BAB 1

PENDAHULUAN

1.1 Latar Belakang

Meningkatnya penggunaan layanan berbasis lokasi menyebabkan jumlah dataset, yang disimpan dalam bentuk trajektori, terus-menerus bertambah dalam waktu yang singkat. Trajektori adalah urutan rekaman lokasi pada waktu tertentu dari benda yang bergerak. Tidak hanya jumlah dataset yang terus bertambah, jumlah layanan berbasis online yang menyediakan streaming data spario-temporal juga terus bertambah. Contohnya adalah AccuTracking yang membantu pemilik toko dan perusahaan pengiriman barang untuk melacak posisi kendaraan pengangkut barang secara online.

1.2 Rumusan Masalah

Bagian ini akan diisi dengan penajaman dari masalah-masalah yang sudah diidentifikasi di bagian sebelumnya.

Suspendisse vel felis. Ut lorem lorem, interdum eu, tincidunt sit amet, laoreet vitae, arcu. Aenean faucibus pede eu ante. Praesent enim elit, rutrum at, molestie non, nonummy vel, nisl. Ut lectus eros, malesuada sit amet, fermentum eu, sodales cursus, magna. Donec eu purus. Quisque vehicula, urna sed ultricies auctor, pede lorem egestas dui, et convallis elit erat sed nulla. Donec luctus. Curabitur et nunc. Aliquam dolor odio, commodo pretium, ultricies non, pharetra in, velit. Integer arcu est, nonummy in, fermentum faucibus, egestas vel, odio.

1.3 Tujuan

Akan dipaparkan secara lebih terperinci dan tersturkur apa yang menjadi tujuan pembuatan template skripsi ini

Sed commodo posuere pede. Mauris ut est. Ut quis purus. Sed ac odio. Sed vehicula hendrerit sem. Duis non odio. Morbi ut dui. Sed accumsan risus eget odio. In hac habitasse platea dictumst. Pellentesque non elit. Fusce sed justo eu urna porta tincidunt. Mauris felis odio, sollicitudin sed, volutpat a, ornare ac, erat. Morbi quis dolor. Donec pellentesque, erat ac sagittis semper, nunc dui lobortis purus, quis congue purus metus ultricies tellus. Proin et quam. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent sapien turpis, fermentum vel, eleifend faucibus, vehicula eu, lacus.

1.4 Batasan Masalah

Untuk mempermudah pembuatan template ini, tentu ada hal-hal yang harus dibatasi, misalnya saja bahwa template ini bukan berupa style LaTeX pada umumnya (dengan alasannya karena belum mampu jika diminta membuat seperti itu)

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Donec odio elit, dictum in, hendrerit sit amet, egestas sed, leo. Praesent feugiat sapien aliquet

odio. Integer vitae justo. Aliquam vestibulum fringilla lorem. Sed neque lectus, consectetuer at, consectetuer sed, eleifend ac, lectus. Nulla facilisi. Pellentesque eget lectus. Proin eu metus. Sed porttitor. In hac habitasse platea dictumst. Suspendisse eu lectus. Ut mi mi, lacinia sit amet, placerat et, mollis vitae, dui. Sed ante tellus, tristique ut, iaculis eu, malesuada ac, dui. Mauris nibh leo, facilisis non, adipiscing quis, ultrices a, dui.

1.5 Metodologi

Tentunya akan diisi dengan metodologi yang serius sehingga templatenya terkesan lebih serius.

Morbi luctus, wisi viverra faucibus pretium, nibh est placerat odio, nec commodo wisi enim eget quam. Quisque libero justo, consectetuer a, feugiat vitae, porttitor eu, libero. Suspendisse sed mauris vitae elit sollicitudin malesuada. Maecenas ultricies eros sit amet ante. Ut venenatis velit. Maecenas sed mi eget dui varius euismod. Phasellus aliquet volutpat odio. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Pellentesque sit amet pede ac sem eleifend consectetuer. Nullam elementum, urna vel imperdiet sodales, elit ipsum pharetra ligula, ac pretium ante justo a nulla. Curabitur tristique arcu eu metus. Vestibulum lectus. Proin mauris. Proin eu nunc eu urna hendrerit faucibus. Aliquam auctor, pede consequat laoreet varius, eros tellus scelerisque quam, pellentesque hendrerit ipsum dolor sed augue. Nulla nec lacus.

1.6 Sistematika Pembahasan

Rencananya Bab 2 akan berisi petunjuk penggunaan template dan dasar-dasar LaTeX. Mungkin bab 3,4,5 dapt diisi oleh ketiga jurusan, misalnya peraturan dasar skripsi atau pedoman penulisan, tentu jika berkenan. Bab 6 akan diisi dengan kesimpulan, bahwa membuat template ini ternyata sungguh menghabiskan banyak waktu.

Suspendisse vitae elit. Aliquam arcu neque, ornare in, ullamcorper quis, commodo eu, libero. Fusce sagittis erat at erat tristique mollis. Maecenas sapien libero, molestie et, lobortis in, sodales eget, dui. Morbi ultrices rutrum lorem. Nam elementum ullamcorper leo. Morbi dui. Aliquam sagittis. Nunc placerat. Pellentesque tristique sodales est. Maecenas imperdiet lacinia velit. Cras non urna. Morbi eros pede, suscipit ac, varius vel, egestas non, eros. Praesent malesuada, diam id pretium elementum, eros sem dictum tortor, vel consectetuer odio sem sed wisi.

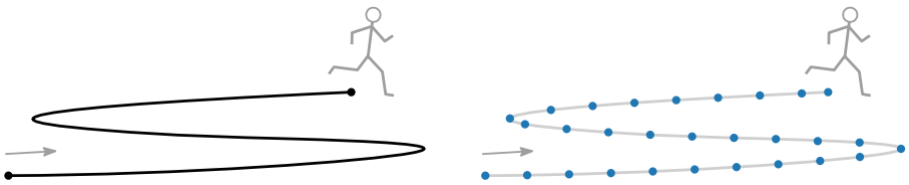
BAB 2

LANDASAN TEORI

2.1 Trajektori

Trajektori adalah lintasan yang dibuat oleh sebuah objek yang sedang bergerak pada sebuah bidang selama periode waktu tertentu. Istilah trajektori juga digunakan untuk menjelaskan data pergerakan sebuah objek yang sedang bergerak. Trajektori dapat dimodelkan dengan model data atau model abstrak.

Pada model abstrak, trajektori adalah representasi dari sebuah objek yang sedang bergerak. Objek tersebut diasumsikan sebagai sebuah titik yang bergerak tanpa terputus. Dengan model ini, posisi spasial objek tersebut bisa ditemukan kapan saja. Trajektori dapat dinyatakan sebagai sebuah fungsi yang memetakan interval waktu ke ruang tempat objek bergerak. Pada model tersebut, lintasan dari sebuah trajektori adalah gambaran dari fungsi yang memiliki bentuk dan arah namun tidak memiliki komponen waktu. Data pergerakan umumnya dikumpulkan oleh alat pelacak yang memberikan informasi spasial pada waktu tertentu ketika sampel lokasi objek diambil. Karena keterbatasan teknologi, alat pelacak biasanya melaporkan lokasi pada saat-saat tertentu yang dipisahkan oleh interval waktu yang bersifat regular atau nonregular meskipun pergerakan objek seringkali bersifat kontinu. Karenanya, pada model data, trajektori didefinisikan sebagai rangkaian lokasi, yang terurut berdasarkan waktu, tempat posisi objek bergerak dicatat. Pada model ini ketepatan geometrik dan *sampling rate* mempengaruhi kualitas data trajektori yang dihasilkan. Perbedaan antara model data dengan model abstrak ditunjukkan oleh gambar 2.1 [1].



Gambar 2.1: Ilustrasi model abstrak (kiri) dan model data (kanan) pada trajektori

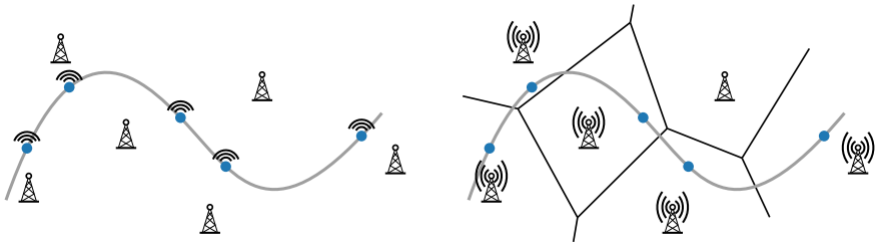
2.1.1 Ruang Gerak

Terdapat beberapa jenis ruang gerak tempat sebuah objek bergerak. Jenis ruang gerak ditentukan oleh jenis objek yang diamati dan jenis aplikasi atau program yang menggunakan data pergerakan. Jika tinggi atau kedalaman sebuah posisi dianggap tidak penting, maka ruang gerak yang digunakan adalah bidang euclidean $\mathbb{R}^2$. Jika sebaliknya, maka ruang gerak yang digunakan adalah bidang euclidean $\mathbb{R}^3$ [1].

2.1.2 Persepektif Lagrangian atau Eulerian

Terdapat dua pendekatan untuk mengekstrak data trajektori untuk model dat. Menurut persepektif langrangian, data trajektori diperoleh dengan melacak posisi objek saat objek tersebut bergerak selama durasi tertentu. Metode ini berguna untuk mendapatkan detail gerakan objek. Persepektif

lagrangian disebut juga persepektif berbasis objek. Berbeda dengan persepektif lagrangian, persepektif elulerian bersifat berbasis lokasi. Persepektif ini berfokus kepada proses observasi gerakan objek dari lokasi tertentu dengan memanfaatkan teknologi seperti *tag* RFID, jaringan WiFi, atau jaringan GSM. Perbedaan persepektif lagrangian dan eulerian ditunjukkan oleh gambar 2.2 [1].



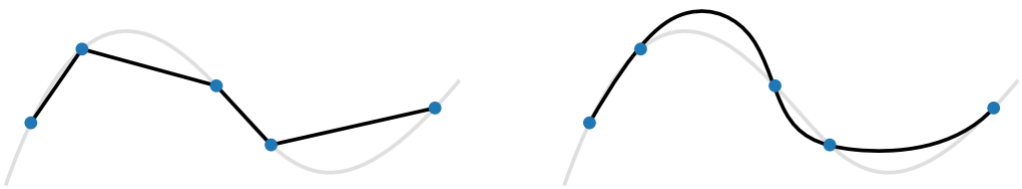
Gambar 2.2: Ilustrasi persepektif lagrangian (kiri) dan persepektif eulerian (kanan) pada pergerakan sebuah benda. Pada persepektif lagrangian, data pergerakan benda dikumpulkan selama benda berwarna biru bergerak menyusuri lintasan abu-abu. Informasi berupa rekaman lokasi-lokasi diperoleh dari alat pelacak yang terpasang pada benda. Pada persepektif eulerian, pergerakan benda dilacak dari lokasi tertentu.

2.1.3 Interpolasi Trajektori

Meskipun mengumpulkan data trajektori dengan menggunakan metode lagrangian memungkinkan pengambilan sampel trajektori dengan resolusi spasial yang bagus, manfaat utama data trajektori bergantung kepada skala yang digunakan pada proses analisis atau aplikasi yang memanfaatkan data tersebut. Jika menggunakan skala yang lebih besar, maka mungkin saja posisi objek pada selang waktu yang terletak di antara dua sampel tidak diketahui. Namun, karena pergerakan benda bersifat kontinu, data trajektori yang bersifat diskrit harus dianalisa dan diproses dalam bentuk kontinu. Terdapar beberapa metode untuk menyelesaikan masalah ini.

Metode pertama adalah mengacuhkan masalah ini dan hanya menganalisa trajektori pada waktu saat lokasi diketahui. Keunggulan metode ini adalah dapat memproses data trajektori secara ringkas dan cepat. Namun, jika data trajektori tidak di-*sampling* dengan jumlah sampel yang mencukupi, maka lokasi dan waktu yang terletak di antara dua rekaman lokasi dapat diperoleh melalui proses interpolasi. Asumsi paling sederhana untuk lokasi dari sebuah objek yang bergerak pada waktu yang terletak di antara dua titik sampel yang terletak berurutan adalah dengan melakukan interpolasi linear pada dua lokasi tersebut.

Pada interpolasi tersebut, benda yang dapat bergerak diasumsikan bergerak dengan kecepatan konstan pada garis lurus yang menghubungkan kedua lokasi. Jika data trajektori disampel dengan jumlah yang mencukupi, interpolasi ini bisa dianggap tidak memiliki *error* yang signifikan. Metode ini sangat umum digunakan pada bidang GIS(*Geographic Information System*), *computational geometry*, dan domain lainnya. Kadangkala, interpolasi linear tidak bersifat realistis pada sejumlah permasalahan. Sehingga dibutuhkan metode interpolasi yang lebih kompleks [1].



Gambar 2.3: Trajektori sungguhan sebuah objek (garis abu-abu) dan perbedaan hasil interpolasi pada data berupa urutan lokasi objek (lingkaran-lingkaran berwarna biru). Setiap lokasi memiliki atribut waktu. Pada interpolasi linear (kiri), objek tersebut diasumsikan bergerak pada lintasan lurus yang terletak di antara dua lingkaran dengan kecepatan tetap. Pada interpolasi nonlinear (kanan), lintasan yang dilalui tidak berbentuk garis lurus

2.1.4 Notasi

Misalkan T adalah trajektori sebuah entitas. Pada model abstrak, trajektori adalah sebuah fungsi yang memetakan sebuah interval waktu $I = [t_\alpha, t_\beta]$ ke ruang. Lokasi pada waktu mulai $t_\alpha(t_{start})$ adalah pangkal trajektori T dan lokasi pada waktu akhir $t_\beta(t_{end})$ adalah tujuan trajektori T.

Pada model abstrak, trajektori terdiri dari urutan lokasi yang masing-masing memiliki atribut waktu (*timestamp*) $(p_1, t_1), (p_2, t_2), \cdots, (p_r, t_r)$ dengan $p_i = (x_i, y_i)$ menyatakan posisi objek pada waktu t_i dan r menyatakan banyaknya data yang dicatat [1].

2.1.5 Data Trajektori

Data trajektori berasal dari berbagai benda bergerak, contohnya kendaraan, binatang, dan pejalan kaki. Terdapat beragam metode untuk mendapatkan data trajektori benda-benda tersebut. Kualitas data yang diperoleh dipengaruhi oleh beberapa faktor seperti jenis benda, lingkungan tempat observasi dilakukan, teknologi yang digunakan, dan lainnya.

Data trajektori dapat diperoleh secara manual atau dengan memanfaatkan teknologi tertentu. Keunggulan metode manual adalah tidak membutuhkan sumber tenaga eksternal. Namun, metode ini memiliki beberapa kelemahan yaitu rendahnya kepresisian dari lokasi-lokasi yang diperoleh dan banyaknya data trajektori yang dapat diperoleh terbatas. Metode berbasis teknologi digunakan untuk memperoleh data trajektori dengan presisi yang tinggi dalam skala besar untuk durasi waktu lebih lama. Contohnya adalah teknologi *VHF Radio Telemetry, argos-doppler system*, dan GPS. Secara umum terdapat dua jenis alat yang digunakan yaitu alat pemancar(*transmitter*) atau alat penerima (*receiver*). Alat tersebut dipasang pada objek yang diobservasi trajektorinya. Pada lingkungan dengan ukuran terbatas (misalnya, didalam bangunan), alat yang digunakan adalah alat yang dikhususkan untuk lingkungan tersebut seperti sensor, bluetooth, sensor jangkauan 3D statis, dan lainnya.

Kualitas data trajektori yang bisa diperoleh dengan menggunakan alat-alat tersebut bergantung pada beberapa faktor seperti cuaca, kekuatan baterai, dan kekuatan sinyal. Sehingga data trajektori yang diperoleh, contohnya pada dua lokasi yang berurutan, mungkin saja tidak memiliki interval waktu yang seragam. Masalah yang hampir serupa juga muncul pada trajektori yang diperoleh dari kumpulan benda bergerak; interval waktu yang tercatat mungkin berbeda-beda.

Kadangkala alat pelacak sangat sulit untuk digunakan di beberapa lingkungan. Untuk mengatasi masalah ini, pergerakan objek bisa direkam menggunakan kamera video. Lalu data trajektori diekstrak dari rekaman video menggunakan algoritma tertentu. Contoh implementasi metode ini adalah trajektori ikan di lautan atau trajektori pejalan kaki di dalam maupun di luar ruangan.

Jika data trajektori sulit diperoleh dengan menggunakan salah satu dari metode-metode sebelumnya, maka pergerakan dari objek bergerak dapat dimodelkan menggunakan metode tertentu. Trajektori diperoleh dari hasil simulasi program komputer yang mengimplementasikan model yang dibuat sebelumnya. Data yang dihasilkan disebut data trajektori artifisial. Meskipun proses memo-

delkan gerakan benda-benda cukup rumit, Metode ini memiliki keunggulan dibandingkan dengan metode sebelumnya yaitu parameter-parameter penting simulasi dapat diatur contohnya: durasi simulasi, ukuran dan jenis lingkungan, jumlah benda pada lingkungan, dan atribut-atribut benda bergerak (kecepatan,perilaku) [1].

2.2 Analisis Trajektori

Terdapat sejumlah metode yang baru-baru ini dikembangkan untuk menganalisis data trajektori. Metode-metode tersebut adalah segmentasi, *similarity determination*, *clustering*, representasi, dan berbagai jenis *pattern* yang mungkin muncul dari gerakan benda-benda.

2.2.1 Similaritas

Analisa similaritas bertujuan untuk menentukan apakah dua trajektori terlihat serupa atau tidak. Kemiripan dua trajektori ditentukan oleh berbagai jenis definisi seperti mengunjungi lokasi yang sama, atau memiliki perubahan kecepatan yang sama (misalnya pelan diawal, cepat diakhir). Analisa similaritas juga digunakan untuk *preprocessing* atau sebagai *subroutine* untuk metode analisa lain.

Similaritas pada dua buah trajektori kadang ditentukan oleh jarak diantara dua trajektori tersebut. Terdapat beberapa metode pengukuran jarak untuk menentukan similaritas trajektori. Jika hanya lintasan trajektori yang dianggap penting, maka metode yang digunakan adalah *euclidean distance, hausdorff distance*, atau *frechet distance*. Metode pengukuran lain mengolah aspek temporal trajektori. Contohnya adalah *Dynamic Time Warping, time-focused distance, edit distance*, dan *longest common subsequence*. Atribut tambahan pada gerakan seperti kecepatan, percepatan, dan arah gerakan bisa digunakan untuk menentukan similaritas.

2.2.2 Clustering

Clustering adalah proses mempartisi sebuah objek menjadi beberapa *cluster*. Objek-objek yang berada pada *cluster* yang sama memiliki sifat yang mirip tetapi berbeda dengan objek anggota *cluster* lain. *Clustering* dapat diterapkan pada trajektori yang berasal dari beberapa objek. Trajektori dapat dibagi ke dalam beberapa *cluster* berdasarkan similaritas antar trajektori atau subtrajektori. *Clustering* pada trajektori dapat digunakan untuk mendeteksi *movement pattern* tertentu.

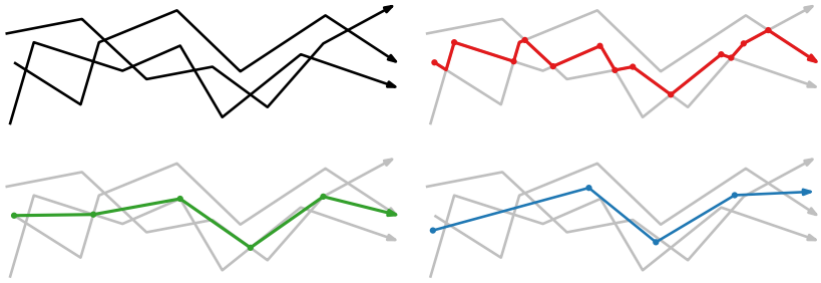
2.2.3 Representasi

Sebuah *cluster* yang berisi trajektori dapat direpresentasikan oleh sebuah trajektori yang dipilih menggunakan metode tertentu. Trajektori tersebut dapat merepresentasikan beberapa hal seperti rute yang ditempuh oleh sebuah benda, atau pergerakan benda tertentu. Gambar 2.4 mengilustrasikan beberapa contoh trajektori representatif yang mungkin dari sebuah *cluster*.

Trajektori representatif memiliki beberapa manfaat. Pertama, mengurangi jumlah data trajektori yang perlu dianalisis karena trajektori tersebut sudah merpresentasikan trajektori lain yang serupa. Kedua, trajektori representatif memiliki visualisasi yang lebih baik karena lebih berfokus ke satu atau sedikit trajektori. Ketiga, untuk menemukan *outlier* dengan cara menganalisis similaritas atau *closeness* antara trajektori representatif dengan trajektori lainnya.

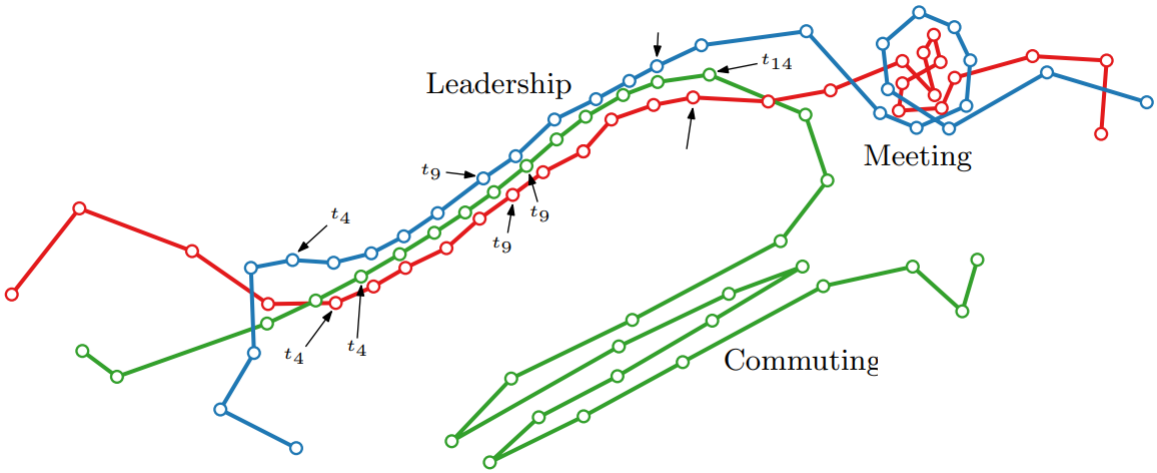
2.2.4 Movement Patterns

Movement pattern sebuah objek adalah atribut-atribut yang diperoleh dari trajektori objek tersebut. Pada kumpulan objek yang bergerak, *movement pattern* mengacu pada interaksi atau relasi antar objek. Karena setiap jenis benda bergerak memiliki tipe relasi dan atribut yang berbeda-beda, *pattern-pattern* yang nampak pada benda-benda tersebut juga berbeda-beda. Contohnya, pola gerakan pengunjung di taman bisa saja mengandung informasi tempat-tempat menarik yang sering dikunjungi.



Gambar 2.4: Contoh trajektori yang merepresentasikan sebuah kumpulan trajektori yang berwarna hitam (kiri atas). Terdapat tiga trajektori representatif yaitu trajektori berwarna merah yang menggunakan bagian dari kumpulan trajektori (kanan atas), trajektori berwarna hijau yang menggunakan verteks pada himpunan trajektori(kiri bawah), dan membentuk sebuah trajektori baru (trajektori warna biru, kanan bawah)

Pada trajektori tunggal, pola gerakan yang umum ditemui adalah *periodic pattern, commmuting*, dan *concentration pattern*. Sedangkan pada sepasang trajektori, pola gerakan yang kerap kali ditemui adalah *chasing behavior* dan *avoidance movement*. Pada kumpulan trajektori, pola gerakannya adalah *leadership pattern, meeting pattern, convergence pattern*, dan lainnya. Contoh pattern tersebut ditunjukkan oleh gambar 2.5.



Gambar 2.5: Tiga trajektori dengan tiga jenis *pattern* yang ditunjukkan pada gambar. Pada selang waktu antara t_4 hingga t_{14}, benda berwarna hijau menjadi pemimpin (*leader*) dan diikuti oleh dua benda lainnya, yaitu biru dan merah.

2.3 Collective Motion

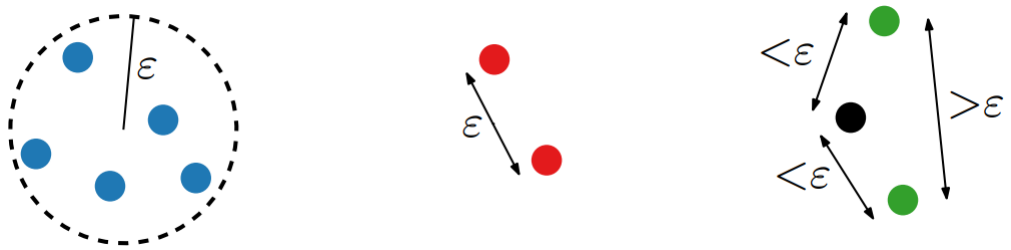
Collective motion adalah kemunculan gerakan teratur yang terjadi secara spontan pada sistem yang terdiri dari beberapa *self-proplelled-agent* [2]. Fenomena tersebut dapat ditemukan pada perilaku makhluk hidup contohnya pada kawanan ikan, burung, atau kerumunan orang yang merupakan hasil interaksi lokal antara individu-individu pada proses swaorganisasi (self-organization). Swaorganisasi adalah proses untuk mencapai tujuan tertentu dengan cara mengatur perilaku antar individu secara mandiri [3].

Collective motion adalah salah satu jenis *pattern* yang sudah dipelajari dalam beberapa metode. Istilah *collective movement* kadang disebut *group*. *Collective motion* memiliki hubungan dengan *clustering*. Namun *clustering* berbeda dengan *collective motion*. Pada *clustering*, seluruh trajektori

dilibatkan pada proses pembuatan *cluster*. Pada *collective motion*, sebuah benda bisa menjadi anggota *group* yang berbeda pada waktu yang berbeda atau pada waktu yang sama. Terdapat beberapa aspek yang harus diperhatikan pada pemodelan *collective movement*, yaitu:

1. *Input trajectories*
   Data gerakan (*movement data*) dikumpulkan dalam bentuk trajektori diskrit (urutan lokasi-lokasi yang memiliki atribut waktu) dan data tersebut dapat diubah ke bentuk kontinu dengan menggunakan interpolasi. Jika model *collective motion* hanya memproses data dalam bentuk diskrit, maka waktu mulai dan waktu akhir sebuah *group* harus sesuai dengan catatan waktu yang direkam pada data trajektori. Jika menggunakan data dalam bentuk kontinu, maka waktu dan posisi benda-benda di antara lokasi-lokasi yang diketahui dapat diestimasi. Karena itu, *group* yang sama mungkin memiliki interval waktu yang lebih panjang karena perbedaan pada waktu awal dan / atau waktu akhir.

2. *Spatial proximity*
   Umumnya, benda-benda yang bergerak bersama-sama memiliki posisi yang saling berdekatan. Definisi *closeness* yang paling sederhana adalah sebuah variabel $\epsilon > 0$ yang menyatakan jarak maksimum di antara dua benda. Definisi tersebut lalu dikembangkan untuk mengakomodasi *collective movement*. Pada *collective motion*, setiap pasang benda pada *group* harus terletak sedekat mungkin pada setiap waktu. Terdapat juga definisi *closeness* lain. Menurut Gudmundsson dan van Kreveld [4], sebuah *group* direpresentasikan oleh sebuah lingkaran dengan jari-jari ϵ dan objek-objek anggota *group* harus terletak di dalam lingkaran tersebut. Pada metode lainnya, terdapat sebuah objek lain yang menjadi perantara, dua objek i dan j disebut saling berdekatan meskipun $d_{ij}(t) > \epsilon$ selama terdapat objek lain yaitu k sehingga $d_{ik}(t) \leq \epsilon$ dan $d_{kj}(t) \leq \epsilon$.



Gambar 2.6: Jenis-jenis *spatial proximity*. Pada gambar di sebelah kiri, semua objek terletak di dalam sebuah lingkaran dengan jari-jari ϵ. Pada gambar di tengah, jarak dua benda harus tidak lebih dari ϵ. Pada gambar di sebelah kanan, dua objek dianggap berdekatan dengan menggunakan objek berwarna hitam sebagai perantara.

3. *Size*
   Dua objek yang yang bergerak bersama-sama dapat dianggap sebagai sebuah *group*. Batasan ukuran sebuah *group* dipengaruhi oleh aplikasi yang digunakan. Selain itu, sebuah objek dapat menjadi anggota satu *group* atau beberapa *group*.

4. *Temporal Component*
   Temporal component waktu minimal yang dibutuhkan pada kumpulan benda-benda untuk tetap bersama hingga dapat dianggap sebagai *group*. hal paling penting pada durasi sebuah *group* adalah kontinuitas durasi *group* tersebut.

Terdapat bermacam-macam model dari *collective movement* dengan definisi yang sedikit berbeda-beda. Salah satu definisi yang digunakan adalah *flock*. Istilah lain yang berhubungan erat dengan konsep *flock* adalah *micro-cluster, moving cluster, mobile group, herd, convoy, swarm, traveling*

companion, gathering, platoon, dan *group*. Seluruh definisi tersebut bergantung pada paling sedikit tiga parameter yaitu: ukuran(*size*), parameter temporal, dan parameter spasial. Mayoritas definisi yang disebutkan menggunakan trajektori yang dimodelkan secara diskrit, namun ada juga yang menggunakan trajektori yang dimodelkan secara kontinu, contohnya *group*.

2.4 *Online Algorithm*

Pada komputasi online, sebuah algoritma harus menghasilkan keluaran berupa rangkaian keputusan yang memiliki dampak terhadap kualitas akhir dari keseluruhan kinerja algoritma tersebut. Setiap keputusan harus dibuat berdasarkan kejadian-kejadian di masa lalu tanpa informasi yang pasti tentang masa depan. Algoritma yang memiliki ciri-ciri demikian disebut algoritma online (*online algorithms*). Algoritma online adalah topik yang menarik di berbagai bidang ilmu. Banyak masalah komputasi yang bersifat online secara intrinsik sehingga membutuhkan keputusan yang dibuat sesegera mungkin.

Pendekatan tradisional untuk mempelajari algoritma *online* adalah bagian dari *distributional complexity framework*. Pada *framework* tersebut, hipotesis tentang distribusi kejadian dibuat dan *total cost* atau *expected cost* setiap kejadian dipelajari. [5]

Algoritma *online* adalah jenis algoritma yang menerima rangkaian masukan satu per satu sesuai urutan tibanya dan menghasilkan keluaran untuk setiap masukan sesegera mungkin. Setiap pasangan input dan keluaran yang dihasilkan memiliki *cost* tertentu. Algoritma *online* digunakan pada situasi-situasi yang membutuhkan keputusan instan meskipun data masukan yang dimiliki tidak lengkap.

algoritma *online* memiliki sejumlah perbedaan dengan algoritma *offline*. Pada algoritma *offline*, seluruh masukan yang akan dikerjakan sudah tersedia dari awal. Algoritma *offline* harus melakukan aksi tertentu untuk setiap masukan yang diterima namun, pilihan aksi yang dapat dipilih dapat didasarkan pada seluruh masukan yang ada. Algoritma *offline* dianggap mengetahui masa depan sementara algoritma *online* tidak. Pada banyak keadaan, ketidaktahuan tentang masa depan sangat tidak menguntungkan. Karenanya, algoritma *online* seringkali memiliki kinerja yang lebih buruk dibandingkan dengan algoritma *offline* optimal pada masalah yang sama.

Algoritma *offline* disebut optimal jika algoritma tersebut memilih aksi-aksi dengan *cost* minimal untuk setiap rangkaian masukan yang diterima. Mendefinisikan ukuran performa untuk algoritma *online* lebih sulit daripada algoritma *offline* karena biasanya apapun aksi yang dipilih oleh algoritma untuk merespons masukan saat ini, terdapar masukan lainnya yang seolah-olah membuat algoritma terlihat buruk. Karena sulitnya menentukan definisi performa rasional atau optimal sebuah algoritma *online*, algoritma tersebut kerapkali diabaikan dalam berbagai bidang.

Metode mengevaluasi algoritma *online* yang paling umum adalah memodelkan sumber masukan secara stokastik. Pada model yang dibuat, sebuah algoritma *online* dapat dianggap optimal jika aksi-aksi yang dipilih memiliki *cost* yang minimal. Nilai *cost* yang dihasilkan bergantung pada rangkaian masukan yang dihasilkan oleh model yang dibuat dan aksi yang dipilih oleh algoritma untuk setiap masukan yang diterima. Namun, model stokastik [6]

2.4.1 *Competitive Analysis*

Competitive analysis adalah subbidang pada studi tentang algoritma (*study of algorithms*), subbidang lainnya adalah *classical computational complexity*. *Classical computational complexity* menghitung sumber daya apa saja yang digunakan oleh sebuah algoritma saat melakukan komputasi tertentu. *Competitive analysis* menentukan apakah sebuah algoritma lebih unggul dari algoritma lain dalam menyelesaikan masalah tertentu. Studi kompleksitas algoritma membedakan kualitas algoritma-algoritma berdasarkan sumber daya komputasional yang digunakan dan kualitas solusi yang dihasilkan. Bagaimanapun, fokus utama pada algoritma-algoritma yang berkerja pada keadaan yang tidak pasti bukanlah kompleksitas komputasi tetapi *competitive analysis*.

Competitive analysis berguna pada analisis sistem-sistem yang memiliki konsep perkembangan waktu, memiliki *environment* tertentu, merespons perubahan pada *environment* dengan aksi tertentu, dan memiliki *memory state*. Lebih jelasnya, sistem yang memiliki konfigurasi yang berubah dari waktu ke waktu dan bergantung pada konfigurasi tersebut untuk merespons perubahan yang mungkin terjadi pada *environment*. Banyak masalah yang dapat diilustrasikan menggunakan definisi tersebut. Baik pada masalah yang memiliki syarat ketepatan waktu maupun tidak.

Competitive analysis biasanya digunakan pada algoritma *online* yang harus merespons kejadian-kejadian yang terjadi dari waktu ke waktu. Namun dapat juga diugnakan pada konteks lain selain algoritma *online*. *Competitive analysis* digunakan untuk menyelesaikan masalah yang melibatkan pengambilan keputusan meskipun informasi yang dimiliki tidak lengkap. Kondisi ini mungkin disebabkan karena beberapa kejadian belum terjadi (contohnyapada pasar saham, harga saham besok siang tidak dapat diketahui sekarang). Kejadian-kejadian tersebut belum terjadi karena tindakan algoritma dibutuhkan untuk memperoleh informasi yang belum diperoleh atau karena algoritma bersifat *distributed*, yaitu algoritma yang melakukan proses komputasi pada beberapa komputer yang terhubung lewat jaringan, dan tidak ada komputer yang memiliki *global information*. Meskipun demikian, banyak laporan ilmiah yang menggunakan *competitive analysis* berususan dengan *online problems*. Begitu banyaknya hingga penggunaan *competitive analysis* seringkali dianggap sama dengan algoritma *online* [7].

Competitive analysis adalah metode yang digunakan untuk mengukur kualitas sebuah algoritma *online*. Kualitas sebuah algoritma *online* ditentukan oleh nilai *competitive ratio*. Nilai *competitive ratio* dihitung dengan rumus:

$$\frac{ALG(\sigma)}{OPT(\sigma)} \tag{2.1}$$

Pada rumus 2.1, $ALG(\sigma)$ adalah biaya yang ciperlukan oleh algoritma *online* ALG untuk menghasilkan sebuah keluaran untuk setiap σ. σ adalah kejadian atau input yang mungkin terjadi di masa depan. Sedangkan $OPT(\sigma)$ adalah biaya (*cost*) terkecil yang mungkin untuk menghasilkan keluaran yang sama.

Contoh kasus yang mengilustrasikan masalah umum pada *online decision-making* adalah *rent-or-buy problem*. Pada masalah tersebut misalkan seseorang mencoba bermain ski. Terdapat dua pilihan yaitu beli atau sewa papan ski. Jika membeli papan ski baru, maka biaya yang dibutuhkan adalah lima ratus dollar. Sedangkan jika menyewa papan ski untuk sekali pakai, biaya yang dibutuhkan adalah lima puluh dollar. Karena orang tersebut tidak tahu kalau dia akan menyukai ski maka orang itu menyewa sebuah papan ski. Orang tersebut lalu menyewa ski secara terus-menerus hingga menyadari seharusnya dia membeli papan ski dari awal. Sehingga strategi optimalnya: jika seseorang akan bermain ski lebih dari sepuluh kali, maka solusi terbaik adalah membeli papan ski dari awal. Jika orang itu hanya bermain ski 9 kali atau kurang, maka solusi terbaik adalah menyewa papan ski.

Algoritma 'langsung membeli papan ski', sebut saja algoritma A, memiliki *worst case* yaitu saat seseorang hanya bermain ski satu kali. Sehingga nilai *competitive ratio* nya adalah $500/50 = 10$. Sedangkan algoritma 'terus-menerus menyewa papan ski' memiliki *competitive ratio* yang tidak terbatas (terus bertambah). Algoritma lainnya, sebut saja algoritma B, adalah tetap menyewa papan ski selama beberapa kali lalu beli papan ski. Jika biaya sewa dalah r dan biaya beli adalah p, maka sewa papan ski sebanyak $\lceil p/r \rceil - 1$ kali lalu beli papan ski.

Misalkan terdapat dua kasus. Pada kasus pertama, hanya bermain ski sebanyak $\lceil p/r \rceil$ kali atau kurang dan $p = 500, r = 50$. *competitive ratio* algoritma ini adalah nol karena baik algoritma ini maupun solusi optimalnya adalah tidak membeli ataupun tidak menyewa. Pada kasus kedua, jika seseorang bermain ski $\geq \lceil p/r \rceil$ kali, maka solusi optimalnya adalah membeli papan ski ($OPT = p$). Total biaya yang dibayarkan ($ALG(\sigma)$) adalah $r(\lceil p/r \rceil - 1) + p$ (450+500 dollar = 950 dollar). Sehingga *competitive rationya* adalah $950/500 = 1.95$.

Sebuah algoritma *online* disebut kompetitif jika *competitive ratio* algoritma tersebut memiliki

batasan tertentu (*bounded*)[8]. Algortma ALG disebut sebagai *asymptotic c-approximation algorithm* jika terdapat sebuah konstanta $\alpha \geq 0$ sehingga untuk seluruh input yang mungkin:

$$ALG(\sigma) - c.OPT(\sigma) \leq \alpha. \tag{2.2}$$

Jika $\alpha = 0$, maka algoritma ALG disebut *c-approximation algorithm*. Sebuah algoritma *online* ALG disebut $c - competitive$ jika terdapat sebuah konstanta α sehingga memenuhi kondisi:

$$ALG(\sigma) \leq c.OPT(\sigma) + \alpha. \tag{2.3}$$

Konstanta α disebut sebagai *additive constant*. Jika nilai α lebih kecil atau sama dengan nol, maka algoritma ALG disebut *strictly c-competitive*. Jika *alpha* bernilai positif, maka untuk masalah yang bersifat *online* secara intrinsik, terdapat sebuah input yang sangat panjang dengan biaya pengerjaan (*cost*) yang tak terbatas. Konstanta α dianggap tidak signifikan pada input-input yang lebih banyak. Selain itu, pada rangkaian input yang bersifat *finite*, penggunaan konstanta tambahan (α) memungkinkan penggunaan rasio performansi intrinsik yang tidak bergantung pada kondisi awal.

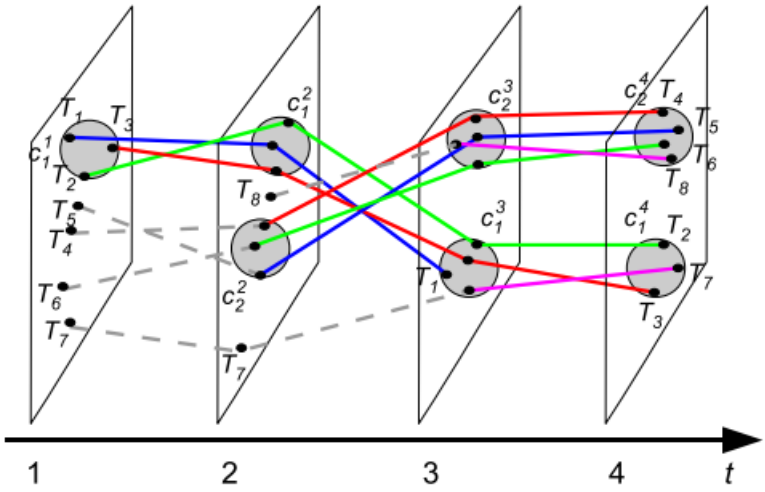
Sebuah algoritma *online* ALG yang bersifat *strictly c-competitive* adalah *c-approximation algorithm* jika ALG melakukan komputasi secara *online* dan untuk setiap input σ, sebuah algoritma yang bersifat *c-competitive* memiliki *cost*, untuk setiap keluaran, yang berada di rentang c-kali *cost* algoritma optimal ($\leq \alpha$). *Competitive ratio*(c) memiliki nilai minimum yaitu 1. Nilai *competitive ratio* yang kecil menandakan bahwa algoritma ALG memiliki performa yang lebih baik dari algoritma OPT. Algoritma yang memiliki *competitive ratio* c disebut *c-competitive*. Sebuah algoritma disebut *competitive* jika algoritma tersebut memiliki *competitive ratio* yang nilainya konstan. Meskipun nilai c bisa jadi adalah fungsi dari parameter masalah, c harus bersifat independen dari input yang diberikan. Contohnya pada masalah penjadwalan yang terdiri dari N mesin, nilai *competitive ratio* dipengaruhi oleh N, namun nilai c tidak dipengaruhi oleh jumlah dan jenis *job* yang dikerjakan. Nilai minimum dari nilai-nilai c untuk setiap input yang diterima disebut sebagai *competitive ratio* algoritma ALG.

Competitive analysis adalah bagian dari *worst-case complexity framework*. *Competitive analysis* sangat dibutuhkan pada situasi-situasi yang membutuhkan jaminan kinerja, contonhya adalah perencanaan keuangan. [5]

2.5 Flock Pattern Problem

Flock Pattern Problem adalah permasalahan mengidentifikasi seluruh kumpulan trajektori yang terletak berdekatan selama periode waktu tertentu. Kumpulan tersebut disebut *flock*. Setiap pasangan elemen pada *flock* tidak boleh memiliki jarak lebih besar dari *treshold* tertentu selama waktu hidup *flock* tersebut. *Flock* bisa digambarkan sebagai sebuah lingkaran dengan diameter tertentu yang mencakup seluruh anggota *flock* selama periode waktu tertentu [9]. Secara umum, terdapat dua jenis *flock* yaitu *flock* dengan jumlah anggota yang tetap (*fixed-flock*) atau *flock* dengan jumlah anggota yang berubah-ubah (*varying-flock*) [4].

Flock Pattern Problem didefinisikan sebagai berikut: misalkan T adalah himpunan yang berisi kumpulan trajektori, μ adalah banyaknya trajektori minimal pada himpunan T; $\mu > 1, \mu \in \mathbb{N}$. ϵ adalah *treshold* jarak antar elemen pada satuan tertentu; $\epsilon > 0, \epsilon \in \mathbb{R}^+$, δ adalah banyaknya *time instances* minimal; $\delta > 1, \delta \in \mathbb{N}$. Sebuah *flock pattern* (μ, ϵ, δ) terdiri dari sebuah himpunan F yang berisi seluruh *flock* f_k yang merupakan himpunan dengan ukuran maksimal yang memiliki minimal μ buah trajektori dan minimal δ *timestamp* yang berturut-turut $t_j, \ldots, t_(j + \delta - 1)$. Setiap *timestamp* mewakili sebuah cakram (*disk*) dengan diameter ϵ dan berpusat di $c_k^{t_i}$ yang mencakup seluruh trajektori dalam *flock* f_k pada waktu $t_i(f_k^{t_i}), j \leq i \leq j + \delta - 1$ [9]. Ilustrasi flock pattern ditunjukkan oleh 2.7.



Gambar 2.7: Ilustrasi *flock pattern* menurut [9]

2.6 Basic Flock Evaluation Algorithm

Misalkan trajektori T_{id} merepresentasikan pergerakan sebuah objek dengan id tertentu (O_{id}) yang bergerak pada suatu bidang. Trajektori T_{id} terdiri dari n buah titik $p(t_i)$.

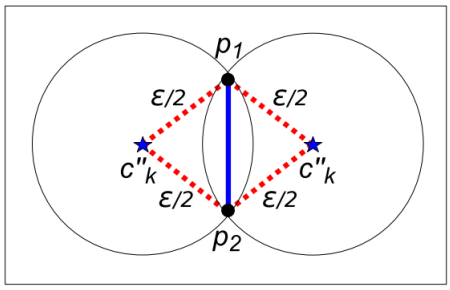
$$T_{id} = \{p(t_1), p(t_2), \cdots, p(t_n)\} \tag{2.4}$$

Titik $p(t_i)$ adalah lokasi objek O_{id} pada ruang dua dimensi $\mathbb{R}^2$ pada waktu t_i. Titik-titik tersebut terurut berdasarkan *timestamp* $t_i (t_i \in N, t_{i-1} < t_i, 0 < i \leq n)$. Sedangkan L_p menyatakan jarak di antara dua titik $p_a^{t_i}$ dan $p_b^{t_i}$ pada waktu t_i yang dihitung oleh fungsi $d(p_a^{t_i}, p_b^{t_i})$ menggunakan metrik tertentu.

Salah satu masalah utama pada algoritma *basic flock evaluation* adalah titik pusat sebuah *flock* pada *flock pattern* tidak selalu terletak pada trajektori. Karena jumlah titik lokasi trajektori (p_{id}) terlalu banyak maka tidak mungkin untuk memeriksa seluruh titik tersebut secara satu per satu. Maka sebuah teorema digunakan untuk membatasi ukuran ruang pencarian.

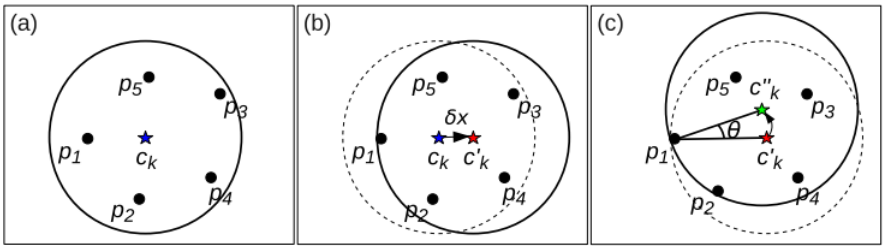
Teorema 1: Jika untuk setiap waktu t_i terdapat sebuah titik $c_k^{t_i}$ pada ruang sedemikian rupa sehingga $\forall T_j \in f, d(p_j^{t_i}, c_k^{t_i}) \leq \epsilon/2$, maka terdapat titik lain ($c'^{t_i}_k$) yang memenuhi $\forall T_j \in f, d(p_j^{t_i}, c'^{t_i}_k) \leq \epsilon/2$ dan terdapat trajektori $T_a \in f$ dan $T_b \in f$ sehingga $\forall T_j \in \{T_a, T_b\}, d(p_j^{t_i}, c'^{t_i}_k) = \epsilon/2$.

Teorema tersebut menjelaskan jika terdapat sebuah cakram $c_k^{t_i}$ dengan diameter ϵ yang mencakup seluruh trajektori pada *flock* f pada waktu t_i, maka terdapat cakram lain dengan diameter yang sama namun memiliki titik pusat yang berbeda yaitu $c'^{t_i}_k$ yang juga mencakup seluruh trajektori pada cakram pertama (gambar 2.9). Pada garis keliling (*circumference*) kedua cakram tersebut terdapat minimal dua titik lokasi trajektori.



Gambar 2.8: Cakram yang menyinggung titik p_1 dan p_2 pada garis kelilingnya dengan $d(p_1, p_2) \leq \epsilon$

Misalkan terdapat sebuah cakram (atau lingkaran) dengan diameter ϵ dan berpusat di c_k yang mencakup seluruh trajektori pada flock pada waktu t_i (gambar 2.9(a)). Asumsikan tidak ada trajektori yang terletak pada garis batas cakram sehingga $\forall T_j \in f, d(T_j, c_k) < \epsilon/2$. Dari cakram ini dapat ditemukan cakram lain yang memeiliki sifat yang sama namun memiliki titik pusat yang berbeda. Cakram tersebut dibentuk dengan melakukan translasi dan rotasi pada cakram yang berpusat di c_k.



Gambar 2.9: Pembuktian teorema 1

Langkah pertama adalah titik c_k digeser searah sumbu x hingga trajektori yang terletak di paling kiri *flock* berada di garis keliling lingkaran (gambar 2.9(b)). Pada gambar 2.9(b), titik (lokasi tertentu pada trajektori) pertama yang bersinggungan dengan garis keliling cakram setelah titik pusat cakram digeser adalah p_1. Titik pusat cakram kini berpindah ke c'_k dan seluruh titik pada *flock* berada dalam radius cakram baru yang berpusat di c'_k.

Langkah kedua adalah cakram yang baru dibentuk diputar dengan menggunakan titik p_1 sebagai titik pusat putaran. Cakram tersebut diputar hingga titik lain (p_2) bersinggungan dengan garis kelilingnya (gambar 2.9(c)). Sehingga terbentuk cakram baru yang mencakup seluruh titik pada *flock* yang berpusat di c''_k dengan diameter ϵ. Cakram yang baru terbentuk memiliki minimal dua titik pada garis kelilingnya.

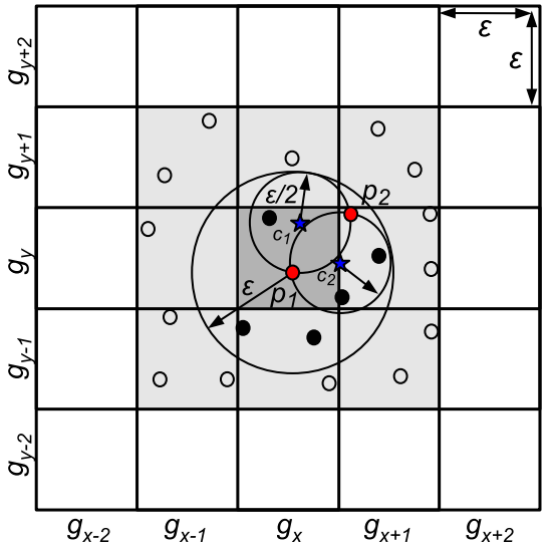
Teorema ini membatasi ukuran ruang pencarian lokasi *flock* pada domain spasial. Pada himpunan yang berisi $|T|$ buah trajektori terdapat $|T|^2$ kemungkinan pasangan kombinasi titik pada waktu tertentu. Setiap pasangan titik memiliki tepat dua cakram dengan radius $\epsilon/2$ dan kedua titik tersebut terletak pada garis keliling kedua cakram (gambar 2.8). Setiap cakram minimal memiliki μ buah trajektori. Untuk setiap *time instance* pada interval waktu δ terdapat $2|T|^2$ *flock pattern* yang mungkin. Banyaknya flock pattern yang mungkin adalah $2|T|^{2\delta}$. Sehingga *flock pattern problem* pada *fixed time duration* memiliki kompleksitas waktu polinomial yaitu $O(|T|^{|\delta|})$ [10].

2.6.1 *Reporting Flock Pattern*

Untuk menghitung flock disks secara efisien, digunakanlah struktur berbasis grid(*grid-based structure*). Struktur berbasis grid yang digunakan terdiri dari kumpulan *grid cell* dengan panjang sisi ϵ. Setiap titik lokasi pada trajektori T_{id} pada waktu t_i, yaitu $(p_{id}^{t_i})$ terletak pada *grid cell* tertentu. Pilihan *grid cell* yang digunakan ditentukan oleh *latitude* dan *longitude* $p_{id}^{t_i}$. Setiap titik lokasi hanya dimasukkan ke satu *grid cell*. Jumlah total *grid cell* pada *grid index* dipengaruhi oleh distribusi

trajektori pada waktu tertentu (t_i) dan nilai ϵ. Semakin kecil nilai ϵ, semakin besar jumlah *grid cell* yang dibutuhkan.

Menurut implementasi yang diusulkan oleh Vieira .et al [10], *grid cell* yang kosong tidak digunakan. Sebuah struktur data digunakan untuk menyimpan trajektori pada *grid cell*. Jika nilai ϵ tidak terlalu besar, maka jumlah titik pada setiap *grid cell* relatif sedikit sehingga struktur data sederhana seperti *list* dapat digunakan. Ilustrasi *grid-based index* ditunjukkan oleh gambar 2.10.



Gambar 2.10: Ilustrasi *grid-based index*. Setiap *grid cell* pada indeks memiliki sisi sepanjang ϵ dan berisi kumpulan titik lokasi ($p_{id}^{t_i}$). *grid cell* yang diarsir abu gelap adalah *grid cell* yang sedang diproses ($g_{x,y}$). Sedangkan *grid cell* yang diarsir abu terang adalah *grid cell* yang bertetangga dengan $g_{x,y}$. Titik-titik berwarna hitam adalah titik-titik yang terletak pada radius ϵ dari titik p_1 (baris 8 algoritma 1). Titik p_1 adalah titik pada $g_{x,y}$ yang saat ini sedang diproses (baris 7 algoritma 1). Titik-titik yang berwarna merah adalah pasangan titik yang jaraknya $\leq \epsilon$ (baris 11 algoritma 1). Bintang berwarna biru menyatakan titik pusat dua lingkaran yang menyinggung titik p_1 dan p_2. Lingkaran c_1 dan c_2 adalah kandidat *disk* pada *flock* untuk waktu t_i

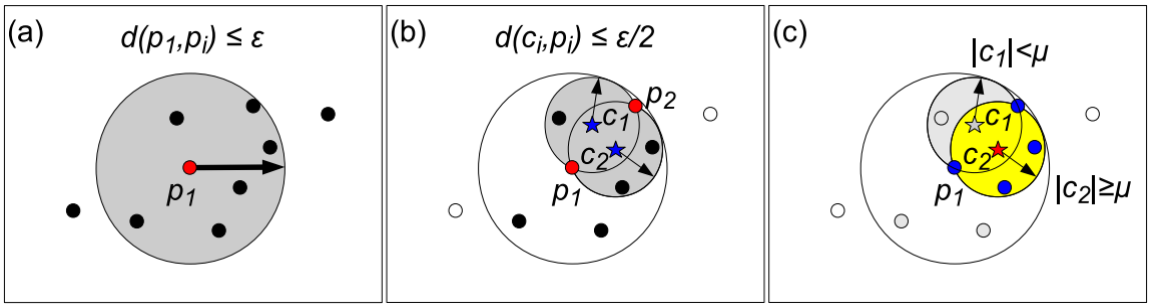
Setelah struktur *grid index* untuk waktu t_i selesai dihitung, Algoritma 1 dapat digunakan untuk memproses cakram-cakram yang dihasilkan. Untuk setiap *grid cell* $g_{x,y}$, hanya sembilan *gridcell* yang bertetangga dengan $g_{x,y}$ dan $g_{x,y}$ itu sendiri yang harus diproses. Algoritma 1 memproses setiap titik lokasi pada *grid cell* $g_{x,y}$ maupun titik lokasi pada *grid cell* pada rentang $[g_{x-1,y-1} \cdots g_{x+1,y+1}]$ untuk menemukan pasangan titik p_r dan p_s dengan jarak diantara kedua titik tersebut tidak lebih dari ϵ ($d(p_r, p_s) \leq \epsilon$). Karena seluruh grid cell pada indeks memiliki ukuran ϵ, grid cell yang terletak di luar rentang $[g_{x-1,y-1} \cdots g_{x+1,y+1}]$ tidak perlu diperiksa. Pasangan titik yang belum diproses saat ini dan memiliki jarak tidak lebih dari ϵ satu sama lain akan digunakan untuk menghitung dua cakram c_1 dan c_2. Jika jarak antara pasangan titik p_r dan p_s tepat sama dengan ϵ, maka cakram c_1 dan c_2 memiliki pusat yang sama sehingga hanya salah satu cakram yang perlu diproses.

Tidak semua titik pada rentang $[g_{x-1,y-1} \cdots g_{x+1,y+1}]$ dapat dipasangkan dengan titik-titik pada $g_{x,y}$. Hanya pasangan titik yang memiliki jarak $d(p_r, p_s) \leq \epsilon$. Langkah berikutnya adalah memeriksa posisi titik-titik hasil *range query* jikalau titik-titik tersebut berada di dalam cakram yang dihitung pada tahap sebelumnya. Untuk setiap titik p_r pada *grid cell* $g_{x,y}$, sebuah *range query* dengan radius ϵ dilakukan pada seluruh *grid* $[g_{x-1,y-1} \cdots g_{x+1,y+1}]$ untuk menemukan titik-titik yang bisa dipasangkan dengan titik p_r sehingga $d(p_r, p_s) \leq \epsilon$ terpenuhi (gambar 2.11(a)). Hasil dari *range query* disimpan pada *list* $\mathcal{H}$ yang digunakan untuk memeriksa cakram yang dihitung. Untuk setiap pasangan titik yang valid, terdapat dua cakram yang dihasilkan. Untuk setiap cakram yang dihasilkan, titik-titik pada list $\mathcal{H}$ diperiksa jikalau mereka terletak di dalam cakram (gambar 2.11(b)). Cakram yang jumlah titiknya kurang dari μ dibuang. Sedangkan cakram yang jumlah

Algorithm 1 Menghitung *disk* pada *grid-based index*

```
1: C ← ∅
2: Index.Build(T[t_i], ε)
3: for setiap grid cell yang tidak kosong g_{x,y} ∈ Index do
4:     P_r ← g_{x,y}
5:     P_s ← [g_{x−1,y−1} ··· g_{x+1,y+1}]
6:     if |P_s| ≥ μ then
7:         for each p_r ∈ P_r do
8:             H ← Range(p_r, ε)
9:             for each p_j ∈ H do
10:                if not computed{p_r, p_j} yet then
11:                    compute disks {c_1, c_2} defined by {p_r, p_j} and diameter ε
12:                    for each disk c_k ∈ {c_1, c_2} do
13:                        c ← c_k ∩ H
14:                        if |c| ≥ μ then
15:                            C.Add(c)
16: return C
```

titiknya $\geq \mu$ disimpan. Pada gambar 2.11(c), cakram c_1 dibuang dan cakram c_2 dianggap valid.



Gambar 2.11: Proses menemukan *flock* pada waktu t.

Jika terdapat dua cakram valid dengan elemen yang hampir sama, maka cakram dengan jumlah titik terbanyak yang akan dipilih. Algoritma 1 menerapkan hal ini dengan cara membandingkan titik pusat cakram dan jumlah total elemen yang sama pada setiap cakram. Sebuah cakram c_1 diperiksa dengan cakram lain c_2 jika jarak keduanya tidak lebih dari ϵ ($d(c_1, c_2) \leq \epsilon$). Jika sebaliknya, maka dua cakram tidak memiliki elemen yang sama. Agar operasi tersebut berjalan dengan seefisien mungkin, titik tengah dan jari-jari $\epsilon/2$ seriap cakram disimpan pada struktur data *k-d-tree*. Untuk setiap cakram c_1 algoritma akan membandingkannya dengan elemen-elemen pada *k-d-tree* untuk mencari cakram yang bersinggungan dengan c_1. Karena titik-titik milik sebuah cakram disimpan dalam *binary tree*, operasi mencari *subset* atau *superset* dapat dilakukan dengan efisien. Karena itu, elemen yang sama pada dua cakram dapat dicari dengan memeriksa (*scan*) setiap elemen pada kedua cakram tepat satu kali. Jika kardinalitas dari elemen yang sama pada cakram c_1 dan c_2 sama dengan kardinalitas elemen c_1 ($|c_1 \cap c_2| = c_1$), maka $c_1 \subset c_2$ sehingga cakram c_1 dapat dibuang. Jika $|c_1 \cap c_2| = |c_2|$, maka cakram c_2 dapat dibuang. Jika $|c_1 \cap c_2| \neq |c_2|$ dan $|c_1 \cap c_2| \neq |c_1|$, maka cakram c_1 dan c_2 disimpan ke dalam $\mathcal{C}$.

Pada algoritma *basic flock pattern evaluation*(BFE), sebuah kandidat *disk* dihitung untuk setiap *time instance* t_i, dimulai dari *time instance* pertama (t_1) dan terus berlanjut sampai *time instance* terakhir. Setiap kandidat *disk* yang dihasilkan untuk setiap *time instance* t_i dianalisa dan digabungkan(*join*) dengan kandidat *flock* yang dihasilkan pada waktu sebelumnya (t_{i-1}). Hanya *flock* yang sukses digabungkan dengan *disk* saat ini yang dipertahankan. Algoritma BFE

mengembalikan sebuah *flock pattern* yang memenuhi syarat temporal δ, yaitu setiap *flock* memiliki δ buah *disk*.

disk yang dihasilkan untuk *time instance* pertama oleh *grid index* dianggap sebagai *flock* parsial yang disimpan pada *list*. *list* tersebut berisi kandidat *flock* pada waktu saat ini ($\mathcal{F}^{t_i}$). Pada *time instance* selanjutnya, *disk-disk* yang dihasilkan oleh *grid-based index* disimpan dalam *list* kandidat *flock* $\mathcal{F}^{t_i}$ dan digabungkan dengan kandidat *flock* pada waktu sebelumnya ($\mathcal{F}^{t_{i-1}}; \mathcal{F}^{t_i} = \mathcal{F}^{t_i} \cap \mathcal{F}^{t_{i-1}}$). $\mathcal{F}^{t_i}$ dapat digabungkan dengan $\mathcal{F}^{t_{i-1}}$ jika jumlah total elemen yang sama di antara kandidat *flock* dan *disk* tidak kurang dari μ ($|c \cap f| \geq \mu$).

Jika kondisi tersebut terpenuhi, maka hasil penggabungan ditambahkan ke dalam *list* kandidat *flock* untuk waktu t_i. Sebuah kandidat *flock* dianggap valid jika terdapat minimal δ buah operasi penggabungan(*join*). $\mathcal{F}^{t_i}$ hanya menyimpan *flock* yang dimulai pada waktu sebelumnya ($t_{start} > t_i - \delta$) dan berakhir pada waktu saat ini ($t_{end} = t_i$). *disk-disk* yang tidak dapat digabungkan dibuang.

Keunggulan algoritma BFE adalah untuk setiap *time instance* yang sedang diproses, $\mathcal{F}^{t_i}$ hanya menyimpan id trajektori. Selain itu, lokasi trajektori untuk setiap *time instance* hanya diproses sekali sehingga data trajektori pada *time window* yang panjangnya δ tidak perlu di-*buffer* [10].

Algorithm 2 *Basic Flock Evaluation*

```
1: F^{t_0} ← ∅                                          ▷ Inisialisasikan flock parsial
2: for each time instance t_i do
3:     L ← T[t_i]                                       ▷ lokasi-lokasi yang terletak pada trajektori pada waktu t_is
4:     C ← Index.Disks() ▷ Hitung kandidat disk untuk waktu t_i dengan menggunakan algoritma 1
5:     F^{t_i} ← ∅                                      ▷ Untuk menyimpan kandidat flock yang potensial
6:     for each c ∈ C do
7:         for each f ∈ F^{t_{i-1}} do                  ▷ flock sebelumnya yang berpotensi
8:             if |c ∩ f| ≥ μ then
9:                 u ← c ∩ f
10:                u.t_start ← f.t_start                ▷ tetapkan waktu mulai
11:                u.t_end ← t                          ▷ tetapkan waktu akhir
12:                if u.t_end − u.t_start = δ then      ▷ jika flock ditemukan
13:                    report flock pattern u from u.t_start to u.t_end
14:                    update u.t_start
15:                F^{t_i} ← F^{t_i} ∪ u                ▷ tambahkan flock u
16:            F^{t_i} ← F^{t_i} ∪ c                    ▷ tambahkan disk c kedalam F^{t_i}
```

2.6.2 Filter flock disks

Jumlah kandidat *disk* untuk *time instance* tertentu bisa saja terlalu banyak sehingga *cost* untuk melakukan *join* kandidat *disk* ke *flock pattern* terlalu tinggi. Terdapat empat heuristik untuk mengatasi masalah ini yaitu:

1. *Top Down Evaluation* (TDE)
   Berbeda dengan algoritma BFE yang menyusun *flock* secara *bottom-up* yaitu menyusun *flock* secara satu per satu mulai dari *time instance* pertama, TDE menyusun *flock* secara *top-down*. Metode *top-down* membandingkan kandidat *disk* pada dua *time instance* yang berjarak δ *time instance* satu sama lain. Metode ini didasarkan pada asumsi bahwa dua kandidat *disk* pada dua *time instance* yang berurutan memiliki perbedaan yang sedikit sedangkan dua kandidat *disk* yang berjarak δ *time instance* memiliki perbedaan yang signifikan. Perbedaan antara dua kandidat *disk* pada dua *time instance* yang berurutan menghasilkan *flock* pendek dalam

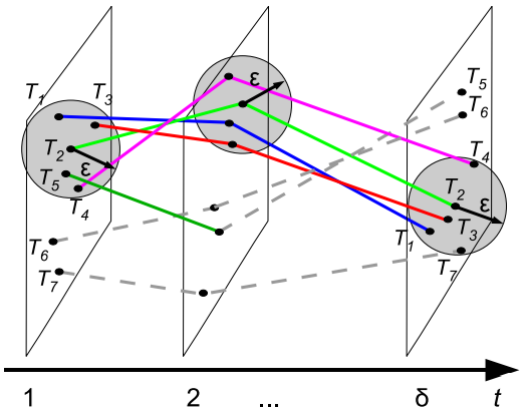
jumlah besar sedangkan perbedaan antara dua kandidat *disk* yang berjarak δ *time instance* menghasilkan kandidat *flock* dalam jumlah kecil.

Heuristik ini mem-*buffer* lokasi trajektori untuk *time window* w yang memiliki panjang δ *time instance*. Untuk menggabungkan *disk* dengan kandidat *flock*, pertama-tama kandidat *disk* $\mathcal{C}^1$ untuk *time instance* pertama $t_{i-\delta+1}$ pada *window* w dihitung. Lalu, *disk* untuk *time instance* terakhir t_i pada w dihitung dan digabungkan dengan *disk* pada $\mathcal{C}^1$. Kandidat *flock* untuk *time window* w yang dihasilkan kemudian diverifikasi dengan menggunakan algoritma BFE.

2. *Pipe Filter Evaluation* (PFE)
Heuristik kedua menggunakan paradigma *filter and refine*. Heuristik akan memilih trajektori-trajektori yang memiliki minimal μ buah entitas yang berada dalam radius ϵ selama minimal δ satuan waktu. Lalu *flock pattern* dicari menggunakan algoritma BFE. Pada heuristik PFE, sebuah *grid-based index* untuk *time instance* $t_{i-\delta}$ pada *time window* w dihitung. Lalu, operasi *range search* dilakukan pada setiap trajektori T_j pada waktu $t_{i-\delta}$. Tujuan kueri tersebut adalah memeriksa jumlah lokasi objek lain yang berada pada radius ϵ dari trajektori yang sedang diproses.

Jika kardinalitas dari hasil operasi *range search* lebih besar atau sama dengan μ, maka operasi serupa dilakukan pada *time instance* $t_{i-\delta}$ hingga t_i. Jika total jumlah trajektori di dalam "pipa" yang terbentuk untuk setiap trajektori T_j adalah $|\mathcal{U}| \geq \mu$, maka trajektori tersebut ditambahkan kedalam *list* kandidat $\mathcal{M}$ untuk diproses lebih lanjut pada tahap *refinement*.

Tahap *refinement* menggunakan algoritma BFE dan hanya memproses trajektori hasil tahap *filter* ($\mathcal{M}$). Berbeda dengan algoritma BFE yang memproses seluruh trajektori yang tersimpan. Metode ini digunakan jika *cost* untuk menghitung kandidat *disk* adalah *computationally expensive* dan konstruksi *flock* dilakukan pada *subset* trajektori. Gambar 2.12 mengilustrasikan *pipe* yang terbentuk untuk trajektori T_2 pada waktu δ dalam radius ϵ.



Gambar 2.12: *pipe filtering* untuk trajektori T_2 pada waktu δ dalam radius ϵ

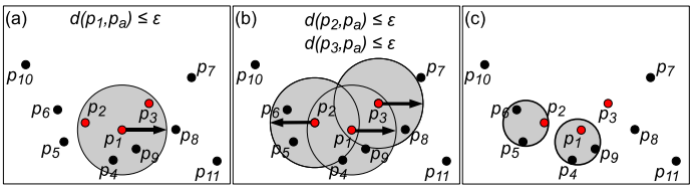
3. *Continuous Refinement Evaluation* (CRE)
Heuristik ini terus-menerus memurnikan kumpulan trajektori yang dapat menjadi bagian sebuah *flock pattern* (*continuous refinement*). Metode ini menggunakan tahap *disk generation* untuk *time instance* t_i sebagai tahap *filtering* untuk *time instance* t_{i+1}. Hanya trajektori yang berada dalam kandidat *disk* pada waktu t_i yang akan dianalisis pada waktu t_{i+1}. Metode ini digunakan pada situasi dengan selektivitas kandidat *disk* tinggi. Contohnya pada kandidat *disk* dalam jumlah kecil dan jumlah trajektori pada kandidat *disk* juga tidak banyak.

Tahap pertama pada *continuous refinement* adalah mencari semua *disk* $\mathcal{C}^1$ menggunakan lokasi-lokasi $\mathcal{L}[1]$ untuk *time instance* $t_{i-\delta}$. Lalu, untuk setiap cakram $c^1, c^1 \in \mathcal{C}^1$, trajektori yang berada di dalamnya diproses lebih lanjut dari waktu $t_{i-\delta+1}$ hingga t_i.

Pada *time instance* pertama, *disk* $\mathcal{C}^1$ untuk *time instance* $t_{i-\delta}$ disimpan dalam $\mathcal{F}^1$. Lalu, setiap objek c_1 diproses lebih lanjut untuk menghitung *disk* lalu di-*merge-join* dengan *disk* milik *time instance*sebelumnya yang disimpan di $\mathcal{F}^t$. Jika $\mathcal{F}^t$ tidak memiliki kandidat *flock* yang potensial pada waktu t, maka pemrosesan c^1 dapat dihentikan. Setelah langkah ini, *flock pattern* untuk waktu $t_{i-\delta}$ hingga t_i selesai dihitung.

4. *Cluster Filtering Evaluation* (CFE)
   Terdiri dari dua tahap utama. Pertama-tama algoritma *DBSCAN Clustering* dengan parameter $eps = \epsilon$ dan $minPts = \mu$ untuk setiap *time instance* t_i. *Cluster* yang ditemukan untuk setiap *time instance* lalu digabungkan dengan *cluster* yang ditemukan pada t_{i-1}. Dua *cluster* dapat digabungkan jika memiliki minimal μ buah trajektori yang sama. Jika *cluster* u dapat disusun dengan cara yang disebutkan untuk δ buah *time instance* yang berturut-turut, maka u dianggap sebagai kandidat *disk* yang valid. Kemudian kandidat *disk* tersebut diproses menggunakan algoritma BFE.

   Gambar 2.13 mengilustrasikan langkah-langkah pada algoritma CFE. Pada gambar 2.13(a) operasi DBSCAN dilakukan pada objek lokasi (titik) tertentu (p_1) dengan parameter $eps = \epsilon$ dan $minPts = \mu$. Lalu, pada gambar 2.13(b), operasi DBSCAN dilakukan pada titik-titik yang bertetangga dengan p_1. Titik yang tidak berada dalam *cluster*. *Cluster* yang terbentuk oleh operasi DBSCAN (gambar 2.13(c)) diproses lebih lanjut pada tahap *refinement*.



Gambar 2.13: Ilustrasi proses pembentukan *cluster* pada algoritma CFE

Kesimpulannya, algoritma BFE memiliki tiga tahapan utama yaitu mencari kandidat-kandidat *disk* dengan bantuan *grid-based index*, memilih kandidat *disk* dengan jumlah trajektori yang maksimal, dan menggabungkan *flock* pada waktu saat ini dengan *flock* pada waktu sebelumnya [9].

2.7 Plane Sweeping

Plane sweeping adalah algoritma yang digunakan untuk mendeteksi perpotongan pada ruas-ruas garis pada bidang tertentu. Algoritma ini berperan penting dalam mengurangi kompleksitas komputasi pada berbagai jenis permasalahan di bidang komputasi geometri. Komponen utama algoritma ini adalah sebuah *sweeping line* yang bergerak dari kiri ke kanan pada sumbu x. Garis tersebut terus bergerak hingga terdapat kondisi yang terpenuhi, contohnya terdapat garis yang memotong sebuah titik. Jika hal tersebut terjadi operasi geometri dilakukan pada titik yang memicu kondisi tersebut. Operasi-operasi tersebut dilakukan pada titik-titik yang terletak tidak jauh dari *sweeping line*. Proses ini terus dilakukan hingga seluruh titik pada dataset telah dilalui oleh *sweeping line*.

Masalah ini dapat diselesaikan secara *brute force* dengan kompleksitas $O(N^2)$. Jika diselesaikan dengan metode *plane sweeping*, maka kompleksitasnya berkurang menjadi $O(N * Log(N))$. Misalkan terdapat $N-1$ titik yang telah diproses oleh algoritma dan jarak terpendek diantara dua titik yang berurutan adalah h. Algoritma akan memproses titik terakhir (P_n) dan berusaha mencari titik lain yang berjarak lebih kecil dari h dari titik P_n. Titik-titik yang sudah diproses oleh algoritma dan terletak pada radius h dari titik p_n disimpan pada sebuah himpunan. Setiap titik baru yang sudah diproses ditambahkan ke set tersebut dan set dikosongkan saat algoritma berpindah ke titik lain atau ketika nilai h mengalami perubahan. Titik titik dalam set diurutkan berdasarkan koordinat sumbu y. Set tersebut diimplementasikan dalam bentuk *balanced binary tree*. Algoritma hanya

mencatat titik-titik yang berada pada rentang $p_n.y - h$ hingga $p_n.y + h$. Untuk menemukan titik yang berjarak lebih kecil dari h dari titik P_n, algoritma akan memeriksa titik-titik yang terdapat pada *binary tree*. Proses tersebut memiliki kompleksitas sebesar $O(log(N))$ untuk setiap N titik sehinnga total kompleksitas algoritma *plane sweeping* adalah $O(N.log(N))$ [9]..

2.8 Spatial Data Types

Secara umum, terdapat tiga jenis data spasial yang digunakan untuk merepresentasikan suatu objek geografi yaitu *point* (titik), *line*, dan *region*. Setiap jenis data tersebut dibagi lagi ke dalam dua kelas yaitu *simple* dan *complex* [11]. Ilustrasi *simple point, line*, dan *region* ditunjukkan oleh gambar 2.14. Ilustrasi *complex point, line*, dan *region* ditunjukkan oleh gambar 2.15.

1. *simple point*
   adalah sebuah titik yang terletak pada ruang dua dimensi.

2. *simple line*
   adalah sebuah objek satu dimensi pada ruang dua dimensi yang memiliki dua ujung.

3. *simple region*
   adalah kurva tertutup sederhana yang memisahkan sebuah bidang menjadi dua bagian yaitu bagian dalam dan bagian luar.

4. *complex point*
   kumpulan simple point yang tidak saling tumpang tindih.

5. *complex line*
   adalah kumpulan garis sederhana yang saling lepas.

6. *complex region*
   Adalah kumpulan *face* dan *hole*. *Face* adalah region sederhana yang memiliki beberapa *hole*. *Hole* memenuhi kriteria *simple region* tetapi bagian dalam sebuah *complex region* berada di bagian luar *hole* dan bagian luarnya berada di bagian dalam *hole*. Batas–batas *face* dan *hole* bisa saling bersentuhan pada titik-titik diskrit yang terbatas.

Gambar 2.14: Ilustrasi *simple point* (kiri), *simple line* (tengah), dan *simple region* (kanan)

Gambar 2.15: Ilustrasi *complex point* (kiri), *complex line* (tengah), dan *complex region* (kanan)

DAFTAR REFERENSI

[1] Wiratma, L. (2019) Computations and Measures of Collective Movement Patterns Based on Trajectory Data. Disertasi. Utrecht University, The Netherlands.

[2] Palacci, J., Sacanna, S., Steinberg, A. P., Pine, D. J., dan Chaikin, P. M. (2013) Living crystals of light-activated coloidal surfers. *Science*, **339**, 936–940.

[3] Willshaw, D. (2006) Self-organization in the nervous system. Bagian dari Morris, R., Tarassenko, L., dan Kenward, M. (ed.), *Cognitive Systems - Information Processing Meets Brain Science*. Academic Press, London.

[4] Gudmundsson, J. dan van Kreveld, M. (2006) Computing longest duration flocks in trajectory data. *Proceedings of the 14th Annual ACM International Symposium on Advances in Geographic Information Systems*, Arlington, USA, 10-11 November, pp. 35—-42. ACM, New York.

[5] Borodin, A. dan El-Yaniv, R. (1998) *Online Computation and Competitive Analysis*, 1st edition. Cambridge University Press, Cambridge.

[6] Karp, R. M. (1992) On-line algorithms versus off-line algorthms: How much is it worth to know the future? Technical Report TR-92-044. International Computer Science Institute Berkeley, United States of America.

[7] Fiat, A. dan Woeginger, G. J. (1998) *Online Algorithms: The State of the Art*, 1st edition. Springer-Verlag, Berlin.

[8] of Computer Science, C. M. U. S. (2013) Online algorithms. Catatan kuliah CMU 15-451 di Carnegie Mellon University School of Computer Science. https://www.cs.cmu.edu/~avrim/451f13/lectures/lect1107.pdf. 8 Maret 2021.

[9] Tanaka, P. S., Vieira, M. R., dan Kaster, D. S. (2016) An improved base algorithm for online discovery of flock patterns in trajectories. *Journal of Information and Data Management*, **7**, 52–67.

[10] Vieira, M., Bakalov, P., dan Tsotras, V. (2009) On-line discovery of flock patterns in spatio-temporal data. *Proceedings of the ACM International Symposium on Advances in Geographic Information Systems*, Seattle, USA, 4-6 November, pp. 286–295. ACM, New York.

[11] Edwardsville, S. I. U. (2015) Introduction to spatial indexing. Catatan kuliah CS490 Advanced Databases di Southern Illinois University Edwardsville. http://www.cs.siue.edu/~marmcke/docs/cs490/_static/AdvancedDatabases.pdf. 20 Februari 2021.

LAMPIRAN A

KODE PROGRAM

Listing A.1: MyCode.c

```c

// This does not make algorithmic sense,
// but it shows off significant programming characters.

#include<stdio.h>

void myFunction( int input, float* output ) {
    switch ( array[i] ) {
        case 1: // This is silly code
            if ( a >= 0 || b <= 3 && c != x )
                *output += 0.005 + 20050;
            char = 'g';
            b = 2^n + ~right_size - leftSize * MAX_SIZE;
            c = (--aaa + &daa) / (bbb++ - ccc % 2 );
            strcpy(a,"hello $@?");
    }
    count = ~mask | 0x00FF00AA;
}

// Fonts for Displaying Program Code in LATEX
// Adrian P. Robson, nepsweb.co.uk
// 8 October 2012
// http://nepsweb.co.uk/docs/progfonts.pdf
```

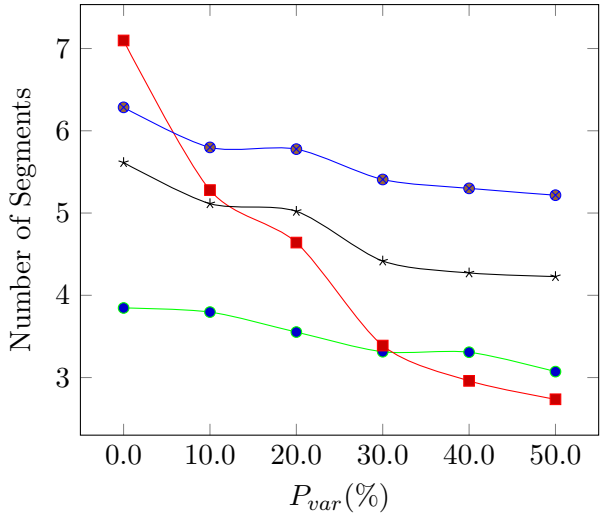
Listing A.2: MyCode.java

```java
import java.util.ArrayList;
import java.util.Collections;
import java.util.HashSet;

//class for set of vertices close to furthest edge
public class MyFurSet {
    protected int id;                                   //id of the set
    protected MyEdge FurthestEdge;                      //the furthest edge
    protected HashSet<MyVertex> set;                    //set of vertices close to furthest edge
    protected ArrayList<ArrayList<Integer>> ordered;    //list of all vertices in the set for each trajectory
    protected ArrayList<Integer> closeID;               //store the ID of all vertices
    protected ArrayList<Double> closeDist;              //store the distance of all vertices
    protected int totaltrj;                             //total trajectories in the set

    /*
     * Constructor
     * @param id : id of the set
     * @param totaltrj : total number of trajectories in the set
     * @param FurthestEdge : the furthest edge
     */
    public MyFurSet(int id,int totaltrj,MyEdge FurthestEdge) {
        this.id = id;
        this.totaltrj = totaltrj;
        this.FurthestEdge = FurthestEdge;
        set = new HashSet<MyVertex>();
        ordered = new ArrayList<ArrayList<Integer>>();
        for (int i=0;i<totaltrj;i++) ordered.add(new ArrayList<Integer>());
        closeID = new ArrayList<Integer>(totaltrj);
        closeDist = new ArrayList<Double>(totaltrj);
        for (int i = 0;i <totaltrj;i++) {
            closeID.add(-1);
            closeDist.add(Double.MAX_VALUE);
        }
    }

}
```

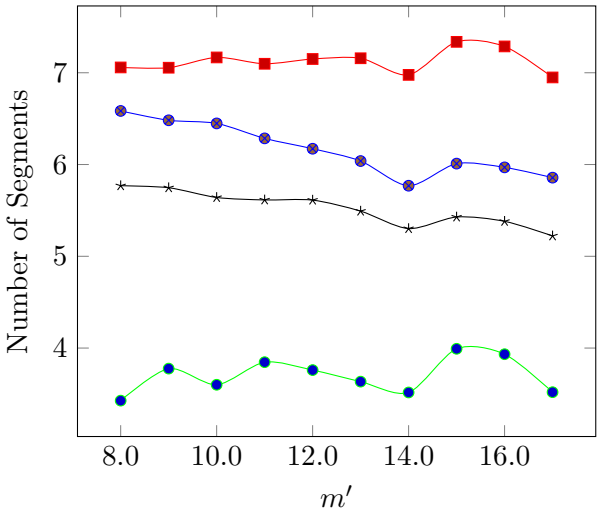
LAMPIRAN B

HASIL EKSPERIMEN

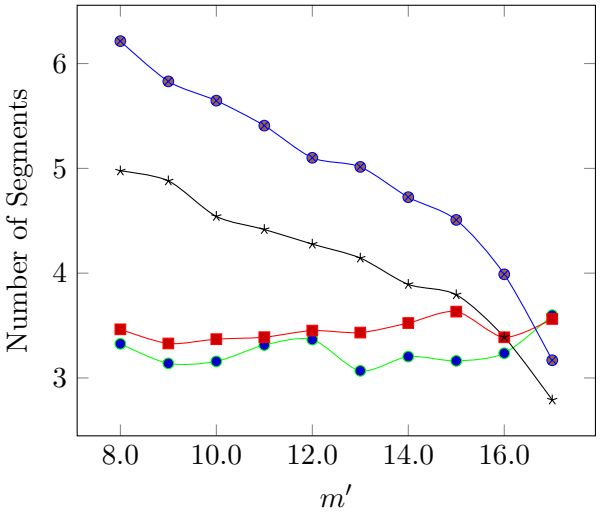
Hasil eksperimen berikut dibuat dengan menggunakan TIKZPICTURE (bukan hasil excel yg diubah ke file bitmap). Sangat berguna jika ingin menampilkan tabel (yang kuantitasnya sangat banyak) yang datanya dihasilkan dari program komputer.



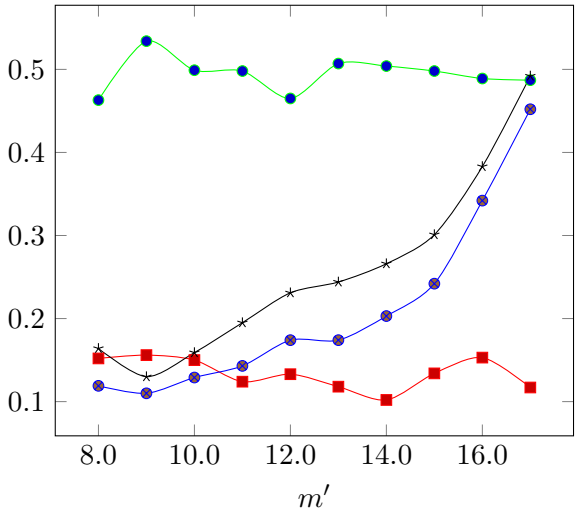
Gambar B.1: Hasil 1



Gambar B.2: Hasil 2



Gambar B.3: Hasil 3



Gambar B.4: Hasil 4