Proof: We use the approach described by Gusfield and Bansal (see Section 7 of [16]) that relies on the Decomposition Optimality Theorem for recurrent mutations. We first construct the conflict graph and identify the non-trivial connected components of it in time $O(nm^2)$. Let κ_i be the set of characters associated with component i. We compute the Steiner minimum tree T_i for character set κ_i. The remaining conflict-free characters in $C \setminus \cup_i \kappa_i$ can be added by contracting each T_i to vertices and solving the perfect phylogeny problem using Gusfield's linear time algorithm [14].

Since $\texttt{penalty}(M) = s$, there are at most $s + t + 1$ distinct bit strings defined over character set $\cup_i \kappa_i$. The Steiner space is bounded by 2^t, since $|\cup_i \kappa_i| = t$. Using the Dreyfus-Wagner recursion [22] the total run-time for solving all Steiner tree instances is $O(3^{s+t}2^t)$. ■

Lemma 4.8: The algorithm described solves the BNPP problem in time $O(18^q + qnm^2)$ with probability at least 8^{-q}.

Proof: For a set of taxa $M_i \in L$ (Step 3, Figure 4), using Lemma 4.7 an optimum phylogeny can be constructed in time $O(3^{s_i}6^{t_i} + nm^2)$ where $s_i = \texttt{penalty}(M_i)$ and t_i is the number of non-isolated vertices in the conflict graph of M_i. We know that $\sum_i s_i \leq q$ (since $\texttt{penalty}(I) \leq q$) and $\sum_i t_i \leq q$ (stopping condition of the `while` loop). Therefore, the total time to reconstruct optimum phylogenies for all $M_i \in L$ is bounded by $O(18^q + nm^2)$. The running time for the while loop is bounded by $O(qnm^2)$. Therefore the total running time of the algorithm is $O(18^q + qnm^2)$. Combining Lemmas 4.2 and 4.6, the total probability that all guesses performed by the algorithm is correct is at least 8^{-q}. ■

Lemma 4.9: The algorithm described above can be derandomized to run in time $O(72^q + 8^q nm^2)$.

Proof: It is easy to see that Step 2c can be derandomized by exploring all possible states for the unfixed characters. Since there are at most q unfixed characters throughout the execution, there are 2^q possibilities for the states.

However, Step 2a cannot be derandomized naively. We use the technique of bounded search tree [6] to derandomize it efficiently. We select an arbitrary vertex v from $\cup_i N(M_i)$. We explore both the possibilities on whether v mutates once or multiple times. We can associate a search (binary) tree with the execution of the algorithm, where each node of the tree represents a selection v from $\cup_i N(M_i)$. One child edge represents the execution of the algorithm assuming v mutates once and the other assuming v mutates multiple times. In the execution where v mutates