Original Article:

RPNCH: A Method for Constructing Rooted Phylogenetic Networks from Rooted Triplets based on Height Function

Mohammad Hossein Reyhani [1,*], Hadi Poormohammadi [2,3]

[1]Department of Mathematics, Islamic Azad University, Yazd branch, Yazd, Iran.
[2]Faculty of Engineering, Haeri University, Meybod, Iran.
[3]School of Biological Sciences, Institute for Research in Fundamental Sciences (IPM), Tehran, Iran.

*Corresponding Author: email address: mh_reihani@iauyazd.ac.ir (MH. Reyhani)

ABSTRACT

Phylogenetic networks are a generalization of phylogenetic trees which permit the representation the non-tree-like events. It is NP-hard to construct an optimal rooted phylogenetic network from a given set of rooted triplets. This paper presents a novel algorithm called RPNCH. For a given set of rooted triplets, RPNCH tries to construct a rooted phylogenetic network with the minimum number of reticulation nodes that contains all the given rooted triplets. The performance of RPNCH algorithm on simulated data is reported here.

Keywords: Rooted phylogenetic network; Rooted Triplet; Density; Height function; Reticulation node

INTRODUCTION

Phylogenetic trees are the simplest possible model in determining the evolutionary relationships between currently living species. A rooted phylogenetic tree is a rooted unordered leaf labeled tree. However, there are some evolutionary events like recombination, hybridization, gene conversion, and horizontal gene transfer that are not adequately modeled by phylogenetic trees. In such situations, phylogenetic networks are valuable tools for determining non-tree-like events [1].

A rooted phylogenetic network is a rooted directed acyclic graph in which there exists exactly one node of indegree 1 and outdegree 2, called root. No node has indegree greater than 2 and the outdegree of each node with indegree 2 is 1. Such nodes are called reticulation nodes. In rooted phylogenetic networks, leaves are the nodes with indegree 1 and outdegree 0 and are distinctly labeled by a set of given taxa. Mathematicians are interested in developing methods that infer a phylogenetic tree or network from basic building blocks [1]. In the computation of a rooted tree or network, one group of the basic building blocks are rooted triplets which are the rooted binary trees on three taxa [1]. In 1981, Aho et al., studied the problem of constructing a rooted tree from a set of rooted triplets [2]. They proposed an algorithm called BUILD algorithm which constructs a rooted tree containing all the given rooted triplets in polynomial time, if such a tree exists. If there is no tree for a given set of rooted triplets, BUILD algorithm halts and outputs nothing.

When there is no rooted tree for a given set of rooted triplets, one may try to produce an optimal rooted phylogenetic network that contains all the rooted triplets.

Minimizing the *level* of the rooted phylogenetic network, which is defined as the maximum number of reticulation nodes contained in any biconnected component of the rooted phylogenetic network, and minimizing the number of reticulation nodes are considered as the two possible optimality criteria [1]. In [3] and [4] the authors showed that it is NP-hard to construct a level-1 rooted phylogenetic network that contains all the input rooted triplets. A set of rooted triplets is called dense if for each set of three taxa there is at least one rooted triplet in the input set [4]. In [4] the authors showed that for a dense set

of rooted triplets, the problem can be solved in polynomial time. After their results, all research in this new area up to this point have focused on constructing rooted phylogenetic networks from dense sets of rooted triplets. LEV1ATHAN is an algorithm which generates a level-1 rooted phylogenetic network from a set of rooted triplets, if such a network exists. [5]. Specifically, it attempts to find a level-1 rooted phylogenetic network consistent with as many of the input rooted triplets as possible. This problem is known to be NP-hard [5]. The algorithm by [6] can be used to find at most a level-2 rooted phylogenetic network which minimizes the number of reticulation nodes, if such a network exists. In [6] the authors also showed that for a dense set of rooted triplets τ, if τ is precisely equal to the set of rooted triplets consistent with some rooted phylogenetic network, then they can construct such a rooted phylogenetic network with smallest possible level in time $O(|\tau|^{k+1})$, where k is a fixed upper bound on the level of the network. In addition, based on the ideas described in [6], for a given dense set of rooted triplets τ, the authors proposed SIMPLISTIC algorithm which always returns some rooted phylogenetic network consistent with τ. But it does not give any minimality guarantees. In [7] the authors presented TripNet algorithm which tries to construct a rooted phylogenetic network with the minimum number of reticulation nodes from an arbitrary set of rooted triplets. This paper follows the same definitions and notation of [7]. It also presents a new heuristic algorithm called RPNCH. The next section presents some definitions and notation. First, the concepts of the directed graph G_τ related to a set of triplets τ and the height function of a tree are introduced and BUILD algorithm is restated based on these two concepts. Then, the concept of the height function from trees to networks is generalized and RPNCH is presented. The following discusses the runtime of RPNCH. And then, the results are presented and compared with the SIMPLISTIC results; and finally, the performance of RPNCH is discussed.

DEFINITIONS

Let X be a set of taxa. A *rooted phylogenetic tree* (*tree* for short) on X is a rooted unordered leaf labeled tree whose leaves are distinctly labeled by X and every node which is not a leaf has at least outdegree 2. A *directed acyclic graph* (DAG) is a directed graph that is free of directed cycles. A DAG G is *connected* if there is an undirected path between any two nodes of G. It is *biconnected* if it contains no node whose removal disconnects G. A biconnected component of a graph G is a maximal biconnected subgraph of G. A *rooted phylogenetic network* (*network* for short) on X is a rooted DAG in which *root* has indegree 0 and outdegree 2 and every node except the root satisfies one of the following conditions:

a) It has indegree 2 and outdegree 1. These nodes are called *reticulation nodes*.

b) It has indegree 1 and outdegree 2.

c) It has indegree 1 and outdegree 0. These nodes are called *leaves* and are distinctly labeled by X.

A network is said to be a level-k network if each of its biconnected components contain at most k reticulation nodes. A tree can be considered as a level-0 network. A *rooted triplet* (*triplet* for short) is a rooted binary unordered tree with three leaves. We use $ij|k$ to denote a triplet with taxa i and j on one side and k on the other side of the root (Figure.1).

A set of triplets τ is called *dense* if for each subset of three taxa, there is at least one triplet in τ. A triplet $ij|k$ is c*onsistent* with a network N or equivalently N is consistent with $ij|k$ if the leaf set of $ij|k$ is the subset of the leaf set of N, and N contains a subdivision of $ij|k$, i.e. if N contains distinct nodes u and v and pairwise internally node-disjoint paths $u \to i, u \to j, v \to u$ and $v \to k$. Figure 2 shows an example of a network which is consistent with $ij|k$. A set τ of triplets is consistent with a network N if all the triplets in τ are consistent with N. We use the symbols $\tau(N)$ and L_N to represent the set of all triplets that are consistent with N and the set of labels of its leaves respectively. For any set τ of triplets define $L(\tau) = \cup_{t \in \tau} L_t$. The set τ is called a set of triplets on X if $L(\tau) = X$.

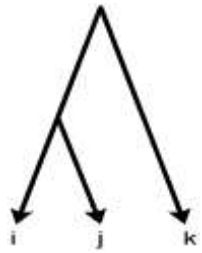

Figure 1. Triplet *ij|k*.

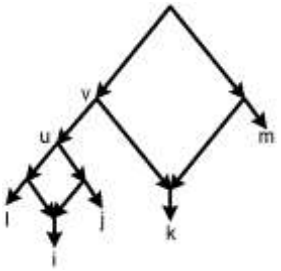

Figure 2. *ij|k* is consistent with the network.

METHODS

This section reviews the concept of the directed graph related to a set of triplets, the height function of a tree, and restate BUILD algorithm based on these concepts which is introduced in [7]. In [7], the authors introduced a generalization of the concept of the height function to the networks. The new heuristic algorithm is as follows:

Definition 1 *Let* τ *be a set of triplets. Define* G_τ, *the directed graph related to* τ, *by* $V(G_\tau)=\{\{i,j\}: i,j\in L(\tau), i\neq j\}$ (*we denote* $\{i,j\}$ *by ij for short) and* $E(G_\tau)=\{(ij,ik): ij \mid k\in\tau\}\cup\{(ij,jk): ij \mid k\in\tau\}$.

Figure 3(a) shows the directed graph related to the given set of triplets τ = {*kl |j, kl |i, jk |i, jl |i*}.

The graph G_τ has an important role in the remaining of the paper. Let $\binom{X}{2}$ denote the set of all subsets of *X* of size 2.

Definition 2 *Let X be an arbitrary finite set. A function* $h:\binom{X}{2}\to N$ *is called a height function on X.*

Let *T* be a rooted tree with the root *r*, c_{ij} be the lowest common ancestor of the leaves *i* and *j*, and l_T denote the length of the longest path started from *r*. For any two nodes *x* and *y*, let $d_T(x,y)$ denote the number of edges of the path between *x* and *y*.

Definition 3 *The height function of T,* h_T *is defined as* $h_T(i,j)=l_T-d_T(r,c_{ij})$ *where i and j are two distinct leaves of T.*

Let τ be a set of triplets, G_τ be a DAG and l_{G_τ} denotes the length of the longest path in G_τ. Since G_τ is a DAG, the set of nodes with outdegree zero is nonempty. Assign $l_{G_\tau}+1$ to the nodes with outdegree zero and remove them from G_τ. Assign l_{G_τ} to the nodes with outdegree zero in the resulting graph and continue this procedure until all nodes are removed.

Definition 4 *For any two distinct* $i,j\in L(\tau)$, *define* $h_{G_\tau}(i,j)$ *as the value that is assigned by the above procedure to the node ij and call it the height function related to* G_τ.

Let τ be a set of triplets. In [7] the authors showed that if τ is consistent with a tree then G_τ is a DAG and h_{G_τ} is well-defined. Now we restate BUILD algorithm, using height function. which is referred to by *HBUILD* [7].

Let *h* be a height function on *X*. Define a weighted complete graph (*G*,*h*) where *V*(*G*)=*X* and edge {*i*,*j*} has weight *h*(*i*,*j*). Remove the edges with maximum weight from *G*. If removing these edges result in a connected graph, the algorithm stops. Otherwise, the process of removing the edges with maximum weight is continued in each connected component until each connected component contains only one node. At the end of this procedure one can reconstruct the tree by reversing the steps of the algorithm similar to BUILD algorithm. The algorithm above decides in polynomial time whether a tree with height function *h* exists (See Figure 3).

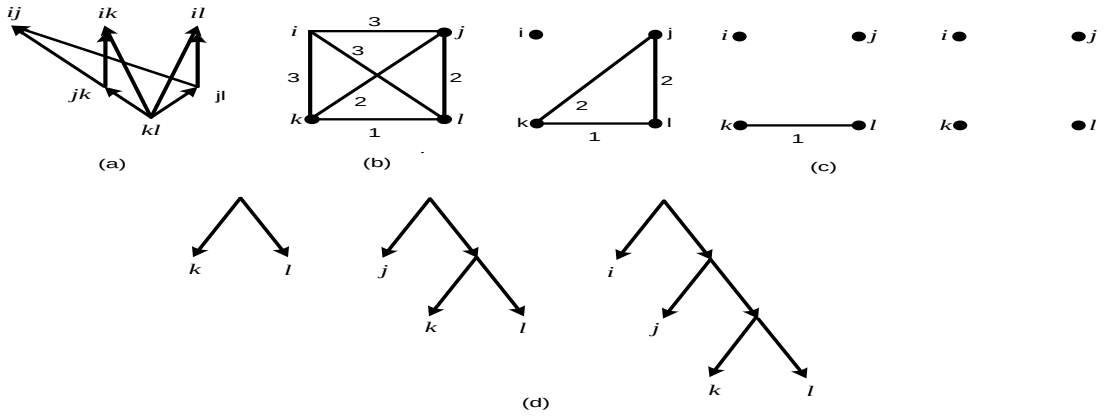

Figure 3. The steps of constructing T_τ from the given set τ = {*kl |j, kl |i, jk |i, jl |i*}, using HBUILD. (a) Graph G_τ , (b) Graph (*G*,*h*), (c) Removing maximum weights from the graph (*G*,*h*), (d) Constructing T_τ using step c.

Let T_τ be the unique tree that is produced by HBUILD. Now if τ is a set of triplets which is consistent with a tree then G_τ is a DAG, $h_{T_\tau} \le h_{G_\tau} = h$, and HBUILD constructs T_τ [7].

The generalization of the concept of the height function form trees to networks is not straightforward because the concept of (lowest) common ancestor of two leaves of a network is not well-defined.

Let *N* be a network with the root *r* and l_N be the length of the longest directed path from *r* to the leaves. For each node *u* consider *d*(*r*,*u*) as the length of the longest directed path from *r* to *u*. For any two nodes *u* and *v*, *u* is called an *ancestor* of *v*, if there exists a directed path from *u* to *v*. If *u* is an ancestor of *v* then *v* is *lower* than *u*. Let *i* and *j* be two leaves of *N*. *c* is called a *lowest common ancestor* of *i* and *j* in *N*, if *c* is a common ancestor of *i* and *j* and there is no common ancestor of *i* and *j* lower than *c*. For any two leaves *i* and *j*, let C_{ij} denote the set of all lowest common ancestors of *i* and *j*. Definition 5 *For each pair of leaves i and j, define* $h_N(i,j) = \min\{l_N - d(r,c) : c \in C_{ij}\}$ *and call it the height function of N.* Obviously, every network *N* indicates a unique height function h_N. But two different networks may have the same height function. In [7] the authors proved that for a given height function *h*, there is a network *N* such that h_N=*h*. In [7] the authors introduced a computational method for computing h_N using Integer Programming and proved that a triplet *ij*|*k* is consistent with a tree *T* if and only if $h_T(i,j)<h_T(i,k)$ or $h_T(i,j)<h_T(j,k)$. Also in [7] it was proved that for a given network *N* and its three distinct leaves *i*, *j*, and *k*, if $h_N(i,j)<h_N(i,k)$ or $h_N(i,j)<h_N(j,k)$ then *ij*|*k* is consistent with *N*. Based on these concepts, the authors introduced the following Integer Programming $IP(\tau, s)$ for a given set of triplets τ with $|L(\tau)| = n$.

Maximize $\sum_{1 \le i,j \le n} h(i,j)$,

Subject to: $h(i,k)-h(i,j)>0 \quad ij \mid k \in \tau$,

$h(j,k)-h(i,j)>0 \quad ij \mid k \in \tau$,

$0 < h(i,j) \le s \quad 1 \le i, j \le n$.

It was proved that G_τ is a DAG if and only if for some integer *s*, $IP(\tau, s)$ has a feasible solution. In this case the minimum number *s* is $l_{G_\tau} + 1$, for which $IP(\tau, s)$ has a feasible solution. Also it was proved that if τ is consistent with a tree, then h_{T_τ} is the unique optimal solution to the $IP(\tau, l_{G_\tau} + 1)$. The above contents imply that the solution of the above IP is a good approximation of the height function of a network *N* which is consistent with τ [7]. This section presents RPNCH method. If there is a tree consistent with a set of triplets τ , using HBUILD, this tree can be constructed. If there is no tree consistent with a set of triplets τ , one possibility is that G_τ is not a DAG. In this case we remove some edges from G_τ in such a way that the resulting graph G'_τ is a DAG. Removing minimum number of edges from a directed graph to make it a DAG is known as the *minimum*

Feedback Arc Set problem which is NP-hard [8]. Thus we use the heuristic algorithm GR [9], and try to remove as minimum number of edges as possible from G_τ in order to lose minimum information. For simplicity from here, G'_τ is denoted by G_τ. Now similar to HBUILD, the edges are removed with maximum weight from (*G*,*h*). If in one step, removing the edges with maximum weight from a connected component *C* results in a connected component *C'*, the following three methods are used to disconnect *C'*.

i. In the first method, the process of removing edges with maximum weights from *C'* is continued until it becomes disconnected.

ii. In the second method, Min-cut algorithm is applied and the edges are removed from *C*' in such a way that the sum of the weights of the removed edges are minimum and *C*' is converted into two connected components.

iii. In the third method, assume that in *C*' the weight of the edges with maximum weight is *m*. For an arbitrary edge in *C*' with weight *w*, the weight is updated to *m*-*w*+1 and the Min-cut algorithm is performed on *C*' with these new edge weights.

For each method continue its process for each connected component until each connected component contains only one node. At the end, like HBUILD, one can reconstruct a unique tree for each method by reversing its steps.

Now for each tree, the goal is to add some edges in order to obtain a network consistent with the given set of triplets τ based on the concept of the height function of networks. Let *T* be a tree obtained from one of the three methods, *r* be the root of *T*, *i* be one of its leaves, and l_i be the length of the path between *r* and *i*. The method of constructing *T* shows that for each leaf *i*, $l_i \le l_{G_\tau}+1$. $l_{G_\tau}+1-l_i$ nodes are added to the edge for which one of its two ends is *i*. In this new tree for each two leaves *i* and *j* if *h*(*i*,*j*) is the same as the value which is obtained from the above IP, nothing is done. Else, let $h_{IP}(i,j)$ be its IP value which is not the same as *h(i,j)*. A new edge is added so that one of its ends is the node of the path between *r* and *i* which has distance $l_{G_\tau}+1-h_{IP}(i,j)$ from *r* and the other end is the node *of* the path between *r* and *j* which has distance $l_{G_\tau}+1-h_{IP}(i,j)$ from *r* and randomly assign a direction to it. Suppose that the triplet $ij|k \in \tau$ is not consistent with this network. Connect the node which its child is *i* to the node which its child is *j*. Now *ij*|*k* is consistent with this new network. This process will be continued until the final network is consistent with τ. Finally the best network is reported as the output of the algorithm. This algorithm is named Rooted Phylogenetic Network Construction with Height: Algorithm RPNCH for short.

RUNTIME

Here, the time complexity of RPNCH is studied. Let $|L(\tau)| = n$ and $|\tau| = m$. At the beginning, G_τ should be computed. Its time complexity is *O*(*m*). Then, if G_τ is not a DAG, the algorithm GR is applied in *O*(|edges|) time, which is equivalent to *O*(*m*). Now the nodes with outdegree zero are recognized and then Topological sort is performed on G'_τ Its time complexity is O(|nodes|+|edges|) or equivalently $O(m+n^2)$. The next step is assigning the height to each node of DAG in $O(n^2)$. After these steps, the graph (*G*, *h*) is constructed in $O(n^2)$. So the runtime of the above steps is $O(m+n^2)$. Now we are ready to perform the tree construction methods. For the first method, in each step, removing the edges with maximum weight is done for each connected component in *O*(*m*). In addition, in each step, it is necessary to compare the number of connected components with the previous step. Thus, *DFS* algorithm is performed in *O*(*n*). The overall run time is *O*(*mn*). Since there are *n* nodes, the total runtime is $O(mn^2)$. For the second method, in each step, removing the edges with maximum weight is performed in *O*(*mn*). Then, Min-cut is used in $O(mn+ n^2\log n)$. The overall run time is $O(mn+ n^2\log n)$. Like the first method, the total runtime is $O(mn^2+ n^3\log n)$. The runtime of the third method is exactly the same as the second method. The network construction procedure from tree is done in *O*(*mlogn*). So the overall runtime of RPNCH is $O(mn^2+ n^3\log n)$.

RESULTS

In order to study the performance of RPNCH, the following scenario is performed. The standard methods are used to obtain triplets from (biological) sequences data [2].

Triplets are easy to construct using the input sequences: Maximum Parsimony or Maximum Likelihood are existing methods for constructing triplets [2]. PhyML is a software that construct weighted unrooted binary trees based on the input sequences using Maximum Likelihood criterion. PhyML can be used to produce triplets. It is enough to add an outgroup to all the sets consisting of three sequences in the input and construct a quartet using the current methods. At the end, the triplet corresponding to each of these groups can be extracted by removing the outlier sequence. Finally, note that this simple and intuitive method works with a certainty threshold where there is the option to adjust this threshold. In fact, the unique inner edge weight in the corresponding quartet is considered as the *threshold* and the triplets with the threshold at most zero are not considered. 1000 sets of sequences of size at least 10 and at most 30 under biological presumptions are generated, and for each of them a set of triplets is obtained using Maximum Likelihood criterion. Then, the RPNCH results are compared with the SIMPLISTIC results on these sets of triplets. SIMPLISTIC is a method which constructs networks from dense sets of triplets. Since for a set of triplets which is obtained from the above method there might be triplets with the threshold at most zero, then a set of triplets might not be dense. In order to make it dense, some triplets are randomly added to it The results show that when the level of the SIMPLISTIC network is at most 6, SIMPLISTIC outperforms RPNCH in both the number of reticulation nodes and the level of the resulting networks. In these cases on average the difference between these numbers is at most 4 and in average 2.5. In the 82 % of these cases, the runtime of both algorithms is nearly the same. In the remaining 18 % of the cases, the runtime of RPNCH is at most 10 seconds, but the Simplistic runtime is at least 55 seconds. For the network with level greater than 6, the runtime of SIMPLISTIC is very high and in most cases after one hour, SIMPLISTIC does not give any output. However, the RPNCH runtime for all these cases is at most 4 minutes and the resulting network contains at most 19 reticulation nodes. Let N_{finite} be the set of triplets sets in which SIMPLISTIC outputs a results in time less than one hour. Note that randomly for 20% of triplets sets belonging to N_{finite} it was checked if SIMPLISTIC can construct a network in time less than 4 hours. But in all such cases the response was negative. Table 1 shows the details of RPNCH and SIMPLISTIC results on 1000 simulated data.

Table 1. RPNCH and SIMPLISTIC results on 1000 sets of triplets

Percent of triplets sets that belong to N_{finite}	62%
Average differences of the number of reticulation nodes for triplets in N_{finite} for both methods (in all cases SIMPLISTIC outperforms RPNCH)	2.5
Percent of triplets sets in N_{finite} in which the runtime of both methods is nearly the same	82%
Average runtime for triplets sets in N_{finite} with the same runtime for both methods (Sec)	10
Average SIMPLISTIC runtime for triplets in N_{finite}; both methods do not have the same runtime (Sec)	126
Average RPNCH runtime for triplets in N_{finite} : both methods do not have the same runtime (Sec)	7
Average level of SIMPLISTIC results that belong to N_{finite}	3.8
Average level of RPNCH results that belong to N_{finite}	6.3
Average number of reticulation nodes for RPNCH results for triplets not in N_{finite}	17
Average runtime of RPNCH results for triplets not in N_{finite} (Sec)	200

DISCUSSION

This paper introduces a new method called RPNCH which constructs a network for an arbitrary given set of triplets. In order to show the performance of RPNCH,1000 sets of triplets were generated and then the results were compared with SIMPLISTIC results. The results showed that when the level of the resulting network exceeds 6, SIMPLISTIC has not the ability to construct a network in an appropriate time and in all cases after at least one hour it does not give any output. But for these cases, RPNCH outputs a network with at most 19 reticulation nodes in at most 4 minutes. Also on average for networks with level less than 6 the runtime of RPNCH outperforms SIMPLISTIC.

In general, the results show that for a dense set of triplets, SIMPLISTIC checks all possible solutions starting from trees until it finds an answer with the minimum level. When the resulting network is more complex, the search space grows rapidly. So SIMPLISTIC take much time to provide a solution. The results

show that SIMPLISTIC is appropriate for dense sets of triplets which are consistent with the networks with level at most 6. For more complex networks with the level greater than 6, SIMPLISTIC does not work well and it takes much time to provide output. It means that for complex networks with high levels, the SIMPLISTIC runtime increased exponentially. However, in all cases, RPNCH outputs a network in an appropriate time. It is remarkable that in contrast with SIMPLISTIC which merely works for dense sets of triplets, RPNCH works for any arbitrary given set of triplets.

ACKNOWLEDGMENT

We would like to thank Islamic Azad Yazd University for its supports. The research of the second author was in part supported by a grant BS-1394-01-06 from the Institute for Research in Fundamental Sciences (IPM), Tehran, Iran.
"The authors declare no conflict of interest"

REFERENCES

1.Huson DH, Rupp R, Scornavacca C. Phylogenetic Networks Concepts, Algorithms and Applications. Cambridge University Press. 2010.
2.Aho AV, Sagiv Y, Szymanski TG, Ullman optimization of relational expressions. SIAM J. Comp. 1981; 10: 405-421.
3.Jansson J, Nguyen NB, Sung WK. Algorithms for combining rooted triplets into a galled phylogenetic network. SIAM Journal on Computing. 2006; 35: 1098-1121.
4.Jansson J, Sung W.K. Inferring a Level-1 Phylogenetic Network from a Dense Set of Rooted Triplets. Theoretical Computer Science. 2006; 363: 60-68.
5.Huber K, Iersel LV, Kelk S, Suchecki R. A Practical Algorithm for Reconstructing Level-1 Phylogenetic Networks. IEEE/ACM Transactions on Computational Biology and Bioinformatics. 2010.
6.Iersel LV, Kelk S. Constructing the Simplest Possible Phylogenetic Network from Triplets. Algorithmica. 2011; 207-235.
7.Poormohammadi H, Eslahchi Ch, Tusserkani R. TripNet: A method for Constructing Rooted Phylogenetic Networks from Rooted Triplets. PLoS ONE. 2014; 9(9): e106531. doi:10.1371/journal.pone.0106531.
8.Karp R. Reducibility among combinatorial problems. Proc. Sympos., IBM Thomas J. Watson Res. Center, Yorktown Heights, N.Y. 1972; 85-103.
9.Eades P, Lin X, Smyth WF. A fast and effective heuristic for the feedback arc set problem. Information Processing Letters. 1993; 47: 319-323.