Perbandingan Algoritma Greedy dan Variannya dalam Penyelesaian Persoalan Shortest Common Superstring

Budi Satrio¹, Ivan Kurniawan², Selvira Afifa³

Laboratorium Ilmu dan Rekayasa Komputasi
Departemen Teknik Informatika, Institut Teknologi Bandung
Jl. Ganesha 10, Bandung

E-mail : if14006@students.if.itb.ac.id¹, if14007@students.if.itb.ac.id²,
if14050@students.if.itb.ac.id³

Abstrak

Shortest Common Superstring (SCS) merupakan persoalan pencarian string terpendek yang mengandung semua string dari himpunan string yang diberikan. Persoalan SCS merupakan persoalan yang sangat penting karena penggabungan DNA termasuk dalam SCS tersebut. Untuk menyelesaikan SCS ini dibutuhkan algoritma yang cepat dan mangkus sehingga orang tidak perlu menyelesaikan persoalan ini secara *brute force*. Dalam makalah ini, akan dibahas perbandingan algoritma *greedy* dan variannya dalam penyelesaian kasus SCS.

Kata kunci: *brute force, greedy, GreedySCS, TGreedySCS, shortest common superstring*

1. Pendahuluan

Shortest Common Superstring (SCS) merupakan persoalan yang penting karena salah satunya terdapat persoalan penggabungan DNA dan kompresi data didalamnya. SCS dapat diselesaikan melalui pendekatan beberapa algoritma tertentu, diantaranya adalah algoritma *greedy* dan variannya. Untuk mendapatkan solusi SCS dengan cepat dan mangkus, diperlukan suatu algoritma yang dapat memberikan hasil berkualitas tinggi daripada hasil terburuk yang dicapai secara *brute force*. Dari rata-rata eksperimen komputasional yang dilakukan oleh beberapa ilmuwan [1], algoritma *greedy* dan varian dari algoritma *greedy* yang dipakai dalam penelitian ini dapat memberikan hasil rata-rata yang hanya lebih buruk 1,4 % daripada hasil optimumnya.

2. Shortest Common Superstring

2.1 Definisi Shortest Common Superstring

Shortest Common Superstring (SCS) adalah sebuah persoalan pencarian string terpendek yang mengandung semua string dari himpunan string yang diberikan [4].

Sebagai penjelasan lebih lanjut, diberikan sebuah himpunan string $P = \{s_1, s_2, \dots, s_n\}$. String terpendek S^* dimana setiap $s_i \in P$ adalah substring dari S^* kita kenal sebagai SCS dari P.

```
A B R A C
A C A D A
A D A B R
D A B R A
R A C A D
```

Gambar 1.1 Input string

```
A B R A C A D A B R A
A B R A C
R A C A D
A C A D A
A D A B R
D A B R A
```

Gambar 1.2 Output string

Pencarian S^* ini terdapat dalam beberapa aplikasi persoalan, termasuk diantaranya adalah penggabungan DNA dan kompresi data.

2.2 Elemen Shortest Common Superstring

SCS hanyalah sebuah objek dari persoalan yang menggunakan algoritma *greedy*. Karena hanya berupa objek/entitas, elemen-elemen yang

dimilikinya pun tidak sama dengan elemen dari sebuah algoritma yang memiliki fungsi dan memiliki hasil. Elemen dari SCS hanyalah berupa bagian bagian dari objek SCS itu sendiri yang berupa himpunan string P yang memiliki anggota string yang akan dicari SCSnya ($P = \{s_1, s_2, \dots, s_n\}$), dan status kelayakan SCS, yaitu apakah SCS yang didapatkan mengandung semua substring dari himpunan string P .

3. Algoritma Greedy

3.1 Pengertian Algoritma Greedy

Algoritma *Greedy* merupakan algoritma yang membentuk solusi langkah per langkah. Pada setiap langkah tersebut akan dipilih keputusan yang paling optimal [3]. Keputusan tersebut tidak perlu memperhatikan keputusan selanjutnya yang akan diambil, dan keputusan tersebut tidak dapat diubah lagi pada langkah selanjutnya.

3.1 Prinsip Utama Algoritma Greedy

Prinsip utama algoritma *greedy* adalah “*take what you can get now!*”. Maksud dari prinsip tersebut adalah sebagai berikut: Pada setiap langkah dalam algoritma *greedy*, kita ambil keputusan yang paling optimal untuk langkah tersebut tanpa memperhatikan konsekuensi pada langkah selanjutnya. Kita namakan solusi tersebut dengan optimum lokal. Kemudian saat pengambilan nilai optimum lokal pada setiap langkah, diharapkan tercapai optimum global, yaitu tercapainya solusi optimum yang melibatkan keseluruhan langkah dari awal sampai akhir.

3.2 Elemen Algoritma Greedy

Elemen–elemen yang digunakan dalam penerapan algoritma *greedy* antara lain :

1. Himpunan Kandidat.
Himpunan yang berisi elemen pembentuk solusi.
2. Himpunan Solusi.
Himpunan yang terpilih sebagai solusi persoalan.
3. Fungsi Seleksi.
Fungsi yang memilih kandidat yang paling mungkin untuk mencapai solusi optimal.
4. Fungsi Kelayakan.
Fungsi yang memeriksa apakah suatu kandidat yang dipilih dapat memberikan solusi yang layak. Maksudnya yaitu apakah kandidat tersebut bersama dengan himpunan solusi yang sudah terbentuk tidak melanggar kendala yang ada.
5. Fungsi Solusi
Fungsi yang mengembalikan nilai *boolean*. *True* jika himpunan solusi yang sudah terbentuk merupakan solusi yang lengkap; *False* jika himpunan solusi belum lengkap.

6. Fungsi Objektif
Fungsi yang mengoptimalkan solusi.

3.3 Skema Umum Algoritma Greedy

Misal kita mengasumsikan elemen algoritma *greedy* sebagai berikut :

Himpunan Kandidat = C ,
Himpunan Solusi = S ,
Fungsi Seleksi = $select()$,
Fungsi Kelayakan = $feasible()$,
Fungsi Solusi = $solution()$, dan
Fungsi Obyektif = $objective()$.

Skema umum dari algoritma *greedy* dapat kita tuliskan :

1. Inisialisasi S dengan kosong.
2. Pilih sebuah kandidat dari C (dengan $select()$).
3. Kurangi C dengan kandidat yang telah terpilih di atas.
4. Periksa apakah kandidat yang dipilih tersebut bersama – sama dengan S membentuk solusi yang layak (dengan $feasible()$). Jika ya, masukkan kandidat ke S ; jika tidak buang kandidat tersebut dan tidak perlu ditelaah lagi.
5. Periksa apakah S yang sudah terbentuk telah memberikan solusi yang lengkap (dengan $solution()$). Jika ya, berhenti; jika tidak, ulangi dari langkah 2.

4. Penentuan Shortest Common Superstring dengan Algoritma Greedy

Persoalan *Shortest Common Superstring* dapat diselesaikan melalui pendekatan algoritma *greedy*. Beberapa ilmuwan telah berusaha memperbaiki algoritma-algoritma untuk menyelesaikan persoalan SCS, diantaranya sebagai berikut:

- Blum et al. (1991)
- S. Teng, F. Yao (1993)
- Czumaj et al. (1994)
- R. Kosaraju, J. Park, C. Stein (1994)
- Armen, C. Stein (1996)
- Breslauer, T. Jiang, Z. Jian (1997)
- Z. Sweedyk (1999)

Disini, kita akan membatasi persoalan SCS ini dengan pendekatan algoritma *greedy* dan salah satu variannya dari Blum et al.

4.1 Penyelesaian dengan algoritma greedy

Sebuah algoritma *greedy* sederhana, kita sebut *GreedySCS(S)* dimana S adalah himpunan string, yang digunakan secara luas dalam persoalan penggabungan SCS ini adalah sebagai berikut (lihat gambar 2.1 untuk pseudo-code)

$GreedySCS(S)$ akan mengecek dua string mana yang mempunyai *overlap* antara kedua string tersebut yang bernilai maksimal. Kemudian $GreedySCS(S)$ akan menggabungkan kedua string tersebut menjadi string baru dan memasukkannya kedalam himpunan string masukan $GreedySCS(S)$, lalu hapus dua string yang lama dari himpunan yang akan menjadi masukan $GreedySCS(S)$ selanjutnya. String yang tersisa adalah SCS dari $GreedySCS(S)$.

Sebagai contoh, diberikan himpunan $S_0 = \{s_1, s_2, s_3, s_4\}$; $s_1 = abba$, $s_2 = banabana$, $s_3 = our$, $s_4 = urban$. $GreedySCS(S)$ akan berjalan sebagai berikut:

1. $s_5 = merge(s_4, s_2) = urbanabana \rightarrow S_1 = \{s_1, s_3, s_5\}$
2. $s_6 = merge(s_3, s_5) = ourbanabana \rightarrow S_2 = \{s_1, s_6\}$
3. $s_7 = merge(s_6, s_1) = ourbanabanabba \rightarrow S_3 = \{s_7\}$
4. return s_7

Sementara pseudo-code dari $GreedySCS(S)$ adalah sebagai berikut:

```

procedure GreedySCS(S)
/*input: S {himpunan string}
output: superstring terpendek dari himpunan S*/

while ( //Sk// > 1)
do
{
    Pilih  $s1 \neq s2 \in S$  sehingga nilai
    overlap( $s1, s2$ ) maksimal

     $S \leftarrow (S \setminus \{s1, s2\}) \cup \{merge(s1, s2)\}$ 
}
return (sisa string yang masih ada pada S)

```

Gambar 2.1 Pseudo-code GreedySCS

4.2 Penyelesaian dengan salah satu varian algoritma *greedy*

Blum et al [1] menyempurnakan $GreedySCS$ dan memperkenalkan algoritma *greedy* lain yaitu $TGreedySCS$ (lihat gambar 2.2 untuk pseudo-code).

$TGreedySCS(S)$ hampir mirip dengan $GreedySCS(S)$, namun $TGreedySCS(S)$ membolehkan pengecekan dua string untuk mendapat *overlap* maksimal termasuk string itu sendiri. Jika *overlap* maksimal terjadi antara dua string yang sama, maka string hasil penggabungan akan disimpan dalam himpunan string baru. Jika *overlap* terjadi antara dua string yang berbeda, maka string hasil penggabungan akan ditambahkan dalam himpunan string masukan $TGreedySCS(S)$ dan kedua string yang lama akan dihapus dari himpunan masukan $TGreedySCS(S)$. Setelah himpunan string masukan habis, $TGreedySCS(S)$ akan menjalankan $GreedySCS(S)$

dengan masukan adalah himpunan string baru. Hasil dari $GreedySCS(S)$ adalah SCS.

Sebagai contoh, diberikan himpunan $S_0 = \{s_1, s_2, s_3, s_4\}$; $s_1 = abba$, $s_2 = banabana$, $s_3 = our$, $s_4 = urban$. $TGreedySCS(S)$ akan berjalan sebagai berikut:

1. $s_5 = merge(s_2, s_2) = banabana \rightarrow S_1 = \{s_1, s_3, s_4\}$, $T1 = \{s_5\}$
2. $s_6 = merge(s_3, s_4) = ourban \rightarrow S_2 = \{s_1, s_6\}$, $T2 = \{s_5\}$
3. $s_7 = merge(s_1, s_1) = abba \rightarrow S_3 = \{s_6\}$, $T3 = \{s_5, s_7\}$
4. $s_8 = merge(s_6, s_6) = ourban \rightarrow S_4 = \emptyset$, $T4 = \{s_5, s_7, s_8\}$
5. $GreedySCS(T)$ dijalankan, dan akan mendapatkan hasil string "ourbanabanabba".

Sementara pseudo-code dari $TGreedySCS(S)$ adalah sebagai berikut:

```

procedure TGreedySCS(S)
/*input: S {himpunan string}
output: superstring terpendek dari himpunan S*/

inisiasi  $T \leftarrow \emptyset$ 
while ( //S// > 0)
do
{
    pilih  $s1 \neq s2 \in S$  sehingga nilai
    overlap( $s1, s2$ ) maksimal

    if ( $s1 \neq s2$ )
    {
         $S \leftarrow (S \setminus \{s1, s2\}) \cup \{merge(s1, s2)\}$ 
    }
    else
    {
         $S \leftarrow S \setminus \{s1\}$ 
         $T \leftarrow T \cup \{merge(s1, s2)\}$ 
    }
}
return (GreedySCS(T))

```

Gambar 2.2 Pseudo-code TGreedySCS

4.4 Analisis

Dalam persoalan persoalan SCS yang dikemukakan dan dibuktikan oleh Blum et al. [1], Terbukti bahwa algoritma $TGreedySCS$ memiliki solusi dengan faktor aproksimasi 3 dari solusi optimumnya. Maksud faktor aproksimasi 3 adalah *superstring* yang ditemukan kebanyakan memiliki 3 kali panjang dari *shortest common superstring* yang optimum.

Walaupun selama ini perkiraan aproksimasi dari algoritma *GreedySCS* adalah 2, tetapi Blum et al. [1] menemukan melalui berbagai eksperimen bahwa *GreedySCS* sebenarnya mempunyai hampiran aproksimasi 4. Perbedaan utama antara *GreedySCS* dan *TGreedySCS* adalah *TGreedySCS* membolehkan untuk mengecek *overlap* antara string dan dirinya sendiri. Ketika maksimal *overlap* terjadi antara sebuah string dan dirinya sendiri, *TGreedySCS* menghapus string dari himpunan string S , dan memasukkan string tersebut kedalam himpunan string baru, T (yang diinisialisasi dengan nilai kosong / *null*). Ketika *overlap* maksimal terjadi antara dua string yang berbeda, *TGreedySCS* akan bekerja seperti *GreedySCS*. Saat semua string sudah terdapat dalam T , *GreedySCS* akan dijalankan dengan parameter adalah T dan string hasilnya akan menjadi output dari *TGreedySCS*.

Algoritma *TGreedySCS* mengimitasi algoritma yang berdasarkan dari teori graf yang mentransformasikan elemen elemen SCS menjadi graf, dan dengan sedikit perubahan menghasilkan algoritma yang memiliki faktor aproksimasi 3.

Sebetulnya dalam persoalan SCS, terdapat beberapa algoritma yang memberikan faktor aproksimasi lebih baik dari 3, yaitu algoritma Sweedyk. Tetapi algoritma ini lebih sulit untuk diimplementasikan karena menggunakan cara yang lebih rumit daripada *GreedySCS* maupun *TGreedySCS*. Sehingga kedua algoritma ini lebih disarankan dalam persoalan SCS.

Kedua algoritma ini termasuk dapat diaplikasikan secara mudah pada berbagai persoalan SCS dan memberikan hasil yang baik dan hasil antara *GreedySCS* dan *TGreedySCS* tidak terlalu menunjukkan perbedaan yang jauh walaupun faktor aproksimasi *TGreedySCS* 3 dan *GreedySCS* 4. Tetapi walaupun begitu, algoritma *GreedySCS* lebih mudah diimplementasikan daripada algoritma *TGreedySCS* dan waktu yang digunakannya pun lebih efisien.

5. Kesimpulan

Berdasarkan analisis diatas, kami menyimpulkan bahwa pada persoalan SCS, algoritma yang digunakan sebaiknya adalah algoritma *GreedySCS*.

Kemudahan aplikasi, faktor aproksimasi yang cukup baik dan tidak berbeda jauh dengan algoritma *TGreedySCS*, serta yang paling utama kepopuleran algoritma *GreedySCS* (terutama digunakan dalam masalah SCS yang cukup populer yaitu persoalan penggabungan string DNA) membuat kami menyimpulkan algoritma *GreedySCS* masih sebagai acuan tolak ukur terbaik dalam menyelesaikan persoalan SCS.

Daftar Pustaka

- [1] A. Blum, T. Jiang, M. Li, J. Tromp, and M. Yannakakis. Linear approximation of shortest superstrings. *Journal of the ACM*, 41:634-647, 1994.
- [2] E. Horowitz and Sartaj Sahni, *Fundamentals of Computer Algorithms*, Pitman Publishing, 1978.
- [3] R. Munir, *Diktat Kuliah IF2251 Strategi Algoritmik*, Lab Ilmu Rekayasa Komputasi, 2006.
- [4] A. Zaritsky dan M. Sipper, *Coevolving solutions to the shortest common superstring problem*, BioSystems, 2003. (draft version: <http://www.cs.bgu.ac.il/~assafza>).