LAPORAN TUGAS BESAR 1

Minimax Algorithm and Alpha Beta Pruning in Simplexity Board Game

Mata Kuliah IF3170 Intelegensi Buatan

Dosen Pengampu: Nugraha Priya Utama, Nur Ulfa Maulidevi,

Fariska Zakhralativa Ruskanda, Anggrahita Bayu Sasmita

Disusun Oleh:

Billy Julius	13519094
Girvin Junod	13519096
Jonathan C.J.	13519144
Kevin Ryan	13519191

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2021

Proses Pencarian dengan Algoritma Minimax dan *Alpha-Beta Pruning*

Pada algoritma ini, akan di-*generate tree* di setiap ronde permainan dengan *root node* merupakan *state board* pada ronde tersebut. Anak dari suatu *node* melambangkan situasi di mana pemain mengambil suatu langkah. Oleh karena itu, anak dari setiap *node* berjumlah banyaknya langkah yang bisa diambil oleh seorang pemain pada suatu *state*. Pada bot yang diimplementasikan, *player* yang gilirannya sedang berlangsung adalah *maximizer* dan lawannya dianggap *minimizer*. Pada *tree* yang dibuat, tiap perpindahan kedalaman melambangkan perpindahan giliran bergerak dari pemain. Jadi jika *state* awal merupakan giliran pemain *shape* O yang mencari nilai maksimum, pada kedalaman selanjutnya di *tree* melambangkan giliran dari lawannya yaitu pemain *shape* X yang mencari nilai minimum.

Setiap *node* memiliki nilai objektif atau *objective value* yang merepresentasikan seberapa dekat dengan kemenangan *state* yang akan dihasilkan jika *piece* ditaruh di kolom tertentu. Nilai objektif dari *node* daun dihitung dengan suatu *objective function* yang menghitung nilainya dari berapa banyak pasang *shape* dan *color* player yang sama yang terletak berdampingan jika suatu *shape* ditaruh di suatu kolom dikurangi dengan pasang *shape* dan *color* musuh yang sama yang terletak berdampingan. Juga diperhitungkan jika suatu *streak* diakhiri dengan shape atau color yang tidak sesuai dengan streak tersebut, maka nilai dari *streak* tersebut akan diubah menjadi nol karena sudah tidak dapat dilanjutkan lagi untuk menang. Nilai objektif untuk *node* yang bukan *node* daun dihitung dengan melakukan perbandingan antara nilai objektif dari *node* anaknya dan dipilih nilai maksimum atau minimum sesuai dengan apakah itu *node maximizer* atau *minimizer*.

Jadi, program akan membuat suatu *tree* minimax dengan kedalaman yang telah ditentukan di awal. Lalu akan dicari nilai objektif dari tiap *node* mulai dari *node* daun sampai ke *node* kedalaman 1 melalui objective function atau pencarian nilai maksimum atau minimum dari *node* anak. Pada akhirnya, akan dipilih *node* pada kedalaman 1 dengan nilai objektif terbaik sebagai langkah yang akan diambil oleh pemain. Karena pemain selalu mencari nilai objektif maksimal, maka dipilih langkah yang memiliki nilai objektif maksimal.

Alpha-beta pruning dilakukan pada algoritma minimax untuk memangkas banyaknya *node* yang dicek. *Alpha-beta pruning* dilakukan dengan mengecek apakah masih perlu untuk melakukan perhitungan lebih lanjut terhadap anak-anak dari suatu *node* atau sudah bisa dihentikan saja karena nilai akhir dari *node* tersebut akan sama saja. *Alpha* dalam *alpha-beta pruning* memiliki arti nilai terbaik yang dapat dijamin *node maximizer* pada tingkatnya. *Beta* memiliki arti nilai terbaik yang dapat dijamin *node minimizer* pada tingkatnya. Pada awalnya, nilai *alpha* dibuat *-infinity* dan *beta +infinity* untuk tiap *node*. *Pruning* terjadi setiap kali nilai $beta \leq alpha$.

Jadi kalau misal suatu *node* A memiliki anak berupa *node* B dan C. *Node* B dan C masing-masing memiliki anak 2 *node* yang merupakan node daun bernilai 1, 2, 3, dan 4. Misal A adalah *node minimizer* dan *node* B serta C *node maximizer*. *Node* A harus memilih nilai minimum dari B dan C. Untuk itu, dipilih untuk meninjau *node* B terlebih dahulu. *Node* B memiliki 2 anak *node* dengan nilai 1 dan 2. Pada peninjauan *node* dengan nilai 1, alpha dari B dan A berubah menjadi 1. Belum dipenuhi $beta \leq alpha$ maka tidak bisa dilakukan *pruning*. Maka ditinjau anak *node* B dengan nilai 2, maka nilai B menjadi 2 dan *alpha* untuk B menjadi 2 dan beta untuk A menjadi 2 juga. Setelah itu ditinjau *node* C. *Node* C memiliki nilai *alpha* *-infinity* dan *beta* bernilai 2 karena *node* parent A yang mempunyai *beta* sebesar 2 serta anak

node dengan nilai 3 dan 4. Ditinjau anak *node* dengan nilai 3, *alpha* dari C menjadi 3. Ini membuat terpenuhinya syarat *beta* ≤ *alpha* di *node* C, dengan nilai *beta* 2 dan nilai *alpha* 3. Oleh karena itu dilakukan *pruning* pada sisa cabang dari *node* C yang belum ditinjau. Dalam kata lain, jika pada peninjauan *node* B sudah dapat dibilang bahwa *node* A pasti akan memilih *node* B dan diketahui nilai dari *node* B, maka *node* C tidak perlu lagi ditinjau dan bisa di-*prune*. Ini berlaku untuk *tree* dengan berbagai kedalaman dan *pruning* dapat dilakukan pada setiap kedalaman selain *node* akar.

Implementasi *alpha-beta pruning* pada algoritma minimax di program kurang lebih sama dengan contoh di atas. Perbedaannya adalah nilai pada *node* daun di *tree* akan didapat dari penggunaan *objective function* pada *state* di *node*. Penggunaan *alpha-beta pruning* akan mengurangi lamanya waktu dalam pemrosesan algoritma minimax dengan mengurangi perbandingan nilai dan pemanggilan *objective function* untuk mendapat nilai heuristik di *node* daun.

Pada algoritma minimax yang digunakan, ditentukan kedalaman tree untuk algoritma minimax adalah 4 tingkat, yang berarti ada kedalaman 0 sampai 3. Hal ini ditentukan berdasarkan alasan *thinking time* yang bernilai 3 detik, sehingga kedalaman sejauh 4 dinilai sesuai dengan waktu berpikir yang diberikan.

Algoritma Local Search yang Digunakan

Pencarian dengan *local search* dilakukan dengan algoritma *Hill-Climbing Steepest Ascent*. Pencarian ini dilakukan setiap giliran pemain. Pada awal giliran, dilakukan pemilihan *piece* dan kolom secara acak untuk *state* awal dari kolom dan *piece* yang masih tersedia. Dari *state* awal ini, dilakukan pencarian terhadap lokal maksimum sebagai solusi akhir dari *local search*. Pertama, dihitung nilai heuristik untuk semua kandidat solusi. Kandidat solusi di sini merupakan semua kombinasi kolom dan *piece* yang bisa dilakukan oleh pemain. Penghitungan nilai heuristik ini dilakukan melalui fungsi heuristik. Setelah menghitung nilai heuristik semua kandidat solusi, maka berpindah *state* ke kandidat solusi yang memiliki nilai heuristik terbaik untuk pemain. Proses ini diulang sampai menyampai lokal maksimum dari nilai heuristik atau waktu 3 detik telah berlalu. Solusi akhir dari *local search*-nya merupakan lokal maksimumnya jika ada atau dipilih secara acak jika tidak ditemukan lokal maksimum.

Dalam menghitung nilai heuristik, digunakan suatu *objective function*. *Objective function* ini menghitung banyaknya *streak shape* atau *color* yang ada di *board* dari suatu state. Dari streak yang ada ini, nilai heuristik dari suatu state akan ditambah jika *streak* sesuai dengan *shape* atau *color* dari pemain yang gilirannya sedang berlangsung. Jika tidak sesuai, maka nilai heuristik akan dikurangi karena *streak* tersebut berarti sesuai dengan *shape* atau *color* dari lawan pemain. Nilai heuristik dikurangi dari *streak* lawan arena permainan berupa suatu *zero sum game*, maka semakin dekat musuh dengan kemenangan semakin dekat pemain dengan kekalahan dan juga sebaliknya. *Objective function* juga mengecek jika *streak* telah diakhiri dengan *shape* atau *color* yang tidak sesuai, seperti *streak shape* X yang diakhiri dengan *shape* O. Ini karena *streak* tersebut sudah tidak mungkin berakhir dalam kemenangan maka nilai heuristik dari streak tersebut dihitung nol.

Sebenarnya, karena nilai heuristik tiap kandidat solusi tidak dipengaruhi oleh pilihan kolom dan bentuk *piece* pemain, maka sebenarnya dari solusi awal sudah dapat dihitung nilai heuristik dari semua kemungkinan langkah yang bisa diambil, jadi hanya perlu melakukan satu perpindahan *state* saja untuk menemukan solusi optimal.

Hasil Pertandingan

Bot Minimax vs Manusia

No. Pertandingan	Hasil
1	Bot Menang
2	Bot Menang
3	Bot Menang
4	Bot Menang
5	Bot Menang
Total	5 Kemenangan Bot Minimax

Persentase kemenangan bot minimax = 100%

Bot minimax memenangkan kelima pertandingan yang dilakukan terhadap manusia. Kemenangan ini dikarenakan bot dapat melihat lebih jauh dari manusia dalam memperhitungkan langkahnya. Ini juga karena manusia terkadang lalai dalam melihat langkah yang baik dan langkah yang buruk.

Bot *Local Search* vs Manusia

No. Pertandingan	Hasil
1	Manusia Menang
2	Manusia Menang
3	Bot Menang
4	Bot Menang
5	Bot Menang
Total	3 Kemenangan Bot *Local Search*

Persentase kemenangan bot *local search* = 60%

Bot local search memenangkan tiga dari lima pertandingan melawan manusia. Ini dikarenakan kemampuan bot untuk memperhitungkan nilai heuristik dari suatu langkah dengan jauh lebih

teliti dan baik dibanding manusia. Manusia masih rawan untuk gagal memperhitungkan seberapa baik atau buruk langkahnya dengan baik dan konsisten.

Bot Minimax vs Bot *Local Search*

No. Pertandingan	Hasil
1	Bot Minimax
2	Bot Minimax
3	Bot Minimax
4	Bot Minimax
5	Bot Minimax
Total	5 Kemenangan Bot Minimax

Persentase kemenangan bot minimax = 100%

Persentase kemenangan bot *local search* = 0%

Bot minimax memenangkan kelima pertandingan melawan bot *local search*. Ini dikarenakan bot minimax memiliki kemampuan untuk juga memperhitungkan gerakan musuh dan melihat ke depan. Sementara itu, bot *local search* hanya bisa untuk melihat dan memperhitungkan langkah untuk *state* saat itu saja, tidak melihat kedepannya. Ini membuat bot minimax lebih baik dan membuatnya dapat memenangkan semua pertandingan.

Total

Persentase kemenangan bot minimax = 10/10 = 100%

Persentase kemenangan bot *local search* = 3/10 = 30%

Persentase kemenangan manusia = 2/10 = 20%

Pembagian Tugas

No	NIM	Nama	Kontribusi
1	13519094	Billy Julius	Penentuan objective function, Penentuan local search, implementasi minimax alpha beta pruning, testing
2	13519096	Girvin Junod	Penentuan objective function, Penentuan local search, implementasi local search, laporan, testing
3	13519144	Jonathan Christhoper Jahja	Penentuan objective function, Penentuan local search, implementasi objective function, testing
4	13519191	Kevin Ryan	Penentuan objective function, Penentuan local search, implementasi objective function, testing