# Marxan with Zones (v1.0.1)

Conservation Zoning using Spatially Explicit Annealing

A Manual Prepared for Ecotrust, the National Heritage Trust and the Applied Environmental Decision Analysis centre

**Matthew Watts**
**Carissa Klein**
**Romola Stewart**
**Ian Ball**
**Hugh Possingham**

November 2008

Suggested citation: Watts, M. E., C. K. Klein, R. Stewart, I. R. Ball, and H. P. Possingham. 2008. Marxan with Zones (V1.0.1): Conservation Zoning using Spatially Explicit Annealing, a Manual.

## Introduction

Marxan with Zones (called Marxan Z for short) is a decision support tool developed by The Ecology Centre, University of Queensland. A number of institutions have played a key role in supporting the development of Marxan Z software. We wish to acknowledge the funding received from Ecotrust, whose interest in this software development arose from the need to support the design of marine protected areas along California's coast as part of California's Marine Life Protection Act. We also acknowledge funding received from the National Heritage Trust and the Applied Environment Decision Analysis centre.

Marxan Z is an extension of Marxan software, developed by Ian Ball and Hugh Possingham. Marxan Z has the same functionality as Marxan but is able to allocate planning units to multiple zones (i.e. marine protected areas of various protection levels) and incorporate multiple costs into a systematic planning framework. The purpose of Marxan Z is to assign each planning unit in a study region to a particular zone in order to meet a number of ecological, social and economic objectives at a minimum total cost.

Marxan is a tool for designing efficiently configured protected area networks or reserve systems. Marxan Z extends on the range of problems the software can solve to include the near optimal allocation of resources to multiple-zone configurations.

This manual is separated into two major sections: 1) Using Marxan Z: Input and Output Files and 2) Objective Function and Algorithms Used in Marxan Z. The first section outlines the basic information necessary to understand and use Marxan Z. The second section of the manual provides additional technical details of Marxan Z. A list of useful publications that discuss or apply Marxan are provided. These references may be very useful in developing Marxan Z applications.

This manual was designed using the Marxan manual (v1.8.2) as a guide but was adapted to include the additional functionality of Marxan Z and exclude Marxan features that are not applicable to Marxan Z.

## Table of Contents

**1.0 USING MARXAN Z: INPUT AND OUTPUT FILES** ---- 3

**1.1 INPUT FILES** ---- **3**
- **Planning Unit File** ---- 3
- **Feature File** ---- 4
- **Planning Unit Versus Feature** ---- 5
- **Zones** ---- 6
- **Costs** ---- 6
- **Zone Cost** ---- 7
- **Boundary Length** ---- 8
- **Zone Boundary Cost** ---- 8
- **Planning Unit Zone** ---- 9
- **Planning Unit Lock** ---- 10
- **Zone Target and Zone Target 2** ---- 10
- **Zone Contribution and Zone Contribution 2** ---- 12
- **Input Parameter File** ---- 13

**1.2 OUTPUT FILES** ---- **19**
- **Solutions for Each Run** ---- 19
- **Best Solution for All Runs** ---- 19
- **Missing Values for Each Run** ---- 19
- **Summary Information** ---- 20
- **Scenario Details** ---- 21
- **Summed Solution** ---- 21
- **Screen Log File** ---- 22

**1.3 SCREEN OUTPUT** ---- **24**

**2.0 OBJECTIVE FUNCTION AND ALGORITHMS USED** ---- 26

**2.1 THE OBJECTIVE FUNCTION** ---- **26**
- **Cost of the Zone Configuration** ---- 27
- **Boundary Length and Fragmentation** ---- 27
- **Representation Requirements for Features** ---- 27
- **Feature Penalty** ---- 27
- **Feature Penalty Factor** ---- 30
- **Cost Threshold Penalty** ---- 30

**2.2 ALGORITHMS METHODS** ---- **31**
- **Simulated Annealing** ---- 31
- **Iterative Improvement** ---- 33

**3.0 ACKNOWLEDGEMENTS** ---- 35

**4.0 KEY REFERENCES** ---- 36

**Appendix A. Example Input Parameter File** ---- **40**

# 1.0 Using Marxan Z: Input and Output Files

## 1.1 INPUT FILES

There are seven fundamental input files used in Marxan Z. In addition, there are optional input files that facilitate additional functionality in Marxan Z.

| Fundamental Input Files | Optional Input Files |
|---|---|
| Planning Unit | Boundary Length |
| Feature | Zone Boundary Cost |
| Planning Unit Versus Feature | Planning Unit Zone |
| Zones | Planning Unit Lock |
| Costs | Zone Target or Zone Target 2 |
| Zone Cost | Zone Contribution or Zone Contribution 2 |
| Input Parameter | |

This section will describe the basic structure of each file. With the exception of the input parameter file, all files consist of a header line and a main body. The header line is a list of specific names describing what is contained in each column of the main body. Each file contains a set of required and optional header names for Marxan Z to run. The headers in each file must be written in all lower-case letters with no punctuation, spaces, or numerals (unless otherwise noted). All of the input files can be created in any spreadsheet or text editing software. Variables in a single line can be separated by a variety of characters, including spaces, tabs, or commas. An example of this format is shown in the planning unit file section.

**Planning Unit File**
Default name: "pu.dat"

Description: This file is required to run Marxan Z. It contains a list of all planning units in the study region and includes the costs associated with assigning a planning unit to a particular zone. Multiple costs can be indicated for each planning unit. The importance of each cost relative to the other costs can be expressed in the planning unit file or in the zone cost file (described later in this document). If the zone cost file is used to weigh each cost, the values of each cost should be normalized to the same scale in the planning unit file.

| Header | Required/Optional | Description |
|---|---|---|
| id | required | The numeric identifier for this planning unit. |
| costname | Optional, will use a default value of 1 if not specified | The individual cost of each planning unit. Multiple cost fields with different names can be used into Marxan Z's planning unit file. The header 'costname' can be replaced with the actual name of the cost but must not include delimiters (spaces, tabs, etc.). |

*Example – portion of pu.dat file*

```
id, salmonfishing, squidfishing
1, 34, 22
2, 4, 0
3, 40, 12
```

**Feature File**

Default name: "feat.dat"

Description: This file is required to run Marxan Z. Each feature is listed in this file and assigned a unique identifier. The range of spatial features that can be accepted include ecological, economic and social features. The areal target amount (or proportional target amount) of each feature to capture in the nominated zones is specified in this file. If an overall target is specified in this file through the target, prop, targetocc or propocc fields, then a zone contribution and or contribution 2 file must be used to specify which zones contribute to these targets, and the contribution rates of zones towards these targets. The overall target specified in the feature file and the contribution rates specified in the zone contribution file *work in tandem* to specify overall targets for features and differential contribution rates of zones to these targets. See the section below on zone contribution for a description of these files.

| Header | Required/Optional | Description |
|---|---|---|
| id | required | The numeric identifier for this feature. The id must be a positive integer. |
| target | optional if prop is used | The target amount (in unit of puvfeat.dat file) of the feature to include across all protected zones (i.e. overall target). |
| prop | optional if target is used | An alternative to target. This is the proportion of the total amount of the feature which must be included in the protected zones. A value of 0.3 would indicate that 30% of that feature should be protected. |

| Header | Required/Optional | Description |
|---|---|---|
| targetocc | optional | The number of occurrences of the feature required. If the feature occurs in a planning unit, regardless of its amount, that is considered one occurrence. This can be used in conjunction with or instead of 'target'. |
| propocc | optional | The percentage of occurrences of the feature required. This can be used in conjunction with or instead of 'prop'. |
| fpf | optional | The penalty factor for that feature. |
| name | optional | Indicates the name of the feature. Do not include any spaces or non-alphanumeric characters in the name. |

*fpf*
The feature penalty factor is a multiplicative factor which can be unique to each feature. It is a measure of how important it is to meet the target for the feature. If it is below 1, then the algorithm is likely to refuse to add a planning unit to include that feature if there are no other features on the planning unit. If the solution falls short of meeting a target for a feature, the fpf should be increased to a value greater than 1.

*Example – portion of feat.dat file*

```
id, prop, fpf, name
1, 0.3, 1, RockyReef
2, 0.9, 1, KelpForest
3, 0.5, 1, FishingGrounds
```

**Planning Unit Versus Feature**
Default name: "puvfeat.dat"

Description: This file is required to run Marxan Z. It contains the information on the distribution of features across the planning units. The range of features that can be accepted include ecological, economic and social information. The file must have three columns in the following order - featureid, puid, amount - sorted by the planning unit id for Marxan Z to execute properly.

| Header | Required/Optional | Description |
|---|---|---|
| featureid | required | Feature idendifier – must be indicated in the feature file. |
| puid | required | Planning unit identifier – must be indicated in the planning unit file. |
| amount | required | Amount of feature in the planning unit. The measurement unit between features can be different. However, the units within a feature |

| | | |
|---|---|---|
| | | must be consistent. |

*Example – portion of puvfeat.dat file*

```
featureid, puid, amount
1, 1, 350
1, 2, 50
1, 3, 100
2, 3, 160
3, 1, 300
3, 2, 200
```

**Zones**
Default name: "zones.dat"

Description: This file is required to run Marxan Z. It contains a list of the names and numeric identifiers of all possible zones. The file must have two fields in the following order – zoneid, zonename - sorted by lowest to highest zoneid.

| Header | Required/Optional | Description |
|---|---|---|
| zoneid | required | The numeric identifier number for the zone. The zoneid must be a positive integer and the file must be sorted by lowest to highest zoneid. |
| zonename | required | Indicates the name of the zone. Do not include any spaces or non-alphanumeric characters in the name. |

*Example – portion of zones.dat file*

```
zoneid, zonename
1, Available
2, MarineReserve
3, MarinePark
4, FishingZone
```

**Costs**
Default name: "costs.dat"

Description: The purpose of this file is to assign each cost name in the planning unit file with a unique numeric identifier. If this file is not included in Marxan Z, the cost values indicated in the planning unit file will be invalid and a default cost of 1 for all planning units will be used. This file contains a list of the names and identifiers for all possible costs. The cost values are included in the planning unit file. The file must have two fields in the following order – costid, costname - sorted by lowest to highest costid.

| Header | Required/Optional | Description |
|---|---|---|

| | | |
|---|---|---|
| costid | required | The numeric identifier for the cost. The costid must be a positive integer and the file must be sorted by lowest to highest costid. |
| costname | required | Indicates the name of the cost. Do not include any spaces or non-alphanumeric characters in the name. |

*Example – portion of costs.dat file*

```
costid, costname
1, area
2, salmonfishing
3, squidfishing
```

**Zone Cost**

Default name: "zonecost.dat"

Description: This file is required for Marxan Z to run. It includes a weighting factor for each cost in each zone. If some combinations of zone and cost are not present, that cost will be given a weighting of 0 and will not be considered to formulate a solution. The file must have three fields in the following order – zoneid, costid, multiplier - sorted lowest to highest by zoneid, then by costid.

| Header | Required/Optional | Description |
|---|---|---|
| zoneid | required | The zone identifier - must be compatible with the zones.dat file. |
| costid | required | The cost identifier - must be compatible with the costs.dat file. |
| multiplier | required | This number can be a fraction or an integer. In a given zone, it will be multiplied by the specified cost. All costs in a given zone will be multiplied by the specified multiplier and then added to give a total cost for each planning unit. For example, if there are 3 costs in one zone, the total cost for that zone would be calculated using the following equation:<br><br>**Total C = (C1 * M1) + (C2 * M2) + (C3*M3)**<br><br>Where C = cost, M = Multiplier |

*Example – portion of zonecost.dat file*

```
zoneid, costid, multiplier
2, 1, 0.2
2, 2, 0.4
```

| |
|---|
| 2, 3, 0.4 |

**Boundary Length**
Default: "bound.dat"

Description: This is an optional file in Marxan Z. It is required if the boundary length modifier (BLM) is greater than 0 in the input.dat file. This file contains information on the boundary "costs" of two planning units. If boundaries are duplicated in this file, they will be summed together. When a planning unit is on the edge of a study region, it is identified as a boundary of a planning unit with itself (i.e. the planning unit identifier is the same for id1 and id2). The file must have three fields in the following order – id1, id2, boundary.

| Header | Required/Optional | Description |
|---|---|---|
| id1 | required | Planning unit identifier |
| id2 | required | Planning unit identifier of the neighbouring planning unit to id1 or the same as id1 for an irremovable boundary. |
| boundary | required | Should be greater than or equal to zero. Can be the actual length of the boundary or it can be modified to reflect the 'cost' of separating two planning units. |

*Example – portion of bound.dat file*

| |
|---|
| id1, id2, boundary<br>1, 1, 1<br>1, 2, 1<br>1, 3, 1<br>2, 3, 1 |

**Zone Boundary Cost**
Default: "zoneboundcost.dat"

Description: This is an optional file in Marxan Z. This file is used if you want boundaries between different zones to have different costs (e.g. If you want two particular zones to be adjacent to one another, you could increase the boundary cost between those zones). If this file is not present, all boundaries between zones will be given a cost of 1. Likewise, if a zone-zone boundary cost is not indicated in this file, it will be given a cost of 1. This file is similar to the bound.dat file but relates the boundary "cost" of zones rather than planning units. The file must have three fields in the following order – zoneid1, zoneid2, cost - sorted lowest to highest by zoneid1, then by zoneid2.

| Header | Required/Optional | Description |
|---|---|---|
| zoneid1 | required | Zone identifier - must be indicated in the |

| | | |
|---|---|---|
| | | zones.dat file. |
| zoneid2 | required | Zone identifier, different from zoneid1 - must be indicated in the zones.dat file. |
| cost | required | The cost between zoneid1 and zoneid2. If cost is not indicated, a cost of 1 will be assigned. |

*Example – portion of zoneboundcost.dat file*

```
zoneid1,zoneid2,cost
1,1,0
2,2,0
3,3,0
1,2,0.1
1,3,0.1
2,3,1
```

**Planning Unit Zone**
Default: "puzone.dat"

Description: This is an optional file in Marxan Z. This file is used if you want to restrict certain planning units to two or more of your zones. This file must have two fields in the following order – puid, zoneid - sorted lowest to highest by puid, then by zoneid.

Do not use this file to lock planning units to a single zone; instead you should use the "Planning Unit Lock" file for this purpose. Do not place entries here for planning unit identifiers that are not in the "Planning Unit File".

| Header | Required/Optional | Description |
|---|---|---|
| puid | required | Planning unit identifier that is restricted to a specific zone. The same planning unit can be listed more than once to indicate restriction to more than one zone. |
| zoneid | required | Zone identifier that the planning unit in puid is restricted to. |

*Example – portion of puzone.dat file*

```
puid, zoneid
18, 1
19, 2
19, 3
20, 1
```

**Planning Unit Lock**
Default: "pulock.dat"

Description: This is an optional file in Marxan Z. This file is used if you want to restrict certain planning units to a single zone. This file must have two fields in the following order – puid, zoneid - sorted lowest to highest by puid, then by zoneid. Do not place entries here for planning unit identifiers that are not in the "Planning Unit File".

| Header | Required/Optional | Description |
|---|---|---|
| puid | required | Planning unit identifier that is restricted to a specific zone. The same planning unit can be listed more than once to indicate restriction to more than one zone. |
| zoneid | required | Zone identifier that the planning unit in puid is restricted to. |

*Example – portion of pulock.dat file*

```
puid, zoneid
57,2
71,2
72,2
86,2
```

**Zone Target and Zone Target 2**
Default name: "zonetarget.dat" or "zonetarget2.dat"

Description: These are optional files in Marxan Z. Zone target and zone target 2 are not required to run Marxan Z. Zone target and zone target 2 are used to allow zone based targets to be set in Marxan Z. These files can be used at the same time as overall targets (target or prop) set in the feature file, or they can be used instead of overall targets.

Zone target: The zone target file allows the user to set a target for each feature in each zone (i.e. zone-specific target). It is not necessary to indicate a zone-specific target for features. The zone target file and zone contribution file are *not related* to each other. The available zone can also have zone-specific target; a zone-specific target in this zone will ensure that a target amount of nonimated features is not protected. The files must have the correct fields in the correct order – specified in the tables below.

*Example of how zonetarget.dat works*

In zonetarget.dat, feature Y has a target of 25 square km in zone 2. In feat.dat, feature Y has an overall target of 50 square km. In the study region, 500 square km of feature Y occurs. This means that we want 10% of feature Y to be protected overall, with 5% of the feature in zone 2, and 5% of the feature in any other protected zone.

Fields required for zonetarget.dat

| Header | Required/Optional | Description |
|---|---|---|
| zoneid | required | The zone identifier |
| featureid | required | The feature identifier |
| target | required | The target amount (with the same unit used in the puvfeat.dat file), percentage, occurrence or occurrence percentage of the feature to include in the specified zone (i.e. zone-specific target). Negative targets are not allowed. |
| targettype | optional | Target type indicates the type of target specified in the target column (i.e. amount, percentage, occurrence, occurrence percentage). If this column is not included, Marxan Z will use a default of 0 (see targettype definitions below) |

Fields required for zonetarget2.dat

| Header | Required/Optional | Description |
|---|---|---|
| zoneid | required | The zone identifier |
| target | required | The target amount (in unit of puvfeat.dat file), percentage, occurrence or occurrence percentage of the feature to include in the specified zone (i.e. zone-specific target). Negative targets are not allowed. |
| targettype | optional | Target type indicates the type of target specified in the target column (i.e. amount, percentage, occurrence, occurrence percentage). If this column is not included, Marxan Z will use a default of 0 (see targettype definitions below) |

*Target Type Values*

0 = Amount (in unit of puvspr2.dat file) target of feature. Similar to target in the feat.dat file.

1 = Percentage target as proportion of total amount of feature. Similar to prop in the feat.dat file.

2 = Occurrence target. If the feature occurs in a planning unit, regardless of its amount, it is considered one occurrence. Similar to targetocc in the feat.dat file.

3 = Percentage target as proportion of total occurrences of feature. Similar to propocc in feat.dat.

Zone target 2:
Zone target 2 is identical to zone target *except* it does not include the featureid field, meaning that the target specified will apply to all features within the specified zone. If you use both files - zone target and zone target 2 - zone target 2 will override zone target.

*Example – portion of zonetarget.dat file*

| |
|---|
| zoneid, featureid, target<br>1, 3, 250<br>2, 1, 50 |

**Zone Contribution and Zone Contribution 2**

Default name: "zonecontrib.dat" or "zonecontrib2.dat"

Description: These are optional files in Marxan Z. Zone contribution and zone contribution 2 are not required to run Marxan Z, although one or other of these files *must be used* if an overall target is specified in the feature file. They *work in tandem* with the overall target specified in the feature file to specify differential contribution rates to the overall target. The zone contribution files are *not related* to the zone target files. The files must have the correct fields in the correct order – specified in the tables below.

Zone contribution: The zone target file allows the user to specify a contribution rate for each feature in each zone to the overall target specified in the feature file. The two files are work in tandem with each other.

The "zonecontrib.dat" file allows users to specify a contribution fraction for each feature in each zone.

Fields required for zonecontrib.dat

| Header | Required/Optional | Description |
|---|---|---|
| Zoneid | required | The zone identifier |
| Featureid | required | The feature identifier |
| Fraction | required | The contribution fraction for this feature in this zone as applied to the overall target specified in the feature file. Negative contributions are not allowed. |

The "zonecontrib2.dat" file allows users to specify a contribution fraction for each zone. This file omits the featureid field as all features in a zone have the same contribution fraction.

Fields required for zonecontrib2.dat

| Header | Required/Optional | Description |
|---|---|---|
| zoneid | required | The zone identifier |
| fraction | required | The contribution fraction for all features in this zone as applied to the overall target specified in the feature file. Negative contributions are not allowed. |

*Example of how zonecontrib.dat works*

In zonecontrib.dat, feature Y has a contribution of 1 in zone 2. In feat.dat, feature Y has an overall target of 50 square km. A planning unit contains 25 square km of feature Y, and is placed in zone 2. It will have a contribution of 25 square km to the target for feature Y (25 * 1), and we will require an additional 25 square km of feature Y to meet target.
If feature Y has a contribution of 0.2 in zone 2, then the contribution of placing the same planning unit in zone 2 is (25 * 0.2 = 5) 5 square km, and we will require an additional 45 square km of feature Y to meet target.

Example – portion of zonecontrib2.dat

*zoneid, fraction*
1, 0
2, 0.2
3, 1

## Input Parameter File

This file is required to run Marxan Z. It contains all of the main parameter definitions that control the way in which Marxan Z works. A convenient way to build and edit this file is using the Inedit.exe program. However, it can be built and edited using a text editor.

There are six sections in this file: General Parameters, Annealing Parameters, Cost Threshold, Input, Save, and Program Control. The variables, their default values, and a brief description of the variables are listed in this section. A more detailed description of the more complex variables is described at the end of each section and sometimes in section 2.0 of this manual. In the input parameter file, any line which does not start with one of the valid variable names is ignored and hence can be used as a comment line.

Note that all of the variable names are in upper case characters with no spaces. To set the value of a variable, start a line with the variable name and follow it with the value to set that variable to. The variables can occur in any order in the file, but an error will result if any variable is defined twice. Most of the variables have default values which will be used if they are not defined. An example input file is provided in Appendix A.

**General Parameters**

| Variable | Default Value | Brief Description |
|---|---|---|
| BLM | 0 | Boundary length modifier |
| PROP | 0.5 | Proportion of planning units to start in run. It must be a number between 0 and 1. When using simulated annealing with iterative improvement (recommended), changing the value has no measurable impact on the final solution. |
| RANDSEED | -1 | Random seed number must be an integer. If the value is negative, the program will randomly select a seed. A positive value would only be used for debugging purposes or if you want more than one application of the program to be identical. |
| NUMREPS | 100 | Number of separate runs with same starting condition. |
| AVAILABLE ZONE | 1 | The available zone is treated as an unprotected zone in Marxan Z. This is where you indicate which zone number from the zones input file is your available zone. |

*Boundary Length Modifier (BLM)*

The BLM is a variable that controls the importance of minimizing the boundary length relative to the planning unit cost. In other words, the BLM is a parameter that directs the assignment of planning units to zones in a cluster formation rather than selecting several disconnected planning units. With a small BLM, the algorithm concentrates on minimizing planning unit cost while a larger BLM places greater emphasis on minimizing the boundary length. A low BLM will allow the program to select several smaller areas, whereas a larger BLM will force fewer, larger areas be selected. If the BLM is set to zero, then the boundary length will have no impact on the solution. There is not a universally good value for BLM, as it is subject to the cost measures and geometry of the study region/planning units for any given planning project. The user must explore the effects of different BLM values to determine an efficient BLM. A method for determining an efficient level of spatial compactness is described by Stewart and Possingham (Stewart and Possingham 2005). This method considers the effect of increasing the BLM and minimizing boundary length of a Marxan solution.

**Annealing Parameters**

| Variable | Default Value | Brief Description |
|---|---|---|
| NUMITNS | 1 million | Number of iterations for annealing for each run (i.e. number |

| Variable | Default Value | Brief Description |
|---|---|---|
| | | of times Marxan Z tries to generate a solution for each run). |
| STARTTEMP | 1 | Starting temperature for annealing. A value of -1 indicates that adaptive annealing will be used and the program will automatically select a starting temperature. This is recommended for use in Marxan Z. |
| COOLFAC | 0 | This determines how quickly the system cools. If the STARTTEMP is set to -1, this variable is not necessary to run Marxan Z. This is recommended for use in Marxan Z. |
| NUMTEMP | 10,000 | Number of temperature decreases for annealing. This must be less than or equal to the number of iterations. A value of 10,000 is ideal and is recommended in Marxan Z. Setting it lower can make the regime too coarse; setting it too high will lead to round-off error problems with temperature. |

*Numer of Iterations*
The more iterations set, the longer the program will run, and the more likely Marxan Z will generate a better solution (i.e. lower objective function value). The ideal number of iterations will vary between projects. If the iterations are varied and a plot of iterations vs. objective function value is generated, it is possible to determine the point in which increasing the number of iterations stops producing results with a significantly improved objective function value.

**Cost Threshold**
If the cost threshold features are not used, this section does not need to be included in the Marxan Z input files. A detailed description of the cost threshold is provided in section 2.0 of this manual.

| Variable | Default Value | Brief Description |
|---|---|---|
| COSTTHRESH | 0 | The cost threshold allows the user to cap the zoning configuration to a set cost. Setting a cost threshold to zero disables this feature. |
| THRESHPEN1 | 0 | Threshold penalty parameter 1 |
| THRESHPEN2 | 0 | Threshold penalty parameter 2 |

**Input**
All possible input files and their default name are listed in this section. However, your input section only needs to include the required Marxan Z files plus the optional files that you choose to utilize, as described in this manual.

| Variable | Default Name | Brief Description |
|---|---|---|
| INPUTDIR | User defined | Input directory where data files are stored on the users computer |
| PUNAME | pu.dat | Name of planning unit file |
| FEATNAME | feat.dat | Name of feature file |
| PUVFEATNAME | puvfeat.dat | Name of planning unit versus feature file |
| ZONESNAME | zones.dat | Name of zones file |
| COSTSNAME | costs.dat | Name of cost file |
| ZONECOSTNAME | zonecost.dat | Name of zone cost file |
| BOUNDNAME | bound.dat | Name of boundary length file |
| ZONEBOUNDCOSTNAME | zoneboundcost.dat | Name of zone boundary cost file |
| PUZONENAME | puzone.dat | Name of planning unit zone file |
| PULOCKNAME | pulock.dat | Name of planning unit lock file |
| ZONETARGETNAME | zonetarget.dat | Name of zone target file |
| ZONETARGET2NAME | zonetarget2.dat | Name of zone target 2 file |
| ZONECONTRIBNAME | zonecontrib.dat | Name of zone contribution file |
| ZONECONTRIB2NAME | zonecontrib2.dat | Name of zone contribution 2 file |

**Save**

To generate any of the listed output files (i.e. save files) in Marxan Z, a value of 2 or 3 (a value of 2 is the ArcView format, 3 is the comma delimited ASCII format) must be specified next to the file name. If a value of 0 is next to the file name, that output file will not be generated. A description of each of these files is given in the output files section.

| Variable | Default Value | Brief Description |
|---|---|---|
| SCENNAME | User defined | Name of scenario for save files |
| SAVERUN | 3 | Save each run? |
| SAVEBEST | 3 | Save best run? |
| SAVESUMMARY | 3 | Save summary information? |
| SAVESCEN | 3 | Save scenario information? |
| SAVETARGMET | 3 | Save targets met information? |
| SAVESUMSOLN | 3 | Save summary solution information? |
| SAVESOLUTIONSMATRIX | 3 | Save all runs in a single matrix? |
| SOLUTIONSMATRIXHEADERS | 1 | Include header rows in solutions matrix. |
| SAVEPENALTY | 3 | Save computed feature penalties? |
| SAVELOG | 3 | Save log file? |
| SAVEANNEALINGTRACE | 3 | Report detail for simulated annealing. |
| ANNEALINGTRACEROWS | 1000 | Number of iterations to report detail for. |
| SAVEITIMPTRACE | 3 | Report detail for iterative improvement. |
| ITIMPTRACEROWS | 1000 | Number of iterations to report detail for. |
| SAVEZONECONNECTIVITYSUM | 3 | Save zone connectivity sum for runs. |
| OUTPUTDIR | User defined | Output directory to store output files on |

| | | |
|---|---|---|
| | | the users computer |
| | | |

**Program Control**

There are two basic algorithms that can be used to formulate a solution in Marxan Z: simulated annealing and iterative improvement. A detailed description of these algorithms is provided in section 2.0 of this manual. These algorithms can be used alone or in combination with each other. The run mode selected in the input parameter file determines which algorithm, or combination of algorithms, is used to formulate a solution in Marxan Z.

| Variable | Default Value | Brief Description |
|---|---|---|
| RUNMODE | 1 | Using the default value of 1 will apply annealing followed by the iterative improvement algorithm, giving the best possible results in Marxan Z. |
| MISSLEVEL | 1 | This is the proportion of the target which a feature must reach in order for it not to be counted as missing. A value of 1 means that 100% of the target for a feature must be included in the solution, or it will be considered an unmet goal (i.e. missing). |
| ITIMPTYPE | 0 | 0 is recommended in Marxan Z |
| VERBOSITY | 2 | This value indicates how much information to print on the Marxan Z screen when running. A value of 0 will display the lowest level of information and a level of 3 will show the highest amount of information. A more detailed description of each value is in the Screen Output section. |

*Run Mode*

There are four possible run modes. Each mode determines which selection algorithm is used. Using the default value of 1 will apply annealing followed by the iterative improvement algorithm, giving the best possible results in Marxan Z.

| Value | Description |
|---|---|
| -1 | Use no methods |
| 1 | Apply annealing followed by the iterative improvement algorithm |
| 4 | Use only the iterative improvement. |
| 6 | Use only annealing |

*Iterative Improvement Type*
There are five possible iterative improvement types. Iterative improvement will only be used when a run mode using the iterative improvement algorithm is selected. Using the default value of 0 is recommended in Marxan Z.

| Value | Description |
|---|---|
| -1 | Do not use iterative improvement |
| 0 | Normal Improvement |
| 1 | Two Step Iterative Improvement |
| 2 | Swap Iterative Improvement |
| 3 | Normal Followed by Two Step |

## 1.2 OUTPUT FILES

In the input parameter file, the user can indicate which output files Marxan Z generates. There are seven types of output files that Marxan Z can generate. The files each take on the scenario name defined in the input parameter file, followed by an abbreviation for each file type. This section will describe each of the output files.

**Solutions for Each Run**
Name in input.dat: SAVERUN

Name of output file: scenario_r001.dat, scenario_r002.dat, etc., where 'scenario' is the name supplied by the user in the SCENNAME parameter of the input.dat parameter file.

Description: A file is produced for each run of the algorithm describing which planning units were selected in each zone. It contains the planning unit identifier, followed by the zone identifier, separated by a comma. Each planning unit identifier is on a new line.

**Best Solution for All Runs**
Name in input.dat: SAVEBEST

Name of output file: scenario_best, where 'scenario' is the name supplied by the user in the input parameter file

Description: A file is produced for the best solution from a set of runs in Marxan Z. It contains the planning unit identifier, followed by the zone identifier, separated by a comma. Each planning unit identifier is on a new line.

**Missing Values for Each Run**
Name in input.dat: SAVETARGMET

Name of output file: scenario_mv001.dat, where 'scenario' is the name supplied by the user in the input parameter file

Description: This file contains information on how well the final solution from each run did with regard to meeting the feature's target. This file will only be generated for individual run solution files which are also generated. It must be selected with either the solutions for each run option or the best solution option. The file contains the following column headers: feature, feature name, target, amount held, occurrence target, occurrences held, target met. Each column is described below. For each zone specified in zones.dat (except for the available zone), there are separate target, amount held, and occurrences held columns associated with the zone.

| Header | Description |
|---|---|
| Feature | The identifier of the feature, indicated in feat.dat |
| Feature Name | The optional name for the feature, indicated in feat.dat |
| Target | The target amount for the feature, indicated in feat.dat |
| Total Amount | Total amount of the feature in the study region |
| Contributing Amount Held | Amount of the feature captured multiplied by the features contribution fraction for the zone it is captured in |
| Occurrence Target | The target number of occurrences for the feature |
| Occurrences Held | Number of occurrences of the feature captured |
| Target Met | 'yes' if both targets above are met, otherwise 'no' |
| Target Zone1 | The target amount for the feature in zone 1, indicated in zonetarget.dat |
| Amount Held Zone1 | Amount of the feature captured in zone 1 |
| Contributing Amount Held Zone1 | Amount of the feature captured in zone 1 multiplied by the features contribution fraction in zone 1 |
| Occurrence Target Zone1 | The target number of occurrences for the feature in zone 1, indicated in zonetarget.dat |
| Occurrences Held Zone1 | The number of occurrences of the feature captured in zone 1 |
| Target Met Zone1 | 'yes' if both targets above are met, otherwise 'no' |

The 6 fields specified for zone 1 are repeated for each zone, with "Zone1" in the field name substituted by the name of the zone.

**Summary Information**

Name in input.dat: SAVESUM

Name of output file: scenario_sum.dat, where 'scenario' is the name supplied by the user in the input parameter file

Description: This file contains summary information on each run and contains the following column headers to explain which is in each line: run no, score, cost (for each cost indicated in the pu.dat file), planning units (per zone), boundary length (per zone), penalty, shortfall, missing values. Each header is explained below.

| Header | Description |
|---|---|
| Run no | The run number for that line. |
| Score | The objective function score for the solution. |
| Cost | Cost value for the solution for each cost in the pu.dat file. |
| Planning Unit Count (for each zone) | Number of planning units in the solution for each zone. |

| | |
|---|---|
| Planning Unit Cost (for each zone) | The total cost of planning units for each zone. |
| Boundary Length | The boundary length of the solution. |
| Penalty | The penalty score for missing features for the solution. |
| Shortfall | Shortfall is equal to the sum - across all features that have not met their target - of the target minus amount in solution. If a feature target is met, there is no shortfall. |
| Missing Values | The number of features that have not met their target. |

When there are some features which haven't met their target, the last three headers are useful in determining how poor the actually is. It is possible to have a high penalty and still be very close to the targets, particularly if the feature penalty factors have been set very high. The shortfall is a good indication of whether the features are very close or very far from their targets (the number of missing values gives further information in this vein). If there are five features which each have missed their targets, but the combined shortfall is very small, then they could all be only narrowly missing their targets (eg 99% or higher) and the user might not be particularly concerned.

**Scenario Details**
Name in input.dat: SAVESCEN

Name of output file: scenario_sen.dat, where 'scenario' is the name supplied by the user in the input parameter file

Description: This file contains a documented list of the options that make up the scenario created in the input parameter file. It is used to keep track of scenario details when multiple scenarios are run.

**Summed Solution**
Name in input.dat: SAVESUMSOLN

Name of output file: scenario_ssoln.dat, where 'scenario' is the name supplied by the user in the input parameter file

Description: This file is the summed solution of all individual runs in a scenario. It indicates how often a planning unit was included in a solution, across all individual runs. In addition, it shows how often a planning unit was included in an individual zone (except for the available zone). It is a useful way to explore the irreplaceability of planning units in a zone.

**Solutions Matrix**
Name in input.dat: SAVESOLUTIONSMATRIX

Name of output file: scenario_solutionsmatrix.dat, where 'scenario' is the name supplied by the user in the input parameter file

Description: This file record which planning units were selected in each zone for all runs. If the SOLUTIONSMATRIXHEADERS parameter, has a value of 1, a header row is included in the file. If you choose to include the header row, a list of the planning unit identifiers is contained in the row. There is one row for each run that contains the zone identifier for each planning unit, indicating which zone each planning unit was placed in for the run. It is a useful way to export information on planning unit selection for cluster analysis in R or other statistical software packages.

**Penalty**
Name in input.dat: SAVEPENALTY

Name of output file: scenario_penalty.dat, where 'scenario' is the name supplied by the user in the input parameter file

Description: This file contains the penalty computed by Marxan for all features. individual runs in a scenario. It indicates how expensive it was to satisfy the objectives for a feature. It is a useful to the relative difficulty Marxan has meeting the objectives for features.

**Screen Log File**
Name in input.dat: SAVELOG

Name of output file: scenario_penalty.dat, where 'scenario' is the name supplied by the user in the input parameter file

Description: This file displays all of the screen output shown when running Marxan Z.

**Annealing Detail**
Name in input.dat: ANNEALINGTRACEROWS

Name of output file: scenario_penalty.dat, where 'scenario' is the name supplied by the user in the input parameter file

Description: This file displays all of the screen output shown when running Marxan Z.

**Iterative Improvement Detail**
Name in input.dat: ITIMPTRACEROWS

Name of output file: scenario_penalty.dat, where 'scenario' is the name supplied by the user in the input parameter file

Description: This file displays all of the screen output shown when running Marxan Z.

**Zone Connectivity Sum**
Name in input.dat: SAVEZONECONNECTIVITYSUM

Name of output file: scenario_penalty.dat, where 'scenario' is the name supplied by the user in the input parameter file

Description: This file displays all of the screen output shown when running Marxan Z.

## 1.3 SCREEN OUTPUT

In the input parameter file, the level of information displayed on the Marxan Z screen can be determined using the VERBOSITY variable in the program controls section. If a verbosity level of zero is set, no information about the scenario will be displayed. A level of three is the highest possible level and will display the most detailed information about the scenario. Any level greater than zero, will display basic summary information for each run. Level 2 will include additional information on the successful loading of files, how many lines were successfully read, the score of the zone configuration at different points in a run, and more time measures. When using adaptive annealing with this level of information, the calculated initial temperature and decrement are also displayed. Level 3 is typically more detail than necessary. Level 3 provides information about the current zone configuration score each time the temperature is decremented when using simulated annealing.

An example of one basic summary information line (included in all levels greater than zero) is:

Run 1 Value 10 Cost 12 PU 100 Zonename2 50 Zonename3 25 Zonename4 25 Boundary 300 Missing 2 Penaly 500

*Run* 1 indicates that this is the first run of the algorithm. The run number does not always appear on the same line as the valuation of the solution. It depends upon the information detail level.

*Value* 10 is the cost plus the boundary value plus the penalty for missing features. Because the penalty is included in this value, the meaning of the value requires interpretation.

*Cost* 12 is the cost for the solution. It is the sum of the costs of all planning units in the solution in the units of the cost field in the pu.dat file.

*PUs* 100 is the total number of planning units in the solution.

*Zonename* x is the number of planning units in that zone. All protected zones will be listed in the summary line as the name of the zone indicated in the zones.dat file. The planning units in the protected zones will sum to equal the total number of planning units in the solution.

*Boundary* 300 is the perimeter, or boundary length, of the solution.

*Missing* 2 is the number of objectives which are under-represented in the solution. These objectives are the targets defined in feat.dat and zonetarget.dat. They are screened according to the 'MISSLEVEL' parameter in the input.dat parameter file.

*Penalty* 500 is the penalty for not representing all of the features. If they are all represented, then it will be 0.0.

# 2.0 Objective Function and Algorithms used

## 2.1 THE OBJECTIVE FUNCTION

The objective function gives a value for each zoning configuration of planning units. This means that a single planning unit or no planning units at all can be given an objective function value. A configuration containing zero planning units would be cheap to implement, but would probably not meet the planning objectives and so the objective function value should be very poor.

If we have an objective function which gives any possible zoning configuration a value, or score, then we can compare any two zone configurations and say which one is better than the other (according to the objective function). Because the objective function value can be evaluated by a computer, the door is open to using a wide range of methods to automatically create zone configurations that have good objective function values.

In Marxan Z, this objective function is used by the iterative improvement algorithm and by simulated annealing. The objective function was designed with the aim to integrate it with a simulated annealing optimiser but the two are distinct entities. Simulated annealing is a general purpose optimiser and the objective function defines the constraints and objectives for a zone configuration without explicitly defining how an optimal zonation will be found.

The objective function consists of two main sections; the first is a measure of the 'cost' of the zone configuration and the second a penalty for breaching various criteria. In its simplest form, it is a combination of the economic cost of the zone configuration and a penalty for not meeting all of the objectives, if any are unmet. These criteria can include a cap on the 'cost' of the zone configuration and always includes the target representation level for each feature. As well as this is the optional measure of the fragmentation of the zone configuration and an optional cost threshold penalty. In this objective function, the lower the value the better the zone configuration:

$$\sum_{PUs} Cost + BLM \sum_{PUs} Boundary + \sum_{ConValue} FPF \times Penalty + CostThresholdPenalty(t)$$

Cost is the sum of each cost measure multiplied by the value indicated in zonecost.dat file of each of the PU within the zone configuration.

'Boundary' is the length (or possibly cost) of the boundary surrounding the zone configuration. The constant BLM is the boundary length multiplier which determines the importance given to the boundary length relative to the cost of the zone configuration. If

a value of 0 is given to BLM then the boundary length is not included in the objective function.

The next term is a penalty given for not adequately representing a feature, summed over all features for each zone. FPF stands for 'feature penalty factor' and is a weighting factor for the feature which determines the relative importance for satisfying that particular feature's target. The penalty term is a penalty associated with each underrepresented feature. It is expressed in terms of cost and boundary length and is roughly the cost and additional modified boundary needed to adequately capture a feature which is not adequately represented in the current zone configuration.

The cost threshold penalty is a penalty applied to the objective function if the target cost is exceeded. It is a function of the cost and possibly the boundary of the system and in some algorithms will change as the algorithm proceeds (which is the *t* in the above formula). This penalty is also optional and can be excluded from the objective function.

**Cost of the Zone Configuration**

The objective function that Marxan Z uses has the constraint that the cost of a zone configuration is the linear combination of costs of all the planning units within the zone configuration, broken down by zone.

**Boundary Length and Fragmentation**

By including a boundary length term in the objective function we can apply a control on the level of fragmentation in the zone configuration. In order to allow the boundary length to be added to the cost measure, a multiplicative factor is used. This is because the boundary length is most probably going to be in units which are different from the cost measure. Not only are the units incompatible without a boundary length modifier, but the importance of compactness over cost is not immediately obvious. Changing the boundary length modifier allows planners to explore this issue.

**Representation Requirements for Features**

In the objective function, if the feature does not meet one or more of its requirements then it will attract a penalty depending upon how far below representation it is and the relative value of the feature to the other features.

**Feature Penalty**

The feature penalty is the penalty given to a zone configuration for not achieving feature targets. It is based on a principle that if a feature is below its target representation level, then the penalty should be close to the cost for raising that feature up to its target representation level. For example: if the requirement was to represent each feature by at least one instance then the penalty for not having a given feature would be the cost of the least expensive planning unit which holds an instance of that feature. If you were missing a number of features, then you could produce a zone configuration that was fully representative by adding the least expensive planning units containing each of the

missing features. This would not increase the objective function value for the zone configuration, in fact, if any of the additional planning units had more than one of the missing features, then the objective function value would decrease. It would appear to be ideal to recalculate the penalties after each change had been made to the zone configuration. However, this would be time consuming and it turns out to be more efficient to work with penalties which change only in the simplest manner from one point in the algorithm to the next. A greedy algorithm is used to calculate the cheapest way in which each feature could be represented on its own and this forms the base penalty for that feature. Marxan Z adds together the cheapest planning units which would achieve the representation target. This approach is described in the following pseudo-code:

I. For each planning unit calculate a 'cost per area' value.
    A. Determine how much of the target for the given feature is contributed by this planning unit.
    B. Determine the economic cost of the planning unit
    C. Determine the boundary length of the planning unit
    D. The overall cost is economic cost + boundary length x BLM
    E. Cost-per-hectare is then the value for feature divided by the overall cost.

II. Select the planning unit with the lowest cost-per-hectare. Add its cost to the running cost total and the level of representation for the feature to the representation level total.
    A. If the level of representation is close to the target then it might be cheaper to pick a 'cheap' planning unit has the required amount of the feature regardless of its cost-per-area.

III. Continue adding up these totals until you have found a collection of planning units which adequately represent the given feature.

IV. The penalty for the feature is the total cost (including boundary length times BLM) of these planning units.

Thus, if one feature were completely unrepresented then the penalty would be the same as the cost of adding the simple set of planning units, assuming that they are isolated from each other for boundary length purposes. This value is quick to calculate but will tend to be higher than optimum. Sometimes, there are more efficient ways of representing a feature than that determined by a greedy algorithm, consider the following example.

Example 1: Feature A appears on a number of planning units, the best ones are:

| Planning Unit | Cost | Amount of A represented |
|---|---|---|
| 1 | 2 | 3 |
| 2 | 4 | 5 |
| 3 | 5 | 5 |

| | | |
|---|---|---|
| 4 | 8 | 6 |

The target for A is 10 units. If we use the greedy algorithm we would represent this with planning units 1, 2, and 3 (selected in that order) for a total cost of 11.0 units. If we chose only planning units 2 and 3, we would still adequately represent A, but our cost would be 9 units.

This example shows a simple case where the greedy algorithm does not produce the best results. The greedy algorithm is rapid and produces reasonable results. The program will tend to overestimate and never underestimate the penalties when using a greedy algorithm. It is undesirable to have a penalty value which is too low because then the objective function might not improve by fully representing all features. If there are some features which need not be fully represented, this should be handled entirely by use of the feature penalty factor, which is described below. It is not problematic to have penalties which are higher then they absolutely need to be, sometimes it is desirable. The boundary cost for a planning unit in the above pseudo-code is the sum of all of its boundaries. This assumes that the planning unit has no common boundaries with the rest of the zone configuration and hence will again tend to overestimate the cost of the planning unit and the penalty.

The penalty is calculated and fixed in the initialization stage of the algorithm. It is applied in a straight forward manner - if a feature has reached half of its target then it scores half of its penalty. The problem with this is that you might find yourself in a situation where you only need a small amount to meet a feature's target, but that there is no way of doing this which would decrease the objective value. If we take Example 1 once again, then the penalty for feature A is 11 units. If you already have planning units 1 and 4 in the zone configuration, then you have 9 units of the feature and the penalty is 11.0 x (10-9)/10 = 1.1 units. So the feature attracts a penalty of 1.1 units and needs only 1 more unit of abundance to meet its target. There is no planning unit with a cost that low - the addition of any of the remaining planning units would cost the zone configuration much more than the gain in penalty reduction.

This problem can be fixed by setting a higher FPF (Feature Penalty Factor) for all features. The FPF is a multiplicative factor for each feature, described below.

It is possible that the target for a feature is set higher than can possibly be met. In Australia where the JANIS (1997) requirements state that 15% of the pre-European area of each forest ecosystem type should be protected, we can easily have targets which are larger than the current area of a given forest ecosystem. Currently, when this is the case, the algorithm will scale up the penalty so that if, for example, it costs 100 units to protect all of a given ecosystem but that represents only half of the target, then the initial penalty will be 200 units. This means that if you get half-way to your target then the penalty for that feature will be half the maximum penalty, no matter how high the target or whether it is a feasible target.

**Feature Penalty Factor**

The feature penalty factor (FPF) is a multiplicative factor which can be unique to each feature. It is primarily based on the relative worth of that feature but it includes a measure of how important it is to get it fully represented. The actual effect that it will have varies between the methods which use the objective function. If it is below 1 then the algorithm might refuse to add a planning unit to protect that feature if there are no other features on the planning unit. An algorithm might fall slightly short in the representation of features, getting close to, but not at or above, the target value. To ensure that each feature meets the target it can sometimes be desirable to set the FPF at greater value than 1.

**Cost Threshold Penalty**

The cost threshold is an option which allows the user to cap the zone configuration to a set cost + modified boundary length. This has been included to make it possible to look at a reverse version of the problem. The reversal of the problem would be to find the zone configuration which has the best representation for all features constrained by a maximum cost. It works by applying a penalty if the given threshold value is exceeded. The penalty is the amount by which the threshold is exceeded multiplied by the cost threshold penalty. The penalty depends upon the stage of the annealing algorithm (how far into the annealing process the system is given as a proportion). The multiplier is:

$$\text{Cost Threshold Penalty} = \text{Amount over Threshold} \times (Ae^{bt} - A).$$

Here t is the time during the run and varies between 0 and 1. A and B are control parameters. B controls how steep the curve is (A high B will have the multiplier varying little until late in the run). A controls the final value. A high A will penalize any excess of the threshold greatly, a lower A might allow the threshold to be slightly exceeded. The multiplier starts at 0 when t is zero. Both A and B require some experimentation to set. The cost threshold penalty is built into the objective function and hence will apply to the annealing module along with the iterative improvement module.

## 2.2 ALGORITHMS METHODS

There are two basic algorithms that can be used to formulate a solution in Marxan Z: simulated annealing and iterative improvement. Each of these algorithms can be used alone or in combination with each other. The run mode selected in the input parameter file determines which algorithm, or combination of algorithms, is used to formulate a solution in Marxan Z.

**Simulated Annealing**

Simulated annealing is based on iterative improvement with stochastic acceptance of bad moves to help avoid getting stuck prematurely in a local minimum. The implementation used in Marxan Z will run for a set number of iterations. For each iteration, a planning unit is chosen at random and may or may not be already in the zone configuration. The change to the value of the zone configuration - which would occur if this planning unit were added or removed from the system - is evaluated just as it was with iterative improvement. This change is combined with a parameter called the temperature and then compared to a uniform random number. The planning unit might then be added or removed from the system depending on this comparison.

The temperature starts at a high value and decreases during the algorithm. When the temperature is high, at the start of the procedure, then both good and bad changes are accepted. As the temperature decreases the chance of accepting a bad change decreases until, finally, only good changes are accepted. For simplicity, the algorithm should terminate before it can only accept good changes and the iterative improvement should follow it, because at this point the simulated annealing algorithm behaves like an inefficient iterative improvement algorithm.

There are two types of simulated annealing in Marxan Z. One is fixed schedule annealing in which the annealing schedule (including the initial temperature and rate of temperature decrease) is fixed before the algorithm commences. The other is adaptive schedule annealing in which the algorithm samples the problem and sets the initial temperature and rate of temperature decrease based upon its sampling.

*Adaptive Annealing Schedule*

The adaptive annealing schedule commences by sampling the system a number of times (number of iterations/100). It then sets the target final temperature as the minimum positive (ie least bad) change which occurred during the sampling period. The maximum is set according to the formula:

$$Tinit = Min\ Change + 0.1 \times (Max\ change - Min\ change)$$

This is based upon the adaptive schedule in (Conolly 1990). Here, Tinit is the initial temperature, the changes (min and max) are the minimum and maximum bad changes which occurred. In our case a bad change is one which increases the value of the objective function (ie a positive value).

*Fixed Annealing Schedule*

With fixed schedule annealing the parameters which control the annealing schedule are fixed for each implementation of the algorithm. This is done typically by some trials of the algorithm with different parameters for a number of iterations which is shorter by an order of magnitude to the number to be used in the final run. The parameters will often benefit by being changed for longer runs but still based on the trials. The trials include looking at final results and also tracking the progress of individual runs. The annealing schedules which arise from the fixed schedule process are generally superior to the adaptive annealing schedule. Adaptive annealing is advantageous as it does not require a skilled user to use the algorithm and because it is quicker. It is faster in terms of the processing time required as there is much less in the way of initial runs, it is considerably faster in terms of the time which the user must apply in running the algorithm. For this reason it is taken as a major algorithm to be examined here. Land-use designers and managers will tend to use the standard options and automatic methods to a large extent so that the ability of adaptive annealing to design zone configuration is very useful, although obviously not definitively important. Adaptive annealing is also important for broad investigations, tests and trials on the system which would precede the more careful and detailed use of a fixed schedule annealing algorithm.

*Setting a Fixed Annealing Schedule*

When setting a fixed schedule the two parameters to change are the initial and final temperature. The final temperature is set by choosing an appropriate value for the cooling function. If the final temperature is too low then the algorithm will spend a lot of time at a local minimum unable to improve the system and continuing to try. If the final temperature is too high then much of the important annealing work will not be completed and the zone configuration will largely be delivered by the iterative improvement schedule which follows simulated annealing and finds a nearby local minimum 'at random'. If the initial temperature is too high then the system will spend too much time at high temperatures around the high temperature equilibrium and less time where most of the annealing work is to be done.

Thus the best way to get the general feel for what the parameters should be is to run the algorithm and sample the value of the current system regularly to see when the equilibrium at various temperatures seems to be achieved, what they are and when the system no longer changes or improves. This makes it easy to set a provisional final temperature and also gives estimates at what reasonable initial temperatures might be. From here tests can be run looking at the final output from multiple runs and different parameters at a much shorter number of iterations.

Once good values have been found they need to be scaled up for the longer number of iterations. This is because the length of time spent at lower or critical temperatures is important and will drive the search for good parameters. Extending the length of the

algorithm will increase the time spent at these temperatures longer than is necessary. Thus the method used is to keep the final temperature the same and increase the level of the initial temperature so that it will spend a similar length of time at lower levels but allow it to search the global space to a greater extent. For a short run it is often best to have the system running at some critical temperature for as long as possible. For a longer run it is advantageous to increase the range of temperatures used.

**Iterative Improvement**

Iterative improvement is a simple optimization method. It has largely been supplanted by simulated annealing but can still be profitably used to aid the other algorithms. There are three basic types of iterative improvement which can be used in Marxan Z. They differ in the set of possible changes which are considered at each step. Each of them starts with a 'seed' solution. This can be any kind of zone configuration with some, all, or no planning units allocated to zones. It is useful to use the final result from another algorithm such as simulated annealing as the starting solution for iterative improvement. In this case the iterative improvement algorithm is used solely to ensure that no simple improvements are possible.

At each iteration, the algorithm will consider a random change to see if it will improve the value of the objective function if that change were made. If the change does improve the system then it is made, otherwise another, as yet untested, change is tested at random. This continues until every possible change has been considered and none will improve the system. The resulting zone configuration is a local minimum (or local optimum).

The three basic types of iterative improvement differ in the types of change that they will consider. The simplest type is called 'normal iterative improvement' and the only changes that are considered are adding or removing each planning unit from the zone configuration. This is the same 'move set' as is considered by the greedy algorithm and by simulated annealing.

The second type of iterative improvement is called 'swap' and it will randomly select planning units, if the selected planning unit can improve the system by being added or removed from it then this is done otherwise an exchange is considered. If the chosen planning unit is already in a zone configuration then the changes considered are removing that planning unit but adding another one somewhere else. If the chosen planning unit is not in the zone configuration then the changes considered are adding this to the system but removing one that is already in the system. Every possible 'swap' is considered in random order, stopping when one is found which will improve the system. Because the number of possible exchanges can be very large, this is a much slower option.

The third type is called 'two step', in this method as well as testing each planning unit (in random order) to see if adding or removing it would improve the system, each possible combination of two changes is considered. These changes include, adding or removing the chosen planning unit in conjunction with adding or removing every other planning

unit. The number of such moves is even greater than in the 'swap' method, so that this method should be used with care.

There is a fourth option which is to run the normal method first, to get a good local optimum and then run the 'two step' method afterward. Because the number of improvements that the 'two step' finds should be much smaller after a normal iterative improvement algorithm has passed over the 'seed' solution this should be much faster than running the 'two step' method on its own.

The main strength of iterative improvement is that the random element allows it to produce multiple solutions. On average the solutions might be poor, but if it can produce solutions quickly enough, then it may produce some very good ones over a great many runs. It is theoretically possible to reach the global minimum by running iterative improvement starting from either an empty zone configuration or a situation where every planning unit starts in a randomly selected zone.

The main use of iterative improvement will still be to follow a different algorithm for some fine-scale polishing up. This is particularly true for the 'swap' and 'two step' methods. Even following another algorithm these might take long to run.


# 3.0 ACKNOWLEDGEMENTS

The ongoing development and support of Marxan software would not be possible without the financial support of generous donor organisations from around the world.

We would like to thank Ecotrust, who provided substantial funding for the development of the software. In addition, the National Heritage Trust contributed substantial funds to development of the software and the research program underpinning the software. We also acknowledge ongoing funding from the Applied Environment Decision Analysis centre.

# 4.0 KEY REFERENCES

Peer reviewed literature and reports using Marxan are compiled at The University of Queensland's Ecology Centre. An updated list can be obtained from the Marxan website: http://www.uq.edu.au/marxan/index.html?page=80365&p=1.1.6.3

Airame, S. 2005. Channel Islands National Marine Sanctuary: Advancing the Science and Policy of Marine Protected Areas. Pages 91-124 *in* A. Scholz and D. Wright, editors. Place Matters: Geospatial Tools for Marine Science, Conservation, and Management in the Pacific Northwest. Oregon State University Press, Corvallis.

Ardon, J.A., J. Lash, D. Haggarty. 2002. Modelling a Network of Marine Protected Areas for the Central Coast of BC, version 3.1. Living Oceans Society, Sointula, BC, Canada.

Ball, I. R. (2000) Mathematical applications for conservation ecology: the dynamics of tree hollows and the design of nature reserves. PhD Thesis, The University of Adelaide.

Ball, I. R., and H. P. Possingham. 2000. MARXAN (V1.8.2): Marine Reserve Design Using Spatially Explicit Annealing, a Manual.

Beck, M. W. and M. Odaya (2001) Ecoregional planning in marine environments: Identifying priority sites for conservation in the northern Gulf of Mexico. Aquatic Conservation 11: 235-242.

Chan, A., A. Cundiff, N. Gardner, Y. Hrovat, L. Kircher, C. Klein. 2006. Marine Protected Areas Along California's Central Coast: A Multicriteria Analysis of Network Design. Thesis. University of California, Santa Barbara.

Cook, R. R. and P. J. Auster (2005) Use of simulated annealing for identifying essential fish habitat in a multispecies context. Conservaton Biology 19:3 876-886

Cook, R. R. (2006) Developing Alternatives for Optimal Representation of Seafloor Habitats and Associated Communities in Stellwagen Bank National Marine Sanctuary

Connolly, D. J. (1990) "an improved annealing scheme for QAP", European Journal of Operations Research, **46,** 93-100.

Ferdaña, Z. (2005) Nearshore marine conservation planning in the PacificNorthwest: Exploring the use of a siting algorithm for representing marine biodiversity, in Wright, D.J. and Scholz, A.J. (eds.), "Place Matters: Geospatial Tools, for Marine

Science, Conservation, and Management in the Pacific Northwest," Corvallis, OR: OSU Press

Game, E. T. and H. S. Grantham. (2008). Marxan User Manual: For Marxan version 1.8.10. University of Queensland, St. Lucia, Queensland, Australia, and Pacific Marine Analysis and Research Association, Vancouver, British Columbia, Canada.

Geselbracht, L., R. Torres, G.S. Cumming, D. Dorfman, M. Beck. 2005. Marine/Estuarine Site Assessment for Florida: A Framework for Site Prioritization. The Nature Conservancy. Accessed on May 9, 2006 at http://myfwc.com/wildlifelegacy/PDF/MarineSitePrioritizationFrameworkfinalrptPart1.pdf.

JANIS (1997) Nationally agreed criteria for the establishment of a comprehensive, adequate and representative reserve system for forests in Australia. A report by the Joint ANZECC/MCFFA National Forest Policy Statement Implementation Subcommittee. National forest conservation reserves: Commonwealth proposed criteria. Commonwealth of Australia, Canberra.

Kirkpatrick, S., Gelatt jr, C. D., and Vecchi, M. P. (1983) Optimization by Simulated Annealing. Science, **220,** 671-680.

Leslie, H.; M. Ruckelshaus, I. Ball, S. Andelman, H. Possingham. 2003. Using Siting Algorithms in the Design of Marine Reserve Networks. Ecological Applications 13(1) S185-S198.

Lewis, A., S. Slegers, D. Lowe, L. Muller, L. Fernandes, J. Day. 2003. Use of Spatial Analysis and GIS techniques to re-zone the Great Barrier Reef Marine Park. Coastal GIS Workshop.

McDonnell, M. D., H. P. Possingham, I. R. Ball and E. A. Cousins (2002) Mathematical methods for spatially cohesive reserve design. Environmental Modeling and Assessment 7: 107-114.

Possingham, H., Ball, I. R., Andelman, S. (2000) “Mathematical methods for identifyingrepresentative reserve networks” in Quantitative methods for conservation biology. Ferson, S., and Burgman, M. (eds). Springer-Verlag, New York.

Possingham, H. P., J. R. Day, M. Goldfinch and F. Salzborn (1993) The mathematics of designing a network of protected areas for conservation. In: D. J. Sutton, C. E. M. Pearce and E. A. Cousins (eds) Decision Sciences: Tools for Today. Proceedings of 12th National ASOR Conference. ASOR, Adelaide, pp. 536-545.

Pressey, R. L., Humphries, C. J., Margules, C. R., Vane-Wright, R. I., and Williams, P. H., (1993) "Beyond opportunism: Key principles for systematic reserve selection", TREE, **8,** 124-128

Pressey, R. L., Johnson, I. R., and Wilson, P. D., (1994) "Shades of irreplaceability: towards a measure of the contribution of sites to a reservation goal."

Pressey, R.L., and V.S. Logan. 1998. Size of selection units for future reserves and its influence on actual versus targeted representation of features: a case study in western New South Wales. Biological Conservation 85: 305-319.

Pressey, R. L., Possingham, H. P., Margules, C. R., (1996) "Optimality in Reserve Selection Algorithms: When Does it Matter and How Much?" Biological Conservation **76**, 259-267.

Richardson, E.A., M.J. Kaiser, G. Edwards-Jones, and H.P. Possingham. Sensitivity of marine-reserve design to the spatial resolution of socioeconomic data. Conservation Biology, in press. Royal Commission on Environmental Pollution. 2004. Turning the tide: Addressing the impact of fisheries on the marine environment. Page 497. RCEP, London.

Sala, E., O. Aburto-Oropeza, G. Paredes, I. Parra, J.C. Barrera, P.K. Dayton. 2002. A General Model for Designing Networks of Marine Reserves. Science 298: 1991-1993.

Stewart, R. R., T. Noyce and H. P. Possingham (2003) The opportunity cost of ad-hoc marine reserve design decisions - An example from South Australia. Marine Ecology Progress Series 253: 25-38.

Stewart, R. R. and H. P. Possingham (2003). A framework for systematic marine reserve design in South Australia: A case study. Proceedings of the Inaugural World Congress on Aquatic Protected Areas, Cairns - August 2002.

Stewart, R. R., and H. P. Possingham. 2005. Efficiency, costs and trade-offs in marine reserve system design. Environmental Modeling & Assessment **10**:203-213.

Stewart, R.R. M. Watts, T.J. Ward and H.P. Possingham (in prep). Identifying the optimal trade-off effects when designing a Multiple-Use Marine Park with multiple objectives

Stewart, R.R., L. Kircher, M. Watts, T.J. Ward and H.P. Possingham (in prep). A systematic planning framework to manage the multiple uses of coastal and marine resources and minimize potential for conflict

Stoms, D.M. 1994. Scale dependence of species richness maps. Professional Geographer 46: 346-358.

Warman, L.D., A.R.E. Sinclair, G.G.E. Scudder, B. Klinkenberg, R.L. Pressey. Sensitivity of Systematic Reserve Selection to Decisions about Scale, Biological Data, and Targets: Case Study from Southern British Columbia. Conservation Biology 18: 655-666.

Watts, M. E., I. R. Ball, H. P. Possingham et al (2008). Marxan with zones - optimised landscape and seascape zoning using spatially explicit annealing, in prep.

## Appendix A. Example Input Parameter File

```
General Parameters
BLM  0
PROP  0.5
RANDSEED -1
NUMREPS 100

Annealing Parameters
NUMITNS 1000000
STARTTEMP -1
COOLFAC  0
NUMTEMP 10000

Input Files
INPUTDIR input
FEATNAME feat.dat
PUNAME pu.dat
PUVFEATNAME puvfeat.dat
BOUNDNAME bound.dat
ZONESNAME zones.dat
COSTSNAME costs.dat
ZONETARGETNAME zonetarget.dat
ZONECONTRIBNAME zonecontrib.dat
ZONECOSTNAME zonecost.dat

Save Files
SCENNAME california
SAVERUN 2
SAVEBEST 2
SAVESUM 2
SAVESCEN 2
SAVETARGMET 2
SAVESUMSOLN 2
SAVELOG 1
OUTPUTDIR output

Program control.
RUNMODE 1
MISSLEVEL  1
ITIMPTYPE 0
VERBOSITY 2
```