# PatchMapperGuide

Version 1.1
March 1, 2024
Kurt Riitters

PatchMapper.c.  A tool to identify and map patches (contiguous pixels of the same type) in a byte-value raster image, and optionally to perform other analyses of the patches in that image.

## OVERVIEW

PatchMapper 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 csv file(s) containing patch-level statistics for each patch.

PatchMapper is a generic raster image analysis tool. It may 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 relevant issues.

For both 4- and 8-neighbor rules, the patchmap file has sequential patch numbers (1,2,3…N), and a patch statistics file includes patch number (from patchmap), the original input pixel type (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, another patch 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).

## COMPILE/RUN

**Linux compilation:**

```
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**

Optional

|  |  |
|---|---|
| patchmap | if parameter x = 1, an unisigned 32-bit integer, bsq file |
| patchstatistics1.csv | if parameter p = 1, a csv-format file |
| patchstatistics2.csv | if parameter e = 1 AND parameter n = 4, 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**

The inputmap must be 8-bit unsigned integer, and the patchmap is 32-bit unsigned integer.

For the header and projection information, It is recommended to follow generic 'bsq' standards (separate header and projection files) 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 or GIS software to correctly read the patchmap if the envi standard is expected. Normally the user wants both input and output maps to be in a GeoTIFF format. In that case, an input GeoTIFF can be converted to the required inputmap bsq format by using command line gdal_translate for file conversions outside of R, specifying the bsq standard. Alternatively, 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 header produced by gdal_translate for 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). *In addition, ensure that the NODATA value for the patchmap is set to 0 (zero).* The projection (.prj) file does not need to be modified. Be sure to re-name the header and projection files as patchmap.hdr and patchmap.prj. Note that PatchMapper requires removal of the ".bsq" suffix from the inputmap.bsq before processing, and that suffix must be added to the patchmap after processing.

## MISSING DATA

Input image -- PatchMapper assumes that 0 (zero) represents missing data on the inputmap; see below for optional re-coding of input pixels to meet this requirement.

Output image – PatchMapper assigns patch numbers sequentially in [1,2,3…N), and the patchmap (see below) will represent missing data (i.e., not in a patch) with pixel value 0 (zero).

## RAM / DISK REQUIREMENTS

RAM – RAM required to identify patches and create the patchmap is approximately 6 times the number of bytes in the input image (the uncompressed bsq file). In addition, parameter P and/or parameter E are set to '1', additional RAM equal to the number of patches times 25 bytes per patch.

Example 1. Inputmap = 4,000 x 6000 pixels containing 2,000,000 patches.

If parameter E and parameter P are both zero:

RAM = 6*(4000 x 6000) = 144MB

If parameter E and/or parameter P are '1':

RAM = 144MB + (2000000 x 25) = 194MB

Example 2. Inputmap = 104,000 x 160,000 pixels containing 1,000,000,000 patches.
If parameter E and parameter P are both zero:
RAM = 6*(104000 x 160000) = 99.84GB
If parameter E and/or parameter P are '1':
RAM = 99.84GB + (1000000000 x 25) = 124.84GB

DISK – The output patchmap bsq file is 4 times the size of the inputput bsq. Conversion to compressed GeoTIFF's will help.
The size of the output patch statistics files are highly dependent on the number of patches. For example 1 above, the uncompressed disk sizes of the two csv files are roughly 115MB and 50MB.

## FILE SPECIFICATIONS

**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 four lines in the following format (the alpha letter in each row can be upper or lower case):

| | |
|---|---|
| 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). If yes, output file is "patchmap". |
| p 1 | Calculate and export basic patch-level statistics? (0=no, 1=yes; default = 0). If yes, output file is "patchstatistics1.csv". |
| e 1 | Calculate and export patch perimeter statistics? (0=no, 1=yes; default = 0). If yes, output file is "patchstatistics2.csv" Note this option is available only for the 4-neighbor rule, and will be skipped if parameter N = 8. |

**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. Only the indicated pixel values will be changed; other pixel values are unaffected. If there are multiple lines to re-code the same input value, the last one listed will be used. 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 "background" missing values as '255' and these must be re-coded to 0 (zero). If an input pixel value 0 does not indicate a missing value, please re-code it to a unique pixel value. The 'type' of each patch is defined by the re-coded values.*** An example of a recode.txt file:
0 42 (if an input value zero does not represent a missing value, the '42' here is arbitrary)
11 1

12 1
…
95 9
255 0 (depending on format conversions, some inputmaps have background missing as '255')

**patchmap**
This is the 32-bit output file.
**patchstatistics1.csv**
This file contains patch-level statistics. 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. |
| Row_org | The row of the upper-left most pixel in the patch. This is by definition Row_min. |
| Col_org | the column of the upper-left most pixel in the patch. |
| 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 patchstatistics1.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. 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) ].
4. (Row_org, Col_org) identifies one pixel that is guaranteed to be in the patch.

**patchstatistics2.csv**
This file contains patch-level statistics. See following notes for additional explanations.

| | |
|---|---|
| Patch_number | Same as in the patchmap file. |
| Out_edges | Number of outside edges (exterior perimeter) |
| In_edges | Number of inside edges (interior perimeter) |
| Out_pixels | Number of outside pixels (enclosing shell) |
| Touch_flag | "1" indicates the patch touches border OR a missing value. |

**Notes for patchstatistics2.csv**
1. Exterior perimeter refers to the outer perimeter of the patch; interior perimeter refers to the boundaries between the patch and all non-patch inclusions (holes in the patch).
2. 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".
3. Touch_flag refers only to the exterior perimeter of a patch.

4. Perimeter lengths include any perimeter segments that touch a border or a missing pixel.

**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; the algorithm was modified 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 (an 8-neighbor implementation may be developed in a future version). Parallel processing is enabled in PatchMapper and the executable will use all cores available in the shell (unfortunately, most of the heavy lifting cannot be executed in parallel).