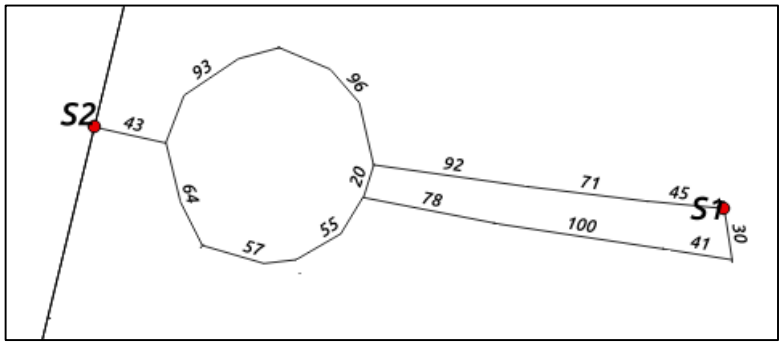
BAB 6 ANALISIS DAN EVALUASI ALGORITME

6.1 Analisis Algoritme

Pada tahap ini, dilakukan analisis mengenai algoritme yang diusulkan dan elemen pendukung dari algoritme tersebut. Analisis yang dilakukan adalah dengan melakukan percobaan terhadap algoritme tersebut. Mulai dari menerapkan algoritme tersebut dan dijalankan dengan menggunakan data yang telah terlebih dahulu di proses pada tahapan sebelumnya. Pada proses analisis akan diilustrasikan dalam bentuk gambar dengan penjelasan. Hasil dari analisis tersebut akan dijadikan bahan dalam melakukan evaluasi pada sub-bab berikutnya.

6.1.1 Algoritme Dijkstra

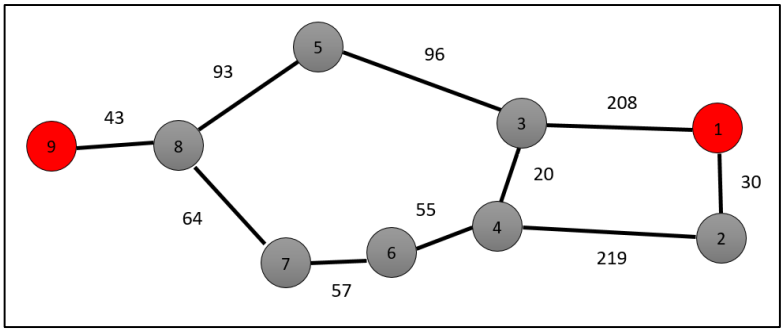
Pada bagian ini membahas alur kerja dari proses algoritme Dijkstra. Algoritme Dijkstra digunakan dalam melakukan pencarian *split nodes* didalam poligon Voronoi yang sama. Pencarian menggunakan algoritme Dijkstra dilakukan dengan membandingkan nilai atau bobot dari titik awal pencarian menuju ke titik tujuan (titik tujuan yang dimaksud bisa berupa titik tujuan pencarian atau pencarian dari titik awal menuju tepi poligon Voronoi), dalam analisis yang dilakukan jarak tempuh merupakan nilai atau bobot yang digunakan. Titik pencarian digambarkan dengan titik berwarna merah. Titik *S1* merupakan titik awal dan titik *S2* merupakan titik tujuan. Pada Gambar 6.1. merupakan gambaran dari keadaan sebenarnya dari jalan yang menjadi bahan percobaan. Pencarian pada Gambar 6.1. dilakukan dari titik *S1* yang berada pada bagian dalam poligon Voronoi menuju ke tepi poligon Voronoi pada titik *S2.*



Gambar 6.1 Contoh dari Titik Pencarian dengan Algoritme Dijkstra

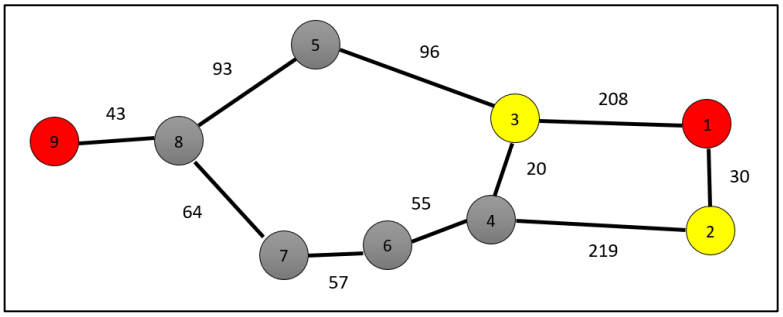
Untuk mempermudah dalam melakukan analisis algoritme Dijkstra dibuat ilustrasi yang digambarkan menggunakan *node* (simpul) dan *edge* (tepian) seperti pada Gambar 6.2. Pembuatan *node* dan *edge* berdasarkan peta sesungguhnya namun untuk jalur yang memiliki arah yang sama dilakukan penggabungan nilai jarak tempuh agar mempermudah dalam memahami ilustrasi dari algoritme Dijkstra. Titik awal adalah *node 1* dan titik tujuan berada pada *node 9.* Warna

merah yang digambarkan pada ilustrasi merupakan *node* yang dituju atau *node* yang dipilih untuk dilalui.

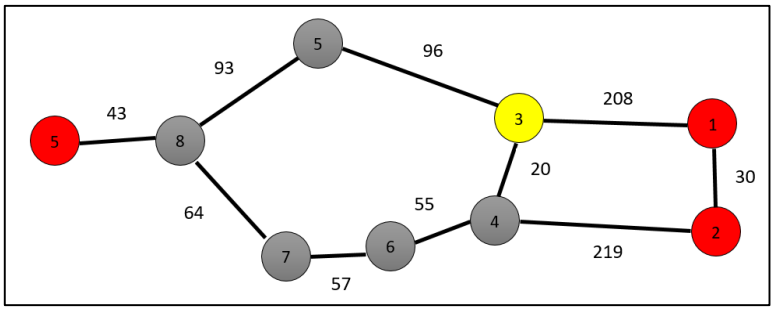


Gambar 6.2 Inisialisasi *Node* Awal dan *Node* Akhir

Pencarian dimulai dengan memperhitungkan *node* tetangga yang terhubung langsung dengan *node* awal (*node 1*), dan membandingkan jarak tempuh dari *node* tetangga tersebut. *Node* yang merupakan tetangga dari *node 1* adalah *node 2* dan *node 3* seperti pada Gambar 6.3. Kemudian kedua *node* tersebut diberi warna kuning yang memiliki arti *node* tersebut merupakan *node* yang mungkin akan dipilih untuk dilalui karena bersinggungan langsung dengan *node* pencarian. Dari perhitungan yang dilakukan jarak tempuh dari *node 1 – node 2* memiliki jarak 30 meter, sedangkan jarak tempuh dari *node 1 – node 3* memiliki jarak 208 meter. Dengan melakukan perbandingan didapatkan hasil berupa *node 2* yang memiliki nilai jarak tempuh terpendek sehingga menjadi *node* berikutnya yang dilalui dan berubah warna menjadi merah seperti pada Gambar 6.4.

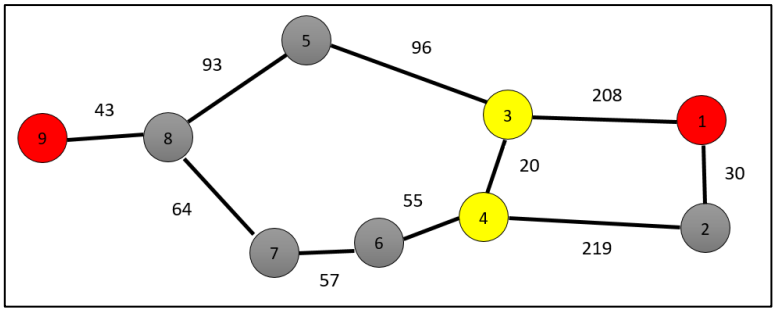


Gambar 6.3 Menentukan Kandidat *Node* Selanjutnya antara *Node 2* dan *Node* 3

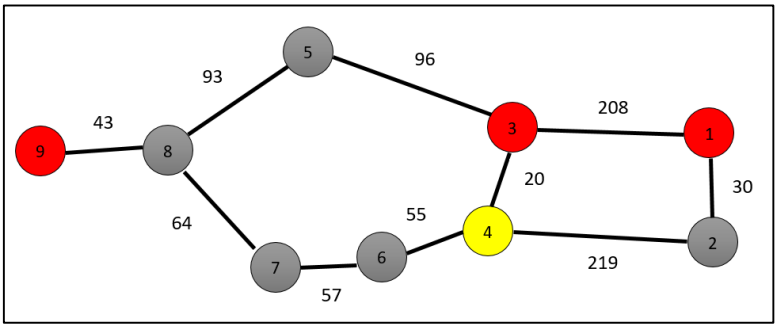


Gambar 6.4 Menentukan Node yang Dilalui yaitu *Node 2*

Pencarian dilanjutkan dengan melakukan perhitungan untuk mencari *node* berikutnya yang dilalui dari *node 2*. *Node 2* hanya memiliki *node* terdekat yaitu *node 4* dengan jarak tempuh 219 meter. Kemudian dilakukan perhitungan jarak yang ditempuh dari *node 1 – node 2 – node 4* yang memiliki jarak tempuh sebesar (30 + 219) = 249 meter. Jarak tersebut kemudian dibandingkan dengan jarak antara *node 1 – node 3* yang memiliki jarak lebih pendek yaitu 208 meter seperti pada Gambar 6.5. Dengan melalui perbandingan yang dilakukan jarak dari *node 1 – node 3* merupakan jalur terpendek yang dapat ditempuh dari titik awal. Maka dari itu, dilakukan perubahan pemilihan *node* yang akan dilalui. *Node 3* pun dipilih menjadi *node* yang memiliki jalur terpendek sedangkan *node 2* tidak dilalui dan *node* berubah menjadi berwarna merah tanda dipilih untuk dilalui seperti pada Gambar 6.6.

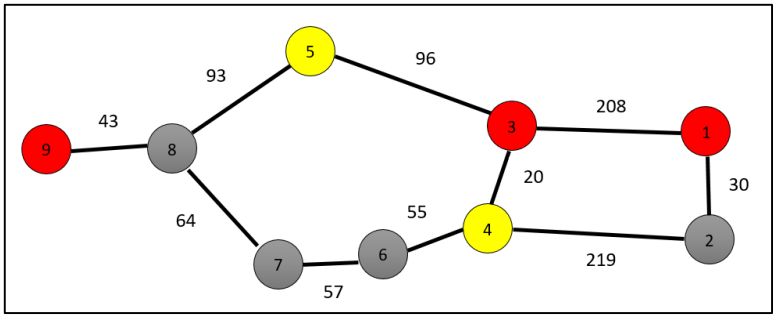


Gambar 6.5 Menentukan Kandidat *Node* Selanjutnya antara *Node 3* dan *Node* 4

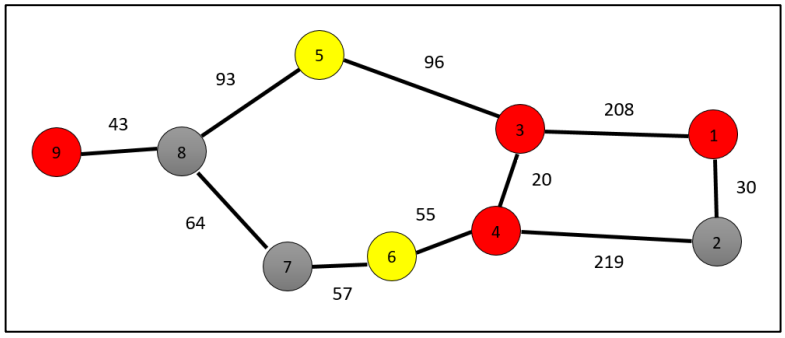


Gambar 6.6 Perubahan *Node* dengan Jarak Terpendek menjadi *Node 3*

Pencarian dilanjutkan dengan memperhitungkan *node* tetangga yang terhubung langsung dengan *node* 3 yaitu *node 4* dan *node 5*. *Node* yang memiliki jarak tempuh terpendek adalah *node 4* dengan jarak tempuh 20 meter, sedangkan jarak tempuh menuju ke *node 5* adalah 96 meter seperti pada Gambar 6.7. Dari perhitungan yang dilakukan jarak tempuh dari *node 1 – node 3 – node 4* memiliki jarak (208 + 20) = 228 meter, sedangkan jarak tempuh dari *node 1 – node 3 – node 5* memiliki jarak (208 + 96) = 304 meter. Dengan hasil yang didapat maka *node 4* yang memiliki nilai jarak tempuh terpendek menjadi *node* yang dilalui sedangkan *node* 5 tetap dijadikan pembanding untuk mencari jalur terpendek secara keseluruhan seperti pada Gambar 6.8.

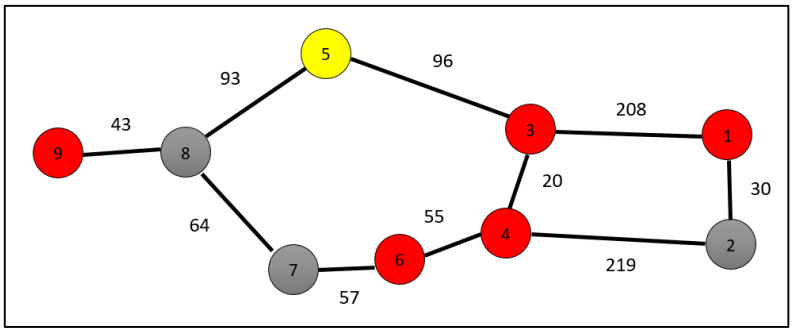


Gambar 6.7 Menentukan Kandidat *Node* Selanjutnya antara *Node 4* dan *Node* 5



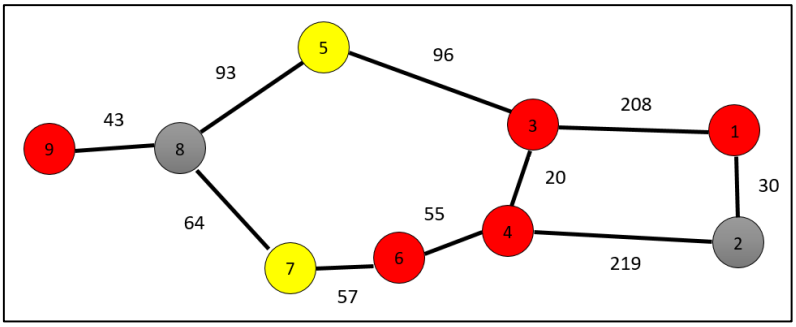
Gambar 6.8 Menentukan Node yang Dilalui yaitu *Node 4*

Pencarian dilanjutkan dengan memperhitungkan *node* tetangga yang terhubung langsung dengan *node* 4 yaitu *node 6* seperti pada Gambar 6.8. Karena tidak memiliki *node* lain maka *node* yang terpilih adalah *node 6* seperti pada Gambar 6.9. Jarak tempuh yang dimiliki dari *node 1 – node 3 – node 4 – node 6* adalah (208 + 20 + 55) = 383 meter. Setelah menjadikan *node 6* sebagai *node* terpendek dan tidak ditemukannya jalur lain maka dilakukan perbandingan kembali dengan jarak tempuh menuju *node 5* yang merupakan jalur yang berbeda hal ini dilakukan untuk menentukan jalur mana yang lebih pendek secara keseluruhan.



Gambar 6.9 Menentukan Node yang Dilalui yaitu *Node 6*

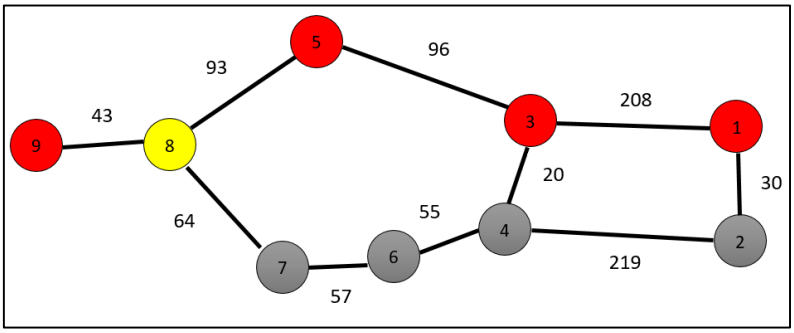
Perbandingan yang dilakukan antara *node 5 dan node 7* untuk mendapatkan jalur terpendek yang akan dilalui seperti pada Gambar 6.10. Perbandingan ini terjadi karena jarak tempuh yang dihasilkan secara keseluruhan mengalami peningkatan apabila mengikut *node 7* sehingga jarak tempuh menjadi lebih jauh, maka dengan ini dilakukan perbandingan kembali antara *node 7* dengan *node* 5 yang merupakan *node* yang berada disisi lain jalur pencarian. Setelah dilakukan perhitungan untuk mencapai *node 7* memiliki nilai jarak tempuh sebesar (208 + 20 + 55 + 57) = 340 meter, sedangkan seperti yang telah diperhitungkan sebelumnya jarak antara *node 1 – node 3 – node 5* adalah sebesar 304 meter. Maka dari itu, pada Gambar 6.10. terjadi perubahan *node* yang digunakan pada jalur pencarian menjadi melalui *node 1 – node 3 – node 5*. Hal ini mengubah jalur pencarian menjadi *node 1 – node 3 – node 5* seperti pada Gambar 6.11.



Gambar 6.10 Menentukan Kandidat *Node* Selanjutnya antara *Node 5* dan *Node 7*

Pencarian dilanjutkan dengan memperhitungkan *node* tetangga yang terhubung langsung dengan *node* 5 yaitu *node 8* seperti pada Gambar 6.11. Karena tidak memiliki *node* lain maka *node* yang terpilih adalah *node 6* seperti pada Gambar 6.12. Jarak tempuh yang dimiliki dari *node 1 – node 3 – node 5 – node 8* adalah (208 + 96 + 93) = 397 meter. Setelah itu menjadikan *node 8* sebagai *node* terpendek yang dilalui berikutnya. Karena tidak ditemukan *node* lain dan *node* yang ada hanya tinggal *node* yang dituju maka dilanjutkan ke pencarian

berikutnya. Kemudian sebelum mencapai titik tujuan dilakukan perbandingan dengan sisi yang lain namun setelah dilakukan perhitungan jarak tempuh yang dilalui pada sisi yang lain memiliki nilai lebih besar yang berarti jarak yang ditempuh lebih jauh. Dengan ini, jalur pencarian tidak mengalami perubahan sesuai dengan Gambar 6.11.

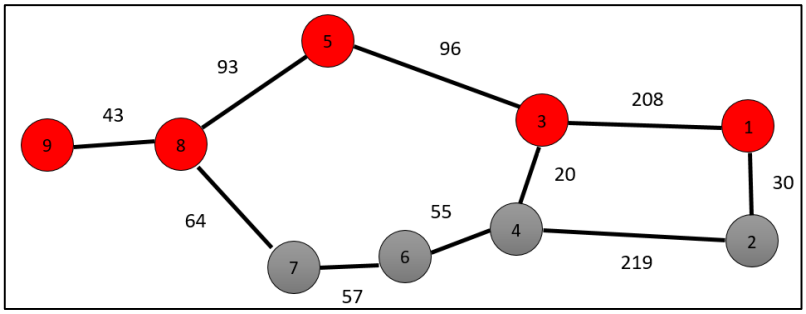


Gambar 6.11 *Node* yang Terpilih Berikutnya adalah *Node 5*

Pencarian memasuki tahap akhir, hal ini dikarenakan pencarian berikutnya merupakan *node* yang dicari yaitu *node 9.* Dengan ditemukannya *node* pencarian maka didapatkan jalur yang ditempuh oleh algoritme Dijkstra adalah sebagai berikut :

Node 1 –> Node 3 –> Node 5 –> Node 8 –> Node 9

Jalur pencarian tersebut menghasilkan jarak tempuh sebesar (208 + 96 + 93 + 43) = 440 meter. Dengan ditemukan titik pencarian berakhir pula algoritme Dijkstra (Gambar 6.12.).



Gambar 6.12 Hasil Pencarian Algoritme Dijkstra

6.1.2 Algoritme Voronoi Continuous K Nearest Neighbor

Pada bagian ini membahas alur kerja dari proses algoritme VCKNN. Algoritme VCKNN akan berfokus dalam mencari CKNN pada jalur pencarian. Penggunaan algoritme VCKNN akan dibuktikan dengan cara melakukan pencarian dari titik awal menuju titik tujuan. Titik pencarian digambarkan dengan titik berwarna merah. Titik *S1* merupakan titik awal pencarian dan titik *S2* merupakan titik tujuan. Poligon Voronoi *V(P)* digambarkan dengan garis berwarna hitam tebal. P_n merupakan *centroid* atau *interest point* dari setiap poligon Voronoi. Jalur yang dilalui akan diberikan garis berwarna merah untuk membedakan dengan jalur yang tidak dilalui.

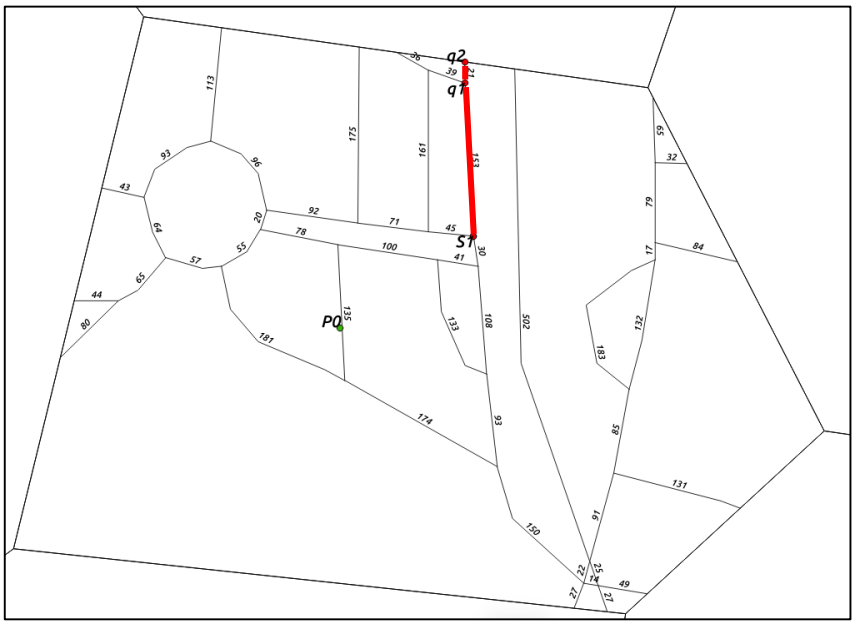
Gambar 6.13 Contoh dari Titik Pencarian pada Algoritme VCKNN

Kumpulan *split nodes* merupakan kumpulan hasil dari perpotongan antara poligon Voronoi dan jalur pencarian seperti pada *Lemma* 1 di penjelasan VCKNN. Pada Gambar 6.1. dapat diketahui titik *S1* berada pada *V(P0).* Maka *V(P0)* ditetapkan sebagai 1NN atau berdasarkan algoritme VCKNN berarti sama dengan *M = 1,* dan memiliki *CS = {P1, P2, P3, P8, P9}* yang dimana merupakan *interest point* terdekat dari *V(P0)*. Pada Gambar 6.13. dapat diketahui pula titik *S2* yang berada pada *V(P10).* Maka *V(P10)* merupakan lokasi dari poligon Voronoi dimana titik *S2* berada.

Titik *S1* merupakan titik yang memiliki pergerakan yang statis, artinya tidak mengalami perubahan pergerakan selama awal pencarian. Dimulai dari titik *S1*

akan dilakukan pencarian ke seluruh tepi poligon Voronoi P0 untuk mendapatkan *split nodes* yang memiliki jalur terpendek. Setelah pencarian terhadap seluruh jalur menuju tepi poligon Voronoi dilakukan kemudian akan ditentukan jalur yang memiliki jarak terdekat dari titik pencarian. Pada Gambar 6.14. didapatkan *split nodes* dengan jarak terpendek yaitu *q2* yang memiliki jarak 174 meter dari titik awal. Dengan ditemukannya jalur terpendek maka jalur tersebut akan dipilih untuk kemudian menjadi jalur yang dilalui.

Pada pencarian terhadap *split nodes* yang berada pada tengah poligon Voronoi dapat menggunakan bantuan dari algoritme Dijkstra. Algoritme Dijkstra digunakan untuk mendapatkan jalur terpendek untuk mencapai tepi poligon Voronoi. Hal ini sesuai dengan alur kerja algoritme VCKNN yang akan menggunakan *moving interval* secara langsung dari tepi poligon Voronoi satu ke tepi poligon Voronoi tetangga terdekat, namun pada pencarian di bagian tengah poligon Voronoi dapat menggunakan algoritme Dijkstra untuk mendapatkan *moving interval* ke tepi poligon Voronoi. Maka akan ditemukan *q1* sebagai jalur yang dipilih, yang merupakan titik yang berada antara titik *S1* dan titik *q2* seperti pada Gambar 6.14. Dengan demikian telah ditemukan jalur pencarian *q2* yang merupakan *moving interval* dari 1NN dan menghasilkan jumlah *split nodes* sebanyak 2 *split nodes*.

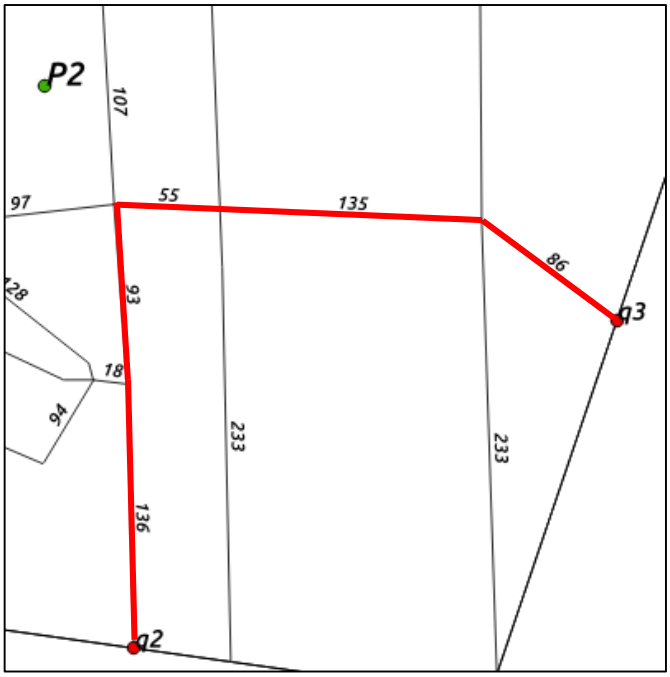


Gambar 6.14 Pencarian pada Poligon Voronoi P0

Sesuai dengan alur kerja VCKNN, setelah ditemukan *moving interval* dari 1NN. Kemudian mencari 2NN atau tetangga terdekat dari *V(P10)* dengan membandingkan jarak antara *generator* dan poligon Voronoi tetangga. Berdasarkan Gambar 6.13. *CS = {P1, P2, P3, P8, P9}* yang memiliki jarak terdekat dengan *split nodes* adalah *P2*. Setelah ditemukan *CS* terdekat adalah *P2,* dilakukan

perubahan nilai terhadap *M* menjadi *M = 2.* Dengan demikian *V(P2)* menjadi 2NN dari *V(P0)*.

Pada tahap ini, dilakukan pencarian kembali terhadap tepi poligon Voronoi tetangga berikutnya yaitu tepi poligon Voronoi untuk 3NN atau tetangga berikutnya dari *V(P2)*. Karena pencarian dimulai dari tepi poligon Voronoi maka dapat langsung menggunakan *moving interval* dari *V(P2)* dari *q2* menuju *q3* yang memiliki jarak tempuh 505 meter. Pada tahap ini mulai terlihat penurunan jumlah *split nodes* yang dihasilkan karena dengan menggunakan *moving interval* pencarian akan langsung menuju tepi poligon Voronoi tetangga berikutnya. Penurunan jumlah *split nodes* terjadi karena satu *moving interval* dihitung sebagai 1 *split nodes.* Dengan demikian tidak perlu dilakukan pencarian kembali terhadap seluruh jalur yang ada dan akan berpengaruh dalam menurunkan waktu pencarian. Pada Gambar 6.15. merupakan ilustrasi dari pencarian menggunakan *moving interval* dari *q2* menuju ke *q3.*

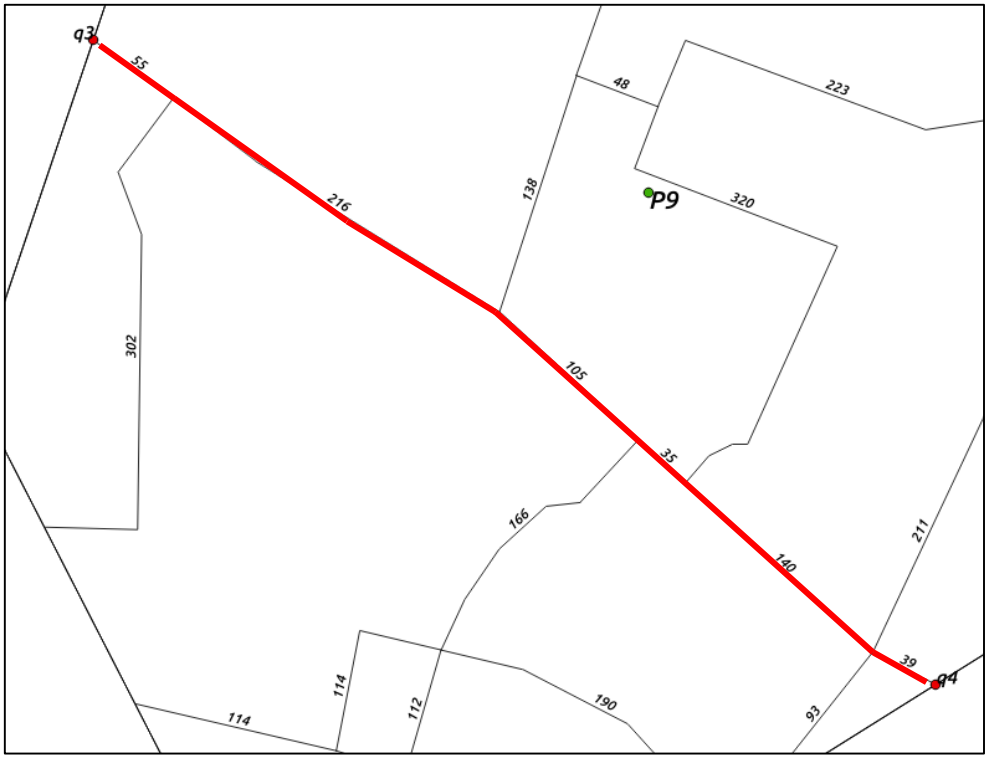


Gambar 6.15 Moving Interval dari Poligon Voronoi P2

Setelah selesai melakukan pencarian sampai ke 2NN. Maka kemudian kembali mencari 3NN atau tetangga terdekat dari *V(P2)* dengan kembali membandingkan jarak antara *generator* dan poligon Voronoi. Kemudian diketahui poligon Voronoi yang memiliki jarak terdekat dengan *split nodes* adalah *P9*. Setelah ditemukan *CS* terdekat adalah *P9,* dilakukan perubahan nilai terhadap *M* menjadi *M = 3.* Dengan demikian *V(P9)* menjadi 3NN dari *V(P2)*.

Pada tahap ini, dilakukan pencarian terhadap tepi poligon Voronoi berikutnya yaitu tepi poligon Voronoi untuk 3NN atau tetangga berikutnya dari *V(P2)*. Karena

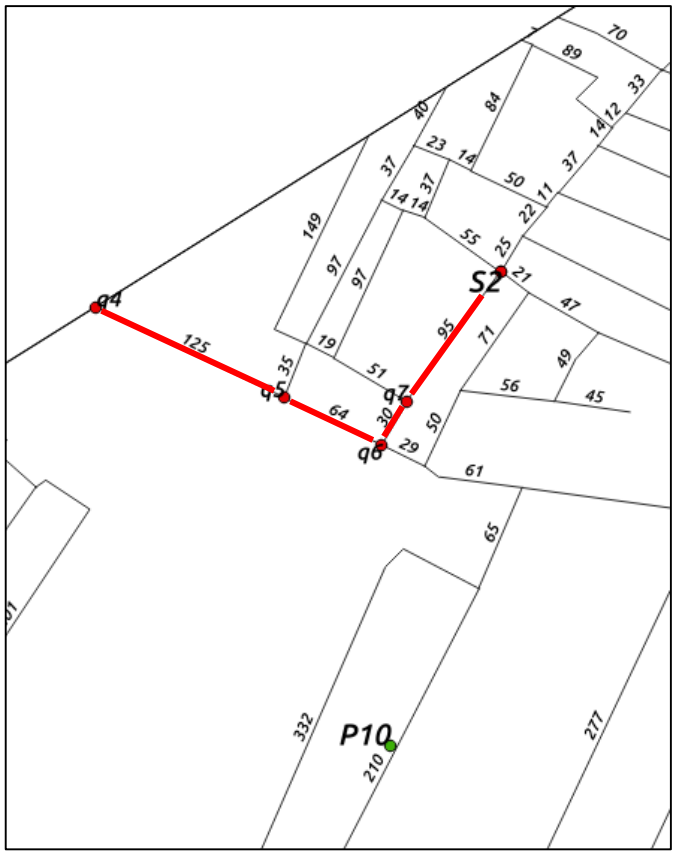
pencarian dimulai dari tepi poligon Voronoi maka dapat langsung menggunakan *moving interval* seperti pada tahapan sebelumnya yang dimulai dari *V(P2)* dengan *q3* sebagai *generator* yang menuju *q4* yang memiliki jarak tempuh 590 meter. Pada tahap ini kembali terlihat penurunan jumlah *split nodes* yang dihasilkan. Dengan demikian tidak perlu dilakukan pencarian kembali terhadap seluruh jalur yang ada. Untuk tahapan mencari tetangga ke-3 dan seterusnya memiliki langkah yang sama yaitu menggunakan *moving interval* apabila lokasi yang dicari memiliki NN lebih dari atau sama dengan 3. Akan tetapi untuk mencapai lokasi yang berada ditengah poligon Voronoi maka akan dilakukan pecarian sesuai penjelasan berikutnya. Pada Gambar 6.16. merupakan ilustrasi dari pencarian menggunakan *moving interval* dari *q3* menuju ke *q4.*



Gambar 6.16 Moving Interval dari Poligon Voronoi P9

Setelah selesai melakukan pencarian sampai ke 3NN. Maka kemudian kembali mencari 4NN atau tetangga terdekat dari *V(P9)* dengan kembali membandingkan jarak antara *generator* dan poligon Voronoi. Kemudian diketahui poligon Voronoi yang memiliki jarak terdekat dengan *split nodes* adalah *P10*. Setelah ditemukan *CS* terdekat adalah *P10,* dilakukan perubahan nilai terhadap *M* menjadi *M = 4.* Dengan demikian *V(P10)* menjadi 4NN dari *V(P9)*. Dengan ditemukannya titik *S2* yang merupakan tujuan, maka 4NN merupakan tetangga terakhir yang dicari sesuai dengan algoritme VCKNN *M = K* pencarian mengenai tetangga dari poligon Voronoi berakhir namum algoritme VCKNN masih mencari jalur dari *generator* pada tepi poligon Voronoi *P10* ke titik *S2.*

Pada tahap ini, dilakukan pencarian terhadap titik *S2* yaitu titik tujuan dari pencarian yang dilakukan. Karena pencarian dimulai dari tepi poligon Voronoi *q4* maka dilakukan pencarian terhadap jalur terdekat dengan titik *S2* menggunakan algoritme Dijkstra. Pencarian dimulai dari *q4* karena hanya terdapat 1 jalur maka jalur tersebut menjadi jalur yang dipilih menjadi *q5*. Dari *q5* dilakukan pencarian kembali untuk menemukan jalur berikutnya, jalur yang dipilih menjadi jalur *q6.* Jalur *q6* merupakan jalur terdekat dengan titik *S2.* Begitu pula terhadap pencarian ke jalur *q7* yang merupakan *split nodes* terakhir sebelum mencapai titik tujuan. Pencarian yang dilakukan dengan algoritme Dijkstra membadingkan seluruh kemungkinan sesuai dengan ilustrasi pada sub-bab sebelumnya. Dengan ditemukan *q7* maka pencarian jalur memasuki tahap akhir yaitu memilih jalur yang menuju langsung ke titik tujuan, titik *S2.* Setelah ditemukan titik *S2* maka berakhirlah pencarian menggunakan algoritme VCKNN. Dengan demikian jarak yang ditempuh dari *q4* sampai titik *S2* adalah 314 meter dan memiliki jumlah *split nodes* sebesar 5 *split nodes*. Pada Gambar 6.17. merupakan ilustrasi dari pencarian menggunakan algoritme Dijkstra dari *q4* menuju ke titik *S2.*



Gambar 6.17 Pencarian Titik Tujuan pada Poligon Voronoi P10

Dari hasil analisis di atas, dapat terlihat proses dari algoritme VCKNN. Dari percobaan pencarian dari titik *S1* menuju titik *S2* didapatkan informasi sebagai berikut. Total jarak tempuh yang dihasilkan adalah sebesar 1,582 meter atau 1,5

km dengan jumlah *split nodes* sebesar 8 *split nodes.* Sebelum menggunakan algoritme VCKNN ditemukan sebanyak 15 *split nodes.* Hasil ini membuat performa dari pencarian meningkat sampai dengan 53%, perhitungan ini didapatkan dari perhitungan jumlah *split* nodes yang terbentuk tanpa menggunakan poligon Voronoi dibandingkan dengan penggunaan poligon Voronoi. Pada pencarian ini pula jumlah segmentasi yang terbentuk adalah sebanyak 4 segmen. Segmentasi yang terbentuk berdasarkan dengan jumlah poligon Voronoi yang dilalui. Dengan waktu tempuh sebesar 0.055 detik. VCKNN. Pada Gambar 6.18. merupakan ilustrasi secara menyeluruh terhadap jalur pencarian dari titik *S1* menuju titik *S2* berserta dengan poligon Voronoi yang dilalui.

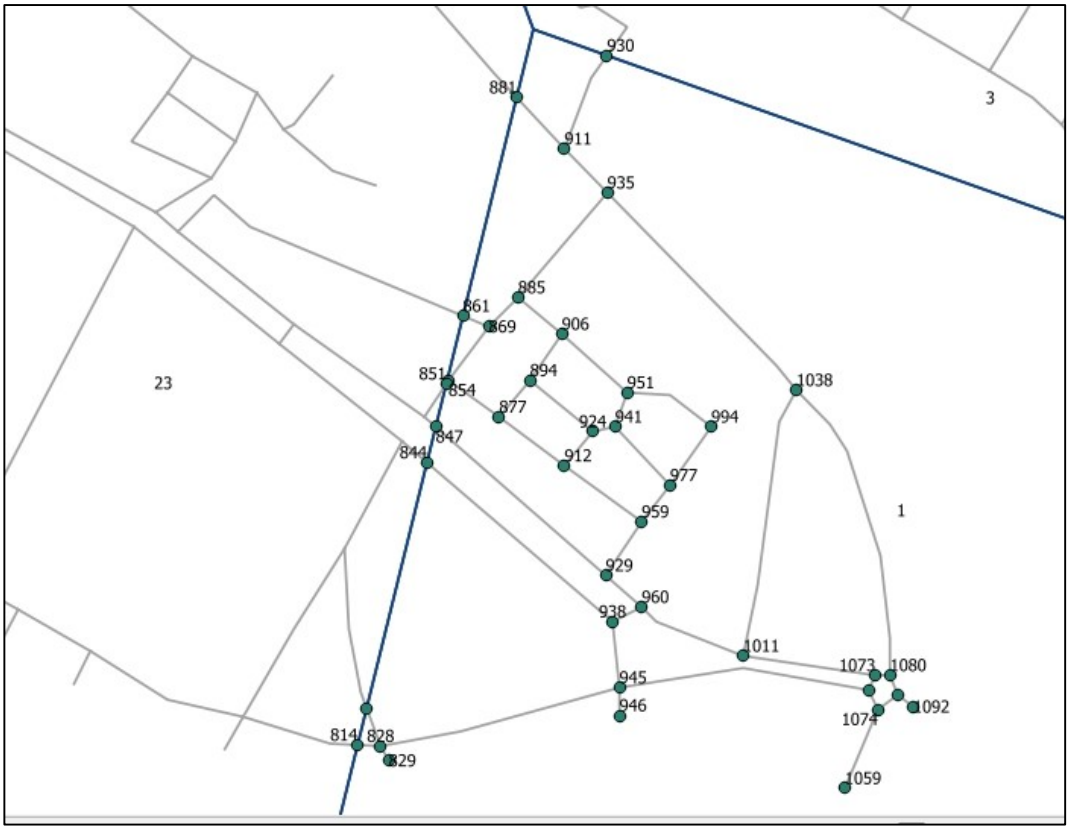
Gambar 6.18 Hasil Pencarian dari Algoritme VCKNN

6.2 Evaluasi Algoritme

Pada tahap ini, dilakukan evaluasi mengenai algoritme yang diusulkan dan elemen pendukung dari algoritme tersebut. Evaluasi yang dilakukan adalah dengan melakukan percobaan terhadap algoritme itu sendiri atau perbandingan dengan algoritme lain untuk memastikan performa dari algoritme tersebut. Proses evaluasi dilakukan dengan menguji beberapa aspek sesuai dengan rumusan masalah yang dipaparkan sebelumnya. Pada proses evaluasi akan di ilustrasikan dalam bentuk gambar dengan penjelasan. Hasil dari evaluasi tersebut akan dijadikan bahan dalam memilih algoritme yang di implementasikan dalam bentuk kode program.

6.2.1 Evaluasi Pencarian *Moving Interval* dengan Algoritme Dijkstra

Evaluasi yang dilakukan pada tahap ini adalah menguji kinerja algoritme Dijkstra terhadap pencarian *moving interval* pada poligon Voronoi. Pada Gambar 6.19. merupakan contoh dari data jaringan jalan yang dibatasi oleh poligon Voronoi, data tersebut akan dicari bagian ujung dari tepi poligon Voronoi untuk menemukan *split nodes* yang akan menjadi *move interval*. Pencarian pada Gambar 6.19. akan dimulai dari titik dengan id 814. Evaluasi dilakukan dengan menggunakan tabel pencarian algoritme Dijkstra. Dengan menggunakan algoritme Dijkstra, pencarian akan dilakukan ke seluruh jalur yang ada dan akan dibandingkan jarak tempuh yang dilalui untuk sampai ke tepi poligon Voronoi beserta jumlah iterasi yang dibutuhkan.



Gambar 6.19 Pencarian *Move Interval*

Pada Lampiran A. merupakan penjabaran dari proses pencarian menggunakan algoritme Dijkstra yang telah dilakukan pada poligon Voronoi pada Gambar 6.19. dari hasil tersebut diketahui, untuk mendapatkan hasil pencarian yang dimulai dari id 814 sampai ke ujung tepi poligon Voronoi melalui 10 kali iterasi. Dari iterasi 1 sampai dengan iterasi ke 10 seluruh jalur pencarian telah dilewati dan ditemukan. Berikut ini merupakan penjabaran dari hasil setiap iterasi mulai dari iterasi terakhir sampai iterasi pertama :

Tabel 6.1 Hasil Pencarian *Moving Interval* dengan Algoritme Dijkstra

Iterasi ke -	ID Jalan yang Ditemukan
10	851
	861
9	854
	881
	894
	896
	906
	930
	951
8	877
	885
	911
	924
	941
	994
7	912
	935
	977
6	847
	959
	1038
	1092
5	929
	1011
	1059
	1080
	1084
4	844
	960
	1073
	1074
3	938
	946
	1069
2	818
	829
	945
1	828
0	814

Dari rangkuman hasil evaluasi pada Tabel 6.1. pencarian menggunakan algoritme Dijkstra, diketahui telah melakukan pencarian ke seluruh jalan yang ada. Hal ini ditunjukan dengan jumlah id yang ada pada Tabel 6.1. sama dengan jumlah id yang ada pada contoh Gambar 6.19. Untuk melakukan pemeriksaan pencarian telah dilakukan dengan benar akan dilakukan ilustrasi pencarian sebagai berikut :

Misalkan diambil contoh sebuah pencarian dari titik awal yaitu pada id 841 menuju tepi poligon Voronoi dengan id 844, maka berdasarkan hasil pengamatan yang tampak pada Gambar 6.19. jalur yang akan dipilih seharusnya melalui jalur

dengan id 938. Dengan menggunakan Tabel 6.1. akan diperiksa mengenai jalur yang dilewati untuk sampai ke id 844.

Hal pertama yang akan dilakukan adalah mencari id 844 berada pada tahap iterasi-*n* dan ditemukan titik yang dicari berada ada pada iterasi-4. Setelah ditemukan iterasi tempat titik pencarian berada, maka akan dilakukan pencarian terhadap iterasi sebelumnya yaitu ke iterasi-3 yang memiliki jalur iterasi-3 = {938,946,1069} kemudian akan dilakukan pemeriksaan menggunakan tabel pada Lampiran A dengan melihat id 844 memiliki jalur sebelumnya 938 dan pada iterasi-3. Setelah dilakukan pemeriksaan didapatkan hasil yang sesuai. Maka dari itu pencarian untuk mendapatkan *moving interval* dinyatakan berhasil dan untuk melakukan pemeriksaan jalur sebelumnya lagi dapat dilakukan dengan tahapan yang sama begitu pula untuk pencarian jalur yang lain.

6.2.2 Evaluasi Perbandingan Data Kecamatan dan Data Kelurahan

Evaluasi yang dilakukan pada tahap ini adalah melakukan perbandingan antara penggunaan data kecamatan dan penggunaan data kelurahan untuk dijadikan *interest point* pada algoritme VCKNN. Kedua data tersebut melalui proses *pre-processing* yang sama dan dari kedua data tersebut diambil lokasi yang sama sebagai *interest point* yaitu pada bagian *centroid* yang merupakan titik tengah dari wilayah kecamatan/kelurahan. Pada evaluasi ini akan menjabarkan mengenai hubungan antara pemilihan *interest point* terhadap jumlah *split nodes* yang dihasilkan.

Pada awal penelitian terdapat hipotesis awal (H_0) sebagai berikut :

- Mengetahui tidak adanya pengaruh antara penggunaan batas administrasi Kecamatan atau Kelurahan sebagai *interest point* terhadap jumlah *split nodes* yang dihasilkan.
- Mengetahui tidak adanya pengaruh antara penggunaan poligon Voronoi berdasarkan *interest point* terhadap jumlah *split nodes* yang dihasilkan.

Untuk mendapatkan jawaban dari hipotesis tersebut dilakukan perbandingan antara data Kecamatan dan data Kelurahan. Perbandingan yang dilakukan adalah menemukan hubungan serta dampak dari penggunaan poligon Voronoi terhadap jumlah *split nodes* yang dihasilkan dan perbandingan penggunaan data Kecamatan dan data Kelurahan sebagai *interest point*.

Pada Tabel 6.2. merupakan hasil dari pencarian *split nodes* pada 11 wilayah Kecamatan. Pencarian dilakukan untuk mendapatkan jumlah *split nodes* dari data Kecamatan yang tidak diolah menggunakan poligon Voronoi, nantinya akan digunakan oleh algoritme Dijkstra dan data Kecamatan yang menggunakan poligon Voronoi, nantinya akan digunakan oleh algoritme VCKNN.

Tabel 6.2 Perbandingan Jumlah *Split Nodes* pada Data Kecamatan

Kecamatan		Jumlah Split Nodes Tanpa V(P)	Jumlah Split Nodes dengan V(P)	Total
Dau	*Observed*	1186	26	1212
	Expected	1111.8494	100.150597	
Lowokwaru	*Observed*	1674	153	1827
	Expected	1676.03041	150.969587	
Karangploso	*Observed*	290	36	326
	Expected	299.061803	26.9381968	
Singosari	*Observed*	1378	28	1406
	Expected	1289.8187	116.181303	
Blimbing	*Observed*	1007	184	1191
	Expected	1092.58469	98.4153139	
Klojen	*Observed*	535	116	651
	Expected	597.206239	53.793761	
Sukun	*Observed*	1505	95	1600
	Expected	1467.78799	132.212009	
Wagir	*Observed*	417	29	446
	Expected	409.145903	36.8540974	
Kedung Kandang	*Observed*	1189	128	1317
	Expected	1208.17299	108.82701	
Tumpang	*Observed*	65	22	87
	Expected	79.810972	7.18902797	
Tajinan	*Observed*	135	28	163
	Expected	149.5309016	13.46909838	
Total (*Observed*)		9381	845	10226
$x_c^2 = \sum \frac{(O_i - E_i)^2}{E_i}$		29.99358788	332.9820685	362.9756564

Pada Tabel 6.2. dilakukan analisis sebagai berikut, jumlah *split nodes* yang dihasilkan pada hasil *pre-processing* menggunakan poligon Voronoi lebih rendah dibandingkan dengan hasil *pre-processing* tanpa menggunakan poligon Voronoi.

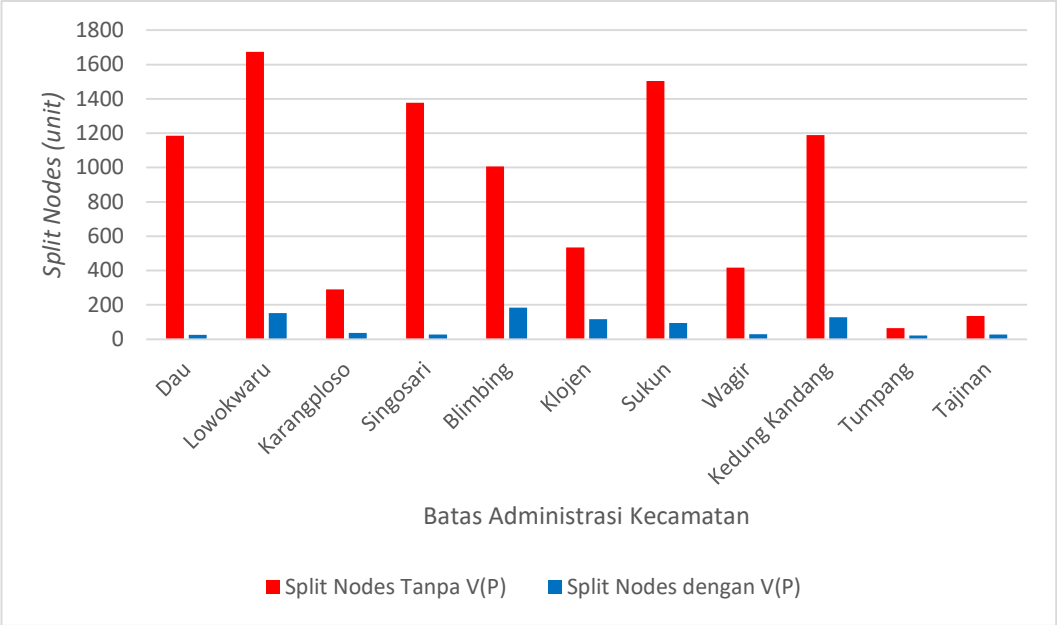
Hal ini terjadi karena jumlah *split nodes* yang dihasilkan menggunakan poligon Voronoi menjadi hanya pada bagian tepi poligon Voronoi sesuai *lemma 1* pada penjelasan VCKNN. Penurunan yang terjadi dari total 9381 *split nodes* pada data yang diolah tanpa menggunakan poligon Voronoi menjadi 845 *split nodes* pada data yang diolah dengan poligon Voronoi, sehingga diketahui prosentase penurunan jumlah *split nodes* adalah sebesar 90,9%.

Pada Tabel 6.2. dilakukan uji statistik menggunakan *chi-square* untuk melakukan pengujian terhadap hipotesis yang diambil pada awal penelitian dengan hasil penelitian yang didapatkan. Pada uji hipotesis tersebut digunakan nilai *observed* berdasarkan hasil *query* terhadap data Kecamatan dan nilai *expected* berdasarkan rata-rata nilai antara data yang diolah tanpa poligon Voronoi dan data yang diolah dengan poligon Voronoi. Nilai dari H_1 merupakan hasil dari perhitungan dengan *chi-squre* terhadap data *observed* dan data *expected.* Nilai dari α merupakan nilai signifikasi dari data. Nilai df didapatkan dari jumlah data uji. Sedangkan untuk nilai H_0 didapatkan dari tabel *chi-squre* berdasarkan df dan α. Berikut ini merupakan hasil dari uji statistik dari hipotesis di atas :

α = 0,05 (5%)	H_0 = 18,307038
df = 10	H_1 = 362,9756564
$H_1 > H_0$	

- H_0, tidak adanya pengaruh antara penggunaan poligon Voronoi berdasarkan *interest point* terhadap jumlah *split nodes* yang dihasilkan.
- H_1, adanya pengaruh antara penggunaan poligon Voronoi berdasarkan *interest point* terhadap jumlah *split nodes* yang dihasilkan.

Pada hasil uji statistik didapatkan $H_1 > H_0$ yang menyatakan bahwa hipotesis awal ditolak dan hipotesis akhir diterima karena ditemukan adanya hubungan antara penggunaan poligon Voronoi dan jumlah *split nodes*. Pada Gambar 6.20 merupakan grafik yang menggambarkan hasil penurunan jumlah *split nodes* yang terjadi pada data Kecamatan yang diolah tanpa menggunakan poligon Voronoi dan diolah dengan menggunakan poligon Voronoi, penurunan terjadi di seluruh wilayah Kecamatan.



Gambar 6.20 Hasil Perbandingan Jumlah *Split Nodes* pada Kecamatan

Pada Tabel 6.3. merupakan hasil dari pencarian *split nodes* pada 8 wilayah Kelurahan. Pencarian dilakukan untuk mendapatkan jumlah *split nodes* dari data Kelurahan yang tidak diolah menggunakan poligon Voronoi, nantinya akan digunakan oleh algoritme Dijkstra dan data Kelurahan yang menggunakan poligon Voronoi, nantinya akan digunakan oleh algoritme VCKNN.

Tabel 6.3 Perbandingan Jumlah *Split Nodes* pada Data Kelurahan

Kelurahan		Jumlah Split Nodes Tanpa V(P)	Jumlah Split Nodes dengan V(P)	Total
Sekar Puro	*Observed*	543	9	552
	Expected	513.1541168	38.84588318	
Asrikaton	*Observed*	35	15	50
	Expected	46.48135116	3.518648839	
Tanjungtirto	*Observed*	85	16	101
	Expected	93.89232935	7.107670654	
Kota Lama	*Observed*	23	21	44
	Expected	40.90359	3.096411	
Rampal Celaket	*Observed*	58	8	66
	Expected	61.35538353	4.644616467	

Bunulrejo	*Observed*	219	7	226
	Expected	210.0957072	15.90429275	
Merjosari	*Observed*	199	15	214
	Expected	198.940183	15.05981703	
Tunjungsekar	*Observed*	159	9	168
	Expected	156.1773399	11.8226601	
Total (*Observed*)		1321	100	1421
$x_c^2 = \sum \frac{(O_i - E_i)^2}{E_i}$		14.65357482	193.5737233	208.2272982

Pada Tabel 6.3. dilakukan analisis sebagai berikut, jumlah *split nodes* yang dihasilkan pada hasil *pre-processing* menggunakan poligon Voronoi lebih rendah dibandingkan dengan hasil *pre-processing* tanpa menggunakan poligon Voronoi sama seperti hasil pada data Kecamatan. Penurunan yang terjadi dari total 1321 *split nodes* pada data yang diolah tanpa menggunakan poligon Voronoi menjadi 100 *split nodes* pada data yang diolah dengan poligon Voronoi, sehingga diketahui prosentase penurunan jumlah *split nodes* adalah sebesar 92,4% lebih tinggi dibandingkan dengan data Kecamatan.

Pada Tabel 6.3. dilakukan uji statistik menggunakan *chi-square* untuk melakukan pengujian terhadap hipotesis yang diambil pada awal penelitian dengan hasil penelitian yang didapatkan. Pada uji hipotesis tersebut digunakan nilai *observed* berdasarkan hasil *query* terhadap data Kelurahan dan nilai *expected* berdasarkan rata-rata nilai antara data yang diolah tanpa poligon Voronoi dan data yang diolah dengan poligon Voronoi. Nilai dari H_1 merupakan hasil dari perhitungan dengan *chi-squre* terhadap data *observed* dan data *expected.* Nilai dari α merupakan nilai signifikasi dari data. Nilai df didapatkan dari jumlah data uji. Sedangkan untuk nilai H_0 didapatkan dari tabel *chi-squre* berdasarkan df dan α. Berikut ini merupakan hasil dari uji statistik dari hipotesis di atas :

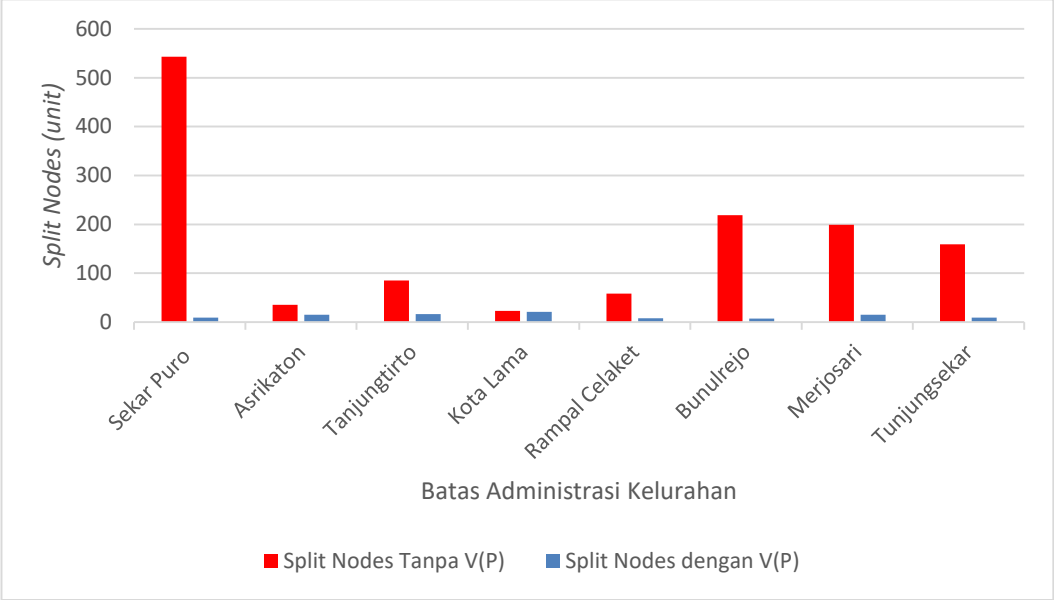
α = 0,05 H_0 = 14,067140

df = 7 H_1 = 208,2272982

$H_1 > H_0$

- H_0, tidak adanya pengaruh antara penggunaan poligon Voronoi berdasarkan *interest point* terhadap jumlah *split nodes* yang dihasilkan.
- H_1, adanya pengaruh antara penggunaan poligon Voronoi berdasarkan *interest point* terhadap jumlah *split nodes* yang dihasilkan.

Pada hasil uji statistik didapatkan $H_1 > H_0$ yang menyatakan bahwa hipotesis awal dibantah dan hipotesis akhir diterima karena ditemukan adanya hubungan antara penggunaan poligon Voronoi dan jumlah *split nodes*. Pada Gambar 6.21 merupakan grafik yang menggambarkan hasil penurunan jumlah *split nodes* yang terjadi pada data Kecamatan yang diolah tanpa menggunakan poligon Voronoi dan diolah dengan menggunakan poligon Voronoi, penurunan terjadi di seluruh wilayah Kelurahan.



Gambar 6.21 Hasil Perbandingan Jumlah *Split Nodes* pada Data Kelurahan

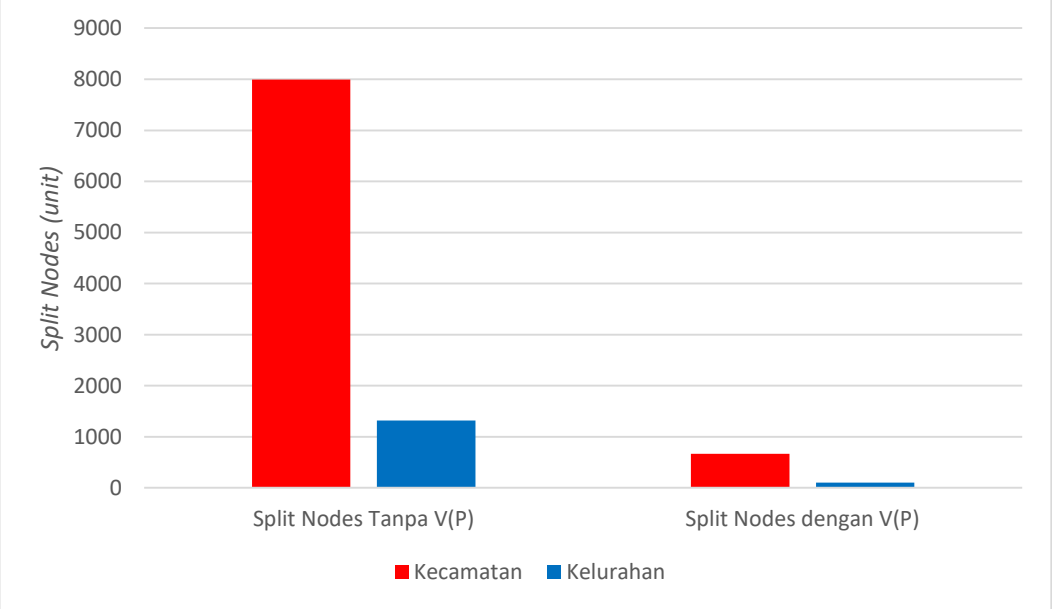
Berdasarkan hasil perbandingan di atas, dapat diketahui bahwa pada kedua data sama sama mengalami penurunan jumlah *split nodes.* Namun pada data Kelurahan penurunan yang terjadi lebih besar yaitu 92,4% dibandingkan dengan data Kecamatan yaitu 90,9%. Pada Gambar 6.22. terlihat perbedaan antara *split nodes* pada 8 wilayah Kecamatan dan 8 wilayah Kelurahan. Pada wilayah Kecamatan daerah Tumpang dan Tanjinan tidak diikut sertakan dalam perbandingan agar jumlah dari wilayah antara Kecamatan dan Kelurahan akan sama yaitu 8 wilayah.

Kemudian dilakukan uji statistik menggunakan *chi-square* untuk melakukan pengujian terhadap hipotesis yang diambil pada awal penelitian dengan hasil penelitian yang didapatkan. Pada uji hipotesis tersebut digunakan hasil dari uji hipotesis sebelumnya untuk mengetahui hubungan antara pemilihan penggunaan data Kecamatan atau data Kelurahan sebagai *interest point* terhadap jumlah *split nodes* yang dihasilkan. Berikut ini merupakan hasil dari uji statistik dari hipotesis tersebut :

$\alpha = 0,05$	$H_0 = 27,587112$
$df = 17$	$H_1 = 495,0191111$
$H_1 > H_0$	

- H_0, tidak adanya pengaruh antara penggunaan batas administrasi Kecamatan atau Kelurahan sebagai *interest point* terhadap jumlah *split nodes* yang dihasilkan.
- H_1, adanya pengaruh antara penggunaan batas administrasi Kecamatan atau Kelurahan sebagai *interest point* terhadap jumlah *split nodes* yang dihasilkan.

Pada hasil uji statistik didapatkan $H_1 > H_0$ yang menyatakan bahwa hipotesis awal dibantah dan hipotesis akhir diterima karena ditemukan adanya hubungan antara penggunaan batas administrasi Kecamantan dan Kelurahan sebagai *interest point* dan jumlah *split nodes.* Dari hasil keseluruhan didapatkan kesimpulan bahwa untuk penelitian ini lebih cocok menggunakan data Kelurahan dibandingkan data Kecamatan karena data Kelurahan memiliki wilayah yang lebih kecil sehingga *split nodes* yang dihasilkan akan lebih sedikit seperti yang ditunjukan Gambar 6.22.



Gambar 6.22 Perbandingan Jumlah *Split Node* di Kecamatan dan Kelurahan

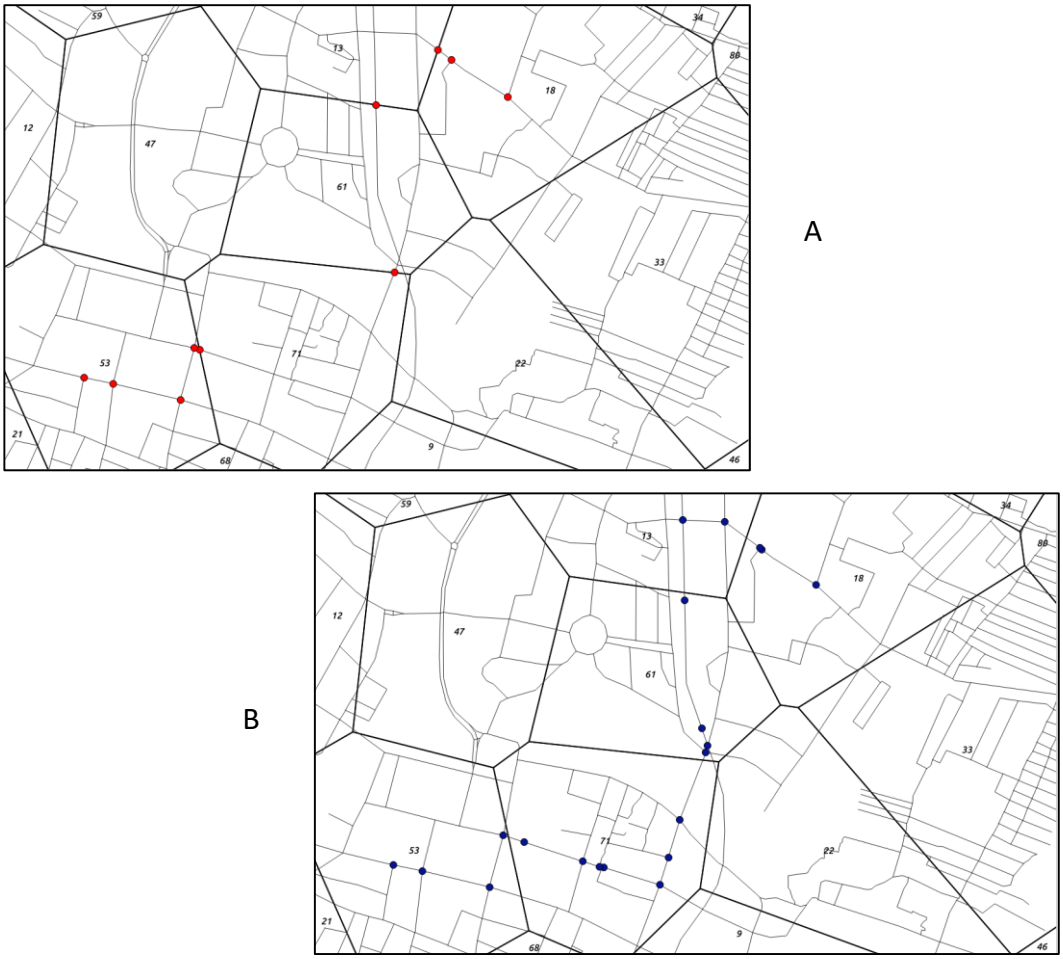
6.2.3 Evaluasi Performa Algoritme VCKNN dan Algoritme Dijkstra

Evaluasi yang dilakukan pada tahap ini adalah melakukan perbandingan performa antara penggunaan algoritme VCKNN dan algoritme Dijkstra. Perbandingan performa yang dilakukan adalah pada aspek pembagian segmentasi, *runtime* dan jumlah *split nodes*. Pengujian yang dilakukan adalah dengan menjalankan kedua algoritme dengan masukkan yang sama. Pada pengujian yang dilakukan seperti pada Gambar 6.23. dilakukan pencarian terhadap koordinat titik awal menuju koordinat titik tujuan. Pada pencarian tersebut diberikan masukkan koordinat sebagai berikut :

Koordinat Titik Awal : 112.627868,-7.9844860009999

Koordinat Titik Tujuan : 112.641449,-7.975516001

Dengan melakukan masukkan di atas, maka didapatkan jalur pencarian seperti pada Gambar 6.23.



Gambar 6.23 Pengujian Pencarian Titik : (A) Algoritme VCKNN dan (B) Algoritme Dijkstra dengan Melalui 5NN

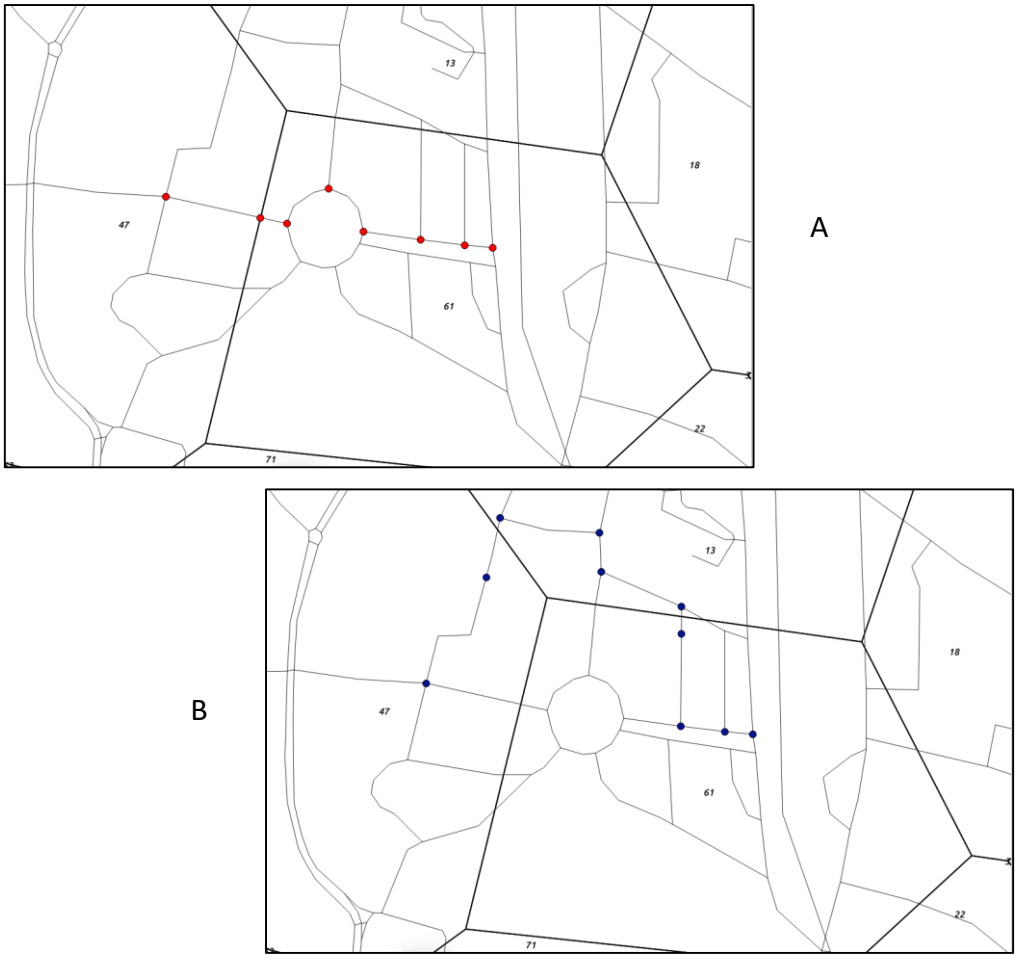
Dari hasil yang didapat pada Gambar 6.23. jalur pencarian yang didapatkan menggunakan algoritme VCKNN dan algoritme Dijkstra sama, namun terdapat perbedaan pada jumlah *split nodes* dan *runtime*. Pada penggunaan algoritme VCKNN *split nodes* yang terbentuk sebesar 10 *split nodes*, sedangkan pada penggunaan algoritme Dijkstra *split nodes* yang terbentuk sebesar 20 *split nodes.* Pada penggunaan algoritme VCKNN *runtime* yang dihasilkan adalah 0,01 detik, sedangkan pada algoritme Dijkstra *runtime* yang dihasilkan adalah 5,03 detik. Kedua algoritme tersebut melalui pembagian segmentasi yang sama yaitu 5 poligon Voronoi atau 5NN.

Pengujian berikutnya dilakukan pencarian pada koordinat titik awal dan koordinat titik tujuan sebagai berikut :

Koordinat Titik Awal : 112.627868,-7.9844860009999

Koordinat Titik Tujuan : 112.641449,-7.975516001

Dengan melakukan masukkan di atas, maka didapatkan jalur pencarian seperti pada Gambar 6.24.



Gambar 6.24 Pengujian Pencarian Titik : (A) Algoritme VCKNN dan (B) Algoritme Dijkstra dengan Melalui 2NN

Dari hasil yang didapat pada Gambar 6.24. jalur pencarian yang didapatkan menggunakan algoritme VCKNN dan algoritme Dijkstra berbeda, selain pada pemilihan jalur perbedaan terjadi pada pembagian segmentasi, jumlah *split nodes* dan *runtime*. Pada penggunaan algoritme VCKNN *split nodes* yang terbentuk sebesar 8 *split nodes*, sedangkan pada penggunaan algoritme Dijkstra *split nodes* yang terbentuk sebesar 10 *split nodes.* Pada penggunaan algoritme VCKNN *runtime* yang dihasilkan adalah 0,02 detik, sedangkan pada algoritme Dijkstra *runtime* yang dihasilkan adalah 5,04 detik. Kedua algoritme tersebut melalui pembagian segmentasi yang berbeda, pada algoritme VCKNN melalui 2 poligon Voronoi atau 2NN sedangkan pada algoritme Dijkstra melalui 3 poligon Voronoi atau 3NN. Dengan hasil yang didapat diketahui bahwa pemilihan jalur menggunakan algoritme VCKNN dan algortime Dijkstra dapat menghasilkan jalur pencarian yang sama atau sebaliknya dapat menghasilkan jalur pencarian yang berbeda.

Kemudian dilakukan pengujian pada algoritme VCKNN sebanyak 10 kali pencarian titik yang menghasilkan pencarian seperti pada Tabel 6.4. *k* merupakan jumlah pembagian segmentasi berdasarkan poligon Voronoi, sedangkan hasil yang dicatat dari pencarian tersebut adalah jumlah *split nodes* yang dihasilkan beserta *runtime* yang dibutuhkan.

Tabel 6.4 Hasil Uji pada Algoritme VCKNN

k	*Split Nodes* VCKNN	*Runtime* VCKNN
5	10	0.01
	16	0.01
4	9	0.01
	18	0.01
3	7	0.01
	10	0.01
2	16	0.01
	8	0.02
1	8	0.01
	2	0.02

Hal yang sama dilakukan pada algoritme Dijkstra yaitu pengujian sebanyak 10 kali dengan pencarian titik yang sama dengan algoritme VCKNN dan menghasilkan pencarian seperti pada Tabel 6.5. *k* merupakan jumlah pembagian segmentasi berdasarkan poligon Voronoi meskipun pada algoritme Dijkstra tidak menggunakan poligon Voronoi sebagai bagian dari pencarian, namun poligon Voronoi ini digunakan agar dapat ditarik perbandingan yang sama pada segmentasi yang dilewati dengan algoritme VCKNN. Hasil yang dicatat dari

pencarian tersebut adalah jumlah *split nodes* yang dihasilkan beserta *runtime* yang dibutuhkan sama seperti algoritme VCKNN.

Tabel 6.5 Hasil Uji pada Algoritme Dijkstra

k	*Split Nodes* Dijkstra	*Runtime* Dijkstra
5	20	5.03
	25	4.98
4	17	5.11
	27	5.02
3	15	4.86
	15	4.56
2	17	5.52
	10	5.04
1	8	4.77
	2	4.72

Berdasarkan hasil uji yang telah dilakukan didapatkan perbandingan sebagai berikut. Pada Tabel 6.6. perbandingan yang dilakukan adalah pada jumlah *split nodes* yang dihasilkan terhadap pembagian segmentasi. Pada Tabel 6.7. perbandingan yang dilakukan adalah pada *runtime* yang dibutuhkan untuk melakukan pencarian terhadap pembagian segmentasi. Pada Tabel 6.8. perbandingan yang dilakukan adalah pada jumlah *split nodes* dan *runtime* terhadap pembagian segmentasi.

Pada Tabel 6.6. algoritme Dijkstra dijadikan acuan dalam membandingkan jumlah *split nodes* yang ditemukan, sedangkan jumlah *split nodes* pada algoritme VCKNN dijadikan hasil observasi. Berikut merupakan hasil perbandingan jumlah *split nodes* terhadap pembagian segmentasi :

Tabel 6.6 Perbandingan *Split Nodes* Berdasarkan Segmen

k	*Split Nodes* VCKNN (*Observed*)	*Split Nodes* Dijkstra (*Expected*)
5	10	20
	16	25
4	9	17
	18	27
3	7	15

	10	15
2	16	17
	8	10
1	8	8
	2	2
$x_c^2 = \sum \frac{(O_i - E_i)^2}{E_i}$		21.39686275

Pada Tabel 6.6. dilakukan analisis sebagai berikut, jumlah *split nodes* yang dihasilkan oleh algoritme VCKNN lebih rendah dibandingkan dengan algoritme Dijkstra. Secara keseluruhan penurunan jumlah *split nodes* terjadi berdasarkan pembagian segmentasi dengan poligon Voronoi dimulai dari 2NN dan seterusnya penurunan akan semakin terlihat antara penggunaan algoritme VCKNN yang dibandingkan dengan algoritme Dijkstra saat jumlah segmentasi semakin besar. Namun pada pembagian segmentasi 1NN jumlah *split nodes* yang dihasilkan sama antara penggunaan algoritme VCKNN dan algoritme Dijkstra.

Pada Tabel 6.6. dilakukan uji statistik menggunakan *chi-square* untuk mengetahui performa dari algoritme VCKNN dan algoritme Dijkstra terhadap jumlah *split nodes* yang dihasilkan. Nilai dari H_0 merupakan asumsi bahwa algoritme Dijkstra menghasilkan jumlah *split nodes* yang lebih baik dibanding algoritme VCKNN. Nilai dari H_1 merupakan asumsi bahwa algoritme VCKNN lebih baik dari algoritme Dijkstra dalam menghasilkan jumlah *split nodes.* Berikut ini merupakan hasil dari uji statistik dari hipotesis di atas :

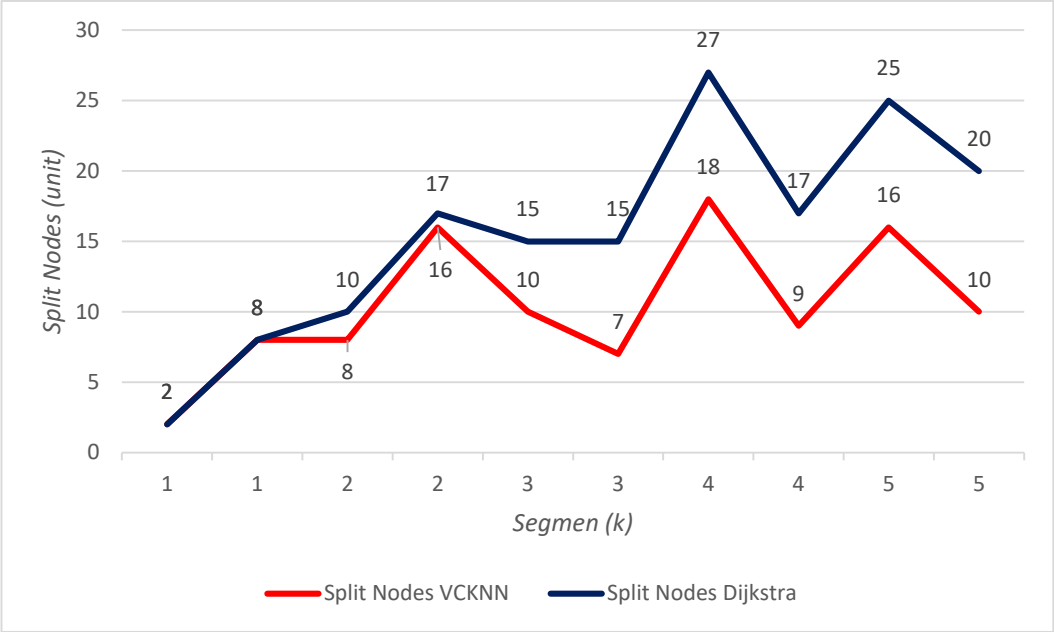
$\alpha = 0{,}05$ $H_0 = 16.918978$

$df = 9$ $H_1 = 21.39686275$

$H_1 < H_0$

- H_0, algortime Dijkstra lebih baik dalam menghasilkan jumlah *split nodes* dibandingkan dengan algoritme VCKNN.
- H_1, algortime VCKNN lebih baik dalam menghasilkan jumlah *split nodes* dibandingkan dengan algoritme Dijkstra.

Pada hasil uji statistik didapatkan $H_1 > H_0$ yang menyatakan bahwa hipotesis awal dibantah dan hipotesis akhir diterima karena performa dari algoritme VCKNN dalam menghasilkan *split nodes* lebih baik dibandingkan dengan algoritme Dijkstra. Dari hasil tersebut didapatkan kesimpulan bahwa pada penelitian ini algoritme VCKNN menghasilkan jumlah *split nodes* yang lebih baik dibandingkan algoritme Dijkstra seperti yang ditunjukan Gambar 6.25.



Gambar 6.25 Hasil Perbandingan *Split Nodes* Berdasarkan Segmen

Pada Tabel 6.7. algoritme Dijkstra dijadikan acuan dalam membandingkan *runtime* untuk menemukan titik tujuan, sedangkan *runtime* pada algoritme VCKNN dijadikan hasil observasi. Berikut merupakan hasil perbandingan *runtime* terhadap pencarian berdasarkan pembagian segmentasi :

Tabel 6.7 Perbandingan *Runtime* Berdasarkan Segmen

k	*Runtime* VCKNN (*Observed*)	*Runtime* Dijkstra (*Expected*)
5	0.01	5.03
	0.01	4.98
4	0.01	5.11
	0.01	5.02
3	0.01	4.86
	0.01	4.56
2	0.01	5.52
	0.02	5.04
1	0.01	4.77
	0.02	4.72

$x_c^2 = \sum \frac{(O_i - E_i)^2}{E_i}$	49.37032515

Pada Tabel 6.7. dilakukan analisis sebagai berikut, *runtime* yang dihasilkan dari pencarian titik oleh algoritme VCKNN lebih rendah dibandingkan dengan algoritme Dijkstra. Secara keseluruhan terjadi penurunan *runtime* pada penggunaan algoritme VCKNN. Pada penggunaan algoritme Dijkstra *runtime* yang dibutuhkan sekitar 4-5 detik untuk pembagian segmentasi sampai dengan 5NN sedangkan pada algoritme VCKNN *runtime* yang dibutuhkan sekitar 0,1-0,2 detik dengan pembagian segmentasi yang sama.

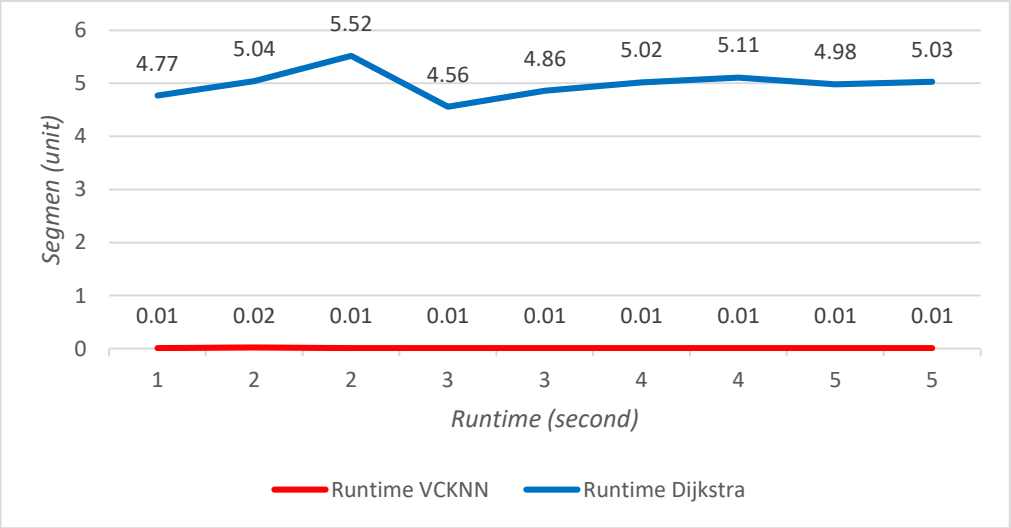
Pada Tabel 6.7. dilakukan uji statistik menggunakan *chi-square* untuk mengetahui performa dari algoritme VCKNN dan algoritme Dijkstra mengenai *runtime* yang dibutuhkan dalam pencarian. Nilai dari H_0 merupakan asumsi bahwa algoritme Dijkstra menghasilkan *runtime* yang lebih baik dibanding algoritme VCKNN. Nilai dari H_1 merupakan asumsi bahwa algoritme VCKNN lebih baik dari algoritme Dijkstra dalam menghasilkan *runtime.* Berikut ini merupakan hasil dari uji statistik dari hipotesis di atas :

α = 0,05 H_0 = 16.918978

df = 9 H_1 = 49.37032515

$H_1 < H_0$

- H_0, algortime Dijkstra menghasilkan *runtime* lebih baik dibandingkan dengan algoritme VCKNN.
- H_1, algortime VCKNN menghasilkan *runtime* lebih baik dibandingkan dengan algoritme Dijkstra.



Gambar 6.26 Hasil Perbandingan *Runtime* Berdasarkan Segmen

Pada hasil uji statistik didapatkan $H_1 > H_0$ yang menyatakan bahwa hipotesis awal dibantah dan hipotesis akhir diterima karena performa dari algoritme VCKNN pada aspek *runtime* lebih baik dibandingkan dengan algoritme Dijkstra. Dari hasil tersebut didapatkan kesimpulan bahwa pada penelitian ini algoritme VCKNN menghasilkan *runtime* yang lebih baik dibandingkan algoritme Dijkstra seperti yang ditunjukan Gambar 6.26.

Pada Tabel 6.8. algoritme Dijkstra dijadikan acuan dalam membandingkan *split nodes* dan *runtime* untuk menemukan titik tujuan, sedangkan *split nodes* dan *runtime* pada algoritme VCKNN dijadikan hasil observasi. Berikut merupakan hasil perbandingan *split nodes* terhadap pencarian berdasarkan *runtime* :

Tabel 6.8 Perbandingan *Split Nodes* dan *Runtime* Berdasarkan Segmen

K	*Split Nodes* VCKNN (*Observed*)	*Split Nodes* Dijkstra (*Expected*)	*Runtime* VCKNN (*Observed*)	*Runtime* Dijkstra (*Expected*)
5	10	20	0.01	5.03
	16	25	0.01	4.98
4	9	17	0.01	5.11
	18	27	0.01	5.02
3	7	15	0.01	4.86
	10	15	0.01	4.56
2	16	17	0.01	5.52
	8	10	0.02	5.04
1	8	8	0.01	4.77
	2	2	0.02	4.72
$x_c^2 = \sum \frac{(O_i - E_i)^2}{E_i}$			70.76718789	

Pada Tabel 6.8. dilakukan analisis sebagai berikut, keseluruhan aspek seperti pembagian segmentasi, *runtime* dan jumlah *split nodes* yang dihasilkan dari pencarian titik oleh algoritme VCKNN lebih rendah dibandingkan dengan algoritme Dijkstra. Secara keseluruhan terjadi perbaikan performa pada penggunaan algoritme VCKNN. Dengan didapatkan hasil seperti pada Tabel 6.8. maka untuk penelitian ini algoritme VCKNN menjadi rekomendasi sebagai algoritme untuk mengelola *split nodes.*

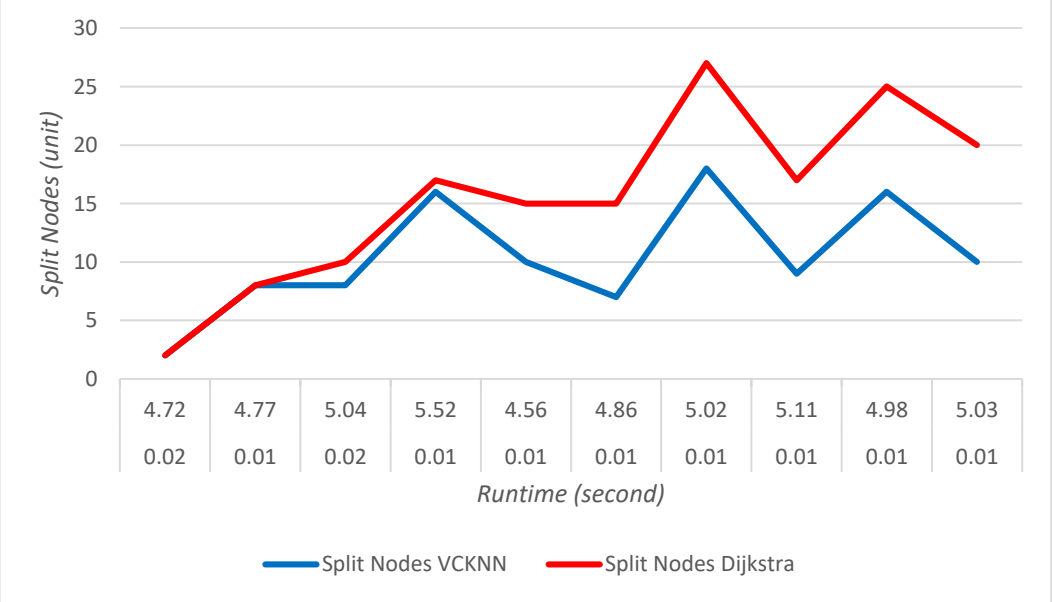
Pada Tabel 6.8. dilakukan uji statistik menggunakan *chi-square* untuk mengetahui performa keseluruhan dari algoritme VCKNN dan algoritme Dijkstra.

Nilai dari H_0 merupakan asumsi bahwa dengan menggunakan algoritme Dijkstra akan mendapatkan performa yang lebih baik dibandingkan dengan algoritme VCKNN. Nilai dari H_1 merupakan asumsi bahwa algoritme VCKNN menghasilkan performa yang lebih baik dibandingkan dengan algoritme Dijkstra. Berikut ini merupakan hasil dari uji statistik dari hipotesis di atas :

$\alpha = 0{,}05$	$H_0 = 28.869299$
$df = 18$	$H_1 = 70.76718789$
$H_1 < H_0$	

- H_0, algortime Dijkstra menghasilkan performa yang lebih baik dibandingkan dengan algoritme VCKNN.
- H_1, algortime VCKNN menghasilkan performa lebih baik dibandingkan dengan algoritme Dijkstra.

Pada hasil uji statistik didapatkan $H_1 > H_0$ yang menyatakan bahwa hipotesis awal dibantah dan hipotesis akhir diterima karena performa dari algoritme VCKNN lebih baik dibandingkan dengan algoritme Dijkstra. Dengan hipotesis ini memperkuat pernyataan bahwa algoritme VCKNN menghasilkan performa yang lebih baik dibandingkan algoritme Dijkstra seperti yang ditunjukan Gambar 6.27.



Gambar 6.27 Hasil Perbandingan *Split Nodes* dan *Runtime* Berdasarkan Segmen