PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
Instituto de Ciências Exatas e de Informática

Projeto e Análise de Algoritmos
Shortest Common Supersequence (SCS)[*]

Gabriel Luciano Gomes[1]
Luigi Domenico Cecchini Soares[2]

[*]Artigo apresentado ao Instituto de Ciências Exatas e Informática da Pontifícia Universidade Católica de Minas Gerais como pré-requisito para obtenção do título de Bacharel em Ciência da Computação.

[1]Aluno, Ciência da Computação, Brasil, glgomes@sga.pucminas.br.
[2]Aluno, Ciência da Computação, Brasil, luigi.soares@sga.pucminas.br.

Sumário

1 INTRODUÇÃO

Dado uma string $S = x_0x_1x_2x_3...x_k$ sobre um alfabeto Σ, defini-se uma subsequência S' de S como sendo qualquer sequência de S, removendo-se de 0 a k termos. Da mesma forma, pode-se definir uma supersequência S' de S como sendo qualquer string $S' = x_0w_0x_1w_1x_2w_2...x_kw_k$ sobre um alfabeto Σ, tal que $w_i \in \Sigma^*$. Assim, dado um conjunto de strings $R = \{S_0, S_1, S_2, S_3, ..., S_u\}$, uma supersequência S do conjunto R será uma supersequência comum se, e somente se, for uma supersequência de cada S_i.

Para exemplificar, tome o conjunto $R = \{S_0, S_1, S_2\}$ de strings sobre o alfabeto $\Sigma = \{a, b\}$. Além disso, considere as seguintes strings formadas a partir de Σ^*: $S_0 = abbb$, $S_1 = bab$ e $S_2 = bba$. Um exemplo de uma supersequência comum do conjunto R é a concatenação das strings:

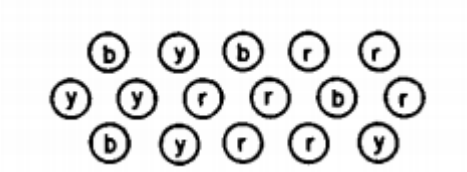
$$\sum_{i=0}^{2} S_i = \mathbf{abbbbabbba}$$

Tendo-se definido o conceito de uma supersequência comum, o problema de decisão da menor supersequência comum (*SCS*) pode ser estabelecido da seguinte forma: Dado um alfabeto Σ, um conjunto finito R de strings sobre Σ^* e um inteiro positivo k, existe uma supersequência comum do conjunto R de tamanho $\leq k$? (RäIHä; UKKONEN, 1981). Em outras palavras, a menor supersequência comum de R, *SCS*(R), é a menor sequência S' tal que $|S'| > |S_i|$, $i = 0, 1, 2, ...\ u$. Por exemplo, tomando o conjunto $R = \{abbb, bab, bba\}$ anterior, temos que SCS$(R) = abbab$. (MAIER, 1978)

2 MODELAGEM ATRAVÉS DE THREADS

É comum pensar na modelagem, tanto do problema da maior subsequência comum (*LCS*) quanto da menor supersequência comum (*SCS*), em termos de encadeamento de nós. Nesta abordagem, cada nó representa uma letra de uma string $S_i \in R$, tal que R é um conjunto de strings. Além disso, cada string é representada em uma linha individual. Suponha três strings $S_1 = bybrr, S_2 = yyrrbr, ands_3 = byrry$. Para estruturar o problema, elas podem ser representadas da seguinte forma:

Figura 1 – Representação das strings por meio de nós

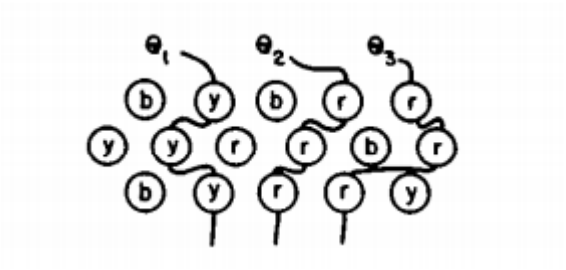


Fonte: (MAIER, 1978)

Para o problema *LCS*, cada *thread* θ_i deve conter exatamente um nó de cada linha,

de forma que todos os nós de θ_i sejam iguais. Além disso, é condição necessária que duas *threads* nunca se cruzem. Assim, deseja-se saber se k *threads* podem ser utilizadas. No exemplo anterior, k deve ser menor ou igual a 3:

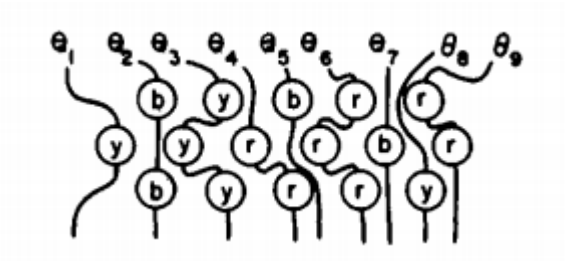
Figura 2 – *LCS - Threads*



Fonte: (MAIER, 1978)

No caso do problema *SCS*, cada *thread* deve conter pelo menos um nó, de forma a se obter uma supersequência comum a todas as strings de R. Assim, para esse problema, deseja-se determinar se k threads são suficientes para concatenar todos os nós. Vale ressaltar que as mesmas regras para o problema *LCS* se aplicam ao problema *SCS*: os nós de uma *thread* θ_i devem ser iguais e duas *threads* não podem se cruzar. No exemplo anterior, k deve ser maior ou igual a 9:

Figura 3 – *SCS - Threads*



Fonte: (MAIER, 1978)

Dado um conjunto de *threads* $\Theta = \theta_1, \theta_2, \theta_3, ..., \theta_j$ associadas a um conjunto de sequências R, é possível obter uma subsquência ou uma supersequência comum, dependendo do tipo de problema que está sendo trabalhado, concatenando os tipos de cada *thread* (letra a que se refere). É importante ressaltar que um conjunto de *threads* gera uma única subsequência ou supersequência, mas o contrário não se aplica, uma vez que mais de um conjunto de *threads* podem produzir a mesma subsequência ou supersequência. (MAIER, 1978)

3 COMPLEXIDADE DO PROBLEMA

MAIER (1978) prova que o problema de decisão *SCS* é da classe de problemas NP-completo, para $|\Sigma| \geq 5$. Tal demonstração se dá através da redução do problema de cobertura

de vértices para o problema SCS. Dado um grafo $G = (N, E)$ e um inteiro k, a entrada do problema da cobertura de vértices é definida como um inteiro k, seguido de uma lista de arestas E: $k; (x_1, y_1); (x_2, y_2); (x_3, y_3); ...(x_r, y_r)$, tal que $x_1, y_1, x_2, y_2, x_3, y_3, ..., x_r, y_r \in N$. Com essa entrada em mãos, é gerado um conjunto R contendo uma sequência *template* T e diversas sequências S_i e, a partir destas, utilizando o esquema de *threads* discutido acima, é provado que existe uma cobertura de vértices de tamanho k correspondente a menor supersequência comum de R. Como a redução do problema da cobertura de vértices para o problema SCS pode ser feita em tempo polinomial, ele é, de fato, um problema NP-completo.

Da mesma forma, utilizando os mesmo métodos de prova, RäIHä; UKKONEN (1981) demonstram que o problema *SCS* pertence a classe de problemas NP-completo para um alfabeto Σ de tamanho maior ou igual a dois. Nesse caso, o alfabeto utilizado consiste de $0's$ e $1's$ ($\Sigma = \{0, 1\}$). De forma análoga as provas apresentadas em (MAIER, 1978), como o problema da cobertura de vértices é NP-completo, uma vez que existe uma redução deste para o problema *SCS* em tempo polinomial, *SCS* é um problema NP-completo para $|\Sigma| \geq 2$.

Vale ressaltar que, para $|\Sigma| = 1$, é fácil perceber que é trivial encontrar uma solução em tempo polinomial, uma vez que a quantidade de *threads* será igual ao tamanho da maior string S_i envolvida. Além disso, o problema *SCS* também se mostra tratável para um conjunto de strings R, tal que $|R|$ = 2, encontrando a *LCS*(R) e inserindo os elementos que não pertençam a *LCS*(R) (respeitando a ordem das strings). Por fim, também já se foi provado que, caso todas as strings $S_i \in R$ possuam tamanho ≤ 2, o problema pode ser resolvido em tempo polinomial.

4 APROXIMAÇÃO PARA SOLUÇÃO

Existem diversos algoritmos para uma solução aproximada para o *SCS*. Pensando nisso, neste relatório será apresentado uma solução utilizando o *Majority-Merge (MM)*. Tendo em consideração a taxa de aproximação $q = |\Sigma|$, com complexidade de tempo $O(qn)$, de um determinado alfabeto Σ^* fixo, o *MM* não apresenta nenhuma relação de pior caso na aproximação e possui desempenho favorável, com complexidade temporal de $O(qkn)$.(NOAMAN; JARADAT, 2011). Além disso, vale ressaltar que agoritmos gulosos propostos apresentam complexidade temporal de $O(k^2n^2)$ e espacial de $O(kn + n^2)$

4.1 Aproximação Proposta

O Algoritmo utilizado será o da Simulação de Colônia de Abelhas (*ABC*), com adaptações para a solução do *SCS*. Com isso em mente, cabe destacar os seguintes passos, relacionadas ao controle do *SCS*:

1. Calcular o tamanho do alfabeto e sua frequência para todas as strings(L), para guiar a geração randômica dos alfabetos utilizados e mais frequentes.

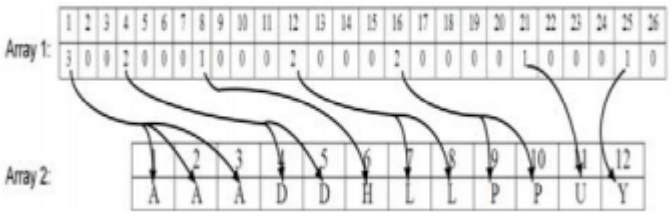
2. Checar a compatibilidade da *SCS* gerada com cada string do alfabeto Σ e atribuir um valor à elas.

Utilizando os seguintes parâmetros:

1. L - o conjunto de Strings inseridas
2. Σ - o alfabeto utilizado
3. f_i (valor adequado) - representação numérica do quanto a solução candidata satisfaz o SCS
4. $Media$ - A média de todos os f_i, para todas as soluções candidatas no ciclo atual.
5. X_i - número de abelhas para especificação do SCS
6. MCN (*maximum cycle number*) - Valor máximo para determinar o número de ciclos que as abelhas geram soluções candidatas.

Em primeiro lugar, a solução aproximada calcula a frequência de cada alfabeto utilizado em L e, então, armazena a frequência de ocorrência de cada caracter em L em um *array* 1. Em seguida, converte-se as frequências em caracteres em um *array* 2. Dado o exemplo: $L = \{APLY, DUHA, LADP\}, \Sigma = \{A, D, H, L, P, U, Y\}$, armazena-se a frequência da cada caracter de L em um *array* 1, sabendo-se que $|\Sigma| = 26$. Assim, converte-se a frequência em caracteres no *array* 2. Esses dois passos garantem a melhor maneira de geração randômica de strings, ilustradas na figura abaixo:

Figura 4 – Geração de strings do alfabeto utilizado



Fonte: (NOAMAN; JARADAT, 2011)

Cada candidato x_i gera um proponente *SCS* escolhendo aleatoriamente caracteres do *array* 2, com tamanhos variados e, com isso, pode-se observar que o alfabeto mais frequente será mais utilizado do que o menos frequente. Dessa forma, um componente espectador irá calcular o f_i para cada candidato SCS, por meio da verificação de cada pretendente com as string de L. Quanto maior o valor f_i, maior a qualificação de ser SCS para L. O valor adequado é calculado por meio do Algoritmo *Merge* adaptado descrito na expressão abaixo:

$$V_{adequado}(S_i) = \frac{\Sigma Merge(S_i, T_1, ...T_n)}{\Sigma V_{adequado}(S_i, ..., S_m)}$$

- n: Número de Strings

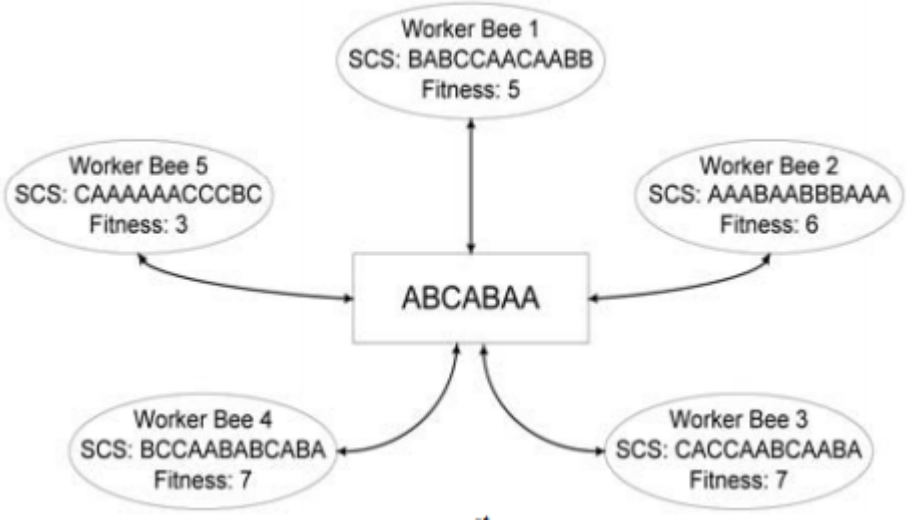
- m: Número de Strings geradas
- S: String gerada
- T: String inserida

Após o cálculo de cada valor adequado, de cada candidato, o espectador deverá calcular a $Media$ para definir se a string gerada possui um valor de compatibilidade bom ou não. Essa medida irá auxiliar na recriação de uma string no próximo ciclo de execução, caso o f_i atual não seja satisfatório. A grandeza é simplesmente calculada por meio da média de todos os f_i das strings geradas pelo número de strings inseridas, como na equação:

$$Media = \frac{\Sigma V_{adequado}(S_i, ..., S_n)}{n}$$

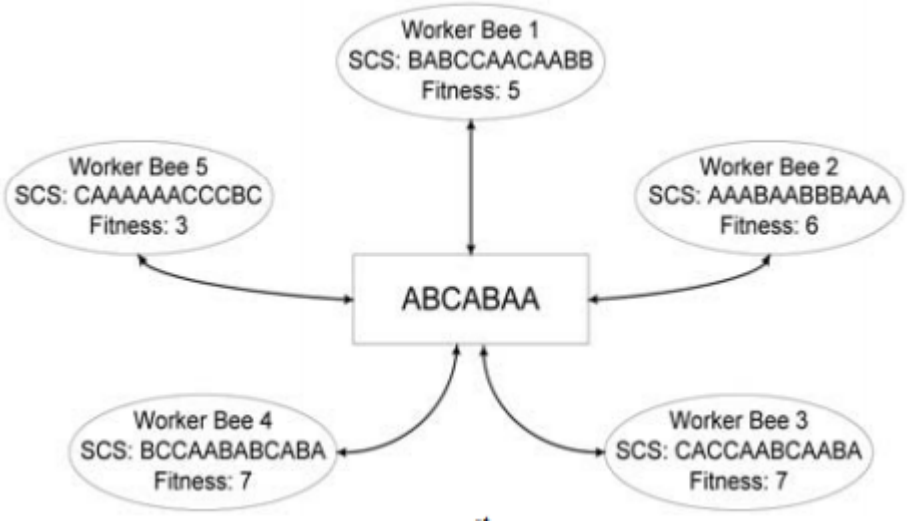
Exemplo: No exemplo, mostra-se que 5 abelhas operarias possuem valores adequados diferentes, resultantes da comparação da SCS gerada com a string "$ABCABAA$"inserida. Pode-se obesevar que as operárias 1 e 5 possuem f_i (*Fitness*) abaixo da $Media$. Assim, geram uma SCS e seus valores aquedados são computados.

Figura 5 – Resultados do primeiro ciclo de execução do programa



Fonte: (NOAMAN; JARADAT, 2011)

Figura 6 – Resultados do segundo ciclo de execução do programa



Fonte: (NOAMAN; JARADAT, 2011)

Para solução do *Shortest Common Supersquence* aplicadas em Colonias Artificias de Abelhas(*ABC-SCS*), utiliza-se o seguinte pseudocódigo:

Algorithm 1: *ABC-SCS*

```
Leitura das strings inseridas
Cálculo do tamanho do alfabeto (|Σ|) e frequências
Gerações Aleatórias de novas soluções para cada abelha operaria
for ciclos = 0, ciclos <= MCN, ciclos++ do
    para cada String Inserida (SI)
    Calcula o f_i para as soluções geradas pelas abelhas operarias
    if f_i < Media then
        Gera uma nova String e calcula o novo f_i
Armazena a solução com o maior f_i e o menor tamanho de SCS
```

Referências

MAIER, David. The complexity of some problems on subsequences and supersequences. **J. ACM**, ACM, New York, NY, USA, v. 25, n. 2, p. 322–336, abr. 1978. ISSN 0004-5411. Disponível em: <http://doi.acm.org/10.1145/322063.322075>.

NOAMAN, Mustafa; JARADAT, Ameera. Solving shortest common supersequence problem using artificial bee colony algorithm. v. 2, p. 80–85, 01 2011.

RäIHä, Kari-Jouko; UKKONEN, Esko. The shortest common supersequence problem over binary alphabet is np-complete. **Theoretical Computer Science**, v. 16, n. 2, p. 187 – 198, 1981. ISSN 0304-3975. Disponível em: <http://www.sciencedirect.com/science/article/pii/030439758190075X>.