User Manual for NetworkDistances 1.0: Calculating Network-wise Distances Between Habitat Patches for Spatially Restricted Species

M.H. Grinnell and J.M.R. Curtis

Fisheries and Oceans Canada
Science Branch, Pacific Region
Pacific Biological Station
3190 Hammond Bay Road
Nanaimo, BC
V9T 6N7

2011

Canadian Technical Report of Fisheries and Aquatic Sciences 2960

Fisheries and Oceans Canada
Pêches et Océans Canada

Canada

Canadian Technical Report of Fisheries and Aquatic Sciences

Technical reports contain scientific and technical information that contributes to existing knowledge but which is not normally appropriate for primary literature. Technical reports are directed primarily toward a worldwide audience and have an international distribution. No restriction is placed on subject matter and the series reflects the broad interests and policies of Fisheries and Oceans Canada, namely, fisheries and aquatic sciences.

Technical reports may be cited as full publications. The correct citation appears above the abstract of each report. Each report is abstracted in the data base *Aquatic Sciences and Fisheries Abstracts*.

Technical reports are produced regionally but are numbered nationally. Requests for individual reports will be filled by the issuing establishment listed on the front cover and title page. Out-of-stock reports will be supplied for a fee by commercial agents.

Numbers 1-456 in this series were issued as Technical Reports of the Fisheries Research Board of Canada. Numbers 457-714 were issued as Department of the Environment, Fisheries and Marine Service, Research and Development Directorate Technical Reports. Numbers 715-924 were issued as Department of Fisheries and Environment, Fisheries and Marine Service Technical Reports. The current series name was changed with report number 925.

Rapport technique canadien des sciences halieutiques et aquatiques

Les rapports techniques contiennent des renseignements scientifiques et techniques qui constituent une contribution aux connaissances actuelles, mais qui ne sont pas normalement appropriés pour la publication dans un journal scientifique. Les rapports techniques sont destinés essentiellement à un public international et ils sont distribués à cet échelon. II n'y a aucune restriction quant au sujet; de fait, la série reflète la vaste gamme des intérêts et des politiques de Pêches et Océans Canada, c'est-à-dire les sciences halieutiques et aquatiques.

Les rapports techniques peuvent être cités comme des publications à part entière. Le titre exact figure au-dessus du résumé de chaque rapport. Les rapports techniques sont résumés dans la base de données *Résumés des sciences aquatiques et halieutiques*.

Les rapports techniques sont produits à l'échelon régional, mais numérotés à l'échelon national. Les demandes de rapports seront satisfaites par l'établissement auteur dont le nom figure sur la couverture et la page du titre. Les rapports épuisés seront fournis contre rétribution par les agents commerciaux.

Les numéros 1 à 456 de cette série ont été publiés à titre de Rapports techniques de l'Office des recherches sur les pêcheries du Canada. Les numéros 457 à 714 sont parus à titre de Rapports techniques de la Direction générale de la recherche et du développement, Service des pêches et de la mer, ministère de l'Environnement. Les numéros 715 à 924 ont été publiés à titre de Rapports techniques du Service des pêches et de la mer, ministère des Pêches et de l'Environnement. Le nom actuel de la série a été établi lors de la parution du numéro 925.

Canadian Technical Report of
Fisheries and Aquatic Sciences 2960

2011

USER MANUAL FOR NetworkDistances 1.0: CALCULATING NETWORK-WISE DISTANCES BETWEEN HABITAT PATCHES FOR SPATIALLY RESTRICTED SPECIES

by

M.H. Grinnell[1] and J.M.R. Curtis[2]

Fisheries and Oceans Canada
Science Branch, Pacific Region
Pacific Biological Station
3190 Hammond Bay Road
Nanaimo, BC
V9T 6N7

[1]E-mail: matt.grinnell@dfo-mpo.gc.ca | tel: (250) 756.7326
[2]E-mail: janelle.curtis@dfo-mpo.gc.ca | tel: (250) 756.7157

Cat. No. Fs97-6/2960E ISSN 0706-6457
Cat. No. Fs97-6/2960E-PDF ISSN 1488-5379

Correct citation for this publication:

Grinnell, M.H. and Curtis, J.M.R. 2011. User manual for NetworkDistances 1.0: Calculating network-wise distances between habitat patches for spatially restricted species. Can. Tech. Rep. Fish. Aquat. Sci. 2960: iv + 29 p.

CONTENTS

ABSTRACT

Grinnell, M.H. and Curtis, J.M.R. 2011. User manual for NetworkDistances 1.0: Calculating network-wise distances between habitat patches for spatially restricted species. Can. Tech. Rep. Fish. Aquat. Sci. 2960: iv + 29 p.

We develop an approach to calculate distances along network lines between habitat patches using **NetworkDistances** version 1.0. The **NetworkDistances** code is an optional extension to the **GRIP2** script for species that are restricted to moving along defined networks (e.g., rivers). For these species, the Euclidean distance may underestimate the actual distance between patches, which may influence patch dynamics. To more accurately reflect reality, we calculate the shortest 'along the network' distance between connected patches using **NetworkDistances**, and apply our analysis to a stream-dwelling minnow, the redside dace (*Clinostomus elongatus*).

RÉSUMÉ

Grinnell, M.H. and Curtis, J.M.R. 2011. User manual for NetworkDistances 1.0: Calculating network-wise distances between habitat patches for spatially restricted species. Can. Tech. Rep. Fish. Aquat. Sci. 2960: iv + 29 p.

Nous avons élaboré une méthode de calcul des distances le long des lignes de réseaux entre les parcelles d'habitat au moyen de la version 1.0 de **NetworkDistances**. Le code de **NetworkDistances** est une extension optionnelle du script **GRIP2**, utilisé pour les espèces dont les déplacements sont limités le long de réseaux définis (p. ex. des rivières). Pour ces espèces, la distance euclidienne peut sous-estimer la distance réelle entre les parcelles, ce qui peut influer sur la dynamique des parcelles. Pour refléter la réalité avec plus de justesse, nous calculons la distance la plus courte ≪ le long du réseau ≫ entre les parcelles reliées au moyen de **NetworkDistances**, puis nous appliquons notre analyse à un méné fréquentant les cours d'eau du réseau, soit le méné long (*Clinostomus elongatus*).

1 MOTIVATION

A *metapopulation* consists of multiple spatially discrete populations that occasionally exchange organisms, even though each population is in a discrete habitat patch. Organisms may move between patches when distances are less than the maximum dispersal distance, and dispersal success is typically inversely related to dispersal distance (Wolfenbarger 1946; Kitching 1971). The exchange of organisms between patches can affect patch dynamics and metapopulation persistence via patch recolonization and extinction rates (Johst et al. 2002). Dispersal rates among patches are typically modeled as a function of distance and cost(s) associated with moving through suboptimal habitat within a landscape or seascape. For some species, the Euclidean (i.e., straight line) distance may be appropriate for modeling the migration distance between patches. However, landscape features may prevent other species from moving in a straight line between patches (Figure 1). For these spatially restricted species, the Euclidean distance may under-estimate the effective distance between patches, as well as over-estimate migration rates and dispersal success, leading to biases in predicted patch extinction and recolonization rates.

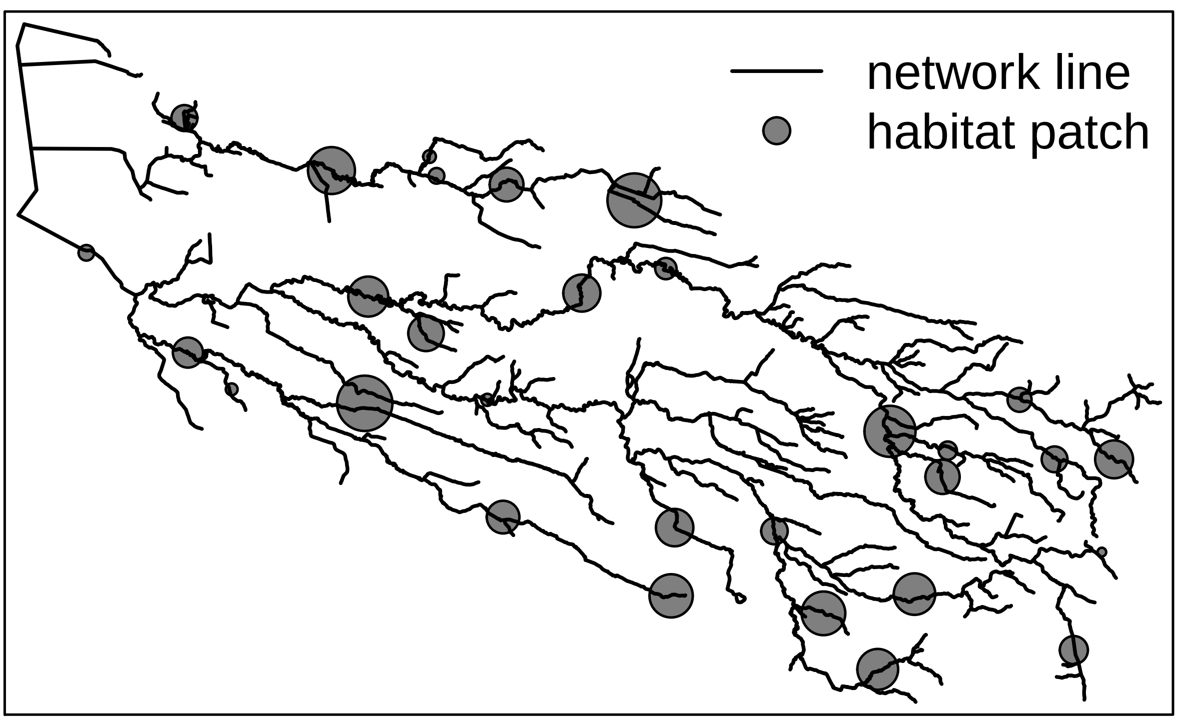

Figure 1. Hypothetical landscape for a lotic species that is restricted to moving along network lines (e.g., rivers) that connect patches of suitable habitat (e.g., riffles). Note that patch size is proportional to circle size.

2 BACKGROUND

We assume that readers contemplating use of the **NetworkDistances** code are familiar with the suite of **RAMAS** software, specifically **RAMAS GIS** and **Metapop** (Akçakaya 2005) as well as **GRIP** (Curtis and Naujokaitis-Lewis 2008*a*,*b*). Briefly, the **GRIP2** script, written in the programming language **R** (RDCT 2011), is designed for use with **RAMAS GIS 5.0** software to perform systematic global sensitivity analysis of habitat and population parameters for spatially explicit population viability analyses. The **RAMAS GIS** programme links spatial data (e.g., habitat suitability maps) to stochastic metapopulation models, and allows users to evaluate the influence of landscape structure on metapopulation dynamics by manually varying habitat parameters. The **GRIP2** script automates this time-consuming process by generating a specified number of random sets of spatial and non-spatial parameters, and running these iterations by submitting batch files to **RAMAS**. By default, **GRIP2** and **RAMAS** software calculate the Euclidean distance between patches.

We developed **NetworkDistances** version 1.0, an optional **GRIP2** extension, to be used when organism movement and dispersal is restricted to defined spatial networks (Figure 2). For example, the movement of lotic fish is restricted to river networks, while other species may be restricted to moving along hedgerows, coastlines, trails, or corridors. Specifically, we developed **NetworkDistances** to automate the calculation of distances between pairs of patches for redside dace (RSD; *Clinostomus elongatus*), an Endangered stream-dwelling minnow found *inter alia* in four watersheds in the Greater Toronto Area that discharge into Lake Ontario, Canada (COSEWIC 2007). We provide some background information on RSD and refer to features of our RSD model in this manual to illustrate concepts, limitations, and opportunities where appropriate. Most code within **NetworkDistances** is generic and could be applied to any spatially-constrained species to calculate pairwise distances. We attempt to highlight sections of code that may require some customization for application with other species.

The **NetworkDistances** code (`NetworkDistances.R`) has extensive comments and should be referenced when reading this document. Please contact the authors if you have questions, comments, suggestions or concerns regarding the manual, or the code. We are attempting to keep track of this code's use; please cite this manual and contact the authors if you use **NetworkDistances**. Note that **NetworkDistances** comes with absolutely no warranty.

3 RSD HABITAT AND NETWORK DATA

River data were acquired from Natural Resources Canada as a vector line shapefile (scale = 1: 50 000; NRC 2011). Georeferenced shapefiles for RSD habitat patches and rivers were projected in Universal Transverse Mercator (UTM, zone 17), in metres (m) using the North American 1983 datum. Data preparation for **RAMAS** input files (e.g., habitat mask, habitat suitability grid) was done using the geographic information system programme **ArcMap 9.2** (ESRI 2006). Although spatial files for **RAMAS** input

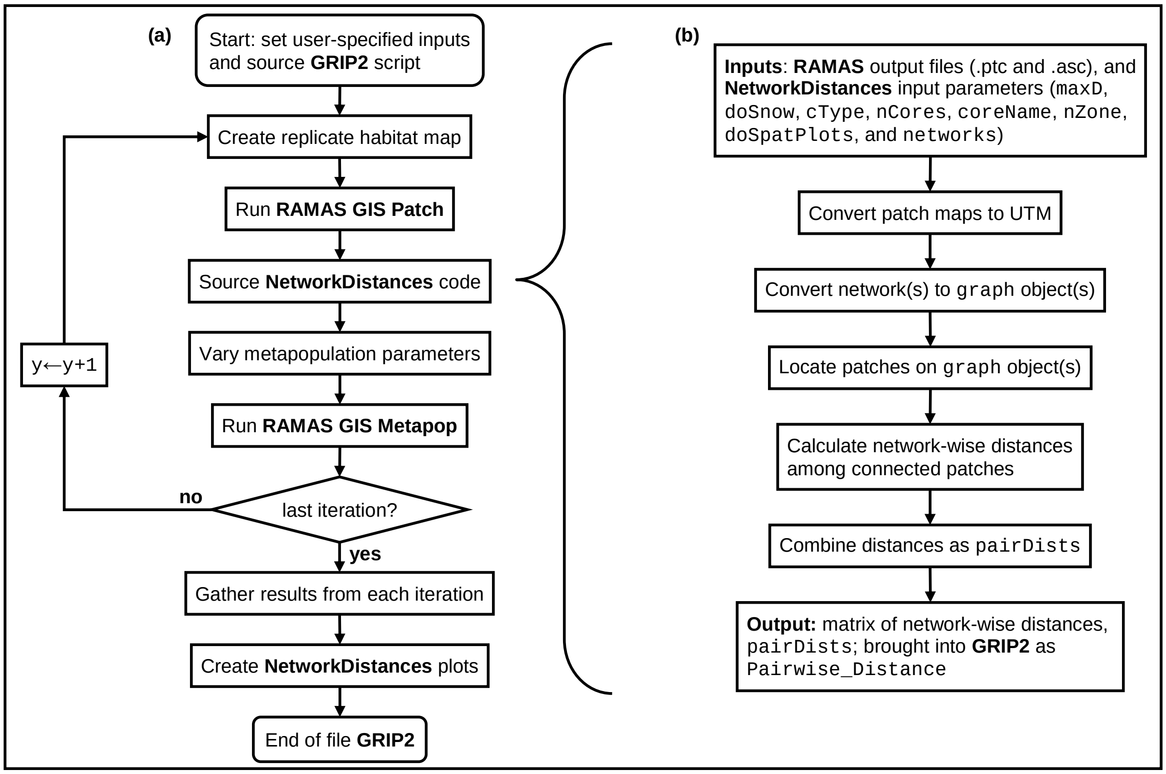

Figure 2. Simplified flow diagram of the **GRIP2** script (a), and the optional **NetworkDistances** code (b). The **NetworkDistances** algorithm is expanded in Subsection 4.2, and inputs are described in Section 5.

(e.g., `*.asc`) must have, *inter alia*, consistent precision and spatial extent (Akçakaya 2005), networks may extend beyond the grid's perimeter. However, larger networks may increase memory and processing time, as discussed in Subsection 4.1.

In our case study, we modeled the metapopulation dynamics of RSD within a small study area, which contains two watersheds (Figure 3). Two RSD characteristics enabled us to simplify our analysis, and thus our code. First, genetic analyses indicate that RSD do not move between watersheds (M. Poos, unpublished data), which allowed us to divide the study area into two independent watersheds, which we refer to as *networks*. We defined individual networks by their Strahler stream order number, a measure of branching complexity (Strahler 1957). Second, mark-recapture studies indicate that RSD dispersal is independent of stream direction (Poos and Jackson, in press), which allowed us to create undirected `graph` objects. The benefits of these two simplifications are explained in later sections.

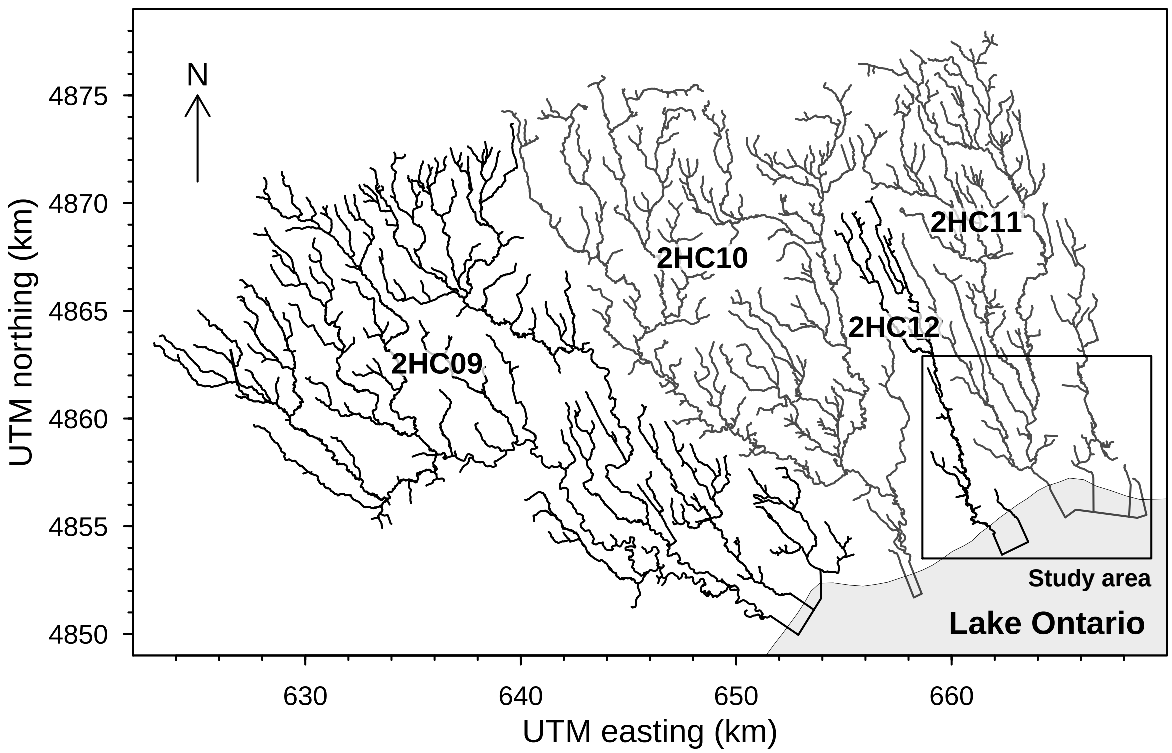

Figure 3. Two of the four watersheds (e.g., 2HC11) run through the redside dace study area, and discharge into Lake Ontario (NRC 2011). Geographic coordinates are projected in Universal Transverse Mercator (UTM, zone 17), in kilometres (km). Note that networks are different colours to aid with differentiation.

4 METHODOLOGY USING R

We assume the user has at least a working ability with the **R** statistical and graphing programme (RDCT 2011), and is familiar with spatial data. In addition to the **R** packages required for **GRIP2** [`sp` (Pebesma and Bivand 2005; Bivand et al. 2008), `rgdal` (Keitt et al. 2010), `spatial` (Venables and Ripley 2002), `spatstat` (Baddeley and Turner 2005), `adehabitat` (Calenge 2006), `maptools` (Lewin-Koh and Bivand 2011), `ade4` (Dray and Dufour 2007), and `gpclib` (Peng 2010)], **NetworkDistances** requires at least five additional packages: `PBSmapping` (Schnute et al. 2010) to handle georeferenced data; `graph` (Gentleman et al. 2010), `RBGL` (Carey et al. 2010), and `igraph` (Csárdi and Nepusz 2006) to handle graph objects; as well as `gtools` (Warnes 2010) to sort strings with embedded numbers. One or two other packages are optional: `snow` (Tierney et al. 2008) to use multiple processors; and (also optionally) `Rmpi` (Yu 2010) to use "MPI" type clusters on non-Windows machines. Non-Windows machines require additional code to ensure that files are compatible with both dos and unix, as well as the **WineHQ** programme (WPD 2010) to run **RAMAS** if **NetworkDistances** is used along with **GRIP2**.

4.1 CHALLENGES AND SOLUTIONS TO MEMORY AND TIME REQUIREMENTS

The main challenge to **NetworkDistances** is that large, complex networks with many patches and high resolution maps may be costly in terms of required memory and processing time. However, memory and time requirements can be reduced significantly using four strategies: (1) splitting large networks into independent sub-networks; (2) loading saved `RData` objects from the first iteration; (3) parallel processing; and (4) reducing map resolution.

First, large networks will require less memory and processing time if they can be split into independent sub-networks in a biologically meaningful way. Splitting a large network (within which some patches are not connected) into multiple sub-networks (within which all patches are connected) reduces the number of pairwise distance calculations. For example, because RSD do not migrate between watersheds, we split the large river network in our study area into two small independent networks by watershed boundaries (Figure 3). More generally, consider a hypothetical scenario with three patches on one network (e.g., patches a, b, c), and three patches on a second, independent network (e.g., d, e, f). Treated as one large network, there are 15 pairwise distances to calculate for the upper triangle of the required 6×6 matrix:

$$\begin{array}{ccccccc} & a & b & c & d & e & f \\ a & 0 & 1 & 2 & \text{NA} & \text{NA} & \text{NA} \\ b & 1 & 0 & 3 & \text{NA} & \text{NA} & \text{NA} \\ c & 2 & 3 & 0 & \text{NA} & \text{NA} & \text{NA} \\ d & \text{NA} & \text{NA} & \text{NA} & 0 & 4 & 5 \\ e & \text{NA} & \text{NA} & \text{NA} & 4 & 0 & 6 \\ f & \text{NA} & \text{NA} & \text{NA} & 5 & 6 & 0 \end{array} \tag{1}$$

where "NA" indicates that the distance cannot be calculated because the two patches are unconnected (i.e., due to migration barriers between patches).[3] Note that the lower triangle is the transpose of the upper triangle because dispersal is assumed to be independent of direction. Alternately, treating the two sub-networks independently reduces the number of pairwise distance calculations from 15 to 6, all of which can be calculated:

$$\begin{array}{cccc} & a & b & c \\ a & 0 & 1 & 2 \\ b & 1 & 0 & 3 \\ c & 2 & 3 & 0 \end{array} \quad \text{and} \quad \begin{array}{cccc} & d & e & f \\ d & 0 & 4 & 5 \\ e & 4 & 0 & 6 \\ f & 5 & 6 & 0 \end{array} \tag{2}$$

To further reduce network sizes, we then removed sections outside the study area boundary.

[3] The algorithm requires more time and memory to attempt to calculate distances between unconnected patches than connected patches, and eventually sets these distances to `NA`.

Second, as a sensitivity analysis programme, **GRIP2** typically runs multiple iterations, and network lines remain constant from one iteration to another. Thus, networks can be saved to disk as `RData` objects on the first iteration, and loaded from disk on subsequent iterations. This strategy reduces processing time considerably for large networks. However, due to changing patch locations associated with random changes in landscape structure each iteration, pairwise distances must be calculated iteratively.

Third, calculating pairwise distances can be computationally demanding, and takes considerable time when networks are large, or when there are many patches. In these cases, calculating pairwise distances requires more processing time than creating networks, and multiple processors can be used to reduce processing time. In this case, the **NetworkDistances** code starts multiple instances of **R**, and divides pairwise distance calculations among processors on the cluster. With sufficient memory, parallel processing may even reduce processing time on computers with only one processor.

Finally, time and memory requirements increase in proportion to increasing network size, branching complexity, number of pairwise distance calculations, and size of input maps. Processing time and memory requirements were considerable because the RSD study area was modeled at a fine spatial resolution: our input maps have approximately 443 000 $15\,\mathrm{m} \times 15\,\mathrm{m}$ grid cells. For example, between 1.0 and 1.5 hours were required to run 50 **GRIP2** iterations of the RSD study area on two different machines. Of the total required time, between 2.8 and 5.2% was used to run the **NetworkDistances** code. Our analysis required fine spatial resolution because RSD are typically distributed in headwater streams that are closely spaced, but networks for other species may not require such fine spatial resolution.

4.2 OUTLINE OF THE NetworkDistances CODE

Our analysis relies on well-developed methods from graph theory to measure distances between patches along networks. Although graph theory has been around for many years, using graphs to model node (e.g., patch) connectivity is relatively new to the field of conservation biology (Urban et al. 2009). We developed the **NetworkDistances** code in the programming language **R** to facilitate implementation with **GRIP2**, and to cope with changing patch locations each iteration.

The **NetworkDistances** code is typically sourced via the **GRIP2** script for spatially restricted species, and we assume that the user is familiar with **GRIP2**. Alternately, **NetworkDistances** can be run independently if all the required inputs are specified appropriately (e.g., arguments for the function `GridToUTM( ptc, asc )` and Section 5). The following algorithm outlines the **NetworkDistances** procedure and explains the code's major functions; we recommend that users follow along in the `NetworkDistances.R` code (Listing 1, Appendix).

This algorithm is run for each `y` replicate iteration in `newNreps` as follows:

1. Convert **RAMAS** spatial output for patches from grid locations to UTM (m).

   `GridToUTM( ptc, asc )` Retrieve spatial data for each patch from **RA-**

MAS output files: `ptcfile` (i.e., `ptc_y.ptc`); and `OrigPatchmap` (i.e., `spp_PA.asc`), where `spp` is typically the species name. The `ptcfile` file indicates patch centers relative to the upper-left corner of the grid (in km), while `OrigPatchmap` georeferences the lower-left corner of the grid (m).[4] Patch locations are converted to UTM (m) using the geographic information in the `OrigPatchmap` file header. The arguments `ptc` and `asc` reference the `ptcfile` and `OrigPatchmap` files, respectively. The function returns a list with two elements: the location of patches in UTM (m), `patchesUTM`, as an `EventData` object; and the grid outer extent in UTM (m), `gridExtent`.

2. Step 2 is done once if there is only one network, or iteratively if there are multiple sub-networks. First, import the network shapefiles in UTM (m) as a `PolySet` object, `netVect`, which indicates node locations (X, Y), and node connectivity. Set `patchesUTM` to `patches`; this is required because `patches` will be subset during the following calculations to include only the patches within the current network. Each time the loop is iterated, `patchesUTM` is re-set to `patches`, which ensures that all of the patches are considered.

   (a) If available, load the saved `netVect.RData` object, and skip to Step 2b. Otherwise, proceed as follows:

   `CreateUniqueID( dat )` Create a unique ID number for each unique node (X, Y) to ensure the `graph` object is continuous. By default, shapefiles have a different ID for each node, which can cause problems for nodes that are common to two lines that merge into a single stem (e.g., river confluence). These two nodes, which have identical (X, Y) location, must have identical IDs for the `graph` object to consider them attached. The argument `dat` is the network object, and the function returns the updated network object, `netVect` with new columns (especially, `UniqueID`).

   `GetToFromDist( dat )` Create a matrix indicating the unique ID for each segment's *start* (e.g., from) and *end* (e.g., to) nodes, as well as the Euclidean distance along each edge. For these short edges, the Euclidean distance sufficiently approximates reality if maps have a fine spatial resolution. Nodes correspond to network vertices, and edges correspond to lines between vertices. Note that the words *to* and *from* simply indicate endpoints because the graph object is *undirected*, meaning that the

[4] Pairwise distances among patches can be calculated from one edge to the other, from the edge of one patch to the center of the other, or from center to center. The center-to-center distance may be very different from the edge-to-edge distance when patches are large (i.e., patch edges extend away from the patch center along network lines). For simplicity, we assume that the distance between patch centers best describes pairwise distances. Thus, network-wise distances are calculated from center-to-center; set `distance <- ''Default: Center to center''` in **GRIP2** prior to running iterations to ensure that the `ptfile` file refers to patch centers.

distance from *Pop 1* to *Pop 2* is equal to the distance from *Pop 2* to *Pop 1*. Species that require a *directed* graph (i.e., dispersal depends on direction) will require the user to modify this function. The argument `dat` is the network object, and the function returns `toFromDist` (later renamed `tfTable`), a matrix of node IDs, node locations, and distances between nodes (i.e., edge weights).

Save the output (e.g., network object, `netVect`, and to-from matrix, `tfTable`) as a `netVect.RData` object to be used in subsequent iterations.

(b) Load the saved `netVect.RData` object, then:

`EnsureUniqueSites( dat )` Ensure that each patch has a unique ID so that pairwise distances are calculated for each patch. Duplicate names are made unique by appending letters (e.g., "`a`") to the second duplicate, and so on.[5] The argument `dat` is the patches, and the function updates the column `Site` with new IDs if duplicates exist.

`SnapToNodes( pops )` Snap each patch to the nearest network node. Basically, identify the closest node to the patch center, and change the node ID to the patch ID (Figure 4). Thus, patch locations are shifted, but are likely within tolerance considering that **RAMAS** uses a grid to define patches. The search radius is controlled by the parameter `maxD`, which restricts candidate nodes to those within `maxD` units of the patch. It is important to note that patches that are more than `maxD` from all nodes are omitted from the analysis. Although a habitat mask will ensure that patches are located near network lines, distances between patch centers and network nodes can be large when network lines are straight (i.e., fewer nodes), or when patches are large. An iterative search reduces processing time: first search for nodes within $\frac{\texttt{maxD}}{4}$ units of the patch; if there are no nodes, search again within `maxD` units. The argument `pops` is set to the patch. The function returns `minDists`, which indicates the distance from each patch to the nearest node (or `NA` if there are no nodes), and the function updates the node ID in `tfTable` (for both *to* and *from* nodes).

`ftM2graphNEL( ft, edgemode, W )` This function is from the `graph` package, and creates an object of class `graph` to represent the network. Prior to calling this function, duplicate rows must be omitted from the `tfTable` object. The argument `ft` is the to-from table, `tfTable`, `edgemode` indicates that the `graph` is not directed, and `W` indicates the edge weights. The function returns the `graph` object, `netGraph`.

`CalcDistMatrix( dat )` Calculate the shortest (e.g., least-cost) network-wise distances (m) between each pair of patches in the network using Dijkstra's (1959) algorithm, implemented by the function `sp.between( )`.

[5] Although patches will have unique IDs if **NetworkDistances** is sourced via **GRIP2**, there may be duplicate patch IDs if **NetworkDistances** is run independently.

This function can use parallel processing to decrease processing time, if `doSnow` is `TRUE` (Section 5). Note that the distance from the patch to the network, `minDists`, is not included in the pairwise distance calculation because the patch is technically *on* the network, even if the patch's center is shifted slightly. The argument `dat` is the `graph` object, and the function returns `distMat`, a matrix of pairwise distances with row and column names corresponding to patch names. The `distMat` object is saved as a `distMat.netVect.RData` object in the folder `RDataOutTemp/` so that distances between points in independent sub-networks can be brought together as one large distance matrix using `CombineDistances( dat )` in Step 3.

3. Combine the pairwise distance matrices from each independent sub-network into one large matrix.

   `RemoveDupPatches( mat )` Ensure that each patch is only in one network, and ignore networks that do not contain any patches. This function ensures that each patch is snapped to the network with the single closest node. For example, large `maxD` values can cause patches to be inadvertently snapped to two networks if patches are located within `maxD` of nodes on both networks. The argument `mat` is the matrix of `minDists`, named `minDistMat`, which has a column for each network, and a row for each patch. The function returns `patchNetwork`, with a column for each network that contains patches, and a row for each patch that has been snapped to a network (patches are assumed to belong in the network with the nearest node).

   `CombineDistances( dat )` Combine distance matrices from each sub-network into one large pairwise distance matrix. The argument `dat` is the set of network name(s) that contain patches, and the function returns the matrix of pairwise distances for the entire group of networks, `pairDists`. The distance between points in different independent sub-networks (i.e., unconnected patches) is `NA`.

   `FillPairDists( mat )` Ensure that `pairDists` has one column and one row for each patch, even if the patch was omitted from the analysis. The argument `mat` is `pairDists`, and the function returns an updated `pairDists`.

   The `pairDists` matrix is ordered by row and column names for compatibility with **GRIP2** requirements. The matrix is saved as `pairDists.y.RData` in the folder `RDataOutDist/` to be retrieved by **GRIP2** for further analysis.

4. Step 4 is done after **GRIP2** has run to completion, and generates figures of networks and patches. These plots can take considerable time to create when there are many large networks and many iterations; skip this step to speed up the analysis (Section 5).

PlotNetworkPatches( ) Plot georeferenced network(s) and patches in UTM (m). There are no arguments, and the function does not return anything; instead, the function creates a portable document format (PDF) file in the folder RDataOutDist/ named plot.pdf, with one page for each iteration (Figure 5). Patch names in green have been snapped to network lines; patch names in red have been omitted from the analysis (i.e., patches more than maxD units from all network nodes).

5 INCORPORATING NetworkDistances INTO GRIP2

In addition to the usual controls and parameters required for **GRIP2** and **RAMAS**, **NetworkDistances** requires several additional user-specified inputs which must be specified in **GRIP2** or at the start of the **NetworkDistances** code:

calc_network_dist Set to TRUE to calculate distances along networks; set to FALSE to calculate Euclidean distances. Value: logical.

maxD Maximum distance (m) from each patch to search for candidate network nodes in SnapToNodes(pops). Value: number.

doSnow Set to TRUE to use parallel processing in CalcDistMatrix(dat); set to FALSE for no parallel processing. Value: logical.

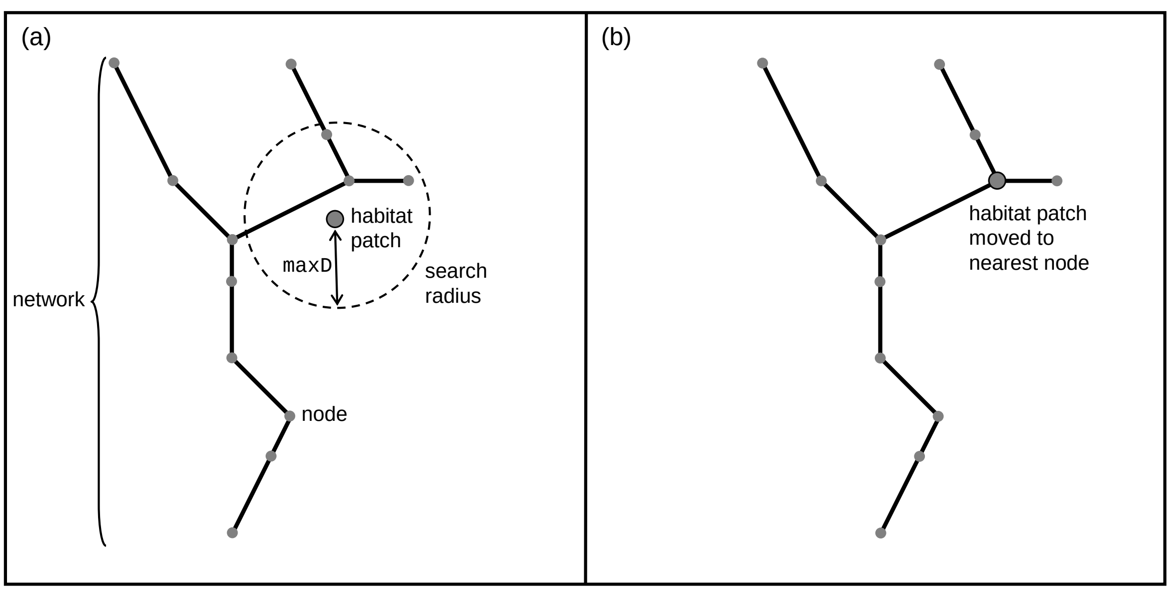

Figure 4. Hypothetical network and habitat patch (a). The network is composed of nodes connected by segments. Three nodes are within maxD units of the patch's center; the SnapToNodes(pops) function shifts the patch's location to the nearest node (b).

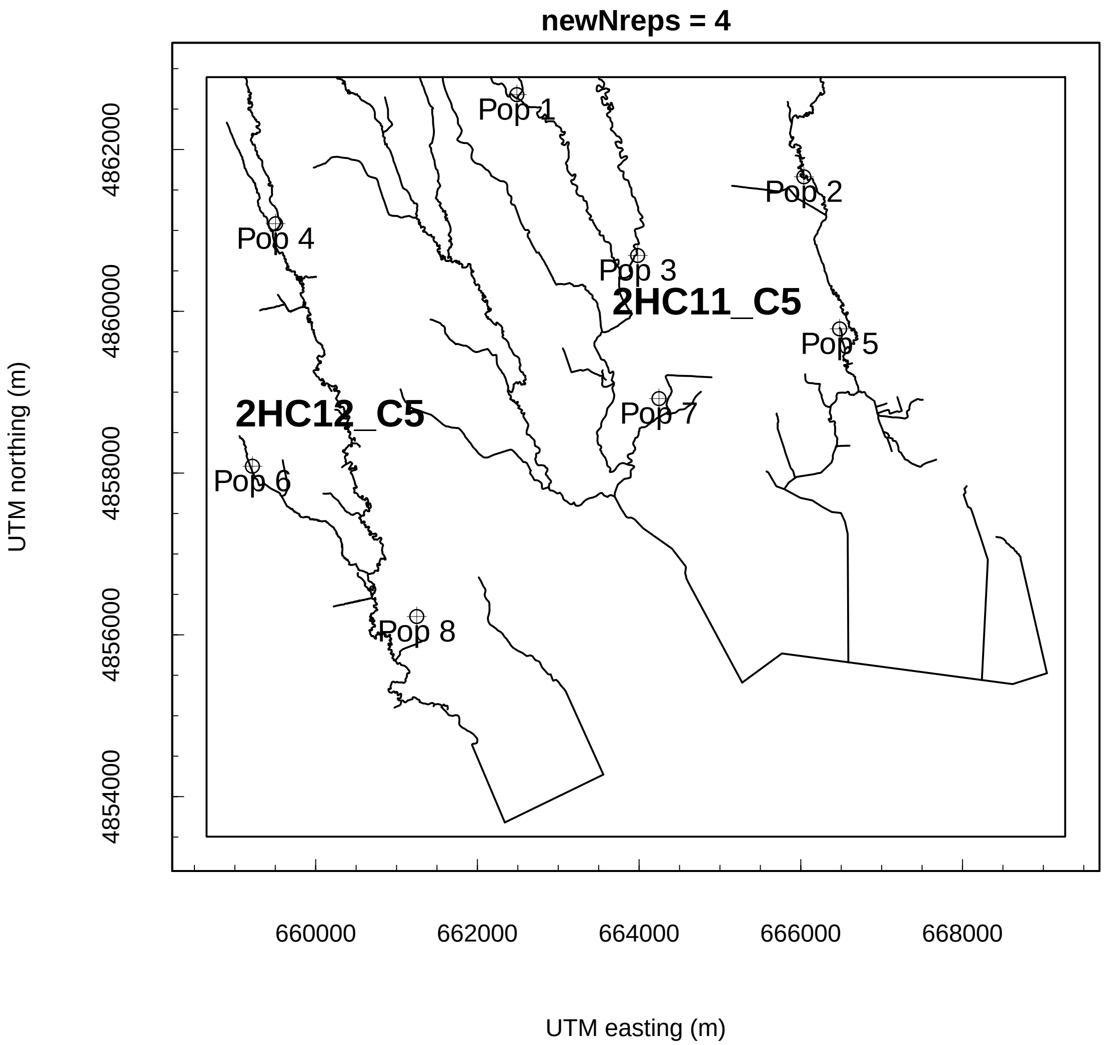

Figure 5. Plot created by the function `PlotNetworkPatches( )` showing the eight habitat patches and two networks on the fourth iteration of the redside dace study area. Note that patches `Pop 4`, `Pop 6`, and `Pop 8` are all connected, and are on network `2HC12_C5`; these patches are not connected to the patches on network `2HC11_C5` (e.g., `Pop_1`). Note also that patch names are shown in black for printing purposes, but would normally be shown in green (i.e., they have all been snapped to network lines).

`cType` Cluster type (if `doSnow`). Use `‘‘SOCK’’` for Windows machines, or either `‘‘SOCK’’` or `‘‘MPI’’` for non-Windows machines. Value: character.

`nCores` Number of cores for parallel processing (if `doSnow`). For example, in Windows, this indicates the number of instances of `Rscript.exe` that are initialized and used in `CalcDistMatrix( dat )`. Note that `nCores` controls a trade-off between the number of cores, and the memory available to each core. Value: integer.

`coreName` Processor name (if `doSnow`). Value: character (e.g., ``‘‘localhost’’``).

`nZone` UTM zone, required to align patches and network lines. Value: integer.

`doSpatPlots` Set to `TRUE` to call `PlotNetworkPatches( )` and generate figures showing patches and networks for each iteration; set to `FALSE` to skip this step. Value: logical.

`networks` Shapefile network name(s), not including extension(s). If there are multiple shapefiles (i.e., the network is broken up into independent sub-networks), `networks` is a concatenation of each sub-network name; if there is only one network, `networks` is the network name. Input shapefiles (in UTM, m) must be in the folder `NetworkShapefiles/`. Value: character vector (e.g., `c(‘‘2HC09’’, ‘‘2HC10’’)` for multiple sub-networks; character (e.g., `‘‘2All’’`) for one network).

The **NetworkDistances** code should be in the same folder as **GRIP2**, the *working directory*. Create two folders in the working directory: `NetworkShapefiles/`, and `SavedNetworkObjects/`. The folder `NetworkShapefiles/` contains the shapefiles for the network(s). The folder `SavedNetworkObjects/` may contain saved `netVect.RData` objects from previous iterations to speed up computations, if they are available (Subsection 4.2, Step 2a).

Two additional folders are created during the simulation: `RDataOutTemp/`, and `RDataOutDist/`. The folder `RDataOutTemp/` holds temporary **R** output each iteration. The folder `RDataOutDist/` contains pairwise distance matrices for each iteration and network. Pairwise distance objects are named `pairDists.y.RData` in order to be retrieved later by **GRIP2** to calculate various statistics. Two other objects are saved in the folder `RDataOutDist/` under the names `patchNetworkDist.y.RData` (contains `minDistMat` and `patchNetwork`), and `patches.y.RData` (contains `patches` and `patchesUTM`). This folder may also contain the figure of patches and network lines, `plot.pdf`.

5.1 DISTANCE UNITS AND CONVERSIONS

Distance units may vary between the **RAMAS** programme, the **GRIP2** script, and the **NetworkDistances** code. For consistency, the **NetworkDistances** code uses metres (converting kilometres to metres when necessary). For the RSD analysis, network shapefiles are in metres, and **RAMAS** output files are in metres (e.g., `Patchmap`) and kilometres (e.g., `ptcfile`). **NetworkDistances** calculations and inputs (e.g., `maxD`) are in metres. However, because **RAMAS** input files require distances in kilometres, `pairDists` is converted from metres to kilometres just prior to being returned to **GRIP2** as a matrix of pairwise distances, `Pairwise_Distance`. It is critical that users ensure that distances have expected units in **RAMAS**, **GRIP2**, and **NetworkDistances** input and output files.

6 VERIFYING NetworkDistances AND EXTENSIONS

Distances calculated using the NetworkDistances code should sufficiently approximate reality if network shapefiles are accurate and of sufficient spatial resolution. However, we recommend that users confirm the accuracy of the **NetworkDistances** output by checking various statistics, the distance matrix `pairDists`, and the figure generated by the function `PlotNetworkPatches( )`. Additionally, ensure that **RAMAS** output files (i.e., `spp.asc`) line up with networks by plotting patches and network lines, as well as verifying the patch map grid location (Subsection 4.2, Step 1). Ensure that `maxD` is sufficiently large to locate candidate nodes in `SnapToNodes( pops )`. Investigate the matrices in `patchNetworkDist.y.RData` to ensure that patches are assigned to the correct network. These matrices will also indicate whether `maxD` is too small (e.g., patches are omitted), or too large (e.g., snapped distances are much shorter than `maxD`). Also, ensure that **RAMAS** files, grids, and network lines have identical projections and correct units.

As previously mentioned, one extension is to use directed graphs; for example, migration distance may depend on direction for fish that inhabit fast-flowing rivers (Step 2a). A second use of directed graphs is to model the effects of patch size (e.g., area) and population on migration; for example, large patches may have more immigrants, while large populations may have more emigrants. Another extension is to incorporate a changing spatial network over time. For example, a previously continuous river network could become fragmented by dams. This type of temporal change would increase processing time because year-specific network objects would have to be calculated each year.

7 ACKNOWLEDGEMENTS

The authors thank K. Kinnersley for facilitating our use of the high performance computing facility at the Institute of Ocean Sciences, I. Naujokaitis-Lewis for technical support, and M. Poos for comments which improved the document. Funding support was provided by Fisheries and Oceans Canada's National Species at Risk Programme.

REFERENCES

Akçakaya, H. R. 2005. RAMAS GIS: Linking spatial data with population viability analysis. Applied Biomathematics. URL `www.ramas.com/ramas.htm#gis`. User manual for version 5

Baddeley, A. and Turner, R. 2005. Spatstat: An R package for analyzing spatial point patterns. *Journal of Statistical Software* **12**(6): 1–42. URL `www.jstatsoft.org`. R package version 1.21-6

Bivand, R. S., Pebesma, E. J. and Gómez-Rubio, V. 2008. Applied spatial data analysis with R. Springer, NY. URL `http://www.asdar-book.org/`. R package version 0.9-80

Calenge, C. 2006. The package "adehabitat" for the R software: A tool for the analysis of space and habitat use by animals. *Ecological Modelling* **197**: 516–519. doi:10.1016/j.ecolmodel.2006.03.017. R package version 1.8-4

Carey, V., Long, L. and Gentleman, R. 2010. RBGL: An interface to the BOOST graph library. URL `http://CRAN.R-project.org/package=RBGL`. R package version 1.26.0

COSEWIC (Committee on the Status of Endangered Wildlife in Canada). 2007. COSEWIC assessment and update status report on the redside dace *Clinosomus elongatus* in Canada. Technical Report, Canadian Wildlife Service and Environment Canada. URL `www.sararegistry.gc.ca/status`

Csárdi, G. and Nepusz, T. 2006. The igraph software package for complex network research. *InterJournal* **Complex Systems**: 1695. URL `http://igraph.sf.net`. R package version 0.5.5-2

Curtis, J. M. R. and Naujokaitis-Lewis, I. R. 2008*a*. Sensitivity of population viability analysis to spatial and nonspatial parameters using GRIP. *Ecological Applications* **18**(4): 1002–1013. doi:10.1111/j.1523-1739.2008.01066.x

Curtis, J. M. R. and Naujokaitis-Lewis, I. R. 2008*b*. Source code for the program GRIP 1.0 (Generation of Random Input Parameters). URL `http://esapubs.org/archive/appl/A018/033/suppl-1.htm`. Ecological Archives: A018-033-S1 (Supplement)

Dijkstra, E. W. 1959. A note on two problems in connexion with graphs. *Numerische Mathematik* **1**(1): 269–271. doi:10.1007/BF01386390

Dray, S. and Dufour, A. B. 2007. The ade4 package: Implementing the duality diagram for ecologists. *Journal of Statistical Software* **22**(4): 1–20. URL `http://CRAN.R-project.org/package=ade4`. R package version 1.4-17

ESRI (Environmental Systems Research Institute). 2006. ArcMap for ArcGIS Desktop. URL http://www.esri.com. Version 9.2, Build 1324, ArcView License

Gentleman, R., Whalen, E., Huber, W. and Falcon, S. 2010. graph: A package to handle graph data structures. URL http://CRAN.R-project.org/package=graph. R package version 1.28.0

Johst, K., Brandl, R. and Eber, S. 2002. Metapopulation persistence in dynamic landscapes: The role of dispersal distance. *Oikos* **98**(2): 263–270. doi:10.1034/j.1600-0706.2002.980208.x

Keitt, T. H., Bivand, R., Pebesma, E. and Rowlingson, B. 2010. rgdal: Bindings for the geospatial data abstraction library. URL http://CRAN.R-project.org/package=rgdal. R package version 0.6-33

Kitching, R. 1971. A simple simulation model of dispersal of animals among units of discrete habitats. *Oecologia* **7**(2): 95–116. doi:10.1007/BF00346353

Lewin-Koh, N. J. and Bivand, R. 2011. maptools: Tools for reading and handling spatial objects. URL http://CRAN.R-project.org/package=maptools. With contributions from E. J. Pebesma, E. Archer, A. Baddeley, H.-J. Bibiko, S. Dray, D. Forrest, M. Friendly, P. Giraudoux, D. Golicher, V. G. Rubio, P. Hausmann, K. O. Hufthammer, T. Jagger, S. P. Luque, D. MacQueen, A. Niccolai, T. Short, B. Stabler and R. Turner. R package version 0.8-6

NRC (Natural Resources Canada). 2011. GeoGratis: Geospatial data available online at no cost and without restrictions. URL http://geogratis.gc.ca. Retrieved 5 January 2011

Pebesma, E. J. and Bivand, R. S. 2005. Classes and methods for spatial data in R. *R News* **5**(2): 9–13. URL http://cran.r-project.org/doc/Rnews/. R package version 0.9-80

Peng, R. D. 2010. gpclib: General polygon clipping library for R. URL http://CRAN.R-project.org/package=gpclib. With contributions from D. Murdoch and B. Rowlingson; GPC library by A. Murta. R package version 1.5-1

Poos, M. S. and Jackson, D. A. (in press). Incorporating species specific data into patch occupancy models: Impact of dispersal on estimates of viability of stream metapopulations. *Landscape Ecology* doi:10.1007/s10980-011-9683-2

RDCT (R Development Core Team). 2011. R: A language and environment for statistical computing. URL www.R-project.org. R Foundation for Statistical Computing. Vienna, Austria. R version 2.13.0

Schnute, J. T., Boers, N., Haigh, R. and Couture-Beil, A. 2010. PBSmapping: Mapping fisheries data and spatial analysis tools. URL `http://CRAN.R-project.org/package=PBSmapping`. R package version 2.61.9

Strahler, A. N. 1957. Quantitative analysis of watershed geomorphology. *Transactions of the American Geophysical Union* **8**(6): 913–920

Tierney, L., Rossini, A. J., Li, N. and Sevcikova, H. 2008. snow: Simple network of workstations. URL `http://CRAN.R-project.org/package=snow`. R package version 0.3-3

Urban, D. L., Minor, E. S., Treml, E. A. and Schick, R. S. 2009. Graph models of habitat mosaics. *Ecology Letters* **12**(3): 260–273. doi:10.1111/j.1461-0248.2008.01271.x

Venables, W. N. and Ripley, B. D. 2002. Modern applied statistics with S. Springer, New York, fourth ed. URL `http://www.stats.ox.ac.uk/pub/MASS4`. ISBN 0-387-95457-0. R package version 7.3.3

Warnes, G. R. 2010. gtools: Various R programming tools. URL `http://CRAN.R-project.org/package=gtools`. Includes R source code and/or documentation contributed by B. Bolker and T. Lumley. R package version 2.6.2.

Wolfenbarger, D. O. 1946. Dispersion of small organisms: Distance dispersion rates of bacteria, spores, seeds, pollen, and insects; Incidence rates of diseases and injuries. *American Midland Naturalist* **35**(1): 1–152

WPD (Wine Project Developers). 2010. WineHQ: Wine is not an emulator. URL `www.winehq.org`. Version 1.2.2

Yu, H. 2010. Rmpi: Interface (wrapper) to MPI (Message-Passing Interface). URL `http://CRAN.R-project.org/package=Rmpi`. R package version 0.5-9

APPENDIX

Electronic copies of the modified **GRIP2** script for use with **NetworkDistances**, the `NetworkDistances.R` code (Listing 1), and the RSD study area data (e.g., river networks, RSD habitat map) are available free from the authors.

Listing 1. The **NetworkDistances** code (`NetworkDistances.R` version 1.0) is written in the programming language **R** (RDCT 2011).

```
###############################################################################
#
# Author:         Matthew H. Grinnell
# Affiliation:    Pacific Biological Station, Fisheries and Oceans Canada
# Research group: Conservation Biology Section (Janelle M. R. Curtis)
# Contact:        e-mail: matt.grinnell@dfo-mpo.gc.ca | tel: (250)756.7326
#                 e-mail: janelle.curtis@dfo-mpo.gc.ca | tel: (250)756.7157
# Project:        GRIP2 extension for movement between patches within networks
# Code name:      NetworkDistances.R
# Code version:   1.0
# Date started:   2010-11-30
# Date modified:  2011-11-16
#
# Goal: Convert spatial lines (i.e., networks) and points (i.e., patches) to a
# graph object, and calculate network-wise distances between each pair of
# habitat patches. Output a matrix of pairwise distances, pairDists, which
# is also saved as a .RData object in the folder RDataOutDist. These objects
# are indexed by iteration number y in newNreps (e.g., pairDists.1.RData).
#
# Requirements: This code is designed for use with GRIP[1], and RAMAS GIS
# 5.0[2]; however, the code could be modified for stand-alone use. Regardless
# of whether NetworkDistances is used alone or with GRIP2, read the
# NetworkDistances user manual[3] for additional details.
#
# Notes: Please contact the authors if you have suggestions, comments, or
# concerns. Additionally, we are attempting to keep track of this code's use;
# please contact the authors if the code was useful for research. We recommend
# that users verify their output by checking various statistics and graphs. We
# assume that the user has a working ability with R[4], and is familiar with
# spatial data. It is crucial that users verify that distances are in the
# correct units throughout this script! This code comes with absolutely no
# warranty.
#
# References:
# [1] Curtis, J. M. R. and Naujokaitis-Lewis, I. R. 2008. Source code for the
#     program GRIP 1.0 (Generation of Random Input Parameters).
#     URL http://esapubs.org/archive/appl/A018/033/suppl-1.htm. Ecological
#     Archives: A018-033-S1 (Supplement)
# [2] Akcakaya, H. R. 2005. RAMAS GIS: Linking spatial data with population
#     viability analysis. Applied Biomathematics. URL www.ramas.com. User
#     manual for version 5
# [3] Grinnell, M. H. and Curtis, J. M. R. 2011. User manual for
#     NetworkDistances 1.0: Calculating network-wise distances between habitat
#     patches for spatially restricted species. Can. Tech. Rep. Fish. Aquat.
#     Sci. 2960: iv + 29 p.
# [4] RDCT (R Development Core Team). 2011. R: A language and environment for
#     statistical computing. URL www.R-project.org. R foundation for
#     Statistical Computing. Vienna, Austria. R version 2.13.0
#
###############################################################################

###########################################
##### Start file 'NetworkDistances.R' #####
###########################################
```

```

# Ensure that all the required user-specified inputs are present. Note: if
# NetworkDistances is not called from GRIP2, these parameter values need to be
# initialized prior to here. Also, ensure the proper file structure has been
# created, the required habitat patch information is available (i.e., the ptc
# and asc arguments to the function GridToUTM), and define y.
if( !all(exists(x=c("maxD", "doSnow", "cType", "nCores", "coreName", "nZone",
            "doSpatPlots", "networks"))) ) {
  # Stop everything
  stop( "Missing required user-specified inputs for 'NetworkDistances.R'" )
}  # End ensure user-inputs

# Create a temporary directory to hold output each iteration
if( "RDataOutTemp" %in% list.files( ) ) {
  # Delete the old directory
  unlink( x="RDataOutTemp", recursive=TRUE )
  # Create a new directory
  dir.create( path="RDataOutTemp" )
  # End if directory exists, otherwise
} else {
  # Make a new directory
  dir.create( path="RDataOutTemp" )
}  # End if the directory does not exist

# Get habitat patch locations (referenced via the grid) from the .ptc file, and
# grid georeference info from the .asc file (to georeference the patches).
# Convert to UTM and return habitat patches: patchesUTM. Note that users *must*
# verify that this function specifies distance units correctly: as is, this
# function expects distances in m (asc) and km (ptc), and converts to m.
GridToUTM <- function( ptc, asc ) {
  # Get the first item from the first nmax lines in the ptc file
  firstItemPTC <- scan( file=ptc, skip=0, sep=",", what="character",
      quiet=TRUE, flush=TRUE, blank.lines.skip=FALSE, nmax=200 )
  # Get line info for the habitat patch data (usually line 58)
  linePatchPTC <- grep( pattern=paste(Npops, "populations", sep=" "),
      x=firstItemPTC, perl=TRUE )[1]
  # If linePatchPTC is NA (possibly due to not enough lines), search again
  if( is.na(linePatchPTC) ) {
    # Get the first item from all the lines in the ptc file: this can take
    # a long time
    firstItemPTC <- scan( file=ptc, skip=0, sep=",", what="character",
        quiet=TRUE, flush=TRUE, blank.lines.skip=FALSE )
    # Get line info for the habitat patch data
    linePatchPTC <- grep( pattern=paste(Npops, "populations", sep=" "),
        x=firstItemPTC, perl=TRUE )[1]
    # Print a warning
    warning( "Increase 'nmax' in firstItemPTC() to avoid scanning whole file" )
  }  # End if linePatchPTC is NA
  # If linePatchPTC is *still* NA, error
  if( is.na(linePatchPTC) ) {
    # Error message
    stop( "Check file '", ptc, "': unable to find patch info" )
  } # End if linePatchPTC is *still* NA
  # Scan to get patch info into a list: Site, X, and Y (in km, referenced via
  # the grid)
  patchData <- scan( file=ptc, skip=linePatchPTC, nlines=Npops, sep=",",
      what=list(Site="", X=0, Y=0), quiet=TRUE, flush=TRUE )
  # Get the first item from the first nmax lines in the (sometimes large) .asc
  # file. This *should* contain the required lines, since they are usually at
  # the top of the .asc file (as a header).
  firstItemASC <- scan( file=asc, skip=0, sep="\t", what="character",
      quiet=TRUE, flush=TRUE, blank.lines.skip=FALSE, nmax=10 )
  # Get the line info for the X location
  lineXASC <- grep( pattern="xllcorner", x=firstItemASC, perl=TRUE )[1]
  # If lineXASC is NA (possibly due to not enough lines), search again
```

```
  if( is.na(lineXASC) ) {
    # Get the first item from all lines in the asc file -- this can take
    # much longer!
    firstItemASC <- scan( file=asc, skip=0, sep="\t", what="character",
        quiet=TRUE, flush=TRUE, blank.lines.skip=FALSE )
    # Get the line info for the X location
    lineXASC <- grep( pattern="xllcorner", x=firstItemASC, perl=TRUE )[1]
    # Print a warning
    warning( "Increase 'nmax' in firstItemASC() to avoid scanning whole file" )
  }  # End if lineXASC is NA
  # If lineXASC is *still* NA, error
  if( is.na(lineXASC) ) {
    # Error message
    stop( "Check file '", asc, "': unable to georeference the grid" )
  } # End if lineXASC is *still* NA
  # Georeference the grid's location: get the left side 'X' location (UTM, m)
  ASCgridLLX <- as.numeric( scan(file=asc, skip=lineXASC - 1, sep="\t",
          what=list(NULL, "numeric"), quiet=TRUE, flush=TRUE, nlines=1)[[2]] )
  # Get the line info for the Y location
  lineYASC <- grep( pattern="yllcorner", x=firstItemASC, perl=TRUE )[1]
  # Georeference the grid's location: get the lower 'Y' location (UTM, m)
  ASCgridLLY <- as.numeric( scan(file=asc, skip=lineYASC - 1, sep="\t",
          what=list(NULL, "numeric"), quiet=TRUE, flush=TRUE, nlines=1)[[2]] )
  # Get the line info for the cell size
  lineCellASC <- grep( pattern="cellsize", x=firstItemASC, perl=TRUE )[1]
  # Get the grid cell size (m)
  ASCgridSize <- as.numeric( scan(file=asc, skip=lineCellASC - 1, sep="\t",
          what=list(NULL, "numeric"), quiet=TRUE, flush=TRUE, nlines=1)[[2]] )
  # Get the line info for the number of rows
  lineRowsASC <- grep( pattern="nrows", x=firstItemASC, perl=TRUE )[1]
  # Get the number of rows
  ASCgridRows <- as.numeric( scan(file=asc, skip=lineRowsASC - 1, sep="\t",
          what=list(NULL, "numeric"), quiet=TRUE, flush=TRUE, nlines=1)[[2]] )
  # Get the line info for the number of columns
  lineColsASC <- grep( pattern="ncols", x=firstItemASC, perl=TRUE )[1]
  # Get the number of columns (grid X dimension)
  ASCgridCols <- as.numeric( scan(file=asc, skip=lineColsASC - 1, sep="\t",
          what=list(NULL, "numeric"), quiet=TRUE, flush=TRUE, nlines=1)[[2]] )
  # If any of lineYASC, lineCellASC, lineRowsASC, or lineColsASC is NA, error
  if( is.na(lineYASC) | is.na(lineCellASC) | is.na(lineRowsASC) |
      is.na(lineColsASC) ) {
    # Error message
    stop( "Check file '", asc, "': unable to georeference the grid (2)" )
  } # End if any of lineYASC, lineCellASC, lineRowsASC, or lineRowsASC is NA
  # Put scanned data into a dataframe with columns EID, X (convert km to m),
  # Y (convert km to m), and Site. Note that users *must* verify that this
  # function specifies distance units correctly
  patchDF <- data.frame( EID=1:Npops, X=patchData$X*1000, Y=patchData$Y*1000,
      Site=patchData$Site )
  # Find the top of the grid (UTM, m)
  gridTop <- ASCgridLLY + ( ASCgridRows * ASCgridSize )
  # Find the right side of the grid (UTM, m)
  gridRight <- ASCgridLLX + ( ASCgridCols * ASCgridSize )
  # Convert patch (X, Y) locations from grid to UTM (m; currently, (X, Y)
  # indicates the number of grid cells from the upper-left corner of the grid)
  patchDF$X <- ASCgridLLX + patchDF$X
  patchDF$Y <- gridTop - patchDF$Y
  # Convert to EventData (georeferenced in UTM, m for PBSmapping)
  patchEvent <- as.EventData( x=patchDF, projection="UTM", zone=nZone )
  # Set up a list to return
  res <- list( patchEvent=patchEvent,
      extent=list(top=gridTop, bottom=ASCgridLLY, left=ASCgridLLX,
          right=gridRight) )
  # Return the list (habitat patches (as EventData) and grid extent)
  return( res )
```

```
}  # End GridtoUTM function

# Run the GridToUTM function to get the patches and grid extent
getGridToUTM <- GridToUTM( ptc=ptcfile, asc=OrigPatchmap )

# Get the patches
patchesUTM <- getGridToUTM$patchEvent

# Get the grid extent (for the plot below)
gridExtent <- getGridToUTM$extent

# Loop over the number of networks
for( i in 1:length(networks) ) {

  # Get the river network (via watershed name). Note that users *must* verify
  # that this function specifies distance units correctly: as is, this function
  # expects distances in m.
  netVect <- importShapefile( readDBF=TRUE, projection="UTM", zone=nZone,
      fn=paste("NetworkShapefiles/", networks[i], ".shp", sep="") )

  # Select only the required columns (to reduce object size)
  netVect <- subset( netVect, select=c(PID, SID, POS, X, Y) )

  # Get the original river points
  patches <- patchesUTM

  # If the saved data object from a previous simulation is NOT available in the
  # working directory, run the entire analysis (this may take some time) and
  # save the required object for the next simulation
  if( !paste(networks[i], ".RData", sep="") %in%
      list.files("SavedNetworkObjects") ) {
    # Create a column to indicate the unique ID for each unique node. Note that
    # some nodes are repeated (i.e., the bottom coordinates of two streams that
    # merge (i.e., become one large stream) will be the same as the top
    # coordinates of the stream that they merge into). These nodes need to have
    # the same ID numbers for the graph object to consider them the same (which
    # they are).
    CreateUniqueID <- function( dat ) {
      # Select the columns of XY data from the data.frame and load as a matrix
      mat <- matrix( cbind(dat$X, dat$Y), nrow=nrow(dat), ncol=2 )
      # Initialize a vector to hold unique IDs
      vec <- 0
      # Fill vec with consecutive numbers for non-duplicated rows
      vec[!duplicated(mat)] <- 1:nrow( mat[!duplicated(mat), ] )
      # Get unique duplicated rows
      dups <- unique( mat[duplicated(mat), ] )
      # Loop over unique duplicated rows
      for( i in 1:nrow(dups) ) {
        # Determine the indices in mat that match the ith duplicated row
        iInd <- mat[ ,1] == dups[i, 1] & mat[ ,2] == dups[i, 2]
        # Set duplicate rows to the first number (the ID)
        vec[iInd] <- vec[iInd][1]
      }  # End loop over unique dups
      # Add the data to dat, and initialize columns for site name and distance
      dat$UniqueID <- vec
      dat$SiteName <- NA
      dat$dist <- NA
      # Return the site locations
      return( dat )
    }  # End CreateUniqueID function
    netVect <- CreateUniqueID( dat=netVect )
    # Collect garbage
    gc( )
    # Get locations "to" and "from", as well as distances between location
    # pairs
```

```
  GetToFromDist <- function( dat ) {
    mat <- matrix( NA, nrow=nrow(dat), ncol=7 )
    colnames( mat ) <- c( "frID", "toID", "frX", "frY", "toX", "toY",
        "dist" )
    count <- 0  # Initialize counter (for rows)
    # Pull required columns into a matrix
    dat <- as.matrix( dat[, 1:6], ncol=6, byrow=TRUE )
    # Loop over PIDs
    for( i in 1:length(unique(dat[, "PID"])) ) {
      datPID <- subset( dat, dat[, "PID"] == unique(dat[, "PID"])[i] )
      # Loop over SIDs within PIDs
      for( j in 1:length(unique(datPID[, "SID"])) ) {
        datSID <- subset( datPID,
            datPID[, "SID"] == unique(datPID[,"SID"])[j] )
        # Loop over rows within SIDs within PIDs
        for(  k in 1:(nrow(datSID)-1) ) {
          # Update the counter for new row
          count <- count + 1
          # Get coordinates
          mat[count, "frX"] <- datSID[k, "X"]
          mat[count, "frY"] <- datSID[k, "Y"]
          mat[count, "toX"] <- datSID[(k+1), "X"]
          mat[count, "toY"] <- datSID[(k+1), "Y"]
          # Get the UniqueID (i.e., the river node numbers)
          mat[count, "frID"] <- datSID[k, "UniqueID"]
          mat[count, "toID"] <- datSID[(k+1), "UniqueID"]
          # Calculate the distance
          mat[count, "dist"] <- sqrt( (datSID[k, "X"] - datSID[(k+1), "X"])^2
                  + (datSID[k, "Y"] - datSID[(k+1), "Y"])^2 )
        }  # End loop over rows within SID within PID
      }  # End loop over SID within PID
    }  # End loop over PID
    # Remove rows that are NA (these are due to repeated points)
    return( na.omit(mat) )
  }  # End GetToFromDist function
  toFromDist <- GetToFromDist( dat=netVect )
  # Collect garbage
  gc( )
  # Change the class of the toFromDist to a data.frame to allow factors for
  # node names (population sampling points)
  tfTable <- data.frame( frID=toFromDist[, "frID"],
      toID=toFromDist[, "toID"], frX=toFromDist[, "frX"],
      frY=toFromDist[, "frY"], toX=toFromDist[ ,"toX"],
      toY=toFromDist[, "toY"], dist=toFromDist[, "dist"] )
  # Remove toFromDist to save memory (this can be a large object)
  rm( toFromDist )
  # Save the required objects for the next simulation
  save( list=c("netVect", "tfTable"),
      file=paste("SavedNetworkObjects/", networks[i], ".RData", sep="") )
}  # End if the saved object is not present in the working directory

# If the saved data object from a previous simulation IS available, load the
# saved object to reduce computation time if desired
if( paste(networks[i], ".RData", sep="") %in%
    list.files("SavedNetworkObjects") ) {
  # Load the object
  load( file=paste("SavedNetworkObjects/", networks[i], ".RData", sep="") )
  # Ensure that site names and distances from the last iteration are NA
  netVect$SiteName <- NA
  netVect$DistToSite <- NA
} # End if using saved object

# Ensure that all sampling points have unique IDs
EnsureUniqueSites <- function( dat ) {
  # Letter index for first site
```

```r
  iLet <- 0
  # Get the original site names
  sites <- as.character( dat$Site )
  # Print a warning that site(s) have been renamed
  warning( "Duplicate site name(s) were renamed (", networks[i],")" )
  # Iterative rename function
  RenameRename <- function( siteNames ) {
    # Index the duplicated sites
    ind <- duplicated( siteNames )
    # Rename the sites
    siteNames[ind] <- paste( siteNames[ind], letters[iLet], sep="" )
    return( siteNames )
  }  # End RenameRename function
  # Send the site names to the RenameRename function while there are
  # duplicates
  while( TRUE %in% duplicated(sites) ) {
    # Update the counter for new letters
    iLet <- iLet + 1
    # Rename sites
    sites <- RenameRename( siteNames=sites )
  }  # End while loop
  # Return the new site names as factors
  return( as.factor(sites) )
}  # End CreateUniqueID function

# If there are duplicate site names, rename them
if( TRUE %in% duplicated(patches$Site) ){
  patches$Site <- EnsureUniqueSites( dat=patches )
}  # End if duplicate rows

# Collect garbage
gc( )

# Add the sampling points to the to-from matrix at the nearest node, within
# maxD units. Outputs the minimum distances for each population, and updates
# the nodes that are now population points. If no nodes are within maxD units
# of the point, the output is NA, and the point is ignored.
SnapToNodes <- function( pops ) {
  # Coordinates
  pt <- c( as.numeric(pops["X"]), as.numeric(pops["Y"]) )
  # Function to get the set of nodes within some proportion of maxD units
  # from pt. Returns a subset of netVect that is within the specified
  # distance (up to a maximum of maxD).
  GetPoints <- function( srchD ) {
    # Get outer range in X
    xs <- c( pt[1] - srchD, pt[1] + srchD )
    # Get outer range in Y
    ys <- c( pt[2] - srchD, pt[2] + srchD )
    # Select the river nodes within the outer range to limit the search
    subRiv <- netVect[ netVect$X>xs[1] & netVect$X<xs[2] & netVect$Y>ys[1] &
            netVect$Y<ys[2], ]
    # Return the subset of netVect
    return( subRiv )
  }  # End GetPoints function
  # Run the GetPoints function with maxD/4
  iNodes <- GetPoints( srchD=maxD/4 )
  # If there are no nodes, run GetPoints again with maxD
  if( dim(iNodes)[1] == 0 )  iNodes <- GetPoints( srchD=maxD )
  # Set up a temporary vector to hold distances
  dVec <- 0.
  # Continue only if there are elements to iNodes
  if( dim(iNodes)[1] >= 1 ) {
    # Loop over nodes and calculate Euclidian distance
    for( i in 1:nrow(iNodes) ) {
      dVec[i] <- sqrt( (pt[1] - iNodes$X[i])^2 + (pt[2] - iNodes$Y[i])^2 )
```

```
    }  # End loop over rows in nodes
    # Ensure that the minimum distance is <= maxD (note that this might not
    # be true because dVec's are for points within the box, but we need to
    # ensure that the radius is <= maxD
    if( min(dVec) <= maxD ) {
      # Determine the position of the closest (select only one if a tie)
      iRow <- which(min(dVec) == dVec)[1]
      # Get its unique ID
      iID <- iNodes$UniqueID[iRow]
      # Determine whether the node is already a site. This can occur if two
      # sampling locations are close together, and snap to the same node.
      # Note that iID is a single number, but may reference multiple nodes;
      # for example, the node is a junction (i.e., location at which two
      # streams merge)
      # If the node is not already a site:
      if( is.na(netVect$SiteName[netVect$UniqueID == iID])[1] ) {
        # Update the node to reflect that it's a sampling location (from)
        tfTable$frID[tfTable$frID == iID] <<- pops["Site"]
        # Update the node to reflect that it's a sampling location (to)
        tfTable$toID[tfTable$toID == iID] <<- pops["Site"]
        # Also, update netVect to indicate population site and distance to
        # node
        netVect$SiteName[netVect$UniqueID == iID] <<- pops["Site"]
        netVect$DistToSite[netVect$UniqueID == iID] <<- min( dVec )
      }  # End if the node is not already a site
      # If the node is already a site:
      else {
        # Get the old site name
        oldName <- netVect$SiteName[netVect$UniqueID == iID][1]
        oldXY <- c( netVect$X[netVect$UniqueID == iID][1],
            netVect$Y[netVect$UniqueID == iID][1] )
        # Make a new row to add to the dataframe, with the distance between
        # the new and old sites equal to zero (i.e., same node/location)
        newRow <- tfTable[1, ]
        newRow$frID <- pops["Site"]
        newRow$toID <- oldName
        newRow$frX <- oldXY[1]
        newRow$frY <- oldXY[2]
        newRow$toX <- oldXY[1]
        newRow$toY <- oldXY[2]
        newRow$dist <- 0.
        # Finally, append the new rows to the to-from matrix
        tfTable <<- rbind( tfTable, newRow )
        # Also, update netVect to indicate that the node has two population
        # sites and the max of the distances
        netVect$SiteName[netVect$UniqueID == iID] <<-
            paste(netVect$SiteName[netVect$UniqueID == iID][1], pops["Site"],
                sep="." )
        netVect$DistToSite[netVect$UniqueID == iID] <<-
            max(min(dVec), netVect$DistToSite[netVect$UniqueID == iID][1])
        # Print a warning to indicate the node is already a site
        warning( "Sites '", pops["Site"], "' and '", oldName,
            "' are on the same node (", networks[i], ")" )
      }  # End if the node is already a site
    }  # End if min(dVec) <= maxD
    # Otherwise, min(dVec) is > maxD
    else  dVec <- NA
  }  # End if there are rows in iNodes
  # If there are no nodes, dist is NA, and print a warning
  if( dim(iNodes)[1] == 0 )  dVec <- NA
  # Return the distance between the population point and the node
  return( min(dVec) )
}  # End SnapToNodes function

# Add patche to network
```

```
  minDists <- apply( X=patches, MARGIN=1, FUN=SnapToNodes )

  # Append minDists to minDistMat. This will allow the identification of
  # patches that occur in multiple networks (i.e., when maxD is too large),
  # and assign the patch to the network that is closest. If it's the first
  # iteration
  if( i==1 ) {
    # Set up the empty matrix
    minDistMat <- matrix(NA, nrow=nrow(patches), ncol=length(networks) )
    # Name columns as networks
    colnames( minDistMat ) <- networks
    # Name rows as patches
    rownames( minDistMat ) <- patches$Site
    # Add minDists to the 1st column
    minDistMat[, 1] <- minDists
    # End if the 1st iteration
  } else {
    # Add minDists to the ith column
    minDistMat[, i] <- minDists
  }  # End if iteration is 2nd or more

  # Collect garbage
  gc( )

  # Make the graph object. Take in the toFromDist data.frame, and output the
  # graph object indicating paths between vertices (some of which are
  # population sampling points) and distances (calculated as weights in the
  # graph object). First, check for and remove edges that are specified
  # multiple times. To save time, only do this if the network has patches.
  if( any(patches$Site %in% tfTable$frID) |
      any(patches$Site %in% tfTable$toID) ) {
    # Make a graph will all the nodes
    netGraphAll <- graph.data.frame( tfTable[ ,1:2], directed=FALSE )
    # Remove nodes that are duplicated
    tfTable <- tfTable[!is.multiple( graph=netGraphAll), ]
    # Make a graph with unique nodes
    netGraph <- ftM2graphNEL( ft=cbind(tfTable$toID, tfTable$frID),
        edgemode="undirected", W=tfTable$dist )
  }  # End if the network has patches

  # Collect garbage
  gc( )

  # Calculate distance matrix between pairs of sampling points. If a pair of
  # points are not connected (i.e., not joined by lines), the distance is NA.
  # The diagonal is zero (i.e., distance between and point and itself, and the
  # distance "to" equals the distance "from" (i.e., direction is irrelevant).
  CalcDistMatrix <- function( dat ) {
    # Get character vector of unique site names from the graph object. This is
    # important because the distance function will work hard to calculate
    # distances for unconnected patches, which takes a very long time. So,
    # only include patches that are actually on the graph
    patchSites <- patches$Site[ which(patches$Site %in% nodes(netGraph)) ]
    patchSites <- as.character( patchSites )
    # Set up the empty distance matrix, with site names for rows and cols
    dMat <- matrix( NA, nrow=length(patchSites), ncol=length(patchSites) )
    colnames( dMat ) <- patchSites
    rownames( dMat ) <- patchSites
    # First, generate a table of indices for the upper triangle of dMat
    num <- length( patchSites )
    idx <- expand.grid( i=1:num, j=1:num )[upper.tri( diag(num),
        diag=FALSE ), ]
    # Function to get the distance: apply over rows of idx to fill in the upper
    # triangle of the distance matrix
    GetDistance <- function( dat, patchSites=patchSites, netGraph=netGraph ) {
```

```
      # Load the required library
      require( RBGL )
      # Get the pair of point names
      ptPair <- c( patchSites[dat[1]], patchSites[dat[2]] )
      # Determine the shortest route and distance between the pair of points
      pairDist <- sp.between( g=netGraph, start=ptPair[1], finish=ptPair[2],
          detail=FALSE )
      # Pull out the distance between the pair
      return( pairDist[[1]]$length )
    }  # End GetDistance function
    # If using only one core (and idx has some data)
    if( !doSnow & dim(idx)[1] > 0 ) {
      # Apply the indices to the function and output distances
      dists <- apply( X=idx, MARGIN=1, FUN=GetDistance,
          patchSites=patchSites, netGraph=netGraph )
    }  # End if only one core
    # If using multiple cores (and idx has some data)
    if( doSnow & dim(idx)[1] > 0 ) {
      # Apply the indices to the function and output distances
      dists <- parApply( cl=snowClust, X=idx, MARGIN=1, FUN=GetDistance,
          patchSites=patchSites, netGraph=netGraph )
    }  # End if using more than one core
    # If idx does not have any data
    if( dim(idx)[1] == 0 ) dists <- NA
    # Place distances into the upper triangle
    dMat[upper.tri(dMat, diag=FALSE)] <- dists
    # And the lower triangle
    dMat[lower.tri(dMat)] <- t(dMat)[lower.tri(dMat)]
    # Update the diagonal
    diag( dMat ) <- 0.
    # Return the matrix
    return( dMat )
  }  # End CalcDistMatrix function

  # To save time, calculate distances only if the network has habitat patches
  if( any(patches$Site %in% tfTable$frID) |
      any(patches$Site %in% tfTable$toID) ) {
    distMat <- CalcDistMatrix( dat=netGraph )
  }  # End if there are patches

  # If there aren't any patches, distMat is "empty"
  if( !any(patches$Site %in% tfTable$frID) &
      !any(patches$Site %in% tfTable$toID) ) {
    distMat <- matrix( NA, nrow=0, ncol=0 )
  }  # End if there are no patches

  # Collect garbage
  gc( )

  # Save the distance matrix as an .RData object
  save( distMat,
      file=paste("RDataOutTemp/distMat.", networks[i], ".RData", sep="") )

}  # End i loop over the number of networks

# Subset minDistMat to only have patches that are in a network, and to only
# have networks that have patches. Also, each patch should only be in one
# network, which is the network that had the nearest node. This is to ensure
# that there are no duplicates in pairDists (output from CombineDistances,
# below)
RemoveDupPatches <- function( mat ) {
  # Determine which rows are ALL NA (i.e., patches that were not snapped
  # to a node in any networks -- these patches are omitted from the analysis)
  naRows <- apply(  X=mat, MARGIN=1, FUN=function(x) all(is.na(x))  )
  # Take the inverse
```

```
  naRows2 <- naRows == FALSE
  # Select the rows with (patches) that were snapped to node(s)
  mat2 <- subset( mat, subset=naRows2 )
  # Loop over rows in mat, and make sure that each patch only "belongs" to
  # one node (network); if there are two, set the further one to NA to ensure
  # that the patch is only in one network
  for( k in 1:nrow(mat2) ) {
    # Get the elements in row k that are not NA
    dat <- mat2[k, is.na(mat2[k, ]) == FALSE]
    # If there are more than one, pick the smallest one. Otherwise, ignore
    if( length(dat) >= 2 ) {
      # Get the min
      minDat <- min( dat )
      # Set the other(s) in the row to NA
      mat2[k, which(mat2[k, ] != minDat)] <- NA
    }  # End if more than 2 networks
  }  # End loop over k rows in mat
  # Determine which columns are ALL NA (i.e., networks with no patches)
  naCols <- apply( X=mat2, MARGIN=2, FUN=function(x) all(is.na(x)) )
  # Select the cols (networks) that contain patches
  mat3 <- subset( mat2, select=which(naCols==FALSE) )
  # Return mat: all patches are in one (and only one) network, and all
  # networks have (at least) one node
  return( mat3 )
}  # End RemoveDupPatches function

# If minDistMat is not all NA
if( !all(is.na(minDistMat)) ) {
  # Get the subset of patches and networks with patches
  patchNetwork <- RemoveDupPatches( mat=minDistMat )
}  # End if is not all NA

# If minDistMat is all NA, set patchNetwork to a 0x0 matrix
if( all(is.na(minDistMat)) ) patchNetwork <- matrix( NA, nrow=0, ncol=0 )

# Finally, combine the distance matrices from each subset
CombineDistances <- function( dat ) {
  # Initialize the big matrix to hold all the pairwise distances
  mat <- matrix( NA, nrow=nrow(patchesUTM), ncol=nrow(patchesUTM) )
  # Set up row and columns indices for the top left corner of the big matrix
  iRC <- 1
  # Initialize a vector to hold point names
  ptNames <- vector( )
  # Loop over data subsets
  for( i in 1:length(dat) ) {
    # Load the distance matrix
    load( file=paste("RDataOutTemp/distMat.", dat[i], ".RData", sep="") )
    # Get the right column (network) from patchNetwork
    iCol <- which( colnames(patchNetwork) == dat[i] )
    # Get the rows that indicate the network's patches
    iRows <- which( is.na(patchNetwork[, iCol]) == FALSE )
    # Get the names of patches from patchNetwork (these are the only ones that
    # should be in distMat
    pnPatches <- rownames( patchNetwork )[iRows]
    # Check that distMat doesn't have patches that aren't supposed to be in
    # a different (i.e., closer) network
    if( all(colnames(distMat) %in% pnPatches) == FALSE ) {
      # Determine which patches should be included
      inclPatch <- which(colnames(distMat) %in% pnPatches)
      # Subset distMat to include only patches that should be in the network
      distMat <- distMat[inclPatch, inclPatch]
      # If distMat is now empty (i.e., there is only one patch) ensure that
      # it's a matrix with 1 row and 1 column with correct names and
      # distance=0
      if( all(distMat == 0) ) {
```

```
        # Make the matrix
        distMat <- matrix( 0, nrow=1, ncol=1 )
        # Give it column and row names
        rownames( distMat ) <- pnPatches
        colnames( distMat ) <- pnPatches
      }  # End if distMat is zero
    }  # End if there are extra patches in distMat
    # If there is data in distMat, append the distances and names
    if( !is.null(dim(distMat)) ) {
      # Get the number of rows
      nr <- nrow( distMat )
      # If there are rows, get data
      if( nr >= 1 ) {
        # Append the names to the vector
        ptNames[iRC:(iRC+nr-1)] <- rownames( distMat )
        # Append distMat to the big matrix
        mat[(iRC:(iRC+nr-1)), (iRC:(iRC+nr-1))] <- distMat
        # Update index for rows and columns
        iRC <- iRC + nr
      }  # End if thre are rows
    }  # End if distMat had data
  }  # End loop over subsets
  # Remove empty rows
  mat <- matrix( mat[(1:length(ptNames)), (1:length(ptNames))],
      nrow=length(ptNames) )
  # If there are rows and columns
  if( !is.null(dim(mat)) )  {
    # Add row and column names
    rownames( mat ) <- ptNames
    colnames( mat ) <- ptNames
  }  # End if there are rows and columns
  # Return the distance matrix
  return( mat )
}  # End CombineDistances function

# Run this function if there is at least one row in patchNetwork (i.e., there
# is at least one point in the area). This only needs to be done for networks
# that have patches (i.e., columns in the object patchNetwork)
if( dim(patchNetwork)[1] >= 1 )  {
  pairDists <- CombineDistances( dat=colnames(patchNetwork) )
}  # End if there are points

# If there are zero points, the pairDists matrix is 0x0.
if( dim(patchNetwork)[1] == 0 )  pairDists <- matrix( NA, nrow=0, ncol=0 )

# Collect garbage
gc( )

# Fill in the pairDists matrix with NA if any patches were omitted
FillPairDists <- function( mat ) {
  # Get indices for the missing sites
  indMissing <- which( !patches$Site %in% colnames(mat) )
  # Get missing site names
  noPatch <- patches$Site[indMissing]
  # Set up a matrix for columns
  colMat <- matrix( NA, nrow=nrow(mat), ncol=length(noPatch) )
  # Give it names
  colnames( colMat ) <- noPatch
  # Append cols
  mat <- cbind( mat, colMat )
  # Set up a matrix for rows
  rowMat <- matrix( NA, nrow=length(noPatch), ncol=ncol(mat) )
  # Give it names
  rownames( rowMat ) <- noPatch
  # Append rows
```

```
  mat <- rbind( mat, rowMat )
  # Return the new table
  return( mat )
}  # End FillPairDists function

# Check if all the sites are in pairDists
if( !all( patches$Site %in% colnames(pairDists) ) ) {
  # If not, run the function
  pairDists <- FillPairDists( mat=pairDists )
}  # End if there are missing sites

# Order pairDists so that patches are in the original order
pairDists <- pairDists[mixedsort(colnames(pairDists)),
    mixedsort(colnames(pairDists))]

# Convert distances from m to km (to conform with GRIP2 and RAMAS); it is
# *crucial* that users verify that distances are in the correct units!
pairDists <- pairDists / 1000

# Set the diagonal (i.e., from the patch to itself) from 0.0 to NA (to
# conform with GRIP2 and RAMAS)
diag( pairDists ) <- NA

# Save the habitat patches
save( patches, patchesUTM,
    file=paste("RDataOutDist/patches.", y, ".RData", sep="") )

# Save the pairwise distances with index y in newNreps
save( pairDists, file=paste("RDataOutDist/pairDists.", y, ".RData", sep="") )

# Save the matrices indicating patches, networks, and distances snapped
save( minDistMat, patchNetwork,
    file=paste("RDataOutDist/patchNetworkDist.", y, ".RData", sep="") )

###########################################
##### End of file 'NetworkDistances.R' #####
###########################################
```

INDEX