Appendix to the article "Practical lower and upper bounds for the Shortest Linear Superstring"

Bastien Cazaux[1,2], Samuel Juhel[2], and Eric Rivals[2]

1 **Department of Computer Science, University of Helsinki, Helsinki, Finland**
`bastien.cazaux@cs.helsinki.fi`

2 **L.I.R.M.M., UMR 5506, Université Montpellier, CNRS, Montpellier, France & Institute of Computational Biology, Montpellier, France**
`samuel.juhel@zaclys.net, rivals@lirmm.fr`

1 Previous approximation algorithms using a greedy strategy for superstring problems

We recall the greedy algorithm for `SLS` (Algorithm 1), and other more complex approximation algorithms for `SLS` that are based on greedy strategy. These algorithms are known as `MGreedy` (Algorithm 3) and `TGreedy` (Algorithm 4). Both used a common procedure which is termed `LCGreedy` ((Algorithm 2), and computes a linear cover of the input set P. A *linear cover* is a set of linear strings that covers all strings of P.

First, we give the `Greedy` algorithm for `SLS`. It yields an approximate linear superstring for P. `Greedy` is conjectured to have an approximation ratio of 2 – its current ratio is 3.5. Ukkonen has proposed a linear time implementation of `Greedy` [2].

Algorithm 1: Algorithm `Greedy`

```
1 Input: a set of strings P; Output: a linear superstring of P;
2 while |P| > 0 do
3     choose u and v in P (necessarily distinct) such that ov(u, v) is maximised;
4     let w be the merge of (u, v);
5     replace u, v by w in P;
6 return the single element of P
```

We call Algorithm 2 `LCGreedy`. It is very similar to `CGreedy`: the only difference is on line 6. In fact, when the chosen overlap would circularise a linear string, it does not use this overlap, but instead store the linear string in the set L. Instead of a cyclic cover, it yields a linear cover of P. Moreover, because of the greedy strategy, the overlap that would have circularise the linear string, is the least from those used to build that linear string. This means that in the EHOG, this overlap corresponds to a node that is the closest to the root in terms of word length, compared to all other nodes traversed for building this linear string. This is related to $cut(P)$, which is defined on page 7 just before Proposition 7.

Now, Algorithm 3 gives `MGreedy`. It first applies `LCGreedy` to P to get L, and then concatenates the word of L.

Algorithm 4 gives `TGreedy`. It first applies `LCGreedy` to P to get L, and then applies `Greedy` on L. `TGreedy` is given below. It is known that `TGreedy` achieves an approximation ratio of 3 for `SLS` [1].

17th International Symposium on Experimental Algorithms (SEA 2018).
Editor: Gianlorenzo D'Angelo; Article No. 18; pp. 18:1–18:3

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Algorithm 2: Algorithm `LCGreedy`

Input: a set of strings P; **Output**: L a set of strings;
$L \leftarrow \emptyset$;
while $|P| > 0$ **do**
    u and v in P (not necessarily distinct) such that $ov(u, v)$ is maximised;
    if $u = v$ **then**
        $L \leftarrow L \cup \{u\}$;
    else
        $P \leftarrow P \setminus \{u, v\} \cup \{pr(u, v)ov(u, v)su(u, v)\}$;
return L

Algorithm 3: Algorithm `MGreedy`

Input: a set of strings P; **Output**: w_m an approximate linear superstring of P;
$L \leftarrow$ `LCGreedy`(P);
$w_m \leftarrow$ concatenate in an arbitrary order the strings of L;
return w_m

Algorithm 4: Algorithm `TGreedy`

Input: a set of strings P; **Output**: w_t an approximate linear superstring of P;
$L \leftarrow$ `LCGreedy`(P);
$w_t \leftarrow$ `Greedy`(L);
return w_t

2 Extension of our algorithm with an improved bound

Algorithm `TGreedy` somehow performs two rounds of greedy optimisation, when the original `Greedy` performs only one. Similarly, we can extend our algorithm `LCGreedyMin` by performing another round of greedy optimisation (instead of simply concatenating the strings of the linear cover output by `LCGreedyMin`). Let us call this algorithm `TGreedyMin`.

Algorithm 5: Algorithm `TGreedyMin`

Input: a set of strings P; **Output**: w_t an approximate linear superstring of P;
$L \leftarrow$ `LCGreedyMin`(P);
$w_t \leftarrow$ `Greedy`(L);
return w_t

The original proof of the approximation ratio of `TGreedy` can be reused to show that `TGreedyMin` also achieves an approximation ratio of 3. This yields the following theorem.

▶ **Theorem 1.** *Let P be a set of strings and let w_{opt} denote an optimal solution of `SLS` of P. Algorithm `TGreedyMin` computes in linear time in $||P||$ an approximate superstring w_t satisfying:*

$$|w_{opt}| \quad \leq \quad |w_t| \quad \leq \quad 3 \times |w_{opt}| .$$

Compared to `LCGreedyMin`, `TGreedyMin` would takes more time because of the other round of optimisation. Given the empirical results of `LCGreedyMin`, we suppose that the improvement in term of superstring length would be rather small compared to the solution of `LCGreedyMin`.

References

1 A. Blum, T. Jiang, M. Li, J. Tromp, and M. Yannakakis. Linear approximation of shortest superstrings. *Journal of the ACM*, 41(4):630–647, 1994.

2 E. Ukkonen. A Linear-Time Algorithm for Finding Approximate Shortest Common Superstrings. *Algorithmica*, 5:313–323, 1990.