Optimasi Urutan Perolehan dan Crafting Item untuk Membunuh Ender Dragon Menggunakan Program Dinamis dan Topological Sort

Fahd Muhammad Zahid - 13524078

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: blacklythning12@gmail.com, 13524078@std.stei.itb.ac.id

Abstract*—Minecraft memiliki sistem progresi berbasis item yang membentuk rantai dependensi dari sumber daya mentah, alat, akses dimensi, hingga payload untuk mengalahkan Ender Dragon. Makalah ini memodelkan progresi tersebut sebagai graf dependensi AND-OR. Program dinamis dengan memoization digunakan untuk mencari vektor kebutuhan sumber daya yang tidak terdominasi, sedangkan Topological Sort menyusun aksi terpilih menjadi urutan yang valid. Eksperimen pada Minecraft Java Edition 26.2 menghasilkan payload lima TNT minecart pada koordinat pusat blok dan interval 12 tick yang berhasil pada lima dari lima percobaan terkontrol. Untuk jalur portal berbasis lava casting, program menghasilkan kebutuhan 3 log, 11 cobblestone, 29 raw iron, 4 coal, 1 flint, 10 sumber lava, 6 blaze rod, 12 ender pearl, 20 sand, dan 25 gunpowder. Benchmark graf sintetis pada kedalaman 16 menunjukkan bahwa program dinamis mengevaluasi 18 state, sedangkan rekursi tanpa memoization mengevaluasi 196.607 state. Hasil ini memperlihatkan bahwa program dinamis efektif menghindari perhitungan submasalah berulang dan Topological Sort menjamin validitas urutan progresi.

***Keywords*—Minecraft; program dinamis; memoization; Topological Sort; graf dependensi; optimasi sumber daya**

I. PENDAHULUAN

A. Minecraft sebagai Domain Permasalahan

Minecraft adalah permainan sandbox yang memberi kebebasan kepada pemain untuk mengeksplorasi dunia prosedural, mengumpulkan sumber daya, membuat item, membangun struktur, dan menghadapi musuh. Walaupun permainan tidak memaksakan satu alur linear, mode Survival mempunyai progresi yang mudah dikenali. Pemain memulai permainan tanpa item, memperoleh kayu, membuat alat, menambang mineral, mengakses Nether, mencari material untuk membuka End Portal, lalu memasuki The End untuk menghadapi Ender Dragon.

Setiap tahap progresi mempunyai dependensi. Iron ingot membutuhkan raw iron, bahan bakar, dan furnace; furnace membutuhkan cobblestone; cobblestone lebih mudah diperoleh setelah pemain mempunyai pickaxe. Dependensi juga dapat bercabang. Portal Nether dapat dibuat dengan menambang obsidian menggunakan diamond pickaxe atau dengan teknik lava casting memakai bucket. Struktur ini membuat Minecraft cocok digunakan sebagai studi kasus strategi algoritma.

1.1 Gambar Minecraft

B. Tujuan Akhir dan Motivasi Penelitian

Tujuan permainan yang umum dipakai sebagai penanda penyelesaian adalah mengalahkan Ender Dragon. Namun penelitian ini tidak hanya mencari cara memasuki The End atau melakukan pertarungan biasa. Target penelitian adalah menyediakan payload eksplosif yang dapat menurunkan 200 health point naga melalui satu aktivasi awal. Satu aktivasi diperbolehkan memicu rangkaian ledakan otomatis yang telah disiapkan sebelumnya.

Pemilihan target tersebut menghubungkan algoritma dengan tujuan nyata di dalam permainan. Output bukan sekadar daftar recipe terpisah, melainkan pembuktian komputasional mengenai sumber daya minimum yang harus tersedia sejak dunia baru dimulai, urutan memperoleh item tersebut, dan konfigurasi payload yang terukur pada server permainan.

C. Rumusan Masalah

Permasalahan utama adalah menentukan kombinasi sumber daya mentah dan urutan aksi yang valid untuk mencapai goal dragon_killed. Program harus menangani recipe dengan banyak requirement, alternatif cara memperoleh satu goal, alat yang dapat digunakan kembali, output recipe dalam jumlah batch, dan surplus yang tersisa setelah suatu batch dipakai.

Pertanyaan penelitian meliputi cara memodelkan recipe sebagai graf AND-OR, cara program dinamis memilih alternatif yang tidak terdominasi, fungsi Topological Sort setelah optimasi selesai, kebutuhan material hasil program, serta perbedaan jumlah submasalah antara program dinamis dan rekursi tanpa memoization.

II. DASAR TEORI

A. Graf Dependensi AND-OR

Graf dependensi merepresentasikan setiap item atau kondisi sebagai goal. Sebuah aksi menghasilkan satu goal dan mempunyai nol atau lebih requirement. Requirement di dalam satu aksi membentuk hubungan AND karena semuanya harus dipenuhi. Jika satu goal dapat diperoleh melalui beberapa aksi, kumpulan aksi tersebut membentuk hubungan OR. Sebagai contoh, nether_portal dapat dihasilkan oleh lava_cast_nether_portal atau build_mined_obsidian_portal.

Arah sisi ditentukan dari produsen menuju pemakai. Jika aksi A menghasilkan iron_ingot dan aksi B membutuhkan iron_ingot, maka A harus mendahului B. Graf recipe yang valid harus bebas siklus pada subgraf aksi terpilih. Siklus menandakan data tidak konsisten, misalnya item X hanya dapat dibuat dari Y sementara Y hanya dapat dibuat dari X tanpa sumber awal.

B. Program Dinamis dan Memoization

Program dinamis memecahkan masalah melalui submasalah yang lebih kecil dan menyimpan hasil submasalah yang dapat muncul kembali. Implementasi penelitian menggunakan pendekatan top-down atau memoization. State tidak hanya memuat goal tunggal, melainkan seluruh demand residual agar dua cabang dianggap sama hanya jika kebutuhan consumed, surplus, kebutuhan reusable, dan reusable yang telah tersedia juga sama.

Untuk demand g dan alternatif a, jumlah batch dihitung dengan pembulatan ke atas terhadap rasio jumlah yang dibutuhkan dan output recipe. Requirement alternatif ditambahkan ke state berikutnya. Jika output batch melebihi demand, kelebihannya disimpan sebagai surplus. Hasil seluruh alternatif digabungkan dan dipangkas menggunakan dominasi Pareto sebelum dimasukkan ke memo.

C. Dominasi Pareto

Penelitian tidak menetapkan satu nilai heuristik untuk sand, iron, diamond, atau drop mob. Pembobotan semacam itu akan memaksakan preferensi subjektif yang belum tentu sesuai bagi semua pemain. Sebagai gantinya, biaya dinyatakan sebagai vektor jumlah sumber daya. Vektor x mendominasi y apabila x tidak lebih besar pada seluruh komponen dan lebih kecil pada setidaknya satu komponen. Solusi yang tidak didominasi dipertahankan dalam Pareto frontier.

Pendekatan Pareto penting ketika dua jalur menggunakan material yang berbeda. Lava casting membutuhkan sumber lava dan bucket, sedangkan penambangan obsidian membutuhkan diamond pickaxe dan obsidian. Tanpa preferensi tambahan, kedua solusi sah dipertahankan karena tidak saling mendominasi.

D. Topological Sort

Topological Sort menghasilkan urutan linear dari DAG sehingga setiap simpul muncul setelah seluruh pendahulunya. Dalam penelitian ini, algoritma tersebut tidak memilih alternatif termurah. Pemilihan dilakukan oleh program dinamis. Topological Sort menerima subgraf aksi dari plan terpilih dan mengurutkannya agar tidak ada crafting atau akses yang dilakukan sebelum requirement tersedia.

Pemisahan peran tersebut menjelaskan hubungan kedua algoritma. Program dinamis menjawab pertanyaan 'apa yang perlu diperoleh', sedangkan Topological Sort menjawab pertanyaan 'dalam urutan apa rencana boleh dilakukan'. Kompleksitas Topological Sort adalah O(V+E), dengan V sebagai jumlah aksi terpilih dan E sebagai jumlah hubungan dependensi di antara aksi tersebut.

E. Algoritma Kahn

Implementasi Topological Sort mengikuti prinsip algoritma Kahn. Mula-mula indegree setiap aksi dihitung. Aksi dengan indegree nol dapat dikerjakan karena tidak lagi menunggu aksi lain, sehingga dimasukkan ke antrean. Ketika sebuah aksi dikeluarkan, seluruh sisi keluar dihapus secara konseptual dan indegree penerus dikurangi. Penerus yang mencapai indegree nol kemudian dimasukkan ke antrean.

Jika jumlah aksi yang berhasil dikeluarkan lebih kecil daripada jumlah simpul, subgraf mengandung siklus dan tidak mempunyai urutan topologis. Pada dataset valid, seluruh aksi akan diproses. Penggunaan nama aksi yang terurut sebagai tie breaker membuat output deterministik ketika beberapa aksi independen sama-sama mempunyai indegree nol.

F. Kebenaran Pengurutan

Setiap aksi hanya dikeluarkan ketika indegree-nya nol. Artinya seluruh aksi yang menghasilkan requirement internal telah lebih dahulu dikeluarkan. Dengan induksi terhadap posisi dalam output, setiap prefix urutan selalu memenuhi dependensi aksi di dalam prefix tersebut. Oleh karena itu urutan akhir valid untuk seluruh subgraf terpilih.

G. Kompleksitas Pencarian

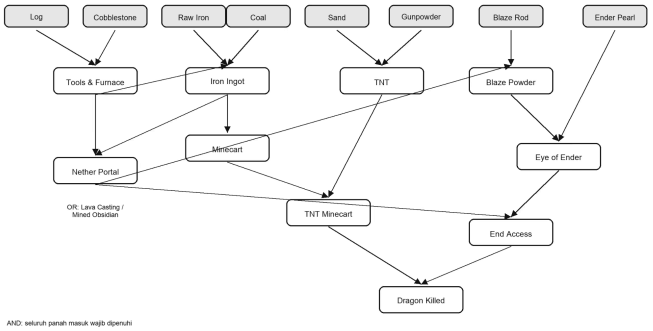
Kompleksitas DP dinyatakan terhadap jumlah residual state unik dan ukuran Pareto frontier pada setiap state. Setiap state unik diekspansi sekali, tetapi satu ekspansi dapat menghasilkan beberapa alternatif dan beberapa kombinasi plan. Pada kasus frontier kecil, memoization membuat pekerjaan mendekati linear terhadap jumlah state. Pada frontier besar, dominasi Pareto dapat menjadi komponen biaya utama.

III. PEMODELAN MASALAH

A. Batasan Penelitian

Eksperimen menggunakan Minecraft Java Edition 26.2 dalam konteks Vanilla Survival. Pemain diasumsikan mulai dari dunia baru dengan inventori kosong dan menyelesaikan rencana dalam satu kehidupan. Stronghold dianggap telah diketahui agar variabel seed dan jarak perjalanan tidak mendominasi model. End Portal diasumsikan membutuhkan tepat 12 Eye of Ender.

Waktu perjalanan, waktu penambangan, probabilitas menemukan struktur, dan jarak antarbioma tidak dioptimalkan karena nilainya berubah pada setiap seed. Metrik yang dikendalikan adalah jumlah drop atau material mentah. Makanan, armor, senjata keselamatan, dan item opsional tidak dimasukkan agar ruang masalah tetap terukur.



3.1 Graf Dependensi Progresi Minecraft

B. Klasifikasi Requirement

Requirement consumed habis saat aksi dilakukan. Contohnya, lima iron ingot habis untuk membuat minecart dan dua plank habis untuk membuat empat stick. Requirement reusable cukup tersedia satu kali. Crafting table, furnace, pickaxe, bucket, nether_portal, dan end_access termasuk reusable pada konteks aksi yang memakainya.

Pembedaan ini mencegah alat dihitung berulang. Sebuah bucket yang digunakan untuk lava casting tidak perlu dibuat sepuluh kali untuk sepuluh sumber lava. Demikian pula crafting table yang dipakai oleh banyak recipe tetap dibuat sekali selama rencana hanya memerlukan satu unit.

C. Representasi State

State D direpresentasikan sebagai empat peta: C untuk consumed demand, S untuk consumable surplus, Rp untuk reusable pending, dan Ra untuk reusable available. Seluruh peta diurutkan berdasarkan nama goal sehingga state mempunyai representasi kanonik dan dapat digunakan sebagai key tabel memo.

Ketika demand baru ditambahkan, program terlebih dahulu memeriksa surplus item yang sama. Misalnya satu coal menghasilkan delapan smelting_capacity. Jika satu iron dilebur, tujuh kapasitas tersisa disimpan dan dipakai saat demand iron berikutnya muncul. Tanpa surplus, perhitungan awal menghasilkan lima coal untuk 24 iron; setelah koreksi, kebutuhan tepat menjadi tiga coal.

D. Struktur Data

Dataset disimpan dalam JSON. Setiap alternatif memuat id, outputGoal, outputAmount, requirements, mode requirement, dan resourceCosts untuk item yang diperoleh langsung. Pemisahan data dan algoritma memungkinkan recipe diubah tanpa mengubah implementasi pencarian.

Elemen	Makna
outputGoal	Goal yang dihasilkan aksi
outputAmount	Jumlah output per batch
CONSUMED	Requirement yang habis
REUSABLE	Requirement yang dapat dipakai ulang
resourceCosts	Daun sumber daya mentah

E. Hierarki Goal

Goal tertinggi adalah dragon_killed. Goal tersebut mempunyai requirement end_access dan tnt_minecart. End_access diturunkan menjadi end_portal_open, dua belas Eye of Ender, blaze powder, blaze rod, ender pearl, serta akses Nether. TNT minecart diturunkan menjadi TNT dan minecart. TNT kemudian diturunkan menjadi sand dan gunpowder, sedangkan minecart diturunkan menjadi iron ingot.

Hierarki tersebut memperlihatkan bahwa satu target akhir menggabungkan dua rantai besar: rantai perjalanan menuju The End dan rantai produksi payload. Kedua rantai berbagi beberapa fasilitas, terutama crafting table, furnace, dan iron ingot. Penggabungan demand dalam satu state memungkinkan program memanfaatkan fasilitas reusable dan batch material secara bersama.

F. Contoh Ekspansi Recipe

Untuk lima TNT minecart, alternatif craft_tnt_minecart dieksekusi lima batch. Demand yang muncul adalah lima TNT dan lima minecart. Lima TNT menimbulkan demand 20 sand serta 25 gunpowder. Lima minecart menimbulkan demand 25 iron ingot. Ketiga angka ini kemudian digabungkan dengan kebutuhan iron untuk bucket dan flint and steel.

Contoh lain adalah craft_blaze_powder yang menghasilkan dua unit per blaze rod. Dua belas Eye of Ender membutuhkan dua belas blaze powder, sehingga program menghitung enam batch dan enam blaze rod. Pembulatan batch tidak menghasilkan surplus pada kasus ini, tetapi mekanisme surplus tetap diperlukan untuk plank, stick, dan smelting capacity.

G. Fungsi Objektif

Objektif penelitian bukan meminimalkan jumlah total semua item dengan menjumlahkan komponen vektor. Penjumlahan akan menyamakan satu diamond dengan satu sand dan menghilangkan informasi jenis material. Program mempertahankan vektor lengkap dan hanya membuang plan jika terdapat plan lain yang tidak lebih buruk pada semua material.

Dengan objektif tersebut, output dapat dibaca tanpa asumsi rarity atau waktu. Pengguna dapat memilih jalur Pareto berdasarkan keadaan seed atau preferensi pribadi setelah algoritma selesai. Pemisahan antara optimasi objektif dan pemilihan subjektif menjaga model tetap transparan.

IV. METODOLOGI PENELITIAN

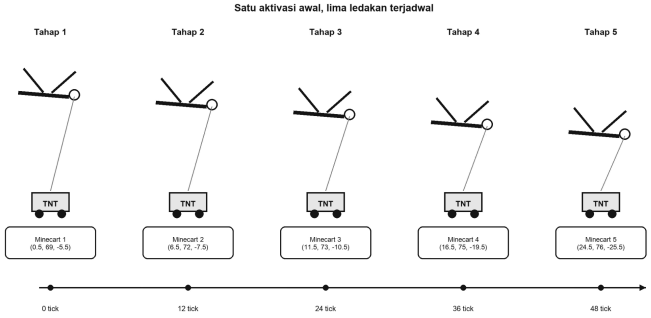
A. Tahap Penelitian

Penelitian dilakukan dalam enam tahap. Pertama, goal akhir dan batasan ditentukan. Kedua, recipe wajib dari kayu hingga akses The End dikumpulkan. Ketiga, recipe dimodelkan sebagai graf AND-OR. Keempat, payload eksplosif diuji pada server resmi Minecraft. Kelima, payload tervalidasi dimasukkan ke dataset progresi. Keenam, program dinamis, baseline, dan Topological Sort dijalankan untuk menghasilkan hasil eksperimen.

B. Eksperimen Damage Ender Dragon

Setiap percobaan menggunakan naga baru dengan health 200, AI aktif, posisi awal (0,5; 70; 0,5), rotasi awal nol, dan fase SittingScanning. End crystal alami dihapus agar tidak terjadi regenerasi health. Damage tidak dihitung menggunakan rumus abstrak, melainkan dibaca dari state entitas setelah ledakan pada server Java Edition 26.2.

Ledakan dijadwalkan oleh fungsi server agar interval mengikuti tick permainan dan tidak terpengaruh waktu query dari program pengendali. Screening awal menunjukkan interval delapan tick masih bertabrakan dengan periode damage immunity, sedangkan interval 12 tick menghasilkan hit terpisah dan tetap cukup cepat mengikuti lintasan kepala naga.



4.1 Konfigurasi Payload TNT Minecart

C. Pemilihan Payload

TNT biasa, end crystal, TNT minecart, dan kombinasi payload diuji. TNT minecart dipilih karena satu unit pada posisi kepala dapat menghasilkan damage jauh lebih besar daripada TNT biasa. Empat minecart pada koordinat kontinu berhasil 5/5 dan dipakai sebagai batas bawah laboratorium. Namun setelah koordinat dibulatkan ke pusat blok, empat minecart gagal 5/5 dan menyisakan sekitar 13 sampai 17 health point.

Penambahan minecart kelima pada koordinat pusat blok menghasilkan keberhasilan 5/5. Konfigurasi terpilih menggunakan titik (0,5;69;-5,5), (6,5;72;-7,5), (11,5;73;-10,5), (16,5;75;-19,5), dan (24,5;76;-25,5), dengan interval 12 tick.

D. Baseline Eksperimen Algoritma

Baseline memakai mesin pencarian state yang sama, tetapi tabel memo dinonaktifkan. Dengan rancangan ini, DP dan baseline menerapkan model, urutan pemilihan demand, serta Pareto pruning yang sama. Perbedaan state yang dievaluasi hanya disebabkan oleh penghitungan ulang submasalah identik.

E. Protokol Reset dan Pengukuran

Sebelum setiap trial, seluruh TNT, end crystal, dan TNT minecart eksperimen sebelumnya dihapus. Naga lama dipindahkan keluar area, lalu naga baru disummon dengan state awal yang sama. Program memverifikasi fase serta posisi sebelum payload dijadwalkan. Prosedur ini mencegah health, posisi, atau entitas sisa dari trial sebelumnya memengaruhi pengukuran berikutnya.

Status kill tidak hanya ditentukan dari nilai health nol. Pada Minecraft, naga memasuki DragonPhase 9 ketika proses kematian dimulai dan health dapat terlihat sebagai 1 pada saat pembacaan. Oleh karena itu trial dinyatakan berhasil jika entitas telah hilang atau berada pada fase kematian.

F. Penentuan Koordinat Payload

Koordinat awal diperoleh melalui screening posisi di sekitar badan dan kepala. Setelah satu prefix ledakan dijalankan, posisi serta rotasi naga pada tick berikutnya dicatat. Titik berikutnya dicari pada grid lokal di sekitar prediksi kepala. Proses diulang sampai diperoleh lintasan statis; konfigurasi akhir tidak mengikuti naga secara adaptif ketika validasi dijalankan.

Lintasan statis penting bagi definisi satu aktivasi. Seluruh titik dan waktu telah diketahui sebelum tombol awal ditekan. Server hanya mengeksekusi jadwal yang telah ditentukan. Query health dilakukan setelah seluruh payload selesai agar tidak menambah delay di antara ledakan.

G. Reproduksibilitas

Konfigurasi payload, versi permainan, interval, koordinat, jumlah trial, dan file hasil disimpan dalam JSON dan CSV. Dengan demikian eksperimen tidak bergantung pada deskripsi naratif saja. Script laboratorium dapat menjalankan reset, scheduling, pembacaan health, dan ekspor hasil secara otomatis.

V. IMPLEMENTASI

A. Arsitektur Program Java

Implementasi menggunakan Java dan dibagi menjadi package model, algorithm, data, serta experiment. DependencyGraph menyimpan alternatif per goal. ResourceVector menyimpan jumlah setiap material. ProgressionPlan menyimpan vektor material dan jumlah eksekusi aksi. DependencyGraphJsonLoader membaca dataset, sedangkan PlanningCsvExporter menulis hasil untuk dianalisis.

ParetoDynamicPlanner dan RecursivePlanner menggunakan AndOrSearchEngine yang sama. Sebuah flag menentukan apakah memoization aktif. Desain tersebut mengurangi risiko baseline menerapkan logika berbeda. ProgressionTopologicalSorter membangun sisi antaraksi yang terpilih dan menjalankan pengurutan topologis.

B. Transisi State

Pencarian dimulai dari state dengan consumed demand dragon_killed sebanyak satu. Mesin memilih demand pertama secara deterministik. Untuk setiap alternatif, jumlah batch dihitung, demand dihapus, surplus output dicatat, dan requirement ditambahkan. Jika goal adalah raw resource, demand dinormalisasi menjadi komponen ResourceVector tanpa ekspansi recipe tambahan.

Ketika state tidak lagi mempunyai demand, plan kosong dikembalikan. Pada saat rekursi kembali, biaya alternatif dan jumlah aksi ditambahkan. Kandidat yang memiliki ResourceVector sama dideduplikasi; kandidat yang didominasi dibuang. Hasil residual state disimpan dalam memo pada planner dinamis.

C. Pseudocode Program Dinamis

```
SOLVE(state):
  normalize raw resource
  if state in memo: return memo[state]
  if no demand: return empty plan
  goal = selectNext(state)
  for each alternative of goal:
    next = satisfy goal and add requirements
    save unused batch output as surplus
    combine alternative cost with SOLVE(next)
  memo[state] = ParetoPrune(candidates)
  return memo[state]
```

D. Penyusunan Urutan Aksi

Setelah DP menghasilkan plan, sorter hanya mempertahankan alternatif dengan actionCounts lebih dari nol. Jika requirement suatu aksi diproduksi oleh aksi lain dalam plan, sisi dependensi ditambahkan. Simpul dengan indegree nol dimasukkan ke antrean. Setiap simpul yang dikeluarkan mengurangi indegree penerusnya sampai seluruh aksi tersusun.

Beberapa aksi independen dapat mempunyai lebih dari satu urutan yang sama-sama valid. Contohnya sand dapat dikumpulkan sebelum atau setelah flint. Kebenaran Topological Sort tidak bergantung pada satu urutan naratif, tetapi pada pemenuhan seluruh hubungan pendahulu.

E. Pengujian

Test suite menguji operasi vektor sumber daya, dominasi Pareto, alternatif AND-OR, reusable requirement, surplus batch, memoization, pengurutan topologis, pembacaan JSON, output CSV, dan hasil dataset utama. Seluruh tes dijalankan melalui scripts/test.sh dan memberikan hasil All tests passed.

F. Invarian Kebenaran

Mesin pencarian mempertahankan beberapa invarian. Seluruh demand mempunyai jumlah positif; peta state bersifat immutable dan terurut; reusable pending tidak melebihi kekurangan terhadap reusable available; serta surplus hanya dikurangi ketika demand item yang sama ditambahkan. Invarian tersebut membantu memastikan bahwa memo key merepresentasikan state secara konsisten.

Plan yang dikembalikan dari base case tidak mempunyai biaya maupun aksi. Setiap tingkat rekursi hanya menambahkan biaya alternatif yang dipilih. Akibatnya ResourceVector akhir merupakan penjumlahan seluruh daun akuisisi langsung, sementara actionCounts merupakan jumlah batch setiap alternatif.

G. Deteksi Siklus

Selain memo, mesin menyimpan activeStates pada jalur rekursi saat ini. Jika state residual yang sama muncul sebelum ekspansi sebelumnya selesai, program menolak dataset dengan pesan dependency cycle. Pemeriksaan ini berbeda dari memo hit karena memo hanya berisi state yang telah selesai dihitung.

H. Format Output

Program menulis resource_vector, action_counts, states_evaluated, recursive_calls, dan memo_hits ke CSV. Output konsol menampilkan setiap plan Pareto beserta urutan aksi hasil Topological Sort. File eksperimen payload menyimpan health akhir, fase akhir, dan status keberhasilan per trial.

Pemisahan output membuat evaluasi dapat dilakukan tanpa membaca log server yang panjang. CSV resource digunakan untuk tabel kebutuhan material, CSV benchmark digunakan untuk perbandingan algoritma, dan CSV laboratorium digunakan untuk rasio keberhasilan payload.

VI. HASIL DAN PEMBAHASAN

A. Hasil Payload

Konfigurasi	Hasil
4 minecart kontinu	5/5; batas bawah
4 minecart pusat blok	0/5; tidak cukup
5 minecart pusat blok	5/5; terpilih

Hasil memperlihatkan bahwa minimalitas mekanik berbeda dari konfigurasi yang lebih mudah diwujudkan. Empat minecart cukup pada koordinat kontinu yang dikendalikan secara presisi, tetapi kehilangan damage setelah posisi dibulatkan ke grid blok. Penambahan satu unit memberikan margin yang cukup untuk lima keberhasilan berturut-turut.

B. Vektor Sumber Daya

Sumber Daya	Jumlah
Log	3
Cobblestone	11
Raw iron	29
Coal	4
Flint	1
Lava source	10
Blaze rod	6
Ender pearl	12
Sand	20
Gunpowder	25

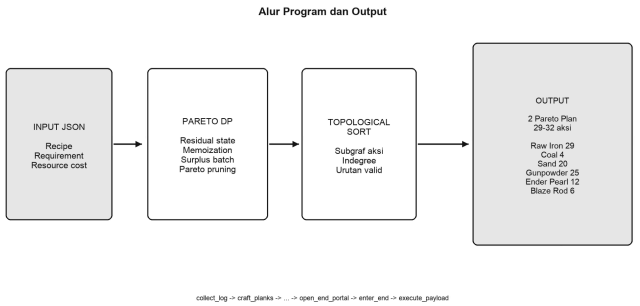
Tabel tersebut merupakan jalur lava casting. Dari 29 iron ingot, 25 digunakan untuk lima minecart, tiga untuk bucket, dan satu untuk flint and steel. Empat coal menyediakan kapasitas 32 smelting sehingga mencukupi 29 raw iron. Lima TNT membutuhkan 20 sand dan 25 gunpowder.

Solusi Pareto kedua mengganti sepuluh sumber lava dengan tiga diamond dan sepuluh obsidian. Kebutuhan log tetap tiga karena surplus plank dan stick digunakan lintas recipe. Kedua solusi tidak saling mendominasi karena menggunakan kategori sumber daya berbeda.

C. Output Topological Sort

Jalur lava casting menghasilkan 29 jenis aksi. Urutan dimulai dari collect_log, craft_planks, craft_sticks, crafting table, wooden pickaxe, dan penambangan cobblestone. Setelah stone pickaxe tersedia, pemain memperoleh raw iron dan coal, membuat furnace, melebur iron, membuat bucket dan flint and steel, lalu mengaktifkan Nether Portal.

Tahap berikutnya memperoleh enam blaze rod dan dua belas ender pearl untuk membuat dua belas Eye of Ender. Secara paralel pemain mengumpulkan sand dan gunpowder, membuat lima minecart, lima TNT, dan lima TNT minecart. Aksi open_end_portal mendahului enter_end, dan eksekusi payload selalu berada setelah end_access serta seluruh TNT minecart tersedia.



6.1 Hasil Urutan Aksi Program

D. Perbandingan Jalur Portal

Pada jalur lava casting, pemain membutuhkan tiga iron untuk bucket dan satu iron untuk flint and steel di luar 25 iron payload. Jalur ini tidak memerlukan diamond maupun penambangan obsidian, tetapi mengasumsikan tersedianya sepuluh sumber lava yang dapat disusun menjadi bingkai portal.

Pada jalur mined obsidian, tiga iron tambahan diperlukan untuk iron pickaxe. Pickaxe tersebut digunakan memperoleh tiga diamond, lalu diamond pickaxe digunakan memperoleh sepuluh obsidian. Meskipun jumlah jenis aksi bertambah, raw iron total tetap 29 karena jalur ini tidak membutuhkan bucket.

E. Verifikasi Aritmetika Material

Lima minecart mengonsumsi 5 x 5 = 25 iron. Bucket dan flint and steel mengonsumsi empat iron, sehingga total raw iron pada jalur lava casting adalah 29. Kapasitas smelting dibeli per delapan unit; ceil(29/8) menghasilkan empat coal dan surplus tiga kapasitas.

Lima TNT mengonsumsi 5 x 4 = 20 sand dan 5 x 5 = 25 gunpowder. Dua belas Eye of Ender mengonsumsi dua belas ender pearl dan dua belas blaze powder. Karena satu blaze rod menghasilkan dua powder, kebutuhan blaze rod adalah enam. Seluruh angka keluaran sesuai dengan ekspansi recipe manual.

F. Interpretasi Pareto Frontier

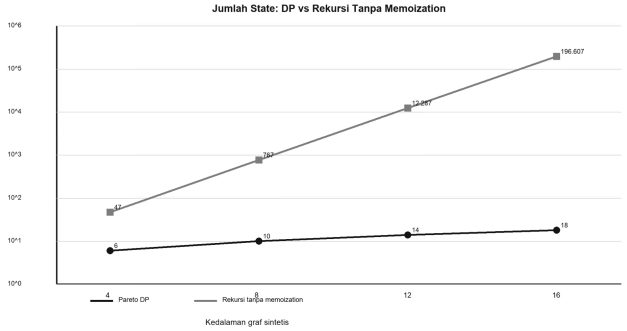
Jumlah plan Pareto adalah dua. Plan lava tidak dapat mendominasi plan obsidian karena menggunakan lava_source yang tidak digunakan plan kedua. Sebaliknya, plan obsidian mempunyai diamond dan obsidian yang tidak digunakan plan lava. Jika pengguna kelak memberikan bobot rarity atau waktu, salah satu plan dapat dipilih dari frontier tanpa menjalankan kembali pencarian recipe.

VII. EVALUASI ALGORITMA DAN KETERBATASAN

A. Benchmark Overlapping Subproblem

Dataset Minecraft utama relatif sempit sehingga DP dan baseline masing-masing mengevaluasi 55 state dan tidak menghasilkan memo hit. Kondisi tersebut dilaporkan apa adanya. Untuk menguji sifat algoritmik memoization, dibuat graf sintetis berjenjang dengan dua alternatif ekuivalen pada setiap level. Kedua cabang kembali ke submasalah level sebelumnya sehingga menghasilkan overlapping subproblem yang terkontrol.

Kedalaman	State DP	State Rekursif
4	6	47
8	10	767
12	14	12.287
16	18	196.607



7.1 Perbandingan Jumlah State

Pada kedalaman 16, baseline mengevaluasi sekitar 10.923 kali lebih banyak state daripada DP. Jumlah state DP bertumbuh linear karena setiap level unik dihitung satu kali. Baseline menghitung ulang subtree yang sama pada setiap cabang sehingga pertumbuhannya eksponensial.

B. Waktu Eksekusi

Median tiga pengulangan pada kedalaman 16 adalah sekitar 0,083 ms untuk DP dan 378,925 ms untuk rekursi tanpa memoization. Waktu DP pada input kecil berfluktuasi akibat warm-up dan optimasi JVM, sehingga jumlah state dan recursive call dipakai sebagai indikator utama. Metrik struktural tersebut lebih stabil dan langsung menggambarkan pekerjaan algoritma.

C. Kompleksitas

Tanpa memoization, jumlah kombinasi alternatif dapat tumbuh eksponensial terhadap kedalaman graf. Dengan memoization, setiap residual state unik diekspansi sekali. Kompleksitas tetap bergantung pada jumlah state yang mungkin dan ukuran Pareto frontier, tetapi pengulangan state identik dapat dihilangkan.

Pareto pruning membandingkan kandidat berdasarkan jumlah dimensi sumber daya. Pada frontier besar, biaya pruning dapat menjadi signifikan. Dalam dataset penelitian, frontier hanya berisi dua plan portal sehingga biaya tersebut jauh lebih kecil daripada biaya eksplorasi graf.

D. Keterbatasan Validitas

Eksperimen damage mengontrol health, posisi, rotasi, dan fase awal naga. Koordinat X dan Z payload terpilih berada di pusat blok, tetapi koordinat Y masih merupakan posisi ledakan entitas yang dikendalikan. Apparatus Survival yang terdiri atas rail, activator rail, redstone, dan blok penyangga belum dibangun serta belum dimasukkan ke cost model.

Hasil karena itu harus dibaca sebagai kebutuhan progresi dan isi payload pada kondisi eksperimen terkontrol, bukan tutorial mesin Survival yang telah tervalidasi penuh. Penelitian juga tidak memodelkan waktu, seed, probabilitas drop, makanan, armor, atau risiko kematian selama pengumpulan item.

E. Ancaman terhadap Kesimpulan

Lima ulangan cukup untuk screening awal tetapi belum mewakili seluruh variasi perilaku naga. Pembaruan versi Minecraft dapat mengubah damage, hitbox, atau fase. Selain itu, model jumlah drop menganggap pemain akhirnya memperoleh material yang dibutuhkan tanpa menghitung jumlah mob ekspektasian.

F. Makna Memo Hit Nol pada Dataset Utama

Memo hit nol pada dataset utama bukan kegagalan implementasi. Urutan pemilihan demand yang deterministik dan sedikitnya alternatif yang bergabung kembali membuat setiap residual state hanya muncul sekali. Memoization tetap memberikan jaminan bahwa perluasan dataset pada masa depan tidak menghitung state identik berulang.

Benchmark sintetis dipisahkan dari hasil Minecraft agar keuntungan DP tidak diklaim berasal dari data yang sebenarnya linear. Pelaporan kedua hasil secara terpisah memberi gambaran yang lebih jujur: kasus utama menunjukkan fungsi perencanaan, sedangkan benchmark menunjukkan skalabilitas algoritma.

G. Implikasi Praktis

Output Topological Sort dapat dipakai sebagai checklist progresi. Namun algoritma tidak menentukan lokasi penambangan atau strategi bertarung selama pengumpulan material. Seorang pemain tetap dapat mengubah urutan aksi independen sesuai kondisi dunia selama seluruh hubungan dependensi dipenuhi.

Model juga dapat digunakan untuk target lain, misalnya enchanting table atau beacon, cukup dengan menambahkan goal dan alternatif recipe. Algoritma inti tidak bergantung pada Ender Dragon atau satu jenis payload tertentu.

VIII. KESIMPULAN

Sistem progresi Minecraft dapat dimodelkan sebagai graf dependensi AND-OR. Program dinamis menentukan kombinasi aksi dan vektor sumber daya yang tidak terdominasi, sedangkan Topological Sort mengubah subgraf terpilih menjadi urutan eksekusi yang valid. Penyimpanan reusable item dan surplus output diperlukan agar alat maupun hasil batch tidak dihitung berulang.

Eksperimen memilih payload lima TNT minecart pada koordinat pusat blok dengan interval 12 tick. Konfigurasi tersebut membunuh Ender Dragon pada lima dari lima percobaan terkontrol. Jalur lava casting dari dunia baru membutuhkan 3 log, 11 cobblestone, 29 raw iron, 4 coal, 1 flint, 10 sumber lava, 6 blaze rod, 12 ender pearl, 20 sand, dan 25 gunpowder. Jalur mined obsidian menjadi solusi Pareto kedua dengan tambahan diamond dan obsidian.

Pada graf benchmark kedalaman 16, DP mengevaluasi 18 state, sedangkan rekursi tanpa memoization mengevaluasi 196.607 state. Perbedaan ini membuktikan manfaat memoization ketika overlapping subproblem muncul. Pada dataset utama yang lebih linear, memoization tidak memberikan pengurangan, tetapi model dan implementasi tetap siap menangani perluasan recipe yang lebih bercabang.

A. Saran Pengembangan

Pengembangan berikutnya dapat memvalidasi apparatus rail dan redstone secara penuh, memasukkan seluruh biayanya, meningkatkan jumlah ulangan, serta menggunakan beberapa kondisi awal naga. Model juga dapat diperluas dengan probabilitas drop, estimasi waktu, atau preferensi pemain untuk memilih satu plan dari Pareto frontier.

B. Jawaban terhadap Rumusan Masalah

Pertama, recipe Minecraft dapat direpresentasikan sebagai graf AND-OR dengan aksi sebagai alternatif dan item sebagai goal. Kedua, DP menghitung kebutuhan secara eksak dari residual state, bukan menggunakan heuristik rarity. Ketiga, Topological Sort menyusun aksi terpilih tanpa mengubah hasil optimasi.

Keempat, payload grid yang dipilih membutuhkan lima TNT minecart dan berhasil 5/5 pada kondisi awal terkontrol. Kelima, kebutuhan dari dunia baru dapat diturunkan sampai material mentah dan menghasilkan dua jalur Pareto. Keenam, memoization mengurangi pertumbuhan state secara drastis pada graf yang mempunyai overlapping subproblem.

C. Kontribusi Penelitian

Kontribusi penelitian terdiri atas model graf progresi yang membedakan consumed dan reusable requirement, state DP yang menyimpan surplus batch, integrasi Pareto frontier dengan Topological Sort, serta pengukuran payload langsung pada server permainan. Seluruh data dan keluaran disimpan dalam format yang dapat direproduksi.

VIII. LAMPIRAN

A. Link Github

https://github.com/fahdmz/MakalahStima-13524078

DAFTAR PUSTAKA

[1] Mojang Studios, Minecraft Java Edition, version 26.2, 2026.

[2] Minecraft Wiki, "Crafting," 2026. [Daring]. Tersedia: https://minecraft.wiki/w/Crafting.

[3] Minecraft Wiki, "Ender Dragon," 2026. [Daring]. Tersedia: https://minecraft.wiki/w/Ender_Dragon.

[4] Minecraft Wiki, "Minecart with TNT," 2026. [Daring]. Tersedia: https://minecraft.wiki/w/Minecart_with_TNT.

[5] T. H. Cormen, C. E. Leiserson, R. L. Rivest, dan C. Stein, Introduction to Algorithms, ed. ke-4. Cambridge, MA: MIT Press, 2022.

[6] R. Bellman, Dynamic Programming. Princeton, NJ: Princeton University Press, 1957.

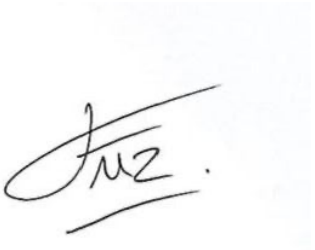
[7] A. B. Kahn, "Topological sorting of large networks," Communications of the ACM, vol. 5, no. 11, hlm. 558-562, 1962.

[8] K. Miettinen, Nonlinear Multiobjective Optimization. Boston, MA: Kluwer Academic Publishers, 1999.

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, 19 Juni 2026



Fahd Muhammad Zahid
13524078