PatchMapperGuide

Version 1
February 18, 2024
Kurt Riitters

PatchMapper.c. A tool to identify and map contiguous patches in a raster image, and optionally to perform other analyses of the patches in that image.

OVERVIEW

PatchMapper currently identifies contiguous patches by either the 4- or 8-neighbor rule, optionally re-coding input pixel values, and optionally writing the patchmap and/or performing additional analyses of the patches and writing a csv file containing patch-level statistics for each patch.

PatchMapper is a generic raster image analysis tool. It map be applied in a geographical context, in which case the image is a map, and for that application this Guide briefly describes format conversions and other issues specific to that context.

For both 4- and 8-neighbor rules, the patch statistics include patch number (from patchmap), the original input pixel class (from inputmap), patch size (number of pixels), patch origin (row, column; a pixel guaranteed to be a part of the patch), and the patch bounding box (min/max row/col). For the 4-neighbor rule ONLY, the statistics file includes three measures of interior or exterior perimeter length, and an indicator whether or not the patch “touches” either the map border or a missing pixel value).

In addition to writing output files, program execution also spews some information and statistics into the terminal (number of patches, average patch size by type, etc). If desired the program can be modified to route this information to additional output files. Please let the author know of any issues, mod requests, and suggestions to improve this Guide.

COMPILE/RUN

Linux only compilation: (windows data typing requires substantial changes)

```
gcc -std=c99 -m64 -Wall -fopenmp PatchMapper.c -o patchmapper  -O2 -lm
```

Linux execution:

```
./patchmapper
```

…with all input/output files in the current directory.

FILES

Input files

Required

size.txt	specify number of input rows and columns on input map
inputmap	unsigned 8-bit integer, bsq file*
parfile.txt	PatchMapper parameter specification

Optional

recode.txt	if parameter z = 1, specifies reclassification of input pixels

Output files

<u>Optional</u>

patchmap	if parameter x = 1, an unisigned 32-bit integer, bsq file*****
patchstatistics.csv	if parameter p = 1, a csv-format file

Georeference files

Typical projection and/or header files are not used (read or written) by PatchMapper. User may copy/modify these from the input map to the output map as needed*****.

***Notes**

<u>Data types and pre-post processing</u>: It is recommended to use the 'bsq' standards instead of 'envi' standards for input and output map format conversions. The ENVI standard does not support the data type of the patchmap, which can affect the capability of some R libraries (and GIS software) to use the patchmap if the envi standard is expected. This can be avoided by using command line gdal_translate for file conversions outside of R, specifying the bsq standard. Within R, the terra library supports the gdal_translate bsq capabilities. In these cases, the header file for the patchmap is the same as the inputmap, except for three fields -- NBITS (change 8 to 32), BANDROWBYTES and TOTALROWBYTES (change from whatever NCOLS is, to 4 times what NCOLS is). The bsq .prj file does not need to be modified. Note that the ".bsq" suffix must be added or removed, before or after file conversions.

<u>FILE SPECIFICATIONS</u>

size.txt

This file specifies the number of rows and columns in the inputmap. The file must be named "size.txt" and must contain two rows with the following format:

```
NROWS 4177
NCOLS 6448
```

Inputmap

This is the 8-bit input file. It must be named "inputmap". (no extension)

parfile.txt

This file specifies the PatchMapper run parameters. The file must be named "parfile.txt". PatchMapper currently does no parameter checking, so please include all five lines in the following format (the alpha letter in each row can be upper or lower case):

m 0	pixel value representing missing value (after re-coding if parameter Z = 1; default=0) <u>***Do not change the missing value code; always use m 0, and recode as needed to conform to "zero is missing" (fix coming)***</u>
z 1	re-code input pixels? (0=no; 1=yes; default = 0)
n 8	neighbor rule (either 4 or 8; default = 4)
x 1	output patch map? (0=no; 1=yes; default = 0)
p 1	calculate and export patch-level statistics? (0=no, 1=yes; default = 0)

recode.txt

This file specifies the optional re-classification of input pixels. The file must be named "recode.txt". Each line of the file will re-code one input pixel value. Use as many lines as needed, to a maximum of 255. Only the indicated pixel values will be changed; other pixel values are unaffected. Values must recode input values in the range [0,255] to re-coded values in the range [0,255]. Note that some inputmaps may have missing values as '255' -- ***Pending a minor fix (coming), always re-code any pixel values considered "missing" to 0 (zero), and do not change the m parameter in parfile.txt. Similarly, if an input pixel value 0 is not supposed to be missing, please re-code it to something else and do not change the m parameter in parfile.txt.*** Patches are identified based on the re-coded values. An example of a recode.txt file:

```
11 1
12 1
…
95 9
255 0
```

patchmap
This is the 32-bit output file.

patchstatistics.csv
This file contains the statistics for each patch. In the following, the columns with an asterisk are not output for 8-neighbor analyses. See following notes for additional explanations.

Patch_number	Same as in the patchmap file
Patch_type	The pixel value in the inputmap
Num_pixels	Number of pixels in the patch
*Out_edges	Exterior perimeter length in units of pixel edges
*In_edges	Interior perimeter length in units of pixel edges
*Out_pixels	Exterior perimeter length in units of whole pixels
Row_org	the row of the upper-left most pixel in the patch
Col_org	the column of the upper-left most pixel in the patch
*Touch_flag	Indicator. If "1" then the patch touches a border and/or a missing pixel
Row_max	Maximum row where the patch occurs
Col_min	Minimum column where the patch occurs
Col_max	Maximum column where the patch occurs

Notes for patchstatistics.csv

1. ***Rows and columns are numbered starting from base 0*.** For example, an input map of size = 8 rows X 10 columns will have maximum row and column numbers of 7 and 9 repectively.
2. Patch_type is the type ***after*** any optional re-coding.
3. Exterior perimeter refers to the outer shell of the patch; interior perimeter refers to the boundaries between the patch and all non-patch "inclusions" (holes in the patch)
4. Depending on exterior perimeter shape, the enclosing shell may require fewer pixels than edges (Consider an L-shaped patch for example). "Out_pixels" is probably just a curiosity since there is no corresponding "In_pixels".
5. Touch_flag refers only to the exterior perimeter of a patch.
6. Perimeter lengths include any perimeter segments that touch a border or a missing pixel.

7. Bounding box: Because program logic ensures that Row_org is the minimum row where the patch occurs, the bounding box of a patch is [(Row_org, Col_min), (Row_org, Col_max), (Row_max, Col_min), (Row_max, Col_max)]
8. Row_org and Col_org is useful for identifying at least one pixel that is guaranteed to be in the patch.

PROGRAMMING NOTES

PatchMapper.c is essentially a subset of the functionality of the circa-1992 program known as LandStat.c, which was developed by the author to support landscape pattern analysis of raster maps as first published in Riitters et al 1995 (doi: 10.1007/BF00158551). The original code implemented a highly efficient, flood-filling, linked-list queue algorithm that was modified slightly here to enable 8-neighbor patch identification in addition to 4-neighbor identification. The original code also implemented a “hand-on-the-wall” maze solution algorithm to identify 4-neighbor exterior perimeters, and modified that algorithm to also identify interior perimeters. The analogous perimeter algorithms for the 8-neighbor case have not been implemented in PatchMapper due to the extreme coding specificity (for efficiency) in the original code; perhaps a future version of PatchMapper could be developed to handle 8-neighbor. Unlike the original code, parallel processing is enabled in PatchMapper (the executable will use all cores available in the shell.