Pencarian Substruktur Molekul Berbasis Graf Menggunakan Backtracking dan Pruning

Raymond Jonathan Dwi Putra Julianto - 13524059
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
Email: annayuliati69@gmail.com , 13524059@std.stei.itb.ac.id

***Abstract*—Pencarian substruktur molekul merupakan salah satu permasalahan penting dalam cheminformatics karena memungkinkan pencarian senyawa berdasarkan pola struktur kimia tertentu, bukan hanya berdasarkan nama atau rumus molekul. Pada makalah ini, molekul direpresentasikan sebagai graf berlabel, dengan atom sebagai simpul dan ikatan kimia sebagai sisi. Permasalahan pencarian substruktur kemudian dimodelkan sebagai pencocokan graf kecil berupa pattern terhadap graf molekul target. Metode utama yang digunakan adalah algoritma runut-balik atau backtracking, yaitu dengan membangun pemetaan kandidat antara atom pada pattern dan atom pada molekul target secara bertahap. Untuk meningkatkan efisiensi pencarian, diterapkan pruning sederhana berdasarkan kesesuaian label atom, kecukupan derajat simpul, validasi ikatan parsial, dan kesesuaian jenis ikatan. Hasil implementasi menunjukkan bahwa backtracking dapat digunakan untuk menyelesaikan pencarian substruktur molekul, sementara pruning membantu mengurangi ruang pencarian tanpa mengubah prinsip utama algoritma.**

***Index Terms*—Backtracking, Pruning, Graf Molekul, Substructure Search, Subgraph Matching.**

I. Pendahuluan

Perkembangan teknologi informasi telah memberikan pengaruh besar terhadap berbagai bidang ilmu, termasuk kimia komputasional dan cheminformatics. Salah satu persoalan yang sering muncul dalam pengolahan data kimia adalah bagaimana mencari senyawa yang memiliki pola struktur tertentu di dalam suatu basis data molekul. Pencarian tersebut tidak cukup dilakukan hanya berdasarkan nama senyawa atau rumus molekul, karena dua molekul dapat memiliki rumus molekul yang sama tetapi susunan atom dan ikatan yang berbeda.

Sebagai contoh, rumus molekul `C2H6O` dapat merepresentasikan ethanol maupun dimethyl ether. Keduanya memiliki jumlah atom yang sama, tetapi struktur ikatan antaratomnya berbeda. Oleh karena itu, pencarian senyawa perlu memperhatikan struktur molekul secara langsung, bukan hanya informasi tekstual seperti nama atau formula.

Salah satu cara yang dapat digunakan untuk merepresentasikan struktur molekul adalah dengan memodelkannya sebagai graf. Dalam representasi ini, atom dipandang sebagai simpul, sedangkan ikatan kimia antaratom dipandang sebagai sisi. Setiap simpul memiliki label berupa jenis atom, seperti C, O, N, atau unsur lainnya, sedangkan setiap sisi memiliki label berupa jenis ikatan, seperti ikatan tunggal, rangkap dua, rangkap tiga, aromatik, atau koordinasi.

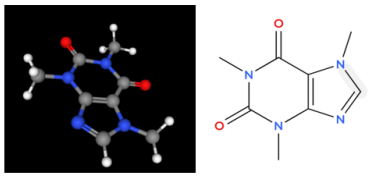

Fig. 1. Contoh struktur molekul dalam representasi dua atau tiga dimensi.

Dengan representasi tersebut, permasalahan pencarian substruktur molekul dapat dipandang sebagai permasalahan pencocokan graf, yaitu mencari apakah sebuah graf kecil sebagai pattern dapat dipetakan ke sebagian graf molekul target. Permasalahan ini sesuai untuk dianalisis menggunakan algoritma runut-balik karena algoritma perlu mencoba berbagai kemungkinan pemetaan antara atom pada pattern dan atom pada molekul target.

Makalah ini bertujuan untuk menjelaskan penerapan algoritma backtracking pada pencarian substruktur molekul berbasis graf, membandingkan pure backtracking dengan backtracking yang menggunakan pruning sederhana, serta menunjukkan bagaimana program menampilkan hasil pencarian berupa molekul yang cocok, mapping atom, dan statistik performa algoritma.

II. Dasar Teori

A. Graf

Graf merupakan model matematis untuk menyatakan objek-objek diskrit beserta hubungan di antara objek tersebut. Sebuah graf dapat dinyatakan sebagai $G = (V, E)$, dengan V sebagai himpunan simpul dan E sebagai himpunan sisi. Dalam pemodelan, simpul digunakan untuk merepresentasikan objek, sedangkan sisi digunakan untuk merepresentasikan keterhubungan atau relasi antarobjek.

Pembagian jenis graf dapat dilakukan berdasarkan beberapa sudut pandang. Berdasarkan arah sisinya, graf dibedakan menjadi graf tidak berarah dan graf berarah. Pada graf tidak berarah, hubungan antara dua simpul tidak memiliki orientasi tertentu. Sebaliknya, pada graf berarah, setiap sisi memiliki arah dari satu simpul ke simpul lain.

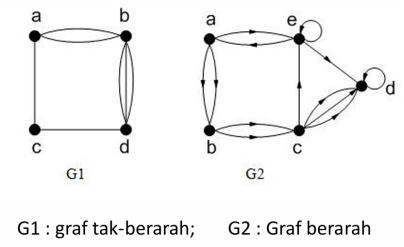

Fig. 2. Pembagian graf berdasarkan arah sisi.

Berdasarkan ada atau tidaknya sisi ganda dan gelang, graf dapat dibedakan menjadi graf sederhana dan graf tak-sederhana. Graf sederhana adalah graf yang tidak memiliki sisi ganda maupun gelang. Dengan demikian, setiap pasangan simpul pada graf sederhana dihubungkan oleh paling banyak satu sisi, dan tidak ada sisi yang berawal serta berakhir pada simpul yang sama.

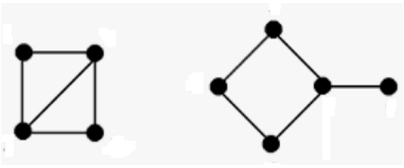

Fig. 3. Contoh graf sederhana.

Graf tak-sederhana adalah graf yang dapat memiliki sisi ganda atau gelang. Sisi ganda muncul ketika terdapat lebih dari satu sisi yang menghubungkan pasangan simpul yang sama, sedangkan gelang adalah sisi yang menghubungkan sebuah simpul dengan dirinya sendiri.

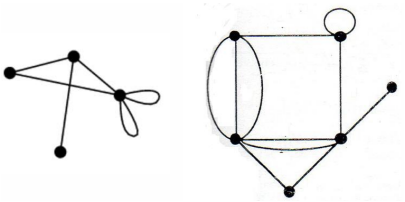

Fig. 4. Contoh graf tak-sederhana.

Beberapa terminologi dasar pada graf penting digunakan dalam analisis struktur. Dua simpul disebut bertetangga apabila keduanya dihubungkan oleh sebuah sisi. Sebuah sisi dikatakan bersisian dengan simpul jika sisi tersebut memiliki simpul tersebut sebagai salah satu ujungnya. Derajat suatu simpul adalah jumlah sisi yang bersisian dengan simpul tersebut. Pada graf berarah, derajat dapat dibedakan menjadi derajat masuk dan derajat keluar.

Selain itu, graf juga memiliki konsep lintasan dan sirkuit. Lintasan adalah barisan simpul dan sisi yang menghubungkan suatu simpul awal ke simpul akhir. Sirkuit adalah lintasan yang berawal dan berakhir pada simpul yang sama. Konsep lintasan dan sirkuit menjadi dasar bagi berbagai persoalan graf, termasuk pencarian rute, keterhubungan, serta pencocokan pola pada graf.

B. Struktur Kimia dan Molekul

Molekul adalah gabungan dua atom atau lebih yang terikat melalui ikatan kimia. Atom berperan sebagai penyusun dasar molekul, sedangkan ikatan kimia menunjukkan adanya hubungan langsung antara atom-atom tersebut. Struktur suatu molekul tidak hanya ditentukan oleh jenis dan jumlah atom yang menyusunnya, tetapi juga oleh susunan keterhubungan antaratom.

Struktur kimia menggambarkan bagaimana atom-atom dalam molekul saling terhubung. Pada banyak senyawa, atom-atom dapat membentuk rantai lurus, rantai bercabang, cincin, atau kombinasi dari beberapa bentuk tersebut. Rantai lurus terbentuk ketika atom-atom tersusun secara berurutan. Rantai bercabang terbentuk ketika suatu atom menjadi titik percabangan bagi beberapa atom lain. Struktur cincin terbentuk ketika rangkaian atom kembali terhubung ke atom awal sehingga membentuk lintasan tertutup.

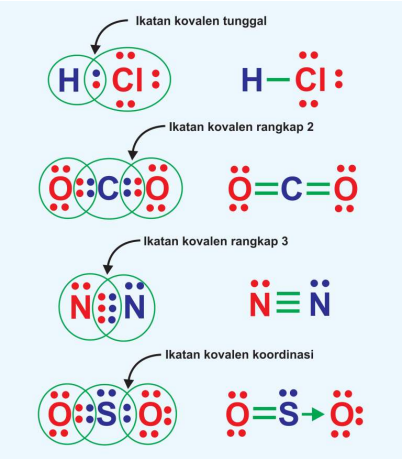

Fig. 5. Ilustrasi ikatan kovalen pada struktur kimia.

Jenis ikatan antaratom juga menjadi bagian penting dari struktur kimia. Ikatan tunggal, rangkap dua, rangkap tiga, aromatik, dan koordinasi menyatakan bentuk hubungan yang berbeda antara dua atom. Perbedaan jenis ikatan dapat memengaruhi sifat molekul dan menentukan pola struktur yang muncul pada senyawa tersebut.

Rumus molekul hanya menyatakan jenis dan jumlah atom dalam suatu molekul. Informasi tersebut belum cukup untuk menjelaskan susunan atom dan ikatan di dalam molekul. Dua senyawa dapat memiliki rumus molekul yang sama, tetapi struktur kimia berbeda karena atom-atomnya terhubung dengan cara yang berbeda. Oleh karena itu, analisis struktur molekul perlu memperhatikan hubungan antaratom dan jenis ikatan, bukan hanya komposisi atomnya.

C. Graf Berlabel

Graf berlabel adalah graf yang setiap simpul, sisi, atau keduanya memiliki informasi tambahan berupa label. Pada graf biasa, struktur utama hanya ditentukan oleh himpunan simpul dan sisi. Pada graf berlabel, setiap simpul atau sisi dapat menyimpan atribut tertentu yang menjelaskan makna

dari elemen graf tersebut. Label ini membuat graf mampu merepresentasikan objek yang lebih kaya daripada sekadar hubungan antar titik.

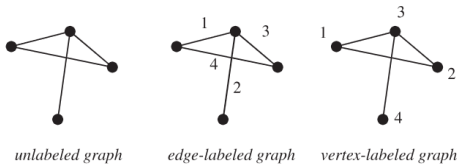

Fig. 6. Contoh graf berlabel.

Label pada simpul dapat digunakan untuk menunjukkan jenis objek yang direpresentasikan oleh simpul tersebut. Label pada sisi dapat digunakan untuk menunjukkan jenis hubungan antara dua simpul. Sebagai contoh, pada graf yang merepresentasikan jaringan jalan, simpul dapat diberi label berupa nama lokasi, sedangkan sisi dapat diberi label berupa jarak atau jenis jalan. Pada konteks struktur kimia, label simpul dapat menyatakan jenis atom, sedangkan label sisi dapat menyatakan jenis ikatan.

Keberadaan label membuat proses pencocokan graf menjadi lebih ketat. Dua simpul tidak cukup hanya memiliki posisi struktural yang sesuai, tetapi juga perlu memiliki label yang cocok. Demikian pula, dua sisi tidak hanya perlu menghubungkan pasangan simpul yang sesuai, tetapi juga perlu memiliki label hubungan yang sesuai. Oleh karena itu, graf berlabel relevan untuk persoalan yang tidak hanya memperhatikan bentuk koneksi, tetapi juga makna dari simpul dan sisi.

D. Subgraph Matching

Subgraph matching adalah persoalan untuk menentukan apakah sebuah graf kecil, yang biasanya disebut graf query atau graf pattern, muncul sebagai bagian dari graf target yang lebih besar. Jika terdapat pemetaan dari simpul-simpul graf query ke simpul-simpul graf target yang mempertahankan hubungan sisi, maka graf query dapat dianggap cocok sebagai subgraf dari graf target.

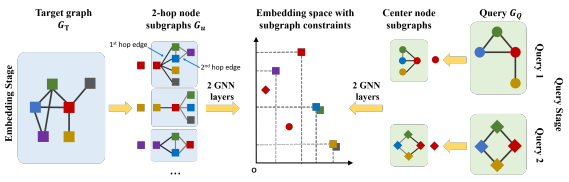

Fig. 7. Ilustrasi pencocokan graf query terhadap graf target.

Pada graf berlabel, proses pencocokan tidak hanya mempertimbangkan keterhubungan antar simpul, tetapi juga label pada simpul dan sisi. Sebuah mapping dianggap valid apabila setiap simpul query dipetakan ke simpul target yang sesuai, setiap sisi query memiliki pasangan sisi pada graf target, dan label yang dibandingkan memenuhi aturan kecocokan. Dengan demikian, subgraph matching dapat digunakan untuk mencari pola struktur tertentu di dalam jaringan atau graf yang lebih besar.

Subgraph matching merupakan persoalan yang sulit secara komputasi karena jumlah kemungkinan pemetaan dapat bertambah sangat cepat seiring bertambahnya jumlah simpul pada graf target dan graf query. Secara umum, masalah ini dikenal sebagai salah satu persoalan NP-complete. Meskipun demikian, subgraph matching tetap penting karena banyak digunakan pada berbagai domain, seperti analisis jaringan, basis data graf, biokimia, dan pencarian pola struktur.

Dalam konteks struktur molekul, graf query dapat dipandang sebagai pola substruktur yang ingin dicari, sedangkan graf target adalah molekul yang diperiksa. Jika pola atom dan ikatan pada graf query dapat dipetakan ke bagian tertentu dari graf target, maka molekul tersebut mengandung substruktur yang dicari.

E. Backtracking

Backtracking atau runut-balik adalah strategi pemecahan masalah yang membangun solusi secara bertahap melalui serangkaian keputusan. Pada setiap tahap, algoritma memilih salah satu kandidat yang mungkin menjadi bagian dari solusi. Jika pilihan tersebut masih memenuhi batasan masalah, pencarian dilanjutkan ke tahap berikutnya. Jika pilihan tersebut menyebabkan solusi tidak mungkin terbentuk, algoritma kembali ke tahap sebelumnya untuk mencoba kandidat lain.

Konsep backtracking dapat dipandang sebagai penelusuran ruang solusi. Ruang solusi berisi seluruh kemungkinan konfigurasi yang dapat dibentuk dari pilihan-pilihan kandidat. Algoritma backtracking biasanya menelusuri ruang solusi secara depth-first, yaitu memperdalam satu cabang pencarian terlebih dahulu sebelum berpindah ke cabang lain. Setiap simpul pada pohon ruang solusi merepresentasikan solusi parsial, sedangkan sisi pada pohon tersebut merepresentasikan penambahan satu keputusan baru.

Backtracking berbeda dari brute force murni karena tidak selalu menunggu sampai solusi lengkap terbentuk. Jika pada suatu tahap solusi parsial sudah melanggar batasan atau tidak mungkin menghasilkan solusi valid, cabang tersebut dapat dihentikan lebih awal. Proses penghentian cabang ini membuat backtracking lebih terarah dibandingkan pemeriksaan seluruh kemungkinan secara buta.

Dalam backtracking, seluruh kemungkinan solusi dapat digambarkan sebagai pohon ruang status. Akar pohon menyatakan keadaan awal ketika belum ada keputusan yang dipilih. Setiap level pohon menyatakan satu tahap keputusan, sedangkan setiap simpul menyatakan solusi parsial. Simpul yang masih mungkin dikembangkan disebut simpul hidup. Simpul hidup yang sedang diperluas disebut simpul-E atau expansion node. Simpul yang tidak lagi dikembangkan, baik karena sudah menjadi solusi maupun karena tidak mungkin menghasilkan solusi, disebut simpul mati.

Secara umum, algoritma backtracking terdiri atas tiga komponen utama, yaitu fungsi pembangkit, fungsi pembatas, dan kondisi solusi. Fungsi pembangkit menentukan kandidat apa saja yang dapat dicoba pada suatu tahap. Fungsi pembatas

menentukan apakah solusi parsial yang terbentuk masih layak untuk dikembangkan. Kondisi solusi menentukan kapan sebuah konfigurasi parsial sudah menjadi solusi lengkap.

```
BACKTRACK(state):
    if IS_SOLUTION(state):
        PROCESS_SOLUTION(state)
        return

    candidates = GENERATE_CANDIDATES(state)

    for each candidate in candidates:
        if IS_FEASIBLE(state, candidate):
            APPLY(state, candidate)
            BACKTRACK(state)
            UNDO(state, candidate)
```

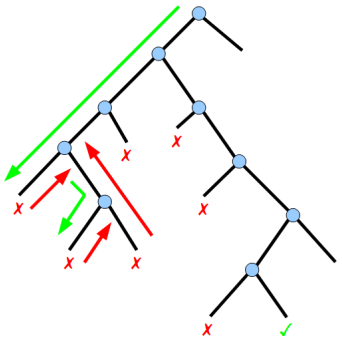

Fig. 8. Ilustrasi alur pencarian pada algoritma backtracking.

Pada pseudocode tersebut, `state` menyatakan solusi parsial yang sedang dibangun. Fungsi `IS_SOLUTION` memeriksa apakah solusi parsial sudah lengkap. Fungsi `GENERATE_CANDIDATES` berperan sebagai fungsi pembangkit yang menghasilkan kandidat pilihan berikutnya. Fungsi `IS_FEASIBLE` berperan sebagai fungsi pembatas yang memeriksa apakah kandidat masih memenuhi batasan. Fungsi `APPLY` menambahkan kandidat ke solusi parsial, sedangkan `UNDO` membatalkan pilihan tersebut agar algoritma dapat mencoba kandidat lain.

F. Pruning

Pruning adalah teknik untuk memangkas cabang pencarian yang tidak mungkin menghasilkan solusi valid. Teknik ini biasanya digunakan bersama algoritma pencarian seperti backtracking untuk mengurangi jumlah kemungkinan yang harus dieksplorasi. Dengan pruning, algoritma tidak perlu menunggu sampai solusi lengkap terbentuk apabila sejak tahap awal sudah terlihat bahwa solusi parsial melanggar batasan masalah.

Pruning bekerja dengan menambahkan fungsi pembatas atau kriteria kelayakan pada proses pencarian. Ketika sebuah kandidat dipilih, kandidat tersebut diperiksa terlebih dahulu terhadap syarat tertentu. Jika kandidat tidak memenuhi syarat, cabang pencarian yang berawal dari kandidat tersebut langsung dihentikan. Sebaliknya, jika kandidat masih memenuhi syarat, algoritma dapat melanjutkan pencarian ke tahap berikutnya.

Tujuan utama pruning adalah mengurangi ruang pencarian tanpa menghilangkan solusi yang benar. Oleh karena itu, aturan pruning harus dirancang secara hati-hati. Aturan yang terlalu lemah tidak banyak mengurangi jumlah kandidat, sedangkan aturan yang terlalu kuat dapat membuang cabang yang sebenarnya masih mungkin menghasilkan solusi. Dalam perancangan algoritma, pruning yang baik adalah pruning yang mampu menolak kandidat tidak valid sedini mungkin, tetapi tetap menjaga kelengkapan pencarian.

Pada konteks backtracking, pruning dilakukan terhadap solusi parsial. Setiap kali solusi parsial diperluas dengan sebuah kandidat baru, algoritma memeriksa apakah solusi parsial tersebut masih konsisten dengan batasan masalah. Jika tidak konsisten, algoritma segera melakukan backtrack. Dengan demikian, pruning membantu menghindari eksplorasi cabang yang sejak awal sudah tidak mungkin menghasilkan solusi akhir.

G. Kompleksitas Algoritma

Kompleksitas algoritma adalah ukuran yang digunakan untuk memperkirakan kebutuhan sumber daya suatu algoritma terhadap ukuran masukan. Sumber daya yang umum dianalisis adalah waktu eksekusi dan penggunaan memori. Kompleksitas waktu menunjukkan banyaknya langkah dasar yang dilakukan algoritma, sedangkan kompleksitas ruang menunjukkan banyaknya memori tambahan yang diperlukan selama algoritma berjalan.

Analisis kompleksitas biasanya dinyatakan sebagai fungsi dari ukuran masukan, misalnya n. Dengan cara ini, performa algoritma dapat dibandingkan tanpa bergantung pada bahasa pemrograman, mesin, atau kondisi eksekusi tertentu. Fokus utama analisis bukan pada waktu absolut, melainkan pada laju pertumbuhan kebutuhan sumber daya ketika ukuran masukan bertambah.

Notasi asimptotik digunakan untuk menyatakan pertumbuhan kompleksitas. Notasi $O(g(n))$ menyatakan batas atas laju pertumbuhan suatu fungsi, sehingga sering digunakan untuk menggambarkan kompleksitas kasus terburuk. Notasi $\Omega(g(n))$ menyatakan batas bawah, sedangkan notasi $\Theta(g(n))$ menyatakan batas ketat ketika suatu fungsi memiliki batas atas dan batas bawah dengan orde pertumbuhan yang sama.

Dalam analisis algoritma, beberapa kategori pertumbuhan yang umum dijumpai adalah konstan $O(1)$, logaritmik $O(\log n)$, linear $O(n)$, linearitmik $O(n \log n)$, kuadratik $O(n^2)$, polinomial, eksponensial, dan faktorial. Algoritma dengan pertumbuhan eksponensial atau faktorial biasanya menjadi sulit digunakan untuk masukan besar karena jumlah langkah meningkat sangat cepat.

Kompleksitas juga dapat dianalisis berdasarkan skenario kasus terbaik, kasus rata-rata, dan kasus terburuk. Kasus terbaik menunjukkan kondisi ketika algoritma bekerja paling cepat, kasus terburuk menunjukkan kondisi paling tidak menguntungkan, sedangkan kasus rata-rata memperkirakan performa algoritma pada masukan yang dianggap umum. Pada banyak pembahasan algoritma, kasus terburuk sering digunakan karena memberikan jaminan batas maksimum jumlah operasi yang mungkin dilakukan.

III. Definisi Permasalahan

A. Ruang Lingkup Masalah

Dalam bidang cheminformatics, salah satu kebutuhan yang sering muncul adalah pencarian molekul berdasarkan pola struktur tertentu. Pola tersebut dapat berupa gugus fungsi, hubungan antaratom, susunan ikatan, atau cabang struktur tertentu yang menjadi ciri dari suatu kelompok senyawa. Permasalahan ini berbeda dari pencarian berdasarkan nama atau rumus molekul karena struktur kimia dipengaruhi oleh cara atom-atom saling terhubung.

Pada makalah ini, setiap molekul dimodelkan sebagai graf berlabel. Atom direpresentasikan sebagai simpul, sedangkan ikatan kimia direpresentasikan sebagai sisi. Label pada simpul menunjukkan jenis atom, seperti C, O, N, Cu, Co, atau Fe. Label pada sisi menunjukkan jenis ikatan, seperti ikatan tunggal, rangkap dua, rangkap tiga, aromatik, atau koordinasi.

Secara formal, diberikan sebuah basis data molekul:

$$D = \{M_1, M_2, \ldots, M_m\} \tag{1}$$

dengan setiap molekul M_i direpresentasikan sebagai graf berlabel:

$$M_i = (V_i, E_i, L_i^V, L_i^E) \tag{2}$$

di mana V_i adalah himpunan atom, E_i adalah himpunan ikatan, L_i^V adalah fungsi label simpul, dan L_i^E adalah fungsi label sisi. Selain itu, diberikan sebuah pattern substruktur:

$$P = (V_P, E_P, L_P^V, L_P^E) \tag{3}$$

Pattern P merepresentasikan struktur kecil yang ingin dicari pada setiap molekul target di dalam basis data. Pattern dapat berbentuk linear, bercabang, atau memiliki cabang bertingkat. Contohnya, `C=O` menyatakan ikatan rangkap dua antara karbon dan oksigen, `C(=O)-O` menyatakan struktur bercabang, sedangkan `C{C-C-C,C-C,C}` menyatakan atom pusat yang memiliki tiga cabang dengan panjang berbeda. Tujuan dari permasalahan ini adalah menentukan semua molekul M_i yang mengandung pattern P sebagai bagian dari strukturnya.

Untuk suatu molekul target M_i, pencocokan dianggap berhasil apabila terdapat fungsi pemetaan:

$$f : V_P \to V_i \tag{4}$$

yang memenuhi syarat-syarat berikut:

- Setiap simpul pada pattern dipetakan ke simpul target yang berbeda.
- Label setiap simpul pattern sama dengan label simpul target hasil pemetaan.
- Untuk setiap sisi $(u, v) \in E_P$, terdapat sisi $(f(u), f(v)) \in E_i$.
- Label sisi pada pattern sama dengan label sisi pada graf target hasil pemetaan.
- Jika pattern menyatakan jumlah hidrogen implisit atau oxidation state tertentu, nilai tersebut harus dapat dipenuhi oleh atom target.

Jika seluruh syarat tersebut terpenuhi, maka molekul M_i dinyatakan mengandung substruktur P. Masukan persoalan terdiri atas basis data molekul dalam format JSON, pattern substruktur yang ditulis pengguna, dan mode pencarian yang dipilih. Keluaran dari permasalahan ini adalah daftar molekul yang cocok beserta mapping atom pattern ke atom target. Mapping tersebut menunjukkan bagian spesifik dari molekul target yang memenuhi pola struktur yang dicari.

Ruang lingkup persoalan dibatasi pada representasi graf molekul sederhana. Program belum ditujukan sebagai parser kimia lengkap seperti SMILES atau SMARTS. Ikatan yang dicocokkan adalah jenis ikatan yang eksplisit berada pada data dan pattern, yaitu ikatan tunggal, rangkap dua, rangkap tiga, aromatik, dan koordinasi. Aspek seperti stereochemistry, ring closure dengan notasi khusus, resonansi, formal charge otomatis, dan validasi kimia penuh berada di luar cakupan implementasi.

IV. Solusi

A. Representasi Graf Molekul

Untuk menyelesaikan permasalahan tersebut, langkah pertama adalah mengubah data molekul dan pattern menjadi representasi graf berlabel. Representasi ini dipilih karena hubungan atom dan ikatan pada molekul dapat dinyatakan secara langsung sebagai simpul dan sisi. Dengan demikian, persoalan pencarian substruktur dapat diproses sebagai persoalan pencocokan graf.

Data molekul dibaca dari file JSON. Setiap atom memiliki id, elemen atom, oxidation state opsional, dan jumlah hidrogen implisit. Setiap ikatan memiliki pasangan atom asal-tujuan dan jenis ikatan. Setelah seluruh data dibaca, program membentuk struktur graf yang menyimpan daftar atom, daftar ikatan, serta adjacency map untuk mempercepat pemeriksaan apakah dua atom terhubung dan jenis ikatan apa yang menghubungkannya.

B. Pembentukan Graf Pattern

Pattern substruktur diberikan oleh pengguna dalam notasi sederhana agar pengguna tidak perlu menuliskan graf secara eksplisit. Notasi seperti `C-O`, `C=O`, dan `C#N` menyatakan hubungan linear antaratom. Notasi `C(=O)-O` menyatakan cabang dengan tanda kurung biasa. Notasi `C{C,C,C}` menyatakan cabang ringkas dari satu atom pusat. Notasi ringkas tersebut juga mendukung rantai dan cabang bertingkat, misalnya `C{C-C-C,C-C,C}` dan `C{C{O,N},C-C}`.

Parser membaca pattern dari kiri ke kanan dan membangun graf pattern secara bertahap. Ketika parser menemukan simbol atom, parser membentuk simpul baru. Ketika parser menemukan simbol ikatan, parser menyimpan jenis ikatan yang akan digunakan untuk menghubungkan simpul berikutnya. Ketika parser membaca cabang, parser mempertahankan atom pusat sehingga setiap cabang dapat dibangun dari atom tersebut tanpa mengubah posisi utama pembacaan pattern. Pada notasi kurung kurawal, setiap item yang dipisahkan koma diproses sebagai satu cabang tersendiri. Karena proses parsing dilakukan secara rekursif, cabang dapat memiliki cabang lain di dalamnya.

C. Pencarian dengan Backtracking

Setelah graf target dan graf pattern terbentuk, program melakukan pencarian mapping menggunakan backtracking. Perancangan backtracking untuk persoalan ini dapat dijelaskan melalui tiga komponen utama, yaitu solusi persoalan, fungsi pembangkit, dan fungsi pembatas.

1) Solusi Persoalan: Solusi persoalan dinyatakan sebagai vektor mapping:

$$X = [x_1, x_2, \ldots, x_p] \tag{5}$$

dengan p adalah jumlah atom pada graf pattern. Elemen x_k menyatakan atom target yang dipasangkan dengan atom pattern ke-k. Dengan demikian, solusi lengkap terbentuk apabila seluruh atom pattern sudah memiliki pasangan atom target.

Solusi parsial pada level ke-k berbentuk:

$$X[1..k] = [x_1, x_2, \ldots, x_k] \tag{6}$$

yang berarti k atom pertama pada pattern telah dipetakan ke atom target tertentu. Jika solusi parsial tersebut dapat diperluas sampai level p dan seluruh syarat pencocokan terpenuhi, maka graf pattern ditemukan sebagai bagian dari graf target. Mapping yang terbentuk kemudian disimpan sebagai salah satu solusi.

2) Fungsi Pembangkit: Fungsi pembangkit bertugas menghasilkan kandidat nilai untuk x_k pada level ke-k. Pada persoalan ini, kandidat yang dibangkitkan adalah atom-atom target yang belum digunakan pada mapping sebelumnya. Dengan kata lain, untuk setiap atom pattern yang sedang diproses, algoritma mencoba memasangkannya dengan setiap atom target yang masih tersedia.

Secara konseptual, fungsi pembangkit dapat dituliskan sebagai:

```
GENERATE_CANDIDATES(k):
    candidates = {}
    for each target atom t:
        if t belum digunakan:
            masukkan t ke candidates
    return candidates
```

Fungsi pembangkit ini membentuk cabang-cabang pada pohon ruang status. Setiap kandidat atom target menghasilkan satu kemungkinan perluasan solusi parsial. Apabila kandidat tersebut lolos fungsi pembatas, algoritma melanjutkan pencarian ke level berikutnya. Jika tidak, cabang tersebut tidak dikembangkan lebih lanjut. Bentuk pattern yang bercabang, termasuk cabang bertingkat, tidak mengubah bentuk solusi backtracking karena setiap atom pattern tetap diproses sebagai simpul pada graf. Perbedaannya terletak pada jumlah relasi sisi yang harus diperiksa oleh fungsi pembatas.

3) Fungsi Pembatas: Fungsi pembatas atau bounding function digunakan untuk menentukan apakah solusi parsial masih berpeluang menghasilkan solusi lengkap. Fungsi ini memeriksa kandidat atom target sebelum algoritma melanjutkan pencarian ke level berikutnya. Jika kandidat tidak memenuhi batasan, kandidat tersebut langsung ditolak.

Pada persoalan pencarian substruktur molekul, fungsi pembatas memeriksa beberapa kondisi berikut.

- Atom target belum digunakan oleh atom pattern lain.
- Label atom pattern sama dengan label atom target.
- Jika oxidation state diperhitungkan, nilainya harus sesuai.
- Jumlah hidrogen implisit pada atom target harus mencukupi kebutuhan atom pattern.
- Degree atom target tidak boleh lebih kecil daripada degree atom pattern.
- Untuk setiap atom pattern yang sudah dipetakan sebelumnya, ikatan yang ada pada pattern harus juga ada pada graf target.
- Jenis ikatan pada pattern harus sama dengan jenis ikatan pada graf target.

Secara konseptual, fungsi pembatas dapat dituliskan sebagai:

```
BOUND(k, candidate):
    if candidate sudah digunakan:
        return false
    if label atom tidak sesuai:
        return false
    if oxidation state tidak sesuai:
        return false
    if implicit hydrogen tidak mencukupi:
        return false
    if degree target tidak mencukupi:
        return false
    for setiap atom pattern yang sudah dipetakan:
        if ikatan pattern tidak cocok di target:
            return false
    return true
```

Jika fungsi pembatas mengembalikan `false`, simpul pada pohon ruang status dianggap sebagai simpul mati dan algoritma melakukan backtrack. Jika fungsi pembatas mengembalikan `true`, solusi parsial masih dianggap menjanjikan sehingga pencarian dapat dilanjutkan ke level berikutnya.

D. Optimisasi dengan Pruning

Pure backtracking dapat menghasilkan ruang pencarian yang besar karena algoritma mencoba banyak kemungkinan mapping. Untuk mengurangi jumlah cabang yang dieksplorasi, digunakan pruning sederhana. Pruning dilakukan saat mapping parsial sedang dibangun, sehingga kandidat yang tidak mungkin menghasilkan solusi dapat ditolak lebih awal.

Aturan pruning yang digunakan meliputi kesesuaian label atom, status penggunaan atom target, kecukupan degree atom target, kecocokan jumlah hidrogen implisit, kesesuaian oxidation state, serta kesesuaian ikatan dengan atom pattern lain yang sudah dipetakan. Jika salah satu aturan tersebut tidak terpenuhi, algoritma tidak melanjutkan pencarian pada cabang tersebut.

Pruning tidak mengubah jawaban yang dihasilkan karena kandidat hanya ditolak apabila sudah pasti melanggar syarat pencocokan. Misalnya, atom target dengan label berbeda tidak mungkin menjadi pasangan atom pattern, atom target dengan degree lebih kecil tidak mungkin memenuhi semua sisi pada pattern, dan ikatan yang tidak sesuai dengan mapping parsial tidak mungkin menjadi bagian dari solusi lengkap.

V. METODOLOGI

A. Representasi Data Molekul

Setiap molekul direpresentasikan sebagai graf berlabel. Data atom menyimpan id atom, elemen atom, oxidation state opsional, dan jumlah hidrogen implisit. Data ikatan menyimpan atom asal, atom tujuan, dan jenis ikatan. Untuk mempercepat pengecekan keterhubungan atom, graf juga menyimpan adjacency map yang memetakan pasangan atom ke jenis ikatan.

Database molekul dibaca dari file JSON. Setelah atom dan ikatan dimuat, program membentuk graf untuk setiap molekul. Graf tersebut kemudian digunakan sebagai graf target dalam proses pencarian.

B. Input Pattern dan Parsing

Pengguna memasukkan pattern substruktur menggunakan notasi sederhana. Pattern linear dapat ditulis sebagai `C-O`, `C=O`, atau `C#N`. Pattern dengan cabang dapat ditulis menggunakan tanda kurung, seperti `C(=O)-O`, atau menggunakan notasi ringkas, seperti `C{C,C,C}`, `C{C-C-C,C-C,C}`, dan `C{C{O,N},C-C}`.

Parsing dilakukan dengan membaca pattern dari kiri ke kanan. Simbol atom menghasilkan simpul baru, simbol ikatan menentukan jenis sisi yang akan dibuat, dan simbol cabang mengalihkan proses pembacaan ke konteks atom pusat. Pada cabang ringkas, koma memisahkan satu cabang dengan cabang lainnya. Karena parser menggunakan pemrosesan rekursif, cabang yang sedang dibaca dapat berisi cabang lagi.

C. Mode Pencarian dan Keluaran Program

Setelah database dan pattern berhasil dibentuk menjadi graf, program menangani proses pencarian berdasarkan mode yang dipilih pengguna. Mode pertama adalah pure backtracking. Pada mode ini, program menjalankan pencarian tanpa pruning awal, kemudian menampilkan molekul yang mengandung pattern beserta mapping atom yang ditemukan.

Mode kedua adalah backtracking dengan pruning. Pada mode ini, program menggunakan aturan pembatas saat pencarian berlangsung sehingga kandidat yang tidak mungkin menjadi solusi dapat ditolak lebih awal. Hasil yang ditampilkan tetap berupa daftar molekul yang cocok dan mapping atom, tetapi dilengkapi statistik pruning yang menunjukkan jumlah kandidat yang dipangkas berdasarkan label, degree, dan ikatan.

Mode ketiga adalah mode komparasi. Pada mode ini, program menjalankan pencarian dengan pure backtracking dan backtracking dengan pruning pada molekul yang sama. Hasil kedua mode dibandingkan melalui statistik pencarian, yaitu jumlah recursive calls, candidates checked, jumlah pruning, dan runtime. Mode ini digunakan untuk memperlihatkan perbedaan eksplorasi antara pencarian tanpa pruning dan pencarian dengan pruning.

Untuk setiap mode, keluaran utama program adalah nama molekul yang cocok, jumlah mapping yang ditemukan, dan pasangan atom pattern ke atom target. Informasi mapping menunjukkan bagian spesifik dari molekul target yang memenuhi pattern substruktur yang dimasukkan pengguna.

VI. IMPLEMENTASI

A. Modul Program

Program diimplementasikan menggunakan C++ dengan beberapa modul utama. Modul `Atom` menyimpan informasi atom, sedangkan modul `Bond` menyimpan informasi ikatan. Modul `MoleculeGraph` merepresentasikan molekul atau pattern sebagai graf dan menyediakan operasi seperti menambah atom, menambah ikatan, mengecek keberadaan ikatan, mengambil jenis ikatan, dan menghitung degree atom.

Modul `MoleculeDatabase` bertugas membaca database molekul dari file JSON. Modul `PatternParser` mengubah input pattern dari pengguna menjadi graf pattern. Modul ini mendukung pattern linear, cabang biasa, cabang ringkas, dan cabang bertingkat melalui parsing rekursif. Modul `SubstructureMatcher` menjalankan algoritma backtracking dan pruning. Modul `UIManager` menangani interaksi pengguna melalui command line interface.

B. Implementasi Struktur Pencarian

Pencarian substruktur dilakukan pada modul `SubstructureMatcher`. Modul ini menyimpan dua struktur utama, yaitu `mapping` dan `usedTarget`. Struktur `mapping` menyimpan pasangan atom pattern ke atom target, sedangkan `usedTarget` menandai atom target yang sudah digunakan agar satu atom target tidak dipakai oleh lebih dari satu atom pattern.

```
std::vector<int> mapping;
std::vector<bool> usedTarget;

// mapping[patternIndex] = targetIndex
// usedTarget[targetIndex] = true jika sudah dipakai
```

Sebelum pencarian dimulai, seluruh elemen `mapping` diisi dengan nilai `-1`, sedangkan seluruh elemen `usedTarget` diisi `false`. Jika jumlah atom pattern lebih besar daripada jumlah atom target, pencarian tidak dijalankan karena mapping injektif tidak mungkin terbentuk.

C. Implementasi Backtracking

Fungsi utama pencarian adalah `backtrack`. Parameter `depth` menyatakan level pencarian, yaitu indeks atom pattern yang sedang dipetakan. Pada setiap level, program mencoba semua atom target yang belum digunakan. Jika kandidat dapat digunakan, mapping diperluas dan fungsi dipanggil kembali untuk level berikutnya.

```
void SubstructureMatcher::backtrack(int depth) {
    result.recursiveCalls++;

    if (depth == pattern.atomCount()) {
        if (usePruning || isFullMappingValid()) {
            // oneMapping berisi salinan mapping
    saat ini
            result.mappings.push_back(oneMapping);
            result.found = true;
        }
        return;
    }

    int patternIndex = depth;
```

```
    for (int targetIndex = 0;
         targetIndex < target.atomCount();
         targetIndex++) {
        result.candidatesChecked++;
        if (usedTarget[targetIndex]) continue;

        if (usePruning &&
            !isPartialMappingValid(patternIndex,
    targetIndex)) {
            continue;
        }

        mapping[patternIndex] = targetIndex;
        usedTarget[targetIndex] = true;

        backtrack(depth + 1);

        usedTarget[targetIndex] = false;
        mapping[patternIndex] = -1;
    }
}
```

Pada mode pure backtracking, pemeriksaan struktur lengkap dilakukan ketika seluruh atom pattern sudah dipetakan melalui fungsi `isFullMappingValid`. Pada mode pruning, kandidat diperiksa lebih awal sebelum pencarian dilanjutkan ke level berikutnya. Dengan demikian, perbedaan utama kedua mode berada pada pemanggilan fungsi pembatas saat mapping parsial sedang dibangun.

D. Implementasi Pruning

Pruning diimplementasikan pada fungsi `isPartialMappingValid`. Fungsi ini memeriksa apakah pasangan atom pattern dan atom target masih mungkin menjadi bagian dari solusi. Pemeriksaan awal dilakukan terhadap label atom, jumlah hidrogen implisit, oxidation state, dan degree atom. Setelah itu, fungsi memeriksa ikatan antara atom pattern yang sedang dipetakan dengan atom pattern lain yang sudah dipetakan sebelumnya.

```
bool isPartialMappingValid(int patternIndex,
                           int targetIndex) {
    const Atom& patternAtom =
        pattern.getAtomByIndex(patternIndex);
    const Atom& targetAtom =
        target.getAtomByIndex(targetIndex);

    if (!atomMatches(patternAtom, targetAtom)) {
        result.prunedByLabel++;
        return false;
    }

    if (target.degree(targetAtom.getId()) <
        pattern.degree(patternAtom.getId())) {
        result.prunedByDegree++;
        return false;
    }

    for (const Bond& bond : pattern.getBonds()) {
        // Pemeriksaan bond parsial dipadatkan
        if bond tidak terhubung ke patternAtom:
            continue;
        if atom pattern lain belum dipetakan:
            continue;
        if ikatan target tidak ada atau tipenya
    berbeda:
            result.prunedByBond++;
            return false;
    }
    return true;
}
```

Fungsi `atomMatches` memastikan bahwa elemen atom sama, jumlah hidrogen implisit pada target mencukupi kebutuhan pattern, dan oxidation state sesuai jika pattern menyatakannya. Pemeriksaan degree digunakan sebagai batas minimum jumlah tetangga yang harus dimiliki atom target. Pemeriksaan ikatan parsial memastikan bahwa setiap hubungan yang sudah dapat divalidasi pada solusi parsial tetap konsisten dengan graf target.

E. Implementasi Parser Cabang Bertingkat

Parser pattern menggunakan fungsi rekursif `parsePatternIntoGraph`. Fungsi ini menerima posisi awal pembacaan, graf pattern yang sedang dibangun, atom pusat saat ini, dan penanda apakah parsing harus berhenti ketika menemukan koma. Parameter tersebut memungkinkan parser membaca cabang di dalam kurung biasa maupun kurung kurawal.

```
parsePatternIntoGraph(text, start, end, graph,
                      nextAtomId, currentAtomId,
                      stopAtComma)
```

Ketika parser menemukan simbol `{`, parser membaca setiap item yang dipisahkan koma sebagai cabang dari atom pusat. Setiap item diproses dengan pemanggilan rekursif terhadap fungsi yang sama. Karena itu, item di dalam kurung kurawal dapat berupa atom tunggal, rantai atom, atau cabang lain.

```
if karakter == '{':
    pos = posisi setelah '{'
    while belum menemukan '}':
        branchEnd = parsePatternIntoGraph(
            text, pos, end, graph,
            nextAtomId, currentAtomId, true)
        lanjutkan ke cabang berikutnya
```

Pendekatan ini membuat pattern seperti `C{C-C-C,C-C,C}` dapat dibentuk sebagai tiga cabang dari atom pusat, sedangkan `C{C{O,N},C-C}` dapat dibentuk sebagai cabang yang memiliki percabangan lanjutan.

F. Format Database JSON

Database molekul disimpan sebagai array JSON. Setiap molekul memiliki nama, formula, daftar atom, dan daftar ikatan. Contoh format molekul adalah sebagai berikut.

```
{
  "name": "Acetic Acid",
  "formula": "C2H4O2",
  "atoms": [
    { "id": 0, "element": "C" },
    { "id": 1, "element": "C" },
    { "id": 2, "element": "O" },
    { "id": 3, "element": "O" }
  ],
  "bonds": [
    { "from": 0, "to": 1, "type": "single" },
    { "from": 1, "to": 2, "type": "double" },
    { "from": 1, "to": 3, "type": "single" }
  ]
}
```

Untuk atom logam atau atom dengan bilangan oksidasi tertentu, atribut `oxidation_state` dapat ditambahkan pada data atom. Untuk atom yang perlu menyatakan hidrogen implisit, atribut `implicit_h` dapat digunakan.

G. Format Pattern

Program mendukung beberapa simbol ikatan, yaitu − untuk ikatan tunggal, = untuk ikatan rangkap dua, # untuk ikatan rangkap tiga, **:** untuk ikatan aromatik, dan ~ untuk ikatan koordinasi.

Contoh pattern yang didukung adalah sebagai berikut.

```
C-O
C=O
C(=O)-O
C{=O,-O}
C#N
Cu~N
CH3-OH
C{C,C,C}
C{C-C-C,C-C,C}
C{C{O,N},C-C}
```

Pattern `C{C,C,C}` merupakan notasi ringkas untuk menggambarkan satu atom karbon pusat yang terhubung ke tiga atom karbon lain. Pattern `C{C-C-C,C-C,C}` menyatakan tiga cabang dengan panjang berbeda dari atom pusat, sedangkan `C{C{O,N},C-C}` menyatakan cabang yang masih memiliki percabangan di level berikutnya.

VII. Pengujian dan Analisis

A. Analisis Kompleksitas

Untuk satu molekul target dengan n atom dan pattern dengan p atom, algoritma pure backtracking mencoba memasangkan setiap atom pattern ke atom target yang berbeda. Jumlah kemungkinan mapping injektif secara teoritis adalah $n \times (n-1) \times \cdots \times (n-p+1)$. Nilai tersebut dapat dibatasi secara kasar dengan $O(n^p)$.

Pada setiap mapping lengkap, program perlu memvalidasi kesesuaian ikatan pada graf pattern. Jika jumlah ikatan pada pattern adalah e_p, maka kompleksitas pure backtracking untuk satu molekul dapat ditulis sebagai:

$$O(n^p \cdot e_p) \tag{7}$$

Jika database berisi m molekul dan ukuran molekul dianggap memiliki batas atas n, kompleksitas total pencarian secara kasar adalah:

$$O(m \cdot n^p \cdot e_p) \tag{8}$$

Ruang memori utama yang digunakan selama pencarian adalah vektor mapping, vektor penanda atom target yang sudah digunakan, dan stack rekursi. Karena kedalaman rekursi maksimum sama dengan jumlah atom pattern, kebutuhan memori tambahan utama adalah $O(p+n)$ untuk satu proses pencocokan.

Pattern bercabang atau bercabang bertingkat dapat menambah jumlah sisi yang harus divalidasi pada graf pattern. Namun, bentuk cabang tersebut tetap direpresentasikan sebagai graf dengan p simpul dan e_p sisi, sehingga analisis kompleksitas tetap bergantung pada jumlah atom pattern, jumlah atom target, dan jumlah ikatan pattern.

Dampak pruning terlihat pada jumlah cabang yang benar-benar dieksplorasi. Pruning tidak mengubah kompleksitas terburuk secara teoritis karena masih terdapat kemungkinan kasus ketika banyak kandidat perlu diperiksa. Namun, pada kasus praktis, pruning dapat menolak kandidat sejak mapping parsial, misalnya ketika label atom tidak cocok, degree target tidak mencukupi, atau ikatan parsial tidak sesuai. Akibatnya, jumlah recursive calls dan runtime dapat menjadi jauh lebih kecil dibandingkan pure backtracking.

B. Analisis Hasil

Pengujian dilakukan dengan menjalankan program pada mode komparasi. Pada mode ini, program menjalankan pure backtracking dan backtracking dengan pruning terhadap pattern dan database yang sama. Pattern yang digunakan adalah `C{=O,-O}`, yaitu pattern bercabang dengan satu atom karbon pusat yang terhubung ke atom oksigen melalui ikatan rangkap dua dan ke atom oksigen lain melalui ikatan tunggal. Pattern ini dipilih karena muncul pada ketiga file JSON yang tersedia dan cukup menunjukkan kebutuhan validasi cabang. Runtime yang ditampilkan merupakan total waktu pencarian untuk seluruh molekul pada file JSON terkait, bukan waktu untuk satu molekul.

1) Pada file `data/molecules.json`, program menguji 28 molekul dan menemukan 5 molekul yang mengandung pattern. Pure backtracking menghasilkan 4545 recursive calls dengan runtime 2.2331 ms, sedangkan pruning menghasilkan 83 recursive calls dengan runtime 0.3372 ms. Pada pengujian ini, pruning berjalan sekitar 6.622479 kali lebih cepat.
2) Pada file `data/molecules_121.json`, program menguji 121 molekul dan menemukan 22 molekul yang mengandung pattern. Pure backtracking menghasilkan 17052 recursive calls dengan runtime 8.4827 ms, sedangkan pruning menghasilkan 451 recursive calls dengan runtime 1.7563 ms. Pada pengujian ini, pruning berjalan sekitar 4.829870 kali lebih cepat.
3) Pada file `data/molecules_1000.json`, program menguji 1000 molekul dan menemukan 134 molekul yang mengandung pattern. Pure backtracking menghasilkan 27496814 recursive calls dengan runtime 11918.5135 ms, sedangkan pruning menghasilkan 8152 recursive calls dengan runtime 97.2130 ms. Pada pengujian ini, pruning berjalan sekitar 122.602054 kali lebih cepat.

Hasil pengujian menunjukkan bahwa representasi graf memungkinkan program mencocokkan struktur molekul berdasarkan atom dan ikatan. Pattern bercabang seperti `C{=O,-O}` menunjukkan bahwa satu atom pusat dapat memiliki lebih dari satu hubungan yang harus dipenuhi. Pada dataset kecil, perbedaan runtime sudah terlihat, tetapi dampaknya menjadi jauh lebih besar pada dataset 1000 molekul karena

pure backtracking harus mengeksplorasi jauh lebih banyak kemungkinan mapping.

Mode pruning membantu mempercepat pencarian dengan menolak kandidat yang tidak mungkin valid sejak tahap awal. Misalnya, atom target dengan label berbeda dari atom pattern tidak perlu diperiksa lebih lanjut. Demikian pula, atom target yang tidak memiliki degree cukup atau tidak memiliki ikatan yang sesuai dengan mapping parsial dapat langsung dipangkas. Hal ini menjelaskan penurunan recursive calls yang besar pada ketiga pengujian.

VIII. Kesimpulan

Pencarian substruktur molekul menunjukkan bahwa representasi graf dapat menjadi cara yang alami untuk memodelkan struktur kimia. Atom sebagai simpul dan ikatan sebagai sisi memungkinkan pencarian dilakukan berdasarkan hubungan struktural, bukan hanya berdasarkan rumus molekul atau kemunculan teks. Dengan sudut pandang ini, pencarian substruktur dapat dipahami sebagai persoalan pencocokan subgraf berlabel.

Backtracking sesuai untuk persoalan ini karena pencarian mapping atom dibangun secara bertahap. Setiap tahap merepresentasikan pemilihan pasangan antara atom pattern dan atom target. Namun, sifat eksploratif backtracking juga menunjukkan bahwa ruang pencarian dapat tumbuh sangat besar ketika ukuran molekul target atau jumlah atom pattern meningkat.

Pruning menjadi bagian penting karena dapat membatasi eksplorasi tanpa mengubah makna solusi. Pemeriksaan label atom, degree, hidrogen implisit, oxidation state, dan ikatan parsial memungkinkan kandidat yang tidak mungkin valid ditolak sejak awal. Hasil pengujian menunjukkan bahwa perbedaan jumlah recursive calls menjadi semakin besar pada dataset yang lebih besar, sehingga manfaat pruning lebih terasa pada kasus praktis..

IX. Lampiran

Kode sumber program dan dokumen makalah tersedia pada repositori GitHub berikut.
https://github.com/Alpaomega1136/Makalah-Strategi-Algoritma-2026

X. Ucapan Terima Kasih

Penulis mengucapkan syukur kepada Tuhan Yang Maha Esa atas kekuatan, ketekunan, dan kesempatan yang diberikan sehingga makalah ini dapat diselesaikan. Penulis juga menyampaikan terima kasih kepada Ir. Rinaldi Munir, M.T., selaku dosen mata kuliah IF2211 Strategi Algoritma, atas materi, arahan, dan bimbingan yang diberikan selama proses pembelajaran. Penulis juga berterima kasih kepada semua pihak yang telah memberikan dukungan dan bantuan selama penyusunan makalah ini.

References

[1] R. Munir, "Graf (Bag. 1)," Bahan Kuliah IF1220 Matematika Diskrit, Program Studi Teknik Informatika, STEI-ITB, 2026. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2025-2026/20-Graf-Bagian1-2026.pdf.

[2] R. Munir, "Graf (Bag. 2)," Bahan Kuliah IF1220 Matematika Diskrit, Program Studi Teknik Informatika, STEI-ITB, 2026. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2025-2026/21-Graf-Bagian2-2026.pdf.

[3] R. Munir, "Graf (Bag. 3)," Bahan Kuliah IF1220 Matematika Diskrit, Program Studi Teknik Informatika, STEI-ITB, 2026. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2025-2026/22-Graf-Bagian3-2026.pdf.

[4] R. Munir, "Algoritma Backtracking (Bag. 1)," Bahan Kuliah IF2211 Strategi Algoritma, Program Studi Teknik Informatika, STEI-ITB, 2026. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2025-2026/15-Algoritma-backtracking-(2026)-Bagian1.pdf.

[5] R. Munir, "Algoritma Backtracking (Bag. 2)," Bahan Kuliah IF2211 Strategi Algoritma, Program Studi Teknik Informatika, STEI-ITB, 2026. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2025-2026/16-Algoritma-backtracking-(2026)-Bagian2.pdf.

[6] R. Munir, "Kompleksitas Algoritma (Bag. 1)," Bahan Kuliah IF1220 Matematika Diskrit, Program Studi Teknik Informatika, STEI-ITB, 2026. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2025-2026/25-Kompleksitas-Algoritma-Bagian1-2026.pdf.

[7] R. Munir, "Kompleksitas Algoritma (Bag. 2)," Bahan Kuliah IF1220 Matematika Diskrit, Program Studi Teknik Informatika, STEI-ITB, 2026. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2025-2026/26-Kompleksitas-Algoritma-Bagian2-2026.pdf.

[8] S. Pemberton, "Phenotype," W3C Talks, 2011. [Online]. Available: https://www.w3.org/2011/Talks/01-14-steven-phenotype/.

[9] Stanford Network Analysis Project, "Neural Subgraph Matching," Stanford University. [Online]. Available: https://snap.stanford.edu/subgraph-matching/.

[10] Wolfram MathWorld, "Labeled Graph." [Online]. Available: https://mathworld.wolfram.com/LabeledGraph.html.

[11] R. Ying, Z. Lou, J. You, C. Wen, A. Canedo, and J. Leskovec, "Neural Subgraph Matching," arXiv:2007.03092, 2020. [Online]. Available: https://arxiv.org/abs/2007.03092.

[12] J. R. Ullmann, "An algorithm for subgraph isomorphism," *Journal of the ACM*, vol. 23, no. 1, pp. 31–42, 1976.

[13] H.-C. Ehrlich and M. Rarey, "Systematic benchmark of substructure search in molecular graphs – From Ullmann to VF2," *Journal of Cheminformatics*, vol. 4, no. 1, 2012.

[14] RDKit, "RDKit Documentation," [Online]. Available: https://www.rdkit.org/docs/.

[15] N. Lohmann, "JSON for Modern C++," [Online]. Available: https://github.com/nlohmann/json.

Pernyataan

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 8 Juni 2026

Raymond Jonathan Dwi Putra Julianto
13524059