MODEL DAN SIMULASI PENCARIAN SOLUSI PERLINTASAN JALAN SEBUAH MOBIL PADA DAERAH TERISOLASI BENCANA MENGGUNAKAN ALGORITMA BACKTRACKING

*Anderias Eko Wijaya[*1], Lukman Maulana[#2]*

Program Studi Teknik Informatika, STMIK Subang
Jl. Marsinu No. 5 - Subang, Tlp. 0206-417853 Fax. 0206-411873
E-mail: ekowjy09@yahoo.com[*1,] lukman89@gmail.com[#2]

ABSTRAKSI

Algoritma runut balik (backtracking) dan penerapannya dalam pencarian solusi perlintasan sebuah mobil adalah algoritma yang berbasis pada DFS (Depth First Search) untuk mencari solusi persoalan secara sistematis di antara semua kemungkinan solusi yang ada. Hanya pencarian yang mengarah ke solusi saja yang selalu dipertimbangkan sehingga waktu pencarian dapat dihemat. Mencari perlintasan mobil merupakan salah satu alat simulasi,dimana sebuah mobil dapat mencari jalan mobil yang terdekat dan tercepat untuk mencapai tujuan dan mengetahui perlintasan yang dapat dilalui sebuah mobil dengan rintangan – rintangan yang bisa dilewati oleh sebuah mobil.

Kata kunci: ***Algoritma, Backtracking, Depth First Search, Artificial Intelligence***

1. Pendahuluan

Algortima backtracking merupakan salah satu yang bisa diterapkan untuk menyelesaikan persoalan pada bidang ilmu *Artificial Intelligence* (AI). Dimana algoritma backtracking ini akan secara sistematis mencari solusi persoalan di antara semua kemungkinan solusi yang ada kemudian membangkitkan simpul dari solusi yang mendekati penyelesaian. Dengan kata lain kita tidak perlu membangkitkan simpul yang menjauhi solusi sehingga waktu pencarian solusi dapat dikurangi.Dalam persoalan ini algoritma backtracking bisa mencari solusi dalam perlintasan sebuah mobil, untuk mencari perlintasan yang terdekat dalam mencapai tujuan dan mengetahui perlintasan - perlintasan yang bisa di lalui sebuah mobil dengan rintangan – rintangan yang ada.

Persoalan ini dapat didekripsikan sebagai berikut, diketahui sebuah mobil akan bergerak dari titik pusat (0,0) ke titik A(m,n). mobil tersebut hanya boleh membelok pada titik-titik *grid* dan selalu melangkah sejajar dengan sumbu-x atau sumbu-y. Mobil tersebut tidak boleh melintasi lintasan yang telah pernah dilaluinya dan tidak boleh melintasi titik yang telah pernah dilaluinya. Setelah itu, disediakan sederetan ketentuan yang membatasi pergerakan mobil tersebut. Pertanyaannya adalah bagaimana bentuk lintasan-lintasan yang dapat dilalui oleh mobil tersebut dengan menggunakan ketentuan-ketentuan yang telah ditetapkan di atas.

2. Tinjauan Pustaka

2.1. *Algoritma Backtracking*

Algoritma backtracking pertama kali diperkenalkan oleh D.H. Lehmer pada tahun 1950. Algoritma ini cukup praktis untuk digunakan dalam beberapa penyelesaian masalah dan juga untuk memberikan kecerdasaan buatan dalam game. Beberapa game populer semisal Sudoku, Labirin, Catur,pencarian jalan,tic tac toe juga bisa di implementasikan dengan menggunakan algoritma backtracking. Algoritma runut-balik (backtracking) merupakan algoritma yang diguna- kan untuk mencari solusi persoalan secara lebih praktis dari pada menggunakan algoritma brute force. Algoritma ini akan mencari solusi berdasarkan ruang solusi yang ada secara sistematis namun tidak semua ruang solusi akan diperiksa, hanya pencarian yang mengarah kepada solusi yang akan diproses (Rinaldi Munir, Diktat Srategi Algoitmik, Teknik Informatika ITB, 2005).

Kelemahan dari algoritma backtracking, yaitu hanya bisa diaplikasikan terbatas pada tipe permasalahan yang memiliki solusi yang dapat dicari secara sistematis dan bertahap. Terdapat masalah-masalah yang tidak bisa diselesaikan dengan menggunakan backtracking, misalnya menemukan suatu nilai yang diminta pada tabel yang tidak terurut. Namun ketika algoritma ini dapat diaplikasikan,backtracking dapat bekerja jauh lebih cepat dari brute force karena jumlah kandidat solusi yang dapat dibuang dengan backtracking cukup besar.

Algoritma backtracking merupakan algoritma pencarian yang berbasis pada DFS (Depth-First Search) atau pencarian mendalam dengan tujuan mencari solusi permasalahan secara lebih praktis. Mekanisme penyelesaian dengan menggunakan backtracking berprinsip pada metode rekursif. Untuk menyelesaikan keseluruhan masalah, dibutuhkan sebuah solusi untuk permasalahan pertama, kemudian permasalahan-permasalahan lainnya akan dicoba untuk diselesaikan secara rekursif berdasarkan solusi pertama tersebut. Jika kemungkinan solusi yang sedang dicoba gagal, atau jika tujuan program adalah untuk menemukan seluruh solusi yang mungkin, maka dilakukan backtrack untuk menguji kemungkinan solusi selanjutnya. Proses backtrack akan selesai ketika tidak ada lagi solusi yang mungkin untuk menyelesaikan permasalahan paling awal. Dalam diktat Strategi Algoritmik Teknik Informatika ITB oleh Rinaldi Munir, dijelaskan bahwa algoritma backtracking memiliki properti umum yaitu:

• Solusi persoalan

Solusi dinyatakan sebagai vektor dengann-tuple:

X = (x1, x2, …, xn), xi∈ Si

Mungkin sajaS 1 = S 2 = … = S n.

Contoh: Si = {0, 1}, xi = 0 atau 1

• Fungsi pembangkit nilaixk

Dinyatakan sebagaiT (k).
T(k) membangkitkan nilai untuk xk, yang merupakan komponen vektor solusi.
• Fungsi pembatas (pada beberapa persoalan fungsi ini dinamakan fungsi kriteria)

Dinyatakan sebagai B(x1, x2, …, xk),B bernilaitr ue jika (x1, x2, …, xk) mengarah ke solusi. Jikatrue, maka pembangkitan nilai untukxk+1 dilanjutkan, tetapi jikafals e, maka (x1, x2,…, xk) dibuang dan tidak dipertimbangkan lagi dalam pencarian solusi.

Solusi persoalan adalah kemungkinan solusi yang didapatkan dari permasalahan yang diberikan, sedangkan fungsi pembatas merupakan fungsi yang akan menentukan langkah selanjutnya berupa penerusan pencarian solusi ataupun melakukan backtrack. Algoritma backtracking merupakan metode paling efisien untuk parsing dan banyak masalah optimasi kombinatorial lainnya.Backt racki ng juga digunakan oleh bahasa pemrograman logika seperti Prolog,Icon, dan Planner.

2.2. Prinsip Pencarian Solusi pada *Algoritma Backtracking*

Untuk menerapkan algoritma runut-balik pada pencarian solusi, hanya akan ditinjau pencarian solusi pada pohon ruang status yang dibangun secara dinamis:

1. Solusi dicari dengan membentuk lintasan dari akar ke daun dengan aturan pembentukan pohon yang dipakai mengikuti metode DFS. Simpul-simpul yang sudah dilahirkan dinamakan simpul hidup. Simpul hidup yang sedang diperluas dinamakan simpul-E (Expand-node).
2. Jika lintasan yang sedang dibentuk tidak mengarah ke solusi, maka simpul- E menjadi simpul mati (dead node) dengan fungsi pembatas (bounding function
3. Jika pembentukan lintasan berakhir dengan simpul mati, maka proses pen- carian diteruskan dengan membangkitkan simpul anak yang lain. Bila tidak ada lagi simpul anak yang dapat dibangkitkan, maka pencarian solusi dilanjutkan dengan melakukan runut-balik ke simpul hidup terdekat.
4. Pencarian dihentikan bila kita telah menemukan solusi (solusi ditemukan) atau tidak ada lagi simpul hidup untuk runut-balik (solusi tidak ditemukan).

Salah satu fungsi yang dimiliki oleh algoritma backtracking dan menjadi ciri khasnya adalah fungsi pemangkasan (pruning). Jika tahap-tahap pencarian solusi suatu masalah direpresentasikan dalam bentuk pohon solusi, proses pemangkasan akan dilakukan terhadap simpul-simpul yang tidak mengarah kepada solusi. Jika suatu simpul telah dipangkas, simpul-simpul yang menjadi anak dan turunan dari simpul tersebut otomatis tidak akan diproses, karena memangkas sebuah simpul sama halnya membuang seluruh lintasan.

Penggunaan terbesar *backtrack* adalah untuk membuat *AI (Artificial Intelligence)* pada *board games*. Dengan algoritma ini program dapat menghasilkan pohon sampai dengan kedalaman tertentu dari *current* status dan memilih solusi yang akan membuat langkah-langkah yang dapat dilakukan oleh *user* akan menghasilkan pohon solusi baru dengan jumlah pilihan langkah terbanyak. Cara ini dipakai sebagai *AI* yang digunakan untuk *dynamic problem solving*.

Beberapa kegunaan yang cukup terkenal dari algoritma *backtrack* dari suatu masalah "statik" adalah untuk memecahkan masalah mencari perlintasan sebuah mobil dan N-Queen problem.mencari perlintasan sebuah mobil adalah bagaimana mencari solusi yang bisa menunjukan jalan sebuah mobil pada jarak terdekat dengan beberapa rintangan yang terdapat pada lintasan tersebut. Meskipun mungkin terdapat lebih dari satu solusi untuk masalah ini, tetapi pencarian semua solusi biasanya tidak terlalu diperlukan, tetapi untuk beberapa kasus tertentu diperlukan pencarian semua solusi sehingga didapatkan solusi yang optimal

2.3. Pencarian Lintasan Pada Bidang Kartesian

Salah satu permasalahan yang dapat diangkat menjadi topik pembahasan AI adalah pencarian lintasan pada bidang kartesian. Permasalahan ini menerapkan metode algoritma backtracking untuk mencari solusi yang mungkin.

Permasalahan ini dapat didekripsikan sebagai berikut, diketahui sebuah mobil akan bergerak dari titik pusat (0,0) ke titik A(m,n) pada bidang kartesian. Mobil tersebut hanya boleh membelok pada titik-titik *grid* dan selalu melangkah sejajar dengan sumbu-x atau sumbu-y (horizontal atau vertikal). Mobil tersebut tidak boleh melintasi lintasan yang telah pernah dilaluinya dan tidak boleh melintasi titik yang telah pernah dilaluinya. Setelah itu, disediakan sederetan ketentuan yang membatasi pergerakan mobil tersebut. Pertanyaannya adalah bagaimana bentuk lintasan-lintasan (yang mungkin) yang dapat dilalui oleh mobil tersebut dengan menggunakan ketentuan-ketentuan yang telah ditetapkan di atas.

Implementasi algoritma bactracking adalah sebagai berikut, pertama-tama kita *setting* keadaan awal sebagai *root node* (*node* akar). Selanjutnya, lakukan pengembangan *node-node* anak yang mungkin menuju keadaan tujuan dengan mematuhi dan tidak melanggar aturan yang ada.

3. Analisa

Pencarian lintasan yang dapat dilalui oleh sebuah mobil pada bidang kartesian dapat dilakukan dengan bantuan algoritma backtracking. Deskripsi dari permasalahan ini adalah sebagai berikut, diketahui sebuah mobil akan bergerak dari suatu titik (x,y) ke titik A(m,n). Mobil hanya boleh membelok atau melintas pada titik-titik *grid* dan selalu melangkah sejajar dengan sumbu-x atau sumbu-y.mobil tidak boleh melintasi lintasan yang telah pernah dilaluinya atau melintasi titik yang telah pernah dilaluinya. Setelah itu, disediakan sederetan ketentuan yang membatasi pergerakan mobil tersebut. Ketentuan yang membatasi adalah batas pergerakan maksimum mobil yang diperbolehkan, posisi-posisi rintangan dimana mobil tidak boleh melintas, posisi-posisi dimana mobil harus melintas dan mobil hanya melintasi kuadran yang diperbolehkan untuk melintas.

Solusi dari permasalahan ini bisa lebih dari satu dan perangkat lunak dituntut untuk mampu menampilkan semua lintasan yang dapat dilalui mobil dari posisi awal ke posisi tujuan dengan mematuhi ketentuan-ketentuan yang berlaku. Karena solusi yang didapatkan bisa lebih dari satu, maka metode pencarian yang digunakan untuk mencari solusi permasalahan ini adalah metode pencarian melebar pertama *Depth first search* (DFS)..

Dimulai dari posisi awal sebagai *node* akar, selanjutnya metode DFS mencari solusi dengan mengembangkan *node* akar ke level-level berikutnya, semua pergerakan yang mungkin, tidak melanggar ketentuan dan menghasilkan kondisi baru dikembangkan semaksimal mungkin. Pencarian berakhir apabila tidak ada lagi *node* atau kondisi baru yang dapat dikembangkan. Semua *node* yang merupakan posisi tujuan merupakan solusi. Adapun pergerakan semut yang diperbolehkan adalah:

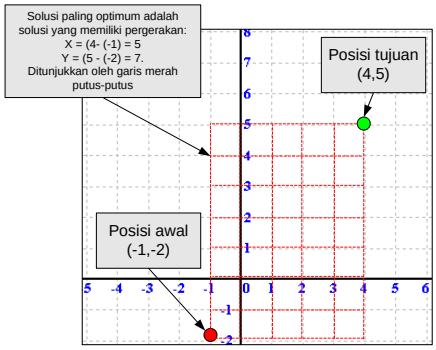
1. Bergerak ke kanan ($x + 1$).
2. Bergerak ke kiri ($x – 1$).
3. Bergerak ke atas ($y + 1$).
4. Bergerak ke bawah ($y – 1$).
5. Semua pergerakan mobil di atas legal apabila mematuhi ketentuan-ketentuan sebagai berikut:
   a. Tidak melintasi posisi titik rintangan.
   b. Tidak melintasi posisi titik yang sudah pernah dilalui sebelumnya.
   c. Tidak melebihi batas pergerakan maksimum mobil (x atau y).

Tidak melintasi kuadran yang tidak diperbolehkan untuk melintas. Kuadran yang dimaksud adalah kuadran I (dimana x bernilai positif dan y bernilai positif), kuadran II (dimana x bernilai negatif dan y bernilai positif), kuadran III (dimana x bernilai negatif dan y bernilai negatif) dan kuadran IV (dimana x bernilai positif dan y bernilai positif).

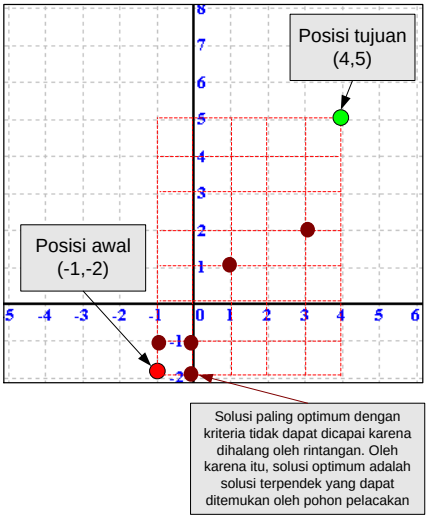
Dari solusi-solusi yang ditemukan, terdapat solusi yang optimum. Dalam implementasinya, solusi optimum merupakan langkah penyelesaian terpendek (*shortest path*) yang ditemukan dengan menggunakan algoritma bactracking. Dalam kasus pergerakan mobil, solusi paling optimum (pergerakan terpendek tanpa halangan) dapat ditentukan kriterianya. Simak contoh berikut, apabila posisi awal mobil adalah (x_1, y_1) dan posisi tujuan mobil adalah (x_2, y_2) maka solusi paling optimum (langkah terpendek) menuju posisi tujuan dapat dihitung dengan rumus: ($abs(x_2 - x_1)$, $abs(y_2 - y_1)$). Ini artinya, pergerakan sepanjang sumbu *x* (horizontal) adalah sebesar $abs(x_2 - x_1)$ langkah dan pergerakan sepanjang sumbu *y* (vertikal) adalah sebesar $abs(y_2 - y_1)$. Misalkan posisi awal mobil adalah (-1, -2) dan posisi tujuan mobil adalah (4, 5), maka solusi yang paling optimum adalah solusi yang memiliki kriteria sebagai berikut:

1. Pergerakan sepanjang sumbu *x* (horizontal) adalah *abs*(4 - (- 1)) = 5 langkah.
2. Pergerakan sepanjang sumbu *y* (vertikal) adalah *abs*(5 - (- 2)) = 7 langkah.

Kriteria solusi paling optimum dapat dilihat pada gambar 3.2 berikut.



Gambar 1 Solusi Optimum



Gambar 2 Solusi Optimum dengan rintangan

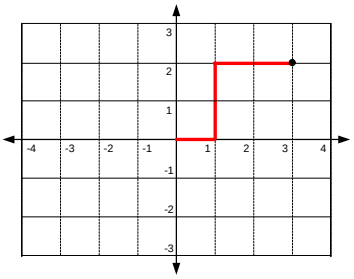
Namun apabila, terdapat halangan pada pergerakan mobil (yang menyebabkan semua solusi paling optimum tidak dapat dilewati, perhatikan gambar 3), maka kriteria untuk solusi optimum tersebut tidak berlaku. Oleh karena itu, solusi optimum adalah solusi terpendek yang ditemukan oleh algoritma backtracking dan tidak ada kriteria yang dapat merumuskan bahwa suatu solusi merupakan solusi yang paling optimum.

Di dalam perangkat lunak, untuk mencari atau mendapatkan solusi yang paling optimum, maka pergerakan maksimum mobil dibatasi dengan rumus di atas dan tidak terdapat halangan yang membatasi semua pergerakan mobil yang ada.

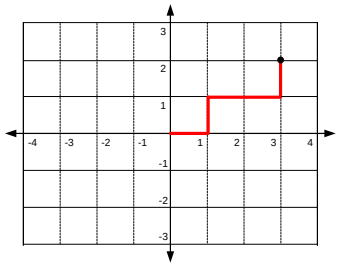
Agar lebih jelas, perhatikan contoh dari permasalahan berikut:

1 Posisi awal mobil: (2,1).
2 Posisi tujuan mobil: (5,5).
3 Pergerakan maksimum mobil, $x = 6$ dan $y = 6$.
4 Posisi rintangan, yaitu:
   a. Rintangan-1: (2,3).
   b. Rintangan-2: (4,3).
   c. Rintangan-3: (3,5).
5 Semua kuadran boleh dilalui oleh mobil.
6 Solusi paling optimum adalah solusi yang memiliki kriteria berikut:
   a. Pergerakan sepanjang sumbu x = *abs*(5 – 2) = 3.
   b. Pergerakan sepanjang sumbu y = *abs*(5 – 1) = 4.

Maka sesuai dengan metode pencarian yang digunakan dan ketentuan yang berlaku, terdapat 306 buah solusi. Contoh dua solusi paling optimum dapat dilihat pada gambar 4 dan gambar 5 berikut:

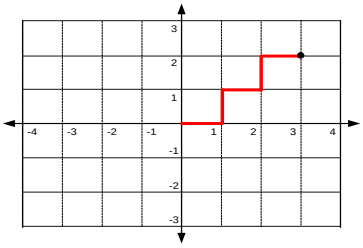


Gambar 3 Solusi Optimum -1



Gambar 4 Solusi Optimum -2

Solusi pada gambar 3 dan gambar 4 merupakan solusi paling optimum karena merupakan solusi dengan pergerakan terpendek (*shortest path*) dan memenuhi kriteria rumus solusi paling optimum, yaitu mempunyai pergerakan sumbu *x* sebanyak 3 langkah dan sumbu *y* sebanyak 4 langkah.
Dua solusi tidak optimum dapat dilihat pada gambar 5



Gambar 5 Solusi tidak Optimum -1

Solusi pada gambar 4 bukan merupakan solusi optimum karena bukan merupakan langkah terpendek (*shortest path*). Solusi pada gambar 5 memiliki 9 pergerakan

4. Hasil dan Pembahasan

4.1 Implementasi

Fungsi dari algoritma ini adalah untuk mengecek *input* posisi awal dan posisi tujuan supaya tidak bertentangan dengan kuadran yang diperbolehkan untuk dilewati. Algoritma ini dijalankan sebelum perangkat lunak mulai mencari solusi pergerakan. Apabila terdapat kesalahan, maka perangkat lunak akan menampilkan pesan kesalahan dan posisi dianggap tidak *valid*. Berikut adalah algoritma validasi posisi:

1. Jika posisi awal kosong, maka munculkan pesan kesalahan ‘Posisi awal belum diatur !”.
2. Jika posisi tujuan kosong, maka munculkan pesan kesalahan ‘Posisi tujuan belum diatur !”.
3. Jika X_Awal > 0 dan Y_Awal > 0 dan kuadran I tidak boleh dilalui mobil, maka munculkan pesan kesalahan ‘Posisi awal semut berada pada kuadran I. Checklist kuadran I !’.
4. Jika X_Tujuan > 0 dan Y_Tujuan > 0 dan kuadran I tidak boleh dilalui mobil, maka munculkan pesan kesalahan ‘Posisi tujuan mobil berada pada kuadran I. Checklist kuadran I !’.
5. Jika X_Awal < 0 dan Y_Awal > 0 dan kuadran II tidak boleh dilalui mobil, maka munculkan pesan kesalahan ‘Posisi awal mobil berada pada kuadran II. Checklist kuadran II !’.
6. Jika X_Tujuan < 0 dan Y_Tujuan > 0 dan kuadran II tidak boleh dilalui mobil, maka munculkan pesan kesalahan ‘Posisi tujuan mobil berada pada kuadran II. Checklist kuadran II !’.
7. Jika X_Awal < 0 dan Y_Awal < 0 dan kuadran III tidak boleh dilalui mobil, maka munculkan pesan kesalahan ‘Posisi awal mobil berada pada kuadran III. Checklist kuadran III !’.
8. Jika X_Tujuan < 0 dan Y_Tujuan < 0 dan kuadran III tidak boleh dilalui mobil, maka munculkan pesan kesalahan ‘Posisi tujuan mobil berada pada kuadran III. Checklist kuadran III !’.
9. Jika X_Awal > 0 dan Y_Awal < 0 dan kuadran IV tidak boleh dilalui mobil, maka munculkan pesan kesalahan ‘Posisi awal mobil berada pada kuadran IV. Checklist kuadran IV !’.
10. Jika X_Tujuan > 0 dan Y_Tujuan < 0 dan kuadran IV tidak boleh dilalui mobil, maka munculkan pesan kesalahan ‘Posisi tujuan mobil berada pada kuadran IV. Checklist kuadran IV !’.

4.2 Solusi Pencarian Pergerakan Mobil

Pencarian solusi pergerakan mobil menggunakan algoritma backtracking yang berbasis kepada metode pencarian *Depth first search* (DFS)., Karena algoritma *backtrack* akan mencoba menelusuri semua kemungkinan solusi yang mungkin, maka hal pertama yang harus dilakukan adalah membuat algoritma dasar yang akan menjelajahi semua kemungkinan solusi. Kemudian algoritma ini diperbaiki sehingga cara pencarian solusinya dibuat lebih sistematis. Lebih tepatnya jika algoritma itu dibuat sehingga akan menelusuri kemungkinan solusi pada suatu pohon solusi abstrak. Algoritma ini memperbaiki teknik *brute-force* dengan cara menghentikan penelusuran cabang jika pada suatu saat sudah dipastikan tidak akan mengarah ke solusi. Dengan demikian jalan-jalan yang harus ditempuh ada dibawah *node* pada pohon tersebut tidak akan ditelusuri lagi sehingga kompleksitas program akan berkurang. Kembali pada pohon solusi yang sebelumnya.

Jika pada pengecekan status di *node* (2) sudah dapat dipastikan bahwa jalan ini tidak akan menghasilkan solusi yang layak maka penelusuran jalan langsung dibatalkan dan dalam kasus ini langsung menelusuri *node* (3). Pembatalan penelusuran pada *node* (2) secara otomatis akan menghilangkan pengecekan jalan (1,2,5) dan (1,2,6) yang merupakan faktor untuk mengurangi kompleksitas waktu yang diperlukan. Semakin cepat terdeteksi bahwa jalan yang ditempuh tidak akan mengarah ke suatu solusi yang layak maka program akan bekerja dengan lebih efisien. Untuk kembali pada *state* sebelumnya (dalam kondisi *backtrack*) maka agar hasil perhitungan sampai dengan *state* tersebut harus disimpan dalam memori. Penyimpanan ini paling mudah dilakukan pada bahasa pemograman yang telah bisa menangani fungsi-fungsi atau prosedur-prosedur secara rekursif, sehingga manajemen memori akan dilakukan sepenuhnya oleh *compiler*. Pada bahasa pemograman yang lain, algoritma *backtrack* masih dapat diimplementasikan meskipun manajemen memori harus dilakukan oleh *programmer*. Cara menajemen memori yang baik adalah menggunakan *pointer* atau *dynamic array* karena kedalaman pohon solusi yang harus ditelusuri biasanya bervariasi dan tidak dapat ditentukan.

Algoritma *backtrack* dapat diimplementasikan dengan mudah pada bahasa-bahasa pemograman yang telah men*support* pemograman rekursif. Bahasa pemograman yang nyaman digunakan adalah Pascal atau Java. Bahasa Pascal dipilih karena bisa diprogram secara rekursif dan mendukung penggunaan *pointer*. Sedangkan bahasa Java meskipun lebih rumit tetapi dapat bekerja secara rekursif dan sangat mudah dalam membuat *dynamic array*.Skema yang umum digunakan pada pemograman dengan fungsi rekursif adalah telusuri solusi yang ada kemudian cek *state* program apakah sedang menuju ke suatu solusi. Jika ya maka panggil kembali fungsi itu secara rekursif. Kemudian cek apakah solusi sudah ditemukan (jika hanya perlu mencari sebuah solusi) atau semua kemungkinan solusi sudah diperiksa (jika ingin mengecek semua kemungkinan solusi). Jika ya maka langsung keluar dari prosedur atau fungsi tersebut. Kemudian pada akhir fungsi kembalikan semua perubahan yang dilakukan pada awal fungsi. Pada bahasa pemograman yang telah mendukung pemanggilan secara rekursif semua *state* setiap fungsi akan diatur oleh *compiler*. Dengan skema ini maka jika program tidak memiliki solusi maka *state* akhir program akan sama dengan *state* awal program.

Karena solusi dari permasalahan bisa lebih dari satu. Dimulai dari posisi awal sebagai *node* akar, selanjutnya metode DFS mencari solusi dengan mengembangkan *node* akar ke level-level berikutnya, semua pergerakan yang mungkin, tidak melanggar ketentuan dan menghasilkan kondisi baru dikembangkan semaksimal mungkin. Pencarian berakhir apabila tidak ada lagi *node* atau kondisi baru yang dapat dikembangkan. Semua *node* yang merupakan posisi tujuan merupakan solusi. Adapun pergerakan mobil yang diperbolehkan adalah:

1. Bergerak ke kanan ($x + 1$).
2. Bergerak ke kiri ($x – 1$).
3. Bergerak ke atas ($y + 1$).
4. Bergerak ke bawah ($y – 1$).
5. Semua pergerakan mobil di atas legal apabila mematuhi ketentuan-ketentuan sebagai berikut:

a. Tidak melintasi posisi titik rintangan.
b. Melintasi posisi-posisi yang telah ditentukan.
c. Tidak melintasi posisi titik yang sudah pernah dilalui sebelumnya.
d. Tidak melebihi batas pergerakan maksimum mobil (*x* atau *y*).
e. Tidak melintasi kuadran yang tidak diperbolehkan untuk melintas. Kuadran yang dimaksud adalah kuadran I (dimana *x* bernilai positif dan *y* bernilai positif), kuadran II (dimana *x* bernilai negatif dan *y* bernilai positif), kuadran III (dimana *x* bernilai negatif dan *y* bernilai negatif) dan kuadran IV (dimana *x* bernilai positif dan *y* bernilai positif).

Berikut adalah algoritma pencarian solusi pergerakan mobil:

1. Set posisi awal mobil.
   a. Mobil(0).X = PosisiAwal.X.
   b. Mobil(0).Y = PosisiAwal.Y.
   c. Mobil(0).StepX = 0.
   d. Mobil(0).StepY = 0.
   e. Mobil(0).Gerakan = “”.
   f. Mobil(0).Sebelum = -1.
   g. Mobil(0).Tujuan = *False*.
2. nIndex = -1.
3. Lakukan perulangan algoritma berikut sampai (nIndex = ukuran *array* variabel ‘Mobil’) atau (proses dibatalkan oleh *user*).
   a. nIndex = nIndex + 1.
   b. Jika Mobil(nIndex).Tujuan = *False* (bukan posisi tujuan), maka lanjutkan ke algoritma berikutnya. Jika tidak, kembali ke poin a.
   c. Untuk nTemp = 1 sampai 4, lakukan looping dari poin d sampai poin g.
   d. Jika nTemp = 1 (gerakan semut vertikal ke atas), maka:
      1) PosTemp.X = Mobil(nIndex).X.
      2) PosTemp.Y = Mobil(nIndex).Y + 1.
      3) PosTemp.StepX = Mobil(nIndex).StepX.
      4) PosTemp.StepY = Mobil(nIndex).StepY + 1.
      5) Jika CekPosisi(PosTemp) = True, maka lanjutkan ke algoritma berikutnya. Jika tidak, maka keluar dan lanjutkan algoritma ke poin e.
      6) bTemp1 = *False*.
      7) nTemp2 = nIndex.
      8) Cek apakah posisi sudah pernah dilalui sebelumnya. Selama nTemp2 > -1 dan bTemp1 = *False*, maka:
         a) bTemp1 = (Mobil(nTemp2).X = PosTemp.X And Mobil(nTemp2).Y = PosTemp.Y).
         b) nTemp2 = Mobil(nTemp2).Sebelum.
      9) Jika bTemp1 = *False* (posisi belum pernah dilewati), maka lanjutkan ke algorimta berikutnya. Jika tidak, maka maka keluar dan lanjutkan algoritma ke poin e.
      10) Bentuk sebuah posisi baru dengan menambah jumlah *array* variabel ‘Mobil’ dengan 1 dan set propertinya sebagai berikut:
         a) .X = PosTemp.X.
         b) .Y = PosTemp.Y.
         c) .StepX = PosTemp.StepX.
         d) .StepY = PosTemp.StepY.
         e) .Gerakan = “Y,+”.
         f) .Sebelum = nIndex

g) Jika (PosTemp.X = PosisiTujuan.X And PosTemp.Y = PosisiTujuan.Y) dan CekLewat = True, maka .Tujuan = True dan nPeny = nPeny + 1. Jika tidak, maka .Tujuan = False.

e. Jika nTemp = 2 (gerakan mobil vertikal ke bawah), maka :
   1) PosTemp.X = Mobil(nIndex).X.
   2) PosTemp.Y = Mobil(nIndex).Y - 1.
   3) PosTemp.StepX=Mobil(nIndex).StepX.
   4) PosTemp.StepY=Mobil(nIndex).StepY + 1.
   5) Jika CekPosisi(PosTemp) = True, maka lanjutkan ke algoritma berikutnya. Jika tidak, maka keluar dan lanjutkan algoritma ke poin f.
   6) bTemp1 = *False*.
   7) nTemp2 = nIndex.
   8) Cek apakah posisi sudah pernah dilalui sebelumnya. Selama nTemp2 > -1 dan bTemp1 = *False*, maka:
      a) bTemp1 = (Mobil(nTemp2).X= PosTemp.X And Mobil(nTemp2).Y = PosTemp.Y).
      b) nTemp2=Mobil(nTemp2).Sebelum.
   9) Jika bTemp1 = *False* (posisi belum pernah dilewati), maka lanjutkan ke algorimta berikutnya. Jika tidak, maka maka keluar dan lanjutkan algoritma ke poin f.
   10) Bentuk sebuah posisi baru dengan menambah jumlah *array* variabel 'Mobil' dengan 1 dan set propertinya sebagai berikut:
      a) .X = PosTemp.X.
      b) .Y = PosTemp.Y.
      c) .StepX = PosTemp.StepX.
      d) .StepY = PosTemp.StepY.
      e) .Gerakan = "Y,-".
      f) .Sebelum = nIndex
      g) Jika (PosTemp.X= PosisiTujuan.X And PosTemp.Y = PosisiTujuan.Y) dan CekLewat = True, maka .Tujuan = True dan nPeny = nPeny + 1. Jika tidak, maka .Tujuan = False.

f. Jika nTemp = 3 (gerakan semut horizontal ke kanan), maka:
   1) PosTemp.X = Mobil(nIndex).X + 1.
   2) PosTemp.Y = Mobil(nIndex).Y.
   3) PosTemp.StepX = Mobil(nIndex).StepX + 1.
   4) PosTemp.StepY = Mobil(nIndex).StepY.
   5) Jika CekPosisi(PosTemp) = True, maka lanjutkan ke algoritma berikutnya. Jika tidak, maka keluar dan lanjutkan algoritma ke poin g.
   6) bTemp1 = *False*.
   7) nTemp2 = nIndex.
   8) Cek apakah posisi sudah pernah dilalui sebelumnya. Selama nTemp2 > -1 dan bTemp1 = *False*, maka:
      a) bTemp1 = (Mobil(nTemp2).X= PosTemp.X And Mobil(nTemp2).Y = PosTemp.Y).
      b) nTemp2= Mobil(nTemp2).Sebelum.
   9) Jika bTemp1 = *False* (posisi belum pernah dilewati), maka lanjutkan ke algorimta berikutnya. Jika tidak, maka maka keluar dan lanjutkan algoritma ke poin g.
   10) Bentuk sebuah posisi baru dengan menambah jumlah *array* variabel 'Mobil' dengan 1 dan set propertinya sebagai berikut:
      a) .X = PosTemp.X.
      b) .Y = PosTemp.Y.
      c) .StepX = PosTemp.StepX.
      d) .StepY = PosTemp.StepY.

e) .Gerakan = "X,+".
f) .Sebelum = nIndex
g) Jika (PosTemp.X= PosisiTujuan.X And PosTemp.Y = PosisiTujuan.Y) dan CekLewat = True, maka .Tujuan = True dan nPeny = nPeny + 1. Jika tidak, maka .Tujuan = False.

g. Jika nTemp = 4 (gerakan mobil horizontal ke kiri), maka:
1) PosTemp.X = Mobil(nIndex).X - 1.
2) PosTemp.Y = Mobil(nIndex).Y.
3) PosTemp.StepX = Mobil(nIndex).StepX + 1.
4) PosTemp.StepY = Mobil(nIndex).StepY.
5) Jika CekPosisi(PosTemp) = True, maka lanjutkan ke algoritma berikutnya. Jika tidak, maka keluar dan lanjutkan algoritma ke poin 4.
6) bTemp1 = *False*.
7) nTemp2 = nIndex.
8) Cek apakah posisi sudah pernah dilalui sebelumnya. Selama nTemp2 > -1 dan bTemp1 = *False*, maka:
a) bTemp1 = (Mobil(nTemp2).X = PosTemp.X And Mobil(nTemp2).Y = PosTemp.Y).
b) nTemp2 = Mobil(nTemp2).Sebelum.
9) Jika bTemp1 = *False* (posisi belum pernah dilewati), maka lanjutkan ke algorimta berikutnya. Jika tidak, maka maka keluar dan lanjutkan algoritma ke poin 4.
10) Bentuk sebuah posisi baru dengan menambah jumlah *array* variabel 'Mobil' dengan 1 dan set propertinya sebagai berikut:
a) .X = PosTemp.X.
b) .Y = PosTemp.Y.
c) .StepX = PosTemp.StepX.
d) .StepY = PosTemp.StepY.
e) .Gerakan = "X,-".
f) .Sebelum = nIndex
g) Jika (PosTemp.X = PosisiTujuan.X And PosTemp.Y = PosisiTujuan.Y) dan CekLewat = True, maka .Tujuan = True dan nPeny = nPeny + 1. Jika tidak, maka .Tujuan = False.

4. Ambil semua posisi yang merupakan posisi tujuan (.Tujuan = True).
a. Untuk nTemp1 = 0 sampai jumlah *array* pada variabel 'Mobil', lakukan perulangan pada algoritma poin b.
b. Jika Mobil(nTemp1).Tujuan = True, maka tambahkan jumlah *array* variabel 'X' dengan 1 dan lakukan algoritma berikut.
1) nIndex = -1.
2) nTemp2 = nTemp1.
3) Selama nTemp2 >= 0, maka set nIndex = nIndex + 1, tambah variabel array 'Step' pada properti variabel 'Mobil' dengan 1, set .Step(nIndex) = Mobil(nTemp2) dan nTemp2 = Mobil(nTemp2).Sebelum.

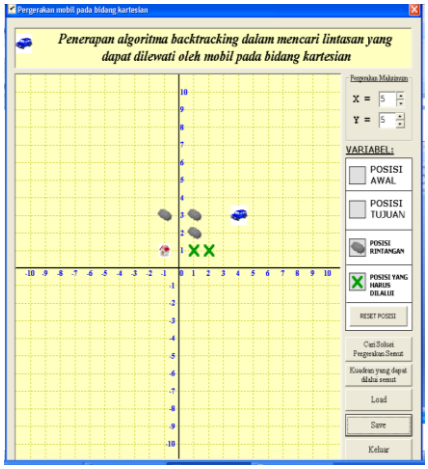
5. Ambil semua langkah-langkah penyelesaian:
a. Untuk nTemp1 = 0 sampai jumlah *array* pada variabel 'X', lakukan perulangan pada algoritma poin b.
b. Jika Mobil(nTemp1).Tujuan = True, maka sesuaikan jumlah *array* variabel SolusiPeny(nTemp1).Step dengan jumlah array Step pada variabel 'X'.
c. Untuk nTemp2 = 0 sampai SolusiPeny(nTemp1), set SolusiPeny(nTemp1). Step(nTemp2) = X(nTemp1) .Step(UBound(SolusiPeny(nTemp1).Step) - nTemp2).

4.3. Pengujian Sistem

Sebagai contoh pengujian, *input* data perangkat lunak adalah sebagai berikut:

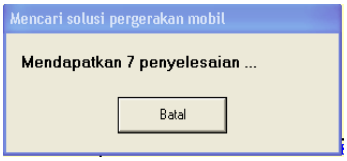
1. Posisi awal mobil = (1, 3).
2. Posisi tujuan mobil = (-3, 1).
3. Pergerakan maksimum mobil = (5, 5).
4. Posisi rintangan adalah sebagai berikut:
   a. Rintangan1: (-1, 4).
   b. Rintangan2: (-1, 2).
   c. Rintangan3: (-3, 3).
   d. Rintangan4: (2, 2).
   e. Rintangan5: (1, 1).
   f. Rintangan6: (0, 5).
   g. Rintangan7: (-2, 0).
5. Posisi yang harus dilalui mobil adalah sebagai berikut:
   a. Posisi1: (-1, 1).
   b. Posisi2: (-2, 1).
6. Hanya kuadran I dan kuadran II yang boleh dilalui mobil.

Tampilan *form input* dapat dilihat pada gambar 6.



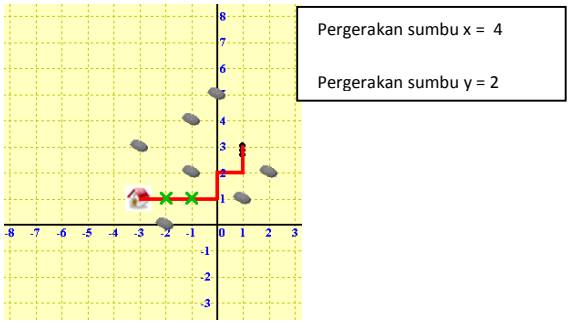
Gambar 6. *Form Input*

Proses pencarian solusi pada *form* pencarian dapat dilihat pada gambar 7.

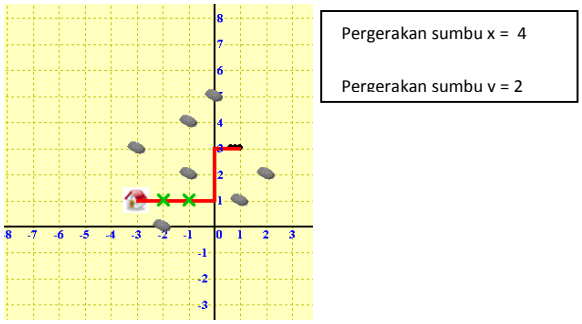


Gambar 7 *Form* pencarian

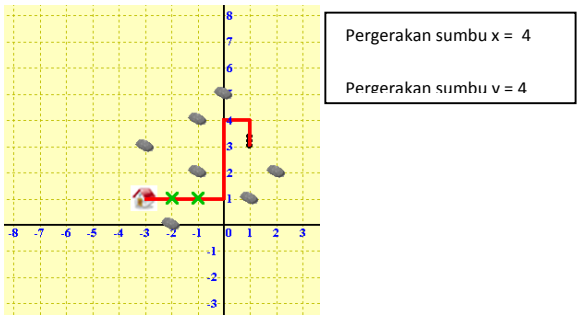
Hasil pencarian solusi adalah sebagai beriku



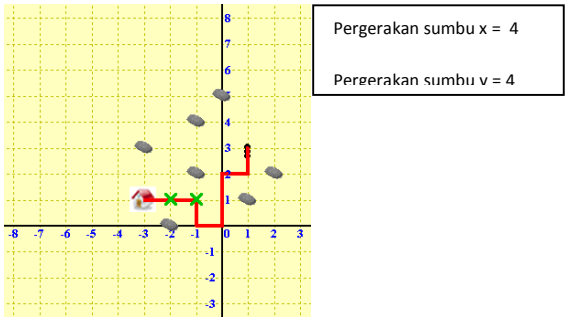
Gambar 8 *Form* Hasil -1



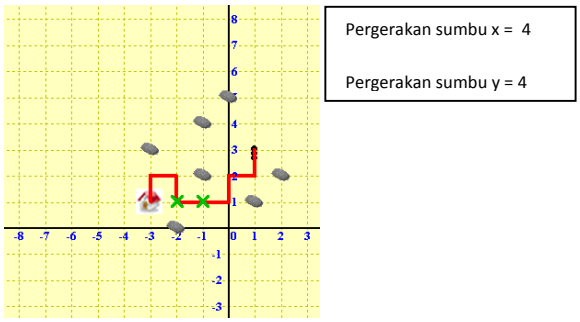
Gambar 9 *Form* Hasil -2



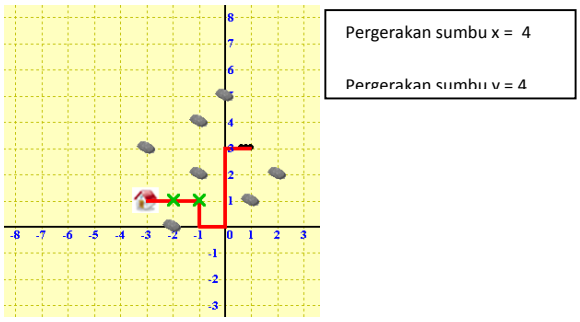
Gambar 10 *Form* Hasil -3



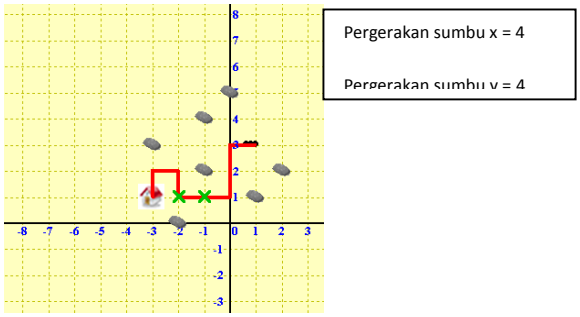
Gambar 11 *Form* Hasil -4



Gambar 12 *Form* Hasil -5



Gambar 13 *Form* Hasil -6



Gambar 14 *Form* Hasil -7

5. Simpulan

Penerapan algoritma backtarcking dalam mencari lintasan yang dapat dilalui oleh sebuah mobil pada bidang kartesian, disimpulkan sebagai berikut:

1. Algoritma bactracking bisa mencari solusi dalam perlintasan sebuah mobil, untuk mencari perlintasan yang terdekat dalam mencapai tujuan.
2. Metode *Depth first search* (DFS).dapat membantu dalam pencarian lintasan, walaupun masih ada kelemahan dan kelebihan dari metode ini.
3. Perangkat lunak merupakan implementasi nyata penerapan algoritma bactracking dalam mencari solusi.
4. Perangkat lunak dapat mensimulasikan semua solusi pergerakan dari posisi awal menuju posisi tujuan.

Daftar Pustaka

Arief Hermawan, "*Jaringan Syaraf Tiruan (Teori dan Aplikasi)*", Penerbit Andi, 2006.

Munir,Rinaldi,"*Algoritma dan Pemograman Dalam Bahasa Pascal*",Informatika,Bandung,2005.

Ramadhan.A, *MS. Visual Basic 6.0*, PT. Elex Media Komputindo, Jakarta, 2004.

Sandi.S, *Aritificial Intelegencia*, Andi Offset, Yogyakarta, 1993.

Supardi.Y, *Microsoft Visual Basic 6.0 Untuk Segala Tingkat*, PT. Elex Media Komputindo, Jakarta, 2006.

Sutedjo,Budi,"*Algoritma dan Teknik Pemograman*"PenerbitAndi,Jogjakarta,2004.

Suyanto,"*Artificial Intelligence*",Informatika,Bandung,2007.