

LAPORAN TUGAS BESAR 1

IF2211 Strategi Algoritma

Pemanfaatan Algoritma Greedy

dalam Pembuatan Bot Permainan Diamonds



Disusun oleh

Shafiq Irvansyah : 13522003

Ellijah Darrellshane S : 13522097

Maulana Muhammad Susetyo : 13522127

**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2024**

Daftar Isi

BAB 1.....	3
Deskripsi tugas.....	3
BAB 2.....	6
Landasan Teori.....	6
BAB 3.....	7
Aplikasi Strategi Greedy.....	7
BAB 4.....	8
Implementasi dan Pengujian.....	8
BAB 5.....	9
Kesimpulan dan Saran.....	9

BAB 1

Deskripsi tugas

Diamonds merupakan suatu programming challenge yang mempertandingkan bot yang anda buat dengan bot dari para pemain lainnya. Setiap pemain akan memiliki sebuah bot dimana tujuan dari bot ini adalah mengumpulkan diamond sebanyak-banyaknya. Cara mengumpulkan diamond tersebut tidak akan sesederhana itu, tentunya akan terdapat berbagai rintangan yang akan membuat permainan ini menjadi lebih seru dan kompleks. Untuk memenangkan pertandingan, setiap pemain harus mengimplementasikan strategi tertentu pada masing-masing bot-nya. Penjelasan lebih lanjut mengenai aturan permainan akan dijelaskan di bawah.

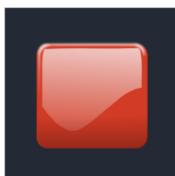
Komponen-komponen dari permainan Diamonds antara lain:

1. Diamonds



Untuk memenangkan pertandingan, kita harus mengumpulkan diamond ini sebanyak-banyaknya dengan melewati/melangkahinya. Terdapat 2 jenis diamond yaitu diamond biru dan diamond merah. Diamond merah bernilai 2 poin, sedangkan yang biru bernilai 1 poin. Diamond akan di-regenerate secara berkala dan rasio antara diamond merah dan biru ini akan berubah setiap regeneration.

2. Red Button/Diamond Button



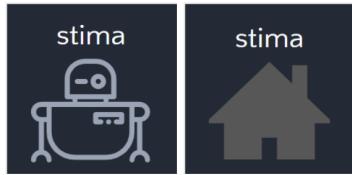
Ketika red button ini dilewati/dilangkahi, semua diamond (termasuk red diamond) akan di-generate kembali pada board dengan posisi acak. Posisi red button ini juga akan berubah secara acak jika red button ini dilangkahi.

3. Teleporters



Terdapat 2 teleporter yang saling terhubung satu sama lain. Jika bot melewati sebuah teleporter maka bot akan berpindah menuju posisi teleporter yang lain.

4. Bots and Bases



Pada game ini kita akan menggerakkan bot untuk mendapatkan diamond sebanyak banyaknya. Semua bot memiliki sebuah Base dimana Base ini akan digunakan untuk menyimpan diamond yang sedang dibawa. Apabila diamond disimpan ke base, score bot akan bertambah senilai diamond yang dibawa dan inventory (akan dijelaskan di bawah) bot menjadi kosong.

5. Inventory

Name	Diamonds	Score	Time
stima	♥♥	0	43s
stima2	♥	0	43s
stima1	♥♥♥♥	0	44s
stima3	♥	0	44s

Bot memiliki inventory yang berfungsi sebagai tempat penyimpanan sementara diamond yang telah diambil. Inventory ini memiliki kapasitas maksimum sehingga sewaktu waktu bisa penuh. Agar inventory ini tidak penuh, bot bisa menyimpan isi inventory ke base agar inventory bisa kosong kembali.

BAB 2

Landasan Teori

2.1. Dasar Teori

Algoritma Greedy adalah algoritma yang memecah masalah menjadi langkah-langkah dimana pada setiap langkah mencari pilihan yang terbaik, sehingga diharapkan mendapatkan solusi yang optimal/mendekati optimal. Dikarenakan Algoritma Greedy tidak melakukan exhaustive search, solusi yang dihasilkan disebut dengan optimum lokal, dari pengambilan nilai optimum lokal pada setiap langkah, diharapkan tercapai optimum global, yaitu tercapainya solusi optimum yang melibatkan keseluruhan langkah dari awal sampai akhir. Sehingga diperlukan pembuktian untuk memverifikasi solusi yang dihasilkan adalah solusi optimal.

Walaupun Algoritma Greedy tidak selalu menghasilkan solusi optimal, tapi jika dibandingkan dengan Algoritma Brute Force, Algoritma Greedy memiliki waktu komputasi yang jauh lebih kecil, hal ini merupakan *trade off* yang didapatkan dengan menggunakan Algoritma Greedy.

Elemen-elemen algoritma greedy:

- Himpunan kandidat, C : berisi kandidat yang akan dipilih pada setiap Langkah (misal: simpul/sisi di dalam graf, job, task, koin, benda, karakter, dsb)
- Himpunan solusi, S : berisi kandidat yang sudah dipilih
- Fungsi solusi: menentukan apakah himpunan kandidat yang dipilih sudah memberikan solusi
- Fungsi seleksi (selection function): memilih kandidat berdasarkan strategi greedy tertentu. Strategi greedy ini bersifat heuristik.
- Fungsi kelayakan (feasible): memeriksa apakah kandidat yang dipilih dapat dimasukkan ke dalam himpunan solusi (layak atau tidak)
- Fungsi obyektif : memaksimumkan atau meminimumkan

2.2. Cara Kerja Program

Permainan diamonds ini merupakan permainan berbasis web. Script python yang dijalankan akan mengirim HTTP request terhadap API endpoint yang sudah disediakan oleh backend. Berikut merupakan proses yang terjadi:

1. Program bot akan mengirimkan POST request terhadap endpoint /api/bots/recover dengan body email dan password untuk mengecek apakah bot sudah terdaftar.
2. Jika bot belum terdaftar, maka bot akan mengirimkan POST request terhadap endpoint /api/bots dengan body email dan password untuk mendaftarkan bot.
3. Ketika ID bot sudah diketahui, bot bergabung ke permainan dengan mengirimkan POST Request terhadap endpoint /api/bots/{id}/join dengan body berisi board id yang diinginkan.
4. Untuk menjalankan bot pada board, program bot mengirimkan POST request terhadap endpoint /api/bots/{id}/move dengan body yang berisi direction

- (“NORTH”, “SOUTH”, “EAST”, atau “WEST”). Langkah ini dilakukan terus menerus sampai waktu permainan habis.
5. Program frontend secara periodik juga mengirimkan GET request terhadap endpoint /api/boards/{id} untuk mendapatkan kondisi board yang terbaru.

Logic bot permainan Diamonds dapat dibuat dengan menggunakan bahasa Python di direktori /src/game/logic. Pada file tersebut, impor beberapa file penting, yaitu game.logic.base dan game.models untuk mengakses class yang diperlukan. Kemudian, buatlah sebuah kelas yang memiliki metode next_move dengan parameter self, board_bot (kelas GameObject), dan board (kelas Board). Dari parameter board_bot, kita dapat mengakses berbagai objek permainan seperti bot, diamond button, diamond, teleporter, serta propertinya untuk mendesain algoritma greedy bot tersebut. Algoritma greedy dapat diimplementasikan dengan merancang next_move untuk menentukan arah gerak bot. Arah gerak bot tersebut kemudian dikembalikan dari metode next_move untuk dikirim ke titik endpoint API. Setelah merancang logic bot, logic tersebut didaftarkan pada *dictionary* CONTROLLERS di file /src/main.py sehingga bisa dijalankan melalui CLI.

Untuk menjalankan bot, lakukan langkah-langkah berikut:

1. Clone repository dan masuk ke root directory project tersebut.

```
git clone https://github.com/shafIrv/Tubes1_PCS.git
```

2. Masuk ke directory /src.

```
cd src
```

3. Untuk menjalankan 1 bot saja,

```
python main.py --logic <LOGIC_NAME> --email=<UNIQUE_EMAIL>
--name=<NAME> --password=<PASSWORD> --team <TEAM_NAME>
```

4. Untuk menjalankan lebih dari beberapa bot, gunakan script yang sudah dibuat

Untuk Windows,

```
./run-bots.bat
```

Untuk Linux / (possibly) macOS,

Sebelum menjalankan script, pastikan untuk memberikan izin kepada shell script agar bisa dieksekusi

```
chmod +x run-bots.sh
```

Jalankan script untuk banyak bot dengan command

```
./run-bots.sh
```

BAB 3

Aplikasi Strategi Greedy

3.1. Proses Pemetaan Persoalan Diamonds Menjadi Elemen-Elemen Algoritma Greedy

Elemen-elemen algoritma greedy:

- **Himpunan kandidat, C :**

Seluruh objek dalam game yang dapat berinteraksi dengan bot kita seperti diamond (biru & merah), diamond button, base, teleporter, dan bot lawan.

- **Himpunan solusi, S :**

Semua objek dalam game yang telah berinteraksi oleh bot kita. Contohnya adalah berlian yang sudah diambil, teleporter yang sudah dilewati, base yang sudah dilewati, *diamond button* yang sudah dilewati, dan lain sebagainya.

- **Fungsi solusi:**

Fungsi untuk menentukan apakah himpunan solusi yang telah dipilih telah memenuhi kriteria solusi atau belum. Fungsi solusi akan mengembalikan nilai true jika waktu permainan sudah habis .

- **Fungsi seleksi (selection function):**

Fungsi seleksi merupakan sebuah mekanisme untuk memilih kandidat berdasarkan kriteria tertentu. Pendekatan fungsi seleksi ini bervariasi tergantung pada strategi greedy yang digunakan. Penjelasan lebih detail tentang fungsi seleksi akan disajikan dalam bab 3.2 untuk setiap strategi greedy yang kami rancang, karena setiap strategi memiliki pendekatan yang berbeda.

- **Fungsi kelayakan (feasible):**

Bot kami mengecek jika sedang menuju suatu *diamond* namun waktunya tidak cukup, maka target tersebut menjadi tidak valid dan bot akan kembali ke base-nya.

- **Fungsi obyektif :**

Memaksimalkan jumlah berlian yang diambil dalam batas waktu yang terbatas. Penjelasan lebih rinci tentang fungsi tujuan akan dipaparkan dalam bab 3.2 untuk setiap strategi greedy yang kami rancang, karena tiap-tiap strategi memiliki fungsi tujuan yang berbeda.

3.2. Eksplorasi alternatif solusi greedy yang mungkin dipilih dalam persoalan Diamonds

1. Manhattan ([manhattan.py](#))

Strategi Manhattan adalah strategi greedy yang memilih diamond berdasarkan *Manhattan Distance* terdekat. Manhattan distance dikenal sebagai jarak kota blok, adalah metrik dalam geometri yang mengukur jarak antara dua titik dalam sebuah grid berbasis grid orthogonal. Dalam konteks dua dimensi, Manhattan distance dihitung sebagai jumlah perbedaan absolut antara koordinat x dan y dari dua titik.

$$\text{Manhattan Distance} = |x_1 - x_2| + |y_1 - y_2|$$

Logic ini juga sudah meng-handle jika waktunya mainnya akan habis, maka bot akan segera kembali ke base untuk mengumpulkan diamond seadanya.

2. Highest Density with Relatively Closest Distance (`closeNdense.py`)

Strategi Highest Density with Relatively Closest Distance (HDRCD) adalah strategi greedy yang memilih diamond dengan densitas poin terbesar dengan jarak yang relatif dekat dengan bot. Program akan mencari *Manhattan Distance* paling dekat ke suatu *diamond* terlebih dahulu. Jarak terdekat tersebut kemudian akan dijadikan patokan untuk menilai jarak bot ke *diamond* lain apakah relatif dekat atau tidak. Suatu *diamond* dikatakan relatif dekat jika *Manhattan Distance*-nya tidak melebihi *Manhattan Distance* minimum + 2 satuan unit. Kemudian *diamond* yang dikategorikan sebagai relatif dekat akan dibandingkan mana yang memiliki densitas poin tertinggi. Densitas poin dihitung dengan menjumlah diamond yang dikategorikan sebagai relatif dekat dengan *diamond* yang bertetanggan jika ada. Diamond dikatakan bertetanggan jika bersebelahan secara horizontal, vertikal, atau diagonal.



Gambar 3.1 Penggambaran Ketetanggan

Diamond yang dilinkari merah merupakan *diamond* *relatively close*, *diamond* yang dilingkari biru merupakan *diamond* yang bertetanggan dengan yang dilinkari merah. Area yang dikotaki kuning adalah area dari ketetanggan *diamond* yang dilinkari merah.

3. Most Valuable Pilihan (`notrandom.py`)

Strategi Most Valuable Pilihan (MVP) merupakan strategi yang mengevaluasi pilihan dengan ‘value’ terbesar. Value dari suatu pilihan dihitung dari jarak bot dengan suatu objek dan bobot dari objek tersebut. Sebagai contoh, nilai dari *diamond* biru/normal adalah 1. Nilai dari *diamond* merah diberi bobot yang lebih tinggi dari *diamond* biru sehingga jarak *diamond* merah dianggap lebih kecil oleh strategi MVP. Strategi MVP juga mengevaluasi button, dimana bot akan menuju button apabila tidak ada *diamond* di sekitar bot. Reporter juga dimasukkan di evaluasi jarak, dimana akan dibandingkan jarak *diamond*/bot tanpa reporter dengan jarak *diamond*/bot menggunakan reporter.

3.3. Analisis efisiensi dan efektivitas dari kumpulan alternatif solusi greedy yang dirumuskan

1. Manhattan

Strategi Manhattan cukup relatif simpel sehingga relatif dapat dapat mengumpulkan *diamond* secara cepat. Namun ada beberapa *edge case* yang berlum ter-handle sehingga menyebabkan bot menjadi stuck, contoh diantaranya adalah stuck saat inventory bot sisa slot 1, tetapi yang diincar adalah diamond merah. Sehingga target dari bot akan selalu menuju ke *diamond* merah karena merupakan yang terdekat, tetapi bot tidak bisa mengambilnya karena slotnya sisa 1.

2. Highest Density with Relatively Closest Distance

Strategi HDRCD cukup efektif dalam mengambil diamond yang berkerumput. Namun dalam implementasinya, sering kali bot akan cenderung memilih yang lebih jauh karena *value* nya lebih besar, sehingga sering terjadi kasus dimana bot tidak mengambil *diamond* yang tepat bersebelahan. Hal ini terjadi cukup sering sehingga banyak *diamond* yang tidak diambil secara efektif dan efisien. Logic ini juga belum mengatasi jika akan melewati teleporter, sehingga objektifnya bisa tidak tercapai akibat tidak sengaja melewati teleporter.

3. Most Valuable Pilihan

Strategi Most Valuable Pilihan (MVP) memiliki efektifitas cukup tinggi dalam mengumpulkan diamonds. Karena strategi MVP berusaha untuk mencari pilihan terbaik pada setiap saat, maka langkah yang diambil seharusnya memiliki efektifitas yang tinggi. Tetapi karena tidak adanya pathfinding, strategi tidak mencari jalur terpendek untuk mengambil secukupnya diamond dan balik ke base, sehingga mengurangi efisiensi.

3.4. Strategi greedy yang dipilih (yang akan diimplementasikan dalam program) beserta alasan dan pertimbangan pemilihan strategi tersebut.

Dari beberapa rancangan logic bot yang telah didesain, algoritma "Most Valuable Pilihan" (*notrandom.py*) telah terpilih. Pemilihan ini didasarkan pada uji coba yang telah dilakukan berulang kali, dimana algoritma ini telah mengalami pengujian pada berbagai *edge-case*, tanpa mengalami kebuntuan atau kesalahan yang signifikan. Logika ini juga memiliki kemampuan dinamis untuk menyesuaikan diri ketika parameter-parameter papan permainan diubah. Dibandingkan dengan logika lainnya, algoritma ini menunjukkan keunggulan dalam mengumpulkan berlian.

BAB 4

Implementasi dan Pengujian

4.1. Implementasi algoritma greedy pada program bot yang digunakan

```
function next_move(self, board_bot: GameObject, board: Board) -> delta_x, delta_y

Deklarasi
diamond_list: list
distance_list: list
distance_pos_list, distance_pos_tp_list: zipped list
teleporter_list: list
button_list: list
dist_tp1, dist_tp2, close_tp_dist, dist_but, dist_base: int

Inisialisasi
props ← board_bot.properties
current_pos_x ← board_bot.position.x
current_pos_y ← board_bot.position.y
base ← board_bot.properties.base

diamond_list ← [d for d in board.game_objects if d.type ← "DiamondGameObject"]
distance_list, distance_tp_list ← [i traversal 0..len(diamond_list)]
distance_pos_list ← list(zip(distance_list,diamond_list))
distance_pos_tp_list ← list(zip(distance_tp_list,diamond_list))
teleporter_list ← [t for t in board.game_objects if t.type = "TeleportGameObject"]
button_list ← [b for b in board.game_objects if b.type = "DiamondButtonGameObject"]

Algoritma
i traversal 0..len(diamond_list):
    distance_list[i] ← abs(current_pos_x - diamond_list[i].position.x) +
    abs(diamond_list[i].position.y - current_pos_y)
    if diamond_list[i].properties.points ← 2 then
        Distance_list[i] ← distance_list[i]*0.6
    distance_pos_list.sort(key=lambda x: x[0])

    tp_1 ← teleporter_list[0]
    tp_2 ← teleporter_list[1]
    dist_tp1 ← abs(tp_1.position.x - current_pos_x) + abs(tp_1.position.y - current_pos_y)
    dist_tp2 ← abs(tp_2.position.x - current_pos_x) + abs(tp_2.position.y - current_pos_y)

    close_tp ← tp_1
    close_tp_dist ← dist_tp1
    far_tp ← tp_2
    if dist_tp1 > dist_tp2:
        close_tp ← tp_2
        close_tp_dist ← dist_tp2
        far_tp ← tp_1

    distance_tp_list ← [i traversal 0..len(diamond_list) (len(diamond_list))]
    i traversal 0..len(diamond_list):
        distance_tp_list[i] ← abs(far_tp.position.x-diamond_list[i].position.x) + abs(far_tp.position.y -
        diamond_list[i].position.y) + close_tp_dist
        if diamond_list[i].properties.points = 2 then
            distance_tp_list[i]*=0.6
    distance_pos_tp_list.sort(key=lambda x: x[0])
```

```

buton ← buton_list[0]
dist_but ← (abs(buton.position.x - current_pos_x) + abs(buton.position.y - current_pos_y))*1.21 + 2
dist_base ← abs(current_pos_x - board_bot.properties.base.x) + abs(board_bot.properties.base.y - current_pos_y)
dist_base_tp ← abs(far_tp.position.x - board_bot.properties.base.x) + abs(board_bot.properties.base.y - far_tp.position.y) + close_tp_dist
dist_base ← abs(current_pos_x - board_bot.properties.base.x) + abs(board_bot.properties.base.y - current_pos_y)
dist_base_tp ← abs(far_tp.position.x - board_bot.properties.base.x) + abs(board_bot.properties.base.y - far_tp.position.y) + close_tp_dist
min_dist ← min(dist_base,dist_base_tp)+1
if props.diamonds = 5 or min_dist>=(props.milliseconds_left / 1000) then
    if min_dist = (dist_base+1) then
        self.goal_position ← base
    else:
        self.goal_position ← close_tp.position

else:
    if props.diamonds = 4 and distance_pos_list[0][1].properties.points = 2 then
        i←1
        while i<len(diamond_list)-1 do
            closest ← min(dist_but, distance_pos_tp_list[i][0], distance_pos_list[i][0])
            if closest = distance_pos_list[i][0] and distance_pos_list[i][1].properties.points=1 then
                self.goal_position ← distance_pos_list[i][1].position
                break
            else if closest=distance_pos_tp_list[i][0] and
distance_pos_tp_list[i][1].properties.points=1 then
                self.goal_position ← close_tp.position
                break
            else if distance_pos_tp_list[i][0] > dist_but and distance_pos_list[i][0] > dist_but then
                self.goal_position ← buton.position
                break
            else
                i++
        if distance_pos_list[i][1].properties.points=2 then
            self.goal_position ← base
        else
            self.goal_position ← distance_pos_list[i][1].position

    else:
        closest ← min(dist_but, distance_pos_tp_list[0][0], distance_pos_list[0][0])
        if closest = distance_pos_list[0][0] then
            self.goal_position ← distance_pos_list[0][1].position
        else if closest = distance_pos_tp_list[0][0] then
            self.goal_position ← close_tp.position
        else
            self.goal_position ← buton.position

    if current_pos_x = self.goal_position.x and current_pos_y = self.goal_position.y then
        self.goal_position ← base
        current_position ← board_bot.position
    if self.goal_position then
        delta_x, delta_y ← get_direction(
            current_position.x,
            current_position.y,
            self.goal_position.x,
            self.goal_position.y,
        )
    → delta_x, delta_y

```

4.2. Penjelasan struktur data yang digunakan dalam program bot Diamonds.

Struktur data yang digunakan dalam program bot diamonds berguna untuk menyimpan data sesuai kebutuhan algoritma. Berikut adalah struktur data yang digunakan:

1. GameObject

GameObject adalah struktur data yang menyimpan informasi mengenai semua objek yang berada di board. GameObject memiliki banyak informasi penting seperti posisi objek, tipe objek, dan nilai objek.

2. Board

Board adalah struktur data papan permainan berjalan, sekaligus tempat menyimpan informasi papan permainan, seperti semua GameObject, ukuran papan, dan lain-lain.

3. List

List adalah Struktur data yang menyimpan suatu objek dalam larik dua dimensi.

4. Zipped List

Zipped List adalah Struktur data yang menggabungkan dua list sehingga elemen pertama suatu list adalah elemen pertama list yang lain.

4.3. Analisis dari desain solusi algoritma greedy yang diimplementasikan pada setiap pengujian yang dilakukan.

Dari beberapa uji coba, Strategi Algoritma Greedy sudah hampir melakukan pilihan yang paling optimal pada setiap langkah. Ada beberapa kasus dimana kemangkusan strategi kurang optimal, seperti jika ada diamond pada jalan menuju base, bot akan lebih memilih untuk pergi ke diamond yang paling dekat, walaupun diamond tersebut menambahkan jarak dari bot ke base. Selain itu, bot tidak mengkalkulasi bot lawan dan base lawan pada pilihan langkah yang diambil, sehingga rentan terhadap bot yang ‘agresif’. Tetapi untuk mengambil diamond secara sangkil, bot sudah mendekati solusi optimal.

BAB 5

Kesimpulan dan Saran

5.1. Kesimpulan

Dalam tugas besar ini, kami menerapkan konsep algoritma greedy yang kami pelajari dari kuliah IF2211 Strategi Algoritma untuk menciptakan algoritma logic bot untuk game *Diamonds*. Proyek ini memberikan pemahaman yang lebih mendalam tentang algoritma greedy serta kemampuannya dalam menyelesaikan masalah yang memerlukan optimisasi. Kami didorong untuk berpikir secara kritis dan kreatif dalam merancang algoritma bot logika yang efektif dan efisien. Selain itu, kami juga memperoleh pemahaman yang lebih baik tentang paradigma pemrograman berorientasi objek (OOP) karena banyaknya konsep OOP yang kami terapkan dalam pembuatan kit bot ini. Proyek ini juga mendorong kami untuk mengeksplorasi berbagai algoritma lain yang dapat kami manfaatkan dalam tugas besar ini.

5.2. Saran

- Menjelajahi opsi solusi alternatif serta algoritma lain yang dapat diterapkan guna meningkatkan efektivitas serta kinerja bot.
- Menguji lebih lanjut bot dengan berbagai skenario dan kondisi permainan untuk memverifikasi kehandalan serta kesempurnaan bot dalam beragam situasi.
- Menyelidiki lebih lanjut algoritma yang diterapkan dalam bot guna mengidentifikasi peluang optimalisasi serta peningkatan performa.

Lampiran

- Link Video : <https://youtu.be/T34Ik3ZafFw>
- Link Github : https://github.com/shafiqlrv/Tubes1_PCS

Daftar Pustaka

- <https://docs.google.com/document/d/1L92Axb89ylkom0b24D350Z1QAr8rujvHof7-kXRAp7c/edit>
- [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-\(2021\)-Bag1.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag1.pdf)
- <https://www.analyticsvidhya.com/blog/2020/02/4-types-of-distance-metrics-in-machine-learning/#:~:text=Manhattan%20Distance%20is%20the%20sum,points%20across%20all%20the%20dimensions>