Advanced Techniques for Combinatorial Algorithms

Adriano De Marino
a.demarino@campus.unimib.it

Department of Informatics, Systems and Communication
University of Milano-Bicocca, Milano, Italy

1 MST

Question 1

Find the minimum spanning tree (MST) of an undirected weighted graph $G = (V, E)$

Let $G = (V, E)$ be an undirected W-weighted graph with weighting function $W : E \to R^+$. We solve MST problem by considering an empty tree T with only one node v and by iteratively adding the neighbourhood node minimizing the current cost. The pseudocode without using parallel paradigma is given in the following Algorithm 1. The parallel extension of it consists in splitting the vertices set V to the available processors and by iteratively executing a local update of the current tree T within each processor and a global update of T among all the processors. The pseudocode is given in Algorithm 2.

Fact 1. *Both the algorithms effectively provide the MST of a given undirected weighted graph* $G = (V, E)$

Proof. First of all the output $\mathcal{Y}$ of the algorithm is always a tree since, by definition, $\mathcal{Y}$ is acyclic and connected. Furthermore, the output $\mathcal{Y}$ is also spanning since the algorithm iterates until the are no more candidate nodes in Q. Finally, to prove that it is minimal we proceed by contradiction and we assume that $\mathcal{Y}_1$ is a MST of G. If $\mathcal{Y}_1 = \mathcal{Y}$ the thesis follows. Suppose that this is not the case and therfore $\mathcal{Y}_1 \neq \mathcal{Y}$. In this case we can

1. select the first edge $(x, y) = e$ added in $\mathcal{Y}$ not present in $\mathcal{Y}_1$ and let V_e be the set of vertices connected by edges added in $\mathcal{Y}$ before e. Then, $x \in V_e$ and $y \notin V_e$ or $x \notin V_e$ and $y \in V_e$. Suppose $x \in V_e$ and $y \notin V_e$ (the other case can be treated similarly);

2. consider a path joining x and y in $\mathcal{Y}_1$. This path exists since $\mathcal{Y}_1$ is a spanning tree (ST) of G;

3. along this path we can consider an edge $f = (x', y')$ linking a node $x' \in V_e$ with a node $y' \notin V_e$. Such an edge exists since during the construction of $\mathcal{Y}$, the edge e was preferred to f since $W(f) \geq W(e)$

4. Let $\mathcal{Y}_2$ be a tree obtained from $\mathcal{Y}_1$ by replacing f with e. $\mathcal{Y}_2$ is connected, the number of its edges is the same of that of $\mathcal{Y}_1$, satisfy $\sum_{e \in E_{\mathcal{Y}_2}} W(e) \leq \sum_{e \in E_{\mathcal{Y}_1}} W(e)$ and contains all the edges added before adding $e \in \mathcal{Y}$ during the construction of $\mathcal{Y}$.

If we then update $\mathcal{Y}_1$ with $\mathcal{Y}_2$ and we repeat steps 3-6 we will finally end up to update $\mathcal{Y}_1$ with $\mathcal{Y}$. But this is a contraddiction, since we are in the case $\mathcal{Y}_1 \neq \mathcal{Y}$. □

Algorithm 1: `MST pseudocode`

Input:
$G = (V, E)$: undirected W-weighted graph
Result: the MST of G

`set` $V_T = \{v\}$ `and` $E_T = \emptyset$, $T = (V_T, E_T)$ `and` $Q = V$;
while $Q \neq \emptyset$ **do**
    $E_T \leftarrow E_T \cup \underset{(v,w)=e \in V_T \times Q}{ARGMIN} W(e)$;
    update $V_T \leftarrow V_T \cup \{w\}$ and $Q \leftarrow Q \setminus \{w\}$
end
return $LPS(S)$;

Algorithm 2: `MST parallel pseudocode`

Input:
$G = (V, E)$: undirected W-weighted graph
Result: the MST of G

`set` $V_T = \{v\}, E_T = \emptyset$;
`set` $V_i = \{$`vertices assigned to processor` $P_i\} \setminus \{v\}$
`set` $Q_i = V_i$
`set` $Q = \bigcup_i Q_i$
while $Q \neq \emptyset$ **do**
    for *every processor* i **do**
        $e_i \leftarrow \underset{(x,w)=e \in V_T \times Q_i}{ARGMIN} W(e)$;
    end
    `MINREDUCE:` `set` $(x, w^*) = e^* \leftarrow \underset{e_i}{ARGMINW}(e_i)$
    `BROADCAST:`
        `update` $Q_i \leftarrow Q_i \setminus \{w^*\}$
        `update` $V_T \leftarrow V_T \setminus \{w^*\}$
        `update` $E_T \leftarrow E_T \cup \{e^*\}$
        `update` $Q = \bigcup_i Q_i$
end
return $T = (V_T, E_T)$;

2 Derandomize MAX-SAT

Question 2

Given a CNF boolean formula $\mathcal{F}$ with m weighted clauses made up of literals in $\{x_1, \ldots, x_n\}$ find the values for the literals which maximize the combined weight of the satisfied clauses and provide a derandomized version of the $(1 - \frac{1}{e})$-approximation.

To obtain a $(1 - \frac{1}{e})$-approximation of the weighted MAXSAT we first define the corresponding linear program relaxation problem as

$$
\begin{aligned}
(y^*, z^*) &= ARGMAX \sum_{j=1}^{m} w_j z_j \\
&s.t. \\
&\sum_{i \in P_j} y_i + \sum_{i \in N_j} (1 - y_i) \geq z_j \qquad \forall j = 1 \ldots m \\
&0 \leq y_i \leq 1 \qquad \forall i = 1 \ldots n \\
&0 \leq z_j \leq 1 \qquad \forall j = 1 \ldots m
\end{aligned}
$$

where

$$
y_i = \begin{cases} 1, & \text{if } x_i = T \\ 0, & \text{if } x_i = F \end{cases} \qquad z_j = \begin{cases} 1, & \text{if } C_j \text{ is SAT.} \\ 0, & \text{if } C_j \text{ is not SAT.} \end{cases}
$$

$$
\begin{aligned}
P_j &= \texttt{set of indexed of } x_i\texttt{'s not negated in } C_j \\
N_j &= \texttt{set of indexed of } x_i\texttt{'s negated in } C_j
\end{aligned}
$$

Then, by setting each $\hat{x}_i$ to TRUE with probability y_i^* independently we obtain a $(1 - \frac{1}{e})$-approximation of MAX-SAT. To get a derandomized version of $\hat{x}$ we first notice that

$$
E[W] = \sum_j E[w_j] \ \texttt{where} \ E[w_j] = w_j Pr(C_j \text{ is SAT.}) = w_j \left(1 - \prod_{i \in P_j} (1 - y_i^*) \prod_{i \in N_j} y_i^* \right)
$$

Furthermore, by the law of total probability,

$$
\begin{aligned}
E[W] &= E[w_j | X_i = T, X_{l \neq i} = b_l] Pr(X_i = T) + E[w_j | X_i = F, X_{l \neq i} = b_l] Pr(X_i = F) \\
&= E[w_j | X_i = T, X_{l \neq i} = b_l] y_i^* + E[w_j | X_i = F, X_{l \neq i} = b_l](1 - y_i^*) \\
&\leq max\{E[w_j | X_i = T, X_{l \neq i} = b_l], E[w_j | X_i = F, X_{l \neq i} = b_l]\}
\end{aligned}
$$

where

$$
\begin{aligned}
E[w_j | X_1 = b_1, \ldots X_n = b_n] &= w_j Pr(C_j \text{ is SAT.} | X_1 = b_1, \ldots X_n = b_n) \\
&= \begin{cases} w_j, & \text{if the setting satisfy } C_j \\ w_j(1 - \prod_{l=1}^{K} (1 - y_l^*)), & \text{otherwise} \end{cases}
\end{aligned}
$$

where K is the number of the literals still free in the clause C_j. Therefore a derandomized version of the $(1 - \frac{1}{e})$-approximation of MAX-SAT for the literal x_i is given by

$$
x_i^* = \underset{x_i \in \{T, F\}}{ARGMAX} \sum_j E[w_j | X_i, X_{l \neq i} = b_l] \tag{1}
$$

indeed

$$
(1 - \frac{1}{e}) OPT \leq (1 - \frac{1}{e}) \sum_j w_j z_j^* \leq E[W] = \sum_j E[w_j] \leq \sum_j max\{E[w_j | X_i = T, X_{l \neq i} = b_l], E[w_j | X_i = F, X_{l \neq i} = b_l]\}
$$

The approximation (1) can be computed for every combination of free literal (x_i) and fixed literals ($X_{l \neq i} = b_l$), that is for each $i = 1 \ldots n$.

3 Deletion of clauses

Question 3

Identify the minimum number of clauses we need to delete from a CNF boolean formula in order that the rest are satisfiable at the same time.

The problem is equivalent to find the maximum number of clauses of a given CNF boolean formula that can be satisfied at once. The pseudocode is given in the following Algorithm 3.

Algorithm 3: `DC pseudocode`

Input:
a CNF boolean formula $\mathcal{F}$
Result: sat. clauses of $MAXSAT(\mathcal{F})$ with weights $w_j = 1$

- `solve the weighted MAX-SAT with all weights` w_c`'s equal to 1 using the LP-relaxed formulation`
- `evaluate all the clauses accordingly`
- `return only the clauses that are satisfied`

Since in our case all the clauses C_j have size fixed at $k = 2$ we have

$$\begin{aligned} E[W] &\geq \left(1 - \left(1 - \frac{1}{k}\right)^k\right) OPT \\ &= \left(1 - \left(1 - \frac{1}{2}\right)^2\right) OPT \\ &= \left(1 - \frac{1}{4}\right) OPT = \frac{3}{4} OPT \end{aligned}$$

Namely, in this case the approximation rate of the optimal solution is $3/4$.

4 Maximal palindromes

Question 4

Determine the longest palindromic substring (LPS) of a given string S.

Let $S = s_0 s_1 \dots s_{N-1}$ be a string of length N and let $R = s_{N-1} \dots s_2 s_1$ its reverse.

Fact 2. *The longest common substring (LCS) of two strings X and Y is determined by first generating the suffix tree (ST) of the concatenated string $X\#Y\$$ and then by traversing it from its root to the deepest (internal) node having child leaves labeled with suffix indexes in both X and Y.*

Fact 3. *The LCS of a string S and its reverse R and the LPS of S coincide if and only if LCS in R has the same starting position of LCS in S, otherwise if S contains a reversed copy of a non-palindromic substring of length greater or equal than LPS in S, then LCS and LPS will be different.*

Fact 4. *Given a substring of length L in S with starting index i and ending index $j(= i + L - 1)$, then in the reversed string R, the reversed substring will start at index $N - 1 - j$ and will end at index $N - 1 - i$. Therefore, if there is a common substring of length L at indices S_i (forward index) and R_i (reverse index) in S and R respectively, then these will come from same position in S if and only if R_i and S_i satisfy*

$$R_i = (N - 1) - (S_i + L - 1)$$

Fact 5. *Given the ST of $S\#R\$$ where R is the reverse of a string S of length N, the label l of the leaf corresponding to a suffix s is $\leq N-1$ or $\geq N$ if $s \in S$ or $s \in R$ respectively. In the first case we define the* ***forward index*** *as $S_{idx} = l$, whereas in the second one we define the* ***reverse index*** *as $R_{idx} = N - l$.*

Given the Facts 2 - 5 the pseudocode for solving LPS is given in the following Algorithm 4.

Algorithm 4: `LPS pseudocode`

Input:
$S = s_0 s_1 \dots s_{N-1}$: a string of length N
Result: the LPS of S

1. `compute` R `as the reversed of` S;
2. `generate the ST of` $S\#R\$$;
3. `set leaves's forward and reverse label according to Fact 5;`
4. `inherit leaves's labels to their ancestors up to the root;`
5. `find the deepest (internal) node` d_{node} `having both forward and reverse indexes and satisfying`

$$R_{idx} = (N-1) - (S_{idx} + L - 1)$$

6. `define` $LPS(S)$ `concatenating the labels of the path from root to` d_{node}

return $LPS(S)$;

5 Longest repeat

Question 5

Find the longest repeated substring (LRS) of a given string S.

Fact 6. *In a suffix tree, by definition, one node can't have more than one outgoing edge starting with the same character. Therefore potential repeated substring will share one path going through one or more internal node(s) down the tree.*

Fact 7. *Therefore, LRS will end at the internal node which is farthest from the root (i.e. deepest node in the tree), since the length of a substring is the label's length of the path going from the root to that internal node. So finding LRS of a string S reduces to finding the deepest node in the suffix tree of S and then get the path label from the root to that deepest internal node.*

Given the Facts 6 and 7 the pseudocode for solving LRS is given in the following Algorithm 5.

Algorithm 5: `LRS pseudocode`

Input:
$S = s_0 s_1 \dots s_{N-1}$: a string of length N
Result: the LRS of S

1. `generate the ST of string` $S\#$;
2. `find the deepest (internal) node` d_{node} `of ST.`
3. `define` $LRS(S)$ `concatenating the labels of the path from root to` d_{node}

return $LRS(S)$;