Jurnal Komputer dan Teknologi Sains (KOMTEKS)
Vol. 3, No. 1, April 2024, hlm. 10-17 e-ISSN: 2964-5247

ANALISIS KELAS KOMPLEKSITAS BERDASARKAN BERBAGAI JENIS MESIN TURING DALAM TEORI KOMPUTASI

Made Santo Gitakarma*[1]

[1]Teknologi Rekayasa Sistem Elektronika, Fakultas Teknik dan Kejuruan, Universitas Pendidikan Ganesha
Email: [1]santo@undiksha.ac.id
*Penulis Korespondensi

(Naskah masuk: 27 Maret 2024, diterima untuk diterbitkan: 1 April 2024)

Abstrak

Di dalam teori komputasi, banyak jenis mesin Turing digunakan untuk menentukan kelas kompleksitas, seperti mesin Turing deterministik, mesin Turing probabilistik, mesin Turing non-deterministik, mesin Turing kuantum, mesin Turing simetris dan mesin Turing bolak-balik. Semuanya sama-sama kuat pada prinsipnya, namun bila sumber daya (seperti waktu atau ruang) dibatasi, beberapa di antaranya mungkin lebih kuat daripada yang lain. Mesin Turing deterministik adalah mesin Turing yang paling dasar, yang menggunakan seperangkat aturan tetap untuk menentukan tindakan masa depannya. Mesin Turing probabilistik adalah mesin Turing deterministik dengan persediaan bit acak tambahan. Kemampuan untuk membuat keputusan probabilistik sering membantu algoritma memecahkan masalah dengan lebih efisien. Algoritma yang menggunakan random bits disebut algoritma acak. Mesin Turing non-deterministik adalah mesin Turing deterministik dengan fitur tambahan non-determinisme, yang memungkinkan mesin Turing memiliki banyak kemungkinan tindakan di masa depan dari keadaan tertentu. Artikel ini menjabarkan kelas-kelas kompleksitas penting yang didefinisikan dengan membatasi waktu (*Time Complexity Class*) atau ruang (*Space Complexity Class*) yang digunakan oleh algoritma, serta kelas kompleksitas dari algoritma acak yang disebut (*Randomized Complexity Class*).

Kata kunci: *mesin turing, teori komputasi, kelas kompleksitas, deterministik, non-deterministik*

ANALYSIS OF COMPLEXITY CLASSES BASED ON VARIOUS TYPES OF TURING MACHINES IN COMPUTATIONAL THEORY

Abstract

In computation theory, many types of Turing machines are used to determine complexity classes, such as deterministic Turing machines, probabilistic Turing machines, non-deterministic Turing machines, quantum Turing machines, symmetric Turing machines, and reversible Turing machines. They are all equally powerful in principle, but when resources (such as time or space) are limited, some may be more powerful than others. A deterministic Turing machine is the most basic type, using a fixed set of rules to determine its future actions. A probabilistic Turing machine is a deterministic Turing machine with an additional supply of random bits. The ability to make probabilistic decisions often helps algorithms solve problems more efficiently. Algorithms that use random bits are called randomized algorithms. A non-deterministic Turing machine is a deterministic Turing machine with the added feature of non-determinism, allowing the Turing machine to have multiple possible future actions from a given state. This paper will outline important complexity classes defined by limiting the time (Time Complexity Class) or space (Space Complexity Class) used by the algorithms, as well as the complexity classes of randomized algorithms (Randomized Complexity Class).

Keywords: *turing machine, computation theory, complexity class, deterministic, non-deterministic*

1. PENDAHULUAN

Teori kompleksitas komputasional adalah cabang teori perhitungan dalam ilmu komputer teoritis yang berfokus pada klasifikasi masalah komputasi sesuai dengan kesulitan inheren mereka, dan menghubungkan kelas - kelas itu satu sama lain. Masalah komputasi dipahami sebagai tugas yang pada prinsipnya dapat dipecahkan dengan komputer, yang setara dengan menyatakan bahwa masalahnya dapat dipecahkan dengan penerapan matematis secara mekanis, seperti algoritma.

Untuk mengukur kesulitan memecahkan masalah komputasi, seseorang mungkin ingin melihat

berapa banyak waktu yang dibutuhkan algoritma terbaik untuk memecahkan masalah. Namun, waktu yang berjalan mungkin, pada umumnya tergantung pada contohnya. Secara khusus, contoh yang lebih besar akan membutuhkan lebih banyak waktu untuk dipecahkan. Dengan demikian waktu yang dibutuhkan untuk memecahkan suatu masalah (atau ruang yang dibutuhkan, atau ukuran kompleksitas) dihitung sebagai fungsi dari ukuran instance. Ini biasanya dianggap sebagai ukuran input dalam bit. Jika ukuran input *n*, waktu yang dibutuhkan dapat dinyatakan sebagai fungsi dari *n*. Karena waktu yang dibutuhkan pada input yang berbeda dengan ukuran yang sama dapat berbeda, kompleksitas waktu terburuk $T(n)$ didefinisikan sebagai waktu maksimum yang diambil dari semua masukan dengan ukuran *n*. Jika $T(n)$ adalah polinomial dalam *n*, maka algoritma tersebut dikatakan sebagai algoritma waktu polinomial.

Untuk definisi yang tepat tentang apa artinya memecahkan masalah dengan menggunakan jumlah ruang dan waktu tertentu, model komputasi seperti mesin Turing deterministik digunakan. Waktu yang dibutuhkan oleh mesin Turing deterministik M pada input x adalah jumlah total transisi, atau langkah-langkah yang harus dilakukan mesin sebelum berhenti dan mengeluarkan jawabannya ("ya" atau "tidak"). Mesin Turing M dikatakan beroperasi dalam waktu $f(n)$, jika waktu yang dibutuhkan oleh M pada setiap masukan dengan panjang n paling banyak $f(n)$. Masalah keputusan A dapat diselesaikan pada waktunya $f(n)$ jika ada mesin Turing yang beroperasi pada waktu $f(n)$ yang memecahkan masalah. Karena dalam teori kompleksitas dilakukan klasifikasi masalah berdasarkan pada kesulitan mereka, seseorang mendefinisikan serangkaian masalah berdasarkan beberapa kriteria. Misalnya, kumpulan masalah yang dapat dipecahkan dalam waktu $f(n)$ pada mesin Turing deterministik kemudian dilambangkan dengan DTIME ($f(n)$).

Untuk mengklasifikasikan waktu komputasi (atau sumber daya yang serupa, seperti konsumsi ruang), perlu dibuktikan batas atas dan bawah pada jumlah minimum waktu yang dibutuhkan oleh algoritma yang paling efisien untuk memecahkan masalah yang diberikan. Untuk menunjukkan batas atas $T(n)$ pada kompleksitas masalah, kita perlu menunjukkan bahwa ada algoritma tertentu dengan waktu berjalan paling banyak $T(n)$.

Dengan tingkat pertumbuhan kebutuhan sumber daya karena input *n* meningkat maka diperlukan kelas kompleksitas. Kelas kompleksitas merupakan pengukuran abstrak, dan tidak memberi waktu atau ruang dalam persyaratan dalam hitungan detik atau byte, yang memerlukan pengetahuan tentang spesifikasi pelaksanaan. Untuk keperluan teori kompleksitas komputasional, beberapa rincian fungsi dapat diabaikan, misalnya banyak kemungkinan polinomial dapat dikelompokkan bersama sebagai kelas.

2. METODE PENELITIAN

Dalam teori komputasi, ada beberapa kelas kompleksitas penting yang didefinisikan dengan membatasi waktu (*Time Complexity Class*) atau ruang (*Space Complexity Class*) yang digunakan oleh algoritma, serta kelas kompleksitas dari algoritma acak yang disebut (*Randomized Complexity Class*). Kelas-kelas kompleksitas tersebut dapat dijelaskan sebagai berikut.

2.1 *Time Complexity Class*

Dalam ilmu komputer, kompleksitas waktu dari sebuah algoritma mengkuantifikasi jumlah waktu yang dibutuhkan oleh algoritma untuk dijalankan sebagai fungsi dari panjang string yang mewakili input (Ladner, 1975). Kompleksitas waktu dari sebuah algoritma biasanya dinyatakan dengan menggunakan notasi O besar, yang mengecualikan koefisien dan urutan pesanan yang lebih rendah. Bila dinyatakan dengan cara ini, kompleksitas waktu dikatakan digambarkan asimtotik , yaitu, karena ukuran masukan tidak terbatas. Misalnya, jika waktu yang dibutuhkan oleh algoritma pada semua input dengan ukuran n paling banyak $5n^3 + 3n$ untuk n (lebih besar dari beberapa n_0), kompleksitas waktu asimtotik adalah O (n^3).

2.1.1 Kelas Waktu Polinomial (P)

Algoritma dikatakan sebagai waktu polinomial jika waktu operasinya berada di atas dibatasi oleh ekspresi polinomial dalam ukuran input untuk algoritma, yaitu $T(n) = O(n^k)$ untuk beberapa *k* konstan (Sipser, 2006). Masalah dimana algoritma waktu polinomial deterministik ada tergolong kelas kompleksitas P, yang merupakan pusat dalam bidang teori kompleksitas komputasi.

Beberapa contoh algoritma waktu polinomial:

- Algoritma pengurutan sortir seleksi pada bilangan bulat *n* dilakukan An^2. Operasi untuk beberapa konstanta *A*. Jadi berjalan pada waktunya $O(n^2)$. Dan merupakan algoritma waktu polinomial.
- Semua operasi aritmatika dasar dapat dilakukan dalam waktu polinomial.
- Pencocokan maksimum dalam grafik dapat ditemukan dalam waktu polinomial.

Dalam teori kompleksitas komputasio-nal, P juga dikenal sebagai PTIME atau DTIME ($n^{O(1)}$), adalah kelas kompleksitas mendasar. Ini berisi semua masalah keputusan yang dapat dipecahkan oleh mesin Turing deterministik dengan menggunakan jumlah waktu komputasi yang polinomial, atau waktu polinomial.

Bahasa *L* ada di P jika dan hanya jika ada mesin Turing deterministik *M* , sedemikian hingga:

- *M* berjalan untuk waktu polinom pada semua input.
- Untuk semua *x* di *L* , *M* output 1
- Untuk semua *x* tidak di *L* , *M* output 0.

2.1.2 Kelas Kompleksitas NP

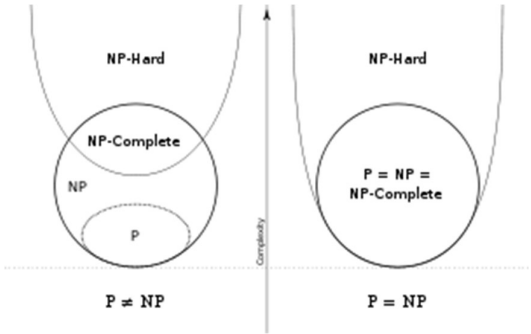
NP (*Nondeterministic Polynomial time*) adalah kelas kompleksitas yang digunakan untuk menggambarkan beberapa jenis masalah keputusan. Secara informal, NP adalah kumpulan semua masalah keputusan yang mana jawabannya adalah "ya" memiliki bukti yang dapat diverifikasi dengan efisien. Lebih tepatnya, bukti ini harus dapat diverifikasi dengan perhitungan deterministik yang dapat dilakukan pada waktu polinomial. Secara ekivalen, definisi formal NP adalah seperangkat masalah keputusan yang dapat dipecahkan dalam waktu polinomial oleh mesin Turing non deterministik teoritis.

NP kelas kompleksitas dapat didefinisikan dalam istilah NTIME :

$$NP = \bigcup_{k \in \mathbb{N}} NTIME(n^k)$$

Dimana $NTIME(n^k)$ Adalah seperangkat masalah keputusan yang dapat diatasi dengan mesin Turing non deterministik di $O(n^k)$ waktu.

Masalah yang tidak terpecahkan dalam ilmu komputer : Apakah P = NP ? Kompleksitas kelas P terkandung dalam NP, namun NP mengandung banyak masalah penting, yang paling sulit disebut masalah NP-complete, yang solusinya cukup untuk menangani masalah NP lainnya pada waktu polinomial. Pertanyaan terbuka seperti di atas dalam teori kompleksitas, masalah P versus NP ("P = NP"), menanyakan apakah algoritma waktu polinomial benar-benar ada untuk memecahkan NP-complete, dan secara wajar, semua masalah ini adalah NP. Karena dipercaya secara luas bahwa P=NP tidak terjadi. Namun jika P = NP = NP-complete, maka selain itu adalah masalah NP-Hard. Diagram Euler untuk hal ini dapat dilihat pada Gambar 1 berikut.



Gambar 1. Diagram Euler untuk P, NP, NP-complete, dan NP-hard [2]

Kelas kompleksitas NP (yang memiliki bukti yang dapat diverifikasi secara efisien dimana jawabannya "ya") juga terkait dengan kompleksitas kelas co-NP (yang memiliki bukti yang dapat diverifikasi dengan efisien dimana jawabannya "tidak"). Apakah atau tidak NP = co-NP adalah pertanyaan lain dalam teori kompleksitas.

2.1.3 Kelas Kompleksitas co-NP

Secara sederhana, co-NP adalah kelas masalah dimana ada algoritma waktu polinomial yang dapat memverifikasi tidak ada *instance*. Secara ekivalen, co-NP adalah seperangkat masalah keputusan dimana *instance* "tidak" dapat diterima dalam waktu polinomial oleh mesin Turing non-deterministik.

P, kelas masalah polinomial yang dapat dipecahkan, adalah subset dari NP dan co-NP . P dianggap sebagai subset ketat dalam kedua kasus tersebut (dan terbukti tidak ketat dalam satu kasus dan tidak ketat di kasus lainnya). NP dan co-NP juga dianggap tidak setara (Hopcroft, 2000). Jika demikian, maka tidak ada masalah NP-complete yang dapat berada di co-NP dan tidak ada masalah co-NP-complete yang dapat terjadi di NP (Goldreich, 2010).

2.1.4 Kelas Kompleksitas EXPTIME

Kelas kompleksitas EXPTIME (kadang-kadang disebut EXP atau DEXPTIME) adalah kumpulan semua masalah keputusan yang memiliki runtime eksponensial, yaitu dapat dipecahkan oleh mesin Turing deterministik dalam waktu $O(2^{p(n)})$, dimana $P(n)$ adalah fungsi polinomial n.

$$EXPTIME = \bigcup_{k \in \mathbb{N}} DTIME\left(2^{n^k}\right)$$

Dari Gambar 1 kita tahu bahwa:

P ⊆ NP ⊆ PSCAPE ⊆ EXPTIME ⊆ NEXPTIME ⊆ EXPSPACE

pada saat teorema hirarki waktu dan teorema hirarki ruang ditentukan

P ⊊ EXPTIME, NP ⊊ NEXPTIME, dan PSPACE ⊊ EXPSPACE

Jadi setidaknya satu dari tiga inklusi pertama dan setidaknya satu dari tiga inklusi terakhir pasti tepat. Jika P = NP , maka EXPTIME = NEXPTIME , kelas masalah dapat dipecahkan dalam waktu eksponensial oleh mesin Turing nondeterministik. Lebih tepatnya, EXPTIME ≠ NEXPTIME jika dan hanya jika ada bahasa *sparse* di NP yang tidak ada di P.

2.1.5 Kelas Kompleksitas FP, FNP, TFNP

Kompleksitas kelas FP adalah himpunan masalah fungsi yang dapat dipecahkan dengan mesin Turing deterministik pada waktu polinomial. FP secara formal didefinisikan sebagai "Hubungan biner P(x, y) ada dalam FP jika dan hanya jika ada algoritma waktu polinomial deterministik diberikan nilai x, dapat menemukan beberapa y sehingga P (x, y) berlaku". Perbedaan antara FP dan P adalah bahwa masalah pada P memiliki jawaban satu bit, ya / tidak, sedangkan masalah pada FP dapat memiliki keluaran yang dapat dihitung dalam waktu polinomial.

Sama seperti P dan FP terkait erat, NP terkait erat dengan FNP. FNP merupakan fungsi pemecahan masalah dari keputusan kelas masalah NP. Definisi formalnya adalah "Hubungan biner P (x, y), dimana y paling banyak polinomialnya lebih lama dari x , ada

dalam FNP jika dan hanya jika ada algoritma waktu polinomial deterministik yang dapat menentukan apakah P (*x*, *y*) berlaku diberikan *x* dan *y* (Hopcroft, 2000).

TFNP adalah subkelas dari FNP dimana sebuah solusi dijamin ada. Total Function Nondeterministic Polynomial (TFNP) definisi formalnya adalah "Hubungan biner P (*x*, *y*) ada di TFNP jika dan hanya jika ada algoritma waktu polinomial deterministik yang dapat menentukan apakah P (*x*, *y*) berlaku baik *x* dan *y*, dan untuk setiap *x*, ada *y* yang paling banyak polinomialnya lebih panjang dari *x* sehingga P (*x*, *y*) berlaku". Tugas dari algoritma TFNP adalah menetapkan, ditentukan sebuah *x* memberi satu kemungkinan nilai untuk *y* sehingga P (*x*, *y*) berlaku. FP juga adalah subkelas dari TFNP. TFNP juga mengandung subclass PLS , PPA , PPAD , dan PPP . TFNP sama dengan F (NP ∩ coNP). Jika TFNP = FP maka P = NP ∩ coNP.

2.2 *Space Complexity Class*

DSPACE atau SPACE adalah sumber komputasi yang menjelaskan sumber daya ruang memori untuk mesin Turing deterministik. Ini mewakili jumlah total ruang memori yang dibutuhkan komputer fisik "normal" untuk memecahkan masalah komputasi tertentu dengan algoritma yang diberikan. Untuk setiap fungsi $f(n)$, ada kelas kompleksitas SPACE ($f(n)$), rangkaian masalah keputusan yang dapat dipecahkan oleh mesin Turing deterministik dengan menggunakan ruang O ($f(n)$).

REG = DSPACE (O(1)), dimana REG adalah kelas bahasa reguler. Sebenarnya, REG = DSPACE (*o*(log log *n*)) diperlukan untuk mengenali bahasa non-reguler) (Alberts, 1985).

Teorema hierarki ruang adalah hasil pemisahan yang menunjukkan bahwa mesin deterministik dan nondeterministik dapat memecahkan lebih banyak masalah dalam ruang (secara asimtotik), tergantung pada kondisi tertentu. Sebagai contoh, mesin Turing deterministik dapat memecahkan lebih banyak masalah keputusan di ruang $n\,log(n)$ daripada di ruang n.

2.2.1 Kelas Kompleksitas PSPACE

PSPACE adalah seperangkat semua masalah keputusan yang dapat dipecahkan oleh mesin Turing dengan menggunakan jumlah ruang polinomial. Jika kita menunjukkan dengan SPACE ($t(n)$), himpunan semua masalah yang dapat dipecahkan oleh mesin Turing menggunakan ruang O ($t(n)$) untuk beberapa fungsi *t* dari ukuran input *n* , maka kita dapat mendefinisikan PSPACE secara formal sebagai,

$$PSPACE = \bigcup_{k\in\mathbb{N}} SPACE(n^k)$$

Dalam teorema Savitch yang memberikan hubungan antara kompleksitas ruang deterministik dan non-deterministik, dinyatakan bahwa untuk fungsi apapun $f \in \Omega(log(n))$ maka,

$$NSPACE(f(n)) \subseteq DSPACE((f(n))^2)$$

Dengan kata lain, jika mesin Turing nondeterministik dapat memecahkan masalah dengan menggunakan ruang $f(n)$, mesin Turing deterministik biasa dapat memecahkan masalah yang sama di kuadrat dari batas ruangnya. Dari teorema Savitch, NPSPACE setara dengan PSPACE, pada dasarnya karena mesin Turing deterministik dapat mensimulasikan mesin Turing yang tidak deterministik tanpa memerlukan lebih banyak ruang (walaupun bisa menggunakan lebih banyak waktu).

Hubungan antara kelas lainnya seperti terlihat pada Gambar 1, dapat dijabarkan lebih lanjut yaitu:

- NL ⊆ P ⊆ NP ⊆ PH ⊆ PSPACE
- PSPACE ⊆ EXPTIME ⊆ EXPSPACE
- NL ⊊ PSPACE ⊊ EXPSPACE
- P ⊊ EXPTIME

Diketahui bahwa pada baris pertama dan kedua, setidaknya satu dari batasan yang ditetapkan harus ketat, namun tidak diketahui mana. Secara luas dicurigai bahwa semuanya ketat. Masalah yang paling sulit di PSPACE adalah PSPACE-complete.

2.2.2 Kelas Kompleksitas PSPACE-complete

Masalah keputusan adalah PSPACE-complete jika dapat dipecahkan dengan menggunakan sejumlah memori yang polinomial pada panjang masukan (ruang polinomial) dan jika setiap masalah lain yang dapat dipecahkan di ruang polinomial dapat ditransformasikan juga ke waktu polinomial. Masalah yang PSPACE-complete dapat dianggap sebagai masalah yang paling sulit di PSPACE , karena solusi untuk masalah apa pun bisa digunakan untuk mengatasi masalah lain di PSPACE .

Masalah PSPACE-complete secara luas diduga berada di luar kelas kompleksitas P dan NP. Diketahui bahwa mereka berada di luar kelas NC (kelas masalah dengan algoritma paralel yang sangat efisien), karena masalah di NC dapat dipecahkan dalam sejumlah polinomial ruang dalam logaritma ukuran masukan, dan kelas masalah dapat dipecahkan dalam sejumlah kecil ruang secara ketat terdapat di PSPACE oleh teorema hirarki ruang.

2.2.3 Kelas Kompleksitas NC

Kelas NC (singkatan "Kelas Nick") adalah kumpulan masalah keputusan yang dapat dipecahkan dalam waktu polikogaritm pada komputer paralel dengan jumlah prosesor polinomial. Dengan kata lain, ada masalah di NC jika ada konstanta *c* dan *k* sehingga dapat diselesaikan pada waktu O ($\log^c n$) dengan menggunakan prosesor paralel O (n^k).
Masalah yang tidak terpecahkan dalam ilmu komputer : Apakah NC = P ?

Sama seperti kelas P dapat dianggap sebagai masalah yang dapat ditundukkan (menurut tesis Cobham), maka NC dapat dianggap sebagai masalah yang dapat diselesaikan secara efisien pada komputer paralel. NC adalah subset dari P karena perhitungan paralel poliamogaritm dapat disimulasikan oleh polinomial-time sequential. Tidak diketahui apakah NC = P , namun kebanyakan peneliti menduga ini salah, berarti mungkin ada beberapa masalah penurut yang "inheren berurutan" dan tidak dapat dipercepat secara signifikan dengan menggunakan paralelisme. Sama seperti kelas NP-complete dapat dianggap sebagai "mungkin sulit diatasi", jadi kelas P-complete , saat menggunakan pengurangan NC , dapat dianggap sebagai "mungkin tidak dapat diparalelkan" atau "mungkin secara inheren berurutan".

2.2.4 Kelas Kompleksitas EXPSPACE

EXPSPACE adalah kumpulan semua masalah keputusan yang dapat dipecahkan oleh mesin Turing deterministik di ruang O ($2^{p(n)}$), dimana $p(n)$ adalah fungsi polinomial n. Jika kita menggunakan mesin nondeterministik, kita akan mendapatkan kelas NEXPSPACE , yang setara dengan EXPSPACE oleh teorema Savitch.

$$EXPSPACE = \bigcup_{k\in\mathbb{N}} DSPACE\left(2^{n^k}\right) = \bigcup_{k\in\mathbb{N}} NSPACE\left(2^{n^k}\right)$$

Masalah keputusan adalah EXPSPACE-complete jika ada di EXPSPACE , dan setiap masalah di EXPSPACE memiliki pengurangan banyak waktu untuk banyak hal. Dengan kata lain, ada algoritma waktu polinomial yang mengubah contoh satu ke contoh yang lain dengan jawaban yang sama. Masalah EXPSPACE-complete mungkin dianggap sebagai masalah tersulit dalam EXPSPACE.

2.3 *Randomized Complexity Class*

Dasar pemikirannya adalah kumpulan komputasi efisien dengan mesin Turing yang berjalan secara *probabilistic polynomial time* (PPT). Efisien dimaksud disini hanya algoritma yang berjalan pada polynomial tetap dalam panjang inputnya (Reshef, 1998). Mesin Turing probabilistik adalah mesin Turing non deterministik yang memilih antara transisi yang tersedia pada setiap titik sesuai dengan beberapa distribusi probabilitas. Sebagai konsekuensinya, mesin Turing probabilistik dapat (tidak seperti Turing Machine deterministik) memiliki hasil stokastik. Pada mesin input dan instruksi yang diberikan, mungkin ada waktu run yang berbeda, atau mungkin tidak berhenti sama sekali. Selanjutnya, ia mungkin menerima masukan dalam satu eksekusi dan menolak input yang sama dalam eksekusi lain. Mesin Turing probabilistik merupakan pengembangan notasi dari komputasi yang efisien dengan membolehkan algoritma (mesin Turing) untuk "melempar koin".

Berbagai kelas kompleksitas acak polinomial yang dihasilkan dari berbagai definisi penerimaan meliputi RP, co-RP, BPP dan ZPP. Jika mesin dibatasi pada ruang logaritmik, bukan waktu polinomial, kelas kompleks RL , co-RL, BPL , dan ZPL diperoleh. Dengan menerapkan kedua batasan tersebut, RLP , co-RLP, BPLP , dan ZPLP dihasilkan. Definisi natural suatu mesin Turing dapat berjalan dalam waktu polinomial (*polynomial running time*) menghasilkan kelas-kelas kompleksitas acak atau random (*Randomized Complexity Class*).

Ada dua cara untuk mendefinisikan *randomized computation*:

- Online; untuk masuk langkah-langkah randomized,
- Offline; untuk menggunakan tambahan input perandoman dan mengevaluasi outputnya pada input yang random.

2.3.1 RP (*Randomized Polynomial time*)

RP adalah kelas kompleksitas masalah dimana mesin Turing probabilistik ada dengan properti:

- Selalu berjalan dalam waktu polinomial dalam ukuran input.
- Jika jawaban yang benar adalah TIDAK, itu selalu mengembalikan NO.
- Jika jawaban yang benar adalah YA, maka akan mengembalikan YA dengan probabilitas setidaknya 1/2 (jika tidak, itu akan mengembalikan TIDAK).

Algoritma dapat mengembalikan YA adalah jika jawaban sebenarnya adalah YA. Oleh karena itu jika algoritma tersebut berakhir dan menghasilkan YA, maka jawaban yang benar pasti YA. Namun, algoritma tersebut dapat berakhir dengan NO terlepas dari jawaban sebenarnya. Artinya, jika algoritma mengembalikan NO, itu mungkin salah. Jika jawaban yang benar adalah YA dan algoritma dijalankan n kali dengan hasil masing-masing dijalankan secara statistik independen dari yang lain, maka akan mengembalikan YA setidaknya satu kali dengan probabilitas paling sedikit $1 - 2^{-n}$. Jadi jika algoritma dijalankan 100 kali, maka kemungkinan itu memberikan jawaban yang salah setiap saat lebih rendah dari pada kemungkinan adanya peristiwa rusaknya memori komputer yang menjalankan algoritma.

Class kompleksitas RP adalah suatu class dari semua bahasa L dimana terdapat sebuah mesin turing M yang PPT, sedemikian hingga

$$x \in L \Rightarrow Prob[M(x) = 1] \geq \frac{1}{2}$$
$$x \notin L \Rightarrow Prob[M(x) = 1] = 0$$

co-RP (*Complementary Random Polynomial-time*) adalah suatu class dari semua bahasa L dimana terdapat sebuah mesin turing M yang PPT, sedemikian hingga

$$x \in L \Rightarrow Prob[M(x) = 1] = 1$$
$$x \notin L \Rightarrow Prob[M(x) = 0] \geq \frac{1}{2}$$

RP dari definisinya tidak meminta kelakuan yang sama pada input-input yang ada di bahasanya. Jika $x \notin L$ maka jawaban mesin harus benar walaupun tebakannya berbeda.

Namun jika $x \in L$, mesin boleh melakukan kesalahan. Dalam kasus ini, ada probabilitas tidak-nol dimana jawaban mesinnya akan menjadi salah (dan probabilitasnya "tidak terlalu besar"). Ini disebut definisi untuk satu tipe kesalahan (*one-sided error*). Error satu-sisi penting dalam membangun NP, karena verifikasi satu-sisi alami, namun kurang berguna dalam mengeksporasi notasi dari komputasi yang efisien.

Algoritma RP dan co-RP Lebih jelasnya dapat dilihat pada Tabel 1 berikut.

Tabel 1. Algoritma RP (1 run dan *n* run)

Algoritma RP (1 run)		
Jawaban yang Dihasilkan / Jawaban Benar	Ya	Tidak
Ya	≥ 1/2	≤ 1/2
Tidak	0	1
Algoritma RP (n run)		
Jawaban yang Dihasilkan / Jawaban Benar	Ya	Tidak
Ya	$\geq 1 - 2^{-n}$	$\leq 2^{-n}$
Tidak	0	1
Algoritma co-RP (1 run)		
Jawaban yang Dihasilkan / Jawaban Benar	Ya	Tidak
Ya	1	0
Tidak	≤1/2	≥ 1/2

2.3.2 Kelas Kompleksitas BPP

BPP (*Bounded Probability Polynomial-time*) adalah kelas masalah keputusan yang dapat dipecahkan oleh mesin Turing probabilistik pada waktu polinomial dengan probabilitas kesalahan dibatasi lebih dari 1/2 untuk semua kejadian. BPP adalah kelas dari semua bahasa L dimana terdapat mesin Turing M yang berjalan secara probabilitas dalam waktu polinomial sedemikian hingga:

$$\forall x : Prob[M(x) = \chi_L(x)] \geq \frac{2}{3}$$

Ini berarti

$$x \in L \Rightarrow Prob[M(x) = 1] \geq \frac{2}{3}$$
$$x \notin L \Rightarrow Prob[M(x) = 1] < \frac{1}{3}$$

Untuk setiap ambang komputabel polynomial-time, notasi f , dan margin $\frac{1}{poly}$, didapatkan kembali ambang ”standar” (dari 1/2) dan margin "aman" 1/6. Buktinya:

$$x \in L \Rightarrow Prob[M(x) = 1] \geq \frac{1}{2} + \frac{1}{6} = \frac{2}{3}$$
$$x \notin L \Rightarrow Prob[M(x) = 1] < \frac{1}{2} - \frac{1}{6} = \frac{1}{3}$$

Maka dapat dikatakan,

$$x \in L \Rightarrow Prob[M(x) = 1] \geq p + \epsilon$$
$$x \notin L \Rightarrow Prob[M(x) = 1] < p - \epsilon$$

Jika terdapat fungsi komputable polynomial-time $f : \mathbb{N} \mapsto [0, 1]$, polinomial positif p(.) dan mesin Turing M yang berjalan probabilistik dalam waktu polinomial maka,

$$\forall x \in L : Prob[M(x) = 1] \geq f(|x|) + \frac{1}{p(|x|)}$$
$$\forall x \notin L : Prob[M(x) = 1] < f(|x|) - \frac{1}{p(|x|)}$$

Dengan memilih $f(|x|) \equiv \frac{1}{2}$ dan $p(|x|) \equiv 6$ maka definisi BPP diatas terpenuhi.

Untuk setiap L ∈ BPP dan setiap polinomial positif p(.) terdapat mesin Turing M yang berjalan probabilistik dalam waktu polinomial maka,

$$\forall x : Prob[M(x) = \chi_L(x)] \geq 1 - 2^{\{-p(|x|)\}}$$

Jika kondisi ini true untuk setiap polinomial, maka pilih $p(|x|) \equiv 2$ sehingga,

$$\forall x : Prob[M(x) = \chi_L(x)] \geq 1 - 2^{-2} = \frac{3}{4} \geq \frac{2}{3}$$

Dari penjelasan diatas, dapat dibuat tabel algoritma BPP seperti terlihat pada Tabel 2.

2.3.3 Kelas Kompleksitas PP (*Probabilistic Polynomial-time*)

PP adalah kelas masalah keputusan yang dapat dipecahkan oleh mesin Turing probabilistik pada waktu polinomial , dengan probabilitas kesalahan kurang dari 1/2 untuk semua kasus. Jika ada mesin yang menjawab benar dengan probabilitas lebih dari 1/2, dan ingin mendapat mesin lain yang menjawab benar juga dengan probabilitas lebih dari 1 + ε (untuk $0 < \varepsilon < \frac{1}{2}$), ini tidak dapat dilakukan dalam polynomial time karena tidak mungkin ada gapnya. Definisi PP sebagai berikut:

$PP \stackrel{\text{def}}{=} \{L \subseteq \{0,1\}*\}$ terdapat mesin Turing M yang berjalan dalam waktu polinomial sedemikian hingga

$$\forall x : Prob[M(x) = \chi_L(x)] > \frac{1}{2}$$

Tabel 2. Algoritma BPP (1 run dan *p* run)

Algoritma BPP (1 run)		
Jawaban yang Dihasilkan / Jawaban Benar	Ya	Tidak
Ya	≥ 2/3	≤ 1/3
Tidak	≤ 1/3	≥ 2/3
Algoritma BPP (p run)		
Jawaban yang Dihasilkan / Jawaban Benar	Ya	Tidak
Ya	$> 1 - 2^{\{-p(\|x\|)\}}$	$< 2^{\{-p(\|x\|)\}}$
Tidak	$< 2^{\{-p(\|x\|)\}}$	$> 1 - 2^{\{-p(\|x\|)\}}$

2.3.4 Kelas Kompleksitas ZPP (*Zero-error Probabilistic Polynomial-time*)

ZPP adalah kelas kompleksitas masalah dimana mesin Turing probabilistik ada dengan sifat-sifat selalu mengembalikan jawaban YA atau TIDAK yang benar, dan waktu yang berjalan adalah polinomial dengan harapan setiap masukan. Sebagai alternatif, ZPP dapat didefinisikan sebagai kelas masalah dimana mesin Turing probabilistik ada dengan properti ini:

- Selalu berjalan dalam waktu polinomial.
- Ini mengembalikan sebuah jawaban YA, TIDAK atau TIDAK TAHU.
- Jawabannya selalu TIDAK TAHU atau jawaban yang benar.
- Ini mengembalikan TIDAK TAHU dengan probabilitas paling banyak 1/2 (dan jawaban yang benar sebaliknya).

TIDAK TAHU ditandai dengan notasi ⊥.

Jika ada suatu bahasa L yang merupakan anggota ZPP, maka terdapat mesin Turing M yang berjalan secara probabilistik dalam waktu polinomial sedemikian hingga,

$$\forall x : Prob[M(x) = \perp] \leq \frac{1}{2}$$
$$\forall x : Prob[M(x) = \chi_L(x) \; atau \; M(x) = \perp] = 1$$

Nilai 1/2 ini sembarang dan dapat diganti dengan nilai antara $2^{-p(|x|)}$ hingga $1 - \frac{1}{p(|x|)}$. Sehingga dapat dibuktikan bahwa ZPP = RP ∩ coRP. Kelas P terkandung dalam ZPP, namun pertanyaannya adalah:

Apakah P = ZPP?

Bukti untuk ZPP = EXPTIME akan menyiratkan bahwa P ≠ ZPP, seperti P ≠ EXPTIME.

3. HASIL DAN PEMBAHASAN

Dalam teori kompleksitas komputasional, kelas kompleksitas adalah serangkaian masalah kompleksitas berbasis sumber daya terkait. Kelas kompleksitas khas memiliki definisi bentuk: "Kumpulan masalah yang dapat diatasi dengan mesin abstrak M menggunakan O ($f(n)$) sumber daya R, di mana n adalah ukuran input".

Tabel 3. Kelas Kompleksitas Waktu (*Time complexity class*)

Kompleksitas Kelas	**Model Perhitungan**	**Batasan Sumber Daya**
Waktu Deterministik		
DTIME (f(n))	TM Deterministik	Waktu f(n)
P (Polinomial)	TM Deterministik	Waktu poli (n)
EXPTIME	TM Deterministik	Waktu $2^{poli(n)}$
Waktu Non-Deterministik		
NTIME (f(n))	TM Non-Deterministik	Waktu f(n)
NP	TM Non-Deterministik	Waktu poli (n)
NEXPTIME	TM Non-Deterministik	Waktu $2^{poli(n)}$

Tabel 4. Kelas Kompleksitas Ruang (*Space complexity class*)

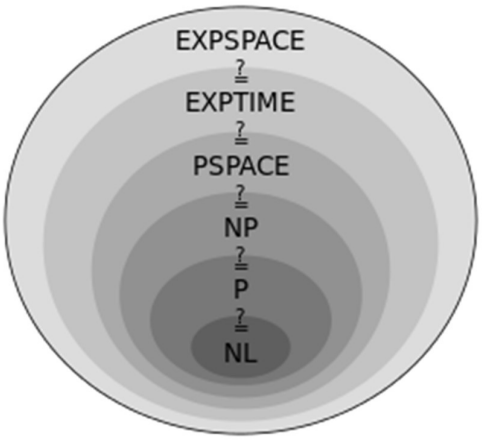
Kompleksitas Kelas	**Model Perhitungan**	**Batasan Sumber Daya**
Waktu Deterministik		
DSPACE (f(n))	TM Deterministik	Ruang f(n)
L (Logaritmik)	TM Deterministik	Ruang O (log n)
PSPACE	TM Deterministik	Ruang poli(n)
EXPSPACE	TM Deterministik	Ruang $2^{poli(n)}$
Waktu Non-Deterministik		
NSPACE (f(n))	TM Non-Deterministik	Ruang f(n)
NL	TM Non-Deterministik	Ruang O (log n)
NSPACE	TM Non-Deterministik	Ruang poli(n)
NEXPSPACE	TM Non-Deterministik	Ruang $2^{poli(n)}$

Kelas kompleksitas yang paling sederhana didefinisikan oleh jenis masalah komputasi, model perhitungan, dan sumber daya (atau sumber daya) yang dibatasi dan dibatasi. Sumber daya dan batas biasanya dinyatakan bersama-sama, seperti "waktu

polinomial", "ruang logaritmik", "kedalaman konstan", dan lain-lain.

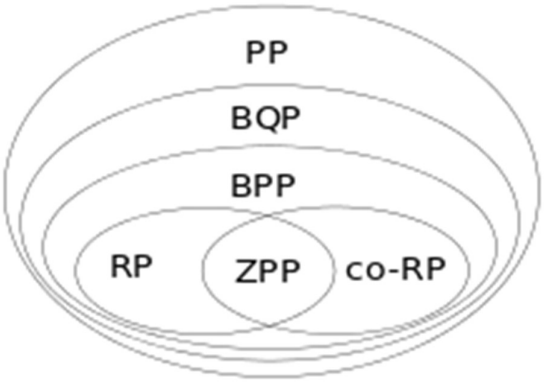
Banyak kelas kompleksitas penting dapat didefinisikan dengan membatasi waktu atau ruang yang digunakan oleh algoritma. Beberapa kelas kompleksitas penting dari masalah keputusan yang didefinisikan dengan cara ini seperti terlihat pada Tabel 3 dan 4.

Hubungan antara kompleksitas kelas dapat direpresentasikan pada Gambar 2. Kelas logaritmik (tentu saja) tidak memperhitungkan ruang yang dibutuhkan untuk mewakili masalah. Dari teorema Savitch, ternyata PSPACE = NPSPACE dan EXPSPACE = NEXPSPACE. Kelas kompleksitas penting lainnya termasuk BPP, ZPP dan RP, yang didefinisikan dengan menggunakan mesin Turing probabilistik.



Gambar 2. Representasi hubungan antara kompleksitas kelas

Hubungan antar kelas kompleksitas Randomized dapat dijelaskan seperti pada Gambar 3.



Gambar 3. Hubungan antar kelas Randomized

Diketahui bahwa BPP hampir sama seperti komplemennya, artinya BPP = co-BPP. BPP rendah untuk dirinya sendiri, yang berarti bahwa mesin BPP dengan kekuatan untuk menyelesaikan masalah BPP seketika (mesin oracle BPP) tidak lagi lebih kuat daripada mesin tanpa tenaga ekstra ini. Dalam simbol, BPP^{BPP} = BPP. Jika akses keacakan dihapus dari definisi BPP, kita mendapatkan kelas kompleksitas P. Dalam definisi kelas, jika kita mengganti mesin Turing biasa dengan komputer kuantum, kita mendapatkan kelas BQP.

4. KESIMPULAN

Beberapa kelas kompleksitas penting didefinisikan dengan membatasi waktu atau ruang yang digunakan oleh algoritma, serta ada beberapa kelas kompleksitas dari algoritma acak. Kelas-kelas kompleksitas yang didefinisikan dengan membatasi waktu disebut *Time Complexity Class*, sedangkan dengan membatasi ruang (memori) disebut *Space Complexity Class*. Apabila mesin Turing yang digunakan probabilistik maka kelas kompleksitasnya disebut *Randomized Complexity Class*. Semua kelas dapat berjalan dalam waktu polinomial.

5. DAFTAR PUSTAKA

Alberts, Maris (1985). Space complexity of alternating Turing machines.

Goldreich, Oded (2010). P, NP, and NP-completeness: The Basics of Computational Complexity. Cambridge University Press. p. 155. ISBN 9781139490092.

Hopcroft, John E. (2000). Introduction to Automata Theory, Languages, and Computation (2nd Edition). Boston: Addison-Wesley. ISBN 0-201-44124-1.

R. E. Ladner (1975). On the structure of polynomial time reducibility. J.ACM, 22, pp. 151–171. Corollary.

Reshef, Eilon (1998). Introduction to Complexity Teory (Lecture Notes).

Sipser, Michael (2006). Introduction to the Theory of Computation. Course Technology Inc. ISBN 0-619-21764-2.