MARXAN conservation solutions

The Nature Conservancy

User Manual

MARXAN

Marxan version 2.43 & above

Issued: September 2020

2020 update by: Norma Serra-Sogas, Alessia Kockel, Edward T. Game, Hugh P. Possingham & Jennifer McGowan

Originally authored by: Edward T. Game & Hedley S. Grantham

With contributions from: Brooke Williams, Jeff Ardron, Carissa Klein, Dave Nicolson & Matt Watts

PacMARA Pacific Marine Analysis & Research Association

The University Of Queensland AUSTRALIA

Suggested citation for this manual:

Serra-Sogas, N., Kockel, A., Game, E. T., Grantham H., Possingham H.P., & McGowan, J. (2020). Marxan User Manual: For Marxan version 2.43 and above. The Nature Conservancy (TNC), Arlington, Virginia, United States and Pacific Marine Analysis and Research Association (PacMARA), Victoria, British Columbia, Canada.

Citation for Marxan software:

Ball, I. R., Possingham, H. P., & Watts, M. E. (2009). Marxan and relatives: Software for spatial conservation prioritization. In A. Moilanen, K. A. Wilson, & H. P. Possingham (Eds.), *Spatial conservation prioritisation: Quantitative methods and computational tools* (pp. 185–210). Oxford University Press.

Preface

Scope and Aim

This manual is intended to equip readers with the basic knowledge required to use Marxan version 2.43 and above (Ball et al., 2009). We cover all the necessary data inputs, as well as the steps required to successfully execute the program and interpret the results. We focus on the practicalities of using Marxan rather than the principles of reserve system design and optimization algorithms that solve the reserve system design problem. Some of this information is available in the appendices. In other cases, we attempt to direct readers to the appropriate sources. We provide some guidance to the sorts of problems Marxan can solve. It is important, however, to understand how Marxan works in order to avoid solving the wrong problem or misinterpreting the solutions. Marxan can be a very powerful tool, but if misused, it can provide misleading advice and undermine the credibility of systematic conservation planning.

Complementary Literature

Where this manual seems light on detail or lacking in specific direction, additional information may be found in the Marxan Good Practices Handbook, available for download at https://marxansolutions.org/ and https://pacmara.org/mgph-v2. This manual and the Marxan Good Practices Handbook should be used together. This Manual and the Handbook provides the resources needed to undertake highly skilled and defensible analysis using Marxan. In addition, we strongly suggest reading some of the many peer reviewed articles that use Marxan in conservation applications. These articles demonstrate what types of questions Marxan is able to answer, how conservation problems are set up, the kinds of data that can be used, and how different objectives and constraints influence the resulting reserve solutions.

Acknowledgements on the update

Funding for this update to the Marxan manual was provided by a grant to The Nature Conservancy from Margaret A. Cargill Philanthropies to advance spatial action mapping and planning in conservation decision-making.

Standing Acknowledgements from 2018 version

Preparation of this version [version 1.8] of the Marxan manual has been as much a compilation exercise as a writing exercise and as a result, we owe a great deal of thanks to a large number of people.

First and foremost, a huge amount of credit must be given to Ian Ball and Hugh Possingham, the original developers of Marxan and authors of the previous manual. It is through their manual that both of us [E.T. Game and H.S. Grantham], and many others, have learned to use Marxan. Much of their original manual has been incorporated into this version, especially in the technical appendix.

We also wish to acknowledge the significant contribution of Jeff Ardron. Not only did Jeff provide valuable technical and editorial advice, but without his enthusiasm and belief in good conservation practices, neither this manual nor the good practices handbook would have become a reality. We would also like to thank Carissa Klein, Dave Nicolson and Matt Watts for their excellent technical and editorial advice throughout the process.

Some of the introductory paragraphs have been modified from chapter 1 of the Marxan Good Practices Handbook and for this we thank Hugh Possingham, Jennifer Smith, Krista Royle, Dan Dorfman and Tara Martin.

Lindsay Kircher and Dan Segan, as new users to Marxan, gave insightful and helpful comments on an early draft of this manual and its tutorials. And finally, thanks go to the funders of this work, the Packard Foundation's Marine Ecosystem-Based Management Tool Innovation Fund, and to the Pacific Marine Analysis and Research Association (PacMARA) for submitting the funding proposal.

Table of Contents

1 Introduction 1-1

1.1 What is Marxan? 1-1

1.2 Marxan as a Decision Support Tool for Systematic Conservation Planning 1-4

1.3 Questions Marxan can Help Answer 1-5

1.4 Mathematical Formulation of Marxan 1-5

1.5 How Does Marxan Find Good Solutions? 1-9

1.6 Advantages of Marxan 1-9

1.7 Primary Assumptions 1-10

1.8 Limitations of Marxan 1-11

1.9 Marxan Relatives 1-11

2 Download, Software Requirements & Supporting Software 2-13

2.1 Software Download 2-13

2.2 System Requirements 2-13

2.2.1 Using Marxan with Windows 2-14

2.2.2 Using Marxan with MacOSX and Linux 2-14

2.3 Supporting Software 2-15

2.3.1 Zonae Cogito 2-16

2.3.2 CLUZ (Conservation Land-Use Zoning) 2-16

2.3.3 ArcMarxan and QMarxan plugins for ArcGIS and QGIS 2-16

3 Getting Started 3-17

3.1 How to Use this Manual 3-17

3.2 Courses & Training 3-17

3.3 Key Steps of a Marxan Analysis 3-18

4 Pre-Processing of Data 4-20

4.1 Overview of Data Preparation 4-20

4.2 Dividing the Planning Region into Planning Units 4-20

4.3 Determining the Distribution of Features & Costs 4-22

5 Input Files, Parameters & Variables 5-27

5.1 Overview of Input Files 5-27

5.2 Input File Management 5-28
5.3 Required Files 5-29
5.3.1 The Input Parameter File (input.dat) 5-29
5.3.2 The Conservation Feature File (spec.dat) 5-48
5.3.3 The Planning Unit File (pu.dat) 5-58
5.3.4 The Planning Unit versus Conservation File (puvspr.dat) 5-62
5.4 Optional File 5-65
5.4.1 The Boundary Length File (bound.dat) 5-65
6 Running the Software 6-69
7 Outputs 7-70
7.1 Overview of Output Files 7-70
7.2 Output File Management 7-71
7.3 Screen Output 7-71
7.3.1 Silent Running 7-71
7.3.2 Results Only 7-72
7.3.3 General Progress 7-72
7.3.4 Detailed Progress 7-74
7.4 Output Files 7-74
7.4.1 Output File Format 7-75
7.4.2 Solution for Each Run (e.g., output_r001.csv) 7-75
7.4.3 Best Solution (e.g., output_best.csv) 7-76
7.4.4 Missing Values for Each Run (e.g., output_mv001.csv) 7-77
7.4.5 Missing Value Information for the Best Run (e.g., output_mvbest.csv) 7-78
7.4.6 Summary Information (e.g., output_sum.csv) 7-78
7.4.7 Scenario Details (e.g., output_sen.dat) 7-80
7.4.8 Summed Solution (e.g., output_ssoln.csv) 7-81
7.4.9 Screen Log File (e.g., output_log.dat) 7-82
7.4.10 Penalty Files (e.g., output_penalty.csv and output_penalty_planning_units.csv) 7-82
7.4.11 Solution Matrix File (e.g., output_solutionmatrix.csv) 7-82
7.4.12 Snapshot File (e.g., output_snap_r00001t01000.txt) 7-83
8 Getting Good Results 8-84

8.1 Experimentation 8-84
8.2 Visual Inspection 8-84
8.3 Sensitivity Analyses 8-85
8.4 Marxan Community 8-86
Glossary 87
References 91
Appendix A- Troubleshooting 94
A-1 Invalid input file (input.dat) 94
A-1.1 “Input file input.dat not found” 94
A-1.2 “Entering in the data files; Planning Unit files input/pu.dat has not been found” 94
A-1.3 “Error: Cannot save to log file” 95
A-1.4 “Species file has not been found” 96
A-1.5 “PU v Species file input/ not found” 96
A-2 Invalid planning unit file (pu.dat) 97
A-2.1 “A connection is out of range” 97
A-2.2 “Entering in the data files” 98
A-2.3 “Species are already adequately represented”, “Run 1” or “Error reading planning units” 98
A-3 Errors related to an invalid species file (spec.dat) 99
A-3.1 “Planning Units names read in” 99
A-3.2 Missing value outputs contain negative feature id and no results 99
A-3.3 “(x number) *species cannot meet target”* with targets set as a proportion 100
A-3.4 “(x number) *species cannot meet target”* with targets set as an amount 101
A-3.5 Outputs files contain zeros in all columns 101
A-3.6 “(x number) conservation values counter, (x number) big matrix size, (x %) density of matrix” 102
A-4 Invalid planning versus species file (puvspr.dat) 102
A-4.1 “(x number) connections entered"” 102
A-4.2 "(x number) conservation values counter, (x number) big matrix size, (x %) density of matrix” 103
A-5.1 Marxan runs but representation numbers are incorrect 103

A-5 Invalid boundary file (bound.dat) 104
A-5.1 Output files do not have all the planning units 104
A-5.2 "A connection is out of range" 104
A-5.3 "(x number) *species read in*" 104
Appendix B- Marxan Technical Information 105
B-1 The Objective Function 105
B-1.1 Cost 105
B-1.2 Boundary and Boundary Length Modifier (BLM) 105
B-1.3 Features Penalty Factor (FPF) 106
B-1.4 Spatial feature penalties 108
B-1.5 Cost Threshold Penalty 110
B-2 Optimisation Methods 112
B-2.1 Simulated Annealing 112
B-2.2 Iterative Improvement 115
B-2.3 Other Heuristic Algorithms 116

List of Figures

Figure 1. Three possible types of planning units (PUs) that could be used in Marxan. 4-21

Figure 2. Determining the distribution of point, line, and polygon features across planning unit (PU). 4-23

Figure 3. GIS workflow for determining the distribution of features and costs. 4-24

Figure 4. Recommended set up for a Marxan folder. 5-28

Figure 5. An example of the Input Parameter File (input.dat). 5-30

Figure 6. An example of the Conservation Feature File (spec.dat). 5-49

Figure 7. An example of a subset of the Planning Unit File (pu.dat). 5-58

Figure 8. An example of a subset of the Planning Unit versus Conservation File (puvspr.dat). 5-64

Figure 9. An example of the Boundary Length File (bound.dat). 5-66

Figure 10. A successful run in Marxan.exe. 6-69

Figure 11 Example of the on screen provided using verbosity 1 (Results Only). 7-72

Figure 12. Example of the on screen provided using verbosity level 2 and 3 (General Progress and Detailed Progress). 7-73

Figure 13. An example of a subset of a solution file (e.g., output_r001.csv). 7-76

Figure 14. An example of a solution file (e.g., output_mv001.csv). 7-78

Figure 15. Example of the summary output file (output_sum.dat). 7-80

Figure 16. Example of the scenario details output file (output_sen.dat). 7-80

Figure 17. Example of a subset of the summed solution output file (output_ssoln.csv) 7-81

Figure 18. Example of a subset of the solution matrix file (output_solutionmatrix.csv). .. 7-83

List of Tables

Table 1. Comparison of Marxan supporting software. 2-15
Table 2. Marxan input files and default names. 5-27
Table 3. Variable names and default values for the Input Parameter File (input.dat). 5-31
Table 4. Variable of the Conservation Feature File (spec.dat). 5-48
Table 5. Variable names and requirements for Planning Unit Fie (pu.dat). 5-58
Table 6. Variable names and requirements for Planning Unit versus Feature File (puvspr.dat). 5-62
Table 7. Variable names and requirements for Boundary Length File (bound.dat). 5-66
Table 8. Basic summary information of each run. 7-73
Table 9. Output file types and names. 7-74
Table 10. Description of Missing Value File Headers. 7-77
Table 11. Description of Summary File Headers. 7-78

List of Boxes

Box 1. A Brief History of Marxan 1-1
Box 2. Key Terms & Definitions 1-2
Box 3. Minimum Set Problem 1-3
Box 4. Key Stages of Systematic Conservation Planning 1-4
Box 5. Marxan Score Calculation 1-8
Box 6. The difference between models, problems, and algorithms 1-9
Box 7. Marxan workflow for a standard project 3-19
Box 8. Limited Availability and Quality of Data 4-20
Box 9. Incorporating Cost Data in Marxan 4-25
Box 10. Options for Developing Input Files 5-28
Box 11. The Boundary Length Modifier (BLM) Value 5-35
Box 12. Setting the Boundary Length Modifier (BLM) 5-36
Box 13. The Feature Penalty Factor (FPF) 5-53
Box 14. Calibrating Feature Penalty Factor (FPF) 5-54
Box 15. Not the Answer! 7-70
Box 16. How to visualize Marxan solutions in QGIS or ArcMap 7-76
Box 17. How to visualize Marxan summed solutions (or selection frequency) file in QGIS or ArcMap 7-82

List of Acronyms

BLM	Boundary Length Modifier
GIS	Geographic Information System
GIU	Graphical User Interface
PU	Planning Unit
SCP	Systematic Conservation Planning
FPF	Feature Penalty Factor

1 Introduction

1.1 What is Marxan?

Marxan is a freely available software designed to help decision makers find solutions to conservation, and other spatial planning problems (see Section 1.5 for examples). Marxan was initially developed to address the problem of creating a protected area system, while accommodating pre-existing users. The basic idea behind a reserve design problem is that a conservation planner has a large number of potential sites (or '*planning units*') from which to select new conservation areas. The objective is to devise a reserve system which is made up of a selection of these planning units which will solve a problem that includes several ecological, social and economic criteria and principles.

Marxan is primarily intended to solve a class of reserve design problem known as the '*minimum set problem*', where the goal is to achieve some minimum representation of biodiversity features for the smallest possible cost (McDonnell et al., 2002). The rationale here is that cheaper or less socially disruptive reserve networks are more likely to be implemented. In minimum set problems, the features of biodiversity that you wish to conserve (hereafter referred to as 'conservation features') are treated as constraints to solutions of the problem (Possingham et al., 2000) – e.g. we must conserve at least 2000 elephants or 3000 hectares of coral reefs. Conservation features can reflect the presence or distribution of species, habitats, and any other elements of biodiversity that can be spatially represented. They may also represent socioeconomic features or values, such as spiritual sites or fishing areas. Given reasonably comprehensive data on features, Marxan aims to find

Box 1. A Brief History of Marxan

The Marxan software is primarily a product of Ian Ball's PhD thesis, which was supervised by Professor Hugh Possingham at the University of Adelaide, Australia. The initial stage of the software, called SPEXAN (stands for 'SPatially EXplicit ANnealing'), was funded by Environment Australia. Marxan was later developed as a modified version of SPEXAN to meet the needs of the Great Barrier Reef Marine Planning Authority (GBRMPA) in their 2003-2004 rezoning plans. It was partially funded by the GBRMPA and the US National Marine Fisheries Service. At that point the software was renamed to Marxan which stands for 'MARine reserve design using spatially eXplicit ANnealing'; although it is just as applicable to terrestrial and freshwater systems.

a suite of solutions for reserve network designs that meet user-defined ' targets' for the minimum 'cost' (Possingham et al., 2000).

Marxan contains a variety of algorithms, but the most commonly used is simulated annealing, to find good solutions to a version of the 'minimum set problem' that includes spatial compactness. Algorithms solve problems, and the Marxan problem is to find a reserve system that is a combination of three objectives: conserve a certain amount of every conservation feature, minimise the impact of those decisions on other users of the landscape, and keep the reserve system compact.

With each run, Marxan produces multiple 'good' solutions for reserve network configurations, thus increasing the chance of finding a solution that maximizes conservation interests while minimizing socioeconomic or other types of impacts. It can enhance the rigor, transparency and repeatability of decisions that are inherently complex and potentially subjective. Marxan is meant to support decision-making and act as a starting point for planning discussions. Its results must be refined by decision-makers to consider the full range of political, socio-economic and practical factors affecting planning implementation.

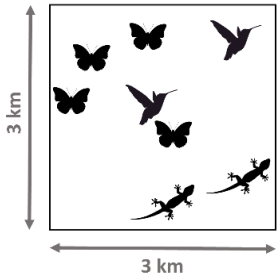
Box 2. Key Terms & Definitions

- Conservation feature: The feature (e.g., species, habitat, process, cultural site, etc.) for which a target is set for inclusion in the reserve system.
- Conservation targets: The minimum quantity or proportion of the conservation feature in the planning region to be included in the solution (e.g., protect 30% of each habitat type in the reserve system).
- Cost: A function Marxan acts to minimize in its pursuit of achieving the conservation targets. It is meant to reflect the cost of including a 'planning unit' in the reserve design. Costs are flexible and often pertain to socioeconomic implications of establishing a reserve (e.g., land acquisition cost; management cost; opportunity cost).
- Planning region (*also known as planning area/extent, study area*): The spatial domain over which the planning process occurs. This area is subdivided into planning units.
- Planning unit: Spatial units within the planning region, which can be defined as regular shapes (grids or hexagons) or irregular landscape-based features (e.g., watersheds).
- Solution: A binary output of Marxan reflecting whether a planning unit is selected (1) or not selected (0) as part of the reserve system.
- Selection Frequency: the summed output of all solutions

See Glossary for other relevant terms and definitions.

Box 3. Minimum Set Problem

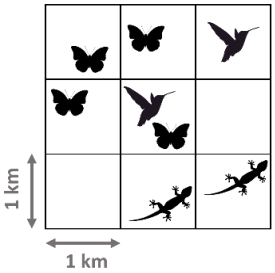
The following vignettes illustrate the minimum set reserve design problem, where conservation targets are set for three different conservation features. Marxan will select planning units that conserve each conservation feature at least once (as in this case a conservation target of one is set for every species) for a minimum total cost.



In this example, the planning region is a 3 km^2 plot of land.

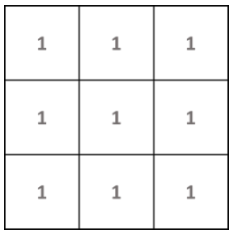
Within this planning region, there are three species. These are the **conservation features** in need of conservation.

The **conservation target** is to protect one occurrence of each species within a reserve system.



Prior to running Marxan, the planning region is divided into smaller sites or '**planning units**' of 1 km^2 in size.

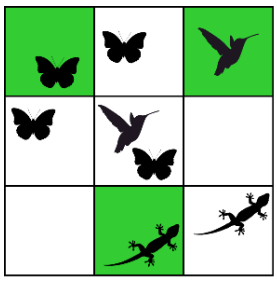
The presence of species in each planning unit is then calculated.



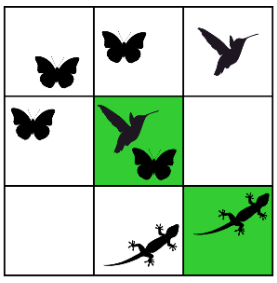
A **cost** value is assigned to each planning unit. In this example, cost is equivalent to the area of the planning unit (1 km^2).

Hence, each planning unit is assigned a cost value of '1'.

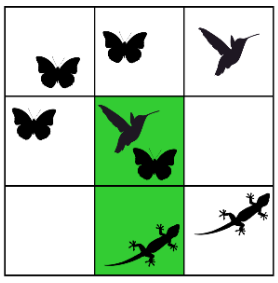
Marxan uses simulated annealing to identify multiple alternative good **solutions** (i.e., combinations of reserve sites) that meet the conservation targets for the minimum cost. Three possible solutions are shown below, one is more expensive than the others, one is both cheap and compact.



Solution 1



Solution 2



Solution 3

1.2 Marxan as a Decision Support Tool for Systematic Conservation Planning

Marxan is often employed as part of a larger systematic conservation planning (SCP) process. SCP is a framework for identifying potential reserves that efficiently achieve a specific set of objectives, commonly some minimum representation of biodiversity (Margules & Pressey, 2000; Pressey & Bottrill, 2009). It is a major departure from conventional conservation planning approaches, which have often been applied to select reserves based on urgency, scenery, the evaluation of planning units independently, and ease of designation (Kukkala & Moilanen, 2013). SCP is widely considered 'best practice' in creating systems of protected areas because it facilitates a transparent, inclusive and defensible decision-making process. The importance of these principals for conservation are discussed in the Marxan Good Practices Handbook at www.marxansolutions.org and https://pacmara.org/mgph-v2.

The SCP framework consists of 11 stages (Box 4) encompassing the design, implementing, and monitoring of conservation areas (Pressey & Bottrill, 2009). The ninth stage, commonly referred to as 'spatial conservation prioritization', involves the selection of new reserves to achieve conservation objectives. Although Marxan can be used for a variety of purposes at various stages of the SCP process, it was designed primarily to facilitate spatial conservation prioritization by providing decision support. It also helps incorporate core conservation planning concepts, such as complementarity[1] and representativeness[2] into the earlier stages of identifying goals and objectives for the process (Kukkala & Moilanen, 2013).

Box 4. Key Stages of Systematic Conservation Planning

1. Scoping and costing the planning process
2. Identifying and involving stakeholders
3. Describing the context for conservation areas,
4. Identifying conservation goals
5. Collecting data on socio-economic variables and threats
6. Collecting data on biodiversity and other natural features
7. Setting conservation objectives (spatially explicit targets)
8. Reviewing current achievement of objectives
9. Selecting additional conservation areas
10. Applying conservation actions to selected areas
11. Maintaining and monitoring conservation areas

[1] A comprehensive reserve system contains examples of many biodiversity features (Possingham et al., 2006).
[2] Representative referred to the need for reserves to represent, or sample, the full variety of biodiversity, ideally at all levels of organization (Margules and Pressey, 2000).

1.3 Questions Marxan can Help Answer

Marxan has been applied to answer different conservation-related questions for terrestrial, marine and freshwater areas. One of the earliest and most successful applications of Marxan was the rezoning of the Great Barrier Reef in Australia (Fernandes et al., 2005). More recently, Marxan has been applied within the 'Development by Design' framework to help balance the needs of biodiversity alongside extractive industries. There is an excellent example of how Marxan was used to create Mongolia's system of protected areas (Heiner et al., 2019). There are now hundreds of examples of Marxan being successfully applied to inform protected area systems on the land or sea (Sinclair et al., 2018).

While this decision support tool was originally designed to ensure species and ecosystem representation in biodiversity conservation planning, and has primarily been applied to that field, it has proven applicable to a broad range of challenges where we wish to plan conservation actions spatially. Marxan can generally assist all problems related to the spatially-explicit selection of 'minimum sets'. For example, Marxan has been employed to identify a spatially efficient suite of "fishing areas" (Natalie C. Ban & Vincent, 2009), where anti-poaching patrols should be deployed (Plumptre et al., 2014) and to prioritize land for restoration for improving habitat availability for mammal species (Crouzeilles et al., 2015). Additional case studies illustrating the various applications of Marxan can be found at www.maxansolutions.org. The Marxan Good Practices Handbook also contains several good examples, along with best practices for using the tool in various contexts.

 As with all decision support software, it is important to understand that **Marxan supports decision-making, it does not make the decisions**. Marxan solutions form the basis of discussions which underpin the planning process. Many additional socio-political and ecological factors will influence the final design of any spatial plan.

1.4 Mathematical Formulation of Marxan

Marxan finds solutions to a well-defined mathematical problem. For example, it minimizes the combined cost of the reserve network and the boundary of the entire network, while meeting a set of conservation targets. This problem can be expressed mathematically as:

$$minimize \sum_i^{N_s} x_i c_i + b \sum_i^{N_s} \sum_h^{N_s} x_i (1 - x_h) c v_{ih} \quad (1)$$

subject to meeting all conservation targets

$$\sum_i^{N_f} x_i r_{ij} \geq T_j \forall j \quad (2)$$

and x_i is either 0 or 1

$$x_i \in \{0,1\} \; \forall \; i$$

where r_{ij} is the occurrence level of conservation feature *j* in planning unit *i,* c_i is the cost of planning unit *i*, N_s is the number of planning units, N_f is the number of conservation features, and T_j is the target for conservation feature *j.* The variable x_i has value '1' for planning units selected to form part of the reserve network, and value '0' for sites not selected.

The first term in Equation 1 is a penalty associated with the cost of the network. The second term is a penalty associated with the spatial configuration or shape of the network, also known as boundary cost. The parameter cv_{ih} reflects the cost of the connection between planning unit *i* and planning unit *h*, typically measured as the shared boundary between these two planning units. If one planning unit is in the reserve system, and the other is not, then a connection cost is applied. If both planning units are out or in, the connection cost is not paid. The parameter b is the boundary multiplier (or the boundary length modifier, BLM), a user-defined parameter that controls the importance of minimizing the boundary cost (or total boundary length) of the reserve system. The higher the *b* value, the more importance is given to achieve a more compact reserve configuration. The units of *b* have to be set in a way that allow the boundary length to be comparable to the cost measure of the planning units in order to have an effect. For example, if the most costly planning unit is 100 and typical values of *cv* are 1000, then the BLM should start at 0.01 in order to see an effect on the spatial configuration of the reserve system.

In Equation 2, T_j is the target for a given biodiversity feature. Targets can be expressed as an amount (e.g., hectares of a particular habitat) or as the number of occurrences (number of individuals) of every feature that is the focus of the reserve system. Marxan also allows for the inclusion of targets that specify the minimum clump size for representing a biodiversity feature in the reserve network, the minimum number of occurrences of a biodiversity feature, or the number of mutually separated occurrences of a feature required in the reserve network. These software features can be used to ensure a minimum reserve size or ensure that several representations of each conservation feature are adequately separated in space to address the idea of risk spreading.

Targets in Marxan are specific to the conservation features and not for other configuration characteristics, such as the minimum size of areas zoned, or the number of distinct areas zoned for conservation.

Marxan solves the problem by placing the objectives (Equation 1) and the constraints (Equation 2) together into an 'objective function' by transforming the constraints into an additional penalty term. This allows Marxan to calculate a value for a collection of planning

units, which in turn can be used to compare alternate solutions (i.e. collections of planning units) and hence identify better solutions.

Thus, the **objective function** in Marxan takes the form:

$$\sum_i^{N_s} x_i\, c_i + b \sum_i^{N_s} \sum_h^{N_s} x_i(1 - x_h) cv_{ih} + \sum_j^{N_f} FPF_j FR_j H(s) \left(\frac{s}{T_j}\right) \quad (3)$$

where the first term is the total cost of the reserve network and the second term if the boundary cost of the reserve network multiplied by the boundary length modifier.

The third term includes the target constraints presented in Equation 2, but now as a shortfall penalty equation. The terms FPF_j and FR_j are the feature penalty factor (also commonly referred to as the 'species penalty factor (SPF)') and feature representation respectively. FPF_j is a scaling factor that determines the relative importance of meeting the representation target for feature *j*. FR_j is computed as the representation cost of meeting the representation target of feature *j*.

The shortfall *s* is the amount of the representation target not met and is given by:

$$s = T_j - \sum_i^{N_f} x\, r_{ij}. \quad (4)$$

The Heaviside function, H(s), is a step function which takes a value of zero when $s \geq 0$ and 1 otherwise. The feature specific parameter T_j is the target representation for feature *j*. The expression $\left(\frac{s}{T_j}\right)$ is the measure of the shortfall in representation for feature *j*. It is reported as a proportion and equals '1' when feature *j* is not represented within the configuration and approaches '0' as the level of representation approaches the target amounts. The Heaviside function ensures the whole equation becomes zero when the representation is greater than the target amount.

The shortfall penalty is zero if every biodiversity feature *j* has met its representation target in the selected reserve network. It is greater than zero if the targets are not met and gets larger as the gap between the target and the amount not included in the solution increases.

More detail on the objective function and how each of the different terms is calculated can be found in Appendix B-1. Section B-1.3 of this manual contains details of how to control which features contribute to the objective function and how penalties inform the results.

Box 5. Marxan Score Calculation

The objective function in Marxan is:

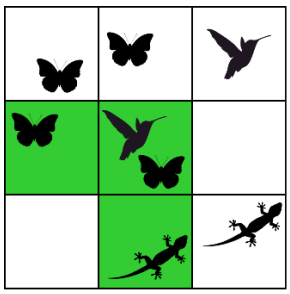
$$\sum_{i}^{N_s} x_i\, c_i + b \sum_{i}^{N_s} \sum_{h}^{N_s} x_i(1 - x_h) cv_{ih} + \sum_{j}^{N_f} FPF_j FR_j H(s) \left(\frac{s}{T_j}\right) \qquad (4)$$

which can be simplified to derive the 'Marxan score':

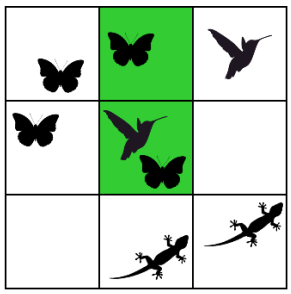
$$\overbrace{\sum_{PUs} \text{Cost}}^{1} + \overbrace{BLM \sum_{PUs} Boudary}^{2} + \overbrace{\sum_{\substack{Con \\ Value}} FPF \times Penalty}^{3} = \boldsymbol{Marxan\ Score} \qquad (5)$$

The score is the sum of the three terms in the simplified function: *1)* the sum of the costs of the selected planning units; *2)* the total perimeter of the selected planning units; and *3)* the total penalty incurred if conservation targets are not met. Marxan's simulated annealing algorithm searches for planning unit combinations that minimizes the objective function.

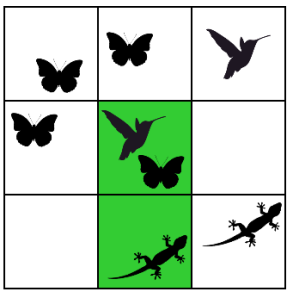
Let's use the simple example below to calculate the 'Marxan Score' for three possible solutions. In this case, all planning units (PU) have the same cost (PU cost = 1) and each PU boundary has a boundary cost of 1. The boundary length modifier (BLM) is set to 1, while the feature penalty factor (FPF) value for missing targets is 10. The conservation target is to protect one occurrence of each species. Thus, solutions 1 and 3 meet all conservation targets, while solution 2 does not (only two of three species are included in reserves).



Solution 1
PU Cost = 3
Boundary Cost = 8
Penalty = 0
Marxan Score = 11



Solution 2
PU Cost = 2
Boundary Cost = 6
Penalty = 10
Marxan Score = 18



Solution 3
PU Cost = 2
Boundary Cost = 6
Penalty =0
Marxan Score = 8

Based on the Marxan scores, solution 3 with the lowest score is the most efficient solution.

1.5 How Does Marxan Find Good Solutions?

The number of possible solutions to even a small reserve selection problem is vast. For a 100 planning unit problem, there are 2^{100} = 1,267,650,600,228,229,401,496,703,205,376 possible combinations to select from! Because finding the best solution to this problem is complex and time consuming, we commonly use computer algorithms to help us.

An algorithm is a set of rules that can be applied to "solve" a well-defined mathematical problem. There are two broad classes of algorithmic solvers commonly applied in conservation planning: exact and heuristic (non-exact). Exact algorithmic solvers can identify the single optimal solution. Heuristics, on the other hand, provide a variety of near-optimal solutions, which provide a set of "good" options for planners and stakeholders to consider (Cabeza, 2003; McDonnell et al., 2002; Possingham et al., 2000). Marxan uses heuristic solvers, specifically, 'simulated annealing' (Appendix B-2.1).

Box 6. The difference between models, problems, and algorithms

There is some confusion about the differences between models, problems and algorithms that we feel is important to clarify for new Marxan users. First, Marxan is not a model. A *model* is a prediction of the future state of a variable (e.g. species density under future climate change), or the state of a variable under conditions for which we have no data (e.g. whether a species exists in a location that we haven't visited). We often use modelled data, such as species distributions, as conservation features but Marxan does not predict or model anything. Marxan is a software that applies an *algorithm* (procedure) to find one or more solutions to a well-defined problem. A *problem*, in this context, is the mathematical expression of what we want to achieve in our conservation plans. Marxan solves a spatially explicit variant of the "minimum set" problem, as defined in Section 1.4.

1.6 Advantages of Marxan

Marxan is the most widely used systematic conservation planning tool globally. Marxan can be used to help solve any spatial allocation problem (e.g., where to fish, where to grow food, where to conserve species, where to restore habitat). Some of the most significant benefits of Marxan is its ability to:

- Generate reserve configurations that meet stated conservation targets
- Produce multiple near optimal solutions to conservation planning problems

- Enhance the rigor, transparency and repeatability of processes that are inherently complex and potentially subjective
- Provide a flexible environment in which to design protected areas by allowing users to experiment with different conservation and management scenarios
- Support participatory planning processes and help negotiate acceptable outcomes amongst multiple stakeholders.

Marxan is a free and open-access software, making it increasingly useful for applications in both developed and developing countries where access to expensive tools and licenses might be cost or capacity-prohibitive. Marxan is also becoming increasingly accessible to users world-wide thanks to the growing user community, technical support through online user groups, and documentation of successful applications.

Additional benefits of Marxan are discussed in the Marxan Good Practices Handbook.

1.7 Primary Assumptions

The use of an automated reserve selection tool such as Marxan rests upon some key assumptions. Although it can be very powerful in solving difficult site selection problems, some issues cannot always be incorporated, so the use of Marxan must necessarily rest on certain assumptions.

Perhaps the assumption most difficult to attain, and thus most frequently violated, is that the spatial distribution of data used in a Marxan analysis is assumed to be consistent. This is not to say that the same features are found everywhere, but that the data was collected in a way that the same features would be found everywhere if they existed there, i.e. the data is not spatially biased. For instance, if using species occurrence data to select reserves, it is highly likely that the detection of species has not been uniform across the planning region. Collections or observations may have occurred more intensively around research field stations, populated areas, or easily accessible places, such as near roads. This will be interpreted by Marxan as a true reflection of the species' full distribution and will subsequently direct the reserve solutions to those well-studied areas. This may have a substantial bearing on the shape of the entire reserve system, particularly if any emphasis is placed on system compactness.

One way of partially overcoming this bias is to model the likely distribution of species or habitats based on biophysical data. An additional and perhaps simpler way to overcome biases due to sampling intensity is to use surrogate measures such as habitat type or even physical variables to represent the distribution of biodiversity we wish to conserve. In some cases, however, it would be irresponsible to neglect known occurrences of valuable

conservation features such as highly threatened species. A method of dealing with such situations is provided in Section 4.3. When it is believed that data may be spatially biased, this bias should be documented.

This version of Marxan does not consider uncertainty in the data[3]. It assumes that all feature representations are true, and that all occurrences of that feature are of equal value. In reality, a conservation planner may be very confident about the presence of a feature in some areas of its distribution and less so in others. Subtleties such as this require careful evaluation of the Marxan outputs to ensure that they actually capture the desired conservation features. Always consider the cliché that the quality of the results you get out of Marxan can be no greater than the quality of the raw or modelled data you put in. The Marxan Good Practices Handbook suggests some methods for conducting robust analysis using weaker datasets.

1.8 Limitations of Marxan

Technical limitations of Marxan will become apparent as you read this manual and, in many cases, can be overcome through data or scenario exploration (for examples see the Marxan Good Practices Handbook).

More subtle and yet more important, however, are the philosophical limitations of reserve design software. These should be well understood. Marxan operates as part of a planning process and is not designed to act as a stand-alone reserve design solution. Its effectiveness is dependent upon the involvement of people, the adoption of sound ecological principles, the establishment of scientifically defensible conservation goals and targets and the development and inclusion of quality spatial datasets. Marxan should be used as part of a systematic conservation planning process and in collaboration with other forms of knowledge. These other forms of knowledge are essential to the refinement of Marxan inputs, the interpretation of Marxan outcomes and the refinement of final conservation area boundaries.

1.9 Marxan Relatives

Marxan is a suite of software that can be applied to solve different types of planning challenges. It is worth mentioning that Marxan evolved from this standard two zone application (i.e., planning units are either “in” or “out” of a single zone if selected by Marxan) to consider more complex challenges like planning for multiple management zones (Marxan with Zones), incorporating symmetric and asymmetric connectivity (Marxan with

[3] Marxan with Probability can deal with certain aspects of uncertainty.

Connectivity), and incorporating probabilities (Marxan with Probability). Although the basic operation of these versions is the same as those described in this manual, they each have idiosyncrasies that require additional steps, inputs and considerations for operation. These extended versions of Marxan are briefly described below and are available at www.marxansolutions.com.

- **Marxan with Zones** (Watts et al., 2009) has the same functionality as Marxan but extends on the range of problems the software can solve and allows for the incorporation of multiple costs and zones into a systematic planning framework.
- **Marxan with Connectivity** (Beger et al., 2010) is an extension of the Marxan software family that allows for more sophisticated connectivity considerations in spatial planning. A supporting GUI called Marxan Connect (Daigle et al., 2018) can assist with using Marxan with Connectivity and is available for download from https://marxanconnect.ca/.
- **Marxan with Probability** (Game et al., 2008; Tulloch et al. 2016) is Marxan with an additional objective function term that incorporates the probability of protecting conservation features given the uncertainty in its distribution, or the probability of features in a site are lost or degraded due to threatening processes.
- **Marxan Web:** Marxan Web is a decision-support platform for doing systematic conservation planning over the web and for sharing the results amongst stakeholders and the conservation community.

2 Download, Software Requirements & Supporting Software

2.1 Software Download

Chances are that if you are reading this manual you have already downloaded Marxan. If not, Marxan can be downloaded from https://marxansolutions.org/software. The download package includes the latest versions of the software (2.43 and above) and a test dataset. Additional supporting materials are also available.

The following files are included in the download package:

1. Marxan.exe (the Marxan program executable)
2. A folder labelled 'MarxanData', containing the input.dat file (an example Input Parameter File) along with the other input files

You will require around 2 MB of free disk space for Marxan and the associated files. These files can be saved anywhere on the computer.

2.2 System Requirements

The system requirements for running Marxan are quite modest. As a rule of thumb, if a computer is powerful enough to run GIS software, then it will be more than adequate for running Marxan. The more planning units, conservation features and optional advanced Marxan settings you have, the slower Marxan will run. Of course, the more powerful your computer (MHz and RAM), the faster Marxan will run. Depending on these factors, the time required for Marxan to provide 100 good solutions to your problem can range from minutes to days[4].

The Marxan software is available for different operating systems, including Windows, MacOSX and Linux. More information is provided below.

[4] It is usually the advanced features of Marxan (such as separation distance and minimum clump size) that can slow the analysis down significantly, especially with large numbers of planning units. We therefore recommend that initial test runs do not make use of these advanced features, such that the basic operationally of the Marxan input is first tested and verified.

2.2.1 Using Marxan with Windows

Marxan.exe Marxan version 2.43 32-bit Windows
Marxan_x64.exe Marxan version 2.43 64-bit Windows

The 64-bit native compiled executable of Marxan allows for increased data segment size greater than 2GB. It is designed for use with monolithic datasets and provides a large speedup and increase in data capacity for monolithic datasets. You should use the 64-bit version of Marxan if you are using an x64 Windows operating system such as Windows XP Professional x64, Windows Server 2003, Windows Server 2008, Windows Vista 64-bit edition and Windows 7 64 bit. These executables also ensure continued compatibility of Marxan with new generations of 64-bit Windows operating systems.

You should use the 32-bit version of Marxan if you are using a 32-bit Windows operating system such as Windows XP.

To use Marxan with Windows:

- Copy the relevant executables to a directory containing your Marxan input parameter file (input.dat)
- Run the relevant executable to perform Marxan analysis, in the usual way

2.2.2 Using Marxan with MacOSX and Linux

Marxan_v243_Mac32 Marxan version 2.43 32-bit MacOSX
Marxan_v243_Mac64 Marxan version 2.43 64-bit MacOSX
Marxan_v243_Linux32 Marxan version 2.43 32-bit Linux
Marxan_v243_Linux64 Marxan version 2.43 64-bit Linux

The MacOSX and Linux versions of Marxan have been tested for robustness and compatibility with other versions of Marxan, however the amount of testing performed is less than that performed for Windows versions of Marxan. If you experience problems with these versions of Marxan, please solicit advice from the Marxan Google group at marxan@googlegroups.com.

For these operating systems you will need to:

- Copy the relevant executables to a directory containing your Marxan input parameter file (input.dat)
- Open a terminal window in MacOSX or Linux (consult your OS documentation for instructions on doing this)
- Use the cd command to change to the directory containing your executable and input.dat files

- Set the execute bit for the MacOSX or Linux executables before you run them, like this:
  chmod a+x Marxan_v243_Linux64
- This will switch on the execute bit for all users with the Linux 64-bit version of Marxan Optimised version 2.43
- Run the relevant executable to perform Marxan analysis, like this:
  /Marxan_v243_Linux64
- This will run the executable for the Linux 64-bit version of Marxan Optimised version 2.43

2.3 Supporting Software

In this manual, we describe how to run Marxan as a stand-alone program. However, there are several, freely available, user interfaces that can assist in running Marxan. These include Zonae Cogito, CLUZ (Conservation Land-Use Zoning), the ArcMarxan plugin for ArcGIS, and the QMarxan plugin for QGIS. A brief description of these supporting software/plugins are described below. Table 1 provides a quick comparison view of the tools described below. Many users have found these interfaces particularly helpful for generating appropriate input files and displaying Marxan outputs. Guidance on using these programs can be obtained from corresponding websites or user manuals.

Table 1. *Comparison of Marxan supporting software*

		Functionalities			
Software	GIS software required	Create Marxan input files	Runs Marxan	Parameter calibration	Direct visualisation of Marxan results
Zonae Cogito	N/A	No*	Yes	Yes	Yes
CLUZ	QGIS	Yes	Yes	Yes	Yes
ArcMarxan Toolbox	ArcGIS	Yes	Yes	Yes	No
QMarxan Plugin	QGIS	Yes	Yes	Yes	No

*Users can modify some parameters of Marxan input files.

2.3.1 Zonae Cogito

Zonae Cogito comes from the Latin 'Zonae', meaning zones, and 'Cogito', meaning to think or reflect on. In other words, the title means to think about zones. The purpose of the software is to act as a decision support system and database management system for the family of Marxan software. It incorporates open source GIS software components, and is a freely available software package at https://marxansolutions.org/software. Zonae Cogito was written by Matthew Watts and Romola Stewart from The Ecology Centre, University of Queensland.

If you are using Zonae Cogito 1.74 to run Marxan, the appropriate windows executable for your operating system will be automatically copied from your Zonae Cogito program folder to your Marxan database folder and executed when pressing the Run button. Zonae Cogito is only compatible with Windows operating systems. The current version of Zonae Cogito is compatible with 32 bit and x64 Windows operating systems.

2.3.2 CLUZ (Conservation Land-Use Zoning)

CLUZ (Conservation Land-Use Zoning software) is a QGIS plug-in that allows users to design protected area networks and other conservation landscapes and seascapes. It can be used for on-screen planning and also acts as a link for the Marxan conservation planning software. It was developed by Bob Smith and funded by the UK Government's Darwin Initiative. The tool can be accessed at https://anotherbobsmith.wordpress.com/software/cluz/

2.3.3 ArcMarxan and QMarxan plugins for ArcGIS and QGIS

The ArcMarxan and QMarxan Toolboxes are a free / open source python toolbox for ArcMap 10.2 (and above) and QGIS 3.x respectively, created by Apropos Information Systems. Both tools allow for Marxan data preparation, export input files for Marxan, calibration of parameters and analysis of results.

Download ArcMarxan at:
https://aproposinfosystems.com/en/solutions/arcgis-plugins/arcmarxan-toolbox/

Download QMarxan at:
https://aproposinfosystems.com/en/solutions/qgis-plugins/qmarxan-toolbox/

3 Getting Started

3.1 How to Use this Manual

The following chapters provide the basic knowledge required to use Marxan. They cover all the relevant parameters and necessary data inputs and provide guidance to successfully execute the program and interpret the results. For supplemental information, we highly recommend consulting the Marxan Good Practices Handbook which is available for download at https://marxansolutions.org. The manual and the Marxan Good Practices Handbook should be used in concert with each other, and together should provide the resources needed to undertake highly skilled and defensible analysis using Marxan.

This manual does not provide detailed instructions for creating input files with Marxan supporting software (see Box 10). Users should consult the appropriate user manuals for supporting software for generating input files.

This manual contains information regarding the calibration of four Marxan parameters: the boundary length modifier (Box 12. Setting the Boundary Length Modifier (BLM)), the number of repeat runs (Section 5.1.3 The Input Parameter File (input.dat)), the number of iterations (Section 5.1.3 The Input Parameter File (input.dat)), and the feature penalty factor (Box 14. Calibrating Feature Penalty Factor (FPF)). Calibration is a key part of running robust Marxan analyses. We strongly recommend consulting the Marxan Good Practices Handbook for guidance on proper calibration.

Along with the supporting materials above, we suggest reading some of the many peer reviewed articles that use Marxan in various conservation applications. Doing so will help build understanding on what types of questions Marxan is able to answer, how conservation problems are set up, the kinds of data that can be used, and how different objectives and constraints influence Marxan outputs.

3.2 Courses & Training

In addition to this manual, there are a number of courses and tutorials available on using Marxan. Tutorials on Marxan and supporting software are available at https://marxansolutions.org. There are also courses and trainings available through various institutions and organizations, such as the Pacific Marine Analysis and Research Association (PACMARA, https://pacmara.org).

PacMARA is a charitable organization of science and planning professionals dedicated to building and increasing capacity in marine and coastal planning. It offers technical and non-technical courses on Marxan (e.g., 3-day introductory training course on Marxan), Marxan extensions (e.g., Marxan with Zones and Marxan Connect) and related concepts and topics (e.g., Systematic Conservation Planning training). It also provides facilitation, analysis, and support for managers, policymakers, academics, and other members of the conservation community. Please visit PacMARA's website (https://pacmara.org) to enquire about support service and upcoming courses/trainings.

3.3 Key Steps of a Marxan Analysis

This section is intended to provide a brief guide to the essential steps of using Marxan. The primary technical steps to running Marxan are:

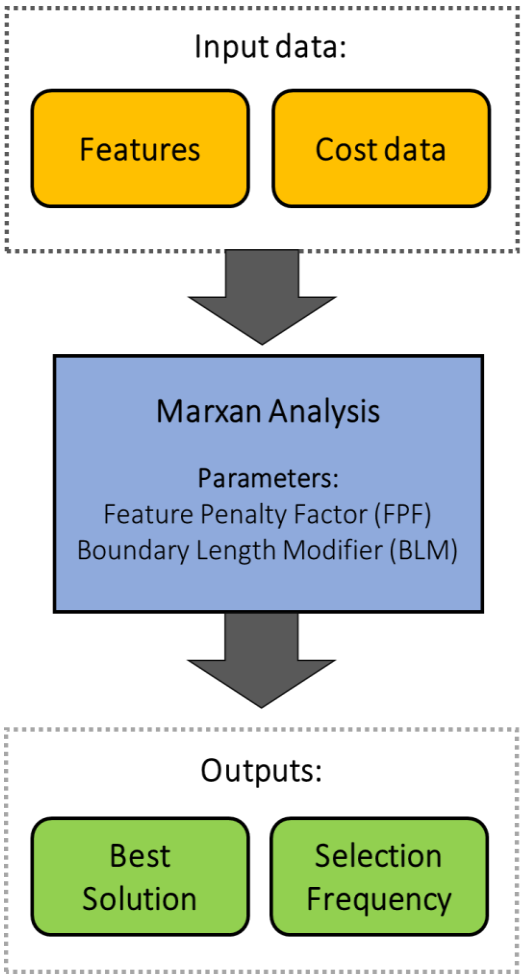
1. Pre-processing of data (see Section 4)
2. Setting up the input files and the scenario parameters (see Section 5)
3. Running Marxan (see Section 6)
4. Viewing and interpreting the results (see Section 7)

It is important to note that the successful use of Marxan will never involve a once-off, sequential application of these four steps. Instead, in any given project steps 2 to 4 should be repeated numerous times and exhaustively explored. Further, the results of each run should be used to refine the details of following runs. Because of this, it is important to be well organised and have an efficient file management protocol (see suggestions on file management in Section 5.2 and 7.2). Once the input files have been set up, it should be quite easy to modify the scenario, re-run Marxan and investigate the results.

A more detailed treatment of each step follows in subsequent chapters.

Box 7. Marxan workflow for a standard project

The diagram below illustrates a generalized Marxan workflow.



Input data: The workflow begins with data inputs. In order to work, Marxan requires certain types of input data (e.g., spatial data on conservation features and conservation targets) and this input dataset needs to be organized into specific input file types.

Marxan analysis: Marxan then runs its algorithm on the dataset to find solutions that meet targets at a minimal cost. You can set parameters (e.g., the feature penalty factor (FPF) and the boundary length modifier (BLM)), to influence how the algorithm interacts with the data. You can also set how many times Marxan is applied or "run", to produce different solutions. Therefore, 100 runs generate 100 different 'good' solutions.

Outputs: Two key outputs include the 'best' solution (the solution with the lowest score) and the selection frequency (the number of times each planning unit was selected across all solutions).

4 Pre-Processing of Data

4.1 Overview of Data Preparation

Data compilation, management and preparation of conservation features and cost data are typically the most time-consuming aspects of any Marxan analysis. These tasks do not actually involve the use of Marxan itself. Rather, they involve using a geographical information system (GIS) to:

- compile, manage, and assess available spatial datasets; and
- pre-process select datasets to convert data into a format that can subsequently be used to generate the Marxan input files (i.e., the files needed to run Marxan).

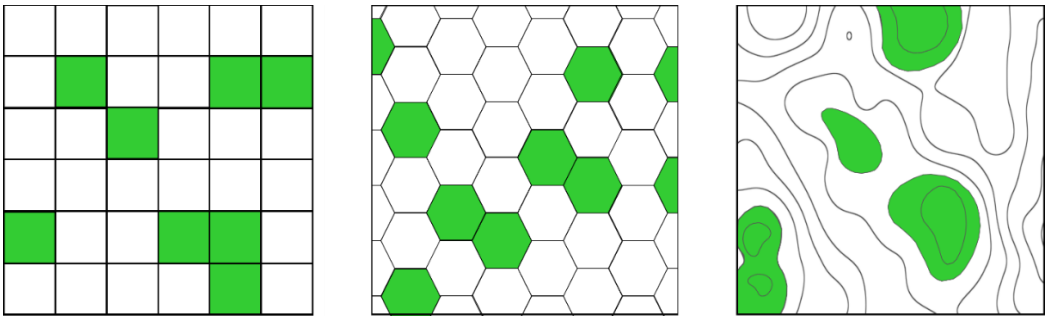
This section focuses on the latter and outlines two essential steps of pre-processing data. The first is to divide the planning region into planning units. The second is to account for the distribution of conservation features and cost/s across planning units. Both steps require some knowledge of a GIS software, such as ArcGIS or QGIS. However, we are working to overcome this prerequisite skill with new Marxan web-based applications (See Section 1.9). We encourage users to acquire the necessary GIS skills through external resources or trainings. This manual does not include instructions for using GIS software. Users should also consult the Marxan Good Practices Handbook for guidance on assessing, managing, and preparing datasets for Marxan exercises.

Box 8. Limited Availability and Quality of Data

The availability of quality spatial data will often limit what conservation features (and cost data) can be used. While it may be tempting to include all conservation features for which there is some data, users must be aware that weak or incomplete data can "drive" the analysis, given that the algorithm will gravitate towards data-rich areas. See the Marxan Good Practices Handbook for best practices for preventing this type of sampling bias.

4.2 Dividing the Planning Region into Planning Units

An essential first step for any Marxan exercise is to divide your planning region into a set of planning units. Planning units may be defined by overlaying your planning region with a grid of squares or lattice of hexagons (see Figure 1a and 1b). This is the most commonly used approach to planning unit design. However, in some instances, non-uniform units such as watersheds or sub-catchments, might be desirable for decision-making (see Figure 1c).



(a) Square PU (b) Hexagonal PUs (c) Irregular PUs

Figure 1. *Three possible types of planning units (PUs) that could be used in Marxan*

 It is important to use extreme care if you are using irregular shaped planning units, as the boundaries and size of the units will affect the selection of priorities.

The Marxan Good Practices Handbook provides guidance for determining the appropriate planning unit size and shape. In general, planning unit size and shape is informed by the scale of planning decisions (i.e., global, regional, national, or local), the resolution of datasets being used, the objective of the planning exercise, and the intended use of the outputs (e.g., general area prioritisation or specific plans for implementation).

The planning unit size should be at a scale appropriate for both the features you wish to capture and the scale at which the actions being prioritized for, are likely to be implemented. In general, they should be no finer in resolution than the data on features and no coarser than is realistic for management decisions.

Keep in mind that smaller planning units are not always ideal. Some features, such as wetlands or reefs, should be treated as larger, functional units. In such cases, it will make more sense to have planning units that are informed by natural ecological divisions, such as hydrological units. For implementation purposes, it may be easier to partition planning units by political/governmental divisions, such as cadastral parcels. For other uses, a uniform planning unit will provide more useful results.

There are some limits on the number of planning units that Marxan can handle. This is not, however, a fixed number as it depends also on the number of features you include and even the power of your computer. The current versions of Marxan (v. 2.43 and above) have been successful at processing larger numbers planning units (over 150,000 planning units) on computers with ample RAM memory. There are some cases where the decision space is constrained by the arrangement of its features that even with huge numbers of planning units, near-optimal results are still tractable. However, these situations are the exception. In general, when there are lots of possible network configurations, optimal solutions will be hard to find when using over 50,000 planning units.

4.3 Determining the Distribution of Features & Costs

Another essential step prior to using Marxan is to determine the distribution of conservation features and cost across the planning units. To do this will generally require some knowledge of a geographical information system (GIS) and geospatial data.

Spatial data on conservation features can typically be defined as one of three basic feature types: points, lines, and polygons (Figure 2).

- Point data exists when a feature is associated with a single location in space. Examples of point features include point locations of small locally or community managed protected areas, spawning aggregation locations, or cultural sites.
- Line data exists when a feature's location is defined by a string of spatial coordinates. Examples of line features include rivers, roads, shipping lanes, and pipelines.
- Polygon data exists when a feature is defined by a closed string of spatial coordinates. Examples of polygon features include species distributions, habitat distributions (e.g., seagrass beds and forest stands), protected areas, and fishing areas.
- Raster data, like vegetation maps, may be desirable to include in your analysis as conservation features. You will need to make the decision if it is best to extract values from raster datasets to each individual planning unit, or, convert rasters to vector files and treat them the same as above.

These data then need to be converted and organized into conservation features, assembled in a database or attribute table, and then organized into the Marxan input files. There are three tasks in this workflow (Figure 3).

1. The assembled data layers need to be geo-processed, typically through an intersection, with the planning unit grid. This calculates how much of each feature is located within each planning unit.
2. The cost surface also needs to be overlaid with the planning unit layer to determine the cost value of each planning unit (see Box 9 on cost data).
3. Once all of the data are assembled into one database or attribute table, they can then be used to generate the input files required to run Marxan.

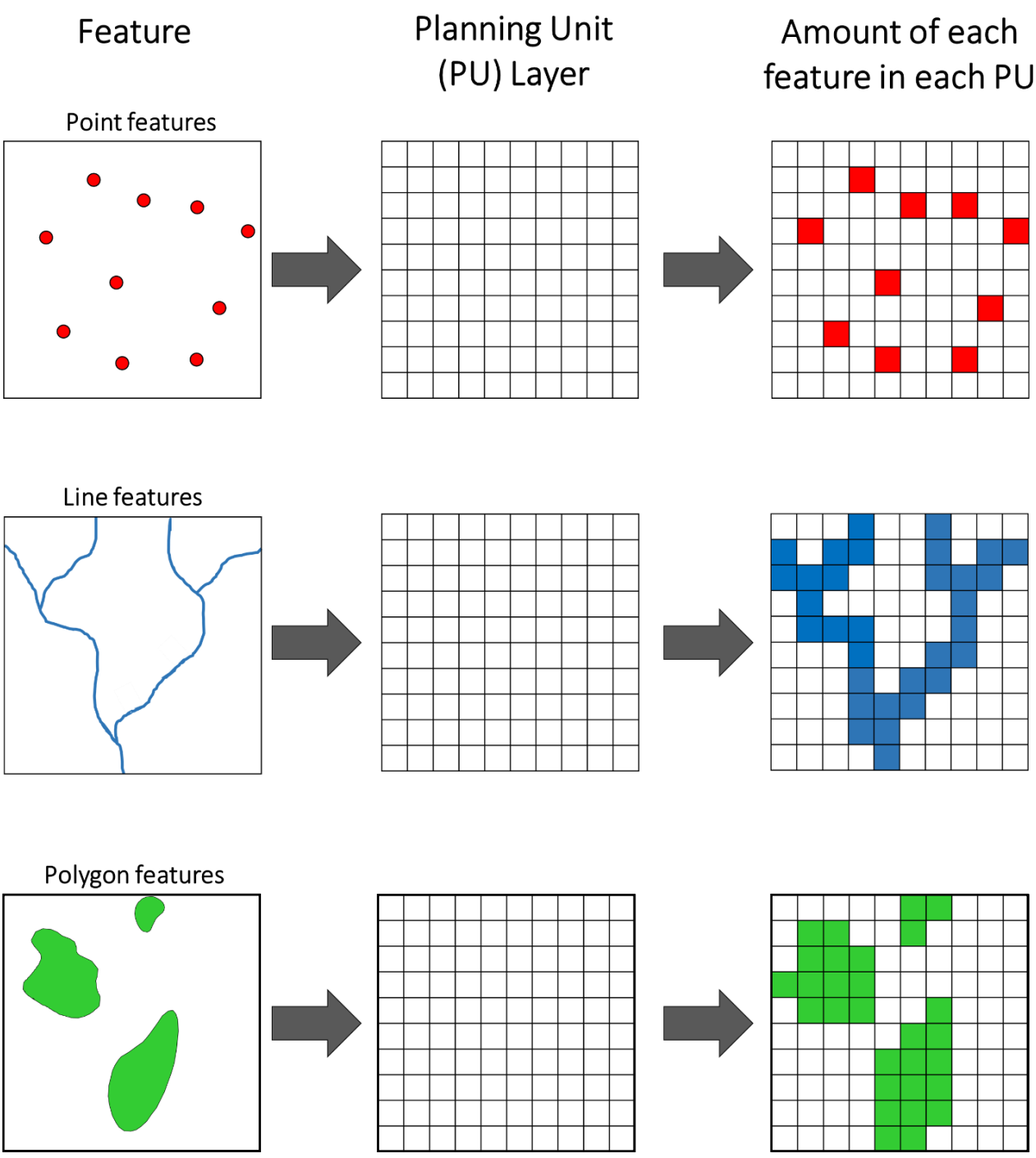
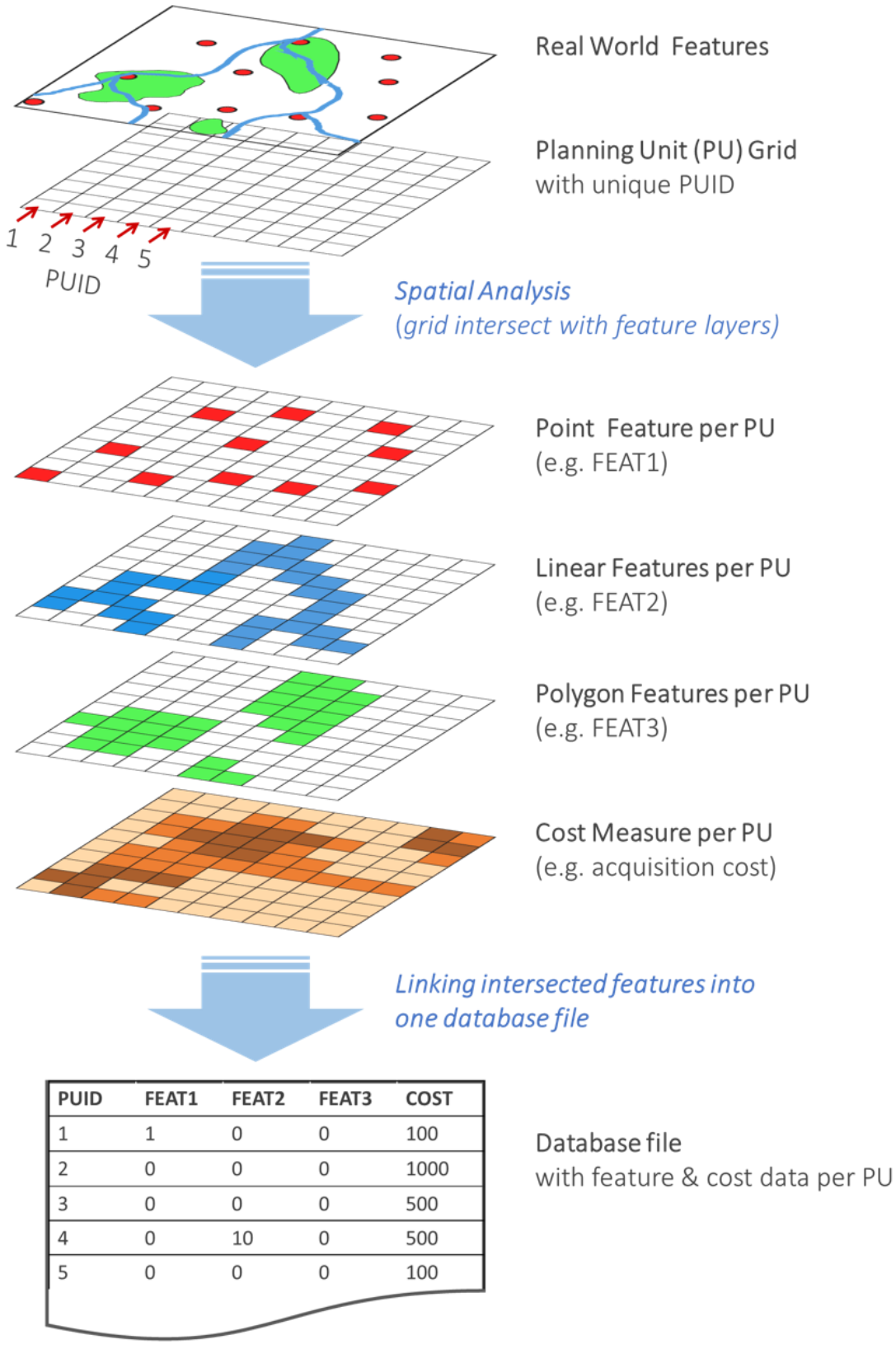


Figure 2. *Determining the distribution of point, line, and polygon features across planning unit (PU)*



PUID	FEAT1	FEAT2	FEAT3	COST
1	1	0	0	100
2	0	0	0	1000
3	0	0	0	500
4	0	10	0	500
5	0	0	0	100

Figure 3. *GIS workflow for determining the distribution of features and costs*

Box 9. Incorporating Cost Data in Marxan

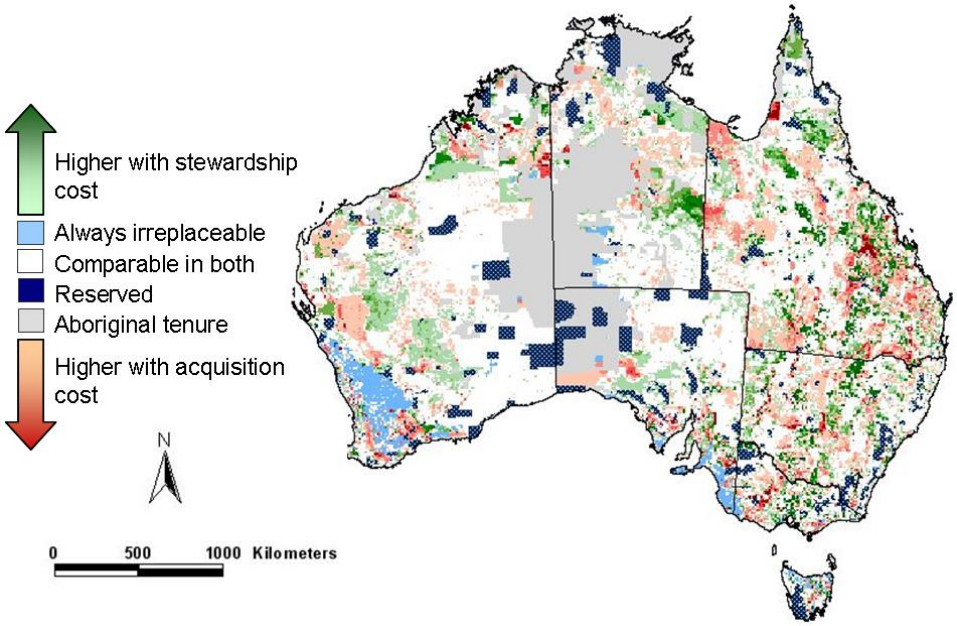
In conservation planning, cost data may reflect any variety of socioeconomic factors, which if minimized, might help the conservation plan be implemented more effectively and reduce conflicts with other uses. Following are some examples of different definitions of "cost" that have been considered in Marxan.

- *Cost equal to the planning unit area:* Many conservation practitioners have defined cost as the area of the planning unit to identify sites that meet conservation targets within the smallest possible area of land or sea. The assumption here is that by minimizing the overall area, the impacts of conservation areas on people will be reduced. When costs are flat (every planning unit has the same cost value), the priorities identified may be used a baseline scenario to explore how alternate costs, that vary in space across the planning region, affect priorities.
- *Forgone opportunity costs:* "Opportunity costs" are the estimates of foregone revenues or economic livelihoods from putting an area into protection (for example, the lost revenue incurred to a fisherman or fishing fleet if they can no longer fish).
- *Action costs*: Costs associated with conservation actions include acquisition costs (e.g., cost of buying or leasing land), management costs (e.g., enforcement and monitoring costs of protected areas), and transaction costs (e.g., costs associated with negotiating protection, such as the time and staff involved in stakeholder negotiations).
- *Proxy costs*: It is often difficult to estimate socio-economic costs due to data limitations, accessibility, and uncertainty. When relevant, high quality spatially explicit cost data are not available, sensible surrogates can be used to represent socioeconomic impacts. For example, coastal population density may be as a proxy of resource dependencies. The distance from ports or roads may be used as an estimate of accessibility (e.g. farther travel means higher costs).

Marxan can only consider one cost value at a time. Combining multiple costs into one cost is a key challenge of incorporating socioeconomic data in Marxan. In many cases, socioeconomic data are measured in different units (e.g., fishing intensity (boats/km^2), and aquaculture production (kg)), and revenues ($).

In this instance, we recommend ***applying each cost individually in a series of Marxan scenarios***. For example, Klein et al.((Klein et al., 2008) developed three scenarios, each with a different cost (recreational fisheries; commercial fisheries; and recreational and commercial fisheries combined). Using the results of this study, Ban and Klein (2009) demonstrate how these scenarios with different costs can inform the design of protected areas: 1) the results of each scenario can be evaluated to determine their costs to each

stakeholder group and to identify trade-offs; 2) statistical analyses (e.g., cluster analyses) can be used to assess the similarity of the results of scenarios; 3) a map showing the difference in the selection frequency between two scenarios is useful in comparing differences (see image below for an example taken from Carwardine et al. (2008).



Difference map shows how selection frequencies change when comparing two scenarios that use different costs.

Sometimes, it may be desirable to combine two costs to produce a single value. This can be done if your costs are in the same unit but vary in how important they are to the planning problem. We recommend the following approach to combining two costs into a single value:

Total cost = $(\alpha Cost_1) + (1 - \alpha Cost_2)$

where alpha (α) is a number between 0-1 that sets the weighing preference for the two costs. For example, an alpha = 0.5 means each cost should contribute equally to the total cost value. An alpha of 0.7 means you preference the total cost to be driven more by Cost 1, leaving a 0.3 weighting for Cost 2.

For guidance on how to prepare and incorporate cost data in Marxan, see the Marxan Good Practice Handbook. Additionally, see the review by Ban and Klein (2009) for alternative methods for combining multiple costs into one cost.

5 Input Files, Parameters & Variables

5.1 Overview of Input Files

To run Marxan, you need a set of input files (Table 2). These files contain all the data you wish to work with and the details of the conservation problem you want Marxan to solve. Four input files are required to run Marxan, as indicated in Table 2. The Boundary Length File is an optional input file.

Table 2. *Marxan input files and default names*

File	Default Name	Required
Input Parameter File	input.dat	Yes
Conservation Feature File	spec.dat	Yes
Planning Unit File	pu.dat	Yes
Planning Unit versus Conservation Feature File	puvspr.dat	Yes
Boundary Length File	bound.dat	No

The input files are summarized below.

- The **Input Parameter File** is used to set values for all the main parameters that control the way Marxan works. It is also used to tell Marxan where to find the input files containing your data and other variables and where to place the output files.
- The **Conservation Feature File** (also referred to as 'Species Feature File') contains information about each of the conservation features being considered, such as their name, targets and representation requirements, and the penalty that should be applied if these representation requirements are not met.
- The **Planning Unit File** contains information about the planning units themselves, such as ID number, cost, location and status.
- The **Planning Unit versus Conservation Feature File** (also known as 'Planning Unit versus Species Feature File') contains information on the distribution of conservation features in each of the planning units.
- The **Boundary Length File** contains information about the length or 'effective length' of shared boundaries between planning units. This file is necessary if you wish to use the Boundary Length Modifier to improve the compactness of reserve solutions, and while not required, is recommended.

Box 10. Options for Developing Input Files

There are many ways of developing the input files necessary for running Marxan. The choice of method will depend on your skills, available software and personal preference. Some software is free and others relatively expensive.

You will need, at minimum, a spreadsheet or Textpad type software to develop, read and manipulate the required input files. Windows operating systems contains the notepad text editor. There are many free text editors available on the internet that can be substituted. Spreadsheet programs such as Microsoft Excel and OpenOffice (a free open source alternative) are useful for applying formulas and easy manipulation of data.

Also, there are some useful supporting software tools that can be used to create the Marxan input files (see Section 2.3).

5.2 Input File Management

All Marxan input files use the .dat file extension. These files can be viewed in basic text editor programs such as Windows Notepad or TextPad. To generate a .dat file, simply add the suffix '.dat' after the file name when saving the file. Files can be comma or tab delineated.

When running Marxan, the executable 'Marxan.exe' should be located in the same folder as the input files for that project. Rather than continually moving files around, we recommend simply copying the Marxan executable to each folder containing a Marxan project.

Apart from the Input Parameter File (input.dat), all the input files should be stored in the same folder. This folder is generally called, 'input', but can have any name that you indicate. This folder should be nested within the same folder as the Marxan program executable, 'Marxan.exe'. The Input Parameter File, 'input.dat', must also be stored in the same place as the Marxan program executable. Figure 4 illustrates a standard Marxan folder or database.

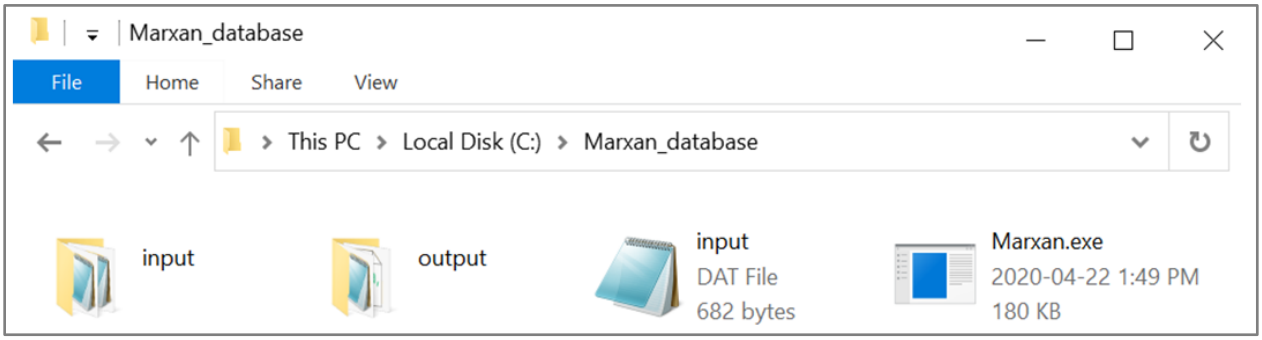


Figure 4. *Recommended set up for a Marxan folder/database*

 Throughout this manual, we refer to the different input files by their default names. You can give any name to the input files, provided the names match those given in the Input Parameter File, and providing you use the correct procedure for starting Marxan. Using consistent file names, however, helps simplify the organization of Marxan input files. The Input Parameter File should have the name, 'input.dat', as this is the name Marxan will look for unless it is told otherwise with a command line parameter. If you give a name other than 'input.dat' to this file (for example scenario1.dat), start Marxan with a command line parameter like "Marxan.exe scenario1.dat" to get it to recognise the Input Parameter File. If instead you prefer to use the default 'input.dat' as the name for the Input Parameter File, there is no need to use a command line parameter to get Marxan to recognise the file. You can start it like this: "Marxan.exe"

5.3 Required Files

This section describes the function, format, and associated variables of the four input files required to run Marxan:

- the input parameter file (input.dat)
- the conservation feature file (spec.dat)
- the planning unit file (pu.dat)
- the planning unit versus Conservation Feature file (puvspr.dat)

An example of each file is contained in the Marxan download package (i.e., a folder with input files is downloaded when you download Marxan).

 It is important that all the required files are included in order to run Marxan properly. Missing required files will cause Marxan to halt. For more information about the type of errors messaging this will cause go to Appendix A- Troubleshooting.

5.3.1 The Input Parameter File (input.dat)

Marxan is an extremely flexible program. The flipside to this, however, is that you must set certain parameters to appropriately deal with the particular problem you wish to solve. This is primarily accomplished through the Input Parameter File (input.dat). This file contains the principal parameters that control the way Marxan finds solutions (i.e. which algorithm(s) are used, what parameters contribute to the objective function). It is also used to tell Marxan where to find the required input files, whether you are using any optional input files, and what output files you want and where they should be saved.

Figure 5 is an example of the input.dat file, included in the Marxan download package. Note that this example does not include all possible variables that can be contained in the file.

```
Input file for Annealing program.

General Parameters
BLM 1
PROP 0.5
RANDSEED -1
NUMREPS 10

Annealing Parameters
NUMITNS 1000000
STARTTEMP -1
NUMTEMP 10000

Cost Threshold
COSTTHRESH  0.00000000000000E+0000
THRESHPEN1  1.40000000000000E+0001
THRESHPEN2  1.00000000000000E+0000

Input Files
INPUTDIR input
PUNAME pu.dat
SPECNAME spec.dat
PUVSPRNAME puvspr.dat
BOUNDNAME bound.dat

Save Files
SCENNAME output
SAVERUN 3
SAVEBEST 3
SAVESUMMARY 3
SAVESCEN 3
SAVETARGMET 3
SAVESUMSOLN 3
SAVEPENALTY 3
SAVELOG 2
OUTPUTDIR output

Program control.
RUNMODE 1
MISSLEVEL 1
ITIMPTYPE 0
HEURTYPE -1
CLUMPTYPE 0
VERBOSITY 3

SAVESOLUTIONSMATRIX 3
```

Figure 5. *An example of the Input Parameter File (input.dat)*

A complete list of variables and default values for the input.dat file is provided in Table 3. Each variable or group of variables are described in the subsections below. The numerous variables may seem a bit daunting at first, but many of these variables will almost never need to be modified. A few, however, may be changed for nearly every scenario run.

Table 3. *Variable names and default values for the Input Parameter File (input.dat)*

Category	Variable Name	Default Value	Description
General Parameter	BLM	1	Boundary Length Modifier
	PROP	0.5	Proportion of planning units in initial reserve system
	RANDSEED	-1	Random seed number
	BESTSCORE	0	Best score hint
	NUMREPS	10	Number of repeat runs (or solutions)
Annealing Parameters	NUMITNS	1000000	Number of iterations for annealing
	START TEMP	-1	Starting temperature for annealing
	COOLFAC	0	Cooling factor for annealing
	NUMTEMP	10000	Number of temperature decreases for annealing
Cost Threshold	COSTTHRESH	0	Cost threshold
	THRESHPEN1	0	Size of cost threshold penalty
	THRESHPEN2	0	Shape of cost threshold penalty
Input Files	INPUTDIR	input	Name of folder containing input data files
	SPECNAME	spec.dat	Name of Conservation Feature File
	PUNAME	pu.dat	Name of Planning Unit File
	PUVSPRNAME	puvspr.dat	Name of Planning Unit versus Conservation Feature File
	BOUNDNAME	bound.dat	Name of Boundary Length File
	BLOCKDEFNAME	blockdef.dat	Name of Block Definition File
Save Files	SCENNAME	output	Scenario name for the saved output file
	SAVERUN	3	Save each run
	SAVEBEST	3	Save the best run

	SAVESUMMARY	3	Save summary information
	SAVESCEN	3	Save scenario information
	SAVETARGMET	3	Save targets met information
	SAVESUMSOLN	3	Save summed solution information
	SAVEPENALTY	3	Save computed feature penalties
	SAVELOG	2	Save log files
	SAVESNAPSTEPS	0	Save snapshots each *n* steps
	SAVESNAPCHANGES	0	Save snapshots after every *n* change
	SAVESNAPFREQUENCY	0	Frequency of snapshots if they are being used
	SAVESOLUTIONSMATRIX	3	Save all runs in a single matrix
	OUTPUTDIR	output	Name of the folder in which to save output files
Program	RUNMODE	1	Run option
Control	MISSLEVEL	1	Species missing proportion
	ITIMPTYPE	0	Iterative improvement
	HEURTYPE	-1	Heuristic
	CLUMPTYPE	0	Clumping rule
	VERBOSITY	3	Screen option

There are two possible ways to create and modify the input.dat file: one, using one of the supporting software for Marxan (See Section 2.3); or two, using the input.dat file that comes bundled with the Marxan download package and making necessary changes to the file with a text file editor (e.g. Windows Notepad)[5].

There are no absolute best values for the parameters contained within the input.dat file. Although similar values may work well for different applications, you will need to determine the most appropriate parameter values for each project. This is best done in an iterative fashion in which the results are investigated, the parameters changed, the program run

[5] To adjust the parameters on the input.dat with a text editor, open the file using a text editor program (e.g., Windows Notepad or equivalent), change the relevant parameter values and resave the file.

again, and the new results compared with the old ones. In sub-sections below, we provide guidance on how to determine appropriate values for key parameters.

 Each variable is given as a single word in capital letters (Table 3). The value for that variable follows on the same line with just a single space between the variable name and value. Any lines that are either not valid variable names or not in capital letters will be ignored by Marxan, so it is possible to include comments or notes between variables in this file. The variables can occur in any order but Marxan will halt with an error if any are defined twice.

 Most of the variables in the input.dat file have default values that will be used if the variable is not defined. The exceptions to this are the variables that tell Marxan where to find the necessary input files, where to save the output files, and the variable, 'RUNMODE', which tells Marxan which method it should use to find the best reserve system (i.e. simulated annealing, heuristic, or both).

5.3.1.1 Boundary Length Modifier

Variable Name:	BLM
Required:	No
Description:	The variable, 'BLM' (Boundary Length Modifier), is used to determine how much emphasis should be placed on minimising the overall reserve system boundary length. Minimising this length will produce a more compact reserve system, which may be desirable for a variety of pragmatic reasons (see example in Box 11). Emphasising the importance of a compact network will mean that your targets are likely to be met in a smaller number of large reserves, generally resulting in on overall larger and more expensive reserve system. Thus, the BLM works counter to the other major goal of Marxan, to minimise the overall cost of the solution. BLM can be thought of as a relative sliding scale, ranging from cheaper fragmented solutions (low BLM) to a more compact expensive one (high BLM). Because this will have a large influence on the final solutions, some work is needed to ensure an appropriate value (or range of values) is found.

Getting Started:	The BLM should be either '0' or a positive number. It is permissible for the BLM to include decimal points (e.g. 0.1). Setting the BLM to '0' will remove boundary length from consideration altogether. There is no universally good value for the BLM, as it works in relation to the costs and geometry of the study region/planning units. With a small BLM, Marxan will concentrate on minimizing overall reserve cost and will only aim for compactness when little extra cost will be incurred. Alternatively, a large BLM will place a high emphasis on minimizing the boundary length, even if it means a more costly solution. The user must explore the effects of different BLM values to determine an appropriate BLM for the project's objectives. Although the 'correct' level of spatial compactness is subjective, Box 12 provides some tips. As a very rough guide, a good starting place for the BLM is to scale it such that the largest boundary between planning units becomes a similar order of magnitude to the most expensive planning unit. For instance, if your highest planning unit cost is 100 and your longest boundary is 1000, you may want to start the BLM at 0.1. Note that it is usually best to explore a range of values that are separated using a fixed multiplier (e.g., 0.04, 0.2, 1, 5, 25), where in this example, these values are each multiplied by 5. Typically, the values are increased exponentially or by orders of magnitude in order to sample a range of values and choose one that balances the order of magnitude of competing terms of the objective function. More information about the impact of spatial compactness and determining an efficient level of it can be found in Stewart and Possingham (2005). Supplemental information on the application of the BLM can be found in the Marxan Good Practices Handbook.

Box 11. The Boundary Length Modifier (BLM) Value

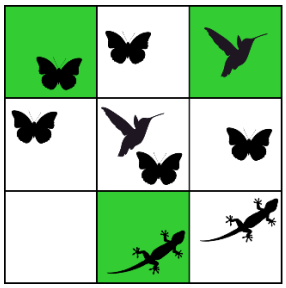
In Marxan's objective function, the boundary cost weighting is controlled by a multiplier called the 'Boundary Length Modifier' or BLM. The higher the BLM value, the more weight given to the boundary cost and the more compact the solutions of the algorithm.

$$\sum_{i}^{N_s} x_i\, c_i + b \sum_{i}^{N_s} \sum_{ii}^{N_s} x_i(1 - x_h) cv_{ih} + \sum_{j}^{N_f} FPF_j FR_j H(s) \left(\frac{s}{T_j}\right)$$

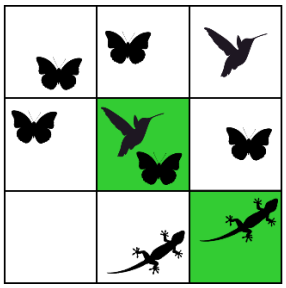
or

$$\sum_{PUs} \text{Cost} + BLM \sum_{PUs} Boudary + \sum_{\substack{Con \\ Value}} SPF \times Penalty$$

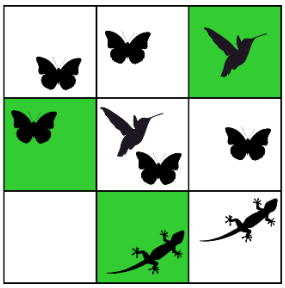
A BLM value of '0' will tend to result in solutions (i.e., combinations of planning units) that are more fragmented. For example:



Solution 1

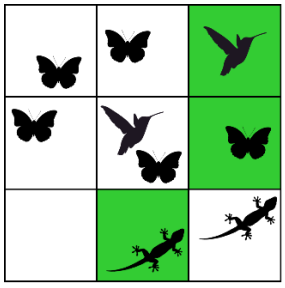


Solution 2

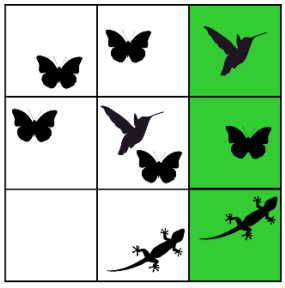


Solution 3

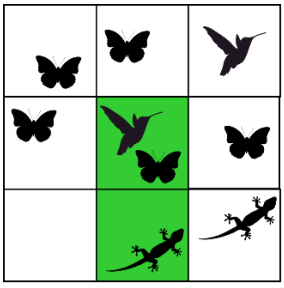
A higher BLM value (BLM > 0) will result in more clumped solutions. For example:



Solution 1



Solution 2

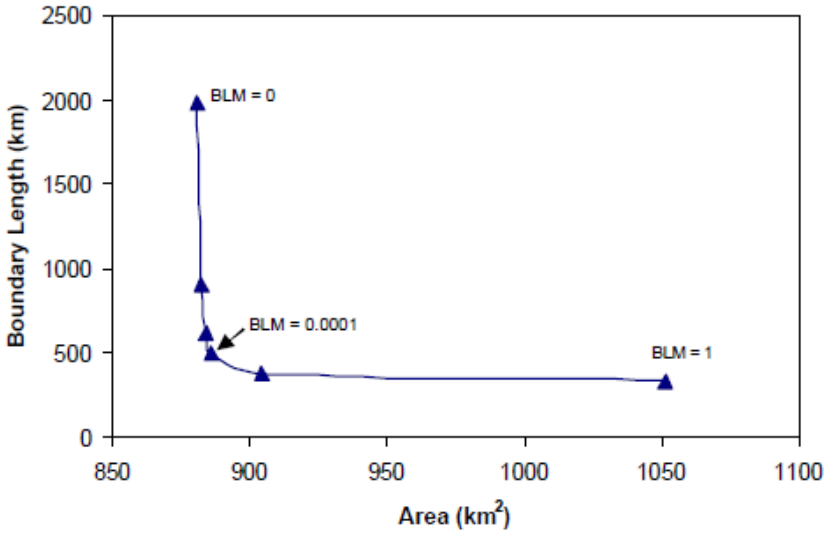


Solution 3

Box 12. Setting the Boundary Length Modifier (BLM)

The following methods for determining an efficient Boundary Length Modifier (BLM) is taken from Steward and Possingham (2005).

- Keeping all the parameters the same, repeat Marxan analysis using a series of different values for the BLM, e.g. 0, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000, etc.
- In a spreadsheet, record the BLM for each scenario, the average value of the total reserve boundary length of all solutions, and the average cost of all solutions. If the selected reserve area is more important than cost, or is being used as a surrogate for cost, record the average reserve area of solutions.
- Plot the average total boundary length versus total cost/area for all the different BLM values. The graph below illustrates the trade-off between reserve system boundary length and the total area of the reserve system.



(Adapted from Steward and Possingham, 2005)

The diminishing returns of increasing the BLM are clearly visible in the graph. In this case, a BLM of 0.0001 is probably sensible. What you are looking for is the turning point at which the increase in reserve cost or area becomes large relative to the corresponding reduction in system boundary length. This represents a good starting BLM. Solutions should, however, always be visually inspected before settling on a final BLM in order to ensure that the reserve system is at an appropriate level of compactness. Always check to make sure that you are meeting your targets after you settle on a BLM that reflects the level of clumping you select.

5.3.1.2 Starting Prop

Variable Name:	PROP
Required:	No
Description:	When Marxan starts a run, it must generate an initial reserve system. This variable defines the proportion of planning units to be included in the initial reserve system at the start of each run.
Getting Started:	The variable 'PROP' must be a number between 0 and 1. If zero is chosen then no planning units will be included in the initial reserve, a value of 1 means all planning units will be included, and a value of 0.5 means 50% of planning units will be randomly included. In practice, the setting has no effect on the operation of simulated annealing, provided a sufficient number of iterations is used. This will only be applied to those planning units whose status does not lock them in or out of solutions (see Section 5.3 under "Planning Unit Status").

5.3.1.3 Random Seed

Variable Name:	RANDSEED
Required:	No
Description:	Do not worry too much about this variable. It controls whether the same 'random' selection of planning units is included in the initial reserve system each run. Using a constant positive integer for this variable will make Marxan use the same random seed each time it is run.
Getting Started:	Except for debugging purposes, it should be set to '-1' in the input.dat file.

5.3.1.4 Best Score Speedup

Variable Name:	BESTSCORE
Required:	No

Description:	This variable tells Marxan not to keep track of the best score until it reaches a specified minimum level. It was intended to be a time saving measure but is seldom required.
Getting Started:	It should always be set to '-1'.

5.3.1.5 Repeat Runs

Variable Name:	NUMREPS
Required:	Yes
Description:	The number of repeat runs you want Marxan to perform; effectively, the number of solutions to the reserve problem you want Marxan to generate. Each new run is independent of the previous one, but they will all use the same parameter and variable values. The frequency with which planning units are selected in multiple runs, gives an indication of the importance of that planning unit for efficiently meeting your reserve targets (see Section 7).
Getting Started:	When running a new scenario for the first time it is always advisable to begin with a very small number of runs (e.g. 10) so you can check the program is performing as desired (i.e. the solutions are meeting the required targets) without having to wait a long time. In order to get an idea of selection frequency, however, you will generally need to do many runs. **100 runs** is the standard best practice, as it is an intuitive value from which to calculate selection frequency. Adding more runs will sample more of the solution space but will of course increase the processing time. The final number you decide on must be a balance between the time taken and the information gained. The Marxan Good Practices Handbook provides some useful suggestions about determining the optimal number of runs.

5.3.1.6 Simulated Annealing Parameters

Variables:	NUMITNS STARTTEMP

	COOLFAC NUMTEMP
Required:	Yes (when using Simulated Annealing)
Description:	These four variables control the way the Simulated Annealing algorithm proceeds. They will come into play when Simulated Annealing is chosen in the 'RUNMODE' (see Section 5.1.3 under "RUNMODE").
Getting Started:	In practice, you will rarely need to adjust most of these. The variables, 'STARTTEMP' and 'COOLFAC', will be set appropriately for you when creating the input file with supporting software tools of your choice or when downloading the file bundled in the Marxan download package. For almost all applications, it is quite reasonable to leave the number of temperature decreases (variable, 'NUMTEMP') set at '10,000'. The number of iterations set (variable, 'NUMITNS') has a substantial bearing on how long each run takes. In general, the number of iterations determines how close Marxan gets to the optimal solution (or at least a very good solution). The number should start high (e.g. 1,000,000) and then be increased (e.g. 10 million or more is commonly applied on large scale datasets) until there is no substantial improvement in score as iterations continues to increase. At some point, the extra time required by a higher number of iterations will be better spent doing more runs than spending a long time on each run. Choose an acceptable trade-off between solution efficiency (score, or number of planning units) and execution time (number of iterations).

5.3.1.7 Cost Threshold

Variable Name:	COSTTHRESH THRESHPEN1 THRESHPEN2
Required:	No
Description:	These variables can be included if you want Marxan to find reserve solutions below a total cost. As discussed in the introduction, Marxan is

	designed to solve a 'minimum set' problem, its goal being to meet all our conservation targets for the least cost. Another class of conservation problem is known as the 'maximum coverage' problem where the goal is to achieve the best conservation outcomes for a given fixed budget. In many cases, this is more representative of how conservation actions operate. Although including a cost threshold does not make Marxan solve the strict 'maximum coverage' problem, it is comparable and can be used in cases where you have conservation targets you hope to meet and cannot exceed a predetermined budget. The actual way this cost threshold is applied within the algorithm is described in detail in Appendix B-1.5.
Getting Started:	Setting this variable to '0' in the 'input.dat' file will disable it. Be careful using this function as it can affect Marxan's ability to find efficient solutions. When using this variable, some users have reported, that the total cost of resulting reserve networks exceeds the cost threshold specified. It is apparent from the output tables if this problem manifests. Marxan is currently being redesigned for improved reliability with regard to this variable. If you obtain resulting reserve networks that exceed the threshold specified, disregard these results.

5.3.1.8 Input Files

Variable Name:	INPUTDIR SPECNAME PUNAME PUVSPRNAME BOUNDNAME BLOCKDEFNAME
Required:	Yes

Description and Getting Started:	The 'INPUTDIR' is used to tell Marxan the name of the folder (called directory or DIR in Marxan) containing the input files. The naming and location protocol for this folder are discussed in Section 5.2. These remaining files are reasonably self-explanatory and their protocols for naming and storage have been discussed previously (see Section 5.2 and Table 3). It is critical that the INPUTDIR is done correctly or Marxan will not run. It is also important to verify that the files are in the correct folder before running Marxan.

5.3.1.9 Save Files

Variable Name:	SCENNAME SAVERUN SAVEBEST SAVESUMMARY SAVESCEN SAVETARGETMET SAVESUMSOLN SAVEPENALTY SAVELOG SAVESNAPSTEPS SAVESNAPCHANGES SAVESNAPFREQUENCY SAVESOLUTIONSMATRIX OUTPUTDIR
Required:	Yes
Description and Getting Started:	The variable, 'SCENNAME' is the name you wish Marxan to append to all output files it saves (e.g. 'scenario1_ssoln.dat' would be the name given the summed solution output). The name should be something you can use to identify the scenario that generated the outputs. The following variables (SAVERUN, SAVEBEST, SAVESUM, SAVESCEN, SAVETARGETMET, SAVESUMSOLN, SAVEPENALTY,

	SAVESOLUTIONSMATRIX, SAVELOG) can be saved in a number of formats: Use code '0' to not save • Use code '1' to save files as .dat • Use code '2' to save them as .txt • Use code '3' to save them as .csv The different outputs and their uses are all described in detail in Section 7.4. If either SAVESNAPSTEPS or SAVESNAPCHANGES are selected, then a SAVESNAPFREQUENCY value must also be specified. This is the predetermined number of either system iterations (SAVESNAPSTEPS) or system changes (SAVESNAPCHANGES) at which the solution progress of the optimisation procedure is saved. Saving snapshots can create enormous amounts of output files which swamp your output folder and drastically slow down the running of Marxan. They are for advanced diagnoses and should only be used by expert users. If you use them, make sure the snap frequency is large enough so that you are not left with tens of thousands of output files. The actual number you choose will depend on how many iterations you are using (variable, 'NUMITNS'). For 1 million iterations, a snap frequency of 100,000 will give 10 output files. 'OUTPUTDIR' is used to tell Marxan the name of the folder (called directory or DIR in Marxan) where it should save the output files. The naming and location protocols for this folder are discussed in Section 7.2. Like the input folder, it is critical that the OUTPUTDIR is done correctly or Marxan will not run.

5.3.1.10 Run Options

Variable Name:	RUNMODE
Required:	Yes

Description:	This is an essential variable that defines the method Marxan will use to locate good reserve solutions. As discussed in the introduction, the real strength of Marxan lies in its use of Simulated Annealing to find solutions to the reserve selection problem. However, Marxan is also capable of using simpler, but more rapid, methods to locate potential solutions, such as heuristic rules and iterative improvement (see Appendix B-2 for more details on these methods). Because heuristic rules can be applied extremely quickly and produce reasonable results, they are included for use on extremely large data sets. Modern computers are now so powerful that heuristics are less necessary as a time saving device, although they are still useful as research tools. Running Iterative Improvement on its own gives very poor solutions. As well as using any of these three methods on their own, Marxan can also use them in concert with each. If more than one are selected they will be applied in the following order: Simulated Annealing, Heuristic, Iterative Improvement. This means that there are seven different run options: 0. Apply Simulated Annealing followed by a Heuristic 1. Apply Simulated Annealing followed by Iterative Improvement 2. Apply Simulated Annealing followed by a Heuristic, followed by Iterative Improvement 3. Use only a Heuristic 4. Use only Iterative Improvement 5. Use a Heuristic followed by Iterative Improvement 6. Use only Simulated Annealing
Getting Started:	Of these combinations, the most useful is Simulated Annealing followed only by Iterative Improvement (variable value, '1'). This is because Simulated Annealing searches the solution space effectively, and the Iterative Improvement then ensures that the solution represents the best option in the immediate area of the decision space (known as a 'local minimum'). For most applications, this will be the best and you will rarely need to change it.

5.3.1.11 Species Missing Proportion

Variable Name:	MISSLEVEL
Required:	No
Description:	This is the proportion of the target a conservation feature must reach in order for it to be reported as met. There are situations where Marxan can get extremely close to the target (e.g. 99% of the desired level) without actually meeting the target. You can specify a level for which you are pragmatically satisfied that the amount of representation is close enough to the target to report it as met.
Getting Started:	This value should always be high, i.e. greater than or equal to '0.95'. If you are setting it lower than '0.95', you should probably think about changing your targets. As a guide, it is often useful to run Marxan with the 'MISSLEVEL' set at '1' and then re-run with it set at a slightly lower value and see if there is much of a difference in system cost. Setting this variable does not change the way the Marxan algorithm works, it merely changes the way target achievement is reported in screen and file output.

5.3.1.12 Iterative Improvement

Variable Name:	ITIMPTYPE
Required:	No
Description:	If Iterative Improvement is being used to help find solutions, this variable defines what type of Iterative Improvement will be applied. There are four different options, details of which can be found in Appendix B-2.2: 0. Normal Iterative Improvement 1. Two Step Iterative Improvement 2. 'Swap' Iterative Improvement 3. Normal Improvement followed by Two Step Iterative Improvement

Getting Started:	The default for this variable is '1' (Two Step Iterative Improvement), and for most scenarios this will be fine.

5.3.1.13 Heuristic

Variable Name:	HEURTYPE
Required:	No
Description:	If you are using an optional heuristic to find reserve solutions, this variable defines what type of heuristic algorithm will be applied. Details of the different Heuristics listed below are given in Appendix B-2. 0. Richness 1. Greedy 2. Max Rarity 3. Best Rarity 4. Average Rarity 5. Sum Rarity 6. Product Irreplaceability 7. Summation Irreplaceability
Getting Started:	We recommend that beginners use simulated annealing initially.

5.3.1.14 Clumping Rule

Variable Name:	CLUMPTYPE
Required:	No
Description:	This variable is useful if some conservation features have a minimum clump size set (target2, see Section 5.1.3 under "Minimum Clump Size"). It tells Marxan if occurrences smaller than the minimum clump size should contribute towards the overall target, and if so, how. Be aware that this will slow down Marxan by an order of magnitude. When using this variable, some users have reported resulting reserve configurations do not meet the clumping requirements specified. It is apparent from the output tables if this problem manifests, and we recommend you disregard any results that do not meet your clumping requirements.

Getting Started:	There are three options for this variable: 0. Partial clumps do not count– Clumps smaller than the target score nothing 1. Partial clumps count half– Clumps smaller than the target score half their amount 2. Graduated penalty– Score is proportional to the size of the clump

5.3.1.15 Screen Output

Variable Name:	VERBOSITY
Required:	Yes
Description:	This variable controls how much information Marxan displays on the screen while it is running. Users can specify how much information Marxan prints to the screen (the verbosity). There are four different options for screen: 0. Silent Running – Only the title of the program is displayed. 1. Results Only – Marxan will display which run it is up to, the basic results of each run and the total run time. 2. General Progress – In addition to the information about each run, Marxan will display information on the data that has been read in as well as details on any conservation features whose targets and requirements are such that they cannot be adequately reserved in the system. 3. Detailed Progress – Shows exactly where the program is up to and gives the value of the system each time the temperature changes.
Getting Started:	The default for this variable is 'General Progress' and in most cases this will be the best choice. Printing results to the screen does not increase Marxan's run time substantially unless 'Detailed Progress' is used. It is generally worthwhile to use at least 'Results Only' so that you have some idea of how many runs have been completed. 'Detailed Progress' is useful for seeing how the process of annealing works and can also help identify problems Marxan runs (e.g. if the

	numbers do not change and it is "stalled"). For this reason, some users always use this setting, to visually check that the program appears to be running OK. Only apply 'Silent Running' if you are confident in Marxan's execution and you are saving all necessary outputs.

5.3.2 The Conservation Feature File (spec.dat)

The Conservation Feature File (spec.dat) contains information about each of the conservation features being considered, such as their name, target amount, and the penalty if the target is not met. The file can contain up to nine variables (Table 4), although not all of these are required. When included, each of these variables is presented in a column with the name of that variable as the column header (see example in Figure 6).

Table 4. *Variable of the Conservation Feature File (spec.dat)*

Variable Name	Required	Description
id	Yes	A unique numerical identifier for each conservation feature
target*	Not required if prop is used	The target amount (in unit of puvfeat.dat file) of the feature
prop*	Not required if target is used	The proportion of the total amount of the feature which must be included in the solution
spf	No	The feature penalty factor (former species penalty factor)
target2	No	Minimum clump size for the representation of conservation features in the reserve system
targetocc	No	Minimum number of occurrences of a conservation feature required in a reserve system
name	No	Name of each conservation feature
sepnum	No	Number of mutually separated occurrences of a feature required in the reserve system
sepdistance	No	Used in conjunction with 'sepnum' (above), this variable specifies the minimum distance at which planning units holding a conservation feature are considered to be separate

* Conservation targets for features can be set as either an amount in the 'target' or as a proportion of the total amount in the 'prop'.

id	prop	spf
1	0.2	10
2	0.2	10
3	0.2	10

Figure 6. *An example of the Conservation Feature File (spec.dat)*

 This file does not contain information on the distribution of conservation features across planning units. This information is held in the Planning Unit versus Conservation Feature File.

Given the default name 'spec.dat', the file is sometimes referred to as the Species Feature File, although features included in this file will often represent surrogates such as habitat type rather than actual species. Further, features are not restricted to representations of biological attributes. They can represent other valued features like spiritual sites in need of preservation.

Targets for conservation features can be set in the spec.dat file as either an targeted amount using the 'target' variable (e.g., a value of '10' can be set in the 'target' field to target at least 10 occurrences of a given species), or as a proportion of the total amount using the 'prop' variable (e.g., a value of '0.2' can be set in the 'prop' field to include at least 20% of the total distribution of a given species).

 It is essential that the header names are exact. All letters must be in lower case. For all variables except 'id' and 'name', the default value is '0'. If data are missing from these variables Marxan will still be able to run, but the conservation features will take on the default values for missing attributes.

5.3.2.1 Conservation Feature ID

Variable Name:	*id*
Required:	Yes
Description:	A unique numerical identifier for each conservation feature. Be careful not to duplicate id numbers as Marxan will ignore all but the last one.
Getting Started:	It is useful to establish a logical system of numbering for features.

5.3.2.2 Conservation Target

Variable Name:	'*target*' or '*prop*'
Required:	Yes (either 'target' or 'prop' is required; do not include both)
Description:	Targets for conservation features can either be set as an amount in the 'target' variable or as the proportion of the total amount in the 'prop' variable. It makes no sense to set both, and if both are set then the variable 'prop' will take precedence and the variable 'target' will be ignored. The variable **'target'** can be used to set a target amount of each conservation feature to be included in the solutions. The values represent constraints on potential solutions to the reserve selection problem. That is, for a reserve solution to be feasible it must include at least this amount of each feature. The target value is expressed in the same units used to define the amount of each feature in each planning unit, contained in the Planning Unit versus Conservation Feature File (see Section 5.3.4). However, units from different conservation features can vary (e.g. hectares of habitat for one feature and number of occurrences for another, nests for a third and length of stream for a fourth). The variable **'prop'** is short for proportion and can be used to set the proportion (i.e. percentage) of a conservation feature to be included in the reserve system. For instance, a prop value of '0.2' can be used to capture at least 20% of the total area coverage of a given habitat type.
Getting Started:	Targets are user defined. The selection of appropriate conservation feature targets may reflect goals for representation in protected area networks set out in either legislation or convention (e.g. conserve at least 20% of each habitat type in reserve systems). Targets do not, however, have to be uniform values for all features. They may instead reflect the perceived importance of conservation for that feature, for instance you may wish that rarer or more threatened conservation features have higher targets than very common ones. Whatever the chosen targets, it is important that they are well justified as they will have an enormous bearing on the character of potential reserve systems. The higher the target the fewer the number of different

	possible solutions Marxan will be able to find. This is discussed further in the Marxan Good Practices Handbook. Targets set in the variable '**target**' can take any value from '0' to the total sum of that feature found in all planning units. You must be careful not to set a higher target than can possibly be achieved given the occurrence of a feature in the planning units as these targets will not be achievable. Targets set in the variable **'prop'** must be a value between '0' and '1'. For instance, if 'prop' for a type of feature is set to 0.2, then Marxan will set the target for all features within that type at 20% of the total abundance, based on the data in the Planning Unit versus Conservation Feature File (see Section 5.3.4). Total abundance is the sum total of the amount found in all planning units, including those that may be locked either in or out of possible reserve solutions. The proportion is based on the total amount defined in the Planning Unit versus Conservation Feature File. If some pre-processing of the data has occurred, for instance to remove small or fragmented occurrences of a feature, then this may not be the same as indicated in the original data set.

5.3.2.3 Feature Penalty Factor

Variable Name:	*spf*
Required:	Yes
Description:	The letters 'spf' stands for Species Penalty Factor. This variable is more correctly referred to as the Feature Penalty Factor or 'fpf'. The penalty factor is a multiplier that determines the size of the penalty that will be added to the objective function if the target for a conservation feature is not met in the current reserve scenario (Box 13). The higher the value, the greater the relative penalty, and the more emphasis Marxan will place on ensuring that feature's target is met. The FPF thus serves as a way of distinguishing the relative importance of different conservation features. Features of high conservation value, for example highly threatened features or those of significant social or economic importance, should have higher FPF values than less important features. This signifies that you are less willing to

	compromise their representation in the reserve system. See Appendix B-1.3 for details of how this penalty is calculated and applied Choosing a suitable value for this variable is essential to achieving good solutions in Marxan. If it is too low, the representation of conservation features may fall short of the targets. If it is too high, Marxan's ability to find good solutions will be impaired (i.e. it will sacrifice other system properties such as lower cost and greater compactness in an effort to fully meet the conservation feature targets).
Getting Started:	It will often require some experimentation to determine appropriate FPFs. This should be done in an iterative fashion (see Box 14). Even if all your targets are being met, always try lower values. By trying to achieve the lowest SPF that produces satisfactory solutions, Marxan has the greatest flexibility to find good solutions. In general, unless you have some a priori reason to weight the inclusion of features in your reserve system, you should start all features with the same FPF. If however, the targets for one or two features are consistently being missed even when all other features are adequately represented, it may be appropriate to raise the FPF for these features. Once again, see the Marxan Good Practices Handbook for more detail on setting and calibrating FPFs.

Box 13. The Feature Penalty Factor (FPF)

In Marxan's objective function, the penalty weighting is controlled by a multiplier called the 'Feature Penalty Factor' or FPF (also commonly referred to as the 'Species Penalty Factor' or SPF).

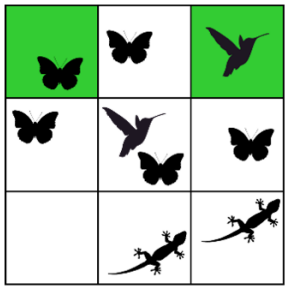
$$\sum_{i}^{N_s} x_i\, c_i + b \sum_{i}^{N_s} \sum_{ii}^{N_s} x_i(1 - x_h) cv_{ih} + \sum_{j}^{N_f} FPF_j FR_j H(s) \left(\frac{s}{T_j}\right)$$

or

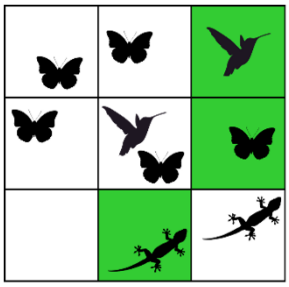
$$\sum_{PUs} \text{Cost} + BLM \sum_{PUs} Boudary + \sum_{\substack{Con \\ Value}} FPF \times Penalty$$

The higher the FPF value, the more Marxan will prioritize meeting the conservation target even if this comes at a higher planning unit cost and boundary cost. The FPF can be set to the same value for all feature targets or adjusted individually to ensure features that might be difficult to meet targets for, can have their targets met. This might occur if you have a small species distribution that happens to co-occur in a high cost location.

The following example compares solutions derived from two Marxan runs that are identical in every respect apart from their FPF values. The first was run with a default FPF value of '1' for all features, the second with an FPF value of '10'.



The Marxan run with an FPF of '1' resulted in several solutions, like the one displayed on the left, that did not adequately achieve all of the conservation targets, which were set to include at least one occurrence of each species in a reserve system (e.g., this solution only represented 2 of 3 species in the reserve system).

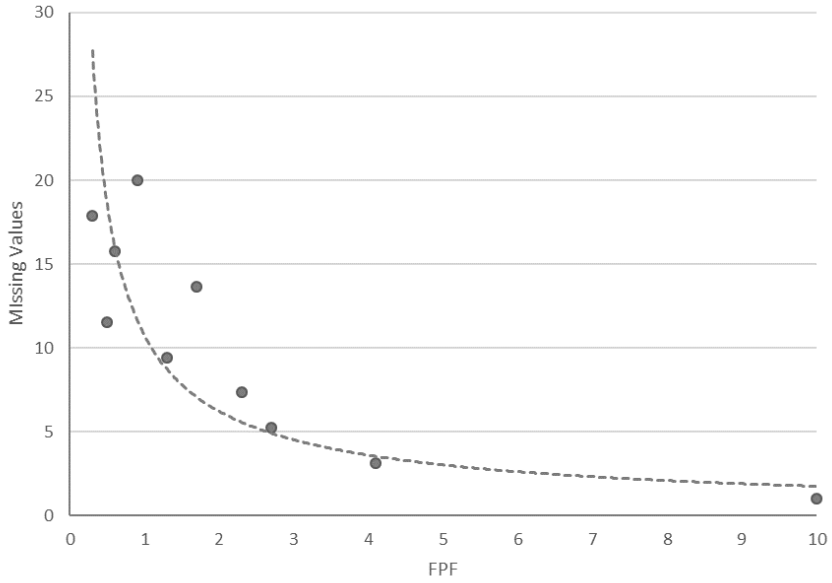


The Marxan run with a higher FPF value (FPF = 10) derived a solution that meets all the conservation targets.

Box 14. Calibrating Feature Penalty Factor (FPF)

One simple approach to calibrating FPF values is to set FPFs for all conservation features the same and iteratively adjust them until your solutions meet all conservation targets. You can do this manually through visual inspections of outputs, or you can be more structured by plotting the results of the number of missing values (e.g. targets missed) against the FPF values used.

First, select an arbitrary value for FPFs. If not all protection targets are being met in most/all solutions, try increasing FPF values (perhaps by a factor in the range of two to ten times greater initially). If all protection targets are being met, try decreasing FPF values iteratively until targets are no longer being met — then increase FPF values slightly. If conservation targets cannot be met despite setting a really high FPF, it is appropriate to revaluate the targets and data, as it might be impossible to achieve the target. Know your data well. A uniform value for FPF can be successful if all conservation features are generally similar in targets, abundance and spatial distribution. When feature abundances or distributions differ substantially, good practice includes at least some individual adjustment of FPF values. For more guidance on other techniques to calibrate the FPF consult the Marxan Good Practices Handbook. Also, the majority of Marxan Supporting Software included in Section 2.3 offer different methods to facilitate FPF calibration.



Calibration plot for FPF: Missing values (average number of features that missed their target for 100 solutions) versus the FPF value applied.

5.3.2.4 Minimum Clump Size

Variable Name:	*target2*
Required:	No
Description:	This variable specifies a minimum clump size for the representation of conservation features in the reserve system. If the amount of a conservation feature found in a clump is less than this value, then it does not count towards meeting the conservation target (the variable – 'target') for that feature. This is useful in cases where small or isolated patches or populations are of lower conservation value than larger, well connected patches or populations.
Getting Started:	It is best, when getting started, to not use this variable, and see how the features clump without it. If then, some particular features require further clumping, this variable can be applied. As with the conservation target, the value of 'target2' must be in the same units used to define the amount of each feature in each Planning Unit, contained in the Planning Unit versus Conservation Feature File (see Section 5.3.4). For instance, if you have included data on the area of different habitat types within planning units, then 'target2' specifies a minimum area (which may in fact be met in a single planning unit). In the case of presence absence data then the 'target2' value indicates the number of occurrences in contiguous planning units that must occur before the clump contributes to meeting targets (which also may be met in a single planning unit). Care must be taken when setting the minimum clump size as targets for some features will have little choice but to be met in small, isolated occurrences. When using this variable, some users have reported that the resulting reserve configurations do not meet the clumping requirements specified. It is apparent from the output tables if this problem manifests, and we recommend you disregard any results that do not meet your clumping requirements.

5.3.2.5 Target for Feature Occurrence

Variable Name:	*targetocc*
Required:	No
Description:	This variable specifies the minimum number of occurrences of a conservation feature required in a reserve system. This value can be used in situations where even though your conservation target may be met in one planning unit, you would like it to be represented in a greater number of planning units, possibly for risk spreading.
Getting Started:	This is a rather specialised feature that is only sometimes used. Unlike 'target' and 'target2', the value of 'targetocc' is not related to the units used to describe the occurrence of conservation features, it is simply the number of planning units the feature must occur in for a viable reserve solution. This variable can be used in conjunction with or instead of 'target'.

5.3.2.6 Conservation Feature Name

Variable Name:	*name*
Required:	No
Description:	The name of each conservation feature (e.g. cloud forest). This variable can include spaces, letters and numbers, but keep names under 10 characters if possible. Avoid using special characters in the name field, such as '_', '-' or '/'.

5.3.2.7 Target for Separation Feature Occurrences

Variable Name:	*sepnum*
Required:	No
Description:	The number of mutually separated occurrences of a feature required in the reserve system. This is similar to 'targetocc', except that if you wish

	to include multiple feature occurrences for the purpose of risk spreading then you may desire that the planning units holding these occurrences are not adjacent to each other. Where a minimum clump size has been set using 'target2', the variable 'sepnum' refers to the required number of mutually separated occurrences in valid clumps.
Getting Started:	This is an advanced feature addressing replication. Marxan should first be run without it, and then later runs can be done with it applied. This feature slows down Marxan by an order of magnitude. When using this variable, some users have reported that the resulting reserve configurations do not meet the clumping requirements specified. It is apparent from the output tables if this problem manifests, and we recommend you disregard any results that do not meet your clumping requirements.

5.3.2.8 Minimum Separation Distance

Variable Name:	*sepdistance*
Required:	No
Description:	Used in conjunction with 'sepnum' (above), this variable specifies the minimum distance at which planning units holding a conservation feature are considered to be separate. This may be useful in situations where multiple occurrences are desired because of the threat of large-scale events such as hurricanes or fires. For example, if hurricanes typically damage habitat over a known distance we may wish that two occurrences of the same feature are separated by a greater distance. Separation distance may also relate to the dispersal capacity of invasive species.
Getting Started:	In order to use this variable, the geographic location of each planning unit must be specified in the Planning Unit File (see Section 5.3.3). This feature can slow down Marxan considerably. When using this variable, some users have reported resulting reserve configurations do not meet the clumping requirements specified. It is apparent from the

	output tables if this problem manifests, and we recommend you disregard any results that do not meet your clumping requirements.

5.3.3 The Planning Unit File (pu.dat)

The Planning Unit File (pu.dat) contains all the information related to planning units, except for the distribution of conservation features across planning units (which is held in the Planning Unit versus Conservation Feature File). The pu.dat file can contain up to five variables (Table 5), although only one of these ('id') is required.

Table 5. *Variable names and requirements for Planning Unit Fie (pu.dat)*

Variable Name	Required	Description
id	Yes	A unique numerical identifier for each planning unit (PU)
cost	No	Defines the cost of including each PU in the reserve system
status	No	Defines whether a PU is locked in or out of the initial and final reserve systems
xloc	No	The x-axis coordinate of the planning unit
yloc	No	The y-axis coordinate of the planning unit

 The header names need to be exact. All must be lower case and contain no punctuation, spaces or numeric characters. The order headers are presented in does not matter. An example is given in Figure 7.

id	cost	status
1	300	0
2	100	0
3	1000	0
4	200	0
5	300	0
...		

Figure 7. *An example of a subset of the Planning Unit File (pu.dat)*

5.3.3.1 Planning Unit ID

Variable Name:	*id*

Required:	Yes
Description:	A unique numerical identifier for each planning unit. Values for the variable 'id' can be any number (i.e. there is no requirement to start at number 1) but they must not contain spaces, letters or punctuation. This number should not be confused with the variable, 'id', in the Conservation Feature File (see Section 5.3.2).

5.3.3.2 Planning Unit Cost

Variable Name:	*cost*
Required:	No
Description:	The cost of including each planning unit in the reserve system. The value entered for this variable will be the amount added to the objective function (see Appendix B-1 and B-1.1) when that planning unit is included in the reserve system. The cost of a planning unit can be based on any number of measures. It is a variable worth thinking carefully about as it can have a very large influence on the solutions Marxan generates.
Getting Started:	Proper use of the cost function can be complicated. In its simplest form, the cost of all planning units can be set to 1. Marxan will then try and minimise the total number of planning units included in the reserve system but will judge the selection of planning units based solely on the features present and not on cost. If your planning units are not equivalent in size, an equivalent "default" measure is to use the area of the planning unit as its cost. Alternatively, the Marxan Good Practices Handbook describes the use of a transformed value of area. The rationale for this is based on the assumption that the larger the reserve size the more costly it will be to implement and manage, although this is not always the case, and costs are almost never linear. A more sophisticated (and generally better) alternative is to use a measure of the actual fiscal cost of including that planning unit in a reserve system (for example see Naidoo et al. 2006). This may be the cost required to purchase that piece of land, or the opportunity cost of

	alternate land and sea uses that are incompatible with conservation. Cost can also be any relative social, economic or ecological measure. For instance, it may reflect the likelihood of success in different areas based on social willingness, enforceability, or the presence of uncontrollable threats. Although only a single cost can be defined for each planning unit, this cost can be a composite of different measures, provided there is a defensible basis with which to combine them. Costs of the same currency can be combined (e.g. if both costs are monetary). Costs of a different currency cannot sensibly be combined without using arbitrary weightings (e.g. if one cost is monetary and one is social).

5.3.3.3 Planning Unit Status

Variable Name:	*status*
Required:	No
Description:	This variable defines whether a planning unit (PU) is locked in or out of the initial and final reserve systems. It can take one of four values: The PU is not guaranteed to be in the initial (or seed) reserve system, however, it still may be. Its chance of being included in the initial reserve system is determined by the 'starting proportion' specified in the Input Parameter File (see Section 5.3.1). Status = 0. The PU will be included in the initial reserve system but may or may not be in the final solution. Status = 1. The PU will be included in the initial reserve system but may or may not be in the final solution. Status = 2. The PU is fixed in the reserve system ("locked in"). It starts in the initial reserve system and cannot be removed. Status = 3.The PU is fixed outside the reserve system ("locked out"). It is not included in the initial reserve system and cannot be added.

Getting Started:	This variable is not necessary and if not included will take the default value of 0. In general, it is helpful to first run Marxan without any sites locked in or out, to provide an unbiased near-optimal solution. However, this variable can be useful to explore various scenarios where planning units have either status '2' (must always be in the reserve system), or status '3' (can never be included in the reserve system). As an example, PUs located in existing protected areas could be assigned status '2' because it is unlikely that areas already protected will be traded for other areas. One could conduct an analysis with all protected areas locked in and all other areas locked out to identify how close an existing protected areas system achieves the stated ecological objectives. However, care must be taken when locking areas into a reserve network as it can make a significant difference to the character of final reserve networks. In scenarios where reserve compactness is important (i.e. a boundary length modifier is used), Marxan is likely to use existing conservation areas as hubs, or "seeds," around which to build the solution. This has the affect of substantially limiting the number of possible solutions. This may in fact be desirable as expanding existing protected areas is often politically and practically easier than creating new ones, but it can also lead to inefficient and possibly costly reserve solutions. It may also be appropriate to use status '2' for known occurrences of rare or highly valuable features (e.g. deep sea coral reefs), which would be irresponsible not to include in a reserve system but whose inclusion in the regular reserve selection process may unreasonably bias the results. In such situations, these features should be explicitly locked in so as not to compromise the transparency or defensibility of the planning exercise. If a strong emphasis is being placed on compactness then as an alternative scenario, Marxan should be run with the boundary length of planning units containing these locked-in features set to zero. This will minimise their influence on the overall reserve system, and allow the most efficient system to be revealed. The same may also apply to important cultural sites that planners would like to include in a reserve system.

	Status '3' is useful in cases where planning units will never be available for inclusion in a reserve system.

5.3.3.4 X Planning Unit Location

Variable Name:	*xloc*
Required:	No
Description and Getting Started:	The x-axis coordinate of the planning unit. This variable is only required if a minimum separation between feature occurrences has been specified in the 'sepdistance' column of the spec.dat file (see Section 5.3.2). The value entered reflects a point location for a planning unit, which may be its centre or some other sensible choice. This variable must be specified in conjunction with a value for the 'yloc' variable (below).

5.3.3.5 Y Planning Unit Location

Variable Name:	yloc
Required:	No
Description:	The y-axis coordinate of the planning unit. This variable is only required if a minimum separation between feature occurrences has been specified in the 'sepdistance' column of the spec.dat file (see Section 5.3.2). The value entered reflects a point location for a planning unit, which may be its centre or some other sensible choice. This variable must be specified in conjunction with a value for the 'xloc' variable (above).

5.3.4 The Planning Unit versus Conservation File (puvspr.dat)

The Planning Unit versus Conservation Feature File (puvspr.dat), also known as the Planning Unit versus Species Feature File, contains information on the distribution of conservation features across planning units. The file must contain three variables, listed in Table 6.

Table 6. *Variable names and requirements for Planning Unit versus Feature File (puvspr.dat)*

Variable Name	Required	Description

species	Yes	The unique id number of each conservation feature
pu	Yes	The id of a planning unit (PU) where the conservation feature listed on the same row occurs
amount	Yes	The amount of the conservation feature occurring in the PU listed on the same row

As displayed in the example in Figure 8, the puvspr.dat file starts with a header row which contains the name of each of the three variables, 'species', 'pu' and 'amount'. Each subsequent row then contains an id for a conservation feature (under the header 'species'), a planning unit id (under the header 'pu'), and a value for the amount of that conservation feature found in that planning unit (under the header 'amount'). Hence, there will be one row for each time a feature occurs in a planning unit. There are no default values for this file and any missing data or incorrect headers will prevent Marxan from running.

species	pu	amount
1	5	90
3	5	10
1	6	30
2	6	50
1	7	70
3	7	10
3	10	90
…		

Figure 8. *An example of a subset of the Planning Unit versus Conservation File (puvspr.dat)*

5.3.4.1 Conservation Feature ID

Variable Name:	*species*
Required:	Yes
Description:	The unique id number of each conservation feature. This must correspond to the id numbers used in the Conservation Feature File (see Section 5.3.2).

5.3.4.2 Planning Unit ID

Variable Name:	*pu*
Required:	Yes
Description:	The id of a planning unit where the conservation feature listed on the same row occurs. The planning unit id numbers must correspond to the numbers used in the Planning Unit File (see Section 5.3.3). PU Ids need to be organized from low to high.

5.3.4.3 Conservation Feature Amount

Variable Name:	*amount*
Required:	Yes
Description:	The amount of the conservation feature occurring in the planning unit listed on the same row. This amount may be related to the abundance of a species or the extent of a certain habitat type.
Getting Started:	There is no requirement to use the same metric for all conservation features. It is essential, however, that the amount for a given feature is in the same units used to set the target representation for that feature (see Section 5.3.4). You should not list cases where a feature does not occur in a planning unit. For example, you should not have a row with an amount of '0'. Instead, the row should be omitted altogether from the file. Marxan will assume that conservation features only occur in planning units where an amount has been entered. The default amount for a planning unit/feature pair that is omitted from the file is zero.

 There is the option to include an additional file together with the puvspr.dat, called the ***puvspr_sporder.dat.*** This file contains exactly the same information as the puvspr.dat, but it is organized by species ids rather than planning unit ids. This allows for a slight reduction of the processing time. If this file is used, it also needs to be listed in the input.dat file (Section 5.3.1) under the Input Files section as MATRIXSPORDERNAME puvsp_sporder.dat.

5.4 Optional File

The Boundary Length File is an optional input file for Marxan. If no file name is given in the Input Parameter File (see Section 5.3.1), Marxan will not attempt to find this file.

5.4.1 The Boundary Length File (bound.dat)

The Boundary Length File (bound.dat) contains information about the length or 'effective length' of shared boundaries between planning units. This file is necessary if you wish to use the Boundary Length Modifier (see Box 11 and Box 12) to improve the compactness of reserve solutions. The file consists of three required variables (Table 7). Each of these variables is presented in a column with the name of that variable as the column header (see example in Figure 9).

Table 7. *Variable names and requirements for Boundary Length File (bound.dat)*

Variable Name	Required	Description
id1	Yes	The id of a planning unit (PU)
id2	Yes	The id of a PU of the neighbouring PU to id1
amount	Yes	Relative measure of how important it is to include one planning unit in the reserve system, given the inclusion of the other

```
id1     id2     boundary
1       1       2000
1       2       1000
1       11      1000
2       2       1000
2       3       1000
2       12      1000
...
```

Figure 9. *An example of the Boundary Length File (bound.dat)*

5.4.1.1 Planning Unit IDs

Variable Name:	*id1* and *id2*
Required:	Yes
Description:	'id1' and 'id2' contain the id number of the two planning units that share a boundary. These do not have to be adjacent planning units, though they usually are – see below, and the Marxan Good Practices Handbook for more details.
Getting Started:	It does not matter in which order id numbers appear but it is important not to duplicate boundaries as Marxan will sum duplicate entries together when calculating boundary length.

5.4.1.2 Boundary Length

Variable Name:	boundary
Required:	Yes

Description:	The value for the variable 'boundary' can be derived in a variety of ways but it is essentially a relative measure of how important it is to include one planning unit in the reserve system, given the inclusion of the other. For instance, if the planning unit in the column, 'id1', has been added to the reserve system, how important is it that the planning unit in the column, 'id2', is also included, and vice versa. Because this is analogous to a 'cost' that must be paid if both planning units are not included, this variable is generally referred to as 'boundary cost'.
Getting Started:	In its most typical application, boundary cost reflects the actual geographical length of the boundary between two adjacent planning units, and this is a good place to begin. This cost can, however, be easily adjusted to reflect some other association between planning units, for instance, boundaries that are particularly desirable or undesirable. As an example, it may be undesirable to have reserves with boundaries adjacent to heavily populated areas, whereas having boundaries that abut private reserves or other conservation areas may be very desirable. In neither of these cases is this information reflected in the actual boundary length between neighbouring planning units. It is very important that if some relative measure, other than actual boundary length is used, the chosen metric must be well justified. Remember you are introducing bias into the reserve selection process and all stakeholders have a right to understand why. Transparency and defensibility are two of the core strengths of systematic conservation planning. Two planning units that are not adjacent to each other can still incur a 'boundary cost' if there is an important relationship between them. For instance, if a species requires habitat found in disparate planning units, either at different times of the year or during different life stages, then it makes little sense to protect one and not the other. If no association between these planning units is specified, Marxan may trade one in place of the other when in reality both are required for persistence of the species. This method could also be used to identify paths of connectivity between planning units, for instance, larval transport in marine systems, or hydrological flows in aquatic systems. In some cases, there will be no possibility of removing a boundary by including a neighbouring planning unit in the reserve system. This may happen, for instance, at the edge of a territorial jurisdiction or mandated

	planning region. Because planning units at the edge of a region will generally have shorter 'shared' boundaries, selection may be biased towards these planning units. This may be undesirable. To avoid biasing the selection of these planning units, they should feature in the Boundary Length File as 'irremovable boundaries'. This can be accomplished by specifying the length of a planning unit's boundary with itself, i.e. by repeating the same planning unit id in both the 'id1' and 'id2' columns. For more information regarding how to set the boundary file consult the Marxan Good Practices Handbook. The value entered in 'boundary' should always be '0' or greater. Although it is not generally necessary to specify cases where there is no cost between planning units, a zero cost boundary can be useful if you want to identify two planning units as neighbours but there is no actual boundary cost. This may be necessary if you have set minimum clump sizes for some conservation features (see Section 5.3.2 under "Minimum Clump Size").

6 Running the Software

Running the Marxan program is straightforward. Once all the input files are ready, all you need to do is double click on the 'Marxan.exe' file and the program will start automatically. To run successfully, however, the folder containing the program must be set up so that Marxan can find the required files and save the necessary outputs (see Section 5.2).

If Marxan executes successfully (Figure 10), a program screen showing information on the details and progress of each run will be displayed (unless Silent Running has been selected; see Section 7.3.1). Don't worry if this proceeds too quickly for you to read, all the necessary details will be saved in the output file folder in the Screen Log File called 'log.dat' (see Section 7.4.9). When Marxan completes the pre-set number of runs, it will stop but the program screen will remain visible. Pressing 'Enter' will exit the program and close the screen.

 If Marxan closes prematurely or halts with an error message it is likely to mean there is a problem with the format of one or more input files (see Appendix A- Troubleshooting for troubleshooting details).

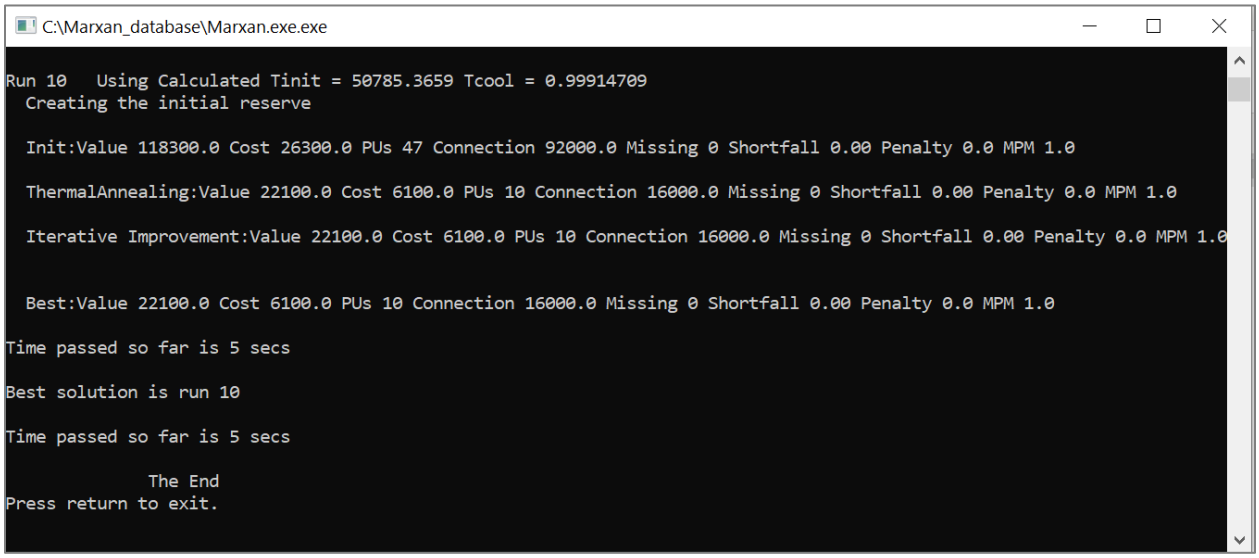


Figure 10. *A successful run in Marxan.exe*

7 Outputs

7.1 Overview of Output Files

Marxan provides a variety of outputs, including multiple solutions (i.e., combinations of planning units) of efficient reserve systems. These outputs are described in subsection 7.4.

All Marxan outputs are provided as .txt, .csv, and .dat format. These types of files can be viewed in basic text editing programs, such as Windows Notepad, or in spreadsheet and database software, such as Microsoft Excel or Access. The particular outputs you want Marxan to save, and the format for saving them, must be specified in the Input Parameter File (see Section 5.3.1).

Marxan does not generate any mapped results, although solutions and other outputs (e.g., selection frequency) can be visualised within select supporting software, such as Zonae Cognito (see Section 2.3), or with the use of GIS software like ArcGIS (www.esri.com) and QGIS (www.qgis.org). Although instructions for using these tools is beyond the scope of this manual, we have included Box 16 and Box 17 to outline the basic steps for visualising Marxan results in QGIS or ArcGIS (www.esri.com), as these are the most popular GIS software. Our aim here is to point you towards useful tools for visualization rather than provide step-by-step instructions.

Box 15. Not the Answer!

Marxan is a decision support tool to help guide the selection of efficient reserve systems; its output should never be interpreted or represented as "the answer." Although the results of a single Marxan run will provide a 'good' solution to your reserve design problem, it will not necessarily be the 'preferred' solution by the decision-makers. Because Marxan tests the utility of planning units in a pseudo-random fashion, each run is likely to be subtly (and sometimes extensively) different. In most cases, there will be many good solutions to the problem at hand. This is a positive attribute as it allows flexibility in planning and stakeholder negotiations.

7.2 Output File Management

Marxan will save output files in a folder, the name of which you have the chance to specify in the Input Parameter File. We suggest using the default folder name 'output' and then changing the file name once the run is complete.

 You must make sure that the output directory you specify has been created before running Marxan. This will help ensure that you do not write over output data when you start a new run.

As with the input file folder, it is useful to have separate output file folders for each scenario (i.e. within the same folder as the Marxan executable). It is also recommended to save a copy of the 'input.dat' file in the same folder, as this will mean you will always know what parameters were used to generate those results. Having this knowledge will help enormously in refining future scenarios and setline on appropriate parameters.

7.3 Screen Output

As Marxan runs, some output can be provided on the screen. This output can be used to check on how the program is running and to give a brief summary of the solutions. It can also give you an idea of how long a single run takes and from there you can gauge how long it will take to finish all runs for that scenario (e.g., if it takes one minute to complete a single run, then it will take approximately 100 minutes to complete 100 runs).

There are different levels of information that can be requested for the screen output; the level requested is termed the verbosity level (See Figure 11 and 12). Users can choose between the four verbosity levels: 0 (Silent Running), 1 (Results Only), 2 (General Progress), and 3 (Detailed Progress). The chosen level is set using the VERBOSITY variable in the Input Parameter File.

Note that you can save the screen output as an output file, called Screen Log File (see Section 7.4.9), to look at it after Marxan has finished running.

7.3.1 Silent Running

If the verbosity level is set to 'Silent Running' (verbosity level 0), then no information about the scenario will be displayed on the screen. This option should only be used if you are confident in Marxan execution and if you are saving all necessary outputs.

7.3.2 Results Only

When 'Results Only' (verbosity level 1) is selected, the screen will display the run numbers and the time it took to generate all runs. Each run number will appear on the screen as Marxan generates a final solution for that run (Figure 11). No other information regarding each run is provided.

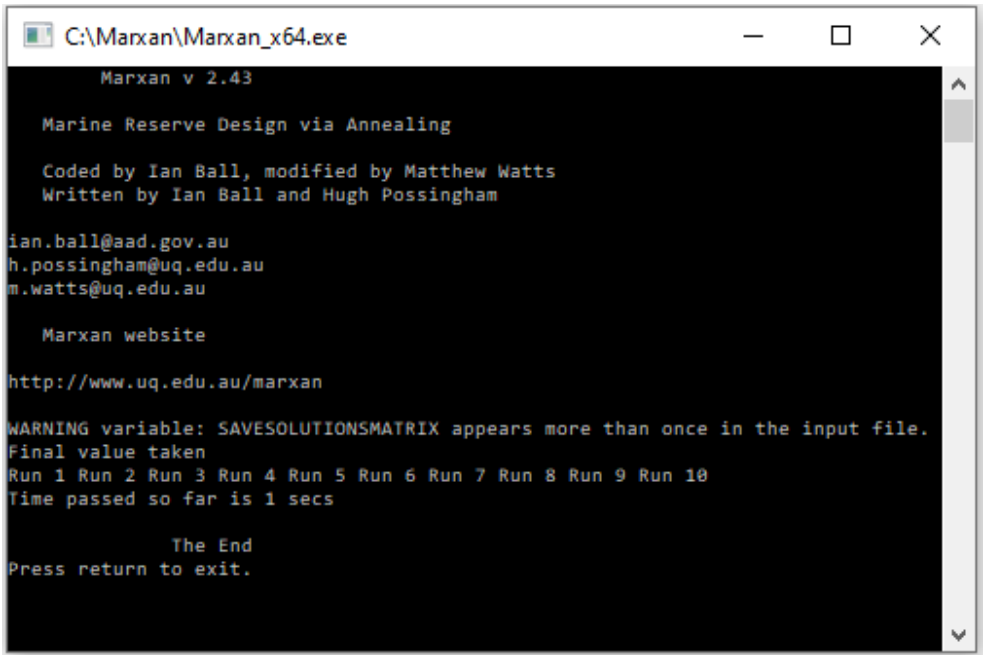


Figure 11*. Example of the on screen provided using verbosity 1 (Results Only)*

7.3.3 General Progress

'General Process' (verbosity level 2) is the default setting for the screen output and is recommended for most cases. If the screen output is set to either 'General Progress' or 'Detailed Progress' (verbosity level 3; see below), then the following summaries will be provided (see Figure 12).

1. Details of the data being entered;
2. The run number and the annealing parameters calculated during pre-processing (if adaptive annealing is being used);
3. The details of the initial or seed reserve system;
4. Results of the final solution from that run (the basic summary information for each run are described in
5. Table 8).
6. An output used by Marxan to quickly sort through all runs to determine which one

gave the 'best' solution.

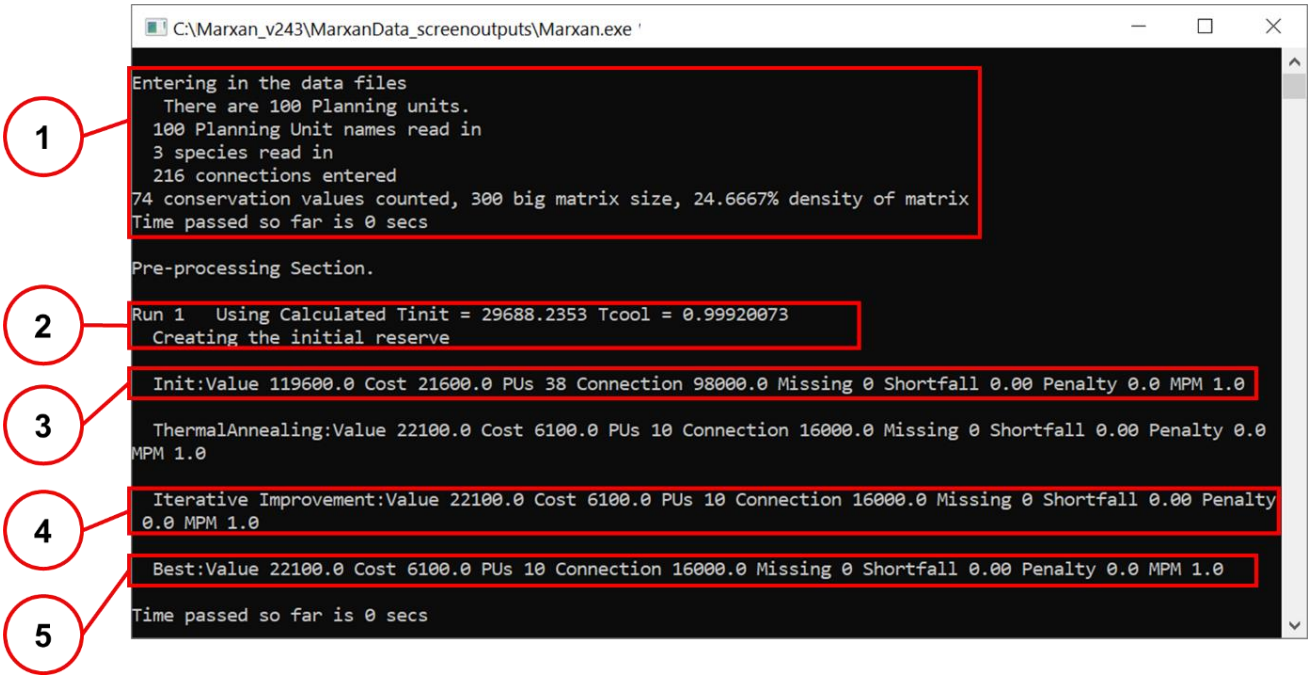


Figure 12. *Example of the on screen provided using verbosity level 2 and 3 (General Progress and Detailed Progress)*

Table 8. *Basic summary information of each run*

Information	Description
Run	The number of the repeat run (e.g., Run 1, Run 2, Run 3, etc.)
Value	The overall objective function value for the solution
Cost	The total cost of the reserve system as determined solely by the costs given to each planning unit
PUs	The number of planning units (PUs) contained in the solution
Boundary	The total boundary length of the reserve system
Missing	The number of conservation features that did not achieve their targets in the final solution
Shortfall	The amount by which the targets for conservation features have not been met in the solution
Penalty	The penalty that was added to the objective function when the reserve system failed to meet the representation targets for all features

This level of information can be particularly useful for a few reasons. First, if you a running

more than one optimisation procedure (i.e. simulated annealing followed by iterative improvement, see Appendix B-2), then it allows you to get a feel for how much work each of the different procedures is doing. For instance, if most of the gains in reserve value and target achievement are being made in the iterative improvement phase, you know that either the annealing parameters or the penalties need alteration. Second, if you want to begin using a fixed annealing schedule, this output can give you an idea of the sort of values Marxan is calculating using its adaptive annealing module (see Appendix B-2.1.1). Finally, if you have set some constraints on the initial reserve system, for example by including existing protected areas, this output will quickly confirm that this information is being included and the value of the existing reserves in terms of meeting your objectives.

7.3.4 Detailed Progress

'Detailed Progress' (verbosity level 3) will display the same screen as 'General Progress', but it will also generate the text file, 'DebugTraceFile_MarOpt'. This file will appear in the same file location as the Marxan executable file. The detailed information in the file shows exactly what the program did during each run. It can be used to confirm that the algorithm has run as it should and has not 'stalled'. It can also help identify certain problems (e.g., if the numbers do not change and it is "stalled") as it allows the user to track the annealing progress in detail. For this reason, some users prefer to use this setting to visually check that the program appears to be running well. In most cases, however, this level of detail is unnecessary. Furthermore, this verbosity level may lengthen the processing time.

7.4 Output Files

Marxan can save up to twelve different output files depending on what was specified in the Input Parameter File (Table 9). The file prefix 'output' will take on whatever name is specified by the user for variable 'SCENNAME' in the Input Parameter File. The format of each file (.dat, .txt, or .csv) is also specified by the user in the Input Parameter File. Where a number is included in the file name (e.g. output_r001.csv), this is the run number (or solution number) that generated that particular output.

Table 9. *Output file types and names*

Example File Name	Corresponding Variable Name in *input.dat*	Description
output_r001.csv	SAVERUN	Solutions for each run
output_best.csv	SAVEBEST	Best solution for all runs

output_mv001.csv	SAVETARGETMET	Missing value information for each run
output_mvbest.csv	SAVETARGETMET	Missing value information for the best run
output_sum.csv	SAVESUMMARY	Summary information
output_sen.dat	SAVESCEN	Scenario details
output_ssoln.csv	SAVESUMSOLN	Summed solution
output_penalty.csv	SAVEPENALTY	Computed penalty values for each feature
output_penalty_planning_units.csv	SAVEPENALTY	Computed penalty values for each planning unit
output_solutionsmatrix.csv	SAVESOLUTIONSMATRIX	Planning units selected in for each run
output_log.txt	SAVELOG	Screen log file
output_snap_r00001t01000.txt	SAVESNAPSTEPS	Snapshot files

7.4.1 Output File Format

The format of each output file (.dat, .txt, or .csv) is specified by the user in the Input Parameter File. It is recommended to use extension .csv for files, apart from the screen log file. The screen log file should be saved as a .txt file.

7.4.2 Solution for Each Run (e.g., output_r001.csv)

A file is produced for each run containing a list of all the planning units selected in the solution for that run. The run number is indicated by "_r001", i.e. run 1.

The file has as headings, 'PUID' for planning unit id and 'SOLUTION', and each line will have a planning unit id number, followed by a '0' if the corresponding planning unit was selected in the solution, or a '1' if the corresponding planning unit was selected to form part of the reserve system. An example of this output file is provided in Figure 13. Of the planning units shown in the figure, planning unit 2 and 6 are selected as part of the reserve system.

PUID	SOLUTION
1	0
2	1
3	0
4	0
5	0
6	1
7	0
8	0
9	0
10	0

Figure 13. *An example of a subset of a solution file (e.g., output_r001.csv)*

Box 16. How to visualize Marxan solutions in QGIS or ArcMap

Marxan results can be visualized using a GIS software, such as QGIS or ArcGIS. The following instructions outline the basic steps to visualize the solution of a given Marxan run (e.g., output_r001.csv) in QGIS or ArcGIS.

- Step 1: Open ArcMap or QGIS and add the file that contains your planning unit layer (e.g., a grid polygon shapefile).
- Step 2: Add the 'output_r001.csv' file to your project and append it to your planning unit layer using the planning unit id field available in both files.
- Step 3: Export the planning unit layer with the new appended information to make the linkage permanent, then add the new layer to your project.
- Step 4: Visualize the solution by using different colors for '0' values in the field SOLUTION, which represents unselected planning units, and '1' values which indicate selected planning units.

7.4.3 Best Solution (e.g., output_best.csv)

The Best Solution file is the same as the file described above, except it is for the run that produced the solution with the best objective value (i.e., the solution with the lowest Marxan score). Note that the solution from the 'best' run is only superior with regard to the objective function value. In reality, it may not be the best reserve system. Furthermore, the best solution may be only marginally better than the other solutions. Thus, 'best' has a very narrow meaning here and should not be communicated to stakeholders or decision-makers as the ideal solution. Rather, it should be viewed as a very good solution, within a continuum

of options. More discussion on this topic can be found in the Marxan Good Practices Handbook.

 Please note that if you are using **Marxan version 2.43**, the **solution labeled as "_best" is NOT the solution with the lowest Marxan score**, but the solution from the last run. For example, if you have asked Marxan to produce 10 solutions, the output_best.csv is equal to output_r010.csv. To find out which solution is the best, consult the Summary File (section 7.4.6) and identify the solution with the lowest Marxan score.

7.4.4 Missing Values for Each Run (e.g., output_mv001.csv)

This file contains information about the representation of conservation features in the solution for each run. It contains nine columns, which basically report on how the solution performed relative to the targets (Table 10). Some of these are simply a summary of the information provided in the Conservation Feature File. An example of this file is provided in Figure 14.

Table 10. *Description of Missing Value File Headers*

Header	Description
Conservation Feature	The unique ID number of the conservation feature.
Feature Name*	The optional name of the conservation feature. If no name has been specified, then nothing will appear in this column.
Target	The target level of representation (if any) for that conservation feature.
Amount Held	The amount of that conservation feature captured in the reserve system. Only amounts in valid clumps are included.
Occurrence Target	The target number of occurrences in the reserve system for that conservation feature.
Separation Target	The number of mutually and adequately separated occurrences of that conservation feature required in the reserve system.
Separation Achieved	The number reported here will be the lowest of either: the number of separate occurrences that are achieved in the reserve system; or the target number of separate occurrences. The separation count (see Appendix B-1.4) never exceeds the separation target for that feature. This is a convention which speeds up the execution of the software but no information is given about how far this target is exceeded.
Target Met	An alphabetic variable that returns 'yes' if all the targets set for that feature are met, otherwise it returns 'no'.
MPM	Stands for Minimum Proportion Met. It is the proportion of the target achieved. For example, if we target is achieved (or the amount held

exceeds the original target amount) the value will be 1.

* This column may appear blank (i.e., names not shown) even if a name has been specified in the spec.dat file. This bug will be addressed in newer version of the software.

Conservation Feature	Feature Name	Target	Amount Held	Occurrence Target	Occurrences Held	Separation Target	Separation Achieved	Target Met	MPM
3		246	260	0	4	0	0	yes	1
2		246	250	0	4	0	0	yes	1
1		272	280	0	4	0	0	yes	1

Figure 14. *An example of a solution file (e.g., output_mv001.csv)*

7.4.5 Missing Value Information for the Best Run (e.g., output_mvbest.csv)

Exactly as above except for the run that produced the solution with the lowest objective function score (i.e., the 'best' solution).

7.4.6 Summary Information (e.g., output_sum.csv)

This file contains the summary information for each repeat run. It contains nine columns, which basically report on how the solution performed relative to the targets (Table 11). An example of this file is provided in Figure 15.

Table 11. *Description of Summary File Headers*

Header	Description
Run_Number	Which of the repeat runs (or solutions) the output pertains to.
Score	This is the overall objective function value for the solution from that run. This includes not only the cost of the planning units and the boundary length but also the penalties for failing to adequately represent all conservation features or exceeding the cost threshold. It is useful to know this value because it is how Marxan chooses the 'best' solution out of you repeat runs.
Cost	This is the total cost of the reserve system as determined solely by the costs given to each planning unit.
Planning_Units	The number of planning units contained in the solution for that run.
Connectivity	The total boundary length of the reserve system. If boundary length is not being considered in the analyses (i.e. no Boundary Length File is provided), then this value will read '0.0'.
Connectivity_Total	Total boundary of planning units in study area.

Connectivity_In	Sum of shared boundary between selected planning units.
Connectivity_Edge	Same as Connectivity.
Connectivity_Out	Sum of the outer boundaries of unselected planning units.
Connectivity_In Fraction	Connectivity_In/Connectivity_Total - the larger this fraction, the more spatially compact the solution.
Penalty	The penalty that was added to the objective function because the reserve system failed to meet the representation targets for all features. If all features are adequately represented, then the penalty value will be either 0.0 or '-0.0'. (Because of round-off error it is not likely to be exactly equal to 0, but with only one decimal place presented the round-off error will probably be hidden). The penalty is useful to know because it can give you an idea of the cost required to meet the remaining targets; this is something that is not captured simply by looking at the shortfall. It is also another way to rank the success of runs, looking only at those solutions that have a low penalty. See Appendix B-1.3 for more information.
Shortfall	The amount by which the targets for conservation features have not been met in the solution for that run. The shortfall reported here is the total shortfall summed across all conservation features. The shortfall is a good indication of whether missing conservation features are very close or very far from their targets. If there are a number of conservation features which have missed their targets but the combined shortfall is very small then a planner might not be too concerned.
Missing Values	The number of features that did not achieve their targets in the final solution for that run. This is screened according to the 'misslevel', which has been set in the Input Parameter File. If the miss level is set to 1 then every conservation feature which falls below its target level is counted as missing. If the miss level is set lower than 1 (e.g. 0.98), Marxan may not report a feature as missing even if the reserve system contains slightly less than the target amount.
MPM	The Minimum Proportion Met for the worst performing feature. That is, this value corresponds to the lowest MPM value in the missing value file.

Run_Number	Score	Cost	Planning_Units	Connectivity	Connectivity_Total	Connectivity_In	Connectivity_Edge
1	22600	4600	8	18000	220000	7000	18000
2	22100	6100	10	16000	220000	12000	16000
3	24671	3700	8	20000	220000	6000	20000
4	24800	4800	10	20000	220000	10000	20000
5	22800	6800	13	16000	220000	18000	16000
6	22100	6100	10	16000	220000	12000	16000
7	24700	4700	9	20000	220000	8000	20000
8	22971	4000	9	18000	220000	9000	18000
9	22100	6100	10	16000	220000	12000	16000
10	22100	6100	10	16000	220000	12000	16000

Connectivity_Out	Connectivity_In_Fraction	Penalty	Shortfall	Missing_Values	MPM
195000	0.031818	0	0	0	1
192000	0.054545	0	0	0	1
194000	0.027273	971	2	1	0.993
190000	0.045455	0	0	0	1
186000	0.081818	0	0	0	1
192000	0.054545	0	0	0	1
192000	0.036364	0	0	0	1
193000	0.040909	971	2	1	0.993
192000	0.054545	0	0	0	1
192000	0.054545	0	0	0	1

Figure 15. *Example of the summary output file (output_sum.dat). Note that the table is divided into two parts for display purposes*

7.4.7 Scenario Details (e.g., output_sen.dat)

This file contains a documented list of all the major parameter values for that scenario (Figure 16). This file is very useful to keep track of the parameters that produced certain results, especially when multiple scenarios are run. This information is necessary to help select appropriate value for commonly modified parameters.

```
Number of Planning Units 100
Number of Conservation Values 3
Starting proportion 0.50
Connection modifier 0.00

Clumping - default step function
Algorithm Used :Annealing and Iterative Improvement
No Heuristic used
Number of iterations 1000000
Initial temperature set adaptively
Cooling factor set adaptively
Number of temperature decreases 10000

Cost Threshold Disabled
Threshold penalty factor A N/A
Threshold penalty factor B N/A

Random Seed -1
Number of runs 10
```

Figure 16. *Example of the scenario details output file (output_sen.dat)*

7.4.8 Summed Solution (e.g., output_ssoln.csv)

Summed solution provides the selection frequency of each planning unit across all runs. The file has as headings, 'planning_unit' and 'number', and each line will have a planning unit id number, followed by the number of times that planning unit was selected in the final solution across all repeat runs. An example is provided in Figure 17.

planning_	number
100	0
99	0
98	0
97	0
96	0
95	1
94	0
93	0
92	0

Figure 17. *Example of a subset of the summed solution output file (output_ssoln.csv)*

This is perhaps the most commonly used of the Marxan output files and certainly the most commonly displayed. It provides an indication of how useful each planning unit is for creating an efficient reserve system.

A map of summed solution output across your planning region can be viewed rather like a map of conservation priority. Those planning units that are commonly selected in final reserve solution (e.g. >70% of the time) are likely to be required for an efficient reserve system. Planning units that are rarely selected are less urgent, as the conservation features, they contain can probably be also acquired in other locations. Note, however, that even if a planning unit is selected in nearly every solution, this does not mean that you cannot get the same features in other places. It simply means that this planning unit helps provide efficient solutions. In reality, this may have little to do with the conservation features present at the site and more to do with either the cost or location of the planning unit.

Box 17. How to visualize Marxan summed solutions (or selection frequency) file in QGIS or ArcMap

- Step 1: Open ArcMap or QGIS and add the file that contains your planning unit layer (e.g., a grid polygon shapefile)
- Step 2: Add the output_ssoln.csv to your project and append it to your planning unit layer using the planning unit id field available in both files.
- Step 3: Export the planning units layer with the new appended information to make linkage permanent, and add it back to your project.
- Step 4: Visualize the selection frequency by using graduate colors and displaying the values in the 'number' field. It is recommended to visualize '0' values in their own class.

7.4.9 Screen Log File (e.g., output_log.dat)

A text file containing exactly what Marxan displayed as screen output for that scenario (see Section 7.3). This can be useful in de-bugging, or if for instance, you want to go back through the runs to investigate how much work is being done during the simulated annealing phase relative to iterative improvement.

7.4.10 Penalty Files (e.g., output_penalty.csv and output_penalty_planning_units.csv)

This file contains the penalty computed by Marxan for all features. 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.

7.4.11 Solution Matrix File (e.g., output_solutionmatrix.csv)

This file records which planning units were selected to form part of the reserve system for all runs. The first row contains a list of the planning unit identifiers (P1, P3, P4, …). The first column with the heading 'SolutionsMatrix' contains as many entries as solutions were generated (S1, S2, S3…). Matrix cell with values of '1' indicate that that particular planning unit was selected in that particular solution, and vice versa for '0' values.

This file is a useful way to export information on planning unit selection for cluster analysis in R or other statistical software packages. An example is presented in Figure 18.

SolutionsMatrix	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
S1	0	0	0	0	0	0	0	0	0	0
S2	0	0	0	0	0	0	0	0	0	0
S3	0	0	0	0	0	0	0	0	0	0
S4	0	0	0	0	0	0	0	0	0	0
S5	0	0	0	0	0	0	0	0	0	0
S6	0	0	0	0	0	0	0	0	0	0
S7	0	0	0	0	0	0	0	0	0	0
S8	0	0	0	0	0	0	0	0	0	0
S9	0	0	0	0	0	0	0	0	0	0
S10	0	0	0	0	0	0	0	0	0	0

Figure 18. *Example of a subset of the solution matrix file (output_solutionmatrix.csv)*

7.4.12 Snapshot File (e.g., output_snap_r00001t01000.txt)

Snapshot output files present the solution progress at stages during the optimisation procedure. The current solution is saved either at a predetermined number of system iterations of system changes. It is saved in the same format as the final solution for each run. These files allow the user to examine the progress of a solution method. It is really only needed for advanced analyses to look at how the annealing proceeds under different parameter values. In general, it is not recommended to save these files as it can slow down the processing time.

8 Getting Good Results

This manual should provide you with all the information you need to successfully and correctly use Marxan. Ensuring that your Marxan analyses are robust and defensible is beyond the scope of this document. This is covered in the Marxan Good Practices Handbook. Once you have a feeling for how the program works, we strongly encourage you to read the Marxan Good Practices Handbook before undertaking more complex analyses. In this section, we simply mention some of the things you must be prepared to undertake in order to ensure that Marxan delivers quality, defendable outputs.

 The most important thing to remember is that **Marxan provides decision-support but is not the decision-maker.** Our job as Marxan users is to ensure we have a robust understanding of the tool, can defend the choices we make in the scenarios we construct, and can explain our results to stakeholders.

8.1 Experimentation

As outlined in this manual, many of the parameters will require a lot of experimentation before you can expect Marxan to deliver reasonable solutions (i.e., conservation targets being met, and each of your other objectives, such as the level of clumping, satisfied to an acceptable degree). Each parameter should ideally be set in a stepwise and systematic manner. However, this can be challenging as parameters are not independent of each other. Parameters often interact in unexpected ways. For instance, the optimal BLM may change dramatically if the FPF is modified. Similarly, changes to your input data can mean that the parameter values should be re-calibrated.

Numerous methods have been suggested to help select appropriate Marxan parameter values (some of them mentioned in this manual). However, there is no substitute for simply exploring and understanding as many scenarios as your project time and budget permit and ensuring that there is adequate time allocated for these experimentations.

8.2 Visual Inspection

Conservation planning is a spatial discipline and its natural medium is cartographic. Although visual display is perhaps the most basic and often subjective of post-processing procedures, the power of the human eye to see visual trends should not be underrated and can detect issues that can be missed with sophisticated spatial statistics. As mentioned in the

first chapter of the manual, available data are not always ideal and there are many subtleties of reserve selection that cannot be incorporated into Marxan. Knowledge of your planning region will help avoid obvious errors in reserve placement. The knowledge of any problems should be used to update future scenarios being run in Marxan. Simply modifying the map of solutions at the end of an analysis is likely to lead to both inadequacies and inefficiencies in the solution.

To help the inspection process, it can be useful to visually compare the solutions with your data layers. For instance, you may notice that solutions are primarily driven by the distribution of costs rather than conservation features, or that the distribution of only a few conservation features largely explains the shape of the solutions. While these are not necessarily problems, they are very good to know, and can influence the next iteration of selecting parameter values.

8.3 Sensitivity Analyses

Even if you are happy with the quality of the reserve system solutions Marxan is producing, it is important to consider how the solutions would change if some of the scenario details changed. If for example, small changes in your cost data lead to large changes in the optimal reserve system, then you would want to ensure that your use of a particular cost structure is well justified.

The robustness of your solutions to small changes in the scenario details should ideally be explored through formal sensitivity analysis where the results of modifying parameters, constraints and data are compared both qualitatively and quantitatively. That said, due to the large number of variables and conservation features in any given analysis, most sensitivity analyses cannot look at everything, and therefore only what are considered key attributes are examined.

Sensitivity analyses should include scenario details not commonly subject to experimentation, such as the size and shape of planning units, the conservation targets, extent of the planning region, and different types of ecological and cost data. There are a number of formal approaches to comparing the similarity of reserve systems following scenario changes, such as the Kappa statistic (see Richardson et al., 2006) and cluster analysis (see Airamé, 2005). Determining if the outputs from different scenarios are similar will help assess the sensitivity of your solutions. Reporting the sensitivity of solutions to different factors can be very useful to highlight the impact of social or political constraints on solutions, or to help direct investment in the collection of data. See Section 8.4 in the Marxan Good Practices Handbook for more information on how to carry out a sensitivity analysis on Marxan results.

8.4 Marxan Community

Marxan has built a large global community consisting of academics, decision scientists, policy makers, spatial planners and a diverse range of stakeholders. Community members are invaluable to supporting each other in various aspects of Marxan applications from technical support to stakeholder engagement. The future of Marxan depends on the continuous support of this community.

Please post any notifications of publications, reports and funding opportunities related to Marxan, as well as bugs, troubleshooting help and general requests for advice to the Marxan Google Group at https://groups.google.com/forum/#!forum/marxan. To become a member, go to https://marxansolutions.org/ and click on 'Join The Marxan Google Group'.

Glossary

Adaptive schedule annealing: An optional function in Marxan where the scheduling of simulated annealing is done automatically. Marxan samples the problem and sets the initial temperature and temperature decrease rates.

Algorithm: A mathematical process that systematically solves a problem using well-defined rules or processes. Marxan can use several optimization algorithms (exact algorithm, heuristic algorithm, simulated annealing and iterative improvement) to identify reserve design solutions for a minimum cost, subject to the constraint that stated objectives are achieved.

Boundary cost: Also referred to as boundary length. A boundary cost is specified between two planning units. When one of the two planning units is included in the reserve system, the boundary cost is a relative measure of the importance of also including the other planning unit, and vice versa. Although the relationship between two planning units is typically the length of the shared boundary, boundary costs can also be specified between non-adjacent planning units reflecting ecological or economic factors.

Boundary Length Modifier (BLM): A variable controlling how much emphasis to place on minimising the overall reserve system boundary length relative to the reserve system cost. Higher BLM values will produce a more compact reserve system.

Clumping: The minimum amount of a conservation feature required within adjacent planning units before that 'clump' is considered to effectively contribute towards achieving the representation target for that feature. A number of unique clumps of a conservation feature can also be assigned (See separation distance).

Conservation feature: An element of biodiversity selected as a focus for conservation planning or action. This can include ecological classifications, habitat types, species, physical features, processes or any element that can be measured in a planning unit.

Cost: The cost of including a planning unit in a reserve system. This cost should reflect the socio-political constraints to setting aside that planning unit for conservation actions. This could be: total area, cost of acquisition or any other relative social, economic or ecological measure. Each planning unit is assigned one cost, although several measures can be combined to create a cost metric.

Compactness: A measure of the clustering or grouping of planning units in a reserve solution. It is calculated as a ratio of the total boundary length of a reserve system to the total area of the reserve system. Stewart and Possingham (2005) describe this concept in more

detail.

Decision support software: A computer-based application that uses information on possible actions and constraints on these actions in order to aid the process of decision-making in pursuit of a stated objective.

Efficiency: Property of a reserve system solution which meets all conservation targets (e.g. ecosystems, habitats, species) at an acceptable cost and compactness.

Feature Penalty Factor: A user-defined multiplier for the penalty applied to the objective function when a conservation feature target is not met in the current reserve scenario.

Fixed schedule annealing: An optional function in Marxan where the scheduling of simulated annealing is set by the user. If fixed schedule annealing is used, the annealing schedule (including the initial temperature and rate of temperature decrease) must be set by the user prior to running the algorithm . (See also Simulated annealing and Adaptive schedule annealing.)

Geographic Information System (GIS): A computer-based system consisting of hardware and software required for the capture, storage, management, analysis and presentation of geographic (spatial) data.

Heuristic algorithm: General class of sub-optimal algorithms which use time-saving strategies , or "rules of thumb", to solve problems. If used in Marxan, planning units are added until biodiversity targets are met.

Irreplaceability: see Selection Frequency.

Iterative improvement: A simple heuristic wherein 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 improves the system, then it is made. In Marxan, iterative improvement can be used to discard redundant planning units from the solutions.

Kappa statistic: An index which compares the spatial overlap / similarity of two reserve systems against that which might be expected by chance alone.

Local minimum/Local optimum: A local minimum occurs at the point where simply adding one favourable planning unit or removing one unfavourable planning unit from a reserve system can no longer improve the objective function value. This essentially means the reserve system cannot be improved without substantially changing its structure.

Marxan Good Practice Handbook: A complementary document to this Marxan User Manual.

Maximum coverage problem: The objective of the maximal coverage problem is to maximize protection of features subject to the constraint that the resources expended do not

exceed a fixed cost. Marxan can approximate the maximum coverage problem using the Cost Threshold function; however, the result will likely be sub-optimal.

Minimum set problem: The objective of the minimum-set problem is to minimize resources expended, subject to the constraint that all features meet their conservation objectives. Marxan was designed to solve this type of conservation problem.

Objective function: An equation associated with an optimization problem which determines how good a solution is at solving the problem. In Marxan, the value of the equation is a function of planning unit costs, boundary costs, and penalties. Each solution to reserve design is assigned a objective function value; a solution with a low value is more optimal than a solution with a high value.

Planning units: Planning units are the building blocks of a reserve system. A study area is divided into planning units that are smaller geographic parcels of regular or irregular shapes. Examples include squares, hexagons, cadastral parcels and hydrological units.

Reserve system design: The approach used to design a network of areas that collectively address the objective of the conservation problem.

Selection frequency: Also commonly known as irreplaceability. How often a given planning unit is selected in the final reserve system across a series of Marxan solutions. This value is reported in the "Summed Solutions" output file.

Sensitivity analysis: The process of modifying input parameters, constraints and data to quantitatively assess the influence of different variables on the final solution; that is, the degree to which the outputs are "sensitive" to variations in these various parameters.

Separation distance: Defines the minimum distance that distinct clumps of a feature should befrom one another in order to be considered as separate representations. This could be considered a type of risk spreading.

Simulated annealing: An optimization method (algorithm) based on iterative improvement but with stochastic (random) acceptance of bad moves early on in the process to help avoid getting stuck prematurely at local minimum objective function value. (See Appendix B-2.1)

Species Penalty Factor (SPF): See Feature Penalty Factor (FPF).

Summed Solution: See Selection Frequency

Systematic conservation planning: Formal method for identifying potential areas for conservation management that will most efficiently achieve a specific set of objectives, commonly some minimum representation of biodiversity. The process, involves a clear and structured approach to priority setting, and is now the standard for both terrestrial and marine conservation. The effectiveness of systematic conservation planning stems from its

ability to make the best use of limited fiscal resources towards achieving conservation goals and do so in a manner that is defensible, accountable, and transparently recognises the requirements of different resource users.

Target / Representation target: Targets are the quantitative values (amounts) of each conservation feature to be achieved in the final reserve solution.

Verbosity: The amount of information displayed on-screen while Marxan is running. (See Section 3.2.1.5.1).

User interface: The means by which people interact with a particular software application. A Graphical User Interface (GUI) presents information in a user-friendly way using graphics, menus and icons.

References

Marxan Software

Ball, I. R., Possingham, H. P., & Watts, M. E. (2009). Marxan and relatives: Software for spatial conservation prioritization. In A. Moilanen, K. A. Wilson, & H. P. Possingham (Eds.), *Spatial conservation prioritisation: Quantitative methods and computational tools* (pp. 185–210). Oxford University Press.

Marxan Manual

Serra, N., Kockel, A., Game, E. T., Grantham H., Possingham H., & McGowan, J. (2020). Marxan User Manual: For Marxan version 2.43 and above. The Nature Conservancy (TNC), Arlington, Virginia, United States and Pacific Marine Analysis and Research Association (PacMARA), Victoria, British Columbia, Canada.

Marxan Good Practice Handbook

Ardron, J.A., Possingham, H.P., and Klein, C.J. (eds). 2010. Marxan Good Practices Handbook, Version 2. Pacific Marine Analysis and Research Association, Victoria, BC, Canada. 165 pages. www.pacmara.org.

References Cited in Manual

Airamé, S. (2005). *Channel Islands National Marine Sanctuary: Advancing the science and policy of marine protected areas* (A. Scholz & D. Wright (eds.)).

Ball, I. R., Possingham, H. P., & Watts, M. E. (2009). Marxan and relatives: Software for spatial conservation prioritization. In A. Moilanen, K. A. Wilson, & H. P. Possingham (Eds.), *Spatial conservation prioritisation: Quantitative methods and computational tools* (pp. 185–210). Oxford University Press.

Ban, Natalie C., & Vincent, A. C. J. (2009). Beyond marine reserves: Exploring the approach of selecting areas where fishing is permitted, rather than prohibited. *PLoS ONE*, *4*(7), 1–8. https://doi.org/10.1371/journal.pone.0006258

Ban, Natalie Corinna, & Klein, C. J. (2009). Spatial socioeconomic data as a cost in systematic marine conservation planning. *Conservation Letters*, *2*(5), 206–215. https://doi.org/10.1111/j.1755-263X.2009.00071.x

Beger, M., Grantham, H. S., Pressey, R. L., Wilson, K. A., Peterson, E. L., Dorfman, D., Mumby, P. J., Lourival, R., Brumbaugh, D. R., & Possingham, H. P. (2010). Conservation planning for connectivity across marine, freshwater, and terrestrial realms. *Biological Conservation*, *143*(3), 565–575. https://doi.org/10.1016/j.biocon.2009.11.006

Cabeza, M. (2003). Habitat loss and connectivity of reserve networks in probability approaches to reserve design. *Ecology Letters*, *6*(7), 665–672.

https://doi.org/10.1046/j.1461-0248.2003.00475.x

Carwardine, J., Wilson, K. A., Watts, M., Etter, A., Klein, C. J., & Possingham, H. P. (2008). Avoiding costly conservation mistakes: The importance of defining actions and costs in spatial priority settings. *PLoS ONE*, *3*(7). https://doi.org/10.1371/journal.pone.0002586

Crouzeilles, R., Beyer, H. L., Mills, M., Grelle, C. E. V., & Possingham, H. P. (2015). Incorporating habitat availability into systematic planning for restoration: A species-specific approach for Atlantic Forest mammals. *Diversity and Distributions*, *21*(9), 1027–1037. https://doi.org/10.1111/ddi.12349

Daigle, R. M., Metaxas, A., Balbar, A., McGowan, J., Treml, E. A., Kuempel, C. D., Possingham, H. P., & Beger, M. (2018). *Operationalizing ecological connectivity in spatial conservation planning with Marxan Connect*. http://dx.doi.org/10.1101/315424

Fernandes, L., Day, J., Lewis, A., Slegers, S., Kerrigan, B., Breen, D., Cameron, D., Jago, B., Hall, J., Lowe, D., Innes, J., Tanzer, J., Chadwick, V., Thompson, L., Gorman, K., Simmons, M., Barnett, B., Sampson, K., De'Ath, G., … Stapleton, K. (2005). Establishing representative no-take areas in the Great Barrier Reef: Large-scale implementation of theory on marine protected areas. *Conservation Biology*, *19*(6), 1733–1744. https://doi.org/10.1111/j.1523-1739.2005.00302.x

Game, E. T., Watts, M. E., Wooldridge, S., & Possingham, H. P. (2008). Planning for persistence in marine reserves: A question of catastrophic importance. *Ecological Applications*, *18*(3), 670–680. https://doi.org/10.1890/07-1027.1

Heiner, M., Galbadrakh, D., Batsaikhan, N., Bayarjargal, Y., Oakleaf, J., Tsogtsaikhan, B., Evans, J., & Kiesecker, J. (2019). Making space: Putting landscape-level mitigation into practice in Mongolia. *Conservation Science and Practice*, *1*(10), 1–15. https://doi.org/10.1111/csp2.110

Klein, C. J., Chan, A., Kircher, L., Cundiff, A. J., Gardner, N., Hrovat, Y., Scholz, A., Kendall, B. E., & Airamé, S. (2008). Striking a balance between biodiversity conservation and socioeconomic viability in the design of marine protected areas. *Conservation Biology*, *22*(3), 691–700. https://doi.org/10.1111/j.1523-1739.2008.00896.x

Kukkala, A. S., & Moilanen, A. (2013). Core concepts of spatial prioritisation in systematic conservation planning. *Biological Reviews*, *88*(2), 443–464. https://doi.org/10.1111/brv.12008

Margules, C. R., & Pressey, R. L. (2000). Systematic conservation planning. *Nature*, *405*, 243–253. https://doi.org/10.1038/35012251

McDonnell, M. D., Possingham, H., Ball, I. R., & Cousins, E. a. (2002). Mathematical models for spatially cohesive reserve design. *Environmental Modelling and Assessment*, *7*, 107–114. https://doi.org/10.1023/A:1015649716111

Plumptre, A. J., Fuller, R. A., Rwetsiba, A., Wanyama, F., Kujirakwinja, D., Driciru, M., Nangendo, G., Watson, J. E. M., & Possingham, H. P. (2014). Efficiently targeting resources to deter illegal activities in protected areas. *Journal of Applied Ecology*, *51*(3), 714–725.

Possingham, H. P., Ball, I. R., & Andelman, S. (2000). Mathematical methods for identifying representative reserve networks. In S. Ferson & M. Burgman (Eds.), *Quantitative methods for conservation biology* (pp. 291–305). Springer-Verlag.

Pressey, R. L., & Bottrill, M. C. (2009). Approaches to landscape- and seascape-scale conservation planning: convergence, contrasts and challenges. *Oryx*, *43*(4), 464. https://doi.org/10.1017/S0030605309990500

Richardson, E. A., Kaiser, M. J., Edwards-Jones, G., & Possingham, H. P. (2006). Sensitivity of marine-reserve design to the spatial resolution of socioeconomic data. *Conservation Biology*, *20*(4), 1191–1202. https://doi.org/10.1111/j.1523-1739.2006.00426.x

Sinclair, S. P., Milner-Gulland, E. J., Smith, R. J., McIntosh, E. J., Possingham, H. P., Vercammen, A., & Knight, A. T. (2018). The use, and usefulness, of spatial conservation prioritizations. *Conservation Letters*, *11*(6), e12459. https://doi.org/10.1111/conl.12459

Stewart, R. R., & Possingham, H. P. (2005). Efficiency, costs and trade-offs in marine reserve system design. *Environmental Modeling and Assessment*, *10*(3), 203–213. https://doi.org/10.1007/s10666-005-9001-y

Watts, M. E., Ball, I. R., Stewart, R. S., Klein, C. J., Wilson, K., Steinback, C., Lourival, R., Kircher, L., & Possingham, H. P. (2009). Marxan with Zones: Software for optimal conservation based land- and sea-use zoning. *Environmental Modelling and Software*, *24*(12), 1513–1521. https://doi.org/10.1016/j.envsoft.2009.06.005

Appendix A- Troubleshooting

This appendix includes examples of common error messages encountered while running Marxan, and a description of the possible causes and solutions.

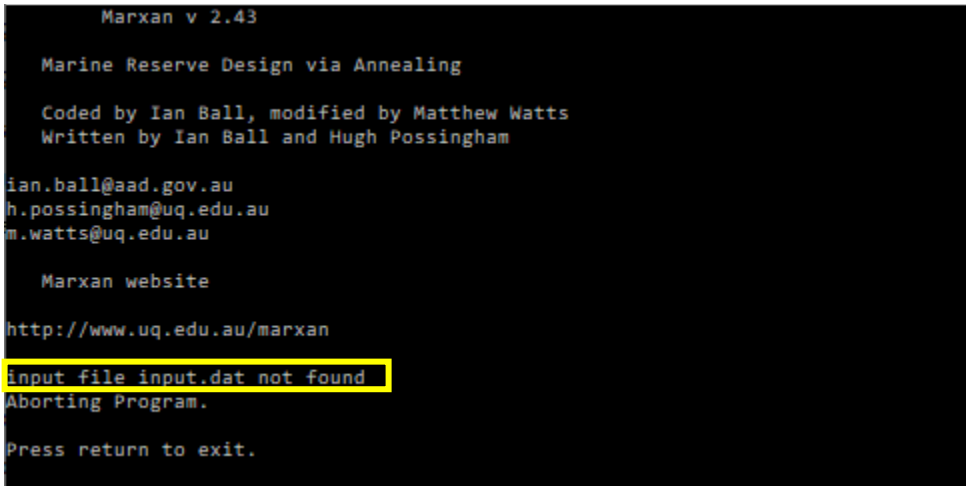
Please note that this list is not exhaustive. If you encounter issues or error messages not listed here, please contact us by sending an e-mail to marxancloud@gmail.com

A-1 Invalid input file (input.dat)

A-1.1 "Input file input.dat not found"

Error Message: The last thing Marxan shows is *"input file input.dat not found*".

Cause of Error Message: This occurs when you don't have an input.dat file containing the Marxan settings in the same folder as your Marxan executable file (e.g. Marxan_x64.exe).



A-1.2 "Entering in the data files; Planning Unit files input/pu.dat has not been found"

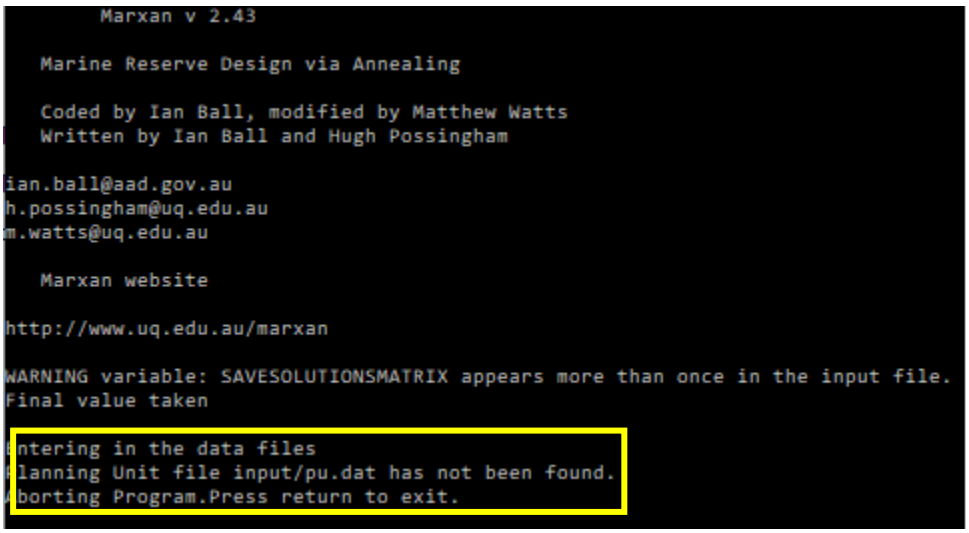
Error Message: The last thing Marxan shows is "*Entering in the data files; Planning Unit files input/pu.dat has not been found; Aborting Program. Press return to exit*".

Cause of Error Message: This error message can be due to different causes.

- This occurs when the folder containing the input files (e.g. 'Input' folder) and specified in input.dat via INPUTDIR is not included or does not match the actual folder name (or directory) on your computer (e.g. the input.dat file contains the text "input" and

your computer has a folder named "input_scenario").

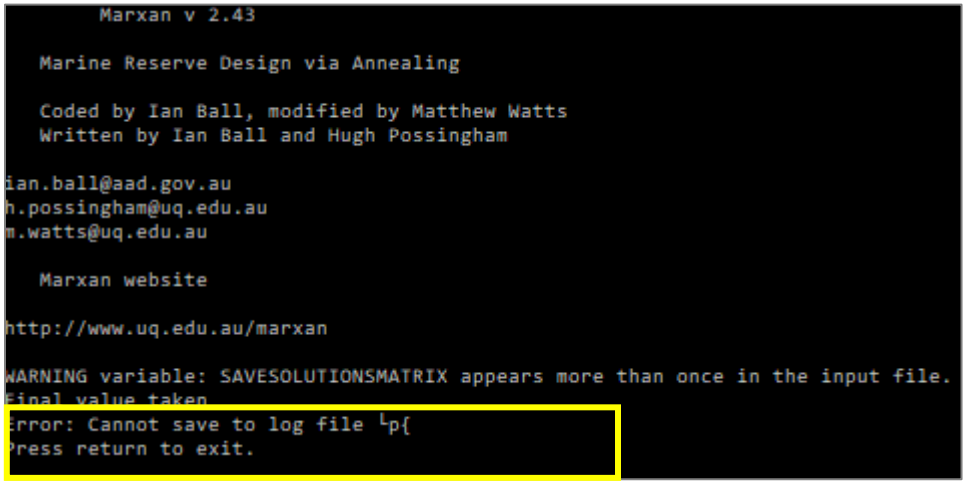
- This occurs when your planning unit file name pu.dat specified in input.dat via PUNAME is not included or does not match the actual planning unit file name on your computer (e.g. the input.dat file contains the text "pu.dat" and your computer has a file named "planningunit.dat").



A-1.3 "Error: Cannot save to log file"

Error Message: The last thing Marxan shows is "*Error: Cannot save to log file…*".

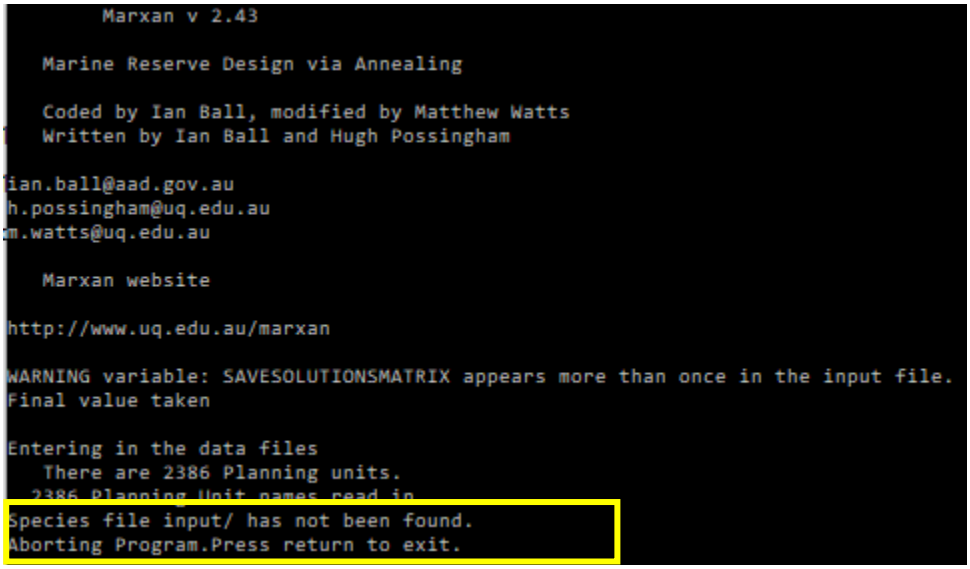
Cause of Error Message: This occurs when the folder for the output files (e.g. 'Output' folder) and specified in input.dat via OUTPUTDIR is not included or does not match the actual folder name (or directory) on your computer (e.g. the input.dat file contains the text "input" and your computer has a folder named "scenario_outputs").



A-1.4 "Species file has not been found"

Error Message: The last thing Marxan shows is "*Species file has not been found. Aborting Program. Press return to exit*".

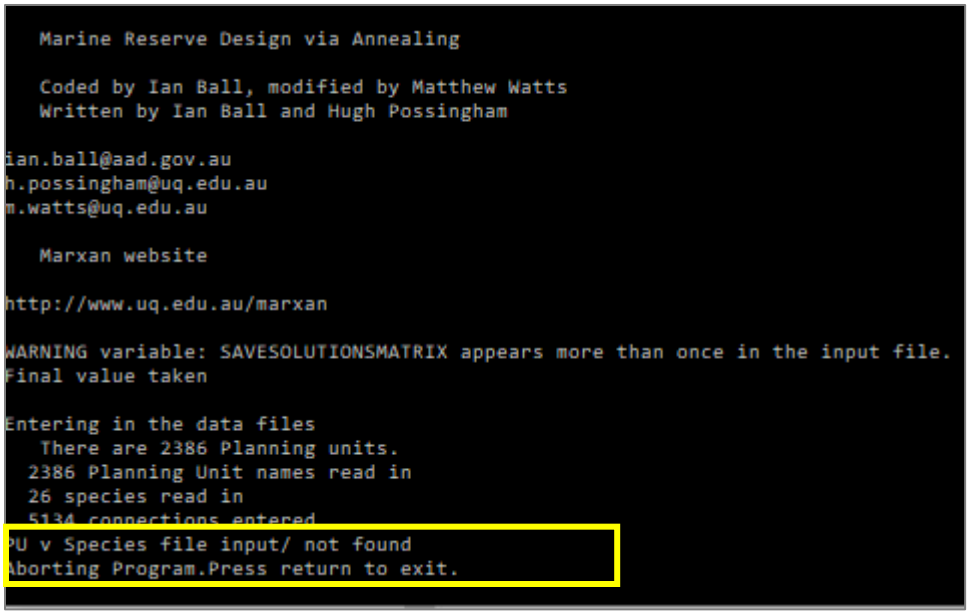
Cause of Error Message: This occurs when your species file name spec.dat specified in input.dat via SPECNAME is not included or does not match the actual species file name on your computer (e.g. the input.dat file contains the text "spec.dat" and your computer has a file named "spc.dat").



A-1.5 "PU v Species file input/ not found"

Error Message: The last thing Marxan shows is " *PU v Species file input/ not found. Aborting Program. Press return to exit*".

Cause of Error Message: This occurs when your planning unit versus species file name 'puvspr.dat' specified in input.dat via PUVSPRANME is not included or does not match the actual species file name on your computer (e.g. the input.dat file contains the text "puvspr.dat" and your computer has a file named "puspc.dat").

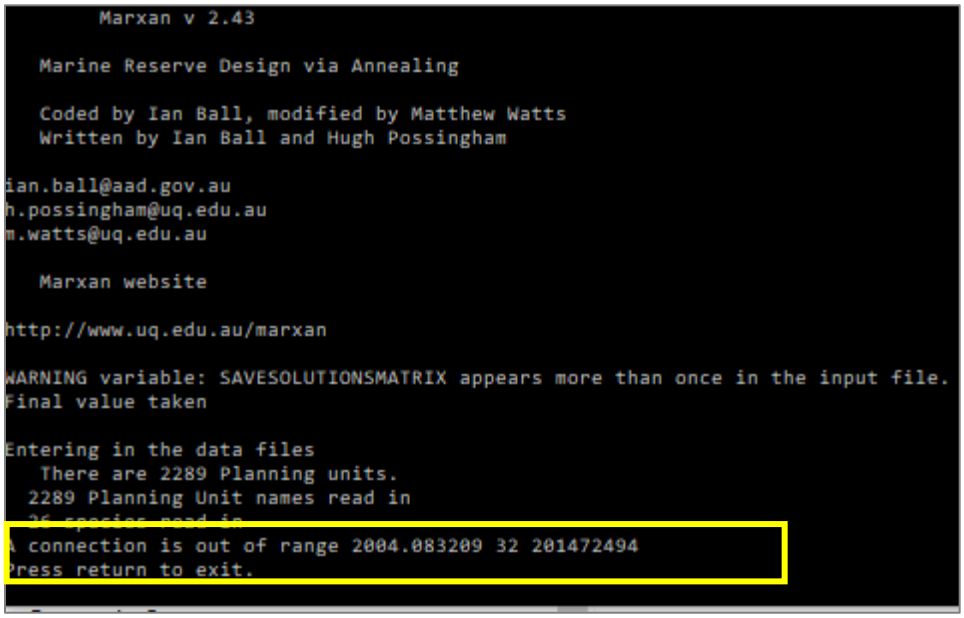


A-2 Invalid planning unit file (pu.dat)

A-2.1 "A connection is out of range"

Error Message: The last thing Marxan shows is " *A connection is out of range …*".

Cause of Error Message: This is caused by the 'pu.dat' file containing a planning unit id that is not present in 'bound.dat'



A-2.2 "Entering in the data files"

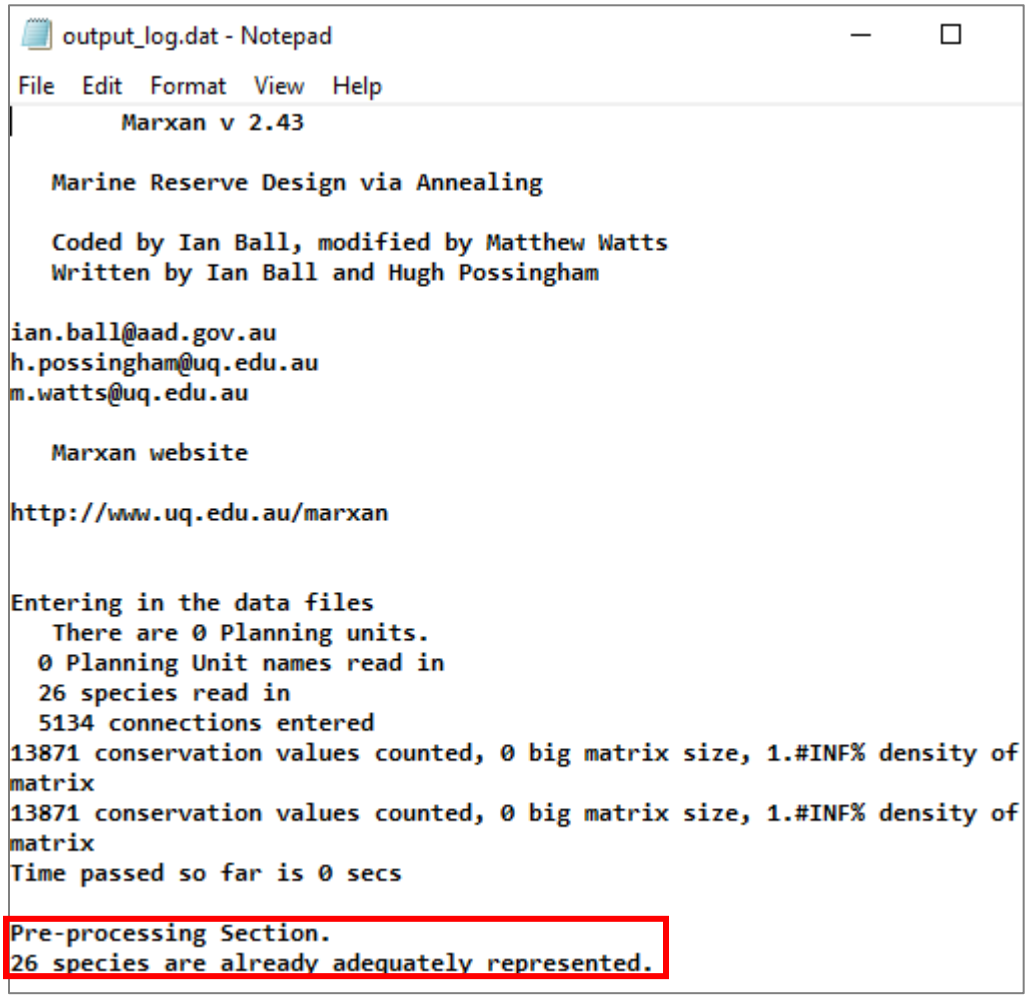
Error Message: The last thing Marxan shows is "*Entering in the data files*".

Cause of Error Message: The 'put.dat' file contains extra newline characters (e.g. blank spaces or extra lines) at the end of the file and these need to be deleted.

A-2.3 "Species are already adequately represented", "Run 1" or "Error reading planning units"

Error Message: The last thing Marxan shows is *"Species are already adequately represented"*, or *"Run 1"*, or *"Error reading planning units",* or Marxan crashes.

Cause of Error Message: This occurs when the pu.dat file is blank and does not contain any data.



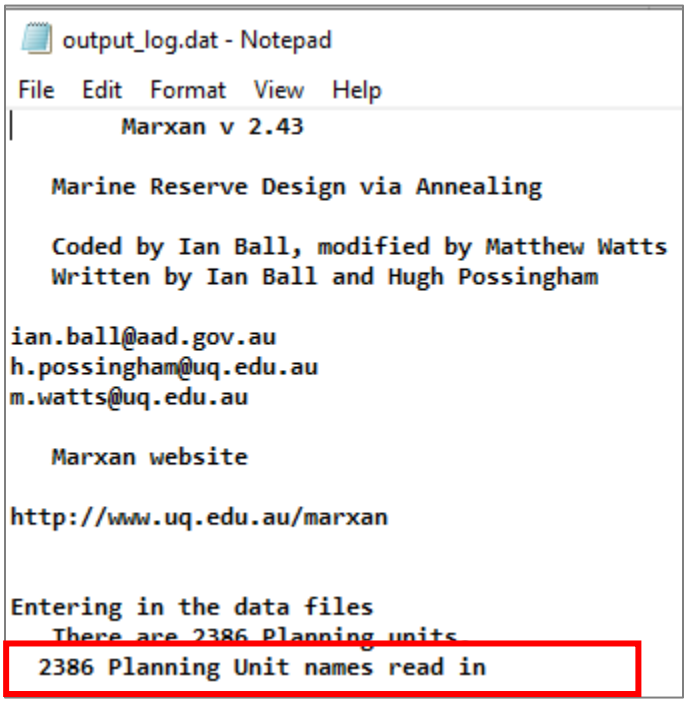
A-3 Errors related to an invalid species file (spec.dat)

A-3.1 "Planning Units names read in"

Error Message: The last thing Marxan shows is "*Planning Units names read in*".

Cause of Error Message: This can be caused by different issues with the spec.dat file.

- The name column in 'spec.dat' contains very long names. Try reducing the length of the feature names and running Marxan again. As a general rule of thumb, please consider using names which are less than 20 characters in length.
- The 'spec.dat' file contains extra newline characters (e.g. blank spaces or extra lines) at the end of the file and these need to be deleted.
- The 'spec.dat' file is blank.

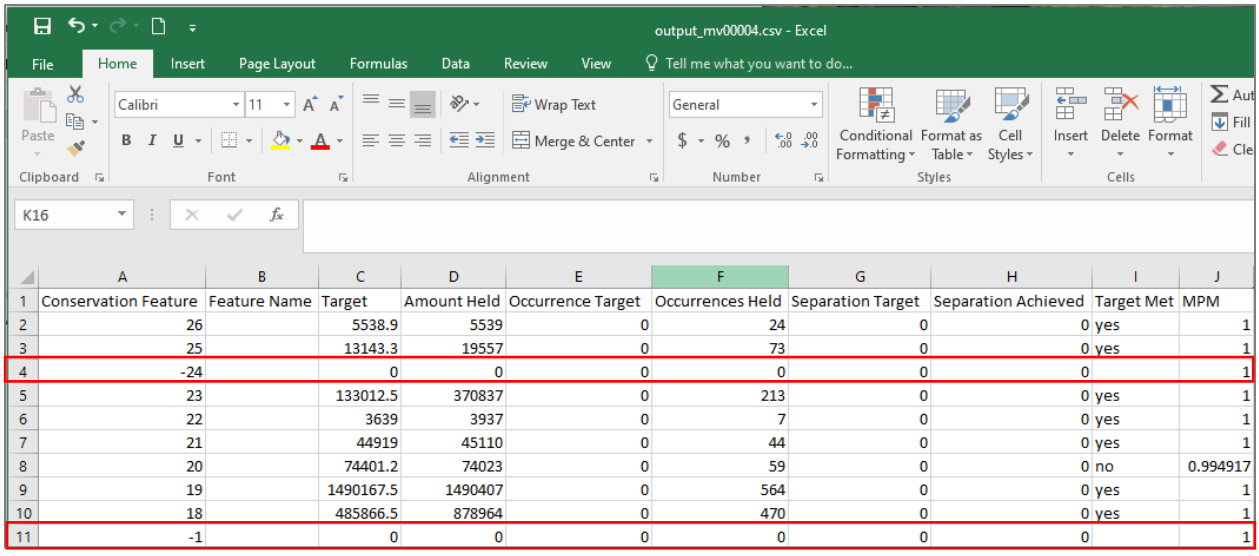


A-3.2 Missing value outputs contain negative feature id and no results

Error message: Marxan runs but the missing value outputs (e.g., output_mv00004.csv) contains a negative feature id and no results for the rest of the fields.

Cause of Error Message:

- This occurs when an 'id' in 'spec.dat' contains letters (e.g. a) or symbol (e.g. +) characters.



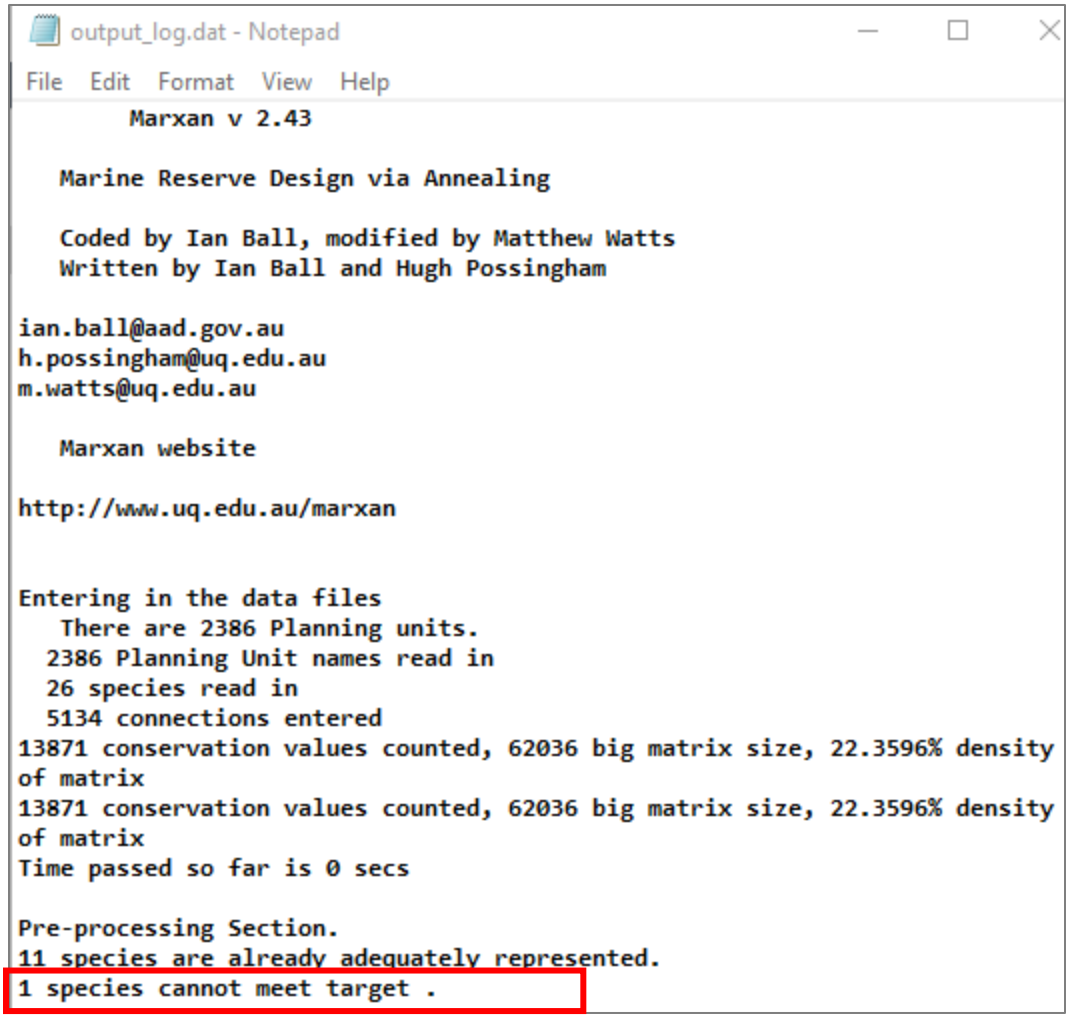
	A	B	C	D	E	F	G	H	I	J
1	Conservation Feature	Feature Name	Target	Amount Held	Occurrence Target	Occurrences Held	Separation Target	Separation Achieved	Target Met	MPM
2	26		5538.9	5539	0	24	0	0	yes	1
3	25		13143.3	19557	0	73	0	0	yes	1
4	-24		0	0	0	0	0	0		1
5	23		133012.5	370837	0	213	0	0	yes	1
6	22		3639	3937	0	7	0	0	yes	1
7	21		44919	45110	0	44	0	0	yes	1
8	20		74401.2	74023	0	59	0	0	no	0.994917
9	19		1490167.5	1490407	0	564	0	0	yes	1
10	18		485866.5	878964	0	470	0	0	yes	1
11	-1		0	0	0	0	0	0		1

A-3.3 "(x number) *species cannot meet target*" with targets set as a proportion

Error Message: The last thing Marxan shows is "(x number) *species cannot meet target*" and I have specified targets as a proportion in the 'prop' field.

Cause of Error Message:

- This occurs when the 'prop' column in 'spec.dat' contains values greater than one.
- This occurs when you have locked out planning units which prevent the target from being met.



A-3.4 "(x number) *species cannot meet target*" with targets set as an amount

Error Message: Marxan runs but shows me "(x amount) *species cannot meet target*" and I have specified targets as an overall amount (e.g. in km^2).

Cause of Error Message:

- This occurs when the 'target' column in 'spec.dat' contains values greater than one.
- This occurs when you have locked out planning units which prevent the target from being met.

A-3.5 Outputs files contain zeros in all columns

Error Message: Marxan runs but outputs files (e.g. the missing value files and the solutions

files) contain zeros in all columns.

Cause of Error Message: This occurs when the spec.dat file does not contain a prop or target column. Even if you don't have targets and just want to assess representation, you need to include these columns with a target/prop value of zero.

A-3.6 "(x number) conservation values counter, (x number) big matrix size, (x %) density of matrix"

Error Message: The last thing Marxan shows is *"*(x number) *conservation values counter,* (x number) *big matrix size,* (x %) *density of matrix*".

Cause of Error Message: This occurs when the 'spec.dat' file contains a species 'id' that does not exist in 'puvspr.dat'.

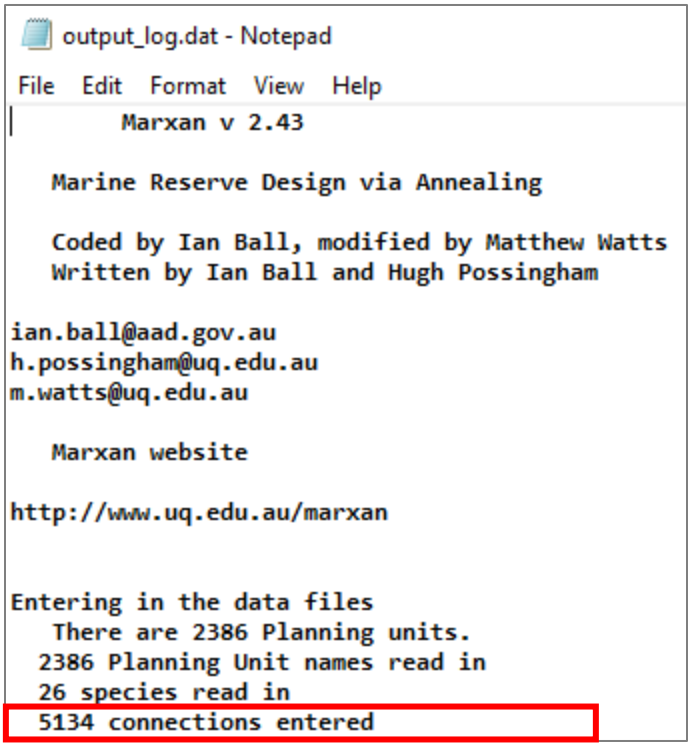
A-4 Invalid planning versus species file (puvspr.dat)

A-4.1 "(x number) connections entered""

Error Message: The last thing Marxan shows is "(x number) *connections entered"* and then crashes.

Cause of Error Message: This could be caused by:

- The 'puvspr.dat' contains a planning unit 'id' that is not included in the 'pu.dat'.
- The amount column 'puvspr.dat' contains a blank value. This could happen if you manually replaced zeros with blank cell values in Excel.
- The 'puvspr.dat' file contains extra newline characters (e.g. blank spaces or extra lines) at the end of the file. These need to be deleted.
- The 'amount' column in 'puvspr.dat' contains a non-numeric value (e.g. letters or symbols).
- The 'pu' column in 'puvspr.dat' contains a non-numeric value (e.g. letters or symbols).



A-4.2 "(x number) conservation values counter, (x number) big matrix size, (x %) density of matrix"

Error Message: Marxan shows me "(x number) *conservation values counter,* (x number) *big matrix size,* (x %) *density of matrix".*

Cause of Error Message: This occurs when the 'puvspr.dat' file contains a species 'id' that does not exist in 'spec.dat'.

A-5.1 Marxan runs but representation numbers are incorrect

Error: Marxan runs but representation numbers are obviously incorrect even though it says that all the targets have been met? For example, Marxan thinks that all the targets have been met, but the solutions are missing a small island which contains endemic species which are not found anywhere else.

Cause of Error: The 'puvspr.dat' file assumes that the first column always contains species data, and the second column always contains planning unit data. The names of the columns are ignored. So, if you have planning unit data in the second column, then Marxan think that it is the species data.

A-5 Invalid boundary file (bound.dat)

A-5.1 Output files do not have all the planning units

Error: Marxan runs but the output files do not have all the planning units.

Cause of Error: This is caused when the bound.dat does not include the all planning unit 'ids' as contained in the 'pu.dat'.

A-5.2 "A connection is out of range"

Error Message: The last thing Marxan shows is "*A connection is out of range…*".

Cause of Error Message: This is caused by the 'bound.dat' file contains more planning unit id that is not present in 'pu.dat'.

A-5.3 "(x number) *species read in*"

Error Message: The last thing Marxan shows is *"*(x number) *species read in*".

Cause of Error Message: The bound.dat file contains extra newline characters (e.g. blank spaces or extra lines) at the end of the file and these need to be deleted.

Appendix B- Marxan Technical Information

This appendix contains technical details about the way Marxan runs. While this information is not necessary to conduct basic runs, knowing how the program runs will assist in understanding how the changes you make to different parameters affect the results.

B-1 The Objective Function

The mathematical "heart" of Marxan is the objective function which evaluates and compares between potential reserve systems. This section provides information about how each of the components in the objective function are calculated.

$$\sum_{PUs} \text{Cost} + BLM \sum_{PUs} Boudary + \sum_{\substack{Con \\ Value}} FPF \times Penalty \quad = \quad \boldsymbol{Marxan\ Score} \qquad \textbf{(6)}$$

B-1.1 Cost

The 'Cost' component of the objective function is simply the sum of the costs given to each of the planning units included in the reserve system. Cost data is located in the Planning Unit File. This cost may be the actual economic cost of purchasing that planning unit, or it may reflect a more abstract concept such as the opportunity cost of putting that planning unit under protection.

B-1.2 Boundary and Boundary Length Modifier (BLM)

The 'Boundary' is the length of the boundary surrounding the reserve system. By including a boundary length term in the objective function we can control the level of fragmentation in the reserve system. The 'Boundary' component of the objective function is calculated first by summing the lengths of all boundaries between planning units that are within the reserve and those that are outside the reserve. Boundaries between two planning units that are both inside the reserve system are not counted. Information about the length of the boundaries between planning units is contained in the Boundary Length File. The value given to a shared boundary can reflect either the actual geographical length of the boundary between two planning units or some other association between planning units, for instance, boundaries that are particularly desirable or undesirable. Two planning units do not need to be adjacent to each to share a boundary. Box 12 describes different possible uses of the boundary value or cost.

Because the boundary length value is most probably going to be in units which are different from the planning unit cost measure, the two cannot simply be added together. In order to allow the boundary length to be added to the cost measure a multiplicative factor is used. This is referred to as the 'Boundary Length Modifier' (BLM). When calculating the objective function value, the sum of the boundaries is multiplied by the BLM. Not only does the inclusion of this modifier allow for compatibility between different metrics, it also provides a method of controlling the importance of reserve compactness, relative to reserve cost.

Changing the BLM allows a conservation planner to explore this issue. If a value of 0 is given to the BLM then boundary length will not be included in the objective function. If a high value is given to the BLM, then obtaining a compact reserve is likely to outweigh all other considerations.

B-1.3 Features Penalty Factor (FPF)

The Penalty component of the Marxan objective function is the penalty given to a reserve system for not adequately representing conservation features. It is based on the principle that if a conservation feature is below its target representation level, then the penalty should be an approximation of the cost of raising that conservation feature up to its target representation level. For example, if the requirement was to represent each conservation feature by at least one instance then the penalty for not having a given conservation feature would be the cost of the least expensive planning unit which holds an instance of that conservation feature. If you were missing a number of conservation features then you could produce a reserve system that was fully representative by adding the least expensive planning units containing each of the missing conservation features.

Marxan uses a greedy algorithm to estimate the cheapest way in which each conservation feature could be represented on its own and this forms the base penalty for that conservation feature. To do this Marxan adds together the cheapest planning units which would achieve the representation target. This approach is described in the following pseudo-code:

1. For each planning unit calculate a 'cost per hectare' value.
    a) Determine how much of the target for the given conservation 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 (Boundary Length Multiplier)

e) Cost-per-hectare is then the value for conservation feature divided by the overall cost.

2. 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 conservation feature to the representation level total.
3. Continue adding up these totals until you have found a collection of planning units which adequately represent the given conservation feature.
4. The base penalty for the conservation feature is the total cost (including boundary length multiplied by boundary length modifier) of these planning units.

Thus, if one conservation feature was completely unrepresented then the penalty would be the same as the cost of adding the simple set of planning units, chosen using the above code, to the system, 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.

There may be more efficient ways of representing a conservation feature than that determined by a greedy algorithm (see Appendix B, Section B-2). Consider the following example:

Conservation Feature A appears in a number of planning units, the best ones are:

Planning Unit	Cost	Amount of feature A
1	$2	3
2	$4	5
3	$5	5
4	$8	6

The target for feature 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. Obviously if we chose only planning units 2 and 3 we would still adequately represent feature A, but our cost would be only $9. 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, however, to have a penalty value which is too low because then the objective function might not improve by fully representing all conservation features. It is not problematic to have penalties which are higher than they absolutely need to be, sometimes it is even desirable. The boundary cost for a planning unit in the above pseudo-code is the sum of all of its boundaries. Unlike in the boundary component of the objective function, this assumes that the planning unit has no common boundaries with the

rest of the reserve and hence will again tend to overestimate the cost of the planning unit and the penalty.

It would be ideal to recalculate the penalties after each change to the reserve system. This, however, would be very 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. The penalty is calculated and fixed in the initialisation stage of the algorithm. It is applied in a straight forward linear manner: if a conservation 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 conservation feature's target but that there is no way of doing this which would decrease the objective value. If we take the example used above, then the penalty for conservation feature A is 11. If planning units 1 and 4 are already in the reserve system, then you have 9 units of conservation feature A and the penalty for under representation (remember the target is 10) will be calculated as 11 x (10-9)/10 = 1.1. So the feature attracts a penalty of 1.1 units and needs only 1 more unit of abundance to meet its target. As there are no planning units with a cost that low, the addition of any of the remaining planning units would increase the cost the reserve system much more than the gain in penalty reduction.

This problem can be fixed by setting a higher SPF or Species Penalty Factor (also known as the Feature Penalty Factor). The SPF can be thought of as way of distinguishing the relative worth of different conservation features and how important it is to get them fully represented. Features of high conservation value (these may be highly threatened features or those of significant social or economic importance) should have higher SPF values than less important features. This signifies that you are less willing to compromise on their representation in the reserve network.

When calculating the value of the objective function, Marxan will first calculate the penalty for any features that are under represented, multiply these penalties by the appropriate SPF value for each feature, and then sum these values across all features. Any features whose representation targets are met satisfactorily will have a penalty of zero and will therefore not increase the objective function value.

B-1.4 Spatial feature penalties

Marxan allows users to set two spatial constraints on the occurrence of features in possible reserve systems. These are: a minimum clump or aggregation size required before occurrences of that feature contribute towards meeting the overall target (Section 5.3.2 under "Minimum Clump Size"), and a minimum separation distance required between multiple occurrences of the same feature (Section 5.3.2 under "Minimum Separation Distance"). Both of these can be specified in the Conservation Feature File.

Before using these additional spatial features, however, we strongly suggest that the other Marxan parameters have already been tested and adjusted, as that these additional spatial features will slow down the algorithm considerably.

When calculating the initial penalty for a conservation feature which has spatial requirements Marxan uses a different method to that applied for the basic conservation feature penalty. The greedy method has been replaced with an iterative improvement method. A conservation feature which has a spatial aggregation rule has a second target value which specifies the smallest clump size which will count to the main target. If a group of contiguous planning units contain a conservation feature, but not as much as the minimum clump size, then the reserve system is penalised for that conservation feature as if none of those planning units contained the conservation feature.

More advanced penalties for sub-sized clumps can also be applied. For instance, instead of the clump not counting at all toward that conservation features amount it can count half of its value. Another alternative is based on the fact that the scaling factor for the amount that a clump contributes is equal to the proportion of the minimum clump size met. In this case, if the clump size is half of the minimum clump then the amount contributes half of its value to the conservation factors. In all cases, if a clump is larger than the minimum clump size then there is no penalty and the amount contributes directly to the total amount for that conservation feature.

These two alternative clumping rules may be useful in encouraging clumping of selected sites but they are inconsistent with the concept of a minimum clump size relating directly to a quantity such as a minimum viable population size. This is because they will both tend to meet some of a conservation feature's target with fragments of that conservation feature (possibly contained in clumps of other conservation features of large size). They could be used where clumping is to be encouraged but the minimum clump size is not as rigid as a minimum viable population size.

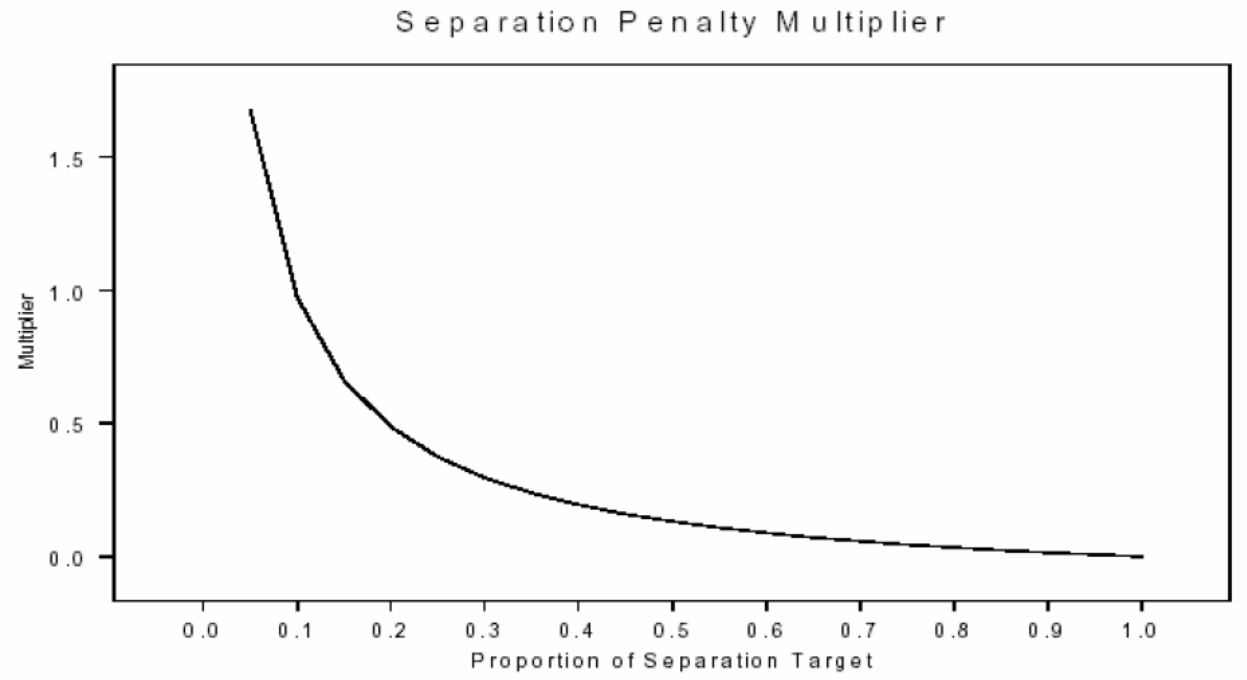
The separation rule is handled such that the algorithm looks through all the planning units for the given conservation feature and determines if there are enough of them at the required separation distance from each other. The minimum separation distance must be specified for each conservation feature which has a target for the minimum number of separated occurrences (Section 5.3.2 under "Target for Separation Feature Occurrences"). Marxan determines the number of planning units containing that feature, which are mutually separated by at least the specified straight-line distance. This number is called the separation count. If a conservation feature has a separation count lower than the target number of separated occurrences then a penalty is added. This penalty is multiplied against the base penalty for the conservation feature (the penalty applied when there is not enough of that

conservation feature in the reserve system) and then added to the conservation feature's overall penalty.

This separation penalty function is:

$$Penalty = \frac{1}{7 * C_p + 0.2} - \frac{1}{7.2}$$

where *Cp* is the separation count for the conservation feature as a fraction of the target number of separated occurrences (target / separation count). If the separation count is zero then the penalty is calculated based on a separation count of 1 / target separation count. The values 7 and 0.2 were chosen after experimentation to give a separation penalty multiplier applicable under a wide variety of conditions. The separation count increases from 1/target separation count to 1 following the curve depicted in graph below.



Separation Penalty Multiplier as a function of the proportion of the separation target met. Note that separation targets are normally low integers so only some values of the multiplier can be taken on.

B-1.5 Cost Threshold Penalty

Marxan is generally used to find a minimum cost reserve system. The Cost Threshold Penalty has been included in the objective function to make it possible to look at a reverse version of the problem, i.e. find the reserve system which has the best representation for all

conservation features constrained by a maximum cost for the reserve system. This is generally referred to as a maximum coverage problem. Because this is philosophically a very different problem, the cost threshold penalty is an attempt to tackle this problem within the existing minimum set framework. Simply adding a cost threshold to the objective function does not mean that Marxan will now optimally solve a maximum coverage problem.

The Cost Threshold Penalty works by applying a penalty to the objective function if the total cost of the system has risen above the desired threshold. The threshold is based on the cost of the system only and does not include costs associated with boundary length. The value added to the objective function is calculated as the amount by which the threshold has been exceeded, multiplied by the cost threshold penalty. The penalty depends upon the stage of the annealing algorithm (i.e. how far into the annealing process the system is given as a proportion), and is calculated as follows:

$$Cost\ Threshold\ Penalty = (amount\ over\ threshold)x(Ae^{bt} - A)$$

where t is the time during the run, and must be between 0 (start of the run) and 1 (end of the run). The penalty always starts at 0 when t is zero.

The term b is a control parameters which controls the shape of the penalty curve; i.e, how gradually the penalty is applied (if it is set high, the penalty will vary little until late in the run).

A is another control parameter that controls the size of the penalty. If A is set high, exceeding the threshold will be penalized very heavily, and if A is set lower it might allow the threshold to be slightly exceeded.

The values for both A and b will require some experimentation to set appropriately and both can be modified in the 'input.dat' file. They are listed as 'THRESHPEN1' and 'THRESHPEN2'. The proper use of this option requires considerable experimentation and is recommended for advanced users only.

The way the Cost Threshold Penalty influences the running of Marxan depends upon which optimisation method is used. If Simulated Annealing is being used, the system will be allowed to go above the cost threshold and the penalty will drive it back down in the end. The application of the penalty will vary according to the parameters described above. This will not limit too many options and thus allow Marxan to still find reasonably efficient solutions. If on the other hand, a basic heuristic algorithm or iterative improvement is being used, Marxan will simply stop adding planning units to the system when the threshold is reached. If iterative improvement is used following simulated annealing, Marxan will simply remove planning units until the threshold is reached. Both of these scenarios are likely to result in quite sub-optimal solutions.

B-2 Optimisation Methods

As described in this manual, Marxan can use a number of different algorithms to try to improve the objective function value of potential reserve systems. The principal and most powerful of these is Simulated Annealing. Marxan also provides the option to use common, less sophisticated heuristics, and even the very basic iterative improvement techniques. It is also possible to use these methods in combination with each other, for instance, we generally recommend using simulated annealing followed by iterative improvement. In this section of the appendix we describe the technical details of how each of the different methods work and what parameters can be used to control their functioning.

B-2.1 Simulated Annealing

Simulated annealing is based on iterative improvement but with stochastic (random) acceptance of bad moves to help avoid getting stuck prematurely at local minimum objective function value. A local minimum occurs at the point where simply adding one favourable planning unit or removing one unfavourable planning unit from a reserve system can no longer improve the objective function value. Such local minimum may well occur at an objective function value that is a long way from the true optima.

Simulated annealing derives its name from a technique in metallurgy involving the heating and controlled cooling of a material to reduce defects. Initially high temperatures cause atoms to become unstuck and to move randomly. Slow cooling then increases the chance of the atoms finding configurations with fewer defects. By analogy, efficiency is achieved in a conservation area network whereby changes that apply additional costs in the conservation area network may be tolerated early in the selection process; however, as the process continues the temperature is cooled and only positive or effective changes in portfolio design are accepted. This allows the algorithm to escape local minima in early sampling rounds and the progressive refinement into efficient solutions in later sampling rounds. Another useful analogy is to imagine the solutions as a mountain range with the optimal solution being the top of the tallest mountain. If you start your climb towards the tallest peak at the base of the range but are only ever allowed to go up, you will quickly get stuck at the top of one of the foot hills (equivalent of a local minimum). If you are instead, allowed to go down sometimes in order to cross valleys, you will ultimately be able to reach higher peaks.

In Marxan, the simulated annealing procedure will run for a user-defined number of iterations. At each iteration, a planning unit is chosen at random and may or may not be already in the reserve system. The change to the objective function's value of the reserve system, which would occur if this planning unit were added or removed from the system, is evaluated. 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 can be accepted or rejected. 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 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 that can be used in Marxan. 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 Marxan samples the problem and sets the initial temperature and rate of temperature decrease based upon its sampling.

B-2.1.1 Adaptive Annealing Schedule

The adaptive annealing option provides an easy and relatively rapid way to set the parameters necessary to run simulated annealing and requires little in the way of pre-analysis by the user. For this reason it should be favoured by first time users or those after a quick indication of possible solutions. Even for experienced users adaptive annealing will be useful for broad investigations, tests and trials on the system which would precede the more careful and detailed use of a fixed schedule annealing algorithm.

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

$$Tinitial = MinChange + 0.1\ x\ (MaxChange - MinChange) \qquad (6)$$

This is based upon the adaptive schedule in (Conolly 1990). Here, *Tinitial* 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 (i.e. a positive value).

The use of the adaptive annealing schedule can be applied by setting the variable 'STARTTEMP', to any negative value in the 'input.dat' file.

B-2.1.2 Fixed Annealing Schedule

With fixed schedule annealing the parameters that control the annealing schedule are fixed by the user for each implementation of the algorithm. This is done typically by trials of the algorithm with different parameters. Trials should include looking at final results and also tracking the progress of individual runs. The annealing schedules which arise from well trialed fixed schedule processes are generally superior to the adaptive annealing schedule, and the processing time will be faster as there is much less in the way of initial runs. It does, however, require some skill to set. For this reason it is examined in detail here.

First, set the "verbosity" of the algorithm to "Detailed Progress" (see Section 7.3.4) so that you can see the simulated annealing at work.

When setting a fixed annealing schedule the two parameters that need to be set 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 "stuck" 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 refining work will not be completed and the reserve system will largely be delivered by the iterative improvement schedule if it follows simulated annealing. As iterative improvement will only ever find nearby local minima, this is unlikely to be a particularly good solution. If the initial temperature is too high then the system will spend too much time at high temperatures, accepting bad moves, and less time where most of the annealing work is to be done.

The best way to get a general feel for what the two parameters should be is to run the algorithm with many different values. Look at 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 gives estimates of what reasonable initial temperatures might be.

Tests can be run looking at the final output from multiple runs and different parameters but with much shorter numbers of iterations. However, these short runs can only show you how things start out. To see the end (or "tail") the full number of iterations will be required. Once good values have been found with a small number of iterations, they need to be scaled up for the larger numbers 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. The best way to proceed 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 solution 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 (more iterations) it is advantageous to increase the range of temperatures used.

Ultimately, however, most users find themselves tweaking both the initial and final temperatures to best suit the full number of iterations.

If you wish to use a fixed annealing schedule, both the initial temperature and the cooling factor (which controls the final temperatures), can be set in the 'input.dat' file using the variable names, 'STARTTE MP' and 'COOLFAC' (see Section 5.3.1).

B-2.2 Iterative Improvement

Iterative improvement is a simple optimization method. It is not very powerful and it makes little sense to run it on its own but can be profitably used to aid the results of simulated annealing.

There are three basic types of iterative improvement which can be used in Marxan. 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 reserve system with some, all, or no planning units contained in the system. 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 further 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 reserve system is therefore at a local optimum (which may or may not be particularly good overall, depending on what preceded the iterative improvement).

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 reserve system. 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 reserve system then the changes considered are removing that planning unit but adding another one somewhere else. If the chosen planning unit is not in the reserve system then the changes considered are adding this to the reserve system but removing one that is already in the system. Possible 'swaps' are considered in random order, until one is found which will improve the system. This process is continued until all possible

swaps have been explored. Because this number 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, thus more time-consuming still. This method is only tractable with smaller numbers of planning units.

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 is much faster than running the 'two step' method on its own.

To implement any of these four different iterative improvement options in a Marxan run, they can be specified directly in the 'input.dat' file using the variable 'ITIMTYPE' (see Section 5.3.1).

B-2.3 Other Heuristic Algorithms

Heuristic is the general term for a class of algorithms which has historically been applied to the reserve selection problem. They spring from an attempt to automate the process of reserve selection by copying the way in which a person might choose reserves 'by hand'.

There are three main types of heuristics algorithms. They are the greedy heuristic, the rarity heuristic, and irreplaceability heuristic. All of the heuristics add planning units to the reserve system sequentially. They start with an empty reserve system and then add planning units to the system until some stopping criteria is reached. The stopping criteria are always that no unreserved site will improve the reserve system, but, as will be seen, the definition has two slightly different meanings. The heuristics can be followed by an iterative improvement algorithm in order to make sure that none of the planning units added have been rendered superfluous by later additions.

The pure greedy heuristic is relatively easy for non-technical audiences to understand and is often good to use as an example starting point. Although this is not the best heuristic it is robust and will work with problems of high complexity and is based on the simple concept of iteratively adding the planning units which improve the objective function the most. Use of the various other heuristics is recommended for the generation of fast solutions to explore design ideas quickly, although not efficiently. For example, the various "greedy" solutions

(which are easy to understand) can later be compared with solutions derived through simulated annealing, which in general will be superior (though harder to understand).

B-2.3.1 Greedy Heuristics

Greedy heuristics are those which attempt to improve a reserve system as quickly as possible. The heuristic adds whichever site has the most unrepresented conservation features in it. This heuristic is usually called the richness heuristic and the site with the most unrepresented conservation feature in it is the richest site. It has the advantage of making immediate inroads to the successful representation of all conservation features (or at least of improving the objective score) and it works reasonably well.

The output from a Greedy Heuristic not only includes a list of planning units that make up a reserve system, but also an order of priority for these planning units. This priority is based on the order in which planning units were added to the solution. This may be useful if there are not sufficient resources to obtain or set aside the entire reserve system at a given point of time. In such cases you can conserve areas in order of priority and the resultant reserve system will still be good relative to its cost. From this perspective they can be quite helpful.

Greedy heuristics can be further divided according to the objective function which they use. The two used in Marxan have been called the Richness and the Pure Greedy heuristic. These are described below.

Richness

When using this heuristic each planning unit is given two scores; the conservation value of the planning unit and the cost of the planning unit. The cost is simply the specified cost for that planning unit plus the potential change in modified boundary length. The conservation value is the sum of the under-representativeness of the conservation features present in that planning unit. The under-representativeness of a feature is simply how much better represented the conservation feature would be if this planning unit were added to the system. If a conservation feature has already met its target then it does not contribute to this sum. The richness of the planning unit is the contribution it makes to representing all conservation features divided by its cost.

Pure Greedy

The pure greedy heuristic values planning units according to how they change the Marxan objective function. This is similar to, but not the same as, the richness heuristic. When using the conservation feature penalty system, which is used with simulated annealing, the pure greedy heuristic has a few differences from the richness algorithm. It might not continue until every conservation feature is represented. It might turn out that the benefit from

raising a conservation feature to its target representation is outweighed by the cost of the planning unit which it would have to add. The pure greedy algorithm employs the usual Marxan objective function, which allows it to look at the boundary length of the reserve system as well as other advanced consideration such as with regard to a conservation feature clumping and minimum separation rules.

B-2.3.2 Rarity Algorithms

The planning units chosen by greedy heuristics will often be driven by the presence of relatively common conservation features. The first planning units added are those which have a large number of conservation features, often resulting in the selection of conservation features that are fairly common in the data set. The rarity algorithms work on the concept that a reserve system should be designed around ensuring that the relatively rare conservation features are reserved first before focusing on the remaining, more common conservation features. In developing Marxan, many rarity algorithms were explored, although they all tend to work in a similar manner. The ones available for use in Marxan are called: Maximum Rarity, Best Rarity, Average Rarity and Summed Rarity.

The rarity of a conservation feature is the total amount of it across all planning units. For example, this may be the total amount of a particular vegetation type available in hectares. There is a potential problem here, as the rarities for different conservation categories could be of different orders of magnitude. This is circumvented in most of the following algorithms by using the ratio, abundance of a conservation feature in a planning unit divided by the overall rarity (or total amount) of the conservation feature. Because the abundance and the rarity of the conservation feature are of the same units, this produces a non dimensionalised value.

Maximum Rarity

This method scores each planning unit according to the formula:

$$\frac{Effective\ Abundance}{Rarity\ x\ Planning\ Unit\ Cost} \qquad (7)$$

is based upon the conservation feature in the planning unit which has the lowest rarity. The abundance is how much of that conservation feature is in the planning unit capped by the target of the conservation feature. For example, suppose that the planning unit's rarest species occurs over 1000 hectares across the entire system. It has a target of 150 hectares of which 100 has been met. In the planning unit there are 60 hectares of the conservation feature. The cost of the planning unit is 1 (including both stated cost and boundary length multiplied by the boundary length modifier). Then the effective abundance is 50 hectares (the extra 10 does not count against the target). And the measure is 50 / (1000 x 1) = 0.05

for this planning unit. Note that the maximum rarity is based upon the rarest species in the planning unit and that rarity on its own is a dimensioned value. For this reason the algorithm is expected to perform poorly where more than one type of conservation feature is in the data set (e.g. vegetation types and fauna species). After the maximum rarity value is calculated for each planning unit, the one with the highest value is added to the reserve system with ties being broken randomly.

This is based upon the conservation feature in the planning unit, which has the lowest rarity. The abundance is how much of that conservation feature is in the planning unit capped by the target of the conservation feature. For example, suppose that the planning unit's rarest species occurs over 1000 hectares across the entire system. It has a target of 150 hectares of which 100 has been met. In the planning unit there are60 hectares of the conservation feature. The cost of the planning unit is 1 (including both stated cost and boundary length multiplied by the boundary length modifier).Then the effective abundance is 50 hectares(the extra 10 does not count against the target). And the measure is 50 / (1000 x 1) = 0.05 for this planning unit. Note that the maximum rarity is based upon the rarest species in the planning unit and that rarity on its own is a dimensioned value. For this reason, the algorithm is expected to perform poorly where more than one type of conservation feature is in the data set (e.g., vegetation types and fauna species). After the maximum rarity value is calculated for each planning unit, the one with the highest value is added to the reserve system withties being broken randomly.

Best Rarity

Best rarity is very similar to the maximum rarity heuristic described above. The same Equation 7 is used. The difference between the two is that when using best rarity, the conservation feature upon which the value is based is the one which has the best Effective Abundance / Rarity ratio and not the one with the best rarity score. This avoids the dimensioning problem but otherwise works in a similar manner.

Summed Rarity

Summed rarity takes the sum of the (Effective Abundance/ Rarity) for each conservation feature in the planning unit and then further divides this sum by the cost of the planning unit. Thus, there is an element of both richness and rarity in this measure. Here it is possible for a planning unit with many conservation features in it to score higher than one with a single, but rare, conservation feature. The formula used is:

$$\frac{\sum_{Cons.Feats} \frac{Effective\ Abundance}{Rarity}}{Planning\ Unit\ Cost} \qquad (8)$$

Average Rarity

Average rarity is the same as summed rarity except that the sum is divided by the cost multiplied by the number of conservation features represented in the planning unit. Through dividing by this number, the heuristic will tend to apply greater weight to rarer conservation features. The formula used is:

$$\frac{\sum_{Cons.Feats} \frac{Effective\ Abundance}{Rarity}}{Cost\ x\ Number\ of\ Conservation\ Features} \quad (9)$$

B-2.3.3 Irreplaceability

Irreplaceability captures some of the ideas of the rarity and greediness heuristics. Irreplaceability works by looking at how necessary each planning unit is to achieve the target fora given conservation feature. This is based on the idea of a conservation feature buffer. The buffer is the total amount of a conservation feature minus the target for that conservation feature. If the target is as large as the total amount then it has a buffer of zero and every planning unit which holds that conservation feature is necessary. The irreplaceability of a planning unit for a particular conservation feature is:

$$Irreplaceability \begin{cases} \frac{Buffer - Effective\ Abundance}{Buffer}, Buffer > 0 \\ Irreplaceability \{0, \quad Buffer = 0 \end{cases} \quad (10)$$

Note that if the buffer is zero then its irreplaceability is 0. If a planning unit is essential it will also have an irreplaceability value of 0. A value close to 1 indicates that the planning unit is not really needed. There are two ways in which this measure is used: Product Irreplaceability and Summed Irreplaceability.

Product

The irreplaceability for each conservation feature is multiplied together to give a value for a planning unit between 0 and 1, with 0 meaning that the planning unit is essential for one or more conservation features. This number is subtracted from 1 so that a high value is better than a low value. The value of this product cannot be higher than the value for an individual conservation feature and as such it is very sensitive to outliers. It is similar to the rarity heuristics in that it will tend to select planning units based on their holdings of hard-to-represent conservation features.

Summed Irreplaceability

When using this heuristic, irreplaceability is subtracted from 1 to produce a value between 0 and 1 where a high valued planning unit is necessary for conservation purposes and a low valued one isn't. These values are summed across all conservation features. As such, it is less sensitive to outliers or weak data. This summation means that the quantity of conservation features is important and it is related to the product irreplaceability heuristic in the same way that the summed rarity heuristic relates to the best rarity heuristic

MARXAN

conservation solutions

