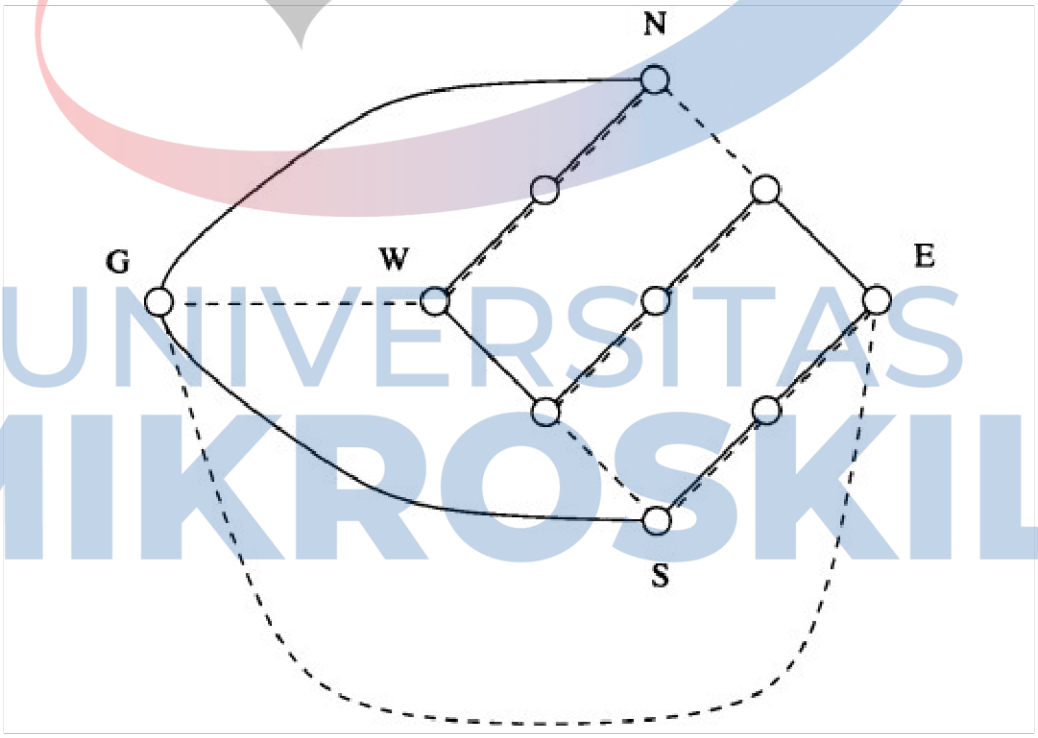
Memperbaiki grafik *planar* G, dua *Hamilton cycle* disebut *orthogonal* jika memiliki properti sebagai berikut: *disjoint union* (dimana terdapat duplikasi tepi pada titik pertemuannya) merupakan grafik *planar* tingkat 4 dengan banyak tepi yang dapat dimasukkan ke dalam bidang sedemikian rupa sehingga tepi yang disekitar setiap node bergantian antara dua siklus. Gambar 2.6 menggambarkan dua *Hamilton* cycles dari grafik *diamond* (ditambah node G lain); merupakan *orthogonal* karena, dengan menduplikasi tiga *path* yang panjangnya dua, satu

mendapatkan grafik tingkat 4 disekitar setiap node yang tepinya dua *Hamilton cycles* bergantian.

Misalkan grafik berisi grafik *diamond* digambarkan pada gambar 2.6 (abaikan node G bertahan sampai grafik yang tersisa). Grafik *diamond* memiliki 4 titik akhir, dinyatakan N,S,E,W, dimana terhubung ke grafik yang tersisa. Setiap *Hamilton cycle* secara keseluruhan grafik harus melintasi *diamond* baik dari N ke S, atau dari E ke W (namun tidak dari E ke N).

Teorema 2: *String folding problem* merupakan *NP-complete*.

Bukti: dimulai dari *Hamilton cycle problem* untuk grafik *special planar*. Diberikan grafik *special planar* G, maka harus memodifikasi grafik sehingga mengandung "*standard" Hamilton cycle* H_0, sedemikian rupa sehingga *Hamilton cycle* apa pun dari grafik asli sesuai dengan *cycle* dari grafik yang dimodifikasi yang *orthogonal* terhadap H_0. Dimulai dari G dan *embedding*-nya, hanya membutuhkan node tingkat 2 dari G, dan mempertimbangkan dua node yang berdekatan jika berada dalam *embedding* yang sama. Karena grafik asli yang special, hasil dari grafik *G'* terhubung.



Gambar 2. 6 Grafik *Diamond*

Sumber : (Crescenzi et al., 1998)

Mempertimbangkan *cycle C* dari *G'* (memperbolehkan perulangan node tetapi tidak untuk perulangan terhadap diri sendiri) yang mengunjungi semua node dari *G'* setidaknya satu kali. Jika dua node tersebut berdekatan dengan sebuah kejadian dari V berada pada tahap yang

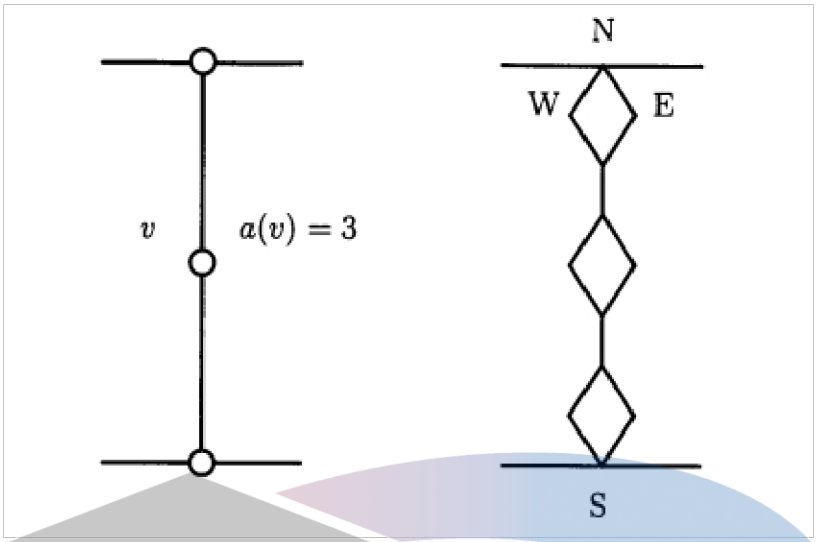
sama, ulangi kejadian itu dua kali. Sekarang, untuk setiap node V, hitung kejadian dari V pada C dan A(v) dijadikan sebagai hasilnya.

Sekarang gantikan setiap node V tingkat 2 dari G, dan ujung yang berdekatan, dengan a(v) salinan dari *diamond;* salinan tersebut dipisahkan, dan node N dan S dari dua yang paling luar (atau yang paling unik, jika $a(v) = 1$) berepatan dengan node dari G berdekatan terhadap v (lihat gambar 2.7). $C = (v_0, v_1, \dots, v_k = v_0$. Untuk $i = 1, \dots, k-1$, seharusnya element ke-I dari C adalah kejadian ke-bi dari node $v_i (b_k = 1)$; untuk setiap $i = 1, \dots, k-1$ gabungkan node E atau W dari salinan ke-bi-1 dari *diamond* menggantikan node v_{i-1}, mana saja yang belum dipertimbangkan sampai sekarang, dengan node E atau W dari Salinan ke-bi dari *diamond* menggantikan node vi, mana saja yang berada pada tahap yang sama dengan node sebelumnya (untuk $v_{0,}\ if\ v_1 \neq v_0$, dimulai dengan titik akhir, E atau W, yang berada pada tahap yang sama dengan $v_{1,}$, dan jika $v_1 = v_0$, mulai dengan titik akhir yang tidak pada v_2). Mengetahui bahwa sudut-sudut yang baru tidak membahayakan *planarity* dari grafik, dan bersama dengan E-W *traversal* dari *diamonds*, dari *standard Hamilton cycle*, $H_{0,}$, dari hasil grafik G".

H_0 merupakan satu-satunya *Hamilton cycle* dari G yang memanfaatkan sebuah *traversal* E-W dari *diamonds*. Semua *Hamilton cycle* memanfaatkan sebuah *tranversal* N-S harus sesuai terhadap *Hamilton cycle* dari grafik G yang asli. Mudah untuk melihat *cycle* manapun akan *orthogonal* dengan H_0 karena *traversals* E-W dan N-s dari *diamonds* merupakan *orthogonal.* Sekarang harus membangun instansi dari *string folding problem.* Ambil node tingkat 2 manapun dan digantikan dengan dua node tingkat 1, dan tetapkan node-node tesebut menjadi titik akhir dari *Hamilton path sought*. H_0 juga menjadi Hamilton *path*. Sekarang transformasi *Trevisan's* mengalami penghapusan E-W dari grafik (titik akhir dari sudut-sudut ini mempunyai jarak *hamming* yang lebar pada *Trevisan code*). Kemudian lakukan pengurangan *multistring,* dengan modifikasi berikut:

1. Jumlah dari *string* yang sesuai dengan setiap *city* L/n janggal. Mudah untuk menyelesaikannya dengan menambahkan satu *string* pada setiap set.
2. Semua *string* yang sesuai dengan *city* yang sama terhubung dengan satu *string*, dengan mengurutkannya dan menghubungkan akhir dari *string* $2i - 1$ dengan ujung dari *string* $2i$, dan awal dari *string* $2i$ dengan awal dari *string* $2i + 1$,

$$\boldsymbol{i = 1, \dots, \frac{\frac{L}{n} - 1}{2} s} \tag{12}$$



Gambar 2. 7 Mengganti Node Tingkat 2 Dengan *Diamond*

Sumber : (Crescenzi et al., 1998)

3. Akhirnya, semua *n long string* terhubung bersama dalam urutan yang disarankan *Hamilton path* H_0 dengan panjang $(2L^4 + 2L^2)$ *string* dari 0's.

Diklaim bahwa ada *folding* dengan E *losses* jika hanya grafik *special* asli mempunyai *Hamilton cycle*, misalkan memang ada *folding* dengan E atau *losses* yang lebih sedikit. *Hamilton path* di dalam grafik G" tidak memanfaatkan tepi E-W, dan karenanya *Hamilton cycle* berada di grafik asli.

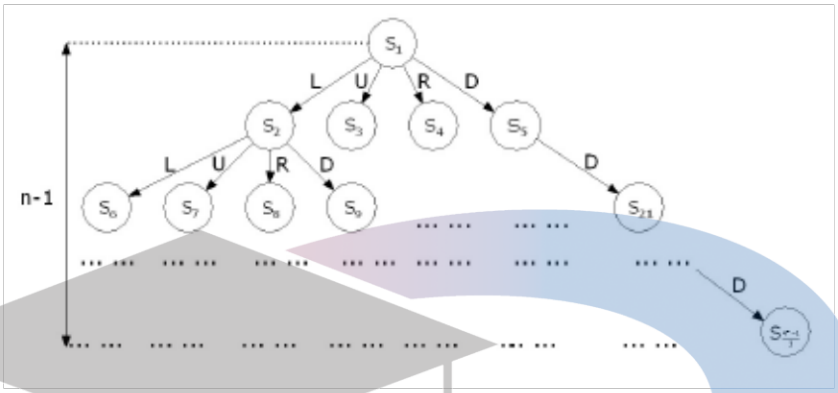
Sebaliknya, misalkan G memiliki *Haimlton cycle*. Maka G" memiliki *Hamilton cycle H* selain H_0, dan *orthogonal* terhadap H_0. Namun ini berarti dapat mengatur *n strings* sesuai dengan *cities* seperti pada *folding* yang dimaksudkan, bergabung bersama karena melalui awalan dan akhir dalam urutan yang disarankan oleh H_0, karena H_0 *orthogonal* terhadap *H*. Hasil *NP-completeness* menetapkan dugaan secara luas bahwa *protein folding problem*, bahkan 2 dimensi HP yang paling sederhana pun, adalah *NP-Complete* (Crescenzi et al., 1998).

2.1.3.3. State Space Pada Model *Bidimensional* HP

Penelitian (Czibula et al., 2011) berfokus terhadap *protein folding problem* dalam model *bidimensional* HP yang termasuk ke dalam permasalahan *NP-complete*. Jumlah *state* pada model *bidimensional* HP terdiri atas :

$$State\ Space\ S = \frac{4^n - 1}{3} \tag{13}$$

Dimana n adalah panjang amino, maka $S = \{s1,\ s2, \ldots, s\frac{4^n-1}{3}\}$. *State space* dalam model *bidimensional* HP dapat direpresentasikan ke dalam bentuk *tree* seperti pada gambar 2.8 dibawah ini. Dapat dilihat jika semakin panjang deret amino n, maka jumlah *nodes* pada *tree* juga akan meningkat secara eksponensial.



Gambar 2. 8 *State Space*

Sumber : (Czibula et al., 2011)

2.2. Reinforcement Learning

Secara umum *machine learning* dibagi menjadi tiga kategori, yaitu *supervised learning, unsupervised learning* dan *reinforcement* learning (Andreanus and Kurniawan, 2017). *Reinforcement Learning* (RL) berbeda dengan *supervised* maupun *unsupervised learning*. RL tidak mengandalkan contoh yang tersedia untuk memperbaiki perilakunya (Sutton and Barto, 2018). Misalnya seperti yang diketahui, sebuah *agent* mampu mempelajari permainan catur dengan *supervised learning* yaitu, dengan diberikannya contoh-contoh dari segala situasi yang dapat terjadi dan langkah terbaik untuk menanggapinya. Namun, bagaimana jika tidak diberikan contoh-contoh tersebut, maka apa yang akan dilakukan *agent* ? (Russell and Norvig, 1995).

Agent tersebut belajar mencapai tujuannya melalui interaksi *trial*-dan-*error* pada lingkungannya. Selama proses pembelajaran, sistem adaptif akan mencoba beberapa tindakan di lingkungannya, kemudian tindakan tersebut diperkuat (*reinforced*) dengan menerima nilai skalar (*reward*) atas tindakannya (Czibula et al., 2011). Algoritma akan secara langsung berinteraksi dengan lingkungannya dan berusaha memaksimalkan akumulatif nilai *reward* (Sutton and Barto, 2018).

2.2.1. *Markov Decisison Process*

Framework markov decision process (MDP) adalah sebuah abstraksi dari jenis permasalahan pembelajaran *goal-directed* melalui interaksi. Semua jenis permasalahan yang bersifat *goal-directed* dapat disederhanakan menjadi tiga sinyal bolak-balik antara *agent* dan *environment*-nya, yaitu:

1. Sinyal untuk mewakili pilihan tindakan yang dibuat oleh *agent*-nya.
2. Sinyal untuk mewakili keadaan atau kondisi berdasarkan tindakan yang dibuat oleh *agent*-nya.
3. Sinyal untuk mendefinisikan tujuan dari *agent* tersebut.

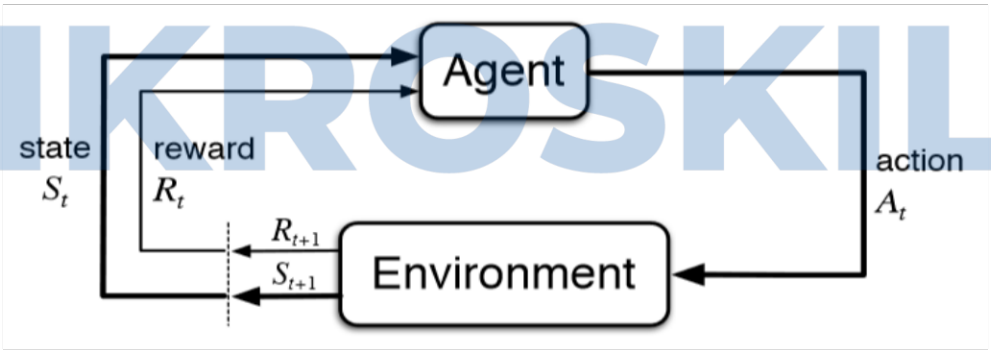
Framework ini mungkin masih tidak cukup untuk mewakili semua jenis permasalahan pembelajaran pengambilan-keputusan, namun *framework* ini telah terbukti sangat bermanfaat untuk diterapkan (Sutton and Barto, 2018).

MDP dapat dituliskan dengan persamaan dibawah ini :

$$M = \langle S, A, P, R, S_0 \rangle \tag{14}$$

Dimana, S merepresentasikan seluruh kemungkinan kondisi (*states*) yang dapat terjadi, A merepresentasikan seluruh kemungkinan tindakan (*action*) yang tersedia, P adalah probabilitas transisi dari suatu *states* S_t menuju *states* selanjutnya S_{t+1}, R adalah fungsi *reward* yang mengembalikan nilai skalar dan diperoleh saat mengambil *action* A_t pada *state* S_t, S_0 adalah *state* awal *agent* (van Seijen et al., 2019).

Berdasarkan *framework* MDP, maka arsitektur RL dapat diilustrasikan seperti gambar 2.9 di bawah ini.



Gambar 2. 9 Interaksi *Agent-Environment* Dalam MDP

Sumber : (Sutton and Barto, 2018)

2.2.2. *Policy* dan *Value Functions*

Secara formal, sebuah *policy* π adalah pemetaan dari *states* terhadap probabilitas pemilihan setiap kemungkinan *action* yang ada. Jika *agent* mengikuti *policy* π pada waktu t,

maka $\pi(a|s)$ adalah probabilitas bahwa $A_t = a$ jika diberikan $S_t = s$ (Sutton and Barto, 2018). Tugas *agent* adalah untuk mempelajari *control policy*, $\pi : S \rightarrow A$ untuk dapat memaksimalkan jumlah *reward* yang diharapkan pada masa yang akan datang (Czibula et al., 2011).

Ada dua tipe nilai fungsi (*value function*) pada *reinforcement learning* :

1. Nilai fungsi dari suatu *state* (*state-value function*) berdasarkan *policy* π, dilambangkan dengan $v_\pi(s)$, didefinisikan sebagai :

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s\right], \text{for all s} \in \mathcal{S}, \tag{15}$$

Dimana $v_\pi(s)$ merupakan ekspetasi $\mathbb{E}$ dari nilai *return* G_t atau jumlah *reward* R_t yang dikurangi (*discounted*) secara eksponensial γ^k saat dimulai pada *state s*, kemudian mengikuti *policy* π pada setiap langkah waktu *t*.

2. Nilai fungsi jika mengambil *action a* dari suatu *state s* (*action-value function*) berdasarkan *policy* π, dilambangkan dengan $q_\pi(s, a)$, didefinisikan sebagai:

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t|S_t = s,\ A_t = a\ ] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s,\ A_t = a\right], \tag{16}$$

Dimana $q_\pi(s, a)$ merupakan ekspetasi $\mathbb{E}$ dari nilai *return* G_t atau jumlah *reward* R_t yang dikurangi (*discounted*) secara eksponensial γ^k saat dimulai pada *state s*, mengambil *action a*, kemudian mengikuti *policy* π pada setiap langkah waktu *t* (Sutton and Barto, 2018).

2.2.3. Eksplorasi versus Eksploitasi

Salah satu kesulitan terbesar dalam RL adalah dilema atas eksplorasi *versus* eksploitasi (Andreanus and Kurniawan, 2017.). Hal ini mengakibatkan adanya sebuah *trade-off* antara eksplorasi dan eksploitasi, dan menjadi salah satu aspek penting dalam RL. Untuk mengumpulkan *reward* yang banyak, algoritma harus memilih *action* terbaik berdasarkan pengalaman sebelumnya, namun pada sisi lain juga harus mencoba untuk mengeksplorasi *action* yang baru (Czibula et al., 2011.).

Salah satu strategi untuk melakukan eksplorasi adalah dengan menerapkan ε-greedy. Dimana dengan probabilitas ε, dari semua kemungkinan *action* yang ada, algoritma akan memilih sebuah *action random* secara *uniform*. Sebaliknya, dengan probabilitas 1 – ε, algoritma

mengeksploitasi *action* terbaik. Dalam hal ini, tentu algoritma juga tidak ingin mengeksplorasi secara terus menerus. Untuk itu, saat algoritma sudah cukup melakukan eksplorasi, maka selanjutnya hanya tinggal melakukan eksploitasi saja, yaitu dengan cara memulai dengan nilai ε yang tinggi dan selanjutnya dikurangi secara bertahap (Alpaydin, 2010).

2.2.4. Q – learning

Q-learning adalah sebuah bentuk dari *model-free reinforcement learning* (Watkins, 1989). Pada *model-free*, sistem bahkan tidak mengetahui perubahan apa yang akan terjadi pada lingkungannya untuk merespon sebuah *action* (Sutton and Barto, 2018). *Q-learning* adalah pemetaan dari pasangan *state action* menjadi sebuah nilai atau disebut sebagai *Q-value*. Definisi adari *Q-value* adalah jumlah dari *reinforcement* yang diterima setelah melakukan sebuah *action* dan kemudian mengikuti *policy*, didefinisikan sebagai berikut :

$$Q(s,a) = r(s,a) + \gamma \max_{a'} Q(s',a'). \tag{17}$$

Dimana pada *Q-learning*, $Q(s,a)$ merupakan nilai *reward* yang diterima pada *state* s, saat mengambil *action* a dilambangkan dengan notasi $r(s,a)$, kemudian dijumlahkan dengan nilai maksimum *Q-value* pada *state* selanjutnya s' dan *action* a'. Nilai $\max_{a'} Q(s',a')$ dikurangi menggunakan sebuah *discount factor* $\gamma \in [0,1]$. Kemudian dengan sebuah *learning rate* $\alpha \in [0,1]$, *Q-value* pada (17) selanjutnya akan di-*update* menggunakan perhitungan (18) di bawah ini :

$$Q(s,a) \leftarrow (1-\alpha)\, Q(s,a) + \alpha \left(r(s,a) + \gamma \max_{a'} Q(s',a') \right). \tag{18}$$

(Czibula et al., 2011.)

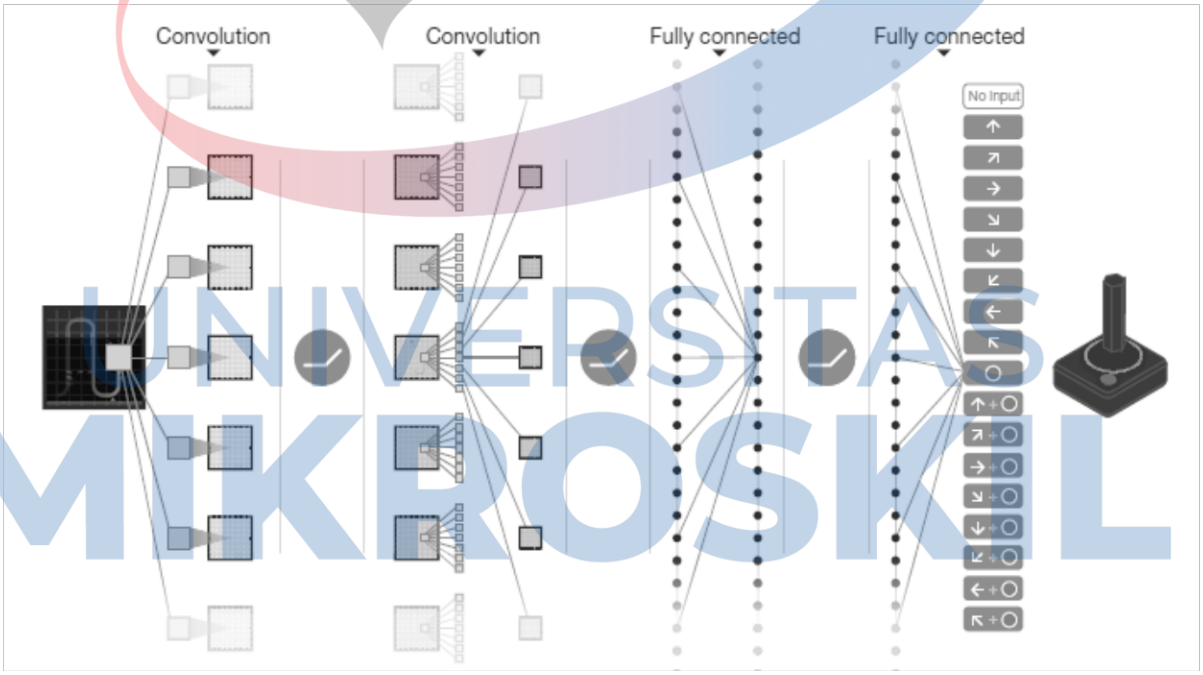
2.2.5. DQN

Pada *reinforcement learning*, ada dua sub permasalahan sulit yang cukup sering dihadapi. Sub masalah yang pertama adalah permasalahan *temporal credit assignment* (pemberian penghargaan sementara) (Sutton, 1984). Misalnya, sebuah *agent* memperoleh suatu hasil atas serangkaian *action* yang dilakukannya. Kesulitannya yaitu, mencari tau bagaimana cara memberikan *credit* (penghargaan) atau *blame* (menyalahkan) pada setiap situasi, sehingga performa pengambilan keputusannya dapat meningkat. Sub masalah yang kedua adalah permasalahan generalisasi atau penyamarataan. Permasalahan generalisasi yaitu, saat *state*

space-nya terlampau besar untuk dieksplorasi secara keseluruhan, *agent* harus mampu memperkirakan situasi selanjutnya berdasarkan pengalaman sebelumnya dengan situasi yang serupa (Lin, 1992.).

Generalisasi seperti ini biasanya membutuhkan sebuah *function approximator* untuk mengkonstruksi sebuah *approximation* (perkiraan) dari keseluruhan fungsi (misalnya, fungsi nilai). *Function approximation* merupakan sebuah contoh dari *supervised learning*, yang dimana merupakan sebuah topik utama yang dipelajari pada *machine learning*, misalnya seperti, *artificial neural network, pattern recognition, statistical curve fitting* (Sutton and Barto, 2018).

Deep Q-network (DQN) adalah kombinasi dari *reinforcement learning* dengan *artificial neural network* atau juga dikenal sebagai *deep neural networks*. DQN menggunakan arsitektur *deep convolutional neural network* untuk memperkirakan nilai *optimal action-value function* $Q_*(s,a)$. Ilustrasi arsitektur *deep convolutional neural network* pada gambar 2.10 digunakan oleh (Mnih et al., 2015) untuk diterapkan pada permainan klasik Atari 2600.

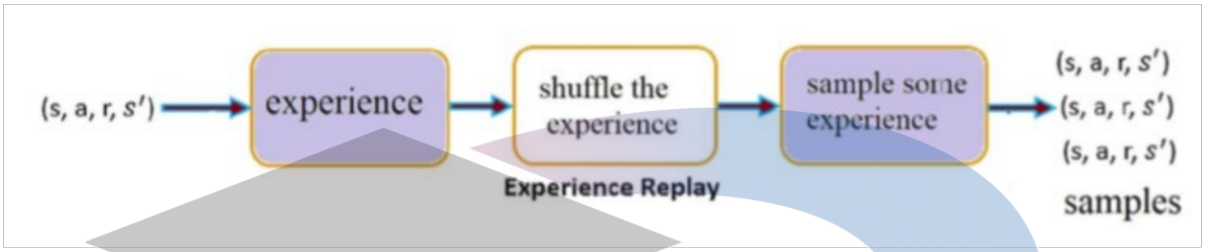


Gambar 2. 10 Ilustrasi Arsitektur *Deep Convolutional Network*

Sumber : (Mnih et al., 2015)

Reinforcement learning dikenal tidak stabil saat sebuah *function approximation non-liner*, seperti *deep neural network* digunakan untuk merepresentasikan nilai dari *action-value function*. Untuk mengatasi ketidakstabilan tersebut digunakan varian terbaru dari *Q-learning*, yang dimana menggunakan dua kunci utama. Pertama, digunakan mekanisme *experience*

replay pada gambar 2.11 yang terinspirasi secara biologis untuk mengacak keseluruhan data, kemudian menghapus kolerasi di dalam rangkaian observasi dan menstabilkan perubahan yang terjadi dalam distribusi data. Kedua, dilakukan *update* secara iteratif untuk menyesuaikan nilai *action-value* Q menuju ke arah nilai target $r + \gamma \max_{a'} Q(s', a')$ dan di-*update* secara berkala, dengan demikian maka dapat mengurangi kolerasi dengan nilai target.



Gambar 2. 11 Mekanisme *Experience Replay*

Sumber : (Jafari and Javidi, 2020)

Nilai *value-function* $Q(s, a;\ \theta_i)$ dapat diperkirakan dengan arsitektur *deep convolutional neural network* seperti pada gambar 2.10 di atas, yang dimana θ_i merupakan sebuah bobot atau parameter dari *Q-network* pada tiap iterasi ke- i. Untuk menerapkan *experience replay,* maka pada setiap langkah waktu ke-t, pengalaman (*experience*) e_t akan disimpan pada sebuah dataset D_t :

$$D_t = (e_1, \dots, e_t), dimana\ e_t = (s_t, a_t, r_t, s_{t+1}) \tag{19}$$

(s_t, a_t, r_t, s_{t+1}) merupakan informasi transisi dari s_t lalu mengambil *action* a_t dan memperoleh *reward* r_t kemudian bertransisi ke s_{t+1}. Selama proses pembelajaran, diterapkan *Q-learning* pada sampel (atau *minibatches*) dari *experience* $(s, a, r, s') \sim U(D)$, dan diambil secara *random uniform* U dari sekumpulan sampel yang disimpan pada D. Proses *update* pada *Q-learning* untuk tiap iterasi ke – i menggunakan *loss function* $L_i(\theta_i)$ sebagai berikut:

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s') \sim U(D)} \left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_i^{-}) - Q(s, a; \theta_i) \right)^2 \right], \tag{20}$$

θ_i^{-} adalah parameter network yang digunakan untuk menghitung target pada iterasi ke – i. Target *network* parameter θ_i^{-} hanya di-*update* menggunakan parameter *Q-network* (θ_i) setiap langkah ke C dan nilai tersebut didiamkan (tidak diubah) (Mnih et al., 2015). Algoritma untuk melatih *DQN* terdapat pada gambar 2.12.

Algorithm : deep Q-learning with experience replay.

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$

For episode = 1, M **do**

  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$

  For t = 1,T **do**

    With probability ε select a random action a_t

    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$

    Execute action a_t in emulator and observe reward r_t and image x_{t+1}

    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D

    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D

    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}) & \text{otherwise} \end{cases}$

    Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the network parameters θ

    Every C steps reset $\hat{Q} = Q$

  End For

End For

Gambar 2. 12 Algoritma *Deep Q-learning* Untuk Melatih *Agent* DQN

Sumber : (Mnih et al., 2015)

2.3. Logarithmic Mapping Reinforcement Learning

Nilai *discount factor* yang rendah pada RL cenderung memiliki performa yang buruk, terutama saat dikombinasikan dengan *function approximation*. Menurut beberapa hipotesis yang ada, dijelaskan bahwa penyebab *discount factor* rendah memiliki performa yang buruk adalah *action-gaps* yang terlalu kecil (van Seijen et al., 2019). *Action-gaps* ε adalah perbedaan antara nilai *action* yang paling optimal dengan yang kedua paling optimal, yaitu:

$$\varepsilon = Q^*(x_1, a_2) - Q^*(x_1, a_1) \tag{21}$$

Dimana $Q^*(x_1, a_1)$ merupakan *action* paling optimal pada *state* x_1 dan $Q^*(x_1, a_2)$ merupakan *action* kedua paling optimal pada *state* x_1

(Bellemare et al., 2015).

Secara umum, diyakini bahwa *action-gaps* memiliki dampak yang besar terhadap proses optimasi (Bellemare et al., 2015; Farahmand, 2011). Ada dua buah hipotesis yang melibatkan *action-gaps*, yakni sebagai berikut :

1. *Discount factor* yang rendah memiliki performa buruk dikarenakan memiliki nilai *action-gaps* yang lebih kecil.
2. *Discount factor* yang rendah memiliki performa buruk dikarenakan memiliki nilai *action-gaps* yang relatif lebih kecil (misalnya, *action gap* pada sebuah *state* dibagi dengan nilai maximum *action-value* pada *state* tersebut).

Namun kedua hipotesis tersebut telah dibuktikan gagal oleh (van Seijen et al., 2019), sehingga diusulkan hipotesis baru yaitu:

Discount factor rendah memiliki performa buruk dikarenakan memiliki *action-gaps* deviasi k yang tinggi di sepanjang state-space tersebut. Maka untuk dapat mengurangi k, diterapkan fungsi *logarithmic mapping* sebagai berikut:

$$f(x) := c \ln(x + \gamma^k) + d, \tag{22}$$

dan sebuah fungsi invers nya:

$$f^{-1}(x) := e^{(x-d)/c} - \gamma^k \tag{23}$$

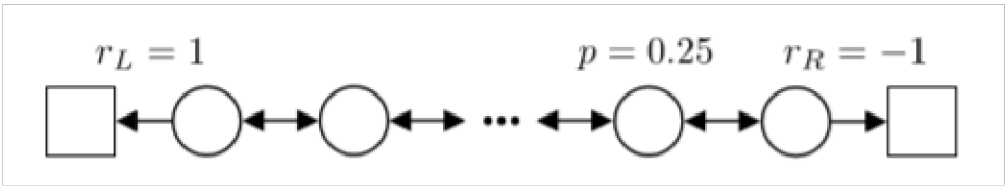
dimana c, d dan k merupakan *mapping hyper-parameters.* Parameter d digunakan untuk mengontrol inisialisasi daripada nilai *Q-value*. Berikut persamaan parameter d:

$$d = -c \ln(q_{init} + \gamma^k) \tag{24}$$

(van Seijen et al., 2019).

2.3.1. *Logarithmic Q – learning*

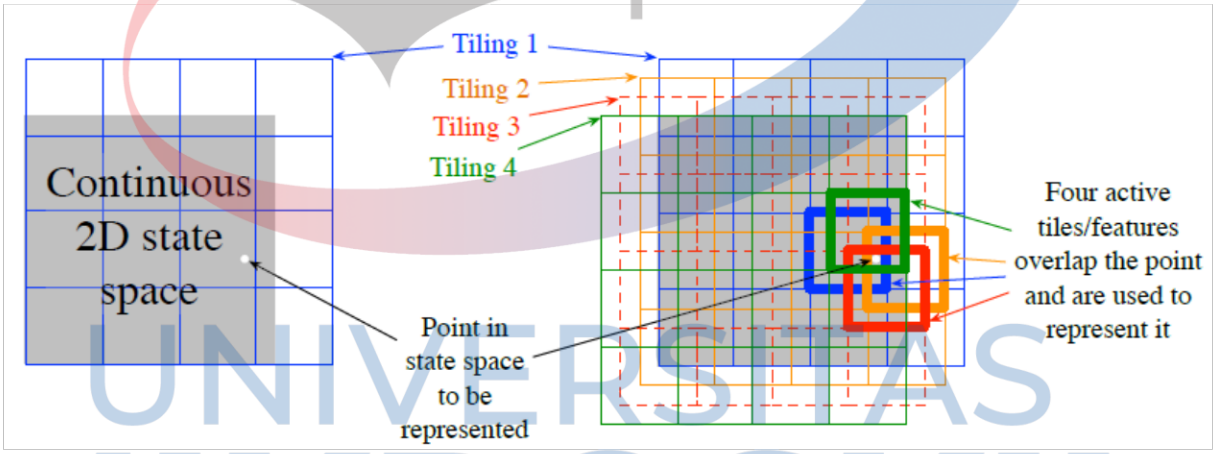
Logarithmic Q-learning adalah salah satu metode yang dapat mengurangi *action-gaps* deviasi k untuk domain *sparse-reward*. Metode ini dilakukan dengan pendekatan yang sama seperti yang dilakukan oleh (Pohlen et al., 2018), yaitu dengan memetakan *update* target pada *space* yang berbeda kemudian dilakukan *update* pada *space* tersebut juga. Pada *Q-learning* reguler dengan *function approximation*, *discount factor* rendah tidak memiliki performa yang baik pada domain *sparse-reward*. Hal tersebut dapat disimpulkan dari hasil eksperimen yang dilakukan oleh (van Seijen et al., 2019) pada *chain task* di gambar 2.13 di bawah ini



Gambar 2. 13 *Chain Task*
Sumber : (van Seijen et al., 2019)

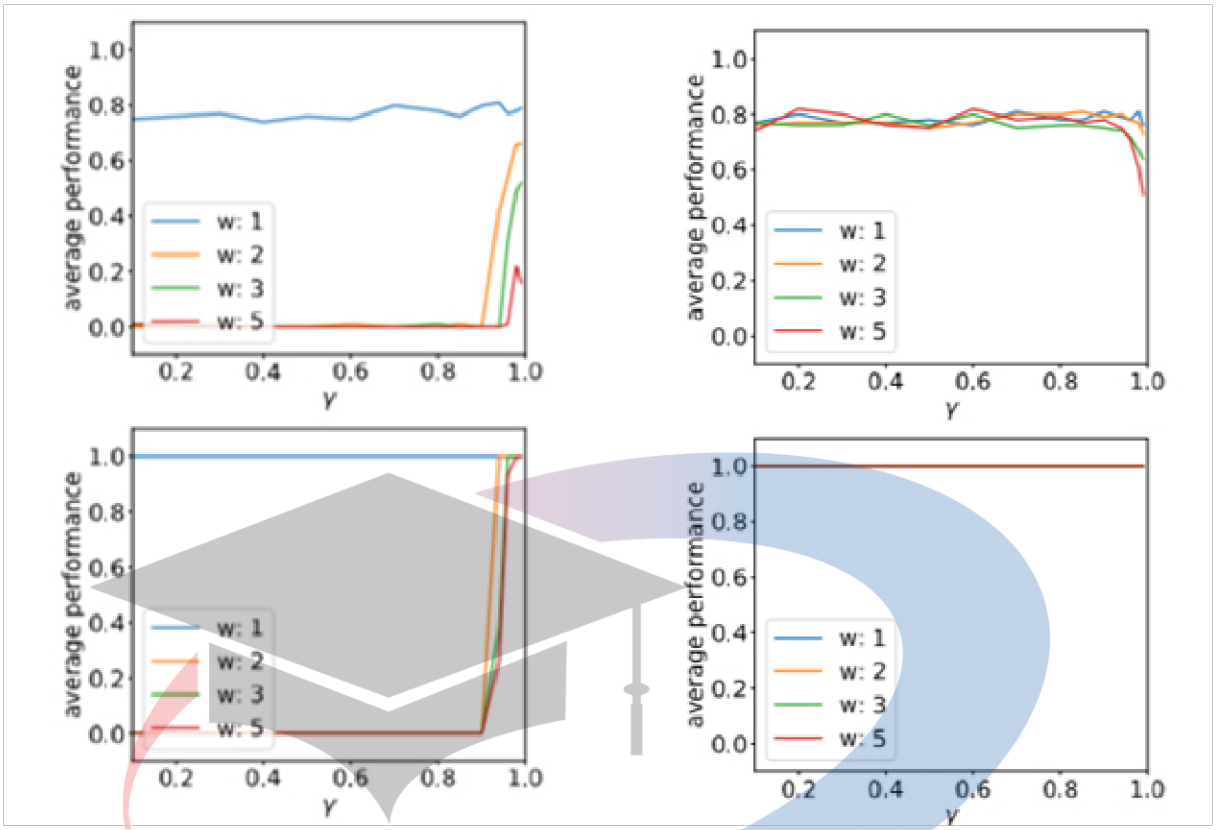
Chain Task terdiri dari 50 *states* dan 2 *terminal states*. Setiap *states* (*non-terminal*) memiliki dua *action*: a_L menghasilkan transisi ke sebelah kiri dengan probabilitas $1 - p$, dan dengan probabilitas p menghasilkan transisi ke sebelah kanan a_R. Keseluruhan nilai *reward* adalah 0, kecuali jika bertransisi ke *terminal states* paling-kiri atau paling-kanan, yang masing-masing menghasilkan r_L dan r_R.

Kemudian diterapkan *linear function approximation tile-coding* seperti pada gambar 2.14 untuk mengevaluasi dampak dari proses optimasi di bawah pemakaian *function approximation*.



Gambar 2. 14 *Linear Function Approximation* Dengan Nama *Tile-Coding*
Sumber : (van Seijen et al., 2019)

Gambar 2.15 menunjukkan bahwa penerapan fungsi *logarithmic mapping* pada *Q-learning* yang dilakukan oleh (van Seijen et al., 2019) berhasil menyelesaikan masalah optimasi yang sebelumnya terjadi pada *Q-learning* reguler terhadap penggunaan *discount factor* rendah dalam hubungannya dengan *function approximation*.



Gambar 2. 15 Performa Awal (atas) Dan Performa Akhir (bawah) *Regular Q-learning* (kiri) Dengan *Logarithmnic Q-learning* (kanan)
Sumber : (van Seijen et al., 2019)

Penulis (van Seijen et al., 2019) menggunakan *tile-width* 1, 2, 3 dan 5. Sebagai catatan bahwa untuk *width* : 1, representasi *features* direduksi menjadi datar (tabular) atau tersusun dalam tabel. Sumbu x merupakan nilai *discount factor* yang digunakan, sementara sumbu y merupakan performa rata-rata *agent.*

2.3.2. Domain Deterministik Dengan Nilai *Reward* Positif

Penerapan fungsi *logarithmic* untuk domain deterministik dan nilai *reward* positif, menggunakan persamaan berikut untuk meng-*update* nilai $\tilde{Q}$ pada *mapping space*:

$$\tilde{Q}_{t+1}(s_t, a_t) := (1 - \alpha)\tilde{Q}_t(s_t, a_t) + \alpha f\left(r_t + \gamma \max_{a'} f^{-1}\left(\tilde{Q}_t(s_{t+1}, a')\right)\right). \quad (25)$$

sebagai catatan, $\tilde{Q}$ pada persamaan tersebut bukanlah sebuah estimasi dari *expected return* di *regular space*, melainkan *expected return* yang di-*mapping* pada *space* yang berbeda. Maka daripada itu, untuk memperoleh nilai Q-*value* yang reguler, maka harus diterapkan fungsi invers

mapping (23) terhadap $\tilde{Q}$. Untuk itu, *action gap* pada state s dalam *mapping space*, didefinisikan sebagai:

$$\tilde{Q}(s, a_{best}) - \tilde{Q}(s, a_{second\ best}) \tag{26}$$

(van Seijen et al., 2019)

2.3.3. Domain Stokastik Dengan Nilai *Reward* Positif

Pada domain stokastik, *step-size* pada *log-space* dilambangkan dengan $\beta_{\log}$, dan *step-size* pada reguler-*space* dilambangkan dengan β_{reg}. Oleh karena itu, *update* target U_t dimodifikasi menjadi $\hat{U}_t$, dan didefinisikan sebagai berikut:

$$\hat{U}_t := f^{-1}\left(\tilde{Q}_t(s_t, a_t)\right) + \beta_{\text{reg}} f^{-1}\left(U_t - f^{-1}\left(\tilde{Q}_t(s_t, a_t)\right)\right), \tag{27}$$

dimana *update* target $\hat{U}_t$ yang telah dimodifikasi digunakan untuk melakukan *update* pada *log-space*, sebagai berikut:

$$\tilde{Q}_{t+1}(s_t, a_t) := \tilde{Q}_t(s_t, a_t) + \beta_{\log}\left(f\left(\hat{U}_t\right) - \tilde{Q}_t(s_t, a_t)\right). \tag{28}$$

Sebagai catatan, jika $\beta_{\text{reg}} = 1$, maka $\hat{U}_t = U_t$, dan *update* (27) direduksi kembali menjadi *update* yang sebelumnya (24), dimana $\alpha = \beta_{\log}$.

(van Seijen et al., 2019)

2.3.4. Domain Stokastik Dengan Nilai *Reward* Positif dan/atau Negatif

Untuk domain stokastik yang *reward* nya bisa bernilai positif atau negatif (atau nol), *reward* r_t akan didekomposisi menjadi dua komponen yaitu, r_t^+ dan r_t^- sebagai berikut :

$$r_t^+ := \begin{cases} r_t & if\ r_t \geq 0 \\ 0 & otherwise \end{cases} \quad ; \quad r_t^- := \begin{cases} |r_t| & if\ r_t < 0 \\ 0 & otherwise \end{cases} \tag{29}$$

sebagai catatan, r_t^+ dan r_t^- selalu bernilai non-negatif, dimana $r_t = r_t^+ - r_t^-$ setiap kalinya. Dengan dilakukan dekomposisi *reward* seperti itu, maka kedua kedua komponen *reward* tersebut bisa digunakan untuk melatih dua $\tilde{Q}$-*value* secara terpisah, dimana:

1. $\tilde{Q}^+$, merepresentasikan nilai fungsi dalam *mapping space* berdasarkan r_t^+.
2. $\tilde{Q}^-$, merepresentasikan nilai fungsi dalam *mapping space* berdasarkan r_t^-.

Untuk melatih nilai fungsi tersebut, maka dikonstruksikan *update* targetnya sebagai berikut:

$$U_t^+ \coloneqq r_t^+ + \gamma f^{-1}\left(\tilde{Q}_t^+(s_{t+1}, \tilde{a}_{t+1})\right) \quad ; \quad U_t^- \coloneqq r_t^- + \gamma f^{-1}\left(\tilde{Q}_t^-(s_{t+1}, \tilde{a}_{t+1})\right), \tag{30}$$

Update target tersebut masing-masing akan dimodifikasi menjadi $\widehat{U}_t^+ \; dan \; \widehat{U}_t^-$ berdasarkan (27), kemudian selanjutnya digunakan untuk meng-*update* nilai $\tilde{Q}_t^+$ dan $\tilde{Q}_t^-$ berdasarkan (28). Pemilihan-*action* pada setiap langkah ke – t , menggunakan $Q_t(s,a)$ yang didefinisikan sebagai berikut:

$$Q_t(s,a) \coloneqq f^{-1}\left(\tilde{Q}_t^+(s,a)\right) - f^{-1}\left(\tilde{Q}_t^-(s,a)\right) \tag{31}$$

Untuk mengeksploitasi action terbaik $argmax_{a'}$, maka digunakan menggunakan persamaan berikut ini:

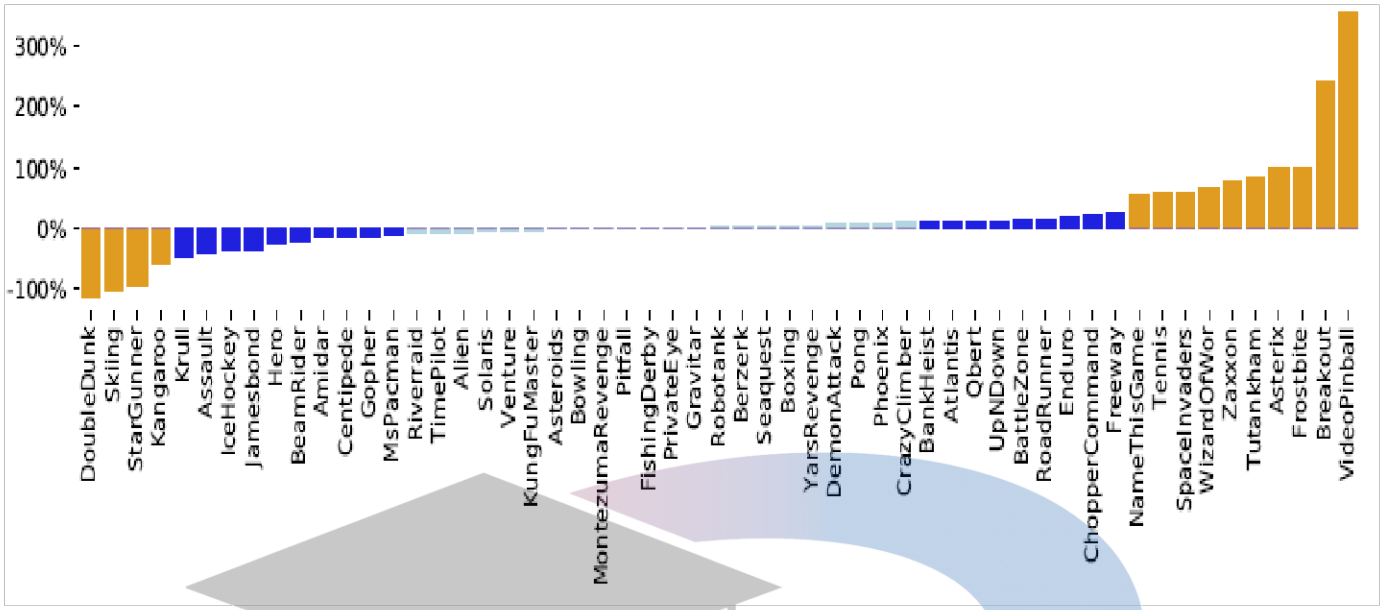
$$\tilde{a}_{t+1} \coloneqq \underset{a'}{argmax}\left(f^{-1}\left(\tilde{Q}_t^+(s_{t+1}, a')\right) - f^{-1}\left(\tilde{Q}_t^-(s_{t+1}, a')\right)\right) \tag{32}$$

Dimana $\tilde{a}_{t+1}$ adalah action selanjutnya yang akan diambil oleh agent (van Seijen et al., 2019)

2.3.5. *Logarithmic* DQN

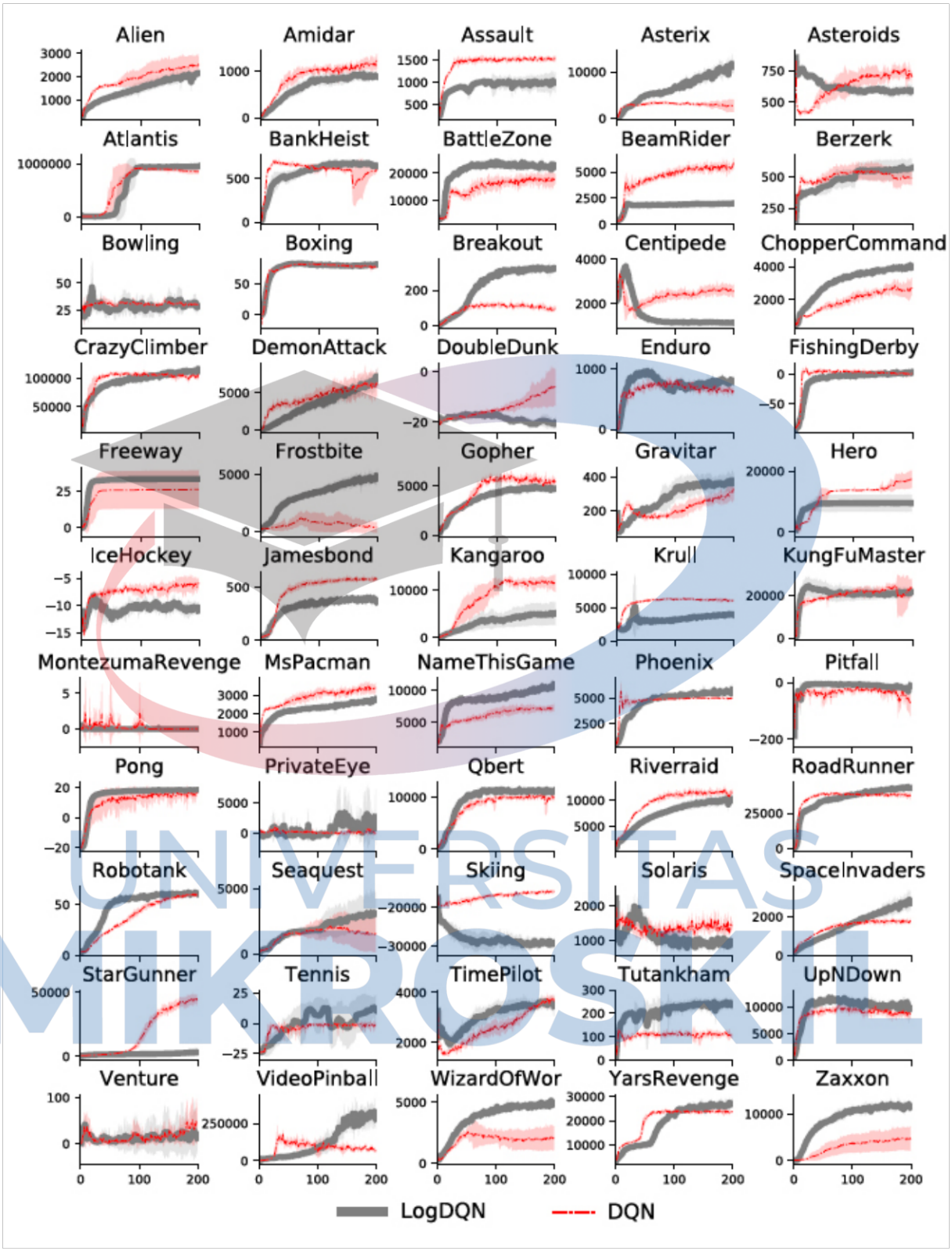
Logarithmic Deep Q-network (LogDQN) mengadaptasi model DQN (Mnih et al., 2015) seperti algoritma DQN pada gambar 2.12. Namun agar dapat memberikan nilai estimasi untuk $\tilde{Q}^+$ dan $\tilde{Q}^-$, LogDQN menggandakan jumlah *output layer*-nya menjadi dua kali lipat, dan setengah darinya digunakan untuk mengestimasi $\tilde{Q}^+$ dan sisa setengahnya digunakan untuk $\tilde{Q}^-$. Kedua nilai $\tilde{Q}^+$ dan $\tilde{Q}^-$ di-*update* menggunakan sampel yang sama, dan tidak ada modifikasi pada mekanisme *replay memory*, sehingga jejak langkah sebelumnya tidak berubah. Selanjutnya, dikarenakan $\tilde{Q}^+$ dan $\tilde{Q}^-$ di-*update* secara bersamaan secara *single pass* pada seluruh model, biaya komputasi antara LogDQN dan DQN adalah sama.

Eksperimen LogDQN yang dilakukan oleh (van Seijen et al., 2019) diterapkan pada *framework* Dopamine (Castro et al., 2018) yang bertujuan untuk mempermudah saat melakukan perbandingan. Alasannya adalah *framework* Dopamine tidak hanya memiliki beberapa kode pemrograman metode *Deep* RL yang terbuka untuk umum, namun juga memiliki hasil statistik yang diperoleh dari 60 set permainan *Arcade Learning Envrionment* (Bellemare et al., 2015). Maka perbandingan "apel dengan apel" dapat dilakukan. Gambar 2.16 merupakan perbandingan performa LogDQN relatif terhadap DQN, dan gambar 2.17 merupakan hasil kurva pembelajaran untuk 55 permainan.



Gambar 2. 16 Performa LogDQN Relatif Terhadap DQN (Persentase Positif Menunjukkan Performa LogDQN Mengungguli DQN)

Sumber : (van Seijen et al., 2019)



Gambar 2. 17 Kurva Pembelajaran Untuk 55 Permainan

Sumber : (van Seijen et al., 2019)

Beberapa pengaturan saat mengimplementasi LogDQN yang dilakukan oleh (van Seijen et al., 2019), yakni sebagai berikut:

1. *Loss function* yang digunakan adalah *Huber loss function* (Huber, 1992), dan untuk mengoptimasi *loss function* tersebut digunakan standar *RMSProp optimizer*.
2. Untuk menginisialisasi *network* LogDQN, digunakan standar inisialisasi Xavier (Glorot and Bengio, 2010) dengan pengecualian pada *output layer* untuk $\tilde{Q}^{-}$ diinisialisasi dengan bobot nol.
3. Nilai aditif γ^k dari *mapping function* diganti menjadi nilai inversnya, dan *hyper-parameter* d dengan *minimum-clipping* pada γ^k (misalnya, membatasi nilai tersebut menjadi nilai minimum yang memungkinkan dalam perhitungan yang sesuai). Hal ini memberikan batasan yang lebih kuat pada nilai yang dapat direpresentasikan, dan dapat meningkatkan independensi antara parameter k dan γ, yang dimana berguna saat hendak mengoptimasi *hyper-parameter*.