BAB II
LANDASAN TEORI

2.1 Metaheuristik

Heuristik berasal dari kata Yunani *heuriskein* yang berarti seni untuk menemukan strategi dalam menyelesaikan persoalan. Sedangkan *meta* berarti metodologi tingkat tinggi atau lanjut. Didalam ilmu komputer, metode heuristik merupakan suatu teknik yang didesain untuk memecahkan masalah dengan sedikit mengabaikan apakah solusinya bisa dibuktikan benar, tetapi biasanya menghasilkan solusi yang bagus, dalam arti optimal atau mendekati optimal [5]. Heuristik dimaksudkan untuk mendapatkan hasil yang secara yang secara komputasi lebih cepat dengan konsekuensi mengurangi kepresisian atau akurasi. Jadi kecepatan perhitungan biasanya lebih baik (dibandingkan optimasi eksak) dengan sedikit mengorbankan akurasi. Walaupun pada kenyataanya solusinya bisa juga mempunyai akurasi yang tinggi. Pendekatan heuristik biasanya sangat spesifik untuk problem tertentu.

Metaheuristik dapat didefinisikan sebagai pendekatan komputasi untuk mencari solusi optimal atau mendekati optimal dari suatu problem optimasi dengan cara mencoba secara iteratif untuk memperbaiki kandidat solusi dengan memperhatikan batasan kualitas solusi yang diinginkan [5]. Metode metaheuristik banyak dipakai dalam optimisasi stokastik (optimisasi stokastim merupakan optimisasi yang memiliki derajat ketidakpastian atau *random*).

Metaheuristik memiliki beberapa karakteristik dasar yaitu:

1. Metaheuristik adalah strategi yang memandu proses pencarian.
2. Tujuan dari metaheuristik adalah untuk menjelajahi ruang pencarian secara efficient untuk menemukan solusi optimal.
3. Teknik metaheuristik berkisar dari prosedur pencarian lokal yang sederhana sampai proses pembelajaran yang komplek.
4. Meteheuristik adalah metode pendekatan dan biasanya non-deterministik.
5. Metaheuristik dapat terdiri dari penggabungan beberapa mekanisme supaya proses pencarian tidak terjebak dalam daerah terbatas di ruang pencarian.

6. Konsep dasar dari metaheuristik memungkinkan pendeskripsian secara abstrak.
7. Metaheuristik bersifat general/umum sehingga dapat diterapkan dalam berbagai macam persoalan.
8. Metaheuristik dapat menggunakan domain pengetahuan khusus dalam bentuk heuristik yang dikendalikan dengan strategi tingkat lanjut.
9. Metaheuristik dapat menggunakan pengalaman yang didapat selama proses pencarian untuk menuntun proses pencarian.

Dalam menentukan apakah metaheuristik adalah metode yang sesuai untuk menyelesaikan suatu permasalahan, ada beberapa hal yang perlu diperhatikan misalnya kompleksitas permasalahan, ukuran input, struktur input dan waktu yang diperlukan untuk menyelesaikan masalah tersebut. Secara umum metehauristik dipakai untuk masalah-masalah yang komplek dan tidak bisa diselesaikan dengan mudah secara analitikal/eksak. Tidaklah terlalu bermanfaat menggunakan metaheuristik untuk persoalan yang dengan mudah dan cepat dapat diselesaikan secara eksak (penyelesaian eksak merupakan penyelesaian terbaik, tetapi seringkali metode ini tidak dapat diterapkan pada permasalahan optimisasi, sehingga dipakailah metode pendekatan). Metaheuristik mencari solusi dengan memadukan interaksi antara prosedur pencarian lokal dan strategi yang lebih tinggi untuk menciptakan proses yang mampu keluar dari titik-titik local optima dan melakukan pencarian diruang solusi untuk menemukan solusi global. Tentu saja diperlukan berbagai modifikasi agar suatu metoda metaheuristik sesuai dapat menyelesaikan masalah khusus yang dihadapi. Banyak pendekatan yang masuk kategori metaheuristik diantaranya adalah algoritma *harmony search* [5].

2.2 Algoritma Harmony Searh

2.2.1 Pengertian Algoritma Harmony Searh

Harmony Search (HS) termasuk pendekatan metaheuristik yang mendasarkan algoritmanya pada musik [3]. Mungkin kita tidak pernah berfikir sebelumnya ada kaitannya antara musik dengan optimasi. Misalnya, setiap alat musik berkaitan dengan variabel keputusan, nada musik berkaitan dengan nilai variabel, harmoni berhubungan dengan vektor solusi. Seperti seorang musisi yang

memainkan musik tertentu, berimprovisasi memainkan nada secara *random* atau berdasarkan pengalaman untuk menemukan harmoni yang indah, variable dalam algoritma *harmony search* memiliki nilai *random* atau nilai yang didapat dari iterasi (memori) dalam usaha mendapatkan solusi optimal.

Algoritma ini terinspirasi dari observasi bahwa tujuan dari orang bermusik adalah mencari status harmoni yang sempurna. Usaha menemukan harmoni dalam musik adalah analogi dengan proses penemuan solusi optimal dalam optimasi. Dengan kata lain, proses improvisasi yang dilakukan seorang pemain musik *jazz* bisa dianalogikan dalam proses pencarian solusi dalam optimasi. Selain itu, harmoni yang sempurna ditentukan oleh standar keindahan audio. Musisi selalu ingin menghasilkan karya musik yang mempunyai harmony yang sempurna. Sementara itu solusi optimal untuk masalah optimasi harus dicari berdasarkan fungsi tujuan dan konstrain yang ada. Kemiripan ini digunakan sebagai dasar penyusunan algoritma *harmony search* (HS). Algoritma *harmony search* pertama kali diperkenalkan oleh Zong Woo Geem [3].

Permainan musik akan mencari harmoni yang paling indah yang ditentukan oleh estimasi keindahan (*aesthetic estimation*), sebagaimana proses optimasi ingin menemukan solusi optimum yang ditentukan oleh adanya fungsi tujuan. Estimasi keindahan dilakukan dengan pengaturan (*pitches*) nada yang dilakukan bersama dari gabungan instrumen. Sebagaimana fungsi tujuan diestimasi pada beberapa nilai variabel, suara yang indah akan didapatkan setelah pemain musik sering berlatih sebagaimana fungsi tujuan dapat diperbaiki dengan menambah iterasi.

Dalam kasus optimasi nyata, setiap pemain musik dapat dianalogikan dengan variabel keputusan, sedangkan suara yang diinginkan dapat dianalogikan dengan nilai setiap variabel. Misalkan setiap variabel mewakili pipa yang dicari diameternya dan pilihan nada. Nada

{Do, Re, Mi, Fa, Sol, La, Si}

Mewakili nilai diameter

{1, 2, 3, 4, 5, 6, 7}

Jika variabel pertama bernilai 5 dari

{1, 4, 3, 5, 2}

Dan yang kedua bernilai 7 dari

{7, 1, 7, 2, 5}

Dan yang ketiga 1 dari

{6, 5, 4, 3, 1}

Akan didapat vektor solusi baru yang terbentuk (5, 7, 1). Jika vektor solusi baru ini lebih baik dari vektor terburuk yang sudah ada, biasanya disimpan dalam *harmony memory* (HM), maka solusi baru ini akan masuk kedalam HM dan vektor terburuk dikeluarkan dari HM. Prosedur ini diulang sampai kriteria penghentian dicapai.

Misalkan ada 3 orang pemain musik masing-masing memainkan saxophone, gitar, dan bass. Mereka ingin mencari harmoni nada yang bagus. Nada yang ada pada memori pemain saxophone adalah

{Do, Fa, Mi, Sol, Re}

Bassis

{Si, Do, Si, Re, Sol}

Dan gitaris,

{La, Sol, Fa, Mi, Do}

Jika saxophonis secara *random* memainkan Sol dari memorinya Do, Fa, Mi, Sol, Re, basis memainkan Si dari {Si, Do, Si, Re, Sol}, dan gitaris memainkan Do dari {La, Sol, Fa, Mi, Do}, maka didapatkan harmoni baru (Sol, Si, Do). Secara musik nada ini mewakili chord C7. Jika harmoni baru ini lebih baik dari harmoni yang ada pada *harmony memory*, maka harmoni ini akan menggantikannya. Langkah ini akan diulang sampai harmoni yang lebih bagus didapat.

Walaupun teknik ini terbilang baru dalam metaheuristik, namun dalam beberapa penerapan pada berbagai problem menunjukan HS termasuk efektif dan mempunyai banyak kelebihan. Beberapa contoh penerapannya adalah pada optimasi fungsi, jaringan, distribusi air, pemodelan air tanah, *energy-saving dispatch*, *truss design*, *vehicle routing*, dan banyak lagi. Kemungkinan

penggabungan HS denga algoritma lain seperti *Particle Swarm Optimization* juga dilakukan untuk memperbaiki performansi [5].

2.2.2 Konsep Harmony Search

Analogi musik dengan proses optimasi menurut HS adalah sebagai berikut :

1. Instrumen Musik ↔ variabel keputusan
2. *Pitch range* ↔ Range nilai variabel
3. *Harmony* ↔ Vektor Solusi
4. *Aesthetic* ↔ Fungsi Tujuan
5. *Practice* ↔ Iterasi
6. *Experience* ↔ Matrik Memori

Dengan adanya analogi seperti ini, disusunlah algoritma HS ini. Ide dasar algoritma HS adalah meniru proses perbaikan harmoni musik yang dilakukan oleh kelompok pemain musik. Ketika kelompok pemain musik melakukan perbaikan pada harmoni musik yang dimainkan, maka akan terdapat tiga kemungkinan pilihan, yaitu :

1. Memainkan harmoni musik yang terkenal berdasarkan ingatan mereka.
2. Memainkan harmoni musik yang serupa dengan harmoni musik yang terkenal namun ada sedikit penyesuaian, atau
3. Membuat harmoni musik yang baru.

Geem menganalogikan ketiga pilihan ini pada proses optimasi secara kuantitatif. Ketiga komponen tersebut diformulasikan menjadi *harmony memory*, penyesuaian nada (*pitch adjusting*), dan proses pembangkitan nada secara *random*. Penggunaan HMCR dan PAR akan berhubungan dengan pilihan mengambil harmoni baru dar memori lalu melakukan pengaturan atau tidak dan membuat harmoni baru. Pencarian harmoni dilakukan untuk setiap instrumen musik. Jika ada 2 instrumen atau ekuivalen dengan 2 variabel, maka setiap instrumen harus diatur sampai menemukan nada terbaik. Artinya kalau ada 2

variabel maka pengaturan nilainya akan dilakukan per variabel untuk setiap siklus. Jadi dalam setiap iterasi akan mendapat siklus sebanyak jumlah variabel.

Penggunaan harmoni memori sangat penting karena harmoni memori tersebut bisa menjamin bahwa harmoni yang bagus akan dipertimbangkan sebaga elemen-elemen dari vektor solusi yang baru. Agar *harmony memory* dapat digunakan secara efektif, algoritma HS mengadopsi sebuah parameter yang disebut *Harmony memory Considering Rate* (HMCR). Nilai HMCR akan menetukan apakah satu nada baru akan dibangkitkan atau mengambil dari *harmony memory*. Jika nilai rate ini terlalu rendah, maka hanya sedikit harmoni elit (yang sudah tersimpan dalam *harmony memory*) yang terpilih untuk dilakukan pengaturan dan dapat menyebabkan proses konvergensi terlalu lambat, jika *rate* ini terlalu besar, maka akan menyebabkan nada-nada pada *harmony memory* banyak terpakai dan tidak sempat mengeksplorasi nada lain yang baru dimana pada akhirnya sulit mencapai solusi yang bagus. Sehingga akan besar kemungkinan terjadi solusi yang local optimum. Oleh karena itu, biasanya digunakan HMCR = 0.7 – 0.95.

Komponen kedua adalah penyesuaian nada dimana proses ini ada beberapa parameter seperti *bandwidth*(bw) dan *Pitch Adjusting Rate* (PAR). Pernyesuaian nada musik berarti pengubahan frekuensi nada. Hal ini berarti membangkitkan nilai yang sedikit berbeda dari nada yang sudah ada. Pengaturan dengan menaikan nada yang sekarang atau menurunkan. Dalam optimasi berarti menaikan atau menurunkan nilai variabel dengan menggunakan *bandwidth* tadi. Jika *bandwidth* nilainya terlalu besar maka pengaturan akan lebih cepat tetapi akan ada kemungkinan melompati nilai optimal yang dicari. Jika nilai *bandwidth* kecil akan besar kemungkinan didapat nilai optimal yang dicari akan tetapi dengn waktu yang lebih lama. Umumnya diperlukan cukup banyak jumlah iterasi (dalam skala ribu) untuk menemukan solusi terbaik dalam HS ini. HS menyelesaikan suatu permasalahan optimasi (minimasi fungsi) dengan langkah-langkah umum sebagai berikut :

1. **Langkah 1. Inisialisasi parameter**

   Parameter dari algoritma Harmony search :

   HMS = Ukuran *Harmony memory*. Hal ini biasanya bervariasi dari 1 sampai 100. (nilai khas = 30)

   HMCR = Laju memilih nilai dari *Harmony memory*. Hal ini biasanya bervariasi 0,7-0,99. (nilai khas = 0,9)

   PAR = Laju memilih nilai tetangga. Hal ini biasanya bervariasi 0,1-0,3. (nilai khas = 0,3)

   f(x) = Fungsi Objektif

   xL = Batas Bawah

   xU = Batas Atas

2. **Langkah 2. Inisialisasi *Harmony memory* (HM)**

HM terdiri dari N solusi awal. Solusi ini terdiri dari satu variabel sampai p variabel. Solusi ini dibangkitkan secara *random*. Semua kandidat solusi ini dievaluasi untuk menemukan solusi terburuk.

$$X = \begin{bmatrix} x_1^1 & x_2^1 & x_3^1 & \cdots & x_{p-1}^1 & x_p^1 \\ x_1^2 & x_2^2 & x_3^2 & & x_{p-1}^2 & x_p^2 \\ \vdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ x_1^{HMS-1} & x_2^{HMS-1} & x_3^{HMS-1} & & x_{p-1}^{HMS-1} & x_p^{HMS-1} \\ x_1^{HMS} & x_2^{HMS} & x_3^{HMS} & \cdots & x_{p-1}^{HMS} & x_p^{HMS} \end{bmatrix}$$

Dimana masing-masing vektor solusi (tiap baris) akan dievaluasi nilai fungsinya

$$f(x) = \begin{pmatrix} f(x^1) \\ f(x^2) \\ \vdots \\ f(x^{HMS-1}) \\ f(x^{HMS}) \end{pmatrix} \tag{2.1}$$

3. **Langkah 3. Lankukan perbaikan / improvisasi terhadap solusi yang ada**

Untuk setiap variabel diambil secara *random* nilai yang ada pada HM. Dengan prosedur tertentu nilai ini akan diadjust sedemikian rupa jika memenuhi aturan tertentu (menggunakan pembangkitan bilangan *random* dan dibandingkan

dengan HMCR dan PAR) hingga akan didapatkan nilai baru. Atau kalau tidak memenuhi aturan, akan dibangkitkan solusi baru secara *random*.

Suatu harmoni baru atau vektor baru akan dibangkitkan berdasarkan aturan berikut : *Harmony memory Consideration Rate* (HMCR), *Pitch Adjusting Rate* (PAR), dan pembangkitan yang benar-benar *random*. Sebagai contoh nilai x_1^1 baru akan diambil dari $x_1^1 \sim x_1^{HMS}$. Variabel yang lain dicari dengan cara yang sama. Besarnya nilai HMCR akan menentukan nilai baru ini besar kemungkinannya akan diambil dari HM atau benar-benar dibangkitkan secara *random*.

$$x_i' = \begin{cases} x_i' \in \{x_i^1, x_i^2, \dots, x_i^{HMS}\} \; d.p & HMCR \\ x_i' \in X_i & d.p\,(1 - HMCR) \end{cases} \quad (2.2)$$

Dimana HMCR adalah probabilitas memilih satu nilai dari HM dan 1-HMCR adalah probabilitas memilih nilai secara *random* dalam range *xl-xu*. Setelah memilih suatu harmoni baru $x^1 = (x_1', x_2', \dots, x_N')$, keputusan melakukan *pitch adjustment* dilakukan untuk setiap komponen vektor solusi. Prosedur ini menggunakan parameter PAR untuk melakukan pengaturan:

$$x_i' = \begin{cases} Atur\ Pitch & d.p\ PAR \\ Tidak\ lakukan\ sesuatu & d.p\,(1 - PAR) \end{cases} \quad (2.3)$$

Dalam proses *pitch adjustment* ini, suatu nilai pindah ke nilai didekatnya dengan peluang (d.p) PAR atau tetap pada nilai aslinya dengan peluang (1-PAR).

4. **Langkah 4. Perbarui *Harmony memory***

Solusi Baru ini akan dibandingkan dengan solusi terburuk dalam N populasi awal. Jika lebih baik maka ia akan menggantikan vektor solusi terburuk tadi.

5. **Langkah 5. Cek kriteria Penghentian**

Jika kriteria penghentian belum dipenuhi, kembali ke langkah 3 untuk mengambil secara acak salah satu vektor solusi dari variabel pertama. Bisa digunakan kriteria penghentian berupa jumlah iterasi atau nilai mutlak selisih dua nilai fungsi tujuan yang berturutan.

2.2.3 Implementasi Algoritma Harmony Search

Berikut adalah contoh bagaimana HS diimplementasikan dalam minimasi fungsi dengan dua variabel.

$$min\ f(x) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2$$

1. Pertama tetapkan beberapa nilai parameter berikut :

   HMS = 10, HMCR = 0.9, PAR = 0.3

   xu = [6, 6]

   xl = [-6, -6]

   b = (*xu* - *xl*) / 1000 = [0.012]

2. Mulai iterasi 1. Bangkitkan HM secara *random*. HM berisi solusi-solusi sementara yang kita bangkitkan. Misalnya berukuran 10 dan hitung nilai *f* yang bersangkutan. Berikut adalah nilai *x* yang kita bangkitkan dan *rand*(1,2) (*xu* - *xl*), dimana *rand*(1,2) adalah vektor bilangan *random* 1*x*2 dengan nilai antara (0,1).

Tabel 2.1 ***Harmony memory***

x_1	x_2
4.5666	5.1052
2.9607	-3.6990
1.91347	0.4705
-4.0163	-5.9232
0.8370	0.4696
-5.5020	-3.5190
2.6383	3.4459
-2.4362	-3.7531
1.3137	4.7428
-4.9343	-0.3410

Masukan nilai x_1 dan x_2 ke dalam fungsi objektif

$$min\ f(x) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2$$

$$min\ f(x) = (4.5666_1^2 + 5.1052_2 - 11)^2 + (4.5666_1 + 5.1052_2^2 - 7)^2$$

$$min\ f(x) = (223.7727) + (558.3612)$$

$$min\ f(x) = 782.1557$$

Dengan perhitungan yang sama terhadap semua x_1 dan x_2 maka akan didapatkan hasil sebagai berikut :

Tabel 2.2 ***Harmony memory***

x_1	x_2	*f (x)*
4.5666	5.1052	782.1557
2.9607	-3.6990	128.1969
1.91347	0.4705	70.8475
-4.0163	-5.9232	579.8802
0.8370	0.4696	131.9409
-5.5020	-3.5190	248.1863
2.6383	3.4459	56.7877
-2.4362	-3.7531	99.3760
1.3137	4.7428	303.0511
-4.9343	-0.3410	308.8373

3. Bangkitkan bilangan *random*, misal *r* = 0.85. bangkitkan dengan HMCR, karena *r* ≤ HMCR, maka ambil satu nilai dari HM sebagai x_1. Misalkan secara *random* diambil baris 2 dari HM, maka $xnew_1$ = 2.9607.
4. Bangkitkan bilangan *random*, *r* = 0.25. bandingkan dengan PAR. Karena *r* ≤ PAR maka perlu dilakukan pengaturan nilai $xnew_1$ menjadi

$$xnew_1 = xnew_1 + (2 * r - 1) * b(1)$$

Disini bilangan *random* r perlu dikalikan 2 dan dikurangi dengan 1 agar ada kemungkinan bernilai – atau +, misal *r* = 0.6, maka

$$xnew_1 = 2.9067 + (2 * 0.6 - 1) * 0.012 = 2.9631$$

Karena nilai ini masih didalam batas xu_1 dan xl_1 bisa kita gunakan untuk langkah berikutnya

5. Bangkitkan nilai xnew untuk variabel kedua. Bangkitkan bilangan *random*, misal *r* = 0.92. karena *r* ≥ HMCR, maka perlu dibangkitkan nilai baru untuk x_2 (tidak diambil dari HM). Gunakan

$$xnew_2 = xl_2 + \text{r} * (xu_2 - xl_2)$$

Misalnya bilangan *random* r = 0.3130, maka

$$xnew_2 = -6 + 0.3130 * (6 - (-6)) = 2.2435$$

Karena hanya ada 2 variabel maka kita mendapatkan

$$xnew = [2.9607, -2.2435]$$

Cek nilai *f* dari *xnew* ini,

$$f([2.9631, -2.2435]) = 20.9160$$

Bandingkan nilai *f* ini dengan nilai *f* terburuk dalam HM. Nilai *f* terburuk dalam HM adalah 782.1557 (baris 1). Maka nilai *x* pada HM dibaris 1 akan digantikan dengan [2.9631, -2.2435].

6. masuk ke iterasi 2, i= 2, ulangi langkah 3 hingga jumlah iterasi maksimum dicapai.

Dalam kasus optimasi kombinatorial, misalkan Travelling Salesman Problem (TSP) maka ada beberapa langkah modifikasi terhadap algoritma HS. Dalam problem ini ingin ditemukan solusi berupa urutan kota yang dikunjungi. Misalkan kota awal p[emberangkatan adalah kota 1 , dan kota berikutnya yang dikunjungi adalah dipilih berdasarkan tiga aturan berikut.

1. Aturan 1 . pilih sembarang kota dari HM sebagai kota berikutnya dengan probabilitas HMCR x (1-PAR), dimana PAR = PAR1 + PAR2 + PAR3

2. Aturan 2. Pilih kota terdekat sebagai kota berikutnya yang dikunjungi dengan probabilitas HMCR x PAR1; atau pilih kota terdekat kedua dengan probabilitas HMCR x PAR2; atau pilih kota terdekat ketiga dengan probabilitas HMCR x PAR3
3. Aturan 3. Pilih kota berikutnya secara *random* dengan probabilitas (1-HMCR).

Penjelasan dari aturan diatas adlah sebagai berikut. Untuk aturan 1 berarti jika bilangan *random* r yang terpilih lebih kecil dari HMCR maka akan dipilih salah satu kota secara *random* dari HM. Aturan 2 menyatakan jika r yang dibangkitkan lebih kecil dari HMCR dan PAR1 maka dipilih kota terdekat; atau pilih kota lebih kecil dari HMCR dan PAR1 maka dipilih kota terdekat, atau pilih kota terdekat kedua jika r lebih kecil dari HMCR, atau pilih kota terdekat ketiga jika r lebih kecil dari HMCR. Aturan 3 menyatakan jika r yang dibangkitkan lebih besar dari HMCR maka pilih kota secara *random*.

Pada algoritma diatas memungkinkan adanya kota yang sama dalam suatu rute diulang lagi, hal ini disebabkan karena adanya kemungkinan terpilihnya satu kota lebih dari satu kali pada tahap memory consideration.

1.3 *Game Puzzle*

1.3.1 Pengertian *Game Puzzle*

Puzzle berasal dari bahasa Inggris yang berarti teka-teki atau bongkar pasang, media *puzzle* merupakan media sederhana yang dimainkan dengan bongkar pasang.

Permainan *puzzle* ditujukan untuk memecahkan suatu masalah tertentu. Hampir semua semua tantangan disini menyangkut masalah logika yang biasanya dibatasi oleh waktu.

2.3.2 Jenis-Jenis *Game Puzzle*

Jenis-Jenis *game puzzle* [6] :

1. *Puzzle* konstruksi

Puzzle rakitan (*construction puzzle*) merupakan kumpulan potongan-potongan yang terpisah, yang dapat digabungkan kembali menjadi beberapa model. Mainan rakitan yang paling umum adalah blok-blok kayu sederhana berwarna-warni. Mainan rakitan ini sesuai untuk orang yang suka bekerja dengan tangan, suka memecahkan *puzzle*, dan suka berimajinasi.

2. *Puzzle* batang (*stick*)

Puzzle batang merupakan permainan teka-teki matematika sederhana namun memerlukan pemikiran kritis dan penalaran yang baik untuk menyelesaikannya. *Puzzle* batang ada yang dimainkan dengan cara membuat bentuk sesuai yang kita inginkan ataupun menyusun gambar yang terdapat pada batang *puzzle*.

3. *Puzzle* lantai

Puzzle lantai terbuat dari bahan *sponge* (karet/busa) sehingga baik untuk alas bermain anak dibandingkan harus bermain di atas keramik. *Puzzle* lantai memiliki desain yang sangat menarik dan tersedia banyak pilihan warna yang cemerlang. Juga dapat merangsang kreativitas dan melatih kemampuan berpikir anak. *Puzzle* lantai sangat mudah dibersihkan dan tahan lama.

4. *Puzzle* angka

Mainan ini bermanfaat untuk mengenalkan angka. Selain itu *puzzle* angka dapat melatih kemampuan berpikir logis dengan menyusun angka sesuai urutannya. Selain itu, *puzzle* angka bermanfaat untuk melatih koordinasi mata dengan tangan, melatih motorik halus serta menstimulasi kerja otak.

5. *Puzzle* transportasi

Puzzle transportasi merupakan permainan bongkar pasang yang memiliki gambar berbagai macam kendaraan darat, laut dan udara. Fungsinya selain untuk melatih motorik, juga untuk stimulasi otak kanan dan otak kiri.

6. *Puzzle* geometri

Puzzle geometri merupakan *puzzle* yang dapat mengembangkan keterampilan mengenali bentuk geometri (segitiga, lingkaran, persegi dan lain-lain).

7. *Puzzle* Angka silang

Puzzle angka silang merupakan *puzzle* yang dapat mengembangkan kemampuan logika matematika.

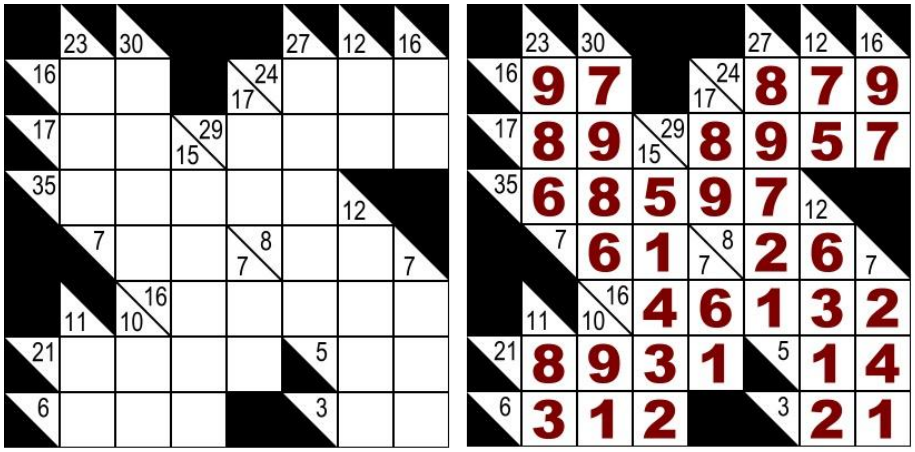
2.3.3 *Puzzle* Kakuro

Kakuro adalah sebuah permainan *puzzle* yang bermula bernama *cross sums. Puzzle,* pertama dikeluarkan pada tahun 1966 oleh *Dell Megazine* di Amerika Serikat. Sepuluh tahun kemudian, *Dell Megazine* memperkenalkan *puzzle* sudoku pada dunia. Maki Kaji presiden dari *Nicoli Puzzle,* pada tahun 1980 membawa masuk *cross games* ke Jepang. Maki Kaji memberi nama *puzzle* buatannya adalah kasan kurosu (penjumlahan silang). Pada tahun 1986, *Nicoli* memberi nama dagang *game* dengan nama kakuro (kependekan dari kasan kurosu). September 2005 adalah titik penentunya, dimana *game puzzle* kakuro diperkenalkan ke barat oleh *The Guardian* dan *The Daily Mail* yang menerbitkan *puzzle* kakuro setiap harinya di Inggris dan setelah itu *game puzzle* kakuro ini menyebar keseluruh dunia [2].

2.3.3.1 Definisi *Puzzle* Kakuro

Puzzle kakuro adalah permainan *puzzle* dengan ukuran *grid* N x M dengan mempunyai dua jenis kotak yaitu kotak berwarna hitam dan kotak berwarna putih. Pada kedua jenis kotak tersebut mempunyai dua posisi yaitu

horizontal dan vertikal. Kotak berwarna hitam digunakan sebagai tanda pembatas dan soal untuk menyelesaikan permainan *puzzle* kakuro, sedangkan kotak berwarna putih digunakan sebagai kotak jawaban. Pada baris paling atas dan kolom paling kiri pada *grid puzzle* kakuro, hanya berisi kotak berwarna hitam (dapat berfungsi sebagai pembatas atau kotak jawaban). Aturan dalam pengisisan *puzzle* kakuro ini adalah dengan mengisi angka bilangan bulat dari 1 hingga 9 dengan tidak boleh terdapat angka yang sama pada satu lajur (baris maupun kolom). Nilai angka yang terdapat pada kotak jawaban yang sudah dijumlahkan harus bernilai sama dengan nilai angka pada kotak soal disetiap lajurnya [1]. Gambar 2.1 adalah sebagian contoh dari *puzzle* kakuro serta solusi penyelesaianya [6].



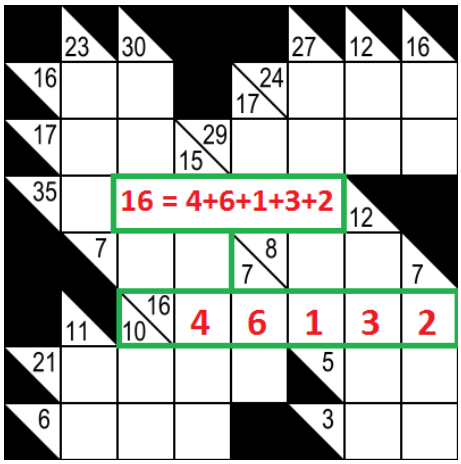
Gambar 2.1 Contoh *puzzle* kakuro dan solusi penyelesaiannya

2.3.3.2 Aturan Permainan *Puzzle* Kakuro

Puzzle kakuro adalah adalah sebuah permainan *puzzle* yang bersifat *puzzle* logika. *Puzzle* ini biasa disebut transliterasi matematis dari *puzzle* silang. Aturan dalam permainan *puzzle* kakuro ini adalah dengan mengisi setiap setiap kotak jawaban yang tersedia dengan angka bulat dari 1 hingga 9 dengan beberapa syarat dalam pengisiannya. Adapun syarat-syarat dalam pengisian kotak jawaban adalah sebagai berikut [1]:

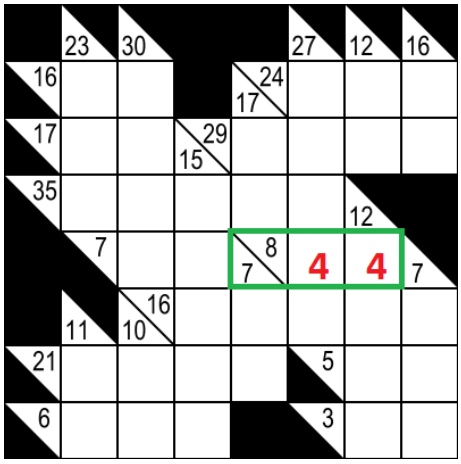
1. Setiap kotak jawaban hanya boleh diisikan dengan bilangan bulat dari angka 1 hingga 9.

2. Setiap kotak jawaban yang berurutan dalam satu lajur (lajur yang dimaksud adalah deretan kotak yang berurutan dalam satu baris atau kolom) tidak boleh memiliki angka yang sama.
3. Isi dari setiap lajur harus memiliki jumlah yang sama dengan kotak soal pada ujung kiri (baris) atau ujung atas (kolom) lajur tersebut.



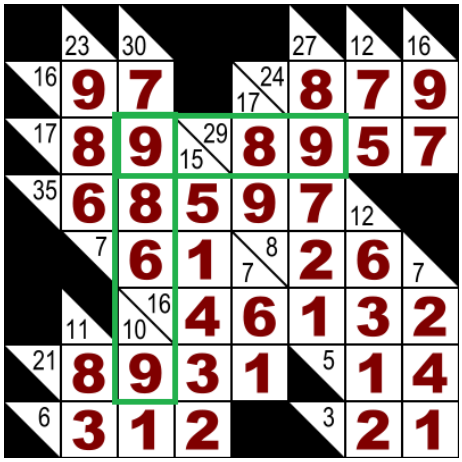
Gambar 2.2 Cara pengisisan dengan jumlah jawaban sesuai dengan kotak soal

Pengisian kotak jawaban seperti pada gambar gambar 2.3 merupakan contoh yang salah karena terdapat angka yang sama pada satu lajur.



Gambar 2.3 Nilai sama pada lajur

Sedangkan pada gambar 2.4 merupakan pengisian kotak jawaban yang benar karena terdapat pada kotak soal yang berbeda walaupun dalam lajur yang sama.



Gambar 2.4 Nilai sama pada lajur yang sama tetapi pada kotak jawaban yang berbeda

2.3.3.3 Fungsi Objektif *Puzzle* Kakuro

Fungsi Objektif untuk baris [8]:

$f = \sum_{h \in H} |TH(h) - CH(h)|$ (2.4)

TH(h) = Jumlah yang terdapat dalam Kotak Soal secara baris

CH(h) = Hasil Perhitungan Penjumlahan dari Vektor Solusi v secara baris.

Fungsi Objektif untuk kolom :

$f = \sum_{v \in V} |TV(v) - CV(v)|$ (2.5)

TV(v) = Jumlah yang terdapat dalam Kotak Soal secara kolom.

CV(v) = Hasil Perhitungan Penjumlahan dari Vektor Solusi secara kolom.

1.4 Tools yang digunakan

Tools adalah unsur yang diguunakan dalam pembangunan sebuah aplikasi

1.4.1 OOP (*Object Oriented Progamming*)

Metodologi berorientasi objek adalah suatu strategi pembangunan perangkat lunak yang mengorganisasikan perangkat lunak sebagai kumpulan objek yang berisi data dan operasi yang diberlakukan terhadapnya. Metodologi berorientasi objek merupakan suatu cara bagaimana sistem perangkat lunak dibangun melalui pendekatan objek secara sistematis. Metode berorientasi objek didasarkan pada penerapann prinsip-prinsip pengelolaan kompleksitas. Metode berorientasi objek meliputi rangkaian aktifitas analisis beorientasi objek, perancangan berorientasi objek, pemrograman berorientasi objek, dan pengujian berorientasi objek.

Pada saat ini, metode berorientais objek banyak dipilih karena metodologi lama banyak menimbulkan masalah seperti adanya kesulitan pada saat mentranformasi hasil dari satu tahap pengembangan ke tahap berikutnya, misalnya pada metode pendekatan terstruktur, jenis aplikasi yang dikembangkan saat ini berbeda dengan masa lalu. Aplikasi yang dikembangkan saat ini sangat beragam (aplikasi bisnis, *real-time, utility* dan sebagainya) dengan platform yang berbeda-beda, sehingga menimbulkan tuntutan kebutuhan metodologi pengembangan yang dapat mengakomodasi ke semua jenis aplikasi.

Keuntungan menggunakan metodologi berorientasi objek adalah sebagai berikut [9]:

1. Meningkatkan Produktivitas

   Karena kelas dan objek yang ditemukan dalam suatu masalah masih dapat dipakai ulang untuk masalah lainnya yang melibatkan objek tersebut (*reuseable*).
2. Kecepata Pengembangan

   Karena sistem yang dibangun dengan baik dan benar pada saat analisis dan perancangan akan mennyababkan berkurangnya kesalahan pada saat pengkodean
3. Kemudahan Pemeliharaan

Karena dengan model objek, pola-pola yang cendrung tetap dan stabil dapat dipisahkan dan pola-pola yang mungkin sering diubah-ubah.

4. Adanya Konsistensi

   Karena sifat pewarisan dan penggunaan notasi yang sama pada saat analisis, perancangan maupun pengkodean.

5. Meningkatkan Kualitas Perangkat Lunak

   Karena adanya pendekatan pengembangan lebih dekat dengan dunia nyata dan adanya konsistensi pada saat pengambangannya, perangkat lunak yang dihasilkan akan mampu memenuhi kebutuhan pemakai serta mempunyai sedikit kesalahan.

Berikut beberapa contoh bahasa pemograman yang mendukung pemrograman berorinentasi objek adalah :

1. Smalltalk

   Smalltalk adalah salah satu bahasa pemograman yang diekmbangkan untuk mendukung pemrograman beroirentasi objek.

2. Bahasa Pemrograman Eiffel

   Eiffel merupakan bahsa pemrograman yang kembangkan untuk mendukung pemrograman berorientasi objek oleh Bertrand Meyer dan *compiler*.

3. Bahasa Pemrograman (Web) PHP

   Php dibuat pertama kali oleh seorang perekayasa perangkat (software engineering) yang bernama Rasmus Lerdoff.

4. Bahasa Pemrograman C++

   C++ merupakan pengembangan lebih lanjut dari bahasa pemrograman C untuk mendukung pemrograman berorientasi objek.

5. Bahasa Pemrograman Java

   Java dikembangkan oleh perusahaan Sun Microsystem. Java menurut definisi dari Sun Microsystem adalah nama untuk sekumpulan teknologi untuk membuat dan menjalankan perangkat lunak pada komputer *standalone* ataupun pada lingkungan jaringan.

1.4.2 Konsep Dasar Berorientasi Objek

Berikut adalah konsep dasar pemrograman berorientasi objek [9]:

1. Objek (*Object*)

   Objek adalah abtraksi dan sesuatu yang mewakili dunia nyata seperti benda, satuan organisasi, tempat, kejadian, struktur, status, atau hal-hal lain yang bersifat abstrak. Objek merupakan suatu entitas yang mampu menyimpan informasi dan mempunyai operasi yang dapat diterapkan atau dapat berpengaruh pada status objeknya.

2. Kelas (*Class*)

   Kelas adalah kumpulan objek-objek dengan karakteristik yang sama. Kelas merupakan definisi statik dan himpunan objek yang sama yang mungkin lahir atau diciptakan dalam kelas tersebut.

3. Pembungkusan (*Encapsulation*)

   Pembungkusan atribut data dan layanan yang mempunyai objek untuk menyembunyikan implementasi dan objek sehingga objek lain tidak mengetahui cara kerjanya.

4. Pewarisan (*Inheritance*) dan Generalisasi/Spesialisasi

   Mekanisme yang memungkinkan satu objek mewarisi sebagian atau seluruh definisi dan objek lain sebagai bagian dirinya.

5. Metode

   Operasi atau metode pada sebuah kelas hampir sama dengan fungsi atau prosedur pada metodologi struktural.

6. *Polimorfisme*

Kemampuan suatu objek untuk digunakan dibanyak tujuan yang berbeda dengan nama yang sama sehingga menghemat baris program.

1.4.3 Pemodelan dengan UML

UML (*Unified Modeling Language*) adalah salah satu alat bantu yang sangat handal di dunia pengembangan sistem yang berorientasi obyek. Hal ini disebabkan karena *UML* menyediakan bahasa pemodelan visual yang memungkinkan bagi pengembang sistem untuk membuat cetak biru atas visi

mereka dalam bentuk yang baku, mudah dimengerti serta dilengkapi dengan mekanisme yang efektif untuk berbagi dan mengkomunikaskan rancangan mereka dengan yang lain.

UML merupakan kesatuan dari bahasa pemodelan yang dikembangkan oleh *Booch*, *Object Modeling Technique* (OMT) dan *Object Oriented Software Engineering* (OOSE). Metode *Booch* dari *Grady Booch* sangat terkenal dengan nama metode *Design Object Oriented*. Metode ini menjadikan proses analisis dan *design* ke dalam empat tahapan *iterative*, yaitu : identifikasi kelas-kelas dan obyek-obyek, identifikasi *semantic* dari hubungan obyek dan kelas tersebut, perincian *interface* dan implementasi. Keunggulan metode *Booch* adalah pada detail dan kayanya dengan notasi dan elemen. Pemodelan *Object Modeling Technique* yang dikembangkan *Rumbaugh* didasarkan pada analisis terstruktur dan permodelan *entity-relationship*. Tahapan utama dalam metodologi ini adalah analisis, *design* sistem, *design* obyek dan implementasi. Keungulan metode ini aladah dalam penotasian yang mendukung semua konsep *Object Oriented Software Engineering*. Metode *Object Oriented Software Engineering Jacobson* lebih memberikan penekanan pada *use case*. *Object Oriented Software Engineering* memiliki tiga tahapan yaitu membuat model *requirement* dan analisis, *design* dan implementasi, dan model pengujian. Keungulan *metode* ini adalah mudah dipelajari karena memiliki notasi yang sederhana namun mencakup seluruh tahapan dalam rekayasa perangkat lunak. Dengan *UML*, metode *Booch*, *Object Modeling Technique* dan *Object Oriented Software Engineering* digabungkan dengan membuang elemen-elemen yang tidak praktis ditambah dengan elemen-elemen dari metode lain yang lebih efektif dan elemen-elemen baru yang belum ada pada metode terdahulu sehingga *UML* lebih *ekspresif* dan seragam dari pada *metode* lainnya.

2.4.4 Diagram-diagram UML

Didalam *UML* terdapat beberapa macam diagram yang dapat menggambarkan suatu sistem, berikut adalah beberapa diagram yang terdapat di dalam *UML*.

2.4.4.1 Struktur Diagram

Struktur Diagram, yaitu kumpulan diagram yang digunakan untuk menggambarkan suatu struktur statis dari sistem yang dimodelkan. Pada Struktur Diagram dibagi menjadi 6 bagian [9]:

1. Diagam Kelas

   Diagram kelas menggambarkan struktur sistem dari segi pendefinisian kelas-kelas yang akan dibuat untuk membangun sistem. Kelas memiliki apa yang disebut attribut dan metode atau operasi.

2. Diagram Objek

   Digram objek menggambarkan struktur sistem dari segi penamaan objek dan jalannya objek dalam sistem.

3. Diagram Komponen

   Diagram komponen dibuat untuk menunjukan organisasi dan ketergantungan diantara kumpulan komponen dalam sebuah sistem.

4. *Composite Structure Diagram*

   Composite Structure Diagram bru muali ada pada UML versi 2.0. diagram ini dapat digunakan untuk menggambarkan struktur dari bagian-bagian yang saling terhubung maupun mendeskripsikan struktur pada saat berjalan (*runtime*) .

5. *Package Diagram*

   Package diagram menyediakan cara mengumpulkan elemen-elemen yang saling terkait dalam diagram UML. Hampir semua diagram dalam UML dapat dikelompokan menggunakan *Package Diagram*.

6. *Deployment Diagram*

   Deployment diagram menunjukan konfigurasi komponen dalam proses eksekusi aplikasi.

2.4.4.2 *Behavior Diagram*

Behavior Diagram, yaitu kumpulan diagram yang digunakan untuk menggambarkan kelakuan sistem atau rangkaian perubahan yang terjadi pada sebuah sistem. Pada *Behavior Diagram* dibagi menjadi 3 bagian [9]:

1. *Use Case Diagram*

   Use case diagram merupakan pemodelan untuk kelakuan (behavior) sistem informasi yang akan dibuat,. *Use case* mendeskripsikan sebuah interaksi antara satu atau lebih aktor dengan sistem informasi yang akan dibuat.
2. *Activity Diagram*

   Activity diagram menggambarkan *workflow* atau aktivitas dari sebuah sistem atau proses bisnis atau menu yang ada pada perangkat lunak.
3. *State Machine Diagram*

   State machine diagram digunakan untuk menggambarkan perubahan status atau transisi status dari sebuah mesin atau sistem atau objek.

2.4.4.3 *Interactions Diagram*

Interactions Diagram, yaitu kumpulan diagram yang digunakan untuk menggambarkan interaksi antar subsistem pada suatu sistem. Pada Interactions Diagram dibagi menjadi 4 bagian [9]:

1. *Sequence Diagram*

   Sequence diagram menggambarkan kelakuan objek pada *use case* dengan mendeskripsikan waktu hidup objek dan message yang dikirimkan dan diterima antar objek.
2. *Communication Diagram*

   Communication diagram menggambarkan interaksi antar bojek/bagian dalam bentuk urutan pengiriman pesan. Diagram komunikasi merepresentasikan informasi yang diperoleh dari diagram kelas, diagram *sequence*, dan diagram *use case* untuk

mendeskripsikan gabungan antara struktur statis dan tingkah laku dinamis dari suatu sistem.

3. *Timing Diagram*

   Timing diagram merupakan diagram yang fokus pada penggambaran terkait batas waktu.

4. *Interaction Overview Diagram*

   Interaction overview diagram mirip dengan diagram aktivitas yang berfungsi untuk menggarbarkan sekumpulan urutan aktivitas, diagram ini adlaah bentuk aktivias diagram yang setiap titik merepresentasikan diagram interaksi.

2.5 Bahasa Pemograman C#

C# (tanda '#' dibaca "Sharp") merupakan bahasa pemograman baru yang diciptakan Microsoft seacra khusus sebagai salah satu bahasa pemrograman dalam teknologi .Net sebagai bahasa baru, C# tidak berevolusi dari bahasa C# versi bukan teknologi .Net. dengan demikian C# dapat memaksimalkan kemampuannya tanpa khawatir dengan masalah kompatibilitas dengan versi-versi sebelumnya. Keharusan sebuah perangkat lunak untuk tetap dapat kompatibel dengan versi-versi sebelumnya sebagaimana yang terjadi pada Visual Basic (VB) maupun C++biasanya menghambat optimalitas kemampuan dari perangkat lunak tersebut [10].

Sejak diluncurkan pada tahun 2000, C# dengan cepat merebut hati progammer C++ bahkan VB. Dengan tata cara penulisan yang mirip C++ dan *interface* mirip VB 6.0 menurut wikipedia, sebuah ensiklopedia gratis di internet pengguna C# .Net pada saat ini sudah melebihi pengguna VB.Net. sementara itu jumlah pengguna bahasa pemrograman lain masih berada dibawah jumlah pengguna VB .Net. masih menurut wikipedia, jumlah buku C# yang terjual pun berada dikisaran 2 hingga 3 kali lebih banyak dari jumlah buku VB yang terjual.

Dari informasi ini dapat disimpulkan bahwa C# merupakan bahasa pemrogrman baru yang sedang berkembang dan dapat diterima dengan baik oleh kebanyakan progammer dan kalangan industri. Di Microsoft sendiri, C#

merupakan bahasa pemrograman yang digunakan untuk membuat perangkat lunak yang bertteknologi .Net dengan demikian dapat diperkirakan bahwa C# akan menjadi bahasa pemrograman yang akan banyak digunakan di masa-masa mendatang.