ANALISIS TRADE-OFF WAKTU EKSEKUSI DAN PENGGUNAAN MEMORI ALGORITMA PRIM, KRUSKAL, DAN BORŮVKA PADA GRAF TSPLIB

Riadi Marta Dinata*
Ilmu Komputer, Universitas Lampung UNILA
Jl. Prof. Sumantri Brojonegoro No. 1, Gd. Meneng, Rajabasa, Bandar Lampung, Lampung

**Penulis korespondensi: riadimrt@gmail.com*

ABSTRAK

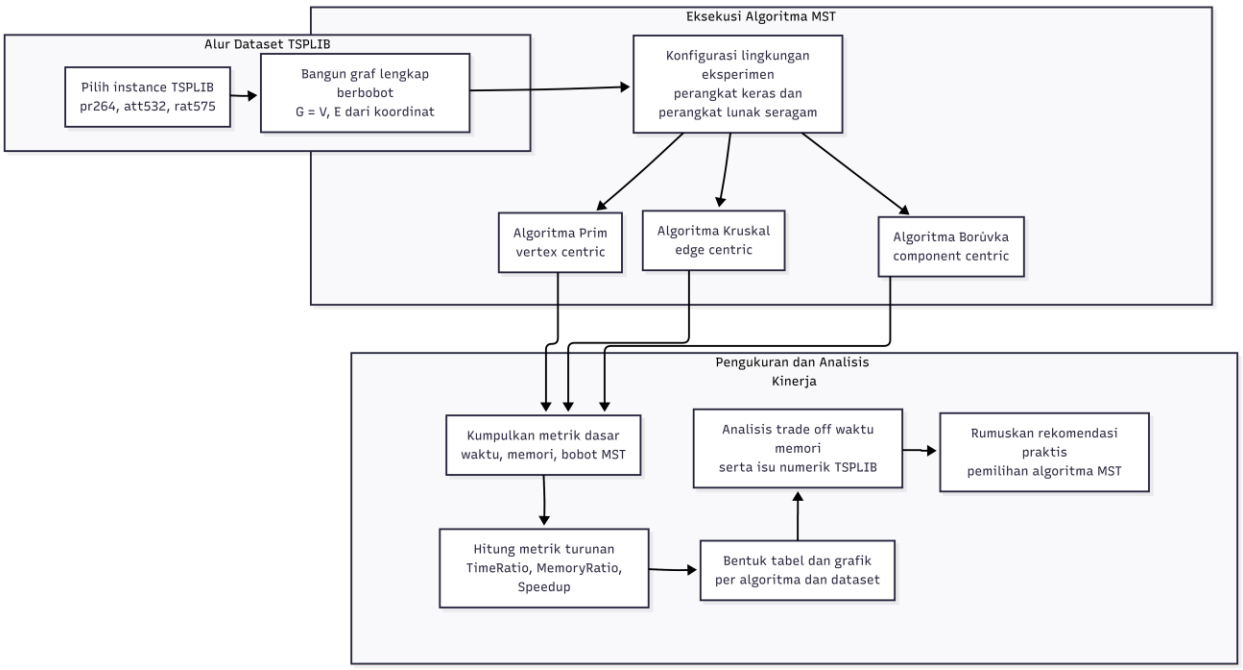
Minimum spanning tree merupakan konsep penting dalam perancangan jaringan berbiaya minimum, dengan algoritma Prim, Kruskal, dan Borůvka sebagai tiga pendekatan klasik yang paling sering digunakan dalam literatur teori graf dan optimasi jaringan. Penelitian ini memfokuskan analisis pada *trade-off* antara waktu eksekusi dan penggunaan memori ketiga algoritma tersebut pada graf lengkap berbobot yang dibangun dari tiga *instance TSPLIB*, yaitu *pr264*, *att532*, dan *rat575*, yang direpresentasikan sebagai graf Euclidean dan pseudo-Euclidean berskala menengah. Setiap *instance* dikonversi menjadi graf tak berarah berbobot dan diproses oleh algoritma Prim, Kruskal, dan Borůvka dalam satu kerangka pemrograman yang seragam, dengan pengukuran waktu eksekusi rata-rata, simpangan baku, koefisien variasi, rata-rata serta puncak penggunaan memori, dan verifikasi bobot *MST* terhadap nilai referensi. Hasil menunjukkan bahwa pada *pr264* dan *att532* seluruh algoritma menghasilkan bobot *MST* identik dan optimal, sedangkan pada *rat575* ketiganya menghasilkan bobot sedikit di atas nilai referensi dengan deviasi relatif sama, yang mengindikasikan adanya isu numerik terkait definisi jarak dan pembulatan di *TSPLIB*. Dari sisi waktu, algoritma Kruskal secara konsisten menjadi algoritma tercepat pada semua *dataset* dengan *TimeRatio* = 1, sementara Prim dan Borůvka dapat mencapai lebih dari 20 dan 40 kali lebih lambat pada *rat575*, sedangkan dari sisi memori, Borůvka selalu menjadi algoritma paling hemat memori dengan *MemoryRatio* = 1 dan Prim tercatat hingga lebih dari 50 kali lebih boros memori pada *dataset* terbesar. Temuan ini menegaskan bahwa untuk graf lengkap berbasis *TSPLIB*, algoritma Kruskal merupakan pilihan utama ketika kecepatan eksekusi menjadi prioritas, algoritma Borůvka sesuai untuk skenario dengan keterbatasan memori atau potensi paralelisasi, sedangkan algoritma Prim lebih tepat diposisikan sebagai *baseline* klasik untuk pembandingan teoretis dan eksperimen algoritma *MST*.

Kata kunci: *Minimum Spanning Tree*; Algoritma Prim; Algoritma Kruskal; Algoritma Borůvka; *TSPLIB*

1 PENDAHULUAN

Minimum *spanning tree* (*MST*) merupakan salah satu konsep fundamental dalam teori graf yang banyak dimanfaatkan pada perancangan jaringan berbiaya minimum, misalnya jaringan telekomunikasi, distribusi energi, dan berbagai sistem informasi modern yang memerlukan konektivitas efisien tanpa siklus (Cormen et al., 2022; Ruzika & Ehrgott, 2009). Dalam sebuah graf tak berarah berbobot $G = (V, E)$, *spanning tree* didefinisikan sebagai subgraf terhubung yang mencakup seluruh himpunan *vertex* V dan bebas siklus dengan jumlah *edge* $|T| = |V| - 1$, sedangkan *MST* adalah *spanning tree* yang meminimalkan total bobot *edge* $\sum_{e \in T} w(e)$ (Karger et al., 1995; Cormen et al., 2022). Secara matematis, persoalan *MST* dapat ditulis sebagai pencarian himpunan *edge* $T \subseteq E$ sehingga $\sum_{(u,v) \in T} w(u, v)$ minimum dengan kendala bahwa

graf (V, T) terhubung dan tidak mengandung siklus, sehingga banyak algoritma *MST* klasik dibangun di atas *cut property* dan *cycle property* yang menjamin optimalitas solusi (Karger et al., 1995; Nešetřil et al., 2001). Dengan dasar tersebut, *MST* menjadi komponen penting dalam berbagai algoritma optimasi jaringan dan sering dijadikan titik awal pengembangan metode lanjut di bidang ilmu komputer dan riset operasi (Moret & Shapiro, 1991; McGeoch, 2001).



Gambar 1. Diagram Alir Kerangka Penelitian Perbandingan Algoritma MST pada Graf TSPLIB

Gambar 1. menampilkan kerangka penelitian bertingkat yang dimulai dari alur pengolahan dataset *TSPLIB* menjadi graf lengkap berbobot, dilanjutkan eksekusi paralel konseptual algoritma Prim, Kruskal, dan Borůvka dalam satu lingkungan eksperimen yang seragam, kemudian pengumpulan dan pengolahan metrik dasar serta turunan hingga menghasilkan analisis *trade off* waktu memori dan rekomendasi praktis pemilihan algoritma *MST* untuk graf lengkap berbasis *TSPLIB*.

Sejak karya awal Borůvka pada tahun 1926, yang kemudian diikuti oleh kontribusi klasik Kruskal dan Prim pada dekade 1950-an, pengembangan algoritma *MST* terus berlangsung baik dari sisi teori maupun implementasi praktis (Nešetřil et al., 2001; Cormen et al., 2022). Tiga algoritma yang paling banyak dirujuk adalah algoritma Prim, algoritma Kruskal, dan algoritma Borůvka–Sollin yang merepresentasikan tiga paradigma serakah berbeda, yaitu *vertex-centric*, *edge-centric*, dan *component-centric* (Nešetřil et al., 2001; Ogunleye & Adewole, 2020). Secara umum, Prim membangun *MST* secara inkremental dari satu *vertex* awal dengan selalu memilih *edge* berbiaya minimum yang menghubungkan himpunan *vertex* yang sudah berada dalam *tree* parsial dengan *vertex* di luar himpunan tersebut, sedangkan Kruskal mengurutkan seluruh *edge* berdasarkan bobot dan menambahkannya satu per satu selama tidak membentuk siklus, sementara Borůvka menggabungkan komponen melalui pemilihan *edge* keluar termurah secara paralel pada setiap fase (Cormen et al., 2022; Nešetřil et al., 2001). Ketiganya secara teoritis menjamin tercapainya solusi *MST* optimal pada graf berbobot dengan asumsi standar, tetapi

perilaku kinerja komputasi yang dihasilkan dapat berbeda signifikan bergantung pada struktur graf dan cara implementasi (Moret & Shapiro, 1994; Cong & Sbaraglia, 2006).

Dalam konteks evaluasi empiris algoritma graf, komunitas optimasi kombinatorial telah lama mengadopsi *TSPLIB* sebagai himpunan *benchmark* terstandardisasi yang berisi sekumpulan *instance* masalah *travelling salesman problem* (*TSP*) berupa graf lengkap berbobot dengan berbagai ukuran dan tipe jarak, seperti Euclidean dan pseudo-Euclidean (Reinelt, 1991; Weise et al., 2014). Beberapa *instance* seperti *pr264*, *att532*, dan *rat575* sering digunakan sebagai dasar pengujian algoritma karena merepresentasikan graf lengkap berskala menengah dengan struktur geometris yang realistis dan terdokumentasi dengan baik (Reinelt, 1991; Akhmetov & Pak, 2023). Dengan memandang setiap *instance TSPLIB* sebagai graf tak berarah berbobot $G = (V, E)$ di mana *vertex* merepresentasikan kota dan bobot *edge* merepresentasikan jarak EUC_2D atau pseudo-Euclidean yang sudah dibulatkan, persoalan *MST* pada graf tersebut dapat diformulasikan sebagai pencarian *tree* T yang memenuhi $\sum_{(i,j)\in T} d_{ij}$ minimum, dengan d_{ij} dihitung dari koordinat geometris sesuai spesifikasi *TSPLIB* (Reinelt, 1991; Khan & Sarker, 2016). Kerangka ini memungkinkan evaluasi algoritma Prim, Kruskal, dan Borůvka pada lingkungan uji yang dapat direplikasi, sekaligus memfasilitasi perbandingan dengan studi lain yang menggunakan *dataset* yang sama (McGeoch, 2001; Cattaneo et al., 2010).

Beragam studi telah secara teoritis dan empiris mengkaji kompleksitas serta efisiensi algoritma *MST* pada berbagai jenis graf, termasuk graf besar, graf dinamis, dan graf dengan struktur khusus (Cattaneo et al., 2010; Mohapatra & Ray, 2022). Namun, banyak analisis tradisional lebih menekankan pada orde kompleksitas asimtotik, misalnya $O(E \log V)$ untuk algoritma Prim dan Kruskal pada struktur data tertentu, tanpa memberikan gambaran kuantitatif yang rinci mengenai *trade-off* waktu eksekusi dan penggunaan memori pada *benchmark* standar seperti *TSPLIB* (Moret & Shapiro, 1994; Liu & Wang, 2009). Di sisi lain, perkembangan arsitektur komputasi modern menunjukkan bahwa perbedaan pola akses memori, jumlah iterasi efektif, dan sifat *locality* dapat menimbulkan perilaku kinerja yang tidak sepenuhnya tercermin hanya dari analisis kompleksitas teoritis, sehingga evaluasi empiris yang terukur menjadi semakin penting (Cong & Sbaraglia, 2006; Qiao & Crput, 2019). Hal ini menimbulkan kebutuhan akan kajian yang tidak hanya membandingkan kecepatan algoritma *MST* pada *dataset TSPLIB*, tetapi juga mengukur secara sistematis konsumsi memori, rasio waktu relatif terhadap algoritma terbaik, dan *speedup* yang dicapai, sehingga dapat memberikan panduan praktis bagi pemilihan algoritma dalam skenario aplikasi nyata (Ogunleye & Adewole, 2020; Syahputra et al., 2020).

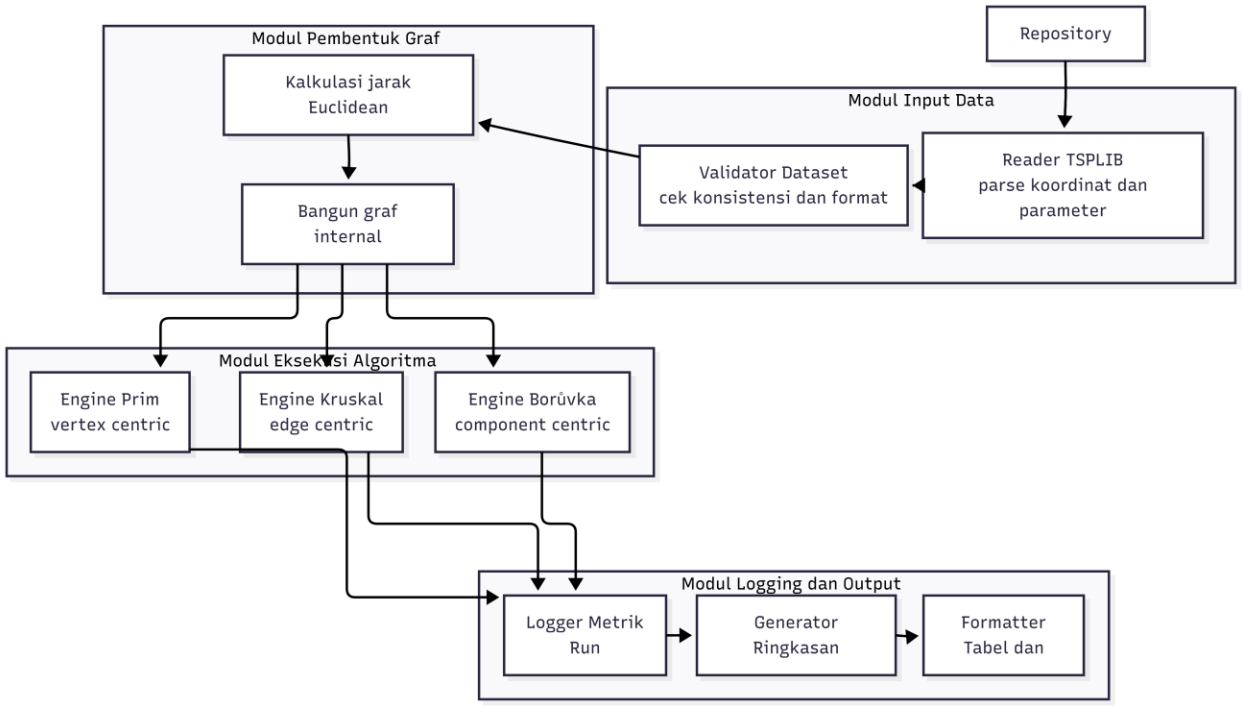
Berdasarkan latar belakang tersebut, penelitian ini memfokuskan analisis pada *trade-off* antara waktu eksekusi dan penggunaan memori algoritma Prim, Kruskal, dan Borůvka pada graf lengkap berbobot yang dibangun dari *instance TSPLIB pr264*, *att532*, dan *rat575*. Secara khusus, penelitian ini mengevaluasi metrik waktu eksekusi rata-rata, simpangan baku dan koefisien variasi, rata-rata dan puncak penggunaan memori, serta indikator turunan berupa *TimeRatio*, *MemoryRatio*, dan *Speedup* yang mengekspresikan efisiensi relatif algoritma terhadap *baseline* terbaik pada setiap *dataset*. Selain itu, penelitian ini menelaah isu numerik yang muncul pada *instance rat575*, di mana seluruh algoritma menghasilkan bobot *MST* sedikit di atas nilai referensi, serta mengkaitkannya dengan definisi jarak dan pembulatan dalam *TSPLIB* (Reinelt, 1991; Karger et al., 1995). Dengan demikian, tujuan utama artikel ini adalah menyusun karakterisasi empiris yang terfokus mengenai *trade-off* waktu–memori dari tiga algoritma *MST* klasik pada graf lengkap *TSPLIB* dan merumuskan panduan praktis pemilihan algoritma berdasarkan prioritas kecepatan komputasi dan keterbatasan memori pada skenario

jaringan berskala menengah, sebagai pelengkap dan pendalaman dari studi komparatif komprehensif yang telah ada di literatur.

2 METODE

2.1 Desain dan Pendekatan Eksperimental

Penelitian ini menggunakan pendekatan eksperimental kuantitatif untuk menganalisis *trade-off* antara waktu eksekusi dan penggunaan memori tiga algoritma *minimum spanning tree* klasik, yaitu algoritma Prim, algoritma Kruskal, dan algoritma Borůvka, pada graf lengkap berbobot yang dibangun dari beberapa *instance TSPLIB* terstandardisasi (Reinelt, 1991; Cormen et al., 2022). Setiap kombinasi algoritma dan *dataset* dieksekusi berulang kali dalam lingkungan pemrograman dan konfigurasi perangkat keras yang seragam, sehingga perbedaan kinerja yang terukur dapat dikaitkan terutama dengan karakteristik algoritmik, bukan variasi platform atau implementasi acak (Moret & Shapiro, 1994; McGeoch, 2001).



Gambar 2. Sketsa arsitektur modul implementasi algoritma MST

Gambar 2 menggambarkan arsitektur modul implementasi, dimulai dari pembacaan dan validasi berkas *TSPLIB*, pembentukan graf lengkap berbobot melalui perhitungan jarak Euclidean atau pseudo Euclidean, eksekusi algoritma Prim, Kruskal, dan Borůvka dalam modul eksekusi bersama, hingga pencatatan metrik kinerja dan pembentukan tabel ringkasan yang menjadi dasar analisis waktu memori pada bagian hasil.

Desain eksperimen mengikuti prinsip analisis algoritma secara empiris yang menekankan replikasi, kontrol terhadap faktor luar, dan pelaporan metrik kinerja secara terstruktur, khususnya untuk algoritma graf pada *benchmark* standar (Cattaneo et al., 2010; Mohapatra & Ray, 2022). Dalam konteks ini, fokus analisis diarahkan pada dua dimensi utama, yaitu dimensi waktu eksekusi dan dimensi penggunaan memori, yang kemudian diperkaya dengan metrik turunan untuk menggambarkan *trade-off* di antara keduanya pada setiap *dataset* (Ogunleye & Adewole, 2020; Syahputra et al., 2020).

2.2 Representasi Graf dan Formulasi Matematis

Graf uji dalam penelitian ini dibangun dari tiga *instance TSPLIB*, yaitu *pr264*, *att532*, dan *rat575*, yang masing-masing merepresentasikan graf lengkap Euclidean atau pseudo-Euclidean berskala menengah dengan karakteristik geometris yang berbeda (Reinelt, 1991; Weise et al., 2014). Secara matematis, setiap *instance* direpresentasikan sebagai graf tak berarah berbobot

$$G = (V, E), \tag{1}$$

di mana himpunan *vertex* didefinisikan sebagai

$$V = \{1,2, \dots, n\},$$

dan himpunan *edge* sebagai

$$E = \{(i, j) \mid i, j \in V, i \neq j\},$$

dengan fungsi bobot

$$w: E \to \mathbb{R}_{\geq 0} \tag{2}$$

yang diperoleh dari jarak geometris antar koordinat kota sesuai definisi *TSPLIB* (Reinelt, 1991; Khan & Sarker, 2016). Penekanan pada definisi ini penting karena seluruh analisis *MST* dalam penelitian ini didasarkan pada struktur graf lengkap simetris yang dibentuk oleh fungsi bobot tersebut.

Untuk *instance* Euclidean seperti *pr264* dan *rat575*, jarak antara kota i dan j dengan koordinat (x_i, y_i) dan (x_j, y_j) dihitung menggunakan metrik Euclidean terbulat EUC_2D:

$$d_{ij} = \text{round}\left(\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}\right) \tag{3}$$

Sementara itu, untuk *att532* digunakan formula pseudo-Euclidean khas *TSPLIB* yang memodifikasi skala dan pembulatan sehingga tetap konsisten dengan spesifikasi jarak bertipe *ATT* (Reinelt, 1991; Weise et al., 2014). Dengan konstruksi ini, persoalan yang diselesaikan setiap algoritma adalah mencari *spanning tree* $T \subseteq E$ yang meminimalkan jumlah bobot

$$\sum_{(i,j)\in T} w(i, j), \tag{4}$$

dengan kendala bahwa graf (V, T) terhubung dan bebas siklus, sehingga seluruh algoritma diuji dalam kerangka *MST* eksak pada *benchmark* yang terdokumentasi (Karger et al., 1995; Ruzika & Ehrgott, 2009).

2.3 Formulasi Algoritma Prim, Kruskal, dan Borůvka

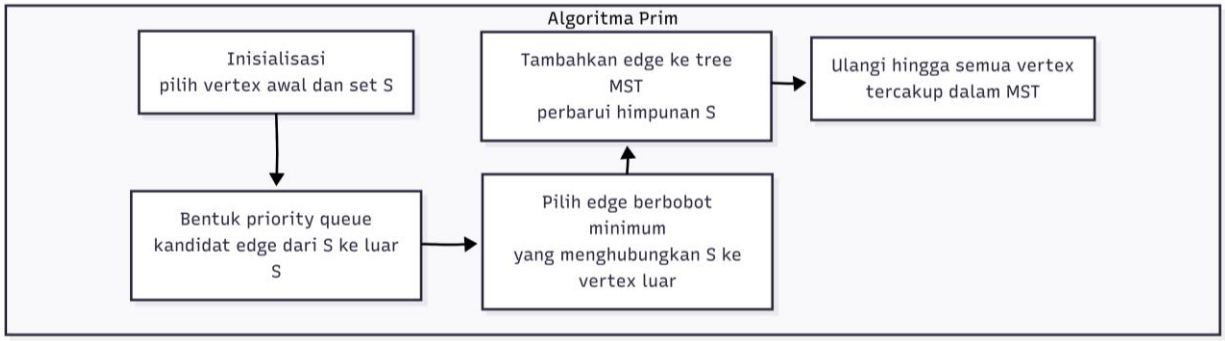
Ketiga algoritma yang dibandingkan diimplementasikan dalam satu kerangka pemrograman dengan struktur data yang sebanding, mengikuti formulasi klasik masing-masing sebagaimana dijelaskan dalam literatur analisis algoritma (Cormen et al., 2022; Nešetřil et al., 2001). Penekanan pada paradigma dan kompleksitas penting untuk menempatkan hasil empiris pada konteks teoritis.

Dalam paradigma *vertex-centric*, algoritma Prim membangun *MST* secara inkremental dengan memulai dari sebuah *vertex* awal $v_0 \in V$ dan pada setiap iterasi memilih *edge* berbiaya minimum yang menghubungkan himpunan *vertex* yang sudah tergabung dalam *tree* parsial S_k dengan *vertex* di luar himpunan tersebut (Cormen et al., 2022).

Jika S_k menyatakan himpunan *vertex* setelah iterasi ke-k, maka *edge* yang dipilih pada iterasi berikutnya adalah

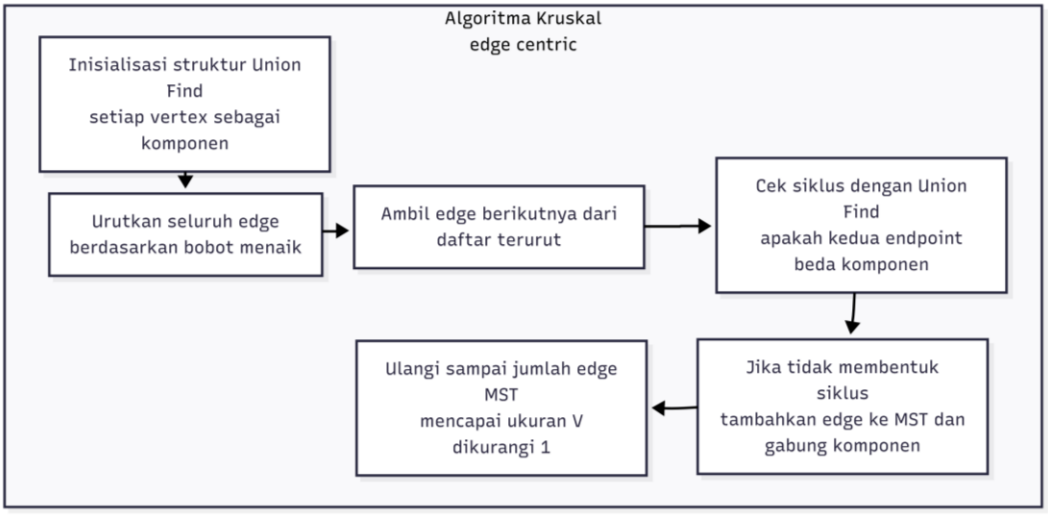
$$e_k = \arg\min\{\, w(u,v) \mid (u,v) \in E, u \in S_k, v \notin S_k \,\}, \quad (5)$$

dan himpunan *vertex* diperbarui menjadi $S_{k+1} = S_k \cup \{v\}$ sampai seluruh *vertex* tercakup (Cormen et al., 2022). Dengan pemanfaatan *priority queue*, Prim dapat diimplementasikan dengan kompleksitas waktu asimtotik sekitar $O(E \log V)$ pada graf umum, yang pada graf lengkap mendekati orde $O(V^2)$, sehingga relevan untuk dibandingkan pada graf *TSPLIB* berskala menengah (Moret & Shapiro, 1994; Cong & Sbaraglia, 2006).



Gambar 3. Diagram Alir Algoritma Prim

Gambar 3 menggambarkan proses serakah Prim yang berpusat pada himpunan *vertex* dengan memulai dari satu *vertex* awal, membentuk himpunan S, lalu secara iteratif memilih *edge* dengan bobot minimum yang menghubungkan S dengan *vertex* di luar S melalui struktur *priority queue*. Setiap *edge* terpilih ditambahkan ke *tree* dan S diperluas hingga seluruh *vertex* tercakup, sehingga *MST* dibangun secara inkremental dari sisi "frontier" pohon parsial dalam paradigma *vertex centric*.



Gambar 4. Diagram Alir Algoritma Kruskal

Diagram alir algoritma Kruskal pada Gambar 4. menekankan paradigma *edge centric* di mana semua *edge* terlebih dahulu diurutkan berdasarkan bobot menaik, kemudian diproses satu

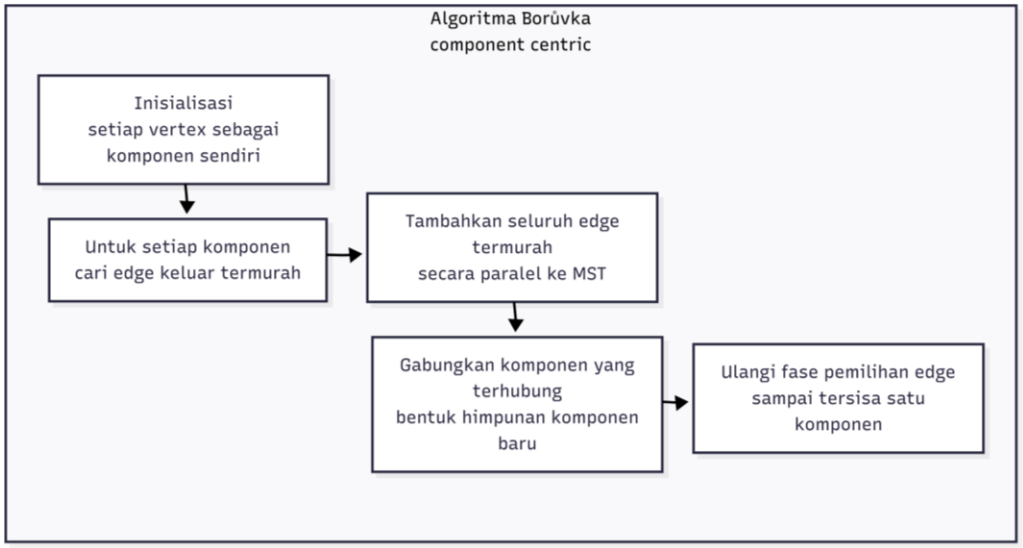
per satu dengan bantuan struktur *Union Find* untuk mendeteksi siklus. Hanya *edge* yang menghubungkan dua komponen berbeda yang ditambahkan ke *MST*, dan proses berlanjut hingga jumlah *edge* dalam *MST* mencapai $|V|-1$, sehingga pohon terbentuk dengan memprioritaskan *edge* termurah secara global sambil menjaga bebas siklus.

Algoritma Kruskal mengadopsi paradigma *edge-centric* dengan terlebih dahulu mengurutkan seluruh *edge* dalam E berdasarkan bobot meningkat, misalnya menghasilkan urutan $(e_1, e_2, \ldots, e_m)$ dengan $w(e_1) \leq w(e_2) \leq \cdots \leq w(e_m)$ (Cormen et al., 2022). Algoritma kemudian memproses *edge* secara berurutan dan menambahkan e_k ke himpunan solusi T hanya jika *edge* tersebut menghubungkan dua komponen berbeda—yang diuji menggunakan struktur *Union–Find*—hingga diperoleh $|T| = |V| - 1$ *edge* (Karger et al., 1995). Kompleksitas waktu total didominasi oleh proses pengurutan *edge* dengan orde $O(E \log E)$, sementara operasi *Union–Find* dapat dianggap hampir konstan, sehingga Kruskal sering dilaporkan sangat kompetitif terutama pada graf lengkap dan graf dengan kepadatan menengah (Moret & Shapiro, 1994; Liu & Wang, 2009).

Algoritma Borůvka–Sollin menggunakan paradigma *component-centric* dengan memulai dari keadaan di mana setiap *vertex* membentuk komponen tersendiri, kemudian pada setiap fase setiap komponen $C \subseteq V$ memilih satu *edge* keluar dengan bobot minimum untuk bergabung dengan komponen lain (Nešetřil et al., 2001; Syahputra et al., 2020). Secara formal, jika $\mathcal{C}_k$ adalah himpunan komponen pada fase ke-k, maka untuk setiap komponen $C \in \mathcal{C}_k$ dipilih *edge*

$$e_C = \arg\min\{\, w(u,v) \mid (u,v) \in E, u \in C, v \notin C \,\}, \tag{6}$$

dan seluruh *edge* $\{e_C \mid C \in \mathcal{C}_k\}$ ditambahkan secara paralel untuk membentuk himpunan komponen baru $\mathcal{C}_{k+1}$ hingga hanya tersisa satu komponen yang mencakup seluruh *vertex* (Nešetřil et al., 2001; Cormen et al., 2022). Analisis menunjukkan bahwa jumlah fase dibatasi oleh $O(\log V)$ dan setiap fase dapat diimplementasikan sehingga total kompleksitas waktu berada pada orde $O(E \log V)$, dengan keuntungan tambahan bahwa struktur pemilihan *edge* minimum per komponen sangat sesuai untuk paralelisasi pada arsitektur multikore dan *GPU* (Qiao & Crput, 2019; Sousa et al., 2015).



Gambar 5. Diagram Alir Algoritma Boruvka

Diagram alir algoritma Borůvka pada Gambar 5. menggambarkan pendekatan *component centric* yang memulai dari kondisi setiap *vertex* sebagai komponen terpisah, lalu pada setiap fase setiap komponen memilih satu *edge* keluar berbiaya minimum untuk bergabung dengan komponen lain. Seluruh *edge* minimum ini ditambahkan secara paralel ke *MST* sehingga komponen-komponen kecil bergabung menjadi komponen lebih besar, dan proses diulang hingga tersisa satu komponen yang mencakup seluruh *vertex*, sehingga *MST* diperoleh melalui serangkaian fase penggabungan komponen berbasis *edge* termurah.

2.4 Protokol Eksperimen dan Metrik Evaluasi

Pengukuran kinerja dilakukan dengan memfokuskan pada dua dimensi utama, yaitu waktu eksekusi dan penggunaan memori, serta beberapa metrik turunan yang menggambarkan *trade-off* keduanya (McGeoch, 2001; Cattaneo et al., 2010). Untuk setiap kombinasi algoritma dan *dataset*, eksperimen diulang sebanyak r kali sehingga diperoleh sampel waktu eksekusi

$$\{t_1, t_2, \dots, t_r\}.$$

Dari sampel ini dihitung:

- Waktu eksekusi rata-rata

$$\bar{t} = \frac{1}{r}\sum_{k=1}^{r} t_k, \tag{7}$$

yang merepresentasikan waktu khas algoritma pada *dataset* tertentu;

- Simpangan baku waktu σ_t yang mengukur variasi absolut;
- Koefisien variasi

$$\mathrm{CV} = \frac{\sigma_t}{\bar{t}} \times 100\%, \tag{8}$$

yang mengekspresikan stabilitas relatif waktu eksekusi terhadap nilai rata-rata (McGeoch, 2001; Cattaneo et al., 2010).

Selain itu, dicatat pula nilai minimum dan maksimum waktu eksekusi untuk mengidentifikasi potensi *outlier* serta perilaku eksekusi terbaik dan terburuk pada setiap kombinasi algoritma–*dataset*. Pada dimensi memori, sistem pemantauan merekam rata-rata penggunaan memori utama $\bar{M}$ dalam kilobyte dan nilai puncak $M_{\max}$ dalam megabyte selama eksekusi, sehingga untuk setiap algoritma diperoleh pasangan $(\bar{M}, M_{\max})$ yang menggambarkan beban memori khas dan kondisi terberat (Qiao & Crput, 2019; Sousa et al., 2015).

Metrik turunan *TimeRatio* dan *MemoryRatio* dihitung dengan menormalisasi masing-masing $\bar{t}$ dan $\bar{M}$ terhadap algoritma tercepat dan algoritma paling hemat memori pada *dataset* yang sama, sedangkan *Speedup* didefinisikan sebagai kebalikan *TimeRatio* relatif terhadap algoritma acuan (Ogunleye & Adewole, 2020; Syahputra et al., 2020). Secara skematis, jika $\bar{t}_{\min}$ adalah waktu rata-rata terkecil di antara tiga algoritma pada suatu *dataset*, maka

$$\mathrm{TimeRatio} = \frac{\bar{t}}{\bar{t}_{\min}}, \mathrm{Speedup} = \frac{\bar{t}_{\min}}{\bar{t}}, \tag{9}$$

sedangkan jika $\bar{M}_{\min}$ adalah penggunaan memori rata-rata terkecil, maka

$$\text{Memory Ratio} = \frac{\bar{M}}{\bar{M}_{\min}}. \tag{10}$$

Penekanan pada rasio ini memudahkan interpretasi, karena nilai $\text{TimeRatio} > 1$ menunjukkan algoritma lebih lambat dibanding algoritma tercepat, sedangkan $\text{MemoryRatio} > 1$ menunjukkan algoritma lebih boros memori dibanding algoritma paling efisien.

Sebagai tambahan, protokol eksperimen juga memverifikasi kualitas solusi dengan cara membandingkan bobot *MST* yang dihasilkan setiap algoritma, MSTWeight, dengan nilai referensi RefWeight yang dilaporkan untuk masing-masing *instance TSPLIB* atau dengan hasil *MST* algoritma lain pada konfigurasi yang sama (Reinelt, 1991; Karger et al., 1995). Deviasi relatif dihitung dengan rumus

$$\text{Deviation} = \frac{\text{MSTWeight} - \text{RefWeight}}{\text{RefWeight}} \times 100\%, \qquad (11)$$

sehingga dapat dipastikan bahwa perbedaan kinerja waktu dan memori tidak disertai perbedaan kualitas solusi kecuali pada kasus khusus seperti *rat575* yang menunjukkan deviasi kecil seragam akibat isu numerik (Ruzika & Ehrgott, 2009; Reinelt, 1991). Seluruh metrik yang dihitung kemudian dirangkum dalam beberapa tabel komparatif lintas algoritma dan *dataset*, yang menjadi dasar untuk menyusun analisis *trade-off* waktu–memori serta rekomendasi pemilihan algoritma pada bagian hasil dan pembahasan.

3 HASIL DAN PEMBAHASAN

3.1 Hasil

Bagian ini menyajikan hasil eksperimen utama berupa bobot *minimum spanning tree*, waktu eksekusi, penggunaan memori, serta metrik turunan *TimeRatio*, *MemoryRatio*, dan *Speedup* untuk algoritma Prim, Kruskal, dan Borůvka pada tiga *instance TSPLIB* yang diuji. Seluruh algoritma menghasilkan bobot *MST* yang identik pada *pr264* dan *att532* serta deviasi kecil seragam pada *rat575*, sehingga perbedaan kinerja dapat dipusatkan pada dimensi waktu–memori dan karakteristik iterasi algoritma.

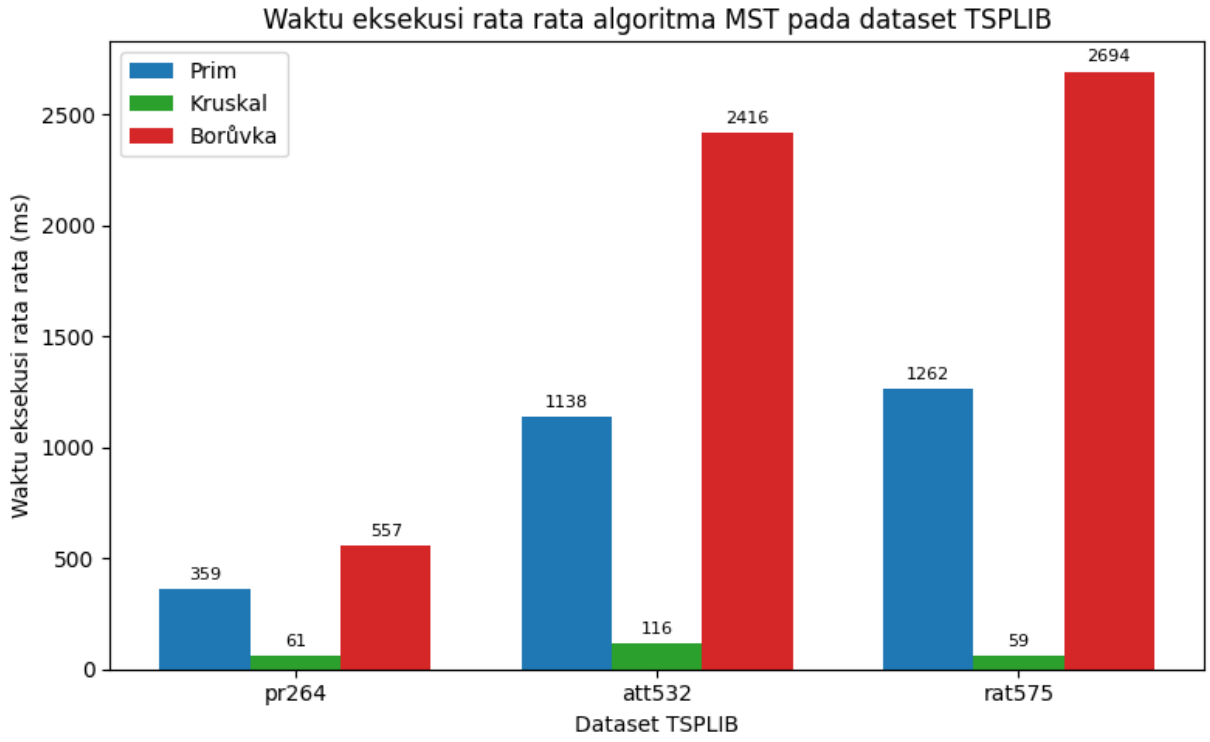
Tabel 1 menyajikan ringkasan hasil faktual yang diperoleh dari *classic_trio_consolidated.csv* dan *statistical_summary_classic_trio.csv* untuk ketiga *dataset* yang diuji. Nilai rata-rata waktu eksekusi menunjukkan bahwa algoritma Kruskal secara konsisten memiliki waktu eksekusi rata-rata paling kecil pada seluruh *dataset*, sedangkan algoritma Borůvka selalu memiliki waktu eksekusi rata-rata terbesar. Pengukuran memori memperlihatkan pola berlawanan, di mana Borůvka menjadi algoritma paling hemat memori, sementara Prim menggunakan memori rata-rata terbesar pada semua *dataset* yang diuji.

Tabel 1. Ringkasan hasil waktu eksekusi dan penggunaan memori algoritma MST pada *TSPLIB*

Dataset	Algoritma	Mean Time (ms)	CV (%)	Mean Memory (KB)	Peak Memory (MB)	TimeRatio vs tercepat	MemoryRatio vs paling hemat
pr264	Prim	358,88	15,28	1906,26	1,86	5,86	11,72
pr264	Kruskal	61,19	4,48	813,19	0,79	1,00	5,00
pr264	Borůvka	557,42	8,75	162,64	0,16	9,11	1,00
att532	Prim	1138,06	6,28	9498,89	9,28	9,80	46,47

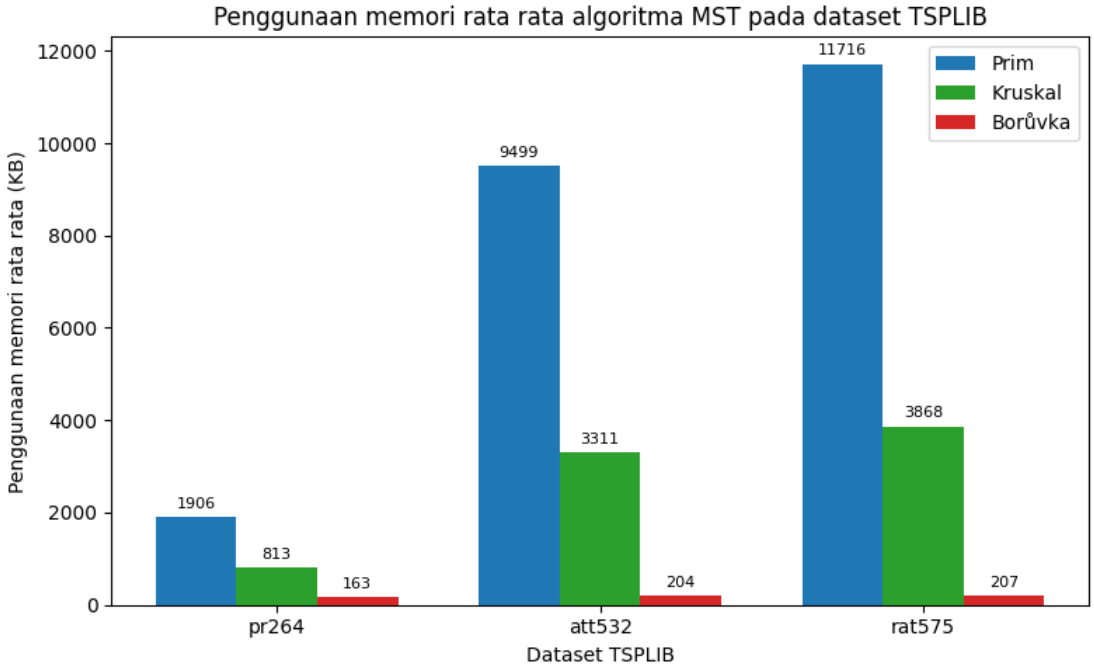
Dataset	Algoritma	Mean Time (ms)	CV (%)	Mean Memory (KB)	Peak Memory (MB)	TimeRatio vs tercepat	MemoryRatio vs paling hemat
att532	Kruskal	116,16	29,89	3310,69	3,23	1,00	16,20
att532	Borůvka	2416,32	3,73	204,41	0,20	20,80	1,00
rat575	Prim	1261,91	0,74	11716,25	11,44	21,26	56,61
rat575	Kruskal	59,35	0,71	3867,80	3,78	1,00	18,69
rat575	Borůvka	2694,13	1,50	206,97	0,20	45,40	1,00

Tabel 1 menunjukkan bahwa untuk *pr264* dan *att532*, nilai *TimeRatio* Prim dan Borůvka jauh lebih besar daripada 1, yang berarti keduanya lebih lambat secara signifikan dibanding Kruskal sebagai algoritma tercepat, sedangkan *MemoryRatio* Prim terhadap Borůvka dapat mencapai lebih dari 40 kali pada *att532* dan lebih dari 50 kali pada *rat575*. Pola ini menegaskan adanya *trade-off* yang tajam antara waktu dan memori: Kruskal menempati posisi tengah dengan waktu sangat cepat dan penggunaan memori sedang, Prim menjadi algoritma dengan waktu sedang namun memori paling boros, sedangkan Borůvka sangat lambat tetapi paling hemat memori pada semua *dataset* yang diuji.



Gambar 6. Grafik Waktu Eksekusi Rata-Rata Algoritma MST pada Tiga Dataset TSPLIB

Gambar 6. menggambarkan perbandingan waktu eksekusi rata rata algoritma Prim, Kruskal, dan Borůvka pada tiga dataset *TSPLIB*, di mana batang Kruskal selalu paling rendah yang menandakan waktu tercepat, sementara batang Borůvka selalu tertinggi sehingga tampak sebagai algoritma paling lambat, dan batang Prim konsisten berada di posisi tengah untuk setiap dataset yang menguatkan pola bahwa Kruskal dominan dalam kecepatan sedangkan Borůvka memiliki biaya waktu sangat besar.



Gambar 7. Penggunaan Memori Rata-Rata Algoritma MST pada Tiga Dataset TSPLIB

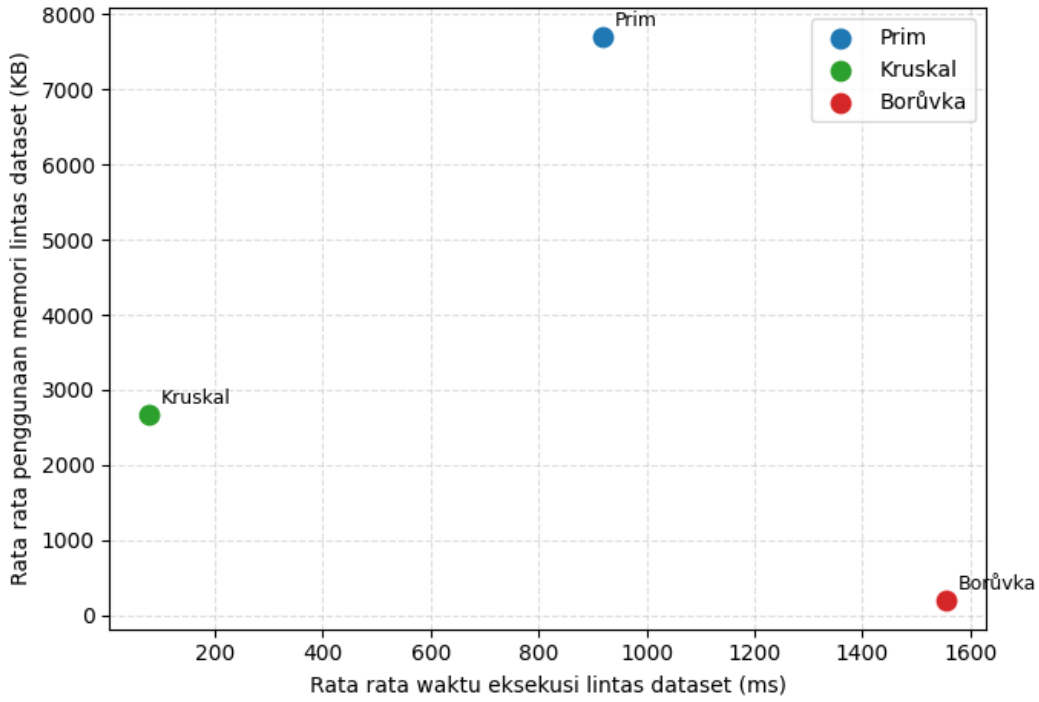
Gambar 7 yang dihasilkan akan memperlihatkan bahwa untuk setiap dataset batang Prim selalu tertinggi, Borůvka selalu terendah, dan Kruskal berada di tengah, sehingga pola *trade-off* memori antar algoritma terlihat jelas.

Dari sisi kualitas solusi, seluruh algoritma menghasilkan bobot *MST* yang sama dan optimal pada *pr264* dan *att532*, yaitu 41.142 dan 75.872, dengan deviasi nol terhadap nilai referensi yang digunakan, sedangkan pada *rat575* seluruh algoritma menghasilkan bobot 6.248 dengan deviasi sekitar 0,032 persen di atas nilai referensi 6.246. Fakta bahwa ketiga algoritma memberikan bobot *MST* identik pada setiap *dataset* yang sama menunjukkan bahwa perbedaan performa hanya muncul pada aspek efisiensi komputasi, bukan pada kualitas solusi, sementara deviasi seragam pada *rat575* mengindikasikan isu numerik sistematis pada definisi jarak dan pembulatan, bukan kesalahan salah satu algoritma secara terpisah.

Jika dilihat dari jumlah iterasi yang tercatat, algoritma Borůvka memerlukan jumlah iterasi jauh lebih besar dibanding dua algoritma lainnya, dengan rata-rata ratusan ribu iterasi pada *pr264* dan *att532* serta lebih dari 800.000 iterasi pada *rat575*, sedangkan Prim dan Kruskal berada pada kisaran ribuan hingga belasan ribu iterasi. Perbedaan jumlah iterasi ini konsisten dengan pola waktu eksekusi, di mana banyaknya fase dan pemrosesan *edge* berulang pada Borůvka menyebabkan waktu eksekusi yang jauh lebih besar, meskipun penggunaan memori rata-rata dan puncak tetap sangat rendah.

Sebagai ikhtisar lintas *dataset*, tabel *performance_ranking_classic_trio.csv* memberikan peringkat waktu dan memori yang mengonfirmasi bahwa Kruskal selalu menempati peringkat

pertama dalam waktu eksekusi dan peringkat kedua dalam memori, Prim selalu berada di peringkat kedua untuk waktu dan ketiga untuk memori, sedangkan Borůvka selalu menjadi yang paling lambat (peringkat ketiga waktu) tetapi paling hemat memori (peringkat pertama memori). Rangkuman ini memperkuat temuan bahwa Kruskal merupakan kandidat utama ketika kecepatan menjadi prioritas, Borůvka relevan untuk skenario yang sangat sensitif memori atau berpotensi paralel, dan Prim cenderung berfungsi sebagai *baseline* klasik yang tidak dominan pada kedua dimensi efisiensi untuk graf lengkap *TSPLIB* berskala menengah.



Gambar 8. Trade Off Global Waktu Memori Algoritma MST pada Tiga Dataset TSPLIB

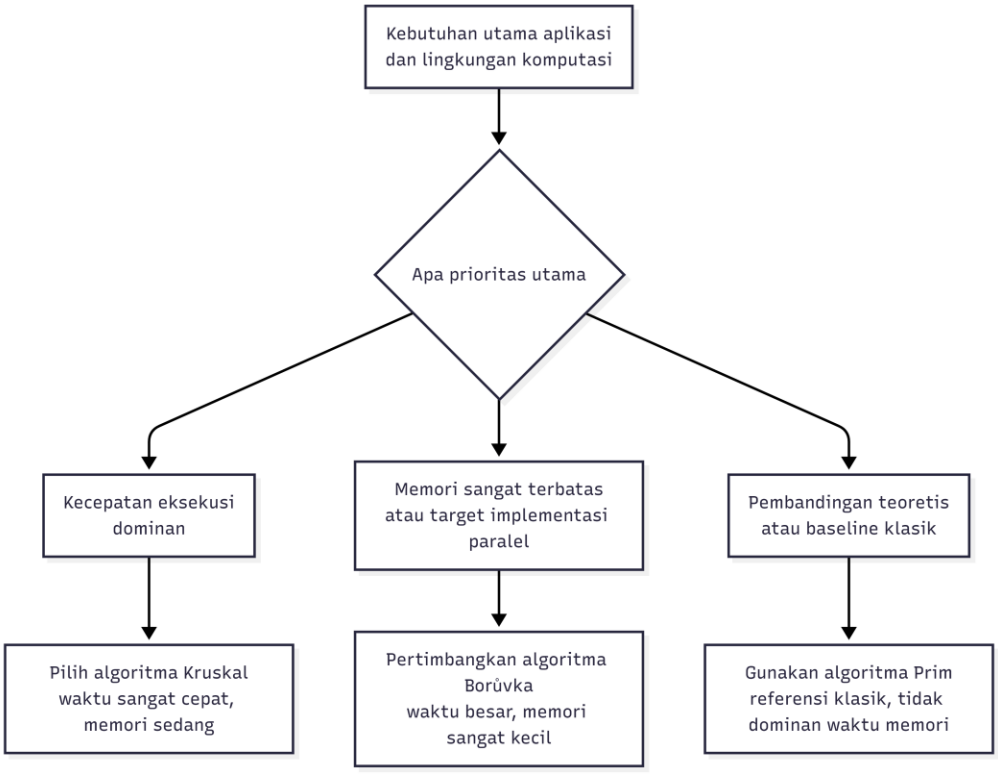
Plot *trade off* global waktu–memori pada Gambar 8. menempatkan setiap algoritma sebagai satu titik pada bidang dua dimensi dengan sumbu horizontal menyatakan rata rata waktu eksekusi lintas tiga dataset dan sumbu vertikal menyatakan rata rata penggunaan memori lintas tiga dataset, sehingga pola efisiensi relatif langsung terlihat. Titik Kruskal berada di area sangat kiri dan sekitar tengah vertikal yang berarti waktu eksekusinya jauh lebih kecil dibanding dua algoritma lain dengan penggunaan memori sedang, sementara titik Borůvka muncul jauh di kanan bawah yang menunjukkan waktu eksekusi sangat besar tetapi memori sangat kecil dan menjadi algoritma paling hemat memori. Sebaliknya, titik Prim berada jauh di kanan atas yang mengindikasikan bahwa algoritma ini sekaligus memiliki waktu rata rata besar dan konsumsi memori tinggi, sehingga secara praktis tidak dominan pada kedua dimensi efisiensi bila dibandingkan dengan Kruskal dan Borůvka.

3.2 Diskusi dan Pembahasan

Hasil eksperimen menunjukkan pola yang sangat konsisten bahwa algoritma Kruskal selalu menjadi algoritma tercepat pada ketiga *dataset TSPLIB* yang diuji, dengan *TimeRatio* sama dengan 1 sebagai acuan, sedangkan Prim dan Borůvka memiliki *TimeRatio* jauh di atas 1, terutama pada *att532* dan *rat575* yang berskala lebih besar. Pola ini selaras dengan analisis kompleksitas

waktu $O(E\log E)$ untuk Kruskal dan $O(E\log V)$ untuk Prim serta Borůvka pada graf umum, tetapi hasil empiris memperlihatkan bahwa pada graf lengkap Euclidean berbobot seperti *TSPLIB*, biaya pengurutan *edge* satu kali di awal dan efisiensi struktur *Union–Find* menjadikan Kruskal sangat kompetitif sehingga keunggulan praktisnya lebih menonjol daripada yang tersirat hanya dari orde kompleksitas asimtotik.

Sebaliknya, algoritma Borůvka yang secara teoritis juga berada pada orde $O(E\log V)$ menunjukkan waktu eksekusi yang jauh lebih besar dibanding dua algoritma lainnya, yang dapat dijelaskan oleh jumlah fase dan iterasi sangat tinggi yang tercatat pada seluruh *dataset*, misalnya mencapai ratusan ribu hingga lebih dari 800.000 iterasi. Fakta bahwa setiap fase Borůvka mensyaratkan pemilihan *edge* minimum keluar untuk setiap komponen, yang pada graf lengkap mengakibatkan penyapuan himpunan *edge* berkali-kali, menyebabkan overhead besar pada lingkungan eksekusi sekuensial sehingga potensi paralelisasi teoritis belum terwujud dalam pengujian ini.



Gambar 9. Diagram Ringkasan Rekomendasi Pemilihan Algoritma MST

Gambar 9. merangkum rekomendasi praktis pemilihan algoritma *MST* dengan menanyakan prioritas utama terlebih dahulu: jika kecepatan eksekusi yang paling penting maka disarankan memilih Kruskal, jika kendala utama adalah memori sangat terbatas atau ada rencana implementasi paralel maka Borůvka menjadi kandidat utama, sedangkan jika tujuan utamanya adalah pembandingan teoretis atau penggunaan sebagai *baseline* klasik maka Prim digunakan sebagai algoritma rujukan standar.

Dari sisi memori, hasil menunjukkan *trade-off* yang tajam: algoritma Prim secara konsisten menjadi algoritma paling boros memori, dengan *MemoryRatio* terhadap Borůvka dapat mencapai lebih dari 40 kali pada *att532* dan lebih dari 50 kali pada *rat575*, sedangkan Kruskal menempati posisi tengah dengan memori sedang dan Borůvka menjadi yang paling hemat memori. Pola ini

sejalan dengan karakter struktur data yang digunakan, di mana Prim memelihara *priority queue* yang memuat informasi frontier *edge* untuk setiap *vertex*, sedangkan Borůvka hanya memerlukan struktur komponen dan *edge* minimum per komponen sehingga jejak memori yang tersimpan dapat dijaga tetap kecil meskipun jumlah iterasi besar.

Kombinasi hasil waktu dan memori tersebut menghasilkan profil *trade-off* yang jelas: Kruskal menempati posisi paling seimbang dengan waktu sangat cepat dan memori sedang, Borůvka mengorbankan waktu secara ekstrem untuk memperoleh efisiensi memori maksimum, sedangkan Prim cenderung tidak dominan karena membutuhkan waktu lebih lama daripada Kruskal dan memori lebih besar daripada keduanya. Dengan demikian, untuk graf lengkap Euclidean berskala menengah seperti *pr264*, *att532*, dan *rat575*, Kruskal dapat direkomendasikan sebagai pilihan utama ketika prioritas utama adalah waktu eksekusi, Borůvka menjadi kandidat relevan untuk skenario yang sangat sensitif terhadap memori atau ketika implementasi paralel tersedia, dan Prim lebih tepat diposisikan sebagai *baseline* klasik untuk keperluan pembandingan teoretis dan eksperimen algoritma *MST* lainnya.

Dari perspektif kualitas solusi, kenyataan bahwa ketiga algoritma menghasilkan bobot *MST* identik pada setiap *dataset* yang sama, serta deviasi nol terhadap nilai referensi pada *pr264* dan *att532*, mengonfirmasi bahwa perbedaan perilaku kinerja yang diamati murni terkait efisiensi komputasi dan bukan akibat perbedaan kualitas solusi. Kasus khusus *rat575*, di mana semua algoritma menghasilkan bobot 6.248 dengan deviasi sekitar 0,032 persen di atas nilai referensi 6.246, menunjukkan adanya isu numerik sistematis terkait definisi jarak Euclidean terbulat dan pembulatan dalam *TSPLIB*, yang perlu diperhatikan ketika menggunakan *dataset* ini untuk evaluasi algoritma sensitif terhadap nilai bobot.

Secara keseluruhan, hasil penelitian ini memperkaya kajian komparatif algoritma *MST* dengan memberikan kuantisasi eksplisit mengenai *TimeRatio*, *MemoryRatio*, dan *Speedup* pada tiga *instance TSPLIB* terpilih, sehingga memberikan dasar numerik yang jelas untuk menyusun panduan praktis pemilihan algoritma dalam skenario jaringan berskala menengah. Dengan memosisikan Kruskal sebagai algoritma umum tercepat, Borůvka sebagai algoritma paling hemat memori, dan Prim sebagai *baseline* teoretis yang stabil tetapi tidak dominan, hasil ini mendukung dan memperjelas rekomendasi yang telah muncul dalam beberapa studi sebelumnya, sekaligus menekankan pentingnya analisis empiris terukur di luar sekadar kompleksitas asimtotik ketika algoritma dijalankan pada *benchmark* terstandardisasi seperti *TSPLIB*.

Tabel 2. Ringkasan *trade-off* waktu–memori dan posisi algoritma MST pada *TSPLIB*

Algoritma	Ringkasan waktu–memori dan implikasi praktis
pr264 Prim	– *TimeRatio* ≈ 5,86 dan *MemoryRatio* ≈ 11,72; posisi waktu menengah namun menjadi yang paling boros memori. Cocok hanya bila memori cukup besar, karena untuk graf lengkap berukuran menengah kombinasi waktu dan memori kurang efisien dibanding algoritma lain.
pr264 Kruskal	– *TimeRatio* = 1,00 (tercepat) dan *MemoryRatio* ≈ 5,00 (sedang). Menjadi pilihan utama ketika kecepatan menjadi prioritas, dengan konsumsi memori yang masih berada dalam kisaran wajar untuk penggunaan praktis.
pr264 Borůvka	– *TimeRatio* ≈ 9,11 (terlambat) dan *MemoryRatio* = 1,00 (paling hemat). Sangat efisien dari sisi memori, tetapi waktu eksekusi jauh lebih besar; relevan terutama pada lingkungan dengan kendala memori ketat.
att532 Prim	– *TimeRatio* ≈ 9,80 dan *MemoryRatio* ≈ 46,47; berada di posisi menengah untuk waktu tetapi paling boros memori. Pada graf yang lebih besar, Prim menjadi sangat tidak efisien karena jauh lebih lambat daripada Kruskal dan mengonsumsi memori yang sangat besar.

Algoritma	Ringkasan waktu–memori dan implikasi praktis
att532 Kruskal	– *TimeRatio* = 1,00 (tercepat) dan *MemoryRatio* ≈ 16,20 (sedang). Tetap menjadi algoritma tercepat pada skala menengah–besar dengan penggunaan memori yang masih dapat diterima, sehingga praktis sebagai default.
att532 Borůvka	– *TimeRatio* ≈ 20,80 (terlambat) dan *MemoryRatio* = 1,00 (paling hemat). Memberikan penghematan memori signifikan dengan pengorbanan waktu eksekusi yang sangat besar; masuk akal hanya jika memori benar-benar menjadi kendala utama.
rat575 Prim	– *TimeRatio* ≈ 21,26 dan *MemoryRatio* ≈ 56,61; menempati posisi menengah dalam waktu tetapi menjadi yang paling boros memori. Pada graf terbesar, kombinasi waktu dan memori Prim merupakan yang paling tidak efisien di antara ketiga algoritma.
rat575 Kruskal	– *TimeRatio* = 1,00 (tercepat) dan *MemoryRatio* ≈ 18,69 (sedang). Tetap menjadi algoritma paling cepat; memori lebih besar dari Borůvka tetapi masih jauh lebih kecil dibanding Prim, sehingga seimbang untuk aplikasi praktis.
rat575 Borůvka	– *TimeRatio* ≈ 45,40 (terlambat) dan *MemoryRatio* = 1,00 (paling hemat). Menawarkan efisiensi memori yang sangat tinggi dengan pengorbanan waktu eksekusi yang ekstrem, sehingga hanya menarik ketika memori sangat terbatas dan waktu bukan faktor kritis.

Tabel 2 memperjelas bahwa pada seluruh *dataset* Kruskal selalu berada di sudut "waktu kecil, memori sedang", Borůvka di sudut "waktu besar, memori kecil", sedangkan Prim cenderung berada pada sudut "waktu besar, memori besar", sehingga mendukung interpretasi *trade-off* waktu–memori yang telah dibahas pada subbab 3.2.

4 KESIMPULAN DAN SARAN

Penelitian ini menunjukkan bahwa pada graf lengkap berbobot yang dibangun dari *instance TSPLIB pr264*, *att532*, dan *rat575*, ketiga algoritma *minimum spanning tree* klasik—Prim, Kruskal, dan Borůvka—selalu menghasilkan bobot *MST* yang identik untuk setiap *dataset* yang sama, dengan deviasi nol terhadap nilai referensi pada *pr264* dan *att532* serta deviasi kecil seragam sekitar 0,032 persen pada *rat575*. Hal ini menegaskan bahwa, dalam konfigurasi eksperimen yang digunakan, perbedaan di antara ketiganya murni terletak pada aspek efisiensi komputasi waktu–memori, bukan pada kualitas solusi yang diperoleh.

Dari dimensi waktu eksekusi, algoritma Kruskal secara konsisten menjadi algoritma tercepat pada seluruh *dataset*, dengan *TimeRatio* yang selalu bernilai 1 sebagai acuan, sedangkan algoritma Prim dan Borůvka memiliki *TimeRatio* semakin besar seiring peningkatan ukuran graf, bahkan mencapai lebih dari 20 kali dan 40 kali lebih lambat dibanding Kruskal pada *rat575*. Di sisi lain, dari dimensi memori, algoritma Borůvka selalu menjadi algoritma paling hemat memori dengan *MemoryRatio* sama dengan 1, sementara Kruskal menempati posisi tengah dan Prim selalu menjadi algoritma paling boros memori, dengan *MemoryRatio* terhadap Borůvka yang dapat melampaui faktor 50 pada *dataset* terbesar.

Berdasarkan kombinasi hasil waktu dan memori, dapat disimpulkan bahwa untuk graf lengkap Euclidean berskala menengah seperti *pr264*, *att532*, dan *rat575*, algoritma Kruskal merupakan pilihan utama ketika prioritas utama adalah kecepatan eksekusi, algoritma Borůvka lebih sesuai untuk skenario yang sangat sensitif terhadap penggunaan memori atau ketika implementasi paralel memungkinkan pemanfaatan struktur *component-centric*, sedangkan algoritma Prim lebih tepat diposisikan sebagai *baseline* klasik untuk keperluan pembandingan teoretis dan eksperimen, karena pada konfigurasi ini tidak dominan baik dalam waktu maupun memori. Temuan ini sekaligus menegaskan pentingnya analisis empiris berbasis

metrik *TimeRatio*, *MemoryRatio*, dan *Speedup* di atas *benchmark* terstandardisasi seperti *TSPLIB* untuk melengkapi informasi dari kompleksitas asimtotik ketika menyusun panduan praktis pemilihan algoritma *MST* pada aplikasi jaringan berskala menengah.

DAFTAR PUSTAKA

Akhmetov, I., & Pak, A. (2023). TSP review: Performance comparison of the well-known methods on a standardized dataset. In 2023 IEEE 13th International Conference on Pattern Recognition Systems (ICPRS) (pp. 1–6). IEEE. https://doi.org/10.1109/ICPRS58416.2023.10275749

Cattaneo, G., Faruolo, P., Italiano, G. F., & Parente, D. (2010). Engineering the computing of minimum spanning trees. Journal of Experimental Algorithmics, 14, 1.3–1–1.3–30. https://doi.org/10.1145/1498698.1594233

Cong, G., & Sbaraglia, S. (2006). A study on the locality behavior of minimum spanning tree algorithms. In High performance computing (Lecture Notes in Computer Science, Vol. 4297, pp. 583–594). Springer. https://doi.org/10.1007/11945918_55

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2022). Introduction to algorithms (4th ed.). MIT Press.

de Aragão, M. P., & Werneck, R. F. (2002). On the implementation of MST-based heuristics for the Steiner problem in graphs. In D. M. Mount & C. Stein (Eds.), Algorithm engineering and experiments (pp. 1–15). Springer. https://doi.org/10.1007/3-540-45643-0_1

Karger, D. R., Klein, P. N., & Tarjan, R. E. (1995). A randomized linear-time algorithm to find minimum spanning trees. Journal of the ACM, 42(2), 321–328. https://doi.org/10.1145/201019.201022

Khan, A., & Sarker, J. (2016). An efficient minimum spanning tree algorithm. In 2016 IEEE Information Technology, Networking, Electronic and Automation Control Conference (ITNEC) (pp. 148–152). IEEE. https://doi.org/10.1109/ITNEC.2016.7543874

Liu, X., & Wang, Y. (2009). Comparison of Prim and Kruskal on Shanghai and Shenzhen 300 index hierarchical structure tree. In 2009 IEEE International Conference on Grey Systems and Intelligent Services (pp. 1–5). IEEE. https://doi.org/10.1109/GSIS.2009.5369459

McGeoch, C. C. (2001). Experimental analysis of algorithms. In P. M. Pardalos & M. G. C. Resende (Eds.), Handbook of applied optimization (pp. 614–625). Oxford University Press.

Mohapatra, C., & Ray, B. N. B. (2022). A survey on large datasets minimum spanning trees. In A. A. Sk, T. Turki, T. K. Ghosh, S. Joardar, & S. Barman (Eds.), Artificial intelligence (ISAI 2022) (Communications in Computer and Information Science, Vol. 1695, pp. 26–35). Springer. https://doi.org/10.1007/978-3-031-22485-0_3

Moret, B. M. E., & Shapiro, H. D. (1991). An empirical analysis of algorithms for constructing a minimum spanning tree. In Proceedings of the Third Annual ACM-SIAM Symposium on Discrete Algorithms (pp. 93–102). SIAM.

Moret, B. M. E., & Shapiro, H. D. (1994). Algorithms for constructing minimum spanning trees: An experimental study. Journal of Algorithms, 18(1), 26–50. https://doi.org/10.1006/jagm.1994.1003

Nešetřil, J., Milková, E., & Nešetřilová, H. (2001). Otakar Borůvka on minimum spanning tree problem: Translation of both the 1926 papers, comments, history. Discrete Mathematics, 233(1–3), 3–36. https://doi.org/10.1016/S0012-365X(00)00191-8[1]

Ogunleye, J. O., & Adewole, A. P. (2020). A comparative study of minimal spanning tree algorithms. In 2020 IEEE Nigeria 4th International Conference on Disruptive Technologies

for Sustainable Development (NIGERCON) (pp. 1–5). IEEE. https://doi.org/10.1109/NIGERCON49700.2020.9077616

Reinelt, G. (1991). TSPLIB—A traveling salesman problem library. ORSA Journal on Computing, 3(4), 376–384. https://doi.org/10.1287/ijoc.3.4.376

Ruzika, S., & Ehrgott, M. (2009). A survey on multiple objective minimum spanning tree problems. In M. Ehrgott, C. Fonseca, X. Gandibleux, J.-K. Hao, & M. Sevaux (Eds.), Evolutionary multi-criterion optimization (pp. 97–127). Springer. https://doi.org/10.1007/978-3-642-02094-0_6

Sousa, F. R., Oliveira, D., Barros, C., & Torres, L. (2015). Parallel implementations of Borůvka's minimum spanning tree algorithm for manycore GPUs. In 2015 IEEE 27th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD) (pp. 38–45). IEEE. https://doi.org/10.1109/SBAC-PAD.2015.10

Syahputra, M. R., Situmorang, M., & Marpaung, T. J. (2020). Comparative of Prim's and Borůvka's algorithm to solve minimum spanning tree problems. Journal of Physics: Conference Series, 1462, 012043. https://doi.org/10.1088/1742-6596/1462/1/012043

Weise, T., Chiong, R., Tang, K., Lässig, J., Tsutsui, S., Chen, W., Michalewicz, Z., & Yao, X. (2014). Benchmarking optimization algorithms: An open source framework for the traveling salesman problem. IEEE Computational Intelligence Magazine, 9(3), 40–52. https://doi.org/10.1109/MCI.2014.2326101

Yang, L., Zhang, D., Li, L., & He, Q. (2024). Energy efficient cluster-based routing protocol for WSN using multi-strategy fusion snake optimizer and minimum spanning tree. Scientific Reports, 14, 16815. https://doi.org/10.1038/s41598-024-66703-9