Coevolution: An Open Access Journal

Publication details, including instructions for authors and subscription information:
http://www.tandfonline.com/loi/tcev20

An improved node mapping algorithm for the cophylogeny reconstruction problem

Benjamin Drinkwater\textsuperscript{a} & Michael A. Charleston\textsuperscript{a}

\textsuperscript{a} School of Information Technologies, University of Sydney, Sydney, Australia

Published online: 24 Apr 2014.

To cite this article: Benjamin Drinkwater & Michael A. Charleston (2014) An improved node mapping algorithm for the cophylogeny reconstruction problem, Coevolution: An Open Access Journal, 2:1, 1-17, DOI: 10.1080/23256214.2014.906070

To link to this article: http://dx.doi.org/10.1080/23256214.2014.906070

PLEASE SCROLL DOWN FOR ARTICLE

Taylor & Francis makes every effort to ensure the accuracy of all the information (the "Content") contained in the publications on our platform. Taylor & Francis, our agents, and our licensors make no representations or warranties whatsoever as to the accuracy, completeness, or suitability for any purpose of the Content. Versions of published Taylor & Francis and Routledge Open articles and Taylor & Francis and Routledge Open Select articles posted to institutional or subject repositories or any other third-party website are without warranty from Taylor & Francis of any kind, either expressed or implied, including, but not limited to, warranties of merchantability, fitness for a particular purpose, or non-infringement. Any opinions and views expressed in this article are the opinions and views of the authors, and are not the views of or endorsed by Taylor & Francis. The accuracy of the Content should not be relied upon and should be independently verified with primary sources of information. Taylor & Francis shall not be liable for any losses, actions, claims, proceedings, demands, costs, expenses, damages, and other liabilities whatsoever or howsoever caused arising directly or indirectly in connection with, in relation to or arising out of the use of the Content.

This article may be used for research, teaching, and private study purposes. Terms & Conditions of access and use can be found at http://www.tandfonline.com/page/terms-and-conditions

It is essential that you check the license status of any given Open and Open Select article to confirm conditions of access and use.
An improved node mapping algorithm for the cophylogeny reconstruction problem

Benjamin Drinkwater* and Michael A. Charleston

School of Information Technologies, University of Sydney, Sydney, Australia

(Received 11 December 2013; accepted 20 January 2014)

The cophylogeny reconstruction problem is central in the inference of ancient relationships among ecologically linked groups of organisms. Although it is known to be NP-Complete, there have been several heuristics proposed that often find good solutions. One such approach is to fix the order of the internal nodes of the host phylogeny and use dynamic programming to recover an optimal reconstruction in polynomial time. A meta-heuristic framework is then leveraged to iterate over the factorial set of fixed node orderings. The original solution used a technique known as node mapping, taking $O(n^3)$ and was designed to handle phylogenetic networks. There has since been a number of $O(n^3)$ implementations for this problem which use specific properties of phylogenetic trees to reduce their running time. In this paper, we introduce a set of new optimisation techniques to achieve a running time improvement matching these cubic time algorithms, while using node mapping, and discuss how these newly proposed optimisation techniques may lead to an improvement in the generalised case of phylogenetic networks and provide valuable insights into how to construct an optimized algorithm for handling evolutionary systems with widespread parasites.

Keywords: cophylogeny; bioinformatics; dynamic programming; NP-Complete

Cophylogenetics considers the macro-scale coevolutionary associations between two or more phylogenetic histories. One aim of cophylogenetics is the development of algorithms for reconstructing the coevolutionary dependencies between ecologically linked groups of organisms, known as cophylogenetic reconstructions. The aim of the algorithms discussed in this paper is to recover the minimum cost cophylogenetic reconstructions.

Although able to consider any evolutionary systems in which more than two phylogenies coexist, this problem is often discussed in terms of a host phylogeny ($H$) and a single corresponding parasite phylogeny ($P$). This problem is not constrained to simply analysing host and parasite systems, however, having been applied to a number of other interesting biological phenomena, including herbivore and plant interactions (Percy et al. 2004), genes and the species that house them (Libeskind-Hadas & Charleston 2009), historical biogeography (Ronquist 1997), mutualism between multiple organisms (Jackson 1999), and mimicry between species (Hoyal Cuthill & Charleston 2012).

This work considers the case where both the host ($H$) and parasite ($P$) phylogenies are rooted trees rather than the more generalised problem of handling phylogenetic networks. We consider the trees as a set of vertices (nodes) and edges where the host tree is defined as $H = (V_H, E_H)$ and the parasite tree is defined as $P = (V_P, E_P)$. To simplify the complexity analysis, we consider the number of leaf nodes in $V_H$ and $V_P$ to both be $n$, with the total number of internal nodes and leaf nodes equal to $2n - 1$. This is not a methodological requirement but is used to allow for a consistent comparison with the existing algorithms. This matches the order of the number of edges in $E_H$ and $E_P$ as a tree with $2n - 1$ nodes has $2n - 2$ edges. This is in contrast to a phylogenetic network which can have up to a quadratic number of edges.

Associations ($\varphi$) between each phylogenetic history are marked only between the leaf nodes of $V_H$ and $V_P$. These associations represent the known interactions between $H$ and $P$. The problem considered here assumes each parasite inhabits exactly one host, while each host may have zero, one or more than one parasite. Although there is an exponential number of combinations possible for associations between $H$ and $P$, the requirement that each parasite node is only permitted a single association results in a tight bound for the number of associations $\varphi$, for a specific instance of $n$ matching the number of leaves in $P$. Extending the proposed algorithm to the case where many associations are permitted for each leaf node $\in V_P$ is discussed in the Further Work section.

The aim of the cophylogeny reconstruction problem is to recover the minimum cost mapping, $\Phi$, of nodes from $V_P$ into $H$ such the initial associations, $\varphi$ are conserved. The mapping, $\Phi$, is constructed using the four recoverable biological events, Codivergence, Duplication, Host Switch and Loss.

A Codivergence event is a concurrent divergence event of both the parasite and host lineages. A signal of coevolution in an evolutionary system is a high number of codivergence events in the cophylogeny reconstruction (Page 2002). A Duplication event is an independent divergence of the parasite lineage with respect to the host, where both

\*Corresponding author. Email: benjamin.drinkwater@sydney.edu.au

© 2014 The Author(s). Published by Taylor & Francis.
This is an open-access article distributed under the terms of the Creative Commons Attribution License http://creativecommons.org/licenses/by/3.0/, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited. The moral rights of the named author(s) have been asserted.
new lineages continue to trace the existing host phylogeny (Charleston 2003). A Loss event includes three separate phenomena which map to a common reconstruction event, including failure to diverge by the parasite lineage, an extinction event in the parasite lineage, or a sampling error occurring when reconstructing the associations (φ) between the host and parasite (Paterson et al. 2003). A Host Switch event is an independent divergence event in the parasite lineage with respect to the host, where one of the new parasite lineages shifts to a new lineage in the host, with the second lineage continuing to trace the original host lineage (Charleston & Perkins 2006). We define the initial host lineage as the take-off edge and the new lineage as the landing edge. A cophylogenetic reconstruction that contains a map Φ including all four events can be seen in Figure 1.

Reconstructing the minimum cost map requires that each event be assigned a penalty cost. This set of costs for each biological event, which we refer to as the cost scheme, is hereby treated as a vector in the form \( \nu_{\text{cost}} = (c, d, h, l) \) where \( c, d, h \) and \( l \) represent the penalty costs associated with codivergence duplication, host switch and loss, respectively. The total cost for a cophylogeny mapping can be constructed as follows:

\[
\text{map cost} = c \times n_c + d \times n_d + h \times n_h + l \times n_l, \tag{1}
\]

where \( n_c, n_d, n_h, n_l \) represent the number of events for codivergence duplication, host switch and loss, respectively.

Traditionally, cost schemes aim to promote codivergence events by assigning a reduced penalty score for codivergence. This has been achieved in a number of ways, such as the maximising codivergence cost scheme, where all other event costs are assigned 0 while codivergence is assigned a cost of \(-1\) such that the minimum cost solution is the case with the highest number of codivergence events. This approach has been noted to overcount loss and host switch events (Ronquist 2002). The ‘Jungle’ approach was later proposed. In this approach, codivergence was assigned a cost less than the remaining events. This cost scheme has been shown to guarantee that all recovered solutions are from the Pareto front (Libeskind-Hadas & Charleston 2009). An alternate approach to minimising the cost of codivergence is to prevent host switch events, which is known as reconciliation (Ronquist 2002). Methods employing reconciliation such as Page’s Reconciled Tree Analysis can solve this problem in \( O(n) \) for all problem instances (Page 1994a, 1994b). Considering all four biological events, however, significantly increases the complexity of the problem. In fact, this problem has been shown to be NP-Complete when permitting all four biological events (Libeskind-Hadas & Charleston 2009; Ovadia et al. 2011).

This increase in complexity is due to the handling of host switch events. Host switch events give rise to a factorial number of possible permutations of the internal nodes in the host phylogeny. This increases the complexity of the relative ordering of the nodes and varies the possible placement of the take-off and landing edges for a host switch and the resultant cost.

A number of methods have been proposed to handle this complexity. The most common are two methods that solve modified versions of the cophylogeny reconstruction problem using dynamic programming. The first of these approaches is implemented in the tools Tarzan (Merkle & Middendorf 2005) and Core-Pa (Merkle et al. 2010). This approach ignores the orderings of the host nodes so as to allow the best host switch to be recovered in polynomial time. This relaxation, however, has been
shown to produce solutions which are biologically infeasible (Conow et al. 2010).

The second approach is to fix the order of the nodes in the host tree. This approach will be the focus of this paper. This method was proposed by Libeskind-Hadas and Charleston (2009). They proposed a novel solution, where the node ordering is fixed, such that the cophylogeny reconstruction problem can be solved in polynomial time. A meta-heuristic framework is then applied to traverse the factorial combinations of node orderings. A genetic algorithm has since been shown to be particularly successful method for recovering an optimal node ordering. The initial dynamic programming algorithm proposed ran in $O(n^3)$ and was later implemented in the Jane 1 software tool (Conow et al. 2010). This algorithm would store the resultant sub-solution for each subtree at the node (see Implementation). This technique has since been called node mapping to differentiate with alternate algorithms which can solve this problem in $O(n^3)$ time, such as the edge mapping approach proposed by Yodpinyanee et al. (2011) and tree slicing proposed by Doyon, Scornavacca, et al. (2011). The edge mapping algorithm, unlike the initial node mapping algorithm, which was designed to handle phylogenetic networks, are designed to run where both $H$ and $P$ are phylogenetic trees. Edge mapping, however, is the most efficient, as although both have a running time of $O(n^3)$, slicing requires $O(n^3)$ space complexity as opposed to the space complexity of edge mapping, which is $O(n^2)$.

**Problem statement**

A question that follows from this research is whether the original node mapping algorithm, with an $O(n^2)$ running time, is the best implementation that can be achieved for node mapping, and if not, is there an implementation that can uncover a cophylogeny mapping as efficiently in terms of both time and space complexity, compared to edge mapping, for the special case where $H$ and $P$ are phylogenetic trees. Further, can these improvements translate to a more efficient algorithm for recovering cophylogeny reconstructions for phylogenetic networks?

Such an algorithmic improvement would be of value as to date only the original $O(n^3)$ node mapping algorithm has offered any extensibility towards handling phylogenetic networks. Algorithms such as edge mapping, when translated to phylogenetic networks, suffer from complexity increases in the size of the dynamic programming table, as the number of mapping locations increases from $O(n)$ for a tree to $O(n^2)$ for a network, which is not the case for node mapping.

In this paper, we present a number of new optimisation techniques which can be applied to the original node mapping algorithm first presented by Libeskind-Hadas and Charleston, where the worst case running time for recovering a cophylogeny mapping for phylogenetic trees is $O(n^2)$.

**Implementation**

The newly proposed algorithm is designed to solve the fixed node ordering permutation of the cophylogeny reconstruction problem using dynamic programming. It is designed to run in three stages. The first stage is the preprocessing step, which allows for optimal coevolutionary event locations to be recovered using an $O(1)$ time query. The second stage is the construction of the dynamic programming table of coevolutionary events, where the optimal solutions stored are those which minimise the total event cost for the cophylogeny reconstruction. Finally, the third stage of the algorithm uses the constructed dynamic programming table to recover an optimal cophylogeny reconstruction.

The node mapping algorithm is designed to store cophylogeny reconstruction events at the nodes in $V_H$. This approach results in codivergence events mapped to the host node at which they occur, while events that occur on host edges, such as duplication and host switch events, are mapped to the child node of the edge on which they occur.

**Reducing to an $O(n)$ set of sub-solutions**

The node mapping algorithm runs by solving the sub-solution for each internal node in $V_P$ and then using these sub-solutions to build up to the complete solution. A sub-solution is defined below:

**Definition 1** A sub-solution is a cophylogeny mapping ($\Phi$) of subtree $P_i$ of $P$ mapped into $H$.

The first optimisation proposed to reduce the running and space complexity of node mapping, is to reduce the set of sub-solutions stored at each step of the dynamic programming algorithm. Previous node mapping algorithms have stored $O(n^2)$ sub-solutions for each $P_i$, requiring $O(n^2)$ comparisons when processing the parent of sibling sub trees $P_1$ and $P_2$. Rather than considering all $O(n^2)$ possible sub-solutions, this algorithm restricts the set of sub-solutions considered to $O(n)$. This reduction is possible by selecting only a single sub-solution for each host node rather than considering the set of all possible permutations which is of size $O(n)$. That is, for a set of $O(n^2)$ sub-solutions, only $O(n)$ minimal cost sub-solutions are required to guarantee the optimal solution is recovered. The remaining $(n^2 - n)$ sub-solutions are discarded. This can still guarantee the optimal solution is recovered, as all minimum cost sub-solutions are represented in this subset and only sub-solutions with duplicate costs or solutions with worse scores, are discarded.

By reducing the sub-solution stored to $O(n)$ at each pass, we ensure that the parent only has $O(n^2)$ comparisons to process for each node $\in V_P$. As the $O(n^2)$ comparisons are run for each of the $n$ nodes this algorithm requires $O(n^3)$ comparisons in total. The set of solutions retained at each step are stored in a dynamic programming table of size $O(n^2)$. In this table, the rows and the
### Algorithm 1: NodeMapping \((H, P, \rho)\)

1. Let \(H, P\) be the host and parasite trees respectively.
2. Let \(Map\) be the two-dimensional dynamic programming table.
3. Let \(L\) be a list of nodes in \(P\).
4. \(L \leftarrow P\).
5. Sort the nodes in \(L\) by their distances from the root of \(P\).
6. For each node \(p_i \in L\): if \(p_i\) is a leaf, then mark it.
7. For each node \(p_i \in L\): for each node \(p_j \in Map[p_i]\): Map\(p_i\)[AddMinCost(AllocCodivergence\((h_i, h_j))\)] Map\(p_i\)[AddMinCost(AllocDuplication\((h_i, h_j))\)] Map\(p_i\)[AddMinCost(AllocHostSwitch\((h_i, h_j))\)].

**Figure 2.** This is the algorithm that constructs a dynamic programming table for node mapping in \(O(n^2)\). It calls a series of methods including three ‘Opt’ event functions (AllocCodivergence, AllocDuplication, AllocHostSwitch) which query the preprocessed events. The fourth function is the AddMinCost function which ensures that only the minimum cost reconstruction for each host node is retained. As this function only compares two values at a time it runs in \(O(1)\).

Columns represent parasite and host nodes, respectively. Each column of this table is constructed using the best sub-solutions of \(p_i \in V_P\). If \(p_i\) does not have a valid mapping to a node \(h_j\), then this column entry is marked invalid and is not considered when processing the parent of \(p_i\). The algorithm for the construction of this table is described in Figure 2.

To recover the minimum cost mapping using the dynamic programming table requires two \(O(n)\) algorithms. First, the minimum cost entry \(MIN(\rho)\) must be recovered, where \(\rho\) is the root of \(P\). Starting with \(MIN(\rho)\), the second algorithm makes \(n\) recursive calls back to the leaves of \(P\), \(p_{n-1} \ldots p_{2n-2}\). This reconstruction runs after the \(O(n^2)\) algorithm to construct the dynamic programming table, and therefore does not impact on the worst case number of comparisons.

This optimisation of the dynamic programming table relies on the assertion that the number of events considered is a constant factor. As \(p_i\) is mapped into \(H\), only \(O(1)\), i.e. a constant number, of optimal events may be recovered for codivergence, duplication and host switch events. This is achieved due to preprocessing and a constant bound of the number of events considered, which are both discussed later.

**Reducing the host switch set size**

The second optimisation that is applied to the node mapping algorithm is to reduce the number of possible host switch events that are considered. This is to overcome the exponential, \(O(2^{\sqrt{n}})\), number of locations possible even in the fixed node ordering case if no reduction is enforced.

Of this set of permutations, the original node mapping algorithm considers \(O(n^2)\) possible locations for host switches for every set of sub-solutions processed. Using this reduced set of sub-solution results in \(O(n^2)\) comparisons to reconstruct the dynamic programming table, which, although an improvement over existing node mapping, is still significantly worse than the existing algorithms running in \(O(n^3)\).

To reduce the host switch location’s solution space for node mapping, a reduction of the search space for optimal switch locations is required. We give a definition for the optimal host switch location which is used to reduce the host switch solution set as:

**Definition 2** Suppose \(p \in V_P\) has children \(p_{left}\) and \(p_{right}\), with \(\Phi(p_{left}) = u\) and \(\Phi(p_{right}) = v\). Then an optimal host switch location for \(\Phi(p)\) is a pair of edges \(e_i\) and \(e_j\) \(\in E_{H}\), where the total number of loss events between \((e_i, u)\) and between \((e_j, v)\) is minimal, and where \(e_i\) and \(e_j\) are edges which share a common interval when considering their distances from the root of \(V_H\).

This definition does not allow for the recovery of all optimal host switches but does always uncover an optimal host switch instance of the \(O(n)\) possible optimal solutions. The recovered host switch for \(\Phi(p)\) minimises the number of loss events between its children which is always optimal. To ensure all optimal sub-solutions are considered, both host switches where \(e_i\) and \(e_j\) are the take-off and landing edges must be computed as potential sub-solutions. This multiplies the solutions space by a constant factor and therefore does not increase the overall complexity.

Using this method, it is possible to reduce the \(O(n^2)\) set of host switch locations considered using the previous implementation down to \(O(1)\). As a result of this optimisation technique, only four items need to be recovered for each node \(p_i \in V_P\): the optimal codivergence, duplication, host switch from \(e_i\) to \(e_j\), and that from \(e_j\) to \(e_i\). Therefore, the assignment of events for each node \(p_i \in V_P\) is constant, assuming each event is found in \(O(1)\). This result maintains the reduced running time for construction of the dynamic programming table in \(O(n^2)\), assuming the optimal codivergence, duplication and host switch locations can be found in \(O(1)\), which is described in the following section.

**Applying preprocessing to cache recoverable events**

As previously noted, \(O(n^3)\) comparisons are required for the construction of the dynamic programming table, but the actual running time cost of each comparison has not yet been discussed. In this section, we show that all comparisons can be reduced to \(O(1)\) by preprocessing \(H\). The aim, therefore, is for each comparison to recover the minimum cost codivergence, duplication and host switch event for a pair of children \((p_i, p_j) \in V_P\) mapped to \(H\) using an \(O(1)\) query.
The minimal cost codivergence and duplication locations are the node and parent edge, respectively, of their children’s lowest common ancestor (Doyon, Ranwez et al. 2011). These locations can be uncovered using an $O(1)$ query provided preprocessing has been performed. The lowest common ancestor preprocessing for an $O(1)$ query requires an $O(n)$ time pass over the tree with a space requirement of $O(n)$ (Harel & Tarjan 1984).

The number of loss events between a pair of parasite nodes mapped into $H$ is the number of host nodes separating the pair. This can be queried in $O(1)$ by storing the number of nodes between each node $\in V_H$ and the root. This is then subtracted when executing the $O(1)$ query.

The optimal host switch location is the pair of edges that overlap, such that the distance to their mapped children is minimised. To allow this edge pair to be recovered for each edge pair $\in E_H$ using a query time of $O(1)$, requires $O(n)$ time and space preprocessing. This result is possible by leveraging Bender and Farach-Colton’s (2004) solution to the level ancestor problem. This allows for a $O(1)$ query for recovering the take-off and landing edges for the host switch as the level ancestor problem allows for a node to be recovered at a set distance from the root in any subtree in $O(1)$ time. Consider a pair of edges $e_i = (a, b)$ and $e_j = (x, y)$ where $a$ and $x$ are the ancestral nodes and $a$ is the node of this pair which is closest to the root. Let the distance from the root to node $a$ be $d$, and the query node $u$ to be $x$. This construction provides the query information in the form $q(u, d)$ as required by the level ancestor problem. The node returned from the query is the parent of the edge required to complete the (take-off, landing) edge pair. Therefore, the host switch edge pair can be recovered in $O(1)$ using preprocessing of $O(n)$.

By applying preprocessing, the running time of this algorithm is $O(n^3)$, and has a space requirement of $O(n^2)$, which matches the running time and space requirement for the best known algorithm for this problem (Yodpinyanee et al. 2011). Further to the authors’ knowledge, the solution described in this paper provides the first $O(n)$ time and space complexity for preprocessing algorithm for the fixed node ordering permutation of the cophylogeny reconstruction problem.

Results

Comparison of results using real data

To ensure the consistency of the cophylogeny reconstructions reported by the node mapping algorithm once the proposed optimisations have been applied, we applied our approach on a number of well-studied coevolutionary cases. The validation data-set contains a number of problems which have previously been used to verify algorithms, and the complete set of instances was used by Conow et al. (2010) to validate Jane’s performance against Tarzan. The validation set also offers a variety of coevolutionary scenarios including herbivore and plant interactions (Percy et al. 2004), mutualism and symbiosis (Jackson 1999) and host–parasite systems (Hafner & Nadler 1988) to ensure this algorithm is sufficiently general.

We utilised three independent event cost schemes, based on previous research, to ensure that this algorithm is able to recover Pareto optimal solutions for more than just a single event cost scheme. Event cost vectors were constructed for each event cost scheme selected. These vectors are described in the form $v_{cost} = (c, d, h, l)$ where $c$, $d$, $h$ and $l$ represent the penalty costs associated with codivergence duplication, host switch and loss, respectively. The event cost schemes included in this analysis are those previously introduced including:

(1) Maximising Codivergence ($-1, 0, 0, 0$): Under this cost scheme, reconstructions match the solution set reported by cost maximisation algorithms (Ronquist 1998, 2002).

(2) Jungle ($0, 1, 1, 1$): Under this cost scheme, all events are assigned the same cost, except for codivergences, which are assigned a cost of 0. This is to promote the selection of codivergences events similar to the maximising codivergence cost scheme, though in contrast, also assigns a penalty to other coevolutionary events to prevent overcounting of such events (Ronquist 2002).

(3) Reconciliation ($1, 1, \infty, 1$): Under this cost scheme, the reconstruction recovered is the same as the Pareto optimal solution uncovered by Page’s Reconciled Tree Analysis (Page 1994a, 1994b; Ronquist 2002).

It is important to note that all three cost schemes, including reconciliation which does not permit switch events as indicated by the $\infty$ cost for host switch events is able to recover biologically feasible solutions for all problem instances.

For consistency, in our analysis, we applied host switch costs as the cost of a host switch plus the additional cost of a duplication event, handling both the duplication and switch events that arise as part of a host switch, as a single event. This was enforced to match the default behaviour of the Jane software series, along with other existing packages in this field (Merkle & Middendorf 2005; Conow et al. 2010; Merkle et al. 2010).

The algorithms selected for this comparison included first the existing node mapping algorithm implemented in Jane 1 (Libeskind-Hadas 2010), which is expected to produce the same solutions set as the newly proposed algorithm, although taking more time to do so and secondly, the most recent implementation of edge mapping implemented in Jane 4 (Libeskind-Hadas 2013), which should take approximately the same amount of time as the new node mapping algorithm.

The comparison of the six real data-sets using all three event cost vectors can be seen in Table 1. This table shows the results achieved for all six test cases and
for all three cost schemes. In all cases, the result recovered by both ‘flavours’ of Jane matched the new node mapping algorithm, validating its performance as a method for recovering cophylogenetic reconstructions.

Comparing actual running time
The new node mapping algorithm was implemented in Java as was Jane 1 and Jane 4. To evaluate the actual execution speed of each algorithm, the largest real data-set, host plants (*Leguminosae*) and phytophagous insects (*Psylloidae*) (Percy et al. 2004), was selected as the baseline.

Two running time evaluations were carried out on a standard PC with an i5-2400 3.1 GHz CPU and 8 GB of RAM. First, the original node mapping algorithm was compared against the new node mapping algorithm where both were run for all numbers of iterations from 1 to 100, with 100 replicates each, giving 10,000 samples. The second evaluation was to run the existing edge mapping algorithm against the newly proposed node mapping algorithm for 10,000 iterations, repeated 1000 times.

The results of these two simulations are captured in Figure 3 and Table 2. The original node mapping algorithm was excluded from the second experiment as we estimated that it would have taken approximately half a year of computation to execute the simulation, and because of the imposed constraints of Jane 1, which only allows up to 200 iterations for each data-set. Figure 3 demonstrates the significant running time improvement of the newly proposed approach for cophylogeny reconstruction.

There is, however, a slightly slower running time for the newly proposed node mapping algorithm, compared with the implementation in Jane 4. This margin is a factor of 1.35 (when comparing the median). With further optimisation within the code base, we believe that it is possible to reduce this further to achieve parity with the running time achieved by Jane 4, which has gone through a number of revisions to achieve its current running time.
Discussion and analysis

These results demonstrate that the performance of the newly proposed node mapping algorithm is comparable to the existing edge mapping algorithm’s implementation. It further demonstrates a significant performance improvement over the existing node mapping algorithm, demonstrated by the proposed algorithm taking less than 1 s to perform the same operation that took 300 s by the previous implementation. It is important to note that this result is the case where both algorithms were run for 100 iterations. In fact, both algorithms use 100 populations for 100 iterations each, to ensure with a high probability that they converge on an optimal solution using a genetic algorithm. Therefore, each algorithm requires solving 10,000 instances to recover an optimal solution, which is a difference between solving this problem in approximately 100 s using the newly proposed algorithm or approximately 8.33 h using Jane 1. Further, it is expected that data-sets will continue to increase in size compared to this data-set. One recently constructed system, fig trees and their pollinating wasps (Cruaud et al. 2011), contains 5.5 times the number of parasite leaf nodes as the case presented here. Based on the previous performance data, it would be expected to require two days to complete, compared to approximately 10 min for the proposed algorithm. Data-sets in this field will only continue to increase in size. Therefore, it is essential that algorithms, such as node mapping, are optimised to reduce running time performance while maintaining the same level of accuracy, as has been achieved here.

These results also indicate the potential of node mapping algorithms to be applied further to the research of cophylogenetic reconstruction of phylogenetic networks. Based on the analysis of the best edge mapping and node mapping algorithms, edge mapping is marginally faster when processing phylogenetic trees, but if this newly proposed technique was expanded to phylogenetic networks, a significant running time improvement would be achieved. This is due to edge mapping requiring up to $O(n^4)$ entries in a dynamic programming table because of the increase in the number of edges in the worst case for phylogenetic networks of $O(n^2)$. This is in contrast to the newly proposed node mapping algorithm, where there is no need to increase the required size of the dynamic programming table, as there are $n$ nodes in both phylogenetic trees and networks, and therefore storage required for mapping locations does not increase. The additional cost for algorithms handling phylogenetic networks would be incurred in the selection of the best codivergence, duplication and host switch event locations, which still remains an open question for this problem, when considering phylogenetic networks.

Further work

Along with extending this algorithm to handle phylogenetic networks, there are a number of areas for further research involving this newly proposed algorithm. One area is expanding this algorithm to consider cases where a parasite leaf node can have more than one association. The proposed algorithm can be used to handle this problem without an expansion of the dynamic programming table. Currently, all parasite leaf nodes have only a single entry in their respective rows; the mapping to a single host leaf node. For the case where parasites are permitted to have multiple associations, this would be replaced with up to $n$ elements which need to be stored in the dynamic programming table for each parasite leaf node depending on the recoverable events used to interpret this permutation of the problem. Therefore, the worst case space complexity would not change as a result of handling widespread (multi-host) parasites using this newly proposed algorithm, and therefore an algorithm for solving this problem where widespread (multi-host) parasites are permitted must exist with a $O(n^2)$ space complexity.

Also, there is potential to reduce the running time of this solution further, by reducing the sub-solution set. Consider the case where it is possible to reduce the size of the sub-solution set maintained to $O(\sqrt{n})$, $O(\log n)$ or $O(1)$: this would result in algorithms with respective running times of $O(n^2)$, $O(n \log^2 n)$ or $O(n)$. By applying the currently defined algorithm, these sub-solution sets would no longer be able to guarantee optimal solutions for the fixed node ordering instance, but by applying a heuristic sampling technique in the selection of sub-solutions, it is possible that the respective algorithms may recover good solutions in many biological cases, much faster. This approach, although not guaranteeing optimal solutions, would in principle, be able to recover all optimal solutions, with high probability, provided sufficient repetitions are performed when applying random sampling. Therefore, it is believed that a trial within a meta-heuristic framework would be worthwhile.

Conclusion

This work presents a number of new optimisations for the existing node mapping algorithm proposed by Libeskind-Hadas and Charleston. It applies several techniques to reduce the time and space complexity of this well-known problem to provide the fastest implementation using node mapping to date to solve the cophylogeny reconstruction problem where the host tree’s node ordering is fixed. Along with an improved complexity performance for node mapping, this paper offers a reduction to the preprocessing complexity to $O(n)$ for time and space complexity. Further, we have presented a formulation for a faster method for recovering cophylogeny mappings for phylogenetic networks. Finally, this work offers future avenues for improving the complexity of the node mapping algorithm, allowing for even faster reconstruction methods and proposes a method to apply this algorithm to the cophylogeny reconstruction problem where widespread parasite associations are permitted.
Availability and requirements

The newly proposed algorithm presented in this paper is available at http://sydney.edu.au/engineering/it/~mcharles/ and runs on any machine with Java 1.6 or higher.

Competing interests

The authors declare that they have no competing interests.

Funding

This work was supported by an Australian Postgraduate Award to BD and an Australian Research Council Grant [grant number DP1094891] to MAC.

References

Appendix 1.

In the following appendix, the algorithm proposed by Libeskind-Hadas and Charleston and the newly proposed algorithm are manually run over a simple tanglegram, where each step is recorded and then compared to demonstrate the improvements offered by the new algorithm. Each algorithm is run on the tanglegram in Figure 1. The following definitions are used to describe the current state of the algorithms at each step.

Definition 3 An image \( i \) is a possible mapping \( (\Phi) \) for a particular parasite node \( n \in P \) where each node \( n \) has a set of images \( S \) and where \( S \neq \emptyset \).

Definition 4 \( n_i \) is the \( i \)th image of the set of images \( S \) for the parasite node \( n \) which is written in the form \( n_i(p_j, q_k) \) where \( p \) and \( q \) are the two children of \( n \) and \( p_j \) is the \( j \)th image of \( p \) and \( q_k \) is the \( k \)th image of \( q \).

Definition 5 \( S_r \) is the reduced set of images, which is the result of a filter applied by a Node Mapping algorithm such that \( S_r \subseteq S \) and \( S_r \neq \emptyset \).

These three definitions allow us to describe each state of both dynamic programming algorithms. For each node \( n \in P \) we will generate a list of all possible images and will filter the set of recorded images using the processes defined for each algorithm to generate the set \( S_r \), the states used to incrementally build the final mapping for the parasite tree.

When describing the state as an image, there are three events which can occur: a codivergence, duplication or a host switch. Depending on the location of these events, this may give rise to a number of loss events. The format we use for each event is \( a - \text{codivergence}, (a,b) - \text{duplication and } ((a,b),(c,d)) - \text{host switch.} \) In the case of a host switch, the edge \((a,b)\) is the starting edge and \((c,d)\) is the landing edge. Also each event is assigned an event cost. Codivergence is assigned a cost of 2, duplication and loss a cost of 1 and host switch a cost of 0, in line with the Jungle cost scheme (Ronquist 2002). This leads to the following notation to describe the state of each algorithm for each iteration.

The previous line shows the first image of the node \( w \) generated from the first images of its two children \( s \) and \( t \) \((s_1 \text{ and } t_1, \text{ respectively})\). The mapping is a host switch from the edge \((b,e)\) to the edge \((c,f)\) in the host tree, and which is assigned a cost of 2. As this is the first mapping of an internal edge, the cost vector only includes a host switch although this will be accumulated as we build up the solution. The form of the cost vector \( (0, 0, 1, 0) \) is the number of codivergence, duplication, host switch and loss events in the current image, respectively.

The results aim to adhere the following lemma which is derived from the implementation details of the new algorithm. This lemma is important as if this property is not preserved the new algorithm is unable to recover the optimal solution in \( O(n^3) \).

Lemma 1 The newly proposed algorithm ensures that the size of \( S_r \) is bound by the number of nodes in \( H \). Proof in main text.

The following results show that both algorithms recover the optimal solution. The newly proposed algorithm, however, only requires 42 image states to be explored as opposed to the previous algorithm which requires 131. Further, this result confirms the assertion that the newly proposed algorithm, is able to recover the optimum solution while bounding the size of \( S_r \) to the number of nodes in \( H \) which in this case is 7. This is shown by size of \( S_r \) for \( y \) and \( z \) which is 6 and 7, respectively, as opposed to the alternate algorithm where the size of the retained set \( S_r \) for both \( y \) and \( z \) are 12 and 13, respectively.

Figure 4. A coevolutionary relationship represented using the host and parasite phylogenies first shown in Figure 1. In this case, each node in the host and parasite tree is labelled to assist in the reconstruction described in Appendix 1. Of note node \( b \) has been fixed to a timing interval before \( c \) which leads to the reconstruction described in Appendix 1 and an edge has been added to the root of \( H \) to map all parasite divergence events which occurred prior to this phylogeny.

\[ w_1(s_1, t_1) = ((b, e), (c, f)), \text{ cost of } 2, (0, 0, 1, 0) \]
Solving Figure 4 in $O(n^7)$

\[
\begin{align*}
  r_1 &= d \\
  s_1 &= e \\
  t_1 &= f \\
  u_1 &= g \\
  v_1 &= g \\
  w_1(s_1, t_1) &= ((b, e), (c, f)), \text{ cost of } 2, (0, 0, 1, 0) \\
  w_2(s_1, t_1) &= ((c, f), (b, e)), \text{ cost of } 2, (0, 0, 1, 0) \\
  w_3(s_1, t_1) &= ((b, e), (a, c)), \text{ cost of } 3, (0, 0, 1, 1) \\
  w_4(s_1, t_1) &= ((a, c), (b, e)), \text{ cost of } 3, (0, 0, 1, 1) \\
  w_5(s_1, t_1) &= ((a, b), (a, c)), \text{ cost of } 4, (0, 0, 1, 2) \\
  w_6(s_1, t_1) &= ((a, c), (a, b)), \text{ cost of } 4, (0, 0, 1, 2) \\
  w_7(s_1, t_1) &= a, \text{ cost of } 2, (1, 0, 0, 2) \\
  w_8(s_1, t_1) &= (a', a), \text{ cost of } 5, (0, 1, 0, 4) \\
  x_1(u_1, v_1) &= (c, g), \text{ cost of } 1, (0, 1, 0, 0) \\
  x_2(u_1, v_1) &= (a, c), \text{ cost of } 3, (0, 1, 0, 2) \\
  x_3(u_1, v_1) &= (a', a), \text{ cost of } 5, (0, 1, 0, 4) \\
  y_1(w_1, x_1) &= ((b, e), (c, g)), \text{ cost of } 5, (0, 1, 2, 0) \\
  y_2(w_1, x_1) &= ((c, g), (b, e)), \text{ cost of } 5, (0, 1, 2, 0) \\
  y_3(w_1, x_1) &= ((b, e), (a, c)), \text{ cost of } 6, (0, 1, 2, 1) \\
  y_4(w_1, x_1) &= ((a, c), (b, e)), \text{ cost of } 6, (0, 1, 2, 1) \\
  y_5(w_1, x_1) &= ((a, b), (a, c)), \text{ cost of } 7, (0, 1, 2, 2) \\
  y_6(w_1, x_1) &= ((a, c), (a, b)), \text{ cost of } 7, (0, 1, 2, 2) \\
  y_7(w_1, x_1) &= a, \text{ cost of } 5, (1, 1, 1, 2) \\
  y_8(w_1, x_1) &= (a', a), \text{ cost of } 8, (0, 2, 1, 4)
\end{align*}
\]
\[ y_{9}(w_{1}, x_{2}) = ((b, c), (a, c)), \text{ cost of 7, } (0, 1, 2, 2) \]
\[ y_{10}(w_{1}, x_{2}) = ((a, c), (b, c)), \text{ cost of 7, } (0, 1, 2, 2) \]
\[ y_{11}(w_{1}, x_{2}) = ((a, b), (a, c)), \text{ cost of 8, } (0, 1, 2, 3) \]
\[ y_{12}(w_{1}, x_{2}) = ((a, c), (a, b)), \text{ cost of 8, } (0, 1, 2, 3) \]
\[ y_{13}(w_{1}, x_{2}) = a, \text{ cost of 6, } (1, 1, 1, 3) \]
\[ y_{14}(w_{1}, x_{2}) = (a', a), \text{ cost of 9, } (0, 2, 1, 5) \]
\[ y_{15}(w_{1}, x_{3}) = (a', a), \text{ cost of 10, } (0, 2, 1, 6) \]
\[ y_{16}(w_{2}, x_{1}) = ((c, f), (c, g)), \text{ cost of 5, } (0, 1, 2, 0) \]
\[ y_{17}(w_{2}, x_{1}) = ((c, g), (c, f)), \text{ cost of 5, } (0, 1, 2, 0) \]
\[ y_{18}(w_{2}, x_{1}) = c, \text{ cost of 3, } (1, 1, 1, 0) \]
\[ y_{19}(w_{2}, x_{1}) = (a, c), \text{ cost of 5, } (0, 2, 1, 2) \]
\[ y_{20}(w_{2}, x_{1}) = (a', a), \text{ cost of 7, } (0, 2, 1, 4) \]
\[ y_{21}(w_{2}, x_{2}) = (a, c), \text{ cost of 6, } (0, 2, 1, 2) \]
\[ y_{22}(w_{2}, x_{2}) = (a', a), \text{ cost of 8, } (0, 2, 1, 4) \]
\[ y_{23}(w_{2}, x_{3}) = (a', a), \text{ cost of 10, } (0, 2, 1, 6) \]
\[ y_{24}(w_{3}, x_{1}) = ((b, e), (a, c)), \text{ cost of 7, } (0, 1, 2, 2) \]
\[ y_{25}(w_{3}, x_{1}) = ((a, c), (b, e)), \text{ cost of 7, } (0, 1, 2, 2) \]
\[ y_{24}(w_{3}, x_{1}) = ((a, b), (a, c)), \text{ cost of 8, } (0, 1, 2, 3) \]
\[ y_{25}(w_{3}, x_{1}) = ((a, c), (a, b)), \text{ cost of 8, } (0, 1, 2, 3) \]
\[ y_{26}(w_{3}, x_{1}) = a, \text{ cost of 6, } (1, 1, 1, 3) \]
\[ y_{27}(w_{3}, x_{1}) = (a', a), \text{ cost of 9, } (0, 2, 1, 5) \]
\[ y_{28}(w_{3}, x_{2}) = ((b, e), (a, c)), \text{ cost of 8, } (0, 1, 2, 3) \]
\[ y_{29}(w_{3}, x_{2}) = ((a, c), (b, e)), \text{ cost of 8, } (0, 1, 2, 3) \]
\[ y_{30}(w_{3}, x_{2}) = ((a, b), (a, c)), \text{ cost of 9, } (0, 1, 2, 4) \]
\[ y_{31}(w_{3}, x_{2}) = ((a, c), (a, b)), \text{ cost of 9, } (0, 1, 2, 4) \]
\[
\begin{align*}
y_{32}(w_3, x_2) &= a, \text{ cost of } 7, (1, 1, 1, 4) \\
y_{33}(w_3, x_2) &= (a', a), \text{ cost of } 10, (0, 2, 1, 6) \\
y_{34}(w_3, x_3) &= (a', a), \text{ cost of } 11, (0, 2, 1, 7) \\
y_{35}(w_4, x_1) &= (a, c), \text{ cost of } 6, (0, 2, 1, 2) \\
y_{36}(w_4, x_1) &= (a', a), \text{ cost of } 8, (0, 2, 1, 4) \\
y_{37}(w_4, x_2) &= (a, c), \text{ cost of } 7, (0, 2, 1, 3) \\
y_{38}(w_4, x_2) &= (a', a), \text{ cost of } 9, (0, 2, 1, 5) \\
y_{39}(w_4, x_3) &= (a', a), \text{ cost of } 10, (0, 2, 1, 6) \\
y_{40}(w_5, x_1) &= ((a, b), (a, c)), \text{ cost of } 8, (0, 1, 2, 3) \\
y_{41}(w_5, x_1) &= ((a, c), (a, b)), \text{ cost of } 8, (0, 1, 2, 3) \\
y_{42}(w_5, x_1) &= a, \text{ cost of } 7, (1, 1, 1, 4) \\
y_{43}(w_5, x_1) &= (a', a), \text{ cost of } 10, (0, 2, 1, 6) \\
y_{44}(w_5, x_2) &= ((a, b), (a, c)), \text{ cost of } 9, (0, 1, 2, 4) \\
y_{45}(w_5, x_2) &= ((a, c), (a, b)), \text{ cost of } 9, (0, 1, 2, 4) \\
y_{46}(w_5, x_2) &= a, \text{ cost of } 8, (1, 1, 1, 5) \\
y_{47}(w_5, x_2) &= (a', a), \text{ cost of } 11, (0, 2, 1, 7) \\
y_{48}(w_5, x_3) &= (a', a), \text{ cost of } 11, (0, 2, 1, 7) \\
y_{49}(w_6, x_1) &= (a, c), \text{ cost of } 7, (0, 2, 1, 3) \\
y_{50}(w_6, x_1) &= (a', a), \text{ cost of } 9, (0, 2, 1, 5) \\
y_{51}(w_6, x_2) &= (a, c), \text{ cost of } 8, (0, 2, 1, 4) \\
y_{52}(w_6, x_2) &= (a', a), \text{ cost of } 10, (0, 2, 1, 6) \\
y_{53}(w_6, x_3) &= (a', a), \text{ cost of } 12, (0, 2, 1, 8) \\
y_{54}(w_7, x_1) &= (a', a), \text{ cost of } 6, (1, 2, 0, 4) \\
y_{55}(w_7, x_2) &= (a', a), \text{ cost of } 7, (1, 2, 0, 5) \\
y_{56}(w_7, x_3) &= (a', a), \text{ cost of } 9, (1, 2, 0, 7)
\end{align*}
\]
Not all the images of \( y \) are required for the computation of the images of \( z \). In fact, the algorithm only needs Pareto Set of mappings to compute the images of \( z \). Therefore the final set (the retained set \( S' \)) of images for \( y \) required for generating the mappings for \( z(y, r) \) is \( S' = (y_1, y_2, y_3, y_4, y_5, y_6, y_7, y_{16}, y_{17}, y_{18}, y_{19}, y_{50}) \), which can easily be seen as the dominating images based on cost. Ties are broken by taking the first mapping with the lowest cost.

\[
\begin{align*}
y_{53}(w_8, x_1) & = (a', a), \text{ cost of } 9, \ (0, 3, 0, 6) \\
y_{54}(w_8, x_2) & = (a', a), \text{ cost of } 10, \ (0, 3, 0, 7) \\
y_{55}(w_8, x_3) & = (a', a), \text{ cost of } 12, \ (0, 3, 0, 9) \\
\end{align*}
\]

\[
\begin{align*}
z_1(y_1, r_1) & = ((b, e), (b, d)), \text{ cost of } 7, \ (0, 1, 3, 0) \\
z_2(y_1, r_1) & = ((b, d), (b, e)), \text{ cost of } 7, \ (0, 1, 3, 0) \\
z_3(y_1, r_1) & = b, \text{ cost of } 5, \ (1, 1, 2, 0) \\
z_4(y_1, r_1) & = (a, b), \text{ cost of } 8, \ (0, 2, 2, 2) \\
z_5(y_1, r_1) & = (a', a), \text{ cost of } 10, \ (0, 2, 2, 4) \\
z_6(y_2, r_1) & = ((c, g), (b, d)), \text{ cost of } 7, \ (0, 1, 3, 0) \\
z_7(y_2, r_1) & = ((b, d), (c, g)), \text{ cost of } 7, \ (0, 1, 3, 0) \\
z_8(y_2, r_1) & = ((a, c), (b, d)), \text{ cost of } 8, \ (0, 1, 3, 1) \\
z_9(y_2, r_1) & = ((b, d), (a, c)), \text{ cost of } 8, \ (0, 1, 3, 1) \\
z_{10}(y_2, r_1) & = ((a, c), (a, b)), \text{ cost of } 9, \ (0, 1, 3, 2) \\
z_{11}(y_2, r_1) & = ((a, b), (a, c)), \text{ cost of } 9, \ (0, 1, 3, 2) \\
z_{12}(y_2, r_1) & = a, \text{ cost of } 7, \ (1, 1, 2, 2) \\
z_{13}(y_2, r_1) & = (a', a), \text{ cost of } 10, \ (0, 2, 2, 4) \\
z_{14}(y_3, r_1) & = ((b, e), (b, d)), \text{ cost of } 8, \ (0, 1, 3, 1)
\end{align*}
\]
\[ z_{15}(y_3, r_1) = ((b, d), (b, c)), \text{ cost of } 8, \ (0, 1, 3, 1) \]
\[ z_{16}(y_3, r_1) = b, \text{ cost of } 6, \ (1, 1, 2, 1) \]
\[ z_{17}(y_3, r_1) = (a, b), \text{ cost of } 9, \ (0, 2, 2, 3) \]
\[ z_{18}(y_3, r_1) = (a', a), \text{ cost of } 11, \ (0, 1, 2, 5) \]
\[ z_{19}(y_4, r_1) = ((a, c), (b, d)), \text{ cost of } 8, \ (0, 1, 3, 1) \]
\[ z_{20}(y_4, r_1) = ((b, d), (a, c)), \text{ cost of } 8, \ (0, 1, 3, 1) \]
\[ z_{21}(y_4, r_1) = ((a, c), (a, b)), \text{ cost of } 10, \ (0, 1, 3, 3) \]
\[ z_{22}(y_4, r_1) = ((a, b), (a, c)), \text{ cost of } 10, \ (0, 1, 3, 3) \]
\[ z_{23}(y_4, r_1) = a, \text{ cost of } 8, \ (1, 1, 2, 3) \]
\[ z_{24}(y_4, r_1) = (a', a), \text{ cost of } 11, \ (0, 2, 2, 5) \]
\[ z_{25}(y_5, r_1) = (a, b), \text{ cost of } 10, \ (0, 2, 2, 4) \]
\[ z_{26}(y_5, r_1) = (a', a), \text{ cost of } 12, \ (0, 1, 2, 6) \]
\[ z_{27}(y_6, r_1) = ((a, c), (a, b)), \text{ cost of } 11, \ (0, 1, 3, 4) \]
\[ z_{28}(y_6, r_1) = ((a, b), (a, c)), \text{ cost of } 11, \ (0, 1, 3, 4) \]
\[ z_{29}(y_6, r_1) = a, \text{ cost of } 9, \ (1, 1, 2, 4) \]
\[ z_{30}(y_6, r_1) = (a', a), \text{ cost of } 12, \ (0, 2, 2, 6) \]
\[ z_{31}(y_7, r_1) = (a', a), \text{ cost of } 8, \ (1, 2, 1, 4) \]
\[ z_{32}(y_{16}, r_1) = ((c, f), (b, d)), \text{ cost of } 7, \ (0, 1, 3, 0) \]
\[ z_{33}(y_{16}, r_1) = ((b, d), (c, f)), \text{ cost of } 7, \ (0, 1, 3, 0) \]
\[ z_{34}(y_{16}, r_1) = ((a, c), (b, d)), \text{ cost of } 8, \ (0, 1, 3, 1) \]
\[ z_{35}(y_{16}, r_1) = ((b, d), (a, c)), \text{ cost of } 8, \ (0, 1, 3, 1) \]
\[ z_{36}(y_{16}, r_1) = ((a, c), (a, b)), \text{ cost of } 9, \ (0, 1, 3, 2) \]
\[ z_{37}(y_{16}, r_1) = ((a, b), (a, c)), \text{ cost of } 9, \ (0, 1, 3, 2) \]
\[ z_{38}(y_{16}, r_1) = a, \text{ cost of } 7, \ (1, 1, 2, 2) \]
\[ z_{39}(y_{16}, r_1) = (a', a), \text{ cost of } 10, \ (0, 2, 2, 4) \]
<table>
<thead>
<tr>
<th>Index</th>
<th>Equation</th>
<th>Cost</th>
<th>Set Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>$z_{40}$</td>
<td>$y_{17}, r_1 = ((c, g), (b, d))$</td>
<td>7</td>
<td>(0, 1, 3, 0)</td>
</tr>
<tr>
<td>$z_{41}$</td>
<td>$y_{17}, r_1 = ((b, d), (c, g))$</td>
<td>7</td>
<td>(0, 1, 3, 0)</td>
</tr>
<tr>
<td>$z_{42}$</td>
<td>$y_{17}, r_1 = ((a, c), (b, d))$</td>
<td>8</td>
<td>(0, 1, 3, 1)</td>
</tr>
<tr>
<td>$z_{43}$</td>
<td>$y_{17}, r_1 = ((b, d), (a, c))$</td>
<td>8</td>
<td>(0, 1, 3, 1)</td>
</tr>
<tr>
<td>$z_{44}$</td>
<td>$y_{17}, r_1 = ((a, c), (a, b))$</td>
<td>9</td>
<td>(0, 1, 3, 2)</td>
</tr>
<tr>
<td>$z_{45}$</td>
<td>$y_{17}, r_1 = ((a, b), (a, c))$</td>
<td>9</td>
<td>(0, 1, 3, 2)</td>
</tr>
<tr>
<td>$z_{46}$</td>
<td>$y_{17}, r_1 = a$, cost of 7, (1, 1, 2, 2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$z_{47}$</td>
<td>$y_{17}, r_1 = (a', a)$, cost of 10, (0, 2, 2, 4)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$z_{48}$</td>
<td>$y_{18}, r_1 = ((a, c), (b, d))$</td>
<td>5</td>
<td>(1, 1, 2, 0)</td>
</tr>
<tr>
<td>$z_{49}$</td>
<td>$y_{18}, r_1 = ((b, d), (a, c))$</td>
<td>5</td>
<td>(1, 1, 2, 0)</td>
</tr>
<tr>
<td>$z_{50}$</td>
<td>$y_{18}, r_1 = ((a, c), (a, b))$</td>
<td>6</td>
<td>(1, 1, 2, 1)</td>
</tr>
<tr>
<td>$z_{51}$</td>
<td>$y_{18}, r_1 = ((a, b), (a, c))$</td>
<td>6</td>
<td>(1, 1, 2, 1)</td>
</tr>
<tr>
<td>$z_{52}$</td>
<td>$y_{18}, r_1 = a$, cost of 4, (2, 1, 1, 1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$z_{53}$</td>
<td>$y_{18}, r_1 = (a', a)$, cost of 7, (1, 2, 1, 3)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$z_{54}$</td>
<td>$y_{19}, r_1 = ((a, c), (b, d))$</td>
<td>8</td>
<td>(0, 2, 2, 2)</td>
</tr>
<tr>
<td>$z_{55}$</td>
<td>$y_{19}, r_1 = ((b, d), (a, c))$</td>
<td>8</td>
<td>(0, 2, 2, 2)</td>
</tr>
<tr>
<td>$z_{56}$</td>
<td>$y_{19}, r_1 = ((a, c), (a, b))$</td>
<td>9</td>
<td>(0, 2, 2, 3)</td>
</tr>
<tr>
<td>$z_{57}$</td>
<td>$y_{19}, r_1 = ((a, b), (a, c))$</td>
<td>9</td>
<td>(0, 2, 2, 3)</td>
</tr>
<tr>
<td>$z_{58}$</td>
<td>$y_{19}, r_1 = a$, cost of 7, (1, 2, 1, 3)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$z_{59}$</td>
<td>$y_{19}, r_1 = (a', a)$, cost of 10, (0, 3, 1, 5)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$z_{60}$</td>
<td>$y_{19}, r_1 = (a', a)$, cost of 9, (1, 3, 0, 6)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Of this set of images for $z$ only $S' = \{z_1, z_2, z_4, z_5, z_6, z_{12}, z_{13}, z_{25}, z_{60}, z_{71}, z_{11}\}$ is retained where $z_{52}$ is the cheapest solution.
Using the Newly Proposed Algorithm only the best mapping for each node in $H$ is required. Therefore, the set of retained images of $y$ is $S^r = \{ y_1, y_2, y_3, y_5, y_7, y_9 \}$.
\[ z_1(y_1, r_1) = ((b, e), (b, d)), \text{ cost of 7, (0, 1, 3, 0)} \]
\[ z_2(y_1, r_1) = ((b, d), (b, e)), \text{ cost of 7, (0, 1, 3, 0)} \]
\[ z_3(y_1, r_1) = b, \text{ cost of 6, (1, 1, 2, 1)} \]
\[ z_4(y_1, r_1) = (a, b), \text{ cost of 8, (0, 2, 2, 2)} \]
\[ z_5(y_2, r_1) = ((c, g), (b, d)), \text{ cost of 7, (0, 1, 3, 0)} \]
\[ z_6(y_2, r_1) = ((b, d), (c, g)), \text{ cost of 7, (0, 1, 3, 0)} \]
\[ z_7(y_2, r_1) = a, \text{ cost of 7, (1, 1, 2, 2)} \]
\[ z_8(y_2, r_1) = (a', a), \text{ cost of 10, (0, 2, 2, 4)} \]
\[ z_9(y_3, r_1) = (a', a), \text{ cost of 11, (0, 1, 2, 5)} \]
\[ z_{10}(y_5, r_1) = ((c, f), (b, d)), \text{ cost of 7, (0, 1, 3, 0)} \]
\[ z_{11}(y_5, r_1) = ((b, d), (c, f)), \text{ cost of 7, (0, 1, 3, 0)} \]
\[ z_{12}(y_5, r_1) = a, \text{ cost of 7, (1, 1, 2, 2)} \]
\[ z_{13}(y_5, r_1) = (a', a), \text{ cost of 10, (0, 2, 2, 4)} \]
\[ z_{14}(y_7, r_1) = ((a, c), (b, d)), \text{ cost of 5, (1, 1, 2, 0)} \]
\[ z_{15}(y_7, r_1) = ((b, d), (a, c)), \text{ cost of 5, (1, 1, 2, 0)} \]
\[ z_{16}(y_7, r_1) = a, \text{ cost of 4, (2, 1, 1, 1)} \]
\[ z_{17}(y_7, r_1) = (a', a), \text{ cost of 7, (1, 2, 1, 3)} \]
\[ z_{18}(y_8, r_1) = ((a, c), (b, d)), \text{ cost of 8, (0, 2, 2, 2)} \]
\[ z_{19}(y_8, r_1) = ((b, d), (a, c)), \text{ cost of 8, (0, 2, 2, 2)} \]
\[ z_{20}(y_8, r_1) = a, \text{ cost of 7, (1, 2, 1, 3)} \]
\[ z_{21}(y_8, r_1) = (a', a), \text{ cost of 10, (0, 3, 1, 5)} \]
\[ z_{22}(y_9, r_1) = (a', a), \text{ cost of 9, (1, 3, 0, 6)} \]

Of this set of images for \( z \) only \( S' = (z_1, z_3, z_5, z_{10}, z_{14}, z_{15}, z_{16}) \) is retained where \( z_{16} \) is the cheapest solution matching the mapping of \( z_{52} \) from the alternate node mapping algorithm.