Aplikasi Algoritma Branch and Bound Untuk Optimisasi Rute Perekrutan Vlandian Banner Knight pada Gim Mount and Blade 2: Bannerlord

Athif Nirwasito - 13521053
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): 13521053@std.stei.itb.ac.id

***Abstract*—Salah satu strategi dalam pembuatan komposisi prajurit dalam gim Mount and Blade 2: Bannerlord adalah dengan hanya merekrut prajurit-prajurit dari *noble line*. Namun, prajurit tipe hanya dapat ditemukan pada desa-desa tertentu. Dari implementasi yang dilakukan, ditemukan bahwa letak geografis dari desa-desa di Vlandia menyebabkan algoritma pencarian rute perekrutan Vlandian Banner Knight melambat. Penggunaan branch and bound untuk menyelesaikan masalah ini kurang dianjurkan. Meskipun itu, algoritma branch and bound tetap dapat menampilkan rute optimal untuk merekrut prajurit tersebut.**

Keywords—Branch and Bound, Traveling Salesman Problem, Hamilton Cycle, Optimisasi, Mount and Blade

I. PENDAHULUAN

Mount and Blade 2: Bannerlord adalah sebuah gim bergenre perperangan dan strategi pada platform PC dan console. Pada gim ini, pemain dapat membuat dan memainkan pemimpin sebuah tentara yang dapat bergerak kemana-mana pada peta. Dari tentara yang dimiliki, pemain dapat memilih untuk bermain sebagai perampok, jendral, raja, tentara bayaran, dan lain-lain sesuai bagaimana pemain ingin memainkan tokohnya.

Salah satu fitur dari gim ini adalah kemampuan untuk berperang. Pada sebuah perperangan, pemain akan memainkan karakternya untuk bertarung sekaligus memimpin pasukannya sedemikian rupa untuk mengalahkan musuhnya. Pemain memenangkan perperangan jika semua pasukan musuh dikalahkan atau moral musuh jatuh sehingga pasukan-pasukan musuh memilih untuk mundur.

Salah satu komponen untuk memenangkan perperangan adalah komposisi prajurit pada sebuah tentara yang dipimpin oleh pemain. Salah satu strategi dari komponen prajurit adalah dengan hanya merekrut prajurit dari *noble line*. Prajurit *noble line* ini merupakan prajurit elit bertipe khusus yang unik pada setiap fraksi.

Salah satu prajurit dari *noble line* adalah Banner Knights yang berasal dari fraksi Vlandia. Prajurit ini adalah prajurit tipe kavaleri yang memiliki senjata utama *lance*. Dengan perlengkapan dan persenjataan tersebut, sebuah kumpulan Banner Knights dapat menerobos formasi *shield wall* dan mampu memenangkan peraduan langsung dengan kavaleri-kavaleri lainnya.

Meskipun prajurit ini, memiliki kelebihan dibandingkan dengan prajurit lainnya, prajurit bertipe *noble line* hanya dapat direkrut dari desa-desa dibawah sebuah kastil. Karena kelangkaan ini, pemain harus mengunjungi desa-desa yang lebih tersebar pada peta sehingga kecepatan untuk regenerasi tentara setelah sebuah perperangan akan memakan waktu yang lebih lama dari proses rekrutmen biasa. Untuk itu diperlukan sebuah rute yang efisien untuk mengunjungi desa-desa tersebut.

II. LANDASAN TEORI

A. Algoritma Branch and Bound

Algoritma branch and bound adalah salah satu algoritma yang dapat digunakan untuk memecahkan masalah optimisasi. Ide dari algoritma ini mirip dengan algoritma BFS atau DFS, yaitu dengan membangkitkan pohon ruang status dengan urutan ekspansi tertentu.

Pada algoritma branch and bound, setiap simpul diberi sebuah nilai cost

ĉ (i) = nilai taksiran cost

Nilai cost didapatkan dengan menaksir lintasan termurah dari simpul status ke simpul tujuan melalui simpul ke-i. Nilai cost ini akan digunakan sebagai urutan ekspansi pada pohon pencarian. Selain itu, nilai cost juga dapat dipakai sebagai fungsi pembatas. Caranya jika nilai taksiran cost memiliki nilai yang lebih buruk dari nilai cost solusi yang didapatkan sejauh ini, maka simpul tersebut dapat dipangkas. Secara umum algoritma Branch and Bound memilikki langkah-langkah sebagai berikut:

1. Masukkan simpul akar ke dalam antrian Q. Jika simpul akar adalah solusi, maka sebuah solusi ditemukan.
2. Jika Q kosong, berhentikan pencarian.
3. Jika Q tidak kosong, ambil satu simpul missal i dari antrian Q, yaitu simpul yang memiliki nilai cost paling

optimal. Jika terdapat 2 simpul yang memenuhi, pilih secara sembarang

4. Jika simpul i adalah simpul solusi, matikan seluruh simpul hidup yang memiliki cost lebih besar dari cost simpul solusi.
5. Jika simpul i bukan simpul solusi, bangkitkan anak-anaknya. Jika tidak memiliki anak, kembali ke langkah 2.
6. Untuk setiap anak dari simpul i, hitung costnya dan masukkan anak-anak tersebut ke dalam Q.
7. Kembali ke langkah 2.

B. *Mount and Blade 2: Bannerlord*

1) Gameplay

Pemain bermain sebegai sebuah tokoh yang telah dibuatnya. Pemain dapat berjalan seputar map dan berinteraksi dengan tokoh-tokoh lain pada gim atau berinteraksi dengan *settlement*. Pemain dapat menyerang tokoh atau settlement lain, disaat yang sama diserang oleh tokoh lain baik secara langsung atau di settlement yang dimiliki oleh pemain. Jika terjadi suatu penyerangan, maka akan dimulai perperangan yang melibatkan karakter dan prajurit-prajurit dari pemain.

2) Settlements

Dalam gim Mount and Blade 2: Bannerlord, settlement adalah wilayah yang dapat dikuasai oleh pemain atau pemimpin-pemimpin lainnya. Pemimpin manapun dapat melakukan aksi pada sebuah settlement yang tidak dimilkinya. Akan tetapi, hanya pemilik *settlement* yang hanya bisa membangun infrastruktur, menerima pajak, dan menukar prajurit di *garrison.* Di dalam gim ini, terdapat 3 jenis settlement:

a) Town

Gambar 1. Kota Ocs Hall.

Town atau perkotaan adalah settlement utama yang menjadi pusat perekonomian pada gim ini. Secara umum, pada sebuah town, pemain dapat rekrut prajurit, dagang, dan lain-lain.

b) Castle

Gambar 2. Kastil Verecsand

Castle atau kastil adalah settlement yang mempunyai fungsi khusus untuk militer bagi pemiliknya. Bagi pemilik, kastil dapat digunakan untuk menukar dan menyimpan prajurit dengan *garrison* kastil tersebut. Semua pemimpin pada gim termasuk pemilik kastil tidak dapat merekrut prajurit apapun dari kastil.

c) Village

Gambar 3. Desa Marin

Village atau desa adalah *settlement* terkecil dalam gim Mount and Blade 2: Bannerlord. Di desa, pemimpin hanya bisa melakukan dagang, merekrut prajurit, menyerang desa tersebut dan menerima pajak desa khusus untuk pemilik. Setiap desa pasti *bounded* atau terikat dengan satu kastil atau satu kota.

3) Prajurit (Troop)

Pada gim Mount and Blade 2: Bannerlord, setiap fraksi memiliki troop yang unik. Pada setiap fraksi, setidaknya terdapat satu *infantry*, *ranged*, dan *mounted line*, serta satu *noble line.* Prajurit dari fraksi tertentu dapat direkrut dari *settlement* yang memiliki budaya dari fraksi tersebut.

Setiap prajurit tergolong pada suatu *tier* dan dapat ditingkat *tiernya* hingga *tier* maksimum. Untuk prajurit biasa dapat dimiliki dan dinaikkan dari *tier 1* hingga *tier 5*, sedangkan untuk prajurit dalam *noble line* dapat dimiliki dan dinaikkan dari *tier 2* hingga tier *6.* Prajurit-prajurit tersebut dapat direkrut dari orang-orang penting dari kota atau desa. Khusus untuk prajurit dari *noble line* hanya bisa direkrut dari desa-desa yang terikat dengan kastil sehingga prajurit jenis tersebut lebih sukar untuk direkrut dibandingkan dengan prajurit biasa.

Gambar 4. Contoh antarmuka prajurit dari desa yang terikat kastil. Baris paling menjual prajurit dari *line* biasa sedangkan dua di atasnya menjual prajurit dari *noble line.*

III. PEMBAHASAN

A. Implementasi

Untuk meneyelesaikan persoalan ini ke dalam algoritma branch and bound, maka perlu ditentukan dulu *cost* yang diperlukan untuk mengunjungi dari desa ke desa lain. Untuk implementasi ini, cost ditentukan waktu yang ditempuh untuk berangkat dari suatu desa ke desa lainnya pada kecepatan 8.2. Karena pada gim ini terdapat sistem yang memperlambat gerak tentara pada malam hari, maka pengukuran dari desa ke desa dilakukan pada waktu pagi di gim tepat saat kecepatan tentara pemain kembali seperti semula. Untuk faktor-faktor penghambat lainnya seperti hutan akan diabaikan karena perlembatan tersebut merupakan perlambatan yang akan selalu ditemui ketika melewati daerah tersebut.

Gambar 5. Peta desa-desa di Vlandia yang menyediakan unit pada *noble line*

Prajurit Vlandian Banner Knight dapat direkrut dari 14 desa: Caleus, Deriat, Tirby, Sirindac, Ormafand, Valanby, Drapan, Verecsand, Marin, Rodetan, Hongard, Ferton, Talivel, dan Usanc. Namun, dapat dilihat bahwa terdapat desa yang jauh terisolasi dari desa-desa yang lain, yaitu Usanc. Maka dari itu, desa ini akan dikeluarkan dari pencarian rute.

Desa yang akan menjadi titik awal pada penerapan kali ini adalah desa Caleus. Desa ini terletak berdekatan dengan wilayah Battania (hijau) dan merupakan tempat terdekat untuk merekrut prajurit dari *troop line* Banner Knight.

Untuk mempermudah pencarian, penyelesaian dari permasalahan ini akan diselesaikan menggunakan python. Algoritma yang digunakan pada program kurang lebih sama dengan algoritma branch bound untuk menyelesaikan persoalan traveling salesmen problem, hanya saja terdapat beberapa perbedaan. Branch and bound diimplementasikan menggunakan priority queue. Setiap elemen priority queue, menyimpan data *adjecancy matrix*, node yang akan dilalui, rute yang telah dilalui serta cost dari simpul tersebut. Nilai tak hingga digantikan dengan dua cara, yang pertama membuat suatu baris atau kolom bernilai tak hingga dengan menghapus baris atau kolom tersebut, yang kedua dengan menggantikannya dengan nilai 999999 untuk menggantikan nilai tak hingga sementara pada kolom start.

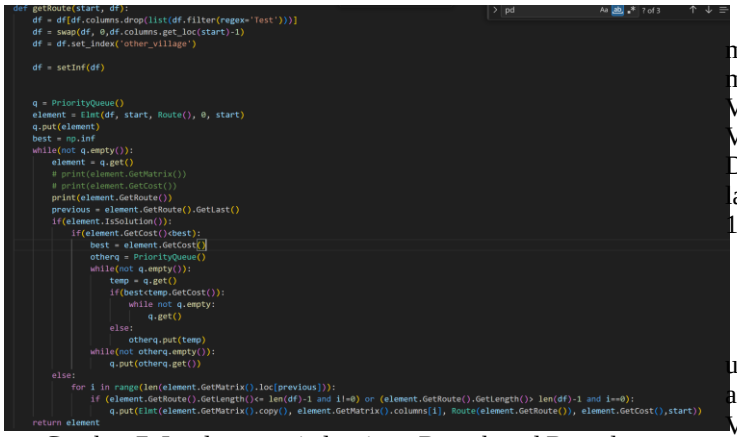
```
class Elmt:
    def __init__(self,  matrix, node,rute : Route, cost, start):
        self.matrix = matrix
        self.node = node
        self.rute = rute

        if(cost == 0):
            self.rute.addNode(node)
            self.cost = self.calculateCost(cost)
        else:
            realCost = self.matrix.at[self.rute.GetLast(),self.node]
            self.matrix = self.matrix.drop(node, axis = 1)
            self.matrix = self.matrix.drop(self.rute.GetLast(), axis = 0)
            if(self.node!= start):
                temp = self.matrix.at[self.node,start]
                self.matrix.at[self.node,start] = 999999


            if len(self.matrix) ==1:
                self.cost = cost + realCost
            else:
                self.cost = self.calculateCost(cost) + realCost
            if(self.node!= start):
                self.matrix.at[self.node,start] = temp
            self.rute.addNode(node)
    def GetNode(self):
        return self.node
    def GetRoute(self):
        return self.rute
    def GetCost(self):
        return self.cost
    def GetMatrix(self):
        return self.matrix
    def IsSolution(self):
        return self.matrix.empty
    def calculateCost(self, cost):
        minRow = self.subtractRow()
        minColumn = self.matrix.min()
        self.matrix = self.matrix.subtract(minColumn ,axis=1)
        return minRow + minColumn.sum()+ cost
    def subtractRow(self):
        total = 0
        for i in range(len(self.matrix)):

            min = self.matrix.at[self.matrix.index[i],self.matrix.columns[0]]
            for j in range(1, len(self.matrix.columns)):
                temp = self.matrix.at[self.matrix.index[i],self.matrix.columns[0]]
                if(min>self.matrix.at[self.matrix.index[i],self.matrix.columns[j]]):
                    min = self.matrix.at[self.matrix.index[i],self.matrix.columns[j]]
            for j in range(len(self.matrix)):
                self.matrix.at[self.matrix.index[i],self.matrix.columns[j]] -= min
            total += min
        return total
```

Gambar 6. Implementasi elemen Priority Queue



```
def getRoute(start, df):
    df = df[df.columns.drop(list(df.filter(regex='Test')))]
    df = swap(df, 0,df.columns.get_loc(start)-1)
    df = df.set_index('other_village')

    df = setInf(df)


    q = PriorityQueue()
    element = Elmt(df, start, Route(), 0, start)
    q.put(element)
    best = np.inf
    while(not q.empty()):
        element = q.get()
        # print(element.GetMatrix())
        # print(element.GetCost())
        print(element.GetRoute())
        previous = element.GetRoute().GetLast()
        if(element.IsSolution()):
            if(element.GetCost()<best):
                best = element.GetCost()
                otherq = PriorityQueue()
                while(not q.empty()):
                    temp = q.get()
                    if(besttemp.GetCost()):
                        while not q.empty:
                            q.get()
                    else:
                        otherq.put(temp)
                while(not otherq.empty()):
                    q.put(otherq.get())
        else:
            for i in range(len(element.GetMatrix().loc[previous])):
                if (element.GetRoute().GetLength()<= len(df)-1 and i!=0) or (element.GetRoute().GetLength()> len(df)-1 and i==0):
                    q.put(Elmt(element.GetMatrix().copy(), element.GetMatrix().columns[i], Route(element.GetRoute()), element.GetCost(),start))
    return element
```

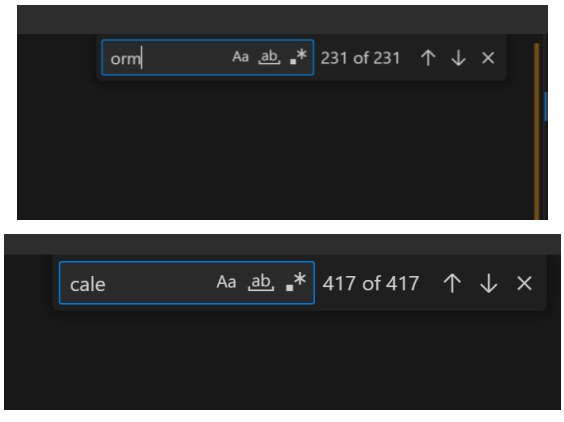
Gambar 7. Implementasi algoritma Branch and Bound

B. Hasil

Dari program yang dibuat, program memakan waktu yang cukup lama untuk menyelesaikan masalah ini. Program memakan waktu sekitar 3 menit. Kelamaan ini diakibatkan oleh kebutuhan algoritma untuk mencari nilai minimum pada setiap baris dan kolom serta mengurangi nilai minimum tersebut untuk setiap baris dan kolom. Selain itu, proses tersebut dilakukan berulang kali pada sebuah ekspansi.

Pada implementasi ini kemungkinan untuk setiap simpul melakukan ekspansi cukup besar. Ambil kasus dari desa Caleus menuju desa Marin dan desa Verecsand. Desa marin dan desa Verecsand merupakan dua desa yang berdekatan. Akibatnya kemungkinan untuk melakukan ekpansi dua kali pada desa yang sama dengan urutan yang berbeda cukup besar.

Dari hasil yang didapatkan ditemukan desa Ormonfard muncul 231 dari 417 kemunculan desa Caelus dari command line. Desa tersebut muncul hampir 50% dari semua simpul akbiat dari letaknya yang terletak di antara desa Caelus dan Deriat dengan desa-desa lainnya.. Akibatnya, jika kita ingin pergi menuju desa lain dari Caleus atau Deriat, setidaknya kita akan mengunjungi Ormanfand. Hal ini menimbulkan tingginya permutasi pada pencarian solusi.



Gambar 8. Perbandingan kemunculan desa Ormafand dengan kota Caelus

Walaupun efisiensi branch and bound kurang efisien, untuk menyelesaikan masalah ini, algoritma branch and bound dapat menemukan rute optimal untuk merekrut Banner Knight sekitar Vlandia. Urutan tersebut adalah Caleus, Deria, Ormanfard, Verecsand, Marin, Rodetan, Talivel, Ferton, Hongard, Drapand, Valanby, Sirindac, Tirby, dan kembali ke Caleus lagi. Untuk mengelilingi desa-desa tersbut, dibutuhkan waktu 160.8282 detik pada kecepatan 8.2.

IV. Kesimpulan

Implementasi branch and bound dapat digunakan untuk mencari rute perekrutan Banner Knight pada gim Mount and Blade 2: Bannerlord. Meskipun itu, karena letak geografis Vlandia, algoritma ini masih dapat dioptimisasi lagi menjadi algoritma yang lebih efisien.

V. Apendiks

Implementasi algoritma branch and bound dapat dilihat pada pranala berikut:

https://github.com/Onyxcodeotto/MakalahStima

Untuk data dari desa ke desa dapat dilihat pada vlandia_time_speed.xlsx.

VI. Ucapan Terima Kasih

Puji dan syukur penulis panjatkan kepada Tuhan yang Maha Esa atas segala rahmat dan hidayahnya sehingga makalah yang berjudul “Aplikasi Algoritma Branch and Bound Untuk Optimisasi Rute Perekrutan Vlandian Banner Knight pada Gim Mount and Blade 2: Bannerlord” dapat diselesaikan walaupun tidak memenuhi spesifikasi. Penulis mengucapkan terima kasih untuk seluruh. Penulis juga mengucapkan terima kasih kepada teman-teman yang baik secara langsung atau tidak langsung membantu penulis dalam menyusun makalah ini. Tidak lupa, penulis mengucapkan terima kasih kepada orang tua dan keluarga atas doa dan dukungannya sehingga penulis dapat menghadapi segala hambatan yang ada dalam penyusunan makalah ini.

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, 22 Mei 2023

Athif Nirwasito - 13521053

References

[1] https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Branch-and-Bound-2021-Bagian2.

[2] https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Branch-and-Bound-2021-Bagian3.

[3] https://www.geeksforgeeks.org/branch-and-bound-algorithm/.

[4] https://www.geeksforgeeks.org/traveling-salesman-problem-using-branch-and-bound-2/