Marxan User Manual

For Marxan version 1.8.10

Written by Edward T. Game and Hedley S. Grantham

Edited by Jeff Ardron, Carissa Klein, Dave Nicolson, Hugh Possingham and Matt Watts

Issued: February 2008

Issued jointly by:

Applied Environmental Decision Analysis Facility
The Ecology Centre
The University of Queensland
St Lucia, Queensland, Australia

Pacific Marine Analysis and Research Association (PacMARA)
Vancouver, British Columbia, Canada

Suggested citation:

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

(This page intentionally blank)

Preface

Scope and Aims

This manual is intended to equip readers with the basic knowledge required to use Marxan. We cover all the relevant parameters and 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 then the theory of reserve system design and the optimization algorithms that solve the reserve system design problem. Some of this information is available in the appendices, but in other cases we attempt to direct readers to the appropriate source. We provide some guidance to the sorts of problems Marxan can solve but in reality only your imagination limits the way it can be applied. It is important, however, to understand how Marxan works in order to avoid solving the wrong problem or misinterpreting the solutions you find. Marxan can be a very powerful tool, but if misused it can undermine a great deal of hard work in collecting and collating good data, not to mention providing misleading advice and undermining the credibility of systematic conservation planning software.

Complementary literature

Where this manual seems light on detail or lacking in specific direction, additional information can almost certainly be found in the Marxan Good Practices Handbook (MGPH), which is expected to be available for download in 2008. This manual and the MGPH should be used in concert with each other, and together should provide 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 various 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. We provide details for some of these publications in the Key References section.

Acknowledgements

Preparation of this version 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, 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. Appendix C was largely based on the Marxan 101 course run by the University of Queensland, written by Carissa Klein and Matthew Watts. Details of this and other courses can be found at: http://www.ecology.uq.edu.au/marxan.htm.

Some of the introductory paragraphs have been modified from chapter 1 of the MGPH 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

List of Tables:

1. Introduction

1.1 What is Marxan?

Marxan is software that delivers decision support for reserve system design.[1] 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. They may wish 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 particular 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 is that cheaper or less socially disruptive reserve networks are more likely to be implemented. Furthermore, meeting a set of targets for all conservation features provides a solid platform for expanding a reserve system in the future; reserve systems biased to habitats of little commercial value are often hard to expand. In minimum set problems the elements of biodiversity that you wish to conserve are entered as constraints to solutions of the problem (Possingham et al 2000). Given reasonably comprehensive data on species, habitats and/or other relevant biodiversity features, Marxan aims to identify the reserve system (a combination of planning units) that will meet user-defined biodiversity targets[2] for the minimum cost (Ball and Possingham 2000; Possingham et al 2000).

As an example, a possible biodiversity target could be to ensure that at least 30% of the abundance of every vegetation type is represented in a protected area network. If this protection must be achieved through the purchase of land, then a conservation planner (and politicians) will probably desire a system of reserves that minimises the total monetary cost required for purchasing the necessary land and meeting those targets (Carwardine et al. 2006). Where information on the actual cost of land is not available, reserve area might be used as a surrogate for cost, based on the assumption that the larger the entire reserve system the more costly it will be to implement and manage (although this is not always the case). The cost used in Marxan can also be any relative social, economic or ecological measure of costs, or combination thereof.

[1] Though it can be used for other purposes as well, as noted below in section 1.3.
[2] We use the term 'target' to refer not to species or features present in the planning region, but to the desired representation of these inside a reserve system.

The number of possible solutions to even a small reserve selection problem is vast (for only a modest number of 200 planning units there are over 1.6×10^{60} possible ways a reserve system could be configured). Because finding the best solution to this problem is complex and time consuming, computer algorithms have been developed. An algorithm is a mathematical process or set of rules used for problem solving. Two general types of reserve design tools have been devised to efficiently solve "reserve design" problems: exact algorithms and heuristic (non-exact) algorithms. Although exact algorithms can identify the single optimal solution, for large reserve design problems it is difficult (and often impossible) to find this solution in a reasonable amount of time (Possingham et al. 2000; Cabeza 2003). Heuristics, on the other hand, provide a number of good, near-optimal solutions, which not only provide a set of options for planners and stakeholders to consider but can also be generated very quickly (Possingham et al. 2000; McDonnell et al. 2002; Cabeza 2003). As a result, heuristics are generally preferred over exact algorithms. Marxan is able to find a range of near-optimal solutions quickly (even for very large planning problems), using a powerful heuristic known as 'simulated annealing' (Appendix B-2.1). Simulated annealing will generally get much closer to the optimal solution than other heuristics such as the Greedy Heuristic, (Appendix B-2.3.1). If desired, Marxan is also able to find solutions using a variety of less sophisticated, but often faster, heuristic algorithms (see Section 3.2.1.2.1). Marxan is part of a lineage of reserve design tools including its direct predecessor, SPEXAN.

1.1.1 Other versions of Marxan

In this manual we only describe Marxan version 1.8.10., the classic Marxan software. There are, however, a number of Marxan variations with modified functionalities that are either available or in development from The Ecology Centre at The University of Queensland (http://www.ecology.uq.edu.au). These include: an optimised version of Marxan for handling very large problems involving greater than 20,000 planning units (people have successfully found solutions to problems with half a million planning units); a version that allows probabilistic information on threats or the presence of conservation features at sites to be included in the reserve design problem; and Marxan with Zones, which is being developed to handle multiple objective zoning. Although the basic operation of these versions is the same as described in this manual, they each have idiosyncrasies which will be described in an appendix provided with each of the versions. While Marxan and its variations are freely downloadable, their development has always been partially dependent on external funding bodies who have stepped forward to support this work.

1.2 Systematic Conservation Planning

Systematic conservation planning is widely considered 'best practice' in conservation because it facilitates a transparent, inclusive and defensible decision making process. Transparency refers to how well people understand the decision-making procedures and output products. A highly transparent planning process will tend to increase the accountability and credibility of conservation planning and decision-making. Inclusive planning processes aim to incorporate information and values from stakeholders to reduce conflicts amongst interests. This, in turn, results in stronger, more widely accepted decisions. Defensibility is derived from the ability of systematic plans to explicitly consider how well a particular selection of reserves meets its objectives, and the validity of the reasoning used to get there. The MGPH discusses each of these principals in detail.

Although Marxan can be used for a variety of purposes at a variety of stages in the systematic conservation planning process, it was designed primarily to help inform the selection of new conservation areas for minimal "cost" and facilitate the exploration of trade-offs between conservation and socio-economic objectives. Marxan can help set priorities for conservation action by highlighting those places that are likely to be important inclusions in an efficient reserve network. Marxan can also be employed as a tool for evaluating the representation and comprehensiveness of existing reserve networks (Stewart et al. 2003).

It is important to understand that the appropriate role for Marxan, as with other decision support software, is to support decision making. Marxan solutions can form the basis of discussions towards a final plan that incorporates additional political, socio-economic and pragmatic factors. Some of the limitations to the use of Marxan are described in Section 1.4 below.

1.3 Questions Marxan can help answer

The backbone of Marxan is to facilitate the efficient selection of subsets from a large set of mapped, spatially constant features. While Marxan 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 planning challenges. Marxan can generally assist all problems related to the spatially-explicit selection of 'minimum sets'. For example, it has been used to identify a spatially efficient suite of "fishing areas" (Ban, Personal Communication); and in the field of coastal and marine natural resource management, Marxan has been employed to support multiple-use zoning plans that balance the varied interests of

fisheries, transportation and conservation, amongst others (e.g. Fernandes et al. 2005). Chan et al. (2006) have explored the use of Marxan in achieving ecosystem service, as well as biodiversity targets. Some of these applications will require a more creative use of Marxan than we have space to provide guidance on here. We suggest that once you are familiar with Marxan's basic operation you seek out some of the many published examples of Marxan in operation, as well as consulting the MGPH.

1.4 Limitations of Marxan

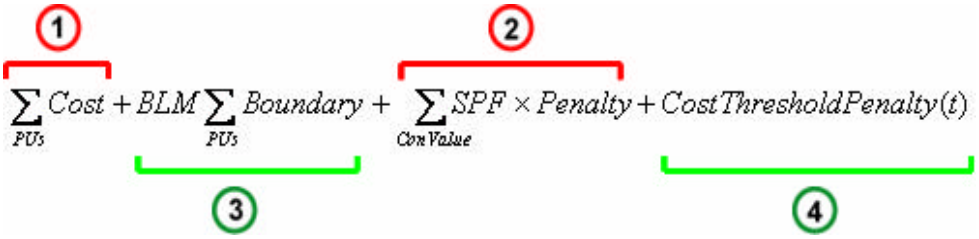
Technical limitations of Marxan will become apparent as you read this manual and in many cases can be overcome through data or scenario manipulation (for examples see the MGPH). More subtle and yet more important, however, are the philosophical limitations of reserve design software. These should be 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.

1.5 The Objective Function

In order for Marxan to find good solutions to a problem it must have some basis by which to compare alternate solutions (i.e. collections of planning units) and hence identify good ones. This is achieved through the use of a mathematical objective function that gives a value for a collection of planning units based on the various costs of the selected set and the penalties for not meeting conservation (or other) targets. Thus, a solution containing zero planning units, though cheap to implement, would probably not meet any biodiversity goals and so the objective function value should be very poor. Having an objective function which gives any possible reserve system a value, allows us to automate the selection of good reserve networks (at least according to the objective function). Marxan works simply by continually testing alternate selections of planning units, aiming to improve the whole reserve system value.

The objective function's value must of course reflect the desirability of that particular reserve system. In its simplest form, the Marxan objective function is a combination of the total cost of the reserve system and a penalty for any of the ecological targets that are not met. This objective function is designed so that the lower the value the better. Marxan also allows a measure of reserve system fragmentation to be taken into account, so that it will generally be desirable for a reserve system not to be too fragmented. Not only will a fragmented reserve system often lead to undesirable fragmentation of ecological communities, it is also likely to make management and compliance more challenging and costly. A more fragmented reserve network will have a greater overall boundary length. It is this boundary length, plus a weighting on its importance relative to the other components of the objective (cost and meeting targets), that can be included in the objective function. The final possible addition to the objective function is a penalty for exceeding some pre-set cost.[3] Although Marxan always tries to find the cheapest satisfactory reserve network, there may occasionally (or frequently) be immoveable fiscal constraints on conservation actions. In these cases we want to ensure the best solutions, given the available budget, will be found.

Thus, the objective function in Marxan takes the form:



1. The total cost of the reserve network (required)
2. The penalty for not adequately representing conservation features (required)
3. The total reserve boundary length, multiplied by a modifier (optional)
4. The penalty for exceeding a preset cost threshold (optional – see footnote 3)

Terms one and three can be thought of as 'costs', whereas terms two and four are penalties for breaching various criteria. In general we do not advise using the cost threshold penalty. More detail on the objective function and how each of the different terms is calculated can be found in Appendix B-1. Section 3 of this manual contains details of how to control which features contribute to the objective function and what the size of the penalties will be.

[3] Due to sometimes inconsistent results, the Cost Threshold Penalty feature of Marxan is currently being re-programmed. Users of Marxan 1.8.10 should be aware of this. It is recommended that this function be used carefully.

1.6 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 subtleties, such as knowledge about data quality, 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 3.2.3.3. When it is believed that data may be spatially biased, this bias should be documented.

Marxan does not consider uncertainty in the data.[4] 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 MGPH suggests some methods for conducting robust analysis using weaker datasets.

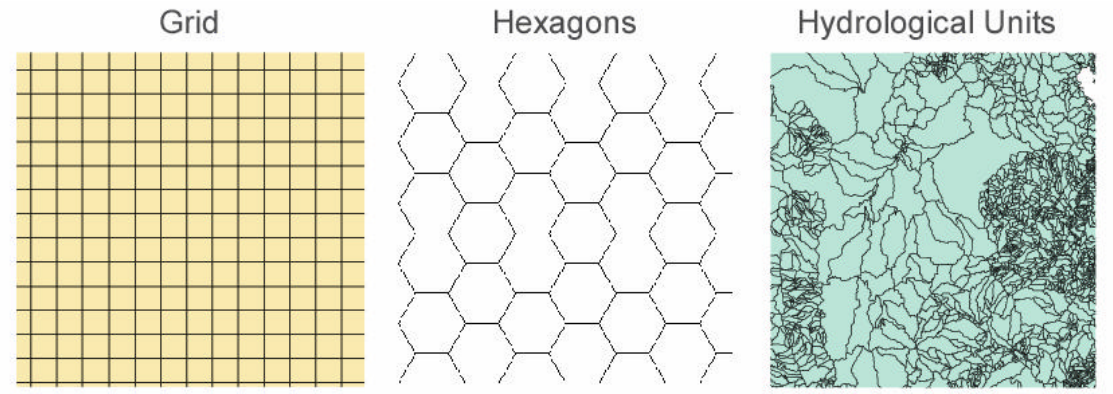
[4] Versions of Marxan under development have some ability to deal with levels of uncertainty.

1.7 Pre-processing of data

Actually running Marxan to generate reserve solutions will generally be the quick part of a conservation planning exercise! Before that, a number of often time-consuming steps must be completed.

1.7.1 Choosing planning units

An essential pre-processing step is to divide your planning region into a set of planning units. A tutorial on some methods to create your planning units is provided in Appendix C-2. In their simplest form, planning units may be defined by overlaying your planning region with a grid of squares or lattice of hexagons. They must capture all the areas that can possibly be selected as part of the reserve system and their size should be at a scale appropriate for both the ecological features you wish to capture and the size of the protected areas likely to be implemented. In general, they should be no finer in resolution than the data on conservation features and no coarser than is realistic for management decisions. There is, however, no necessity to have uniformly shaped planning units. Nor is it always true that smaller planning units are better. In some cases it will make more sense to have planning units that are informed by natural ecological divisions such as hydrological units, or even by political/governmental divisions such as cadastral parcels. For other uses, a uniform planning unit will provide more useful results.



Three possible types of planning units that could be used in Marxan.

There is a limit 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 conservation features you wish to plan for and even to some extent on the power of your computer. Unfortunately we know of no good rule of thumb for assessing this number but we have quite comfortably run Marxan analyses with 10,000 planning units and 100 conservation features. Really big analyses (i.e. >20,000 planning units) should be run

using the optimised version of Marxan (2.0.2.) also available from The Ecology Centre website (see Section 1.1.1). The number of planning units and features this version can handle are essentially limited only by available memory.

A great deal of care should be taken when deciding upon appropriate planning units, as it will influence the results of your Marxan analyses. References on this topic are provided in the Key References section. While seldom done, there is no reason why two analyses using different planning units could not be run.

1.7.2 Determining the distribution of conservation features

A second essential step prior to using Marxan is to determine the distribution of conservation features across your planning units. This means assembling all the requisite data on your conservation features and then calculating how much of each feature is located within each planning unit. To do this will generally require some knowledge of a geographical information system (GIS); a tutorial on one way to do this is provided in Appendix C. In most cases, compiling the necessary data and working out the representation of conservation features across your planning units is likely to involve greater effort than running Marxan. Project managers must be careful to allow sufficient time for this step.

2. Getting Started

2.1 System requirements

The system requirements for running Marxan are quite modest. Any Microsoft operating system will suffice, even a really old one. As a rule of thumb, if a computer is powerful enough to run commercial 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 you problem can range from minutes to days.[5]

2.2 Software installation

Chances are that if you are reading this manual you have already downloaded Marxan. If not, Marxan can be downloaded from http://www.ecology.uq.edu.au/marxan.htm. You will require around 2 MB of free disk space to install Marxan and the associated files.

When you download Marxan you will receive the following files:

1. Marxan.exe (the Marxan program executable)
2. Inedit.exe (a program that allows you to easily generate the Input Parameter File – the file that controls how Marxan works)
3. input.dat (an example Input Parameter File)
4. A folder labelled 'Sample', containing examples of the other input files used to run Marxan (the details of these files are explained later).
5. This manual

These files can be saved anywhere on the computer. For simplicity when running Marxan, the executable, 'Marxan.exe', should be located in the same folder as the input files for that project (see Section 3.1.2). Rather than continually move files around, we recommend simply copying the Marxan executable to each folder containing a Marxan project.

[5] 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.

2.3 Supporting Freeware

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. Many users have found these interfaces particularly helpful for generating appropriate input files and displaying Marxan outputs. Guidance on using these programs (described below) can be obtained from their websites or user manuals.

2.3.1 CLUZ (Conservation Land Use Zoning)

CLUZ is an ArcView GIS interface that links to Marxan. It was developed by Bob Smith at the Durrell Institute of Conservation and Ecology and is available from http://www.mosaic-conservation.org/cluz/. CLUZ provides a dynamic connection with Marxan so the user can easily run Marxan and map the results of Marxan runs. It also contains tools to help develop the input files required by Marxan. CLUZ comes with useful tutorial exercises to guide users through developing input files, modifying run parameters and displaying Marxan results.

2.3.2 P.A.N.D.A. (Protected Areas Network Design Application)

P.A.N.D.A. is a stand-alone application that uses the Visual Basic and ArcObjects software. It was developed by Francesca Riolo to provide ArcGIS users with a user friendly framework for systematic protected areas network design. ArcGIS is required to run the program. It is available from http://www.mappamondogis.it/panda_en.htm

2.3.3 C-Plan

C-Plan is conservation decision support software that links with GIS to map options for achieving explicit conservation targets. It was developed by Matt Watts and Bob Pressey. C-Plan allows users to decide which planning units should be placed under some form of conservation management through manual selection. It also allows automated selection using heuristic selection algorithms. The newer version, C-Plan 3.4, has a Marxan interface which can generate Marxan input files from C-Plan files. It also allows users to run Marxan from C-Plan and import Marxan outputs back into C-Plan to be displayed in GIS. C-Plan is available from http://www.uq.edu.au/~uqmwatts/cplan.html.

2.4 Overview of what is required to run Marxan

There are four main steps to running Marxan:

1. Setting up the input files
2. Setting the scenario parameters
3. Running Marxan
4. Interpreting the results

This section is intended to provide a brief guide to the essential steps of using Marxan. A more detailed treatment of each step follows in subsequent chapters. The successful use of Marxan will never involve a once-off, sequential application of these steps. Instead, in any given project these steps should be repeated numerous times, particularly the last three, and the results of each run 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. Suggestions of possible protocols are provided in the sections below. Once the input files have been set up it should be quite easy to modify the scenario, re-run Marxan and investigate the results.

Marxan is a decision support tool to help guide the selection of efficient reserve systems; its output should never be interpreted as "the answer." Although the results of a single Marxan run will represent a good solution to your reserve design problem, it will not necessarily be the preferred solution. Because Marxan tests the utility of planning units in a pseudo-random fashion (see Appendix B for more details), 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. It is possible to sample the range of different possible solutions (the solution space) by running Marxan many times.

(This page intentionally blank)

3. Input Files, Parameters and Variables

3.1 Introduction

To run Marxan you need a set of input files. These files contain all the data you wish to work with and the details of the conservation problem you wish Marxan to solve. Four input files are required, without them Marxan will not run. The required and optional files are summarized below;

Table 1: Marxan input files and default names.

Input File	Default Name	Required
Input Parameter File	input.dat	Yes
Conservation Feature File[6]	spec.dat	Yes
Planning Unit File	pu.dat	Yes
Planning Unit versus Conservation Feature File	puvspr2.dat	Yes
Boundary Length File	bound.dat	No
Block Definition File	blockdef.dat	No

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** 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** contains information on the distribution of conservation features in each of the planning units.

[6] Previously known as the 'Species File'.

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.

The **Block Definition File** is very similar to the **Conservation Feature File** and can be used to set a series of default variable values for groups of conservation features.

This section describes in detail each of these six potential input files; their function, format and the variables contained therein. More information about some potential ways to generate these files can be found in the tutorials contained in Appendix C.

Symbols for variables

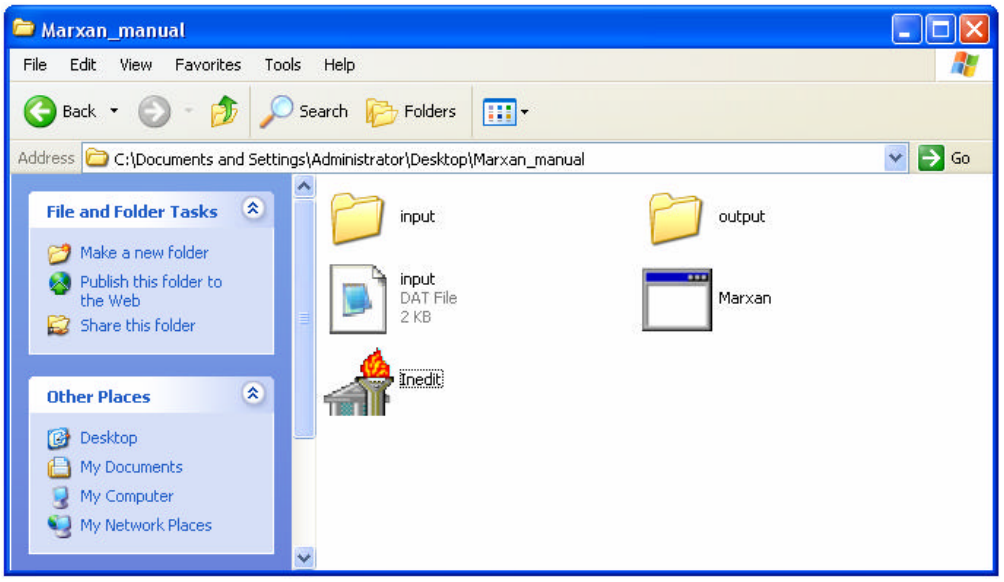
	Very important parameters – we have attempted to carefully describe their role here.
	Highly technical parameters – explanation is left predominantly to Appendix B.
	Easy mistakes that can have big consequences.

3.1.1 Input file types

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.

3.1.2 Input File management

All the input files except for the **Input Parameter File**, 'input.dat', 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.



An example of how we recommend a Marxan folder should be set up.

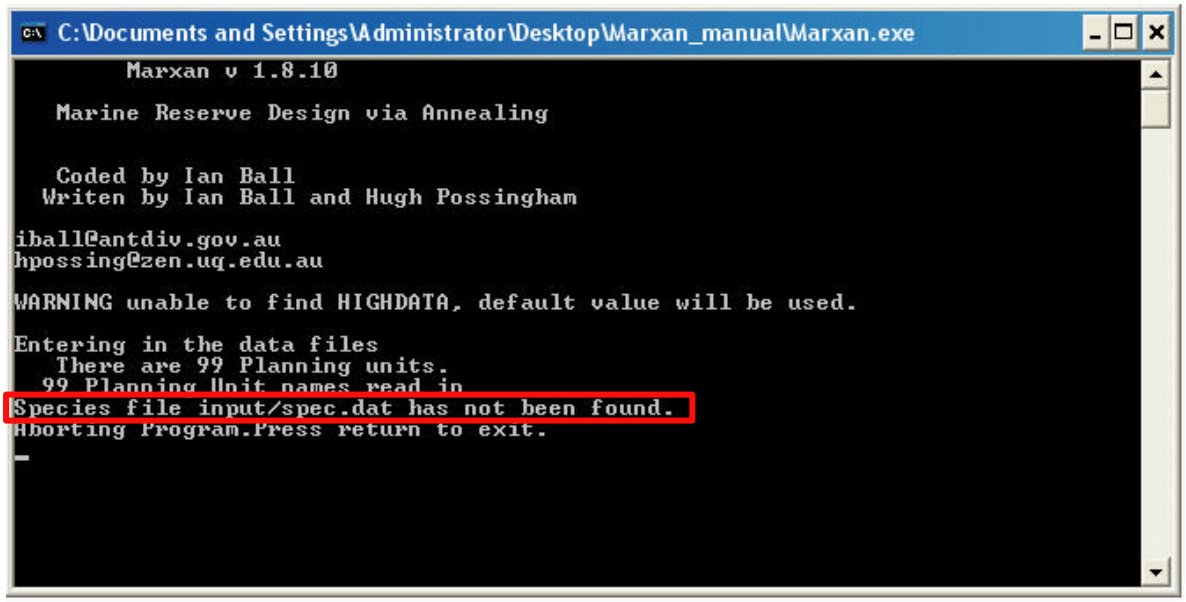
Throughout this manual we refer to the different input files by their default names. You can give any name to the input files, provided the names match those given in the **Input Parameter File**, and providing you use the correct procedure for starting Marxan. Using consistent file names, however, helps 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 this to get it to recognise the **Input Parameter File**: "Marxan.exe scenario1.dat"

If you 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"

3.2 Required files

If one of the four required files (see Table 1 in Section 3.1) is missing, Marxan will halt with an error message.



An example of the error message Marxan will display if it cannot find the necessary input files. In this case, the **Conservation Feature File** (spec.dat) is missing.

3.2.1 The Input Parameter File

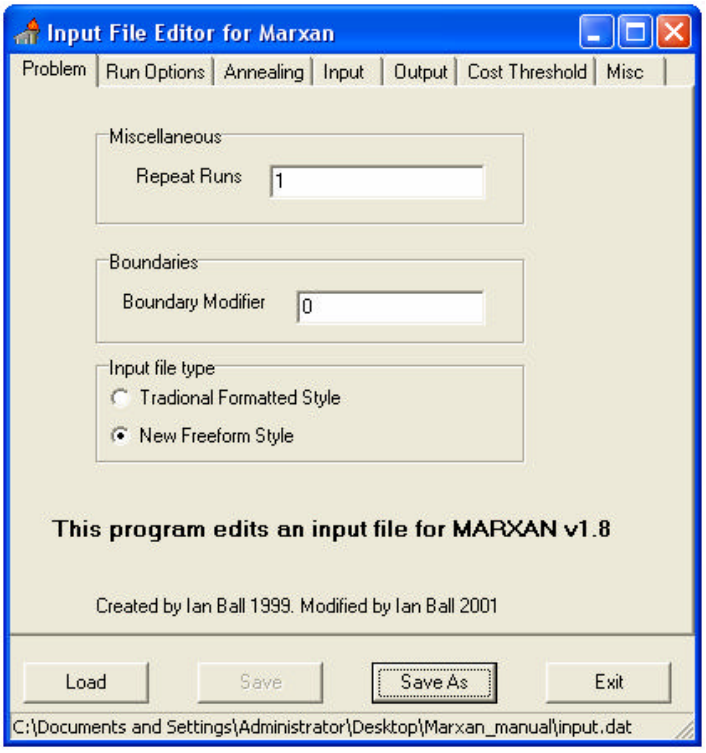
Marxan is an extremely flexible program. The flipside to this, however, is that you must set certain parameters to appropriately deal with the particular problem you wish to solve. This is primarily accomplished through the **Input Parameter File**. This file contains the principal parameters that control 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 either of the two optional input files, and what output files you want and where they should be placed.

Do not worry if the number of variables in this file seems a bit daunting; many of these variables will almost never need to be modified. A few, however, may be changed for nearly every scenario run.

There are two possible ways to create and modify the **Input Parameter File:**

1. Using the program, '**Inedit**', that comes bundled with Marxan (recommended)
2. Directly through a text file editor (i.e. Windows Notepad, etc. – for advanced users)

When creating this file for the first time it is generally best to use **Inedit**. This program provides a graphical user interface that takes you through each of the parameters that need to be set in the **Input Parameter File**.



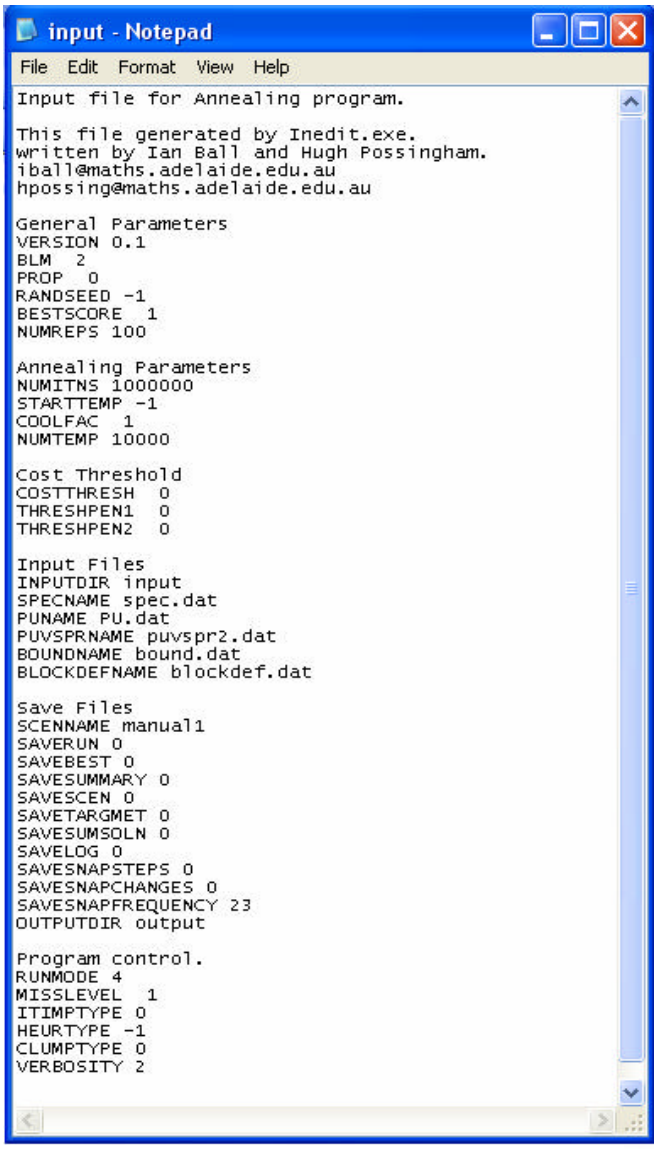
The **Inedit** program used to create the **Input Parameter File** (input.dat) used in Marxan.

Once you have set values for each of the parameters in **Inedit**, simply press 'Save' and the corresponding 'input.dat' file will be generated automatically. When you use the command, 'Save', the 'input.dat' file will be saved in the same folder as the **Inedit** program. If you use, 'Save As', you can select where the 'input.dat' file will be saved.

There are no universally best values for the parameters contained within the **Input Parameter 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 again, and the new results compared with the old ones. There are few short-cuts to this process. In sub-sections below we provide guidance

on how to determine appropriate values for each of the major parameters. As with creating the file, adjusting the parameters can be done either with **Inedit** or directly on the file with a text editor. Once it has been created for the first time using **Inedit** and users have a basic familiarisation of the file, it is often faster to make parameter changes directly in the 'input.dat' file. To do this you simply open the file using a text editor program such as Windows Notepad or equivalent, change the relevant parameter values and resave the file. To modify an existing **Input Parameter File** using **Inedit**, simply load the existing 'input.dat' file using the command, 'Load', change the parameters in the appropriate tabs and then resave the file.

The **Input Parameter File** has the following appearance:



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

Each variable is given as a single word in capital letters. 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 Parameter 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).

The following table contains a very brief description of each variable as well as their default values.

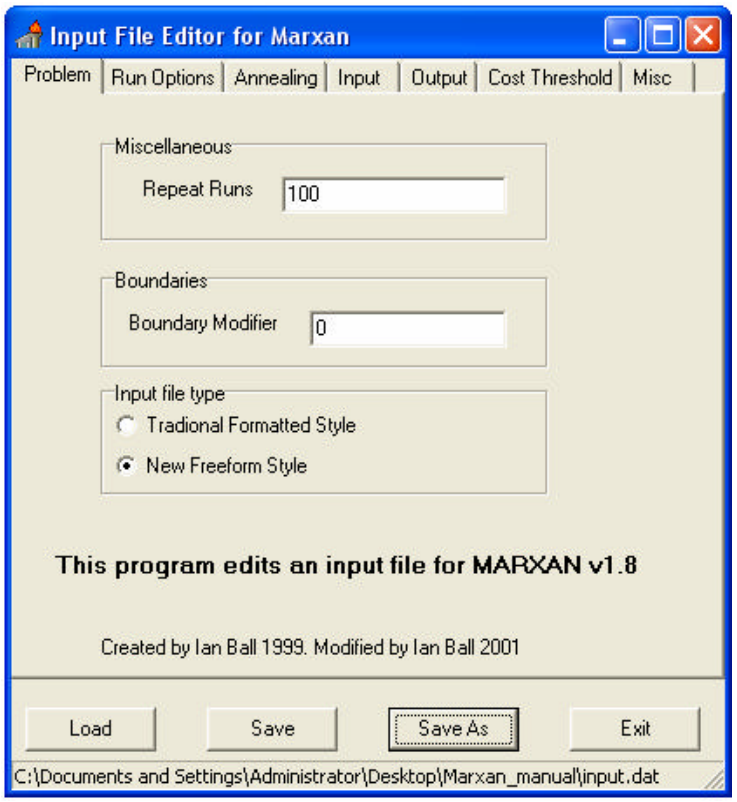
Table 2: Marxan names and default values.

Variable Name	Default Value	Description
VERSION	0.1	Type of input file
BLM	0	Boundary Length Modifier
PROP	0	Proportion of planning units in initial reserve system
RANDSEED	-1	Random seed number
BESTSCORE	0	Best score hint
NUMREPS	1	The number of repeat runs you wish to do
NUMITNS	0	Number of iterations for annealing
STARTTEMP	1	Starting temperature for annealing
COOLFAC	0	Cooling factor for annealing
NUMTEMP	1	Number of temperature decreases for annealing
COSTTHRESH	0	Cost threshold
THRESHPEN1	0	Size of cost threshold penalty
THRESHPEN2	0	Shape of cost threshold penalty
INPUTDIR	*User Defined*	Name of the folder containing input data files
SPECNAME	spec.dat	Name of Conservation Feature File
PUNAME	pu.dat	Name of Planning Unit File
PUVSPRNAME	puvspr2.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
SCENNAME	Temp	Scenario name for the saved output files
SAVERUN	0	Save each run? (0 = no)
SAVEBEST	0	Save the best run? (0 = no)
SAVESUM	0	Save summary information? (0 = no)
SAVESCEN	0	Save scenario information? (0 = no)
SAVETARGMET	0	Save targets met information? (0 = no)
SAVESUMSOLN	0	Save summed solution information? (0 = no)

SAVELOG	0	Save log files? (0 = no)
SAVESNAPSTEPS	0	Save snapshots each n steps (0 = no)
SAVESNAPCHANGES	0	Save snapshots after every n changes (0 = no)
SAVESNAPFREQUENCY	0	Frequency of snapshots if they are being used
OUTPUTDIR	*User Defined*	Name of the folder in which to save output files
RUNMODE	*User Defined*	The method Marxan uses to find solutions
MISSLEVEL	1	Amount or target below which it is counted as 'missing'
ITIMPTYPE	1	Iterative improvement type
HEURTYPE	1	Heuristic type
CLUMPTYPE	0	Clumping penalty type
VERBOSITY	1	Amount of output displayed on the program screen

In the sub-sections below each of these variables are described. We have divided the variables into groups based on which tab of the **Inedit** program they are modified on. The name of the tab is the same as the sub-section heading.

3.2.1.1 Problem



The 'Problem' tab of **Inedit**.

3.2.1.1.1 Repeat Runs

Variable – '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 5.3.7).

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. 5) 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. One hundred runs is probably a minimal value to start with and 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 MGPH provides some useful suggestions about determining the optimal number of runs.

3.2.1.1.2 Boundary Length Modifier

Variable – '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. 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 ones (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 a rather subjective value, the box below 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.

Setting the BLM

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

1. Keeping all other parameters the same, repeat the 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, 100000, 1000000.
2. In a spread sheet, record the BLM for each scenario, and the total reserve system boundary length and the average cost of solutions in adjacent columns. If reserve area is more important than cost, or is being used as a surrogate for cost, record the average reserve area of solutions.
3. Plot total reserve boundary length versus total cost/area for all the different BLM values, as shown below.

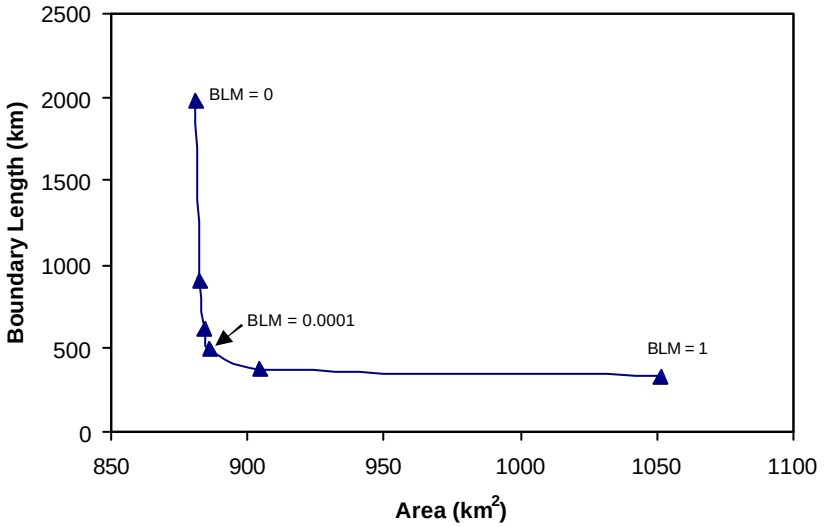


Fig 1 The trade-off between reserve system boundary length and the total are of the reserve system (modified from Stewart and Possingham 2005, figure 1).

The diminishing returns of increasing the BLM are clearly visible in Fig. 1. In this case a BLM of 0.0001 is the probably the most efficient. 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.

More information about the impact of spatial compactness and determining an efficient level of it can be found in Stewart and Possingham (2005). See Appendix B-1.2 for a detailed description of its role in the objective function. Also, more information on the application of the BLM can be found in the MGPH.

3.2.1.1.3 Input file type

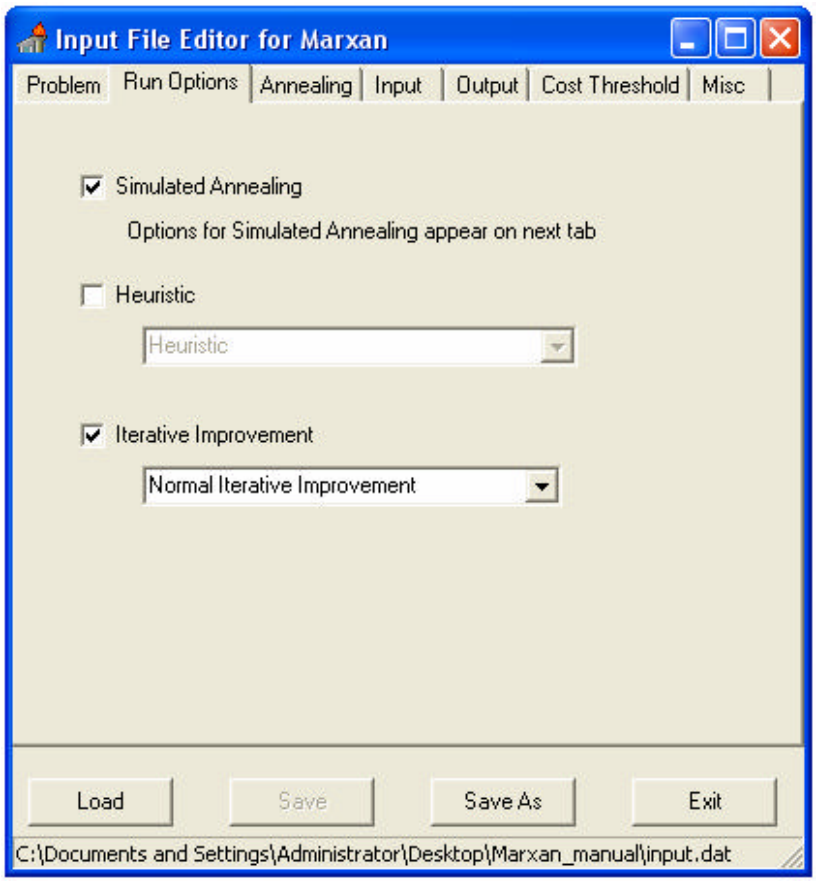
Variable – 'VERSION'
Required: Yes

Description: This variable is only provided to allow backwards compatibility with SPEXAN (a precursor to Marxan) files. The difference between the two file types is that Marxan can read variable values based on their name whereas SPEXAN used their position in the file, requiring strict formatting.

Getting Started: Always select 'New Freeform Style' in **Inedit**, or '0.1' in the 'input.dat' file[7]

[7] As Marxan has a number of improvements over SPEXAN, there is no benefit in using SPEXAN, unless replicating work that used it in the past.

3.2.1.2 Run Options



The 'Run Options' tab of **Inedit**.

3.2.1.2.1 Run Options

Variable – '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. Marxan, however, is also capable of using simpler, but more rapid, methods to locate potential solutions, such as heuristic rules and iterative improvement (see Appendix B-2.2 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

Although each of the above running combinations can be set with a single number in the 'input.dat' file, all three fields on the 'Run Options' tab of **Inedit** are needed to define it. For instance, if you wanted to select Simulated Annealing followed by Iterative Improvement, you need to first tick the 'Simulated Annealing' box, ensure the 'Heuristic' box is blank, and then tick the 'Iterative Improvement' box.

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.

3.2.1.2.2 Iterative Improvement

Variable – '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: To specify these in **Inedit** you use the drop down box labelled 'Iterative Improvement'. The default for this variable is Two Step Iterative Improvement, and for most scenarios this will be fine.

3.2.1.2.3 Heuristic

Variable – '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-2.3.

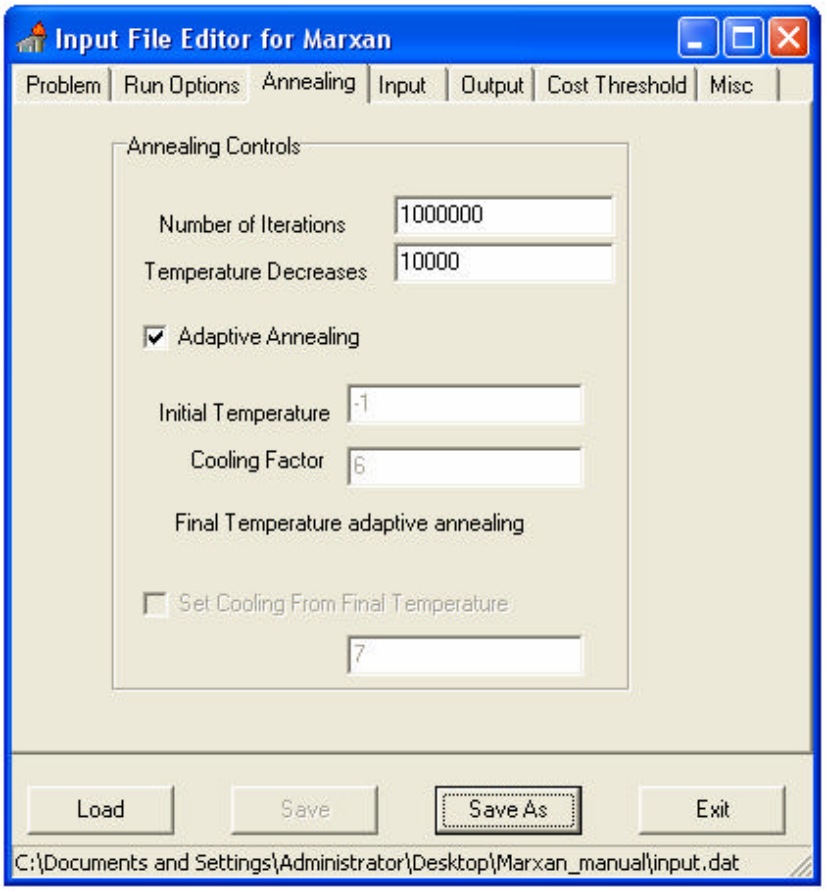
0	Richness
1	Greedy
2	Max Rarity
3	Best Rarity
4	Average Rarity
5	Sum Rarity
6	Product Irreplaceability
7	Summation Irreplaceability

Getting Started: To specify these in **Inedit** you use the drop down box labelled 'Heuristic'. However, we recommend that beginners use simulated annealing initially.

3.2.1.3 Annealing

3.2.1.3.1 Number of Iterations, Temperature Decreases, Initial Temperature and Cooling Factor

Variables – 'NUMITNS', 'STARTTEMP', 'COOLFAC', and 'NUMTEMP'
Required: yes (when using Simulated Annealing)



The 'Annealing' tab of **Inedit**.

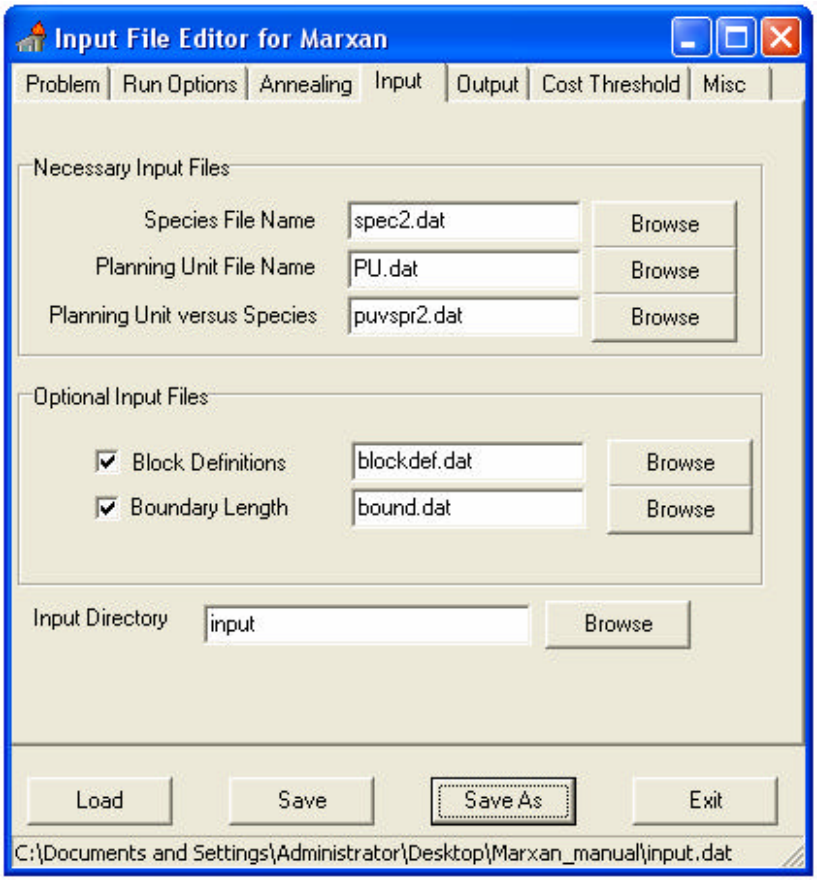
Description: These four variables control the way the Simulated Annealing algorithm proceeds. They will come into play when Simulated Annealing is chosen in the 'RUNMODE' (see Section 3.2.1.2.1). As they are quite technical and a detailed understanding is not necessary to successfully use Marxan, their explanation is left to Appendix B-2.1.

Getting Started: In practice you will rarely need to adjust most of these. The variables, 'STARTTEMP' and 'COOLFAC', will be set appropriately for you simply by ticking the **Inedit** box labelled 'Adaptive Annealing', and 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).

3.2.1.4 Input

3.2.1.4.1 Species File Name, Planning Unit File Name, Planning Unit versus Species, Block Definitions, Boundary Length and Input Folder

Variables – 'INPUTDIR', 'SPECNAME', 'PUNAME', 'PUVSPRNAME', 'BOUNDNAME', and 'BLOCKDEFNAME'
Required: Yes



The 'Input tab of **Inedit**.

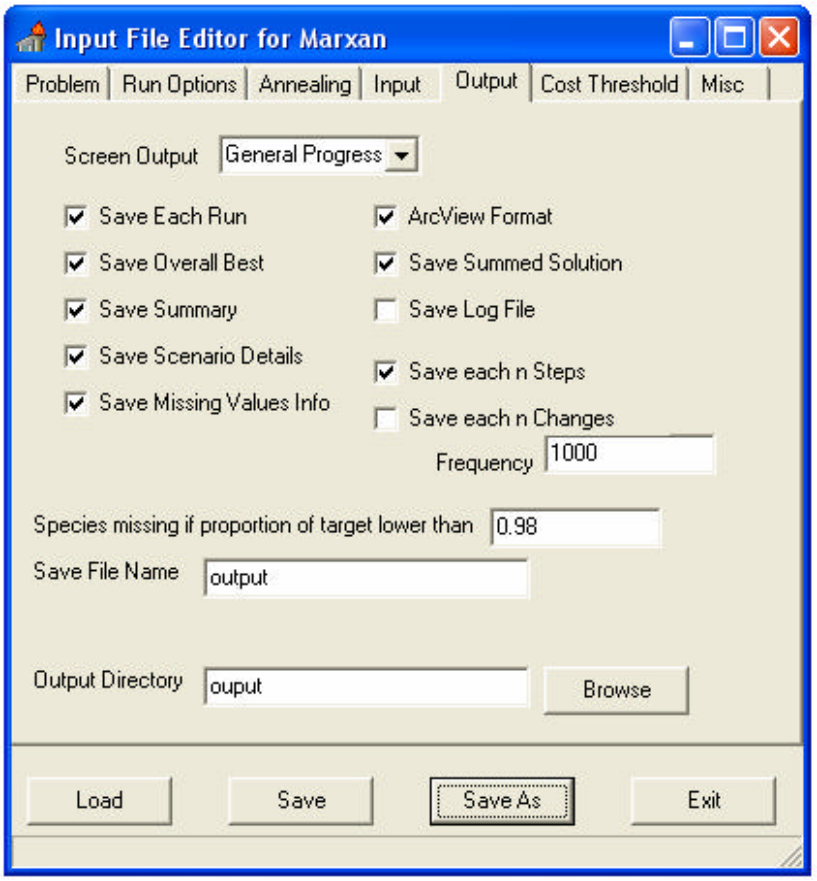
Description and Getting Started: These variables are reasonably self explanatory and their protocols for naming and storage have been discussed previously (see Section 3.1.2). Although on the **Inedit** screen you have a chance to specify the location of each file using the 'Browse' buttons, the files must still be in the correct folder in order for Marxan to run.

3.2.1.5 Output

3.2.1.5.1 Screen Output

Variable – 'VERBOSITY'
Required: Yes

Description: This variable controls how much information Marxan displays on the screen while it is running. In **Inedit** it is set using the drop down box labelled, 'Screen Output'. Users can specify how much information Marxan prints to the screen (the verbosity).



The 'Output' tab of **Inedit**.

There are four different options for screen display:

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.

3.2.1.5.2 Save Files and Save File Name

Variables – 'SAVERUN', 'SAVEBEST', 'SAVESUM', 'SAVESCEN', 'SAVETARGETMET', 'SAVESUMSOLN', 'SAVELOG', 'SAVESNAPSTEPS', 'SAVESNAPCHANGES', 'SAVESNAPFREQUENCY', and 'SCENNAME'
Required: No

Description and Getting Started: With the exception of 'SCENNAME' and 'SAVESNAPFREQUENCY', these variables are all used to tell Marxan what results it should save as output. When using **Inedit** you can tell Marxan to save any of these files simply by ticking the corresponding box. In the 'input.dat' file, set the value to '1' for each output you want Marxan to save. If you wish to display the results in a GIS, tab delimited output should be used in preference to comma delimited. This can be done in **Inedit** by ticking the box labelled 'ArcView Format', and it is done directly in the 'input.dat' by setting the values for these variables to '2'. The different outputs and their uses are all described in detail in the next section.

If either SAVESNAPSTEPS ('Save each n steps' in **Inedit**) or SAVESNAPCHANGES ('Save each n changes' in **Inedit**) are selected then a SAVESNAPFREQUENCY ('Frequency' in **Inedit**) 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.

Beware: 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 (see Section 3.2.1.3.1). For 1 million iterations, a snap frequency of 100,000 will give 10 output files.

The variable, 'SCENNAME' (or 'Save File Name' in **Inedit**), 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.

3.2.1.5.3 Output Directory

Variable – 'OUTPUTDIR'
Required: Yes

Description and Getting Started: The variable is used to tell Marxan the name of the folder (called directory or DIR in Marxan) it should save the output files in. The naming and location protocols for this folder are discussed in Section 5.1) and as with the input folder, it is critical this is correct or Marxan will not run.

3.2.1.5.4 Species missing proportion

Variable – '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. Using **Inedit,** it is specified in the box labelled 'Species missing if proportion of target lower than'. 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 this 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.

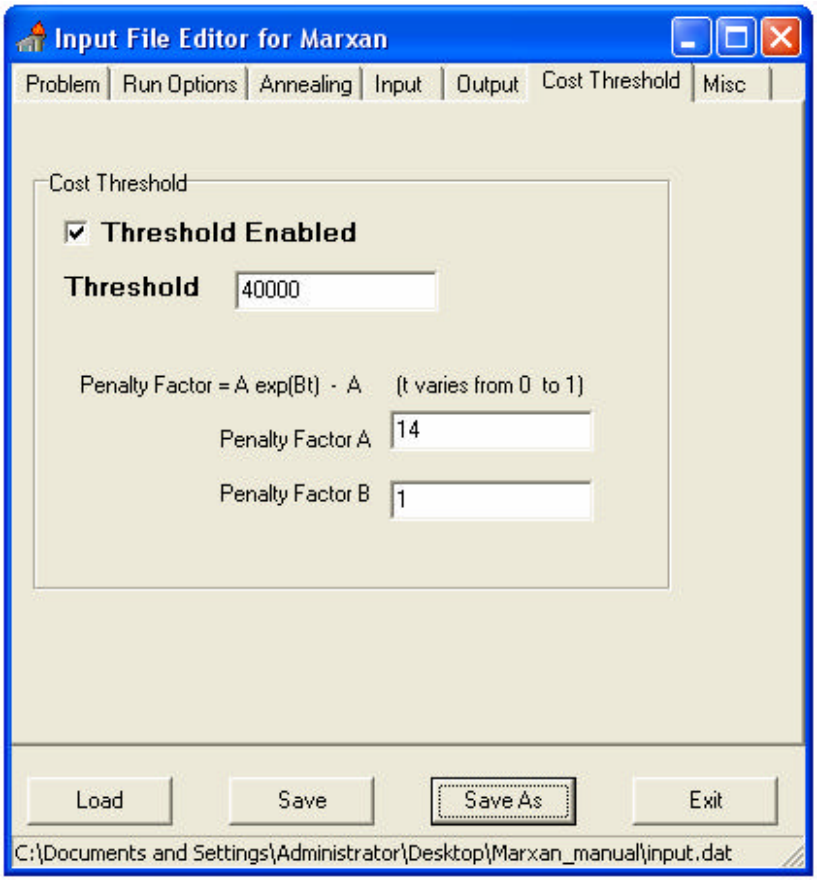
3.2.1.6 Cost Threshold

3.2.1.6.1 Threshold, Penalty Factor A and Penalty Factor B

Variables – 'COSTTHRESH', 'THRESHPEN1', and 'THRESHPEN2'

Required: No.

Note: 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.



The 'Cost Threshold' tab of **Inedit**.

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-1.4

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.

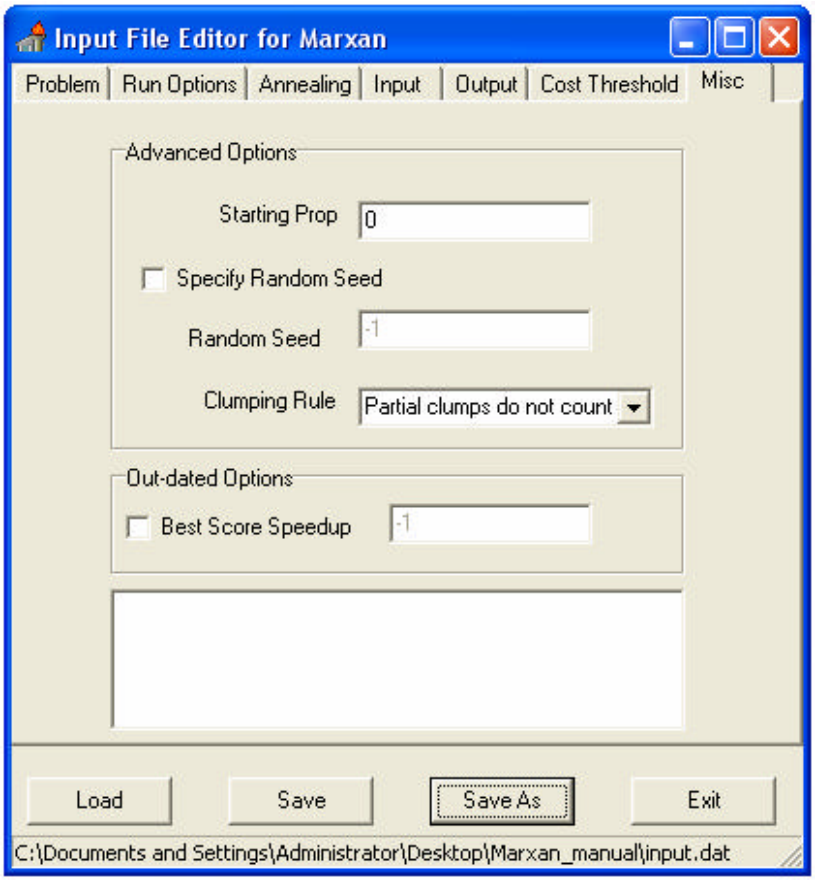
3.2.1.7 Misc

3.2.1.7.1 Starting Prop

Variable – '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, and in **Inedit** it is set using the box labelled 'Starting Prop'. 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.



The 'Misc' tab of **Inedit.**

This will only be applied to those planning units whose status does not lock them in or out of solutions (see Section 3.2.3.3).

3.2.1.7.2 Random Seed

Variable – '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 not checked in **Inedit** (i.e. set to '-1' in the input file).

3.2.1.7.3 Clumping Rule

Variable – 'CLUMPTYPE'
Required: No

Note: 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 (see Section 3.2.2.5 for more information).

Description: This variable is useful if some conservation features have a minimum clump size set (target2, see Section 3.2.2.5). 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.

Getting Started: The 'CLUMPTYPE' is set in **Inedit** using the options in the drop down box labelled 'Clumping Rule'. 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.
3 Graduated penalty – Score is proportional to the size of the clump.

3.2.1.7.4 Best Score Speedup

Variable – 'BESTSCORE'
Required: No

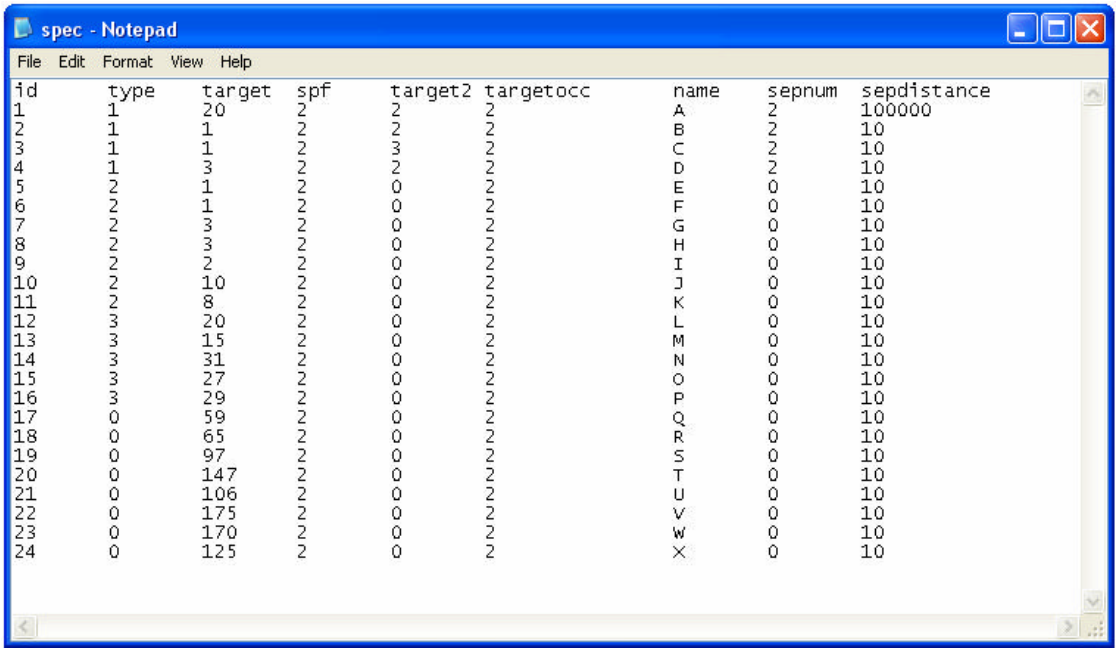
Description: This variable tells Marxan not to keep track of the best score until it reaches a specified minimum level. It is set in **Inedit** using the box labelled 'Best Score Speedup'. It was intended to be a time saving measure, but is seldom required.

Getting Started: It should always be set to '-1'.

3.2.2 The Conservation Feature File

The **Conservation Feature File** contains information about each of the conservation features being considered, such as their name, target representation, and the penalty if the representation target is not met. It has the default name 'spec.dat'. Because of this name it is sometimes referred to as the **Species File**, although conservation features will often be surrogates such as habitat type rather than actual species. Importantly, 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**.

The **Conservation Feature File** can contain up to seven variables, 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.



id	type	target	spf	target2	targetocc	name	sepnum	sepdistance
1	1	20	2	2	2	A	2	100000
2	1	1	2	2	2	B	2	10
3	1	1	2	3	2	C	2	10
4	1	3	2	2	2	D	2	10
5	2	1	2	0	2	E	0	10
6	2	1	2	0	2	F	0	10
7	2	3	2	0	2	G	0	10
8	2	3	2	0	2	H	0	10
9	2	2	2	0	2	I	0	10
10	2	10	2	0	2	J	0	10
11	2	8	2	0	2	K	0	10
12	3	20	2	0	2	L	0	10
13	3	15	2	0	2	M	0	10
14	3	31	2	0	2	N	0	10
15	3	27	2	0	2	O	0	10
16	3	29	2	0	2	P	0	10
17	0	59	2	0	2	Q	0	10
18	0	65	2	0	2	R	0	10
19	0	97	2	0	2	S	0	10
20	0	147	2	0	2	T	0	10
21	0	106	2	0	2	U	0	10
22	0	175	2	0	2	V	0	10
23	0	170	2	0	2	W	0	10
24	0	125	2	0	2	X	0	10

An example of the **Conservation Feature File** (spec.dat) used in Marxan.

Note: It is essential that the header names are exact. Note that all letters are 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.

3.2.2.1 Conservation Feature ID

Variable – '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. Then you can have an idea about what they are at a glance. For example, all birds might have a 1 in the fourth placeholder; e.g. 1003. The fifth placeholder could represent regions if features are treated differently across regions; e.g. 21003 would mean bird 3 in region 2. Whatever system you decide upon, document it and be consistent in its usage.

3.2.2.2 Conservation Feature Type

Variable – 'type'
Required: No

Description: Used to define groups of conservation features for which a number of umbrella attributes can be set for all features within the specified group (or "type"). Each group of features must have a unique numerical identifier. This variable is used in conjunction with the **Block Definition File** (see Section 3.3.2) which will contain the attributes to be assigned to a particular group of conservation features.

Getting Started: All variables you wish to take on Block Definition attributes should have their value entered as -1 in the **Conservation Feature File** (see Section 3.3.2 for example). Otherwise, the value in the **Conservation Feature File** will override the block definition. We recommend using the Conservation Feature Type in conjunction with the **Block Definition File**, whenever feasible, because it streamlines changes, avoids typographic mistakes, and allows for the easy setting of proportional (percentage) targets. It is the only mechanism for setting proportional targets in this and prior versions of Marxan.

3.2.2.3 Feature Representation Target

Variable – 'target'
Required: Yes

Description: The target amount of each conservation feature to be included in the solutions. These 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 3.2.4). 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).

Getting Started: Targets are user defined and 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. 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. 20%). Targets do not, however, have to be uniform values for all species (i.e. always 20%). 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 MGPH.

If **Block Definition File** is being used for this feature, then the Feature Representation Target should be set to -1 here.

3.2.2.4 Conservation Feature Penalty Factor

Variable – 'spf'
Required: Yes

Description: The letters 'spf' stands for Species Penalty Factor. This variable is more correctly referred to as the Conservation Feature Penalty Factor. 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 (see Appendix B-1.4 for details of how this penalty is calculated and applied). 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 SPF 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 SPF values than less important features. This signifies that you are less willing to compromise their representation in the reserve system. 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 SPFs. This should be done in an iterative fashion. A good place to start is to choose the lowest value that is of the same order of magnitude as the number of conservation features, e.g. if you have 30 features, start with test SPFs of, say, 10 for all features. Do a number of repeat of runs (perhaps 10) and see if your targets are being met in the solutions. If not all targets are being met try increasing the SPF by a factor of two and doing the repeat runs again. When you get to a point where all targets are being met, decrease the SPFs slightly and see if they are still being met. After test runs are sorted out, then differing relative values can be applied, based on considerations such as rarity, ecological significance, etc., as outlined above.

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 SPF. 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 SPF for these features. Once again, see the MGPH for more detail on setting SPFs.

If **Block Definition File** is being used for this feature, then the Conservation Feature Penalty Factor should be set to -1 here.

3.2.2.5 Minimum Clump Size

Variable – 'target2'

Required: No

Note: 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.

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 that 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 3.2.4). 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.

Using the Clumping Functionality within Marxan

There are 2 primary reasons why users experience difficulties with the clumping functionality in Marxan. This functionality is related to the parameters target2, sepnum and sepdistance.

1) A solution for the clumping problem does not exist. You can determine whether this is the case by performing GIS analysis on your feature and planning unit layers.
2) The relatively unsophisticated algorithm used to search for solutions containing clumps cannot find a solution for the clumping problem. The algorithm does not perform an exhaustive search of decision space to find a solution.

In spite of this limitation, the clumping functionality has been successfully used to solve research and management problems for a range of datasets around the world. There are some general techniques that can be used to manage this limitation. There is no research and development project currently scheduled to address this limitation.

For a large number of restarts of Marxan, solutions can be found if they exist for some of those restarts. The proportion of restarts required will be related to the difficulty Marxan has in finding a solution to the clumping problem.

In the initialisation routine for Marxan, a solution to the problem is computed, and feature penalties are set based on this solution. If this solution does not meet the clumping objectives, then incorrect penalties will be subsequently used for that run of Marxan, leading to incorrect operation.

You can follow these steps to compute clumping solutions with Marxan;

1) Restart Marxan.
2) Observe if feature targets are met by run 1. If the targets are not met, kill the Marxan run and restart it.
3) Repeat this until the requisite number of good solutions has been computed.

A simple and elegant alternative way to achieve clumping is to use the boundary length modifier. This method is extremely robust and fast, however, it doesn't give the same precise control over size, number and separation of clumps.

3.2.2.6 Target for Feature Occurrences

Variable – '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'.

If **Block Definition File** is being used for this feature, then the Target for Feature Occurrences should be set to -1 here.

3.2.2.7 Conservation Feature Name

Variable – 'name'
Required: No

Description: The alphabetical (no numbers!) name of each conservation feature (e.g. *cloud forest*). This variable is unusual in that it can include spaces. If you want to include spaces in the name, use words not numbers to allow Marxan to read a series of words as a single variable. You can also use numbers with no spaces as the name if you want.

3.2.2.8 Target for Separated Feature Occurrences

Variable – 'sepnum'
Required: No

Note: 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 (see Section 3.2.2.5 for more information).

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.
If **Block Definition File** is being used for this feature, then the Target for Separated Feature Occurrences should be set to -1 here.

3.2.2.9 Minimum Separation Distance

Variable – 'sepdistance'
Required: No

Note: 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 (see Section 3.2.2.5 for more information).

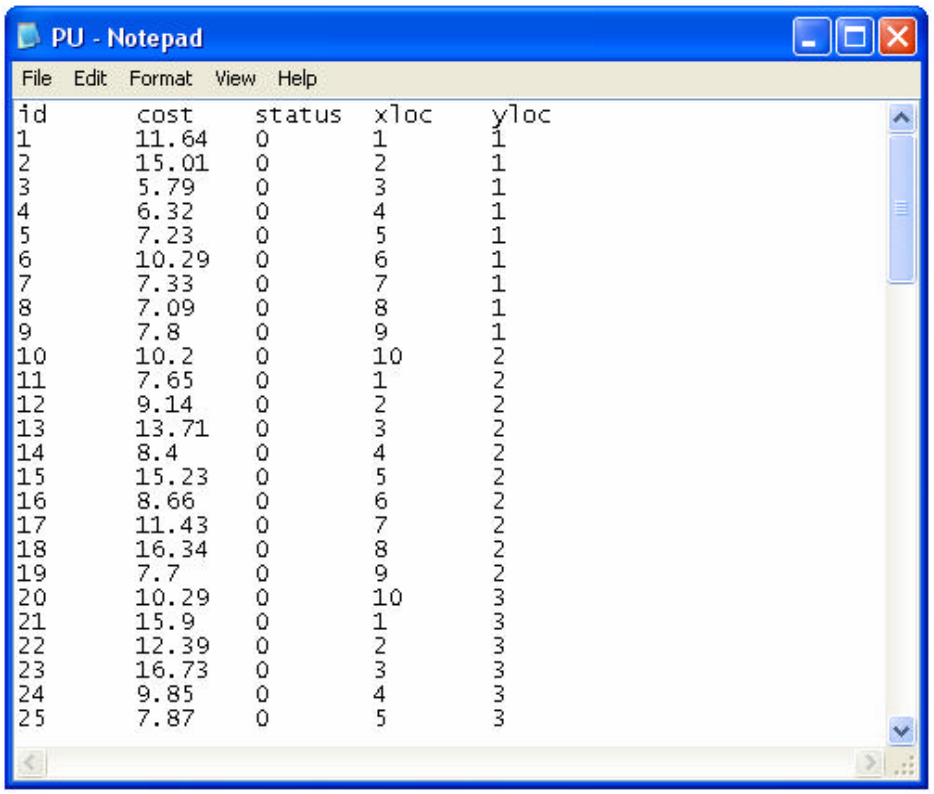
Description: Used in conjunction with 'sepnum' (above), this variable specifies the minimum distance at which planning units holding a conservation feature are considered to be separate. This may be useful in situations where multiple occurrences are desired because of the threat of large-scale events 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 3.2.3.4).

This feature can slow down Marxan considerably.
If **Block Definition File** is being used for this feature, then the Minimum Separation Distance should be set to -1 here.

3.2.3 The Planning Unit File

The **Planning Unit File** 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 default name for this file is 'pu.dat'. The **Planning Unit File** can contain up to five variables, although only one of these ('id') is required. When included, each of these variables is presented in a column with the name of that variable as the column header.



id	cost	status	xloc	yloc
1	11.64	0	1	1
2	15.01	0	2	1
3	5.79	0	3	1
4	6.32	0	4	1
5	7.23	0	5	1
6	10.29	0	6	1
7	7.33	0	7	1
8	7.09	0	8	1
9	7.8	0	9	1
10	10.2	0	10	2
11	7.65	0	1	2
12	9.14	0	2	2
13	13.71	0	3	2
14	8.4	0	4	2
15	15.23	0	5	2
16	8.66	0	6	2
17	11.43	0	7	2
18	16.34	0	8	2
19	7.7	0	9	2
20	10.29	0	10	3
21	15.9	0	1	3
22	12.39	0	2	3
23	16.73	0	3	3
24	9.85	0	4	3
25	7.87	0	5	3

An example of the **Planning Unit File** (pu.dat) used in Marxan.

It is essential that header names are exact. All are lower case and contain no punctuation, spaces or numeric characters. The order headers are presented in does not matter.

3.2.3.1 Planning Unit ID

Variable – '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. There is an upper limit of around 20 000 to 30 000 on the number of planning units that basic Marxan can handle, though the optimised version has no such restrictions (see Section 1.8 7.1).

This number should not be confused with the variable, 'id', in the **Conservation Feature File** (see Section 3.2.2).

3.2.3.2 Planning Unit Cost

Variable – '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 Section 1.5) 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 MGPH 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 can not sensible be combined without using arbitrary weightings (e.g. if one cost is monetary and one is social).

3.2.3.3 Planning Unit Status

Variable – '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:

Table 3: Planning Unit values.

Status	Meaning
0	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 3.2.1).
1	The PU will be included in the initial reserve system but may or may not be in the final solution.
2	The PU is fixed in the reserve system ("locked in"). It starts in the initial reserve system and cannot be removed.
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, see Section 3.2.1.1.2), 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 it 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.

3.2.3.4 X Planning Unit Location

Variable – 'xloc'
Required: No

Description: 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 **Conservation Feature File** (see Section 3.2.2.9). 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).

More information on generating this value can be found in the tutorials (Appendix C-2).

3.2.3.5 Y Planning Unit Location

Variable – '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 **Conservation Feature File** (see Section 3.2.2.9). 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)

More information on generating this value can be found in the tutorials (Appendix C-2).

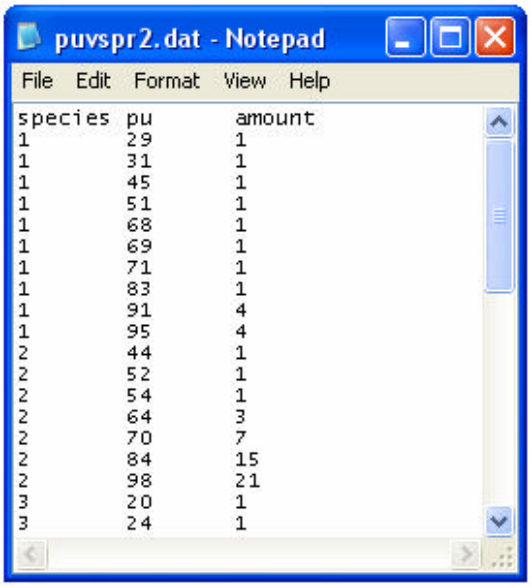
3.2.4 The Planning Unit versus Conservation Feature File

The **Planning Unit versus Conservation Feature File** contains information on the distribution of conservation features across planning units. It has the default file name, 'puvpsr2.dat'. There are two different formats this file can take, vertical and horizontal[8]. Either is acceptable and Marxan will test the header line to determine which format is being used.

More information on generating either format can be found in the tutorials (Appendix C-3).

3.2.4.1 Vertical Format

When using the vertical format, the **Planning Unit versus Conservation Feature File** contains three columns, all of which are required. The 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'). Thus 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.



An example of the vertical form of the **Planning Unit versus Conservation Feature File** (puvspr2.dat) used in Marxan.

[8] In previous Marxan manuals, these vertical and horizontal formats were known as "relational" and "tabular," respectively.

3.2.4.1.1 Conservation Feature ID

Variable – '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 3.2.2.1).

3.2.4.1.2 Planning Unit ID

Variable – '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 3.2.3.1).

3.2.4.1.3 Conservation Feature Amount

Variable – '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.

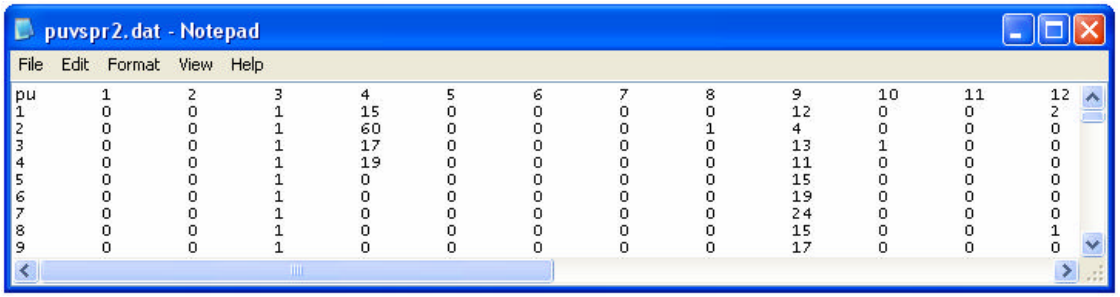
Getting Started: There is no requirement to use the same metric for all conservation features. It is essential, however, that the amount for a given feature is in the same units used to set the target representation for that feature (see Section 3.2.2.3). In the vertical file format, you should not list 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.

Note: It does not matter how the data in this file is ordered (i.e. it does not have to be sequential by any of the variables), however, ordering by conservation features helps to ensure that the data for all features has been entered.

More information on generating either format can be found in the tutorials (Appendix C-3).

3.2.4.2 Horizontal Format

In the horizontal format, the **Planning Unit versus Conservation Feature File** is simply a matrix of planning units versus conservation features. The first column begins with the header, 'id', and is a list of the id number of every planning unit found in the **Planning Unit File** (see Section 3.2.3.1). Each subsequent column has a header with the id number of a conservation feature. These must correspond to the id numbers found in the **Conservation Feature File** (see Section 3.2.2.1), and there must be one column for each conservation feature. Marxan will assume that any conservation feature that does not appear as a column header does not occur in any planning units. Each row will be populated by values that indicate the amount of every conservation feature found in that planning unit. This includes entries of '0' in cases where a particular feature is not found in that planning unit. Although this is perhaps a more intuitive way to present conservation feature distribution data, and easier to read, it results in larger files with a lot of redundant information. As with the vertical format, there are no default values for this file and any missing data or incorrect headers will prevent Marxan from running.



pu	1	2	3	4	5	6	7	8	9	10	11	12
1	0	0	1	15	0	0	0	0	12	0	0	2
2	0	0	1	60	0	0	0	1	4	0	0	0
3	0	0	1	17	0	0	0	0	13	1	0	0
4	0	0	1	19	0	0	0	0	11	0	0	0
5	0	0	1	0	0	0	0	0	15	0	0	0
6	0	0	1	0	0	0	0	0	19	0	0	0
7	0	0	1	0	0	0	0	0	24	0	0	0
8	0	0	1	0	0	0	0	0	15	0	0	1
9	0	0	1	0	0	0	0	0	17	0	0	0

An example of the horizontal form of the **Planning Unit versus Conservation Feature File** (puvspr2.dat) used in Marxan.

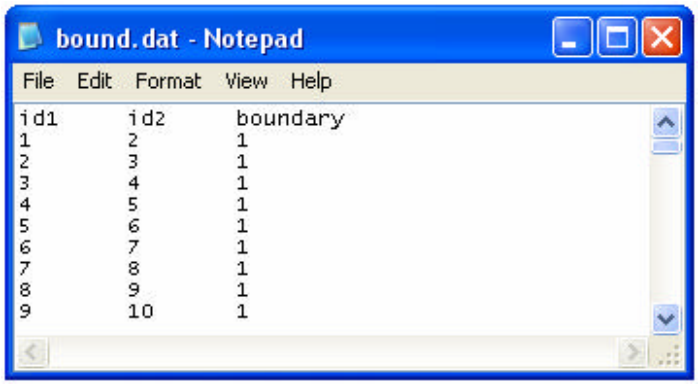
More information on generating either format can be found in the tutorials (Appendix C-3).

3.3 Optional Files

The following two files are both optional, if no file name is given in the **Input Parameter File** (see Section 3.2.1), Marxan will not attempt to find them.

3.3.1 The Boundary Length File

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 (see Section 3.2.1.1.2) to improve the compactness of reserve solutions. It is not necessary to specify boundary lengths for all planning units (where they are not specified, Marxan will assume there is no boundary between planning units). However any missing values within the file will prevent Marxan from running, for instance if 'id1' and 'id2' are set but no value for 'boundary' is entered.



An example of the **Boundary Length File** (bound.dat) used in Marxan.

3.3.1.1 Planning Unit IDs

Variables – '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 MGPH 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.

3.3.1.2 Boundary Length

Variable – '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.

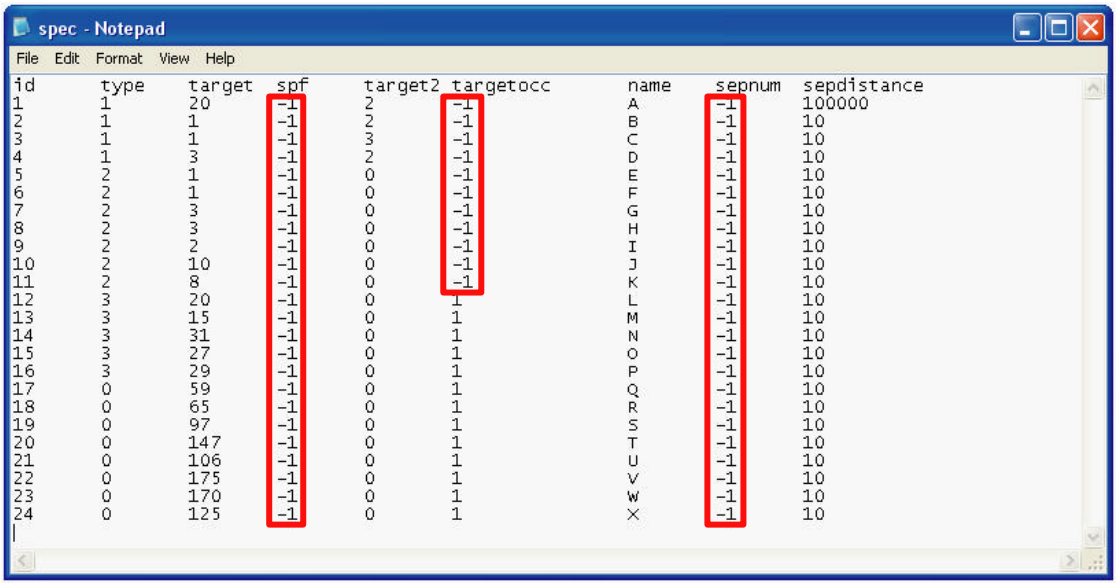
The value entered in 'boundary' should always be '0' or greater. Although it is not generally necessary to specify cases where there is no cost between planning units, a zero cost boundary can be useful if you want to identify two planning units as neighbours but there is no actual boundary cost. This may be necessary if you have set minimum clump sizes for some conservation features (see Section 3.2.2.5).

More information on generating the bound.dat can be found in the tutorials (Appendix C-5).

3.3.2 The Block Definition File

The **Block Definition File** is very similar to the **Conservation Feature File** (see Section 3.2.2) and is used to set default variable values for groups of conservation features. It is always used in conjunction with the **Conservation Feature File**. Groups of conservation features must first be defined using the variable, 'type' in the **Conservation Feature File** (see Section 3.2.2.2). Features may form part of a group because the values for some of their variables are, for example, defined by specific legislation, or they are similar in their ecological characteristics. For each different type of conservation feature, the values for all other variables in the **Conservation Feature File** can then be defined using **Block Definition File**. This provides a quick and easy method for implementing common targets for groups of features and is useful in cases where various targets are being used and different levels of protection explored. It also provides a quick way to provide proportional (percentage) protection, without having to manually calculate these values.

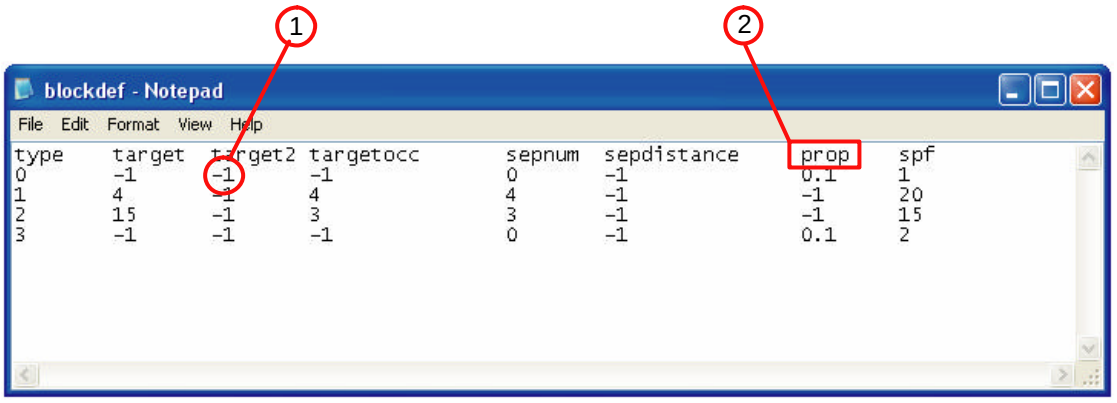
In order for groups of features to take on the values defined in the **Block Definition File**, the variable value in the **Conservation Feature File** must be set to '-1'.



An example of how values must be set in the **Conservation Feature File** (spec.dat) if you wish these parameters to take on the values defined in the **Block Definition File** (blockdef.dat). In this case, all 24 conservation features will take the Block Definition value for the variables 'spf' (Penalty Factor) and 'sepnum', and 11 of the conservation features will take the Block Definition value for the variable 'targetocc' (Target for Separated Feature Occurrences).

This is a convenient way to allow some variables to take on the group value and some to be set individually. For instance, in the example above the variable 'target' is defined for separately for each feature, perhaps because of differing abundance in the planning region. On the other hand, the variable 'spf', the **Conservation Feature Penalty Factor** (see Section3.2.2.4), always takes on the value defined in the **Block Definition File**. This may be useful in cases where the different types of conservation feature are of differing importance but we are unsure of the appropriate 'spf' value. By using the **Block Definition File** we can alter the 'spf' for each type feature without having to alter every entry in the **Conservation Feature File**.

The **Block Definition File** can contain up to eight different variables and should be in the same format as the **Conservation Feature File**.



An example of the **Block Definition File** (blockdef.dat) used in Marxan. 1. Negative 1 must be entered in this file if you want the variable to take on the original value defined in the **Conservation Feature File**. 2. 'Prop' is the only new variable in this file --see below for description.

The only variable that is required in this file is 'type', and the only new variable is 'prop'. Both of these are described below.

3.3.2.1 Conservation Feature Type

Variable – 'type'
Required: Yes
Description: A unique numerical identifier for groups of conservation features. Each 'type' must correspond exactly with the types identified in the **Conservation Feature File** (see Section 3.2.2.2).

3.3.2.2. Proportion Target for Feature Representation

Variable – 'prop'
Required: No
Description: 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.

Getting Started: This should be a number 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 3.2.4). 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. Because 'prop' sets a target value, where it is used

the value for the variable, ‘target’, should be set to ‘-1’. 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 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.

3.3.2.3 All other variables

The remaining variables in the **Block Definition File** (‘target’, ‘target2’, ‘targetocc’, ‘sepnum’, ‘sepdistance’ and ‘spf’), all have the same definition as in the **Conservation Feature File** (see Section 3.2.2) and where not defined here will take on the value in that file. If you wish any of these six variables, that are defined, to take on the original value from the **Conservation Feature File**, simply set the value in the **Block Definition File** to ‘-1’. This is also the default value for these variables and any missing entries will take this value.

(This page intentionally blank)

4. Running the software

Actually running the Marxan program is extremely simple. Once all the input files are ready all you 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 2.2).

If Marxan executes successfully, a program screen showing information on the details and progress of each run will be displayed (unless Silent Running has been selected – see Section 3.2.1.5.1). 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 a file called 'log.dat'. When Marxan completes the preset 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 for troubleshooting details).

(This page intentionally blank)

5. Outputs

In addition to telling you which planning units will make up an efficient reserve system, Marxan can also provide a variety of other outputs, these are described below. All Marxan output 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 wish Marxan to save must be specified in the **Input Parameter File** (see Section 3.2.1).

Although the solutions Marxan generates are commonly displayed graphically, Marxan will not actually generate maps. The user must combine information from Marxan output files with the planning unit details displayed within a GIS. Appendix C includes a tutorial on how to do this. If you are going to import data straight from Marxan into a GIS it may be helpful to select the 'ArcView Format' option provided in the **Input Parameter File** (see Section 3.2.1.5.2).

5.1 Output File Management

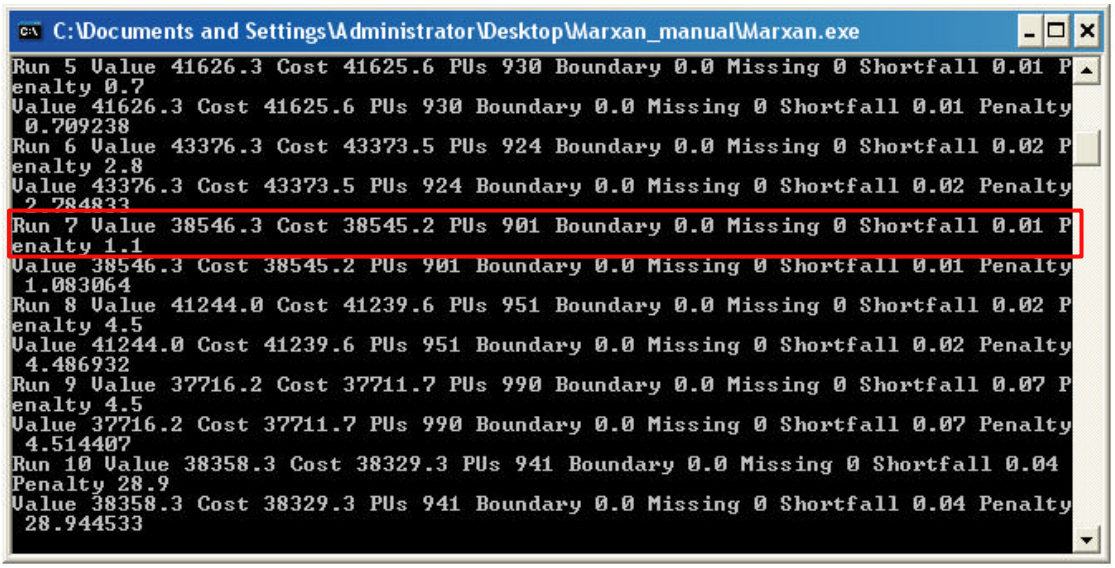
Marxan will save output files in a folder whose name you have the chance to specify in the **Input Parameter File**. We suggest using the default folder name 'output' and then changing the file name once the run is complete. You must make sure that the output directory you specify has been created before 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. See Section 3.1.2). 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.

5.2 Screen Output

As Marxan runs some output can be provided on the screen. This output is useful to check on how the program is running and to give a brief summary of the solutions. It will quickly (or not as the case may be) give you an idea of how long a single run takes and from there you can pretty accurately gauge how long it will take to finish all runs for that scenario. There are a number of levels of information that can be requested for screen output; the level requested is termed the verbosity level (See Section 3.2.1.5.1).

5.2.1 Basic Results

If the verbosity is set to anything other than 'no output' then the following basic summary of each run will be given on the screen. If 'Results Only' (verbose mode 1) is chosen this is the only information that will be provided.



Example of the on screen summary provided using screen output mode 1 (Results Only).

Note: This same information is provided in the 'summary information' output file (See Section 5.3) so you need not look at the screen output after Marxan has finished running.

5.2.1.1 Run

Description: Which of the repeat runs (see Section 3.2.1.1.1) the output pertains to. Depending on the level of detail printed to the screen, the run number will not always appear on the same line as the details of the reserve solution (see 5.2.2). The information for each run will appear as soon as that run is complete.

5.2.1.2 Value

Description: 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 (see Section 1.5). It is useful to know this value because it is how Marxan chooses the 'best' solution out of you repeat runs.

5.2.1.3 Cost

Description: This is the total cost of the reserve system as determined solely by the costs given to each planning unit (see Section 3.2.3.2).

5.2.1.4 PUs

Description: The number of planning units contained in the solution for that run.

5.2.1.5 Boundary

Description: The total boundary length of the reserve system (see Section 3.3.1.2). 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'.

5.2.1.6 Missing

Description: The number of conservation features that did not achieve their targets in the final solution for that run. This is screened according to the 'miss level' which has been set in the **Input Parameter File** (see Section 3.2.1.5.4). 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.

5.2.1.7 Shortfall

Description: 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.

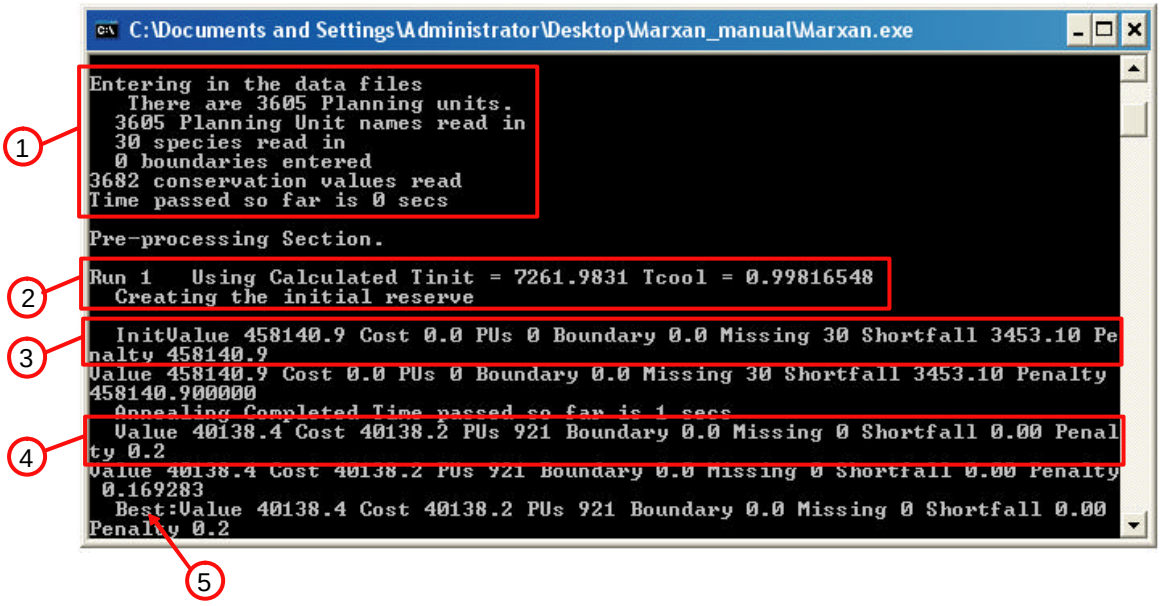
5.2.1.8 Penalty

Description: 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). How this penalty is calculated is described in detail in Appendix B. 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.

5.2.2 General Progress

If the screen output is set to either 'General Progress' (verbose mode 2), or 'Detailed Progress' (verbose mode 3), then, in addition to the basic results above, the following summaries are also provided.



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 reserve solution from that run (as described above).
5. This output is just used by Marxan so it can quickly sort through all runs to determine which one gave the best solution.

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 Section 3.2.1.2.1), 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 Section 3.2.1.3.1). 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. If you are using this output for any of these purposes you may want to save the screen output so you can look at it after Marxan has finished running (see Section 5.3).

5.2.3 Detailed Progress

If desired, Marxan can provide very detailed information on the screen (verbose mode 3). This information shows exactly what the program is up to during each run and how the annealing is progressing. At a quick glance, it allows you to confirm that the algorithm is running as it should, and has not "stalled." Further details of this output are more advanced, and explained in Appendix B.

5.3 Output Files

Marxan can save up to eight different output files depending on what was specified in the **Input Parameter File** (see Section 3.2.1.5.2).

Table 4: Output file types and names.

Output File Type	File Name
Solutions for each run	*scenario*_r001.dat
Best solution from all runs	*scenario*_best.dat
Missing value information for each run	*scenario*_mv001.dat
Missing value information for the best run	*scenario*_mvbest.dat
Summary information	*scenario*_sum.dat
Scenario details	*scenario*_sen.dat
Summed solution	*scenario*_ssoln.dat
Screen log file	*scenario*_log.dat
Snapshot files	*scenario*_snap_r00001t01000.dat

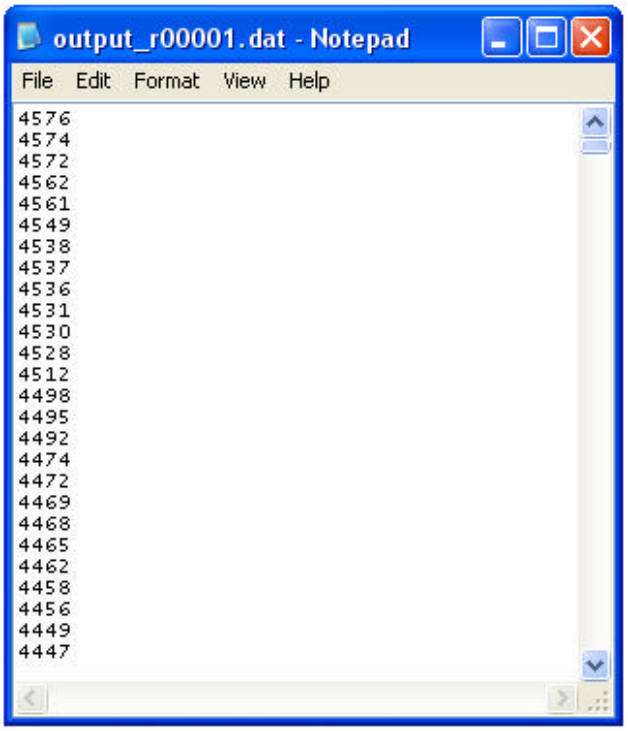
The file prefix '*scenario*' will take on whatever name is specified by the user for variable 'SCENNAME' in the **Input Parameter File** (see Section 3.2.1.5.2). Where a number is included in the file name (e.g. *scenario*_r001.dat), this is the run number that generated that particular output.

5.3.1 Output File Format

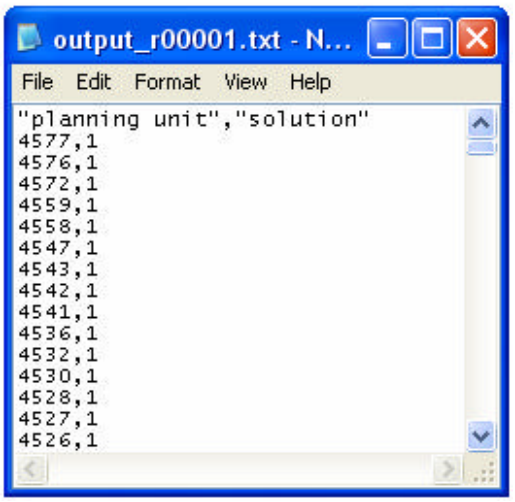
The standard output format is a tab separated text file with the extension '.dat'. If ArcView format is selected within **Inedit** or the **Input Parameter File** (see Section 3.2.1.5.2) then all files except the summary information file and the screen log file will be separated by commas and the headings will be in the necessary format for importing into GIS programs such as ArcView. The extension for ArcView format files is '.txt'.

5.3.2 Solutions for each run

File name: scenario_r001.dat
Description: A file is produced for each repeat 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. Selected planning units are identified by their ID numbers (see Section 3.2.3.1) and each appears on a separate line. If ArcView format (.txt) is specified the file will also have the headings, "planning unit" and "solution", and each line will have a planning unit number followed by a 1 separated by commas. This file allows you to easily display the reserve system solution from each run, however, as you should nearly always do multiple repeat runs, a useful output file is the 'summed solution' file (see Section 5.3.7).



Example of the solution file for run number 1 – standard format.



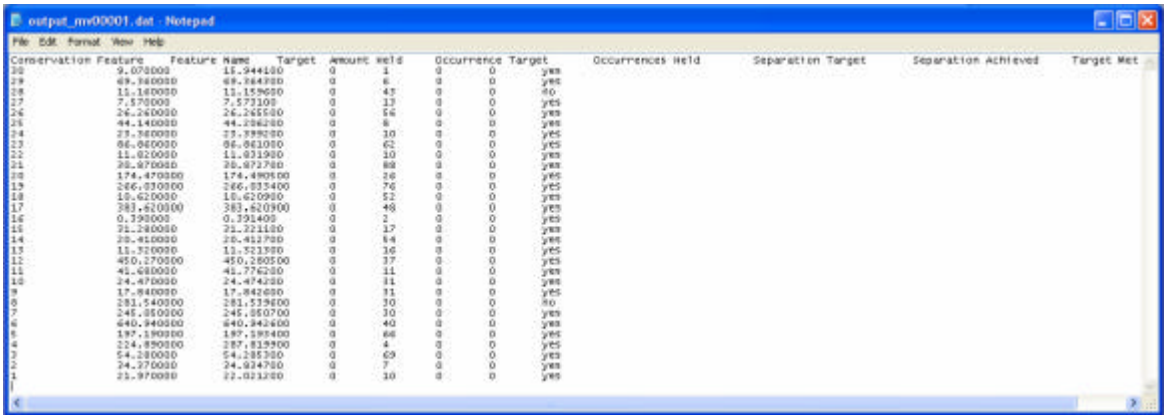
Example of the solution file for run number 1 – ArcView format.

5.3.3 Best solution from all runs

File name: scenario_best.dat
Description: Exactly as above except for the run that produced the solution with the best objective value. Be careful: the solution from the 'best' run is only superior with regard to the objective function value, in reality this does not make it the best reserve system. Similarly, 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 seen as a very good solution, within a continuum of options. More discussion on this topic can be found in the MGPH.

5.3.4 Missing values for each run

File name: scenario_mv001.dat
Description: This file contains information about the representation of conservation features in the solution for each run. The file contains a total of nine columns which basically report on how the solution performed relative to the targets. Some of these are simply a summary of the information provided in the **Conservation Feature File**.



Example of the missing value file for run number 1. Note: the column headers and data do not always line up because of the requirement for tab separation between headers. This visual inconvenience can be addressed by importing the output into a spreadsheet program like MS Excel.

5.3.4.1 Conservation Feature

Description: The unique ID number of the conservation feature (see Section 3.2.2.1).

5.3.4.2 Feature Name

Description: The optional alphabetic name of the conservation feature (see Section 3.2.2.7). If no name has been specified then nothing will appear in this column.

5.3.4.3 Target

Description: The target level of representation (if any) for that conservation feature (see Section 3.2.2.3).

5.3.4.4 Amount Held

Description: The amount of that conservation feature captured in the reserve system. Only amounts in valid clumps are included (see Section 3.2.2.5).

5.3.4.5 Occurrence Target

Description: The target number of occurrences in the reserve system for that conservation feature (see Section 3.2.2.6).

5.3.4.6 Occurrences Held

Description: The number of occurrences of the conservation feature captured in the solution. Again, only occurrences in valid clumps are included.

5.3.4.7 Separation Target

Description: The number of mutually and adequately separated occurrences of that conservation feature required in the reserve system (see Sections 3.2.2.8 and 3.2.2.9).

5.3.4.8 Separation Achieved

Description: The number reported here will be the lowest of either: the number of separate occurrences that are actually achieved in the reserve system; or the target number of separate occurrences. The separation count (see Appendix B-1.3.1) never exceeds the separation target for that feature. This is a convention which speeds up the execution of the software but it means that no information is given about how far this target is exceeded.

5.3.4.9 Target Met

Description: An alphabetic variable that returns 'yes' if all the targets set for that feature are met, otherwise it returns 'no'.

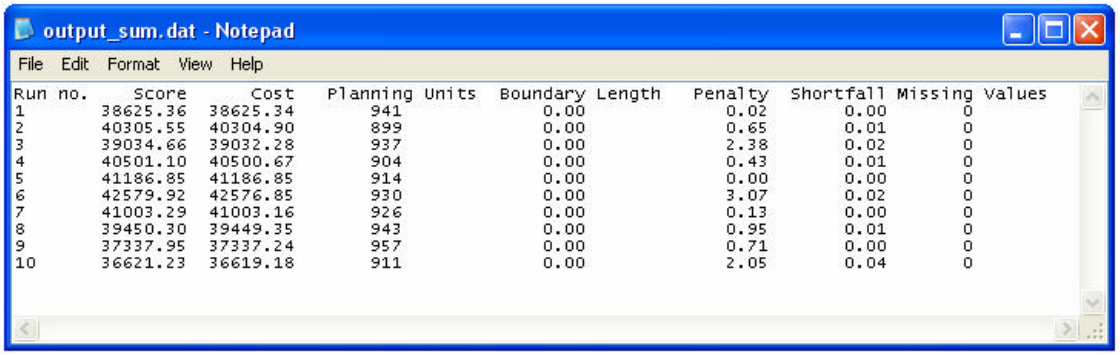
5.3.5 Missing value information for the best run

File name: scenario_mvbest.dat

Description: Exactly as above except for the run that produced the solution with the best objective value.

5.3.6 Summary information

File name: scenario_sum.dat
Description: This file contains the summary information for each repeat run. It is exactly the same information as given in the basic 'on screen' output as described in Section 5.2.1.

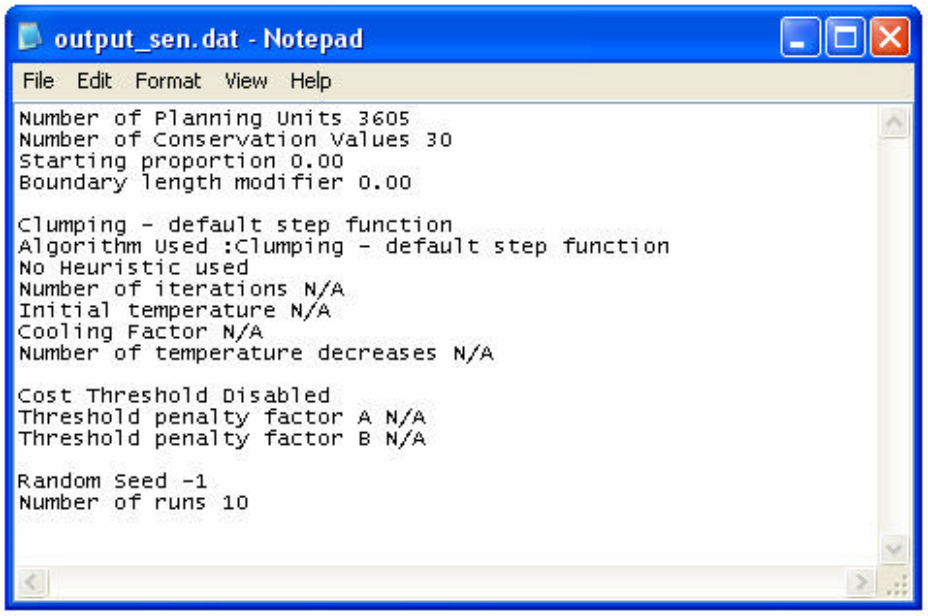


Run no.	Score	Cost	Planning Units	Boundary Length	Penalty	Shortfall	Missing Values
1	38625.36	38625.34	941	0.00	0.02	0.00	0
2	40305.55	40304.90	899	0.00	0.65	0.01	0
3	39034.66	39032.28	937	0.00	2.38	0.02	0
4	40501.10	40500.67	904	0.00	0.43	0.01	0
5	41186.85	41186.85	914	0.00	0.00	0.00	0
6	42579.92	42576.85	930	0.00	3.07	0.02	0
7	41003.29	41003.16	926	0.00	0.13	0.00	0
8	39450.30	39449.35	943	0.00	0.95	0.01	0
9	37337.95	37337.24	957	0.00	0.71	0.00	0
10	36621.23	36619.18	911	0.00	2.05	0.04	0

Example of the summary output file.

5.3.7 Scenario Details

File name: scenario_sen.dat
Description: This file contains a documented list of all the major parameter values for that scenario. 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.

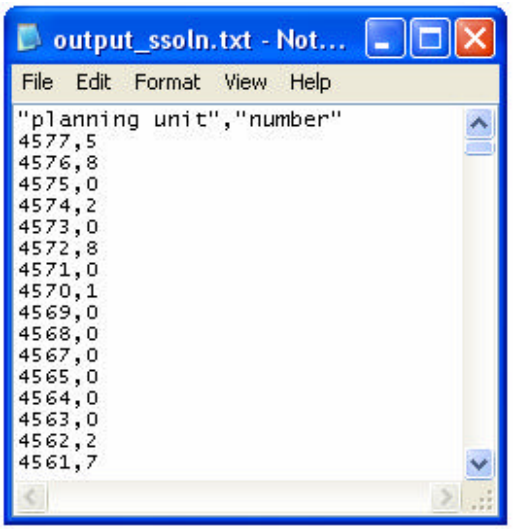


Example of the scenario details output file.

5.3.8 Summed solution

File name: scenario_ssoln.dat

Description: Summed solution provides the selection frequency of each planning unit across all runs. Each line has the ID number of a planning unit (see Section 3.2.3.1) and the number of times that planning unit was selected in the final solution across all repeat runs[9]. 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 that mean 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.



Example of the summed solution output file – ArcView format.

5.3.9 Screen log file

File name: scenario_log.dat

Description: A text file containing exactly what Marxan displayed as screen output for that scenario (see Section 5.1). 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 (see Section 5.2.2).

[9] This measure was previously termed 'Irreplaceability' though 'Selection Frequency' is a more accurate description. This concept is inspired by, but different from, Bob Pressey's notion of irreplaceability (Pressey et al. 1994). For a comparison of irreplaceability and selection frequency see Carwardine et al. (2006).

5.3.10 Snapshot file

*File name: scenario*_snap_r00001t01000.dat (for the 1000th snapshot in the 1st run)
Description: 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 (see Section 3.2.1.5.2). 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. It is not generally recommended.

(This page intentionally blank)

6. 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 MGPH. Once you have a got a feel for how the program works we strongly encourage you to read the 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.

6.1 Experimentation

The most important thing to remember is that Marxan does not provide 'one-stop' reserve solutions. As outlined in Section 3 of 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 can often interact in unexpected ways, for instance, the optimal BLM may change dramatically if the SPF 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 as many scenarios as your project time and budget permit, and ensuring that there is adequate time allocated for these experimentations.

6.2 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 a solution 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 particular 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.

6.3 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 Airame 2005). Determining if the output from different scenarios is 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.

6.4 Becoming an Expert

Do all of the above, a lot! And, please, share your findings with others. There is a Marxan list-serve. Send an email to 'marxan-owner@sib.uq.edu.au' requesting list membership, or download Marxan from its web site and indicate on the download form that you would like to become a list member. If you find a bug in Marxan, please report it to Hugh Possingham by email 'h.possingham@uq.edu.au' or the Marxan list by email 'marxan@sib.uq.edu.au'. Likewise, send any notifications of publications, reports and funding opportunities related to Marxan.

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. (See also *Simulated annealing* and *Fixed schedule annealing*.) (See Appendix B-2.1.1)

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. (See Section 3.2.1.1.2 and Appendix B-1.2)

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 (See Section 3.2.2.5 and Appendix B-1.3.1). 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*.

Conservation feature penalty factor: See *Species penalty factor*

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. (See Section 3.2.3.2 and Appendix B-1.1)

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.

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*.) (See Appendix B-2.1.2)

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 (See Appendix B-2.3).

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 (See Appendix B-2.2).

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.

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.

MGPH: Marxan Good Practices Handbook, a complementary document to this Marxan User Manual.

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. (See Section 1.5)

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. (See Section 1.7.1)

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 be from one another in order to be considered as separate representations. This could be considered a type of risk spreading. (See Section 3.2.2.9)

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): 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. (See Appendix B-1.3)

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.

Key References

Core Marxan references

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

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

Ardron, J. and C. Klein (Eds.). (2008). *Marxan good practices handbook*. St. Lucia, Queensland, Australia: University of Queensland, and Vancouver, British Columbia, Canada: Pacific Marine Analysis and Research Association.

Selected references that use Marxan and describe concepts

Airame, S. (2005). Channel Islands National Marine Sanctuary: Advancing the science and policy of marine protected areas. In: A. Scholz and D. Wright (Eds.), *Place Matters: Geospatial Tools for Marine Science, Conservation, and Management in the Pacific Northwest* (pp 91-124). Corvallis, OR: Oregon State University Press.

Ando, A., J. Camm, S. Polasky and A. Solow. (1998). Species distributions, land values, and efficient conservation. *Science,* 279: 2126-2128.

Araújo, M.B. (2004). Matching species with reserves - uncertainties from using data at different resolutions. *Biological Conservation,* 118: 533-538.

Armsworth, P.R., G.C. Daily, P. Kareiva and J.N. Sanchirico. (2006). Land market feedbacks can undermine biodiversity conservation. *Proceedings of the National Academy of Sciences (PNAS)*, 103: 5403-5408.

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

Balmford, A., K.J. Gaston, A.S.L. Rodrigues and A. James. (2000). Integrating costs of conservation into international priority setting. *Conservation Biology*, 14: 1-9.

Ban, N. (in review). Beyond marine reserves: Exploring the approach of selecting permitted fishing areas.

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

Burgman, M.A., H.P. Possingham, A.J.J. Lynch, D.A. Keith, M.A. McCarthy, S.D. Hopper, W.L. Drury, J.A. Passioura and R.J. Devries. (2001). A method for setting the size of plant conservation target areas. *Conservation Biology,* 15: 603-616.

Cabeza, M. and A. Moilanen. (2001). Design of reserve networks and the persistence of biodiversity. *Trends in Ecology and Evolution*, 16: 242-248.

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

Carwardine, J., W. Rochester, K. Richardson, K. Williams, R. Pressey and H. Possingham. (2006). Conservation planning with irreplaceability: Does the method matter? *Biodiversity and Conservation*, 16: 1-14.

Carwardine, J., K. Wilson, M. Watts, and H.P. Possingham. (2006). Where do we act to get the biggest conservation bang for our buck? A systematic spatial prioritisation approach for Australia. In: European Congress for Conservation Biology conference Proceedings. Eger, Hungary.

Chan, K.M.A., M.R. Shaw, D.R. Cameron, E.C. Underwood and G.C. Daily. (2006). Conservation planning for ecosystem services. *PLoS Biology*, 4: e379.

Church, R.L., D.M. Stoms, and F.W. Davis. (1996). Reserve selection as a maximal covering location problem. *Biological Conservation,* 76: 105-112.

Cocks, K.D. and I.A. Baird. (1989). Using mathematical programming to address the multiple reserve selection problem: An example from the Eyre Peninsula, South Australia, *Biological Conservation*, 49: 113-130.

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

Cowling, R.M. and R.L. Pressey. (2003). Introduction to systematic conservation planning in the Cape Floristic Region. *Biological Conservation*, 112:1-13.

Cowling, R.M., R.L. Pressey, M. Rouget and A.T. Lombard. (2003). A conservation plan for a global biodiversity hotspot - the Cape Floristic Region, South Africa. *Biological Conservation*, 112:191-216.

Cowling, R.M., R.L. Pressey, R. Sims-Castley, A. le Roux, E. Baard, C.J. Burgers and G. Palmer G. (2003). The expert or the algorithm? - Comparison of priority conservation areas in the Cape Floristic Region identified by park managers and reserve selection software. *Biological Conservation*, 112: 147-167.

Cowling, R.M., A.T. Knight, D.P. Faith, S. Ferrier, A.T. Lombard, A. Driver, M. Rouget, K. Maze and P.G. Desmet. (2004). Nature conservation requires more than a passion for species. *Conservation Biology*, 18: 1674-1676.

Csuti, B., S. Polasky, P.H. Williams, R.L. Pressey, J.D. Camm, M. Kershaw, A.R. Kiester, B. Downs, R. Hamilton, M. Huso and K. Sahr. (1997). A comparison of reserve selection algorithms using data on terrestrial vertebrates in Oregon. *Biological Conservation*, 80: 83- 97.

Desmet, P. and R.M. Cowling. (2004). Using the species-area relationship to set baseline targets for conservation. *Ecology and Society*, 9(2): 11.

Erasmus, B.F.N., S. Freitag, K.J. Gaston, B.H. Erasmus and A.S. van Jaarsveld. (1999). Scale and conservation planning in the real world. *Proceedings of the Royal Society of London Series B-Biological Sciences*, 266: 315-319.

Ferdaña, Z. (2005). Nearshore marine conservation planning in the PacificNorthwest: Exploring the use of a siting algorithm for representing marine biodiversity, in D.J. Wright and A.J. Scholz (Eds.), *Place Matters: Geospatial Tools, for Marine Science, Conservation, and Management in the Pacific Northwest*. Corvallis, OR: Oregon State University Press.

Fernandes, L., J. Day, A. Lewis, S. Slegers, B. Kerrigan, D. Breen, D. Cameron, B. Jago, J. Hall, D. Lowe, J. Innes, J. Tanzer, V. Chadwick, L. Thompson, K. Gorman, M. Simmons, B. Barnett, K. Sampson, G. De'ath, B. Mapstone, H. Marsh, H. Possingham, I. Ball, T. Ward, K. Dobbs, J. Aumend, D. Slater and K. Stapleton. (2005). Establishing representative no-take areas in the Great Barrier Reef: Large-

scale implementation of theory on marine protected areas. *Conservation Biology,* 19: 1733-1744.

Ferrier, S. (2002). Mapping spatial pattern in biodiversity for regional conservation planning: Where to from here? *Systematic Biology*, 51(2): 331-363.

Ferrier, S., G. Watson, J. Pearce, M. Drielsma. (2002). Extended statistical approaches to modelling spatial pattern in biodiversity in northeast New South Wales. I. Species-level modelling. *Biodiversity and Conservation*, 11: 2275-2307.

Ferrier, S., M. Drielsma, G. Manion, G. Watson. (2002). Extended statistical approaches to modelling spatial pattern in biodiversity in northeast New South Wales. II. Community-level modeling. *Biodiversity and Conservation,* 11: 2309-2338.

Leslie, H., M. Ruckelshaus, I.R. Ball, S. Andelman and H.P. Possingham. (2003). Using siting algorithms in the design of marine reserve networks. *Ecological Applications,* 13: S185-S198.

Higgins, J.V., M.T. Bryer, M.L. Khoury, and T.W. Fitzhugh. (2005). A freshwater classification approach for biodiversity conservation planning. *Conservation Biology* 19(2): 432-445.

Kirkpatrick, J.B. (1983). An iterative method for establishing priorities for selection of nature reserves: An example from Tasmania. *Biological Conservation*, 25: 127-134.

Kirkpatrick J.B. and M.J. Brown. (1994). A comparison of direct and environmental domain approaches to planning reservation of forest higher plant communities in Tasmania. *Conservation Biology,* 8: 217-224.

Knight, A.T., R.M. Cowling and B.M. Campbell. (2006). An operational model for implementing conservation action. Conservation Biology, 20: 408-419.

Leslie, H., M. Ruckelshaus, I.R. Ball, S. Andelman and H.P. Possingham. (2003). Using siting algorithms in the design of marine reserve networks. *Ecological Applications*, 13(1): S185-S198.

Lombard A.T., R.M. Cowling. R.L. Pressey and A.G. Rebelo. (2003). Effectiveness of land classes as surrogates for species in conservation planning for the Cape Floristic Region. *Biological Conservation*, 112: 45-62.

McDonald, R., M. McKnight, D. Weiss, E. Selig, M. O'Connor, C. Violin. and A Moody. (2005). Species compositional similarity and ecoregions: Do ecoregion boundaries represent zones of high species turnover*? Biological Conservation*, 126: 24-40.

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

Margules, C.R. and R.L Pressey. (2000). Systematic conservation planning . *Nature*, 405: 243-253.

Meir, E., S. Andelman and H.P. Possingham. (2004). Does conservation planning matter in a dynamic and uncertain world? *Ecology Letters,* 7: 615-622.

Naidoo, R. and W.L. Adamowicz. (2006). Modeling opportunity costs of conservation in transitional landscapes. *Conservation Biology*, 20: 490-500.

Naidoo, R., A. Balmford, P.J. Ferraro, S. Polasky, T.H. Ricketts and M. Rouget. (2006). Integrating economic costs into conservation planning. *Trends in Ecology and Evolution*, 21: 681-687.

Nicholls, A.O. and C.R. Margules. (1993). An upgraded reserve selection algorithm. *Biological Conservation*, 64: 165-169.

Nicholson, E., and H.P. Possingham. (2006). Objectives for multiple species conservation planning. *Conservation Biology,* 20: 871-881.

Noss, R.F. (2004). Conservation targets and information needs for regional conservation planning. *Natural Areas Journal*, 24: 223-231.

Pierce S.M., R.M. Cowling, A.T. Knight, A.T. Lombard, M. Rouget and T. Wolf. (2005). Systematic conservation planning products for land-use planning: Interpretation for implementation. *Biological Conservation*, 125: 441-458.

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

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

Possingham, H.P., J. Franklin, K.A. Wilson and T.J. Regan. (2005). The roles of spatial heterogeneity and ecological processes in conservation planning. In: G.M. Lovett, C.G. Jones, M.G. Turner and K.C. Weathers (Eds). *Ecosystem function in heterogeneous landscapes* (pp 389-406). New York: Springer-Verlag.

Pressey, R.L. (2002). The first reserve selection algorithm. *Progress in Physical Geography*, 26(3): 434-441.

Pressey, R.L. (2004). Conservation planning and biodiversity: Assembling the best data for the job. *Conservation Biology*, 18(6): 1677-1681.

Pressey, R.L., M. Cabeza, M.E. Watts, R.M. Cowling and K.A. Wilson. (in press) Conservation planning in a changing world. *Trends in Ecology and Evolution*.

Pressey, R.L., R.M. Cowling and M. Rouget. (2003). Formulating conservation targets for biodiversity pattern and process in the Cape Floristic Region, South Africa. *Biological Conservation*, 112: 99-127.

Pressey, R.L., I.R. Johnson and P.D. Wilson. (1994). Shades of irreplaceability: Towards a measure of the contribution of sites to a reservation goal. *Biodiversity and Conservation*, 3: 242-262.

Pressey, R.L. and V.S. Logan. (1994). Level of geographical subdivision and its effects on assessments of reserve coverage: A review of regional studies. *Conservation Biology*, 8: 1037-1046.

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

Pressey, R.L., P.H. Possingham, and J.R. Day. (1997). Effectiveness of alternative heuristic algorithms for identifying indicative minimum requirements for conservation reserves. *Biological Conservation*, 80: 207-219.

Pressey, R.L. and K.H. Taffs. (2001). Scheduling conservation action in production landscapes: Priority areas in western New South Wales defined by irreplaceability and vulnerability to vegetation loss. *Biological Conservation*, 100: 355-376.

Pressey, R.L., M.E. Watts, and T.W. Barret. (2004). Is maximizing protection the same as minimizing loss? Efficiency and retention as alternative measures of the effectiveness of proposed reserves. *Ecology Letters*, 7: 1035-1046.

Pressey, R.L., G.L. Whish, T.W. Barrett and M.E. Watts. (2002). Effectiveness of protected areas in north-eastern New South Wales: Recent trends in six measures. *Biological Conservation*, 106: 57-69.

Rebelo, A.G. and W.R. Siegfried. (1992). Where should nature reserves be located in the Cape Floristic Region, South Africa? Models for the spatial configuration of a reserve network aimed at maximizing the protection of floral diversity. *Conservation Biology,* 6(2): 243-252.

Richardson, E.A., M.J. Kaiser, G. Edwards-Jones, and H.P. Possingham. (2006). Sensitivity of marine-reserve design to the spatial resolution of socioeconomic data. *Conservation Biology,* 20 (4): 1191–1202.

Rodrigues, A.S.L. and T.M. Brooks. (2007). Shortcuts for biodiversity conservation planning: The effectiveness of surrogates. *Annual Review of Ecology, Evolution, and Systematics*, 38: 713-737.

Rondinini, C., K. Wilson, L. Boitani, H. Grantham, H. Possingham. (2006). Tradeoffs of different types of species occurrence data for use in systematic conservation planning. *Ecology Letters*, 9: 1136-1145.

Sarkar, S., J. Justus, R. Fuller, C. Kelley, J. Garson and M. Mayfield. (2005). Effectiveness of environmental surrogates for the selection of conservation area networks. *Conservation Biology,* 19: 815-825.

Sarkar S., R.L. Pressey, D.P. Faith, C.R. Margules, T. Fuller, D.M. Stoms, A. Moffett, K. Wilson, K.J. Williams, P.H. Williams and S. Andelman. (2006). Biodiversity conservation planning tools: Present status and challenges for the future. *Annual Review of Environment and Resources*, 31: 123-159.

Smith, R., P. Goodman and W. Matthews. (2006). Systematic conservation planning: A review of perceived limitations and an illustration of the benefits using a case study from Maputaland, South Africa. *Oryx*, 40: 400-410.

Stewart, R.R. (2003). Opportunity cost of ad hoc marine reserve design decisions: An example from South Australia. *Marine Ecology Progress Series*, 253: 25-38.

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

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

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

Underhill, L.G. (1994). Optimal and suboptimal reserve selection algorithms. *Biological Conservation*, 70: 85-87.

Warman, L.D., A.R.E. Sinclair, G.G.E. Scudder, B. Klinkenberg, and R.L. Pressey. (2004). Sensitivity of systematic reserve selection to decisions about scale, biological data, and targets: Case study from southern British Columbia. *Conservation Biology,* 18: 655-666.

Wilson, K.A., R.L. Pressey, A.N. Newton, M.A Burgman, H.P. Possingham and C.J. Weston. (2005). Measuring and incorporating vulnerability into conservation planning. *Environmental Management*, 35: 527-543

Wilson, K.A., A.N. Newton, C. Echeverría , C.J. Weston and M.A. Burgman. (2005). A vulnerability analysis of the temperate forests of south central Chile. *Biological Conservation*, 122: 9-21.

Wilson, K.A., M.I. Westphal, H.P. Possingham and J. Elith. (2005). Sensitivity of conservation planning to different approaches to using predicted species distribution data. *Biological Conservation*, 122: 99-112.

Wilson, K.A., M. McBride, M. Bode, and H.P. Possingham. (2006). Prioritising global conservation efforts. *Nature*, 440: 337-340.

Winter, S.J., K.J. Esler and M. Kidd. (2005). An index to measure the conservation attitudes of landowners towards Overberg Coastal Renosterveld, a critically endangered vegetation type in the Cape Floral Kingdom, South Africa. *Biological Conservation*, 126: 383-394.

Zacharias, M.A. and J.C. Roff. (2000). A hierarchiacal ecological approach to conserving marine biodiversity. *Conservation Biology*, 14(5): 1327-1334.

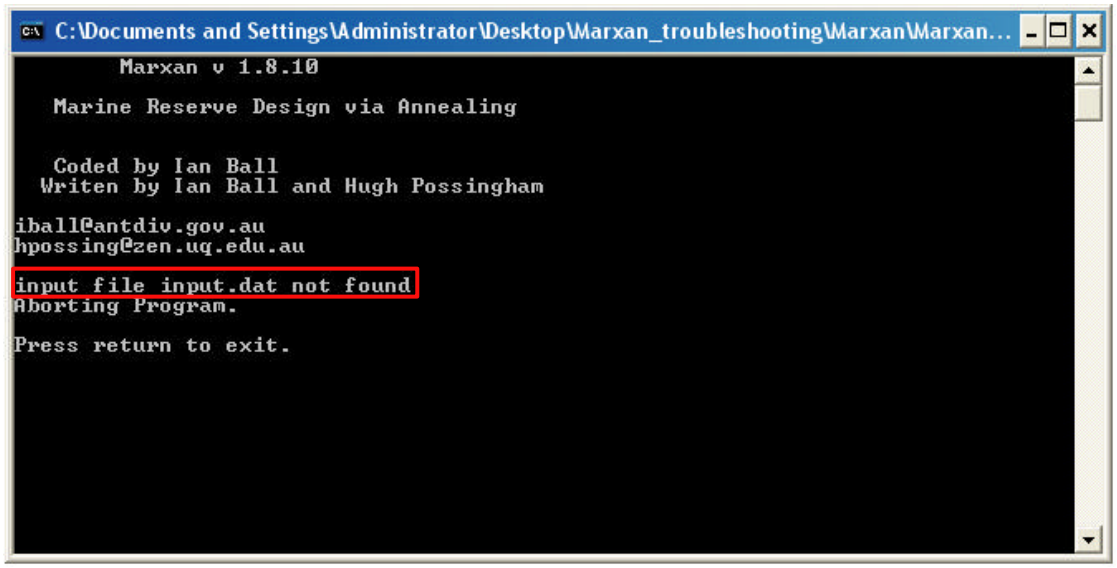
Zacharias, M.A. and J.C. Roff. (2001). Use of focal species in marine conservation and management: A review and critique. *Aquatic Conservation: Marine and Freshwater Ecosystems*, 11: 59-76.

(This page intentionally blank)

Appendix A – Troubleshooting

In this section we provide examples of common error messages encountered while running Marxan, and the sort of mistakes that can cause these messages to be generated.

A-1. Marxan halts because a required input file or parameter has not been found

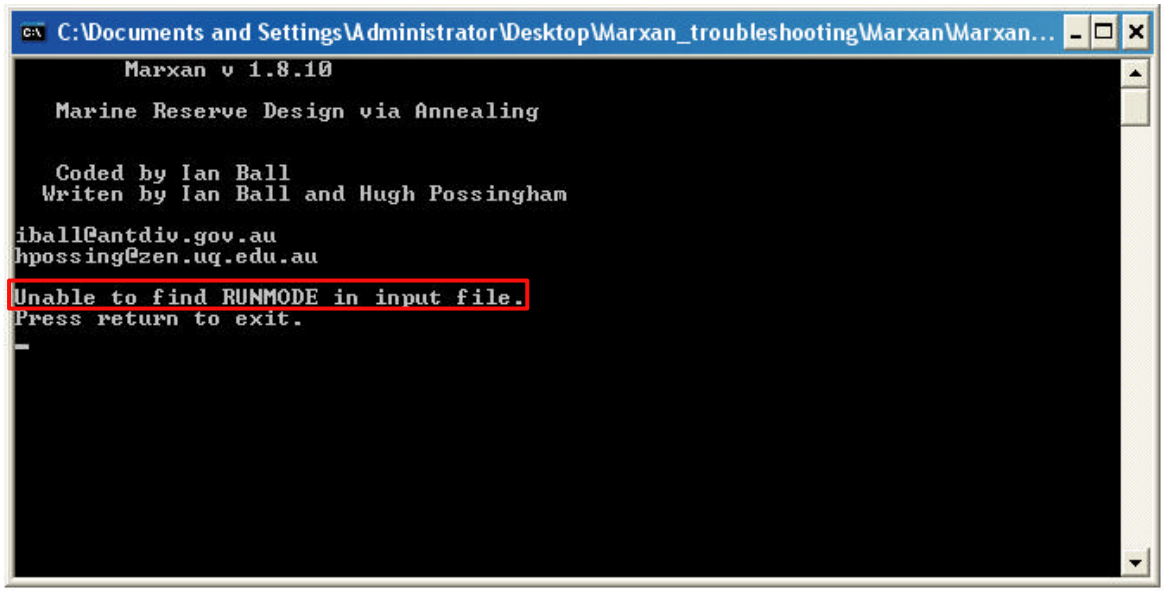


Marxan will halt with this message (above) if it is unable to locate the **Input Parameter File** (input.dat).

What to do.

1. Check that the **Input Parameter File** and the Marxan executable (Marxan.exe) are located in the same directory.
2. Check that the **Input Parameter File** name and extension read exactly – "input.dat".

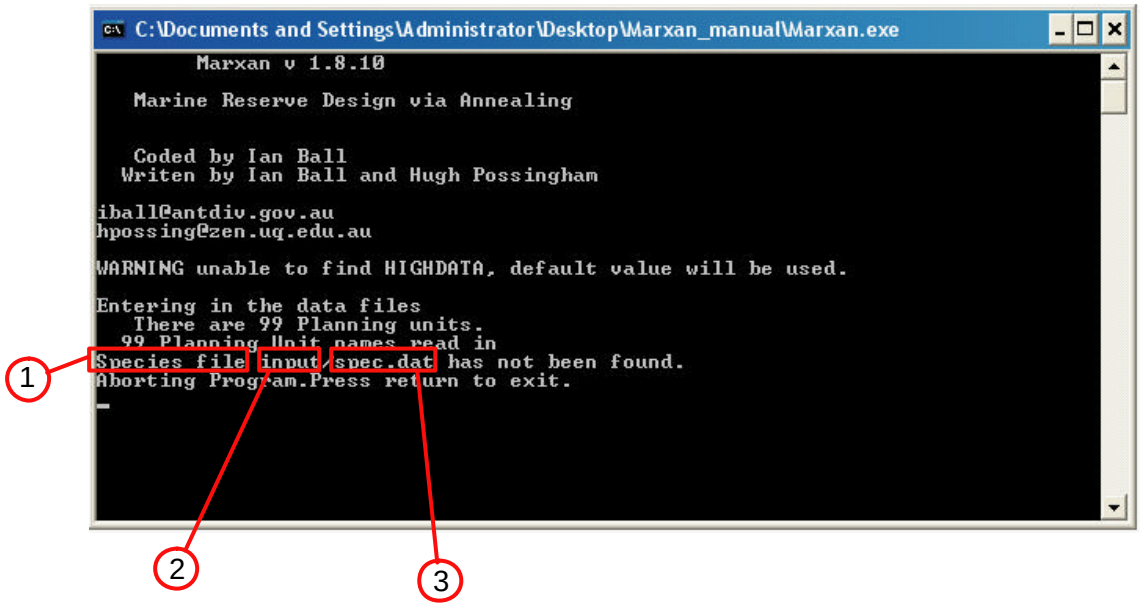
Marxan will halt with a similar error message (below) if it cannot find the critical parameter ‘RUNMODE’.



What to do.

1. Check that there is a value (and a valid value) for this parameter in **the Input Parameter File**. There is no default value for ‘RUNMODE’.
2. Check that the parameter is labeled correctly; it must be in all capitals with no spaces or extra characters. (This is done automatically if **Inedit** is used.)

Marxan will halt with this message (below) if it has found the **Input Parameter File** but has been unable to locate one of the remaining three required input files.

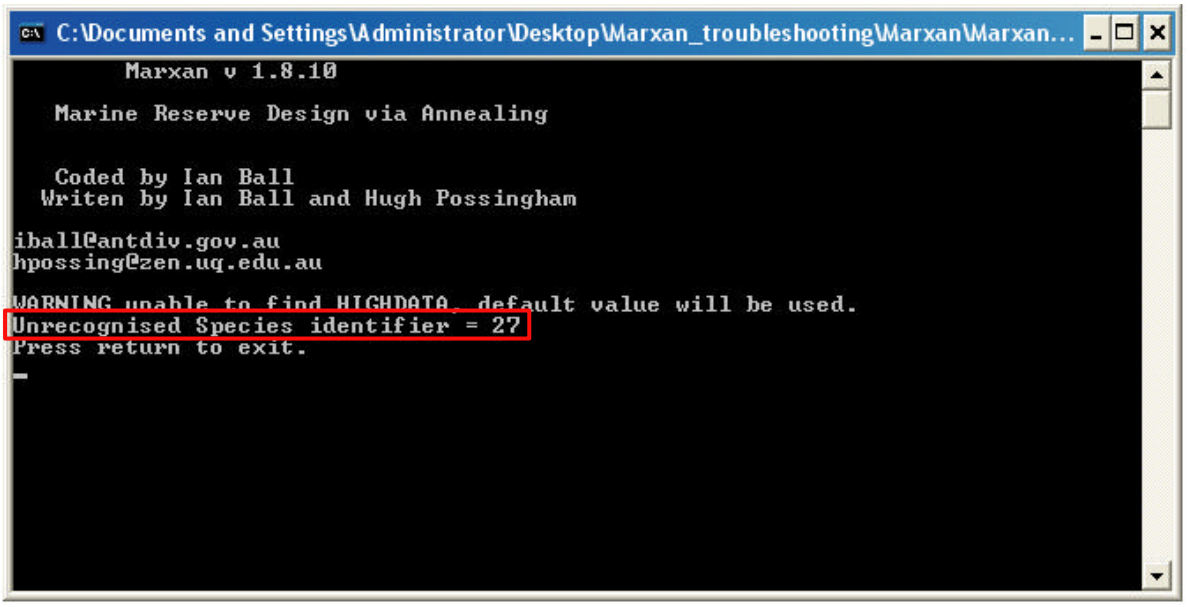


1. The type of input file that is missing (in this example it is the **Conservation Feature File**, a.k.a. the Species File).
2. The name of the directory that Marxan is looking for the input files in. This is the name specified in the **Input Parameter File**.
3. The name of the file Marxan is looking for. This is the name specified in the **Input Parameter File**.

What to do.

1. Check that the name of the directory containing the input files is the same as the name given in the **Input Parameter File** under the variable, Input Directory ('INPUTDIR').
2. Check that the directory containing the input files is either located within the same directory as the Marxan executable (Marxan.exe) or the location has been correctly specified in the **Input Parameter File**.
3. Check that the name of the file given in the **Input Parameter File** exactly matches the name in the input file directory.

A-2. Marxan halts because of an unrecognised identifier



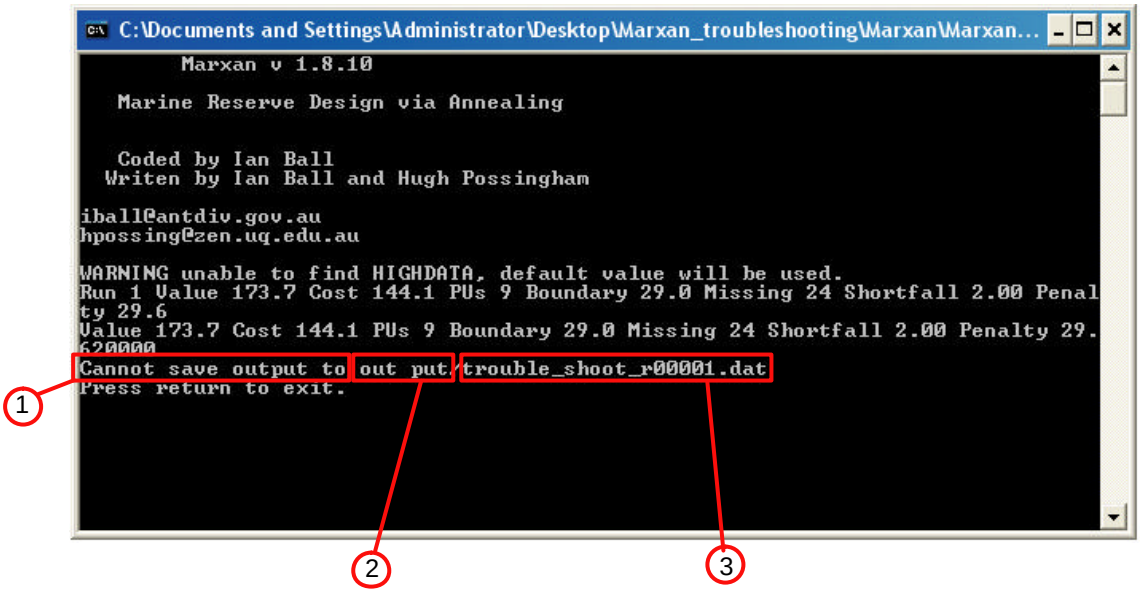
In this example (above) the error is because of an inconsistency in the Conservation Feature (a.k.a. Species) IDs. This error could also have the term “Planning Unit” instead of “Species”.

This error will occur if there is inconsistency in either the Conservation Feature IDs or Planning Unit IDs between files. In other words the IDs listed in **the Conservation Feature File** or **Planning Unit File** are not the same as those in the **Planning Unit versus Conservation Feature File** (and vice versa).

What to do.

1. Check that conservation feature IDs listed in the **Planning Unit versus Conservation Feature File** exactly match the IDs in the **Conservation Feature File**.
2. Check that planning unit IDs listed in the **Planning Unit versus Conservation Feature File** exactly match the IDs in the **Planning Unit File**.

A-3. Marxan begins the first run but then halts because it is unable to save the required outputs

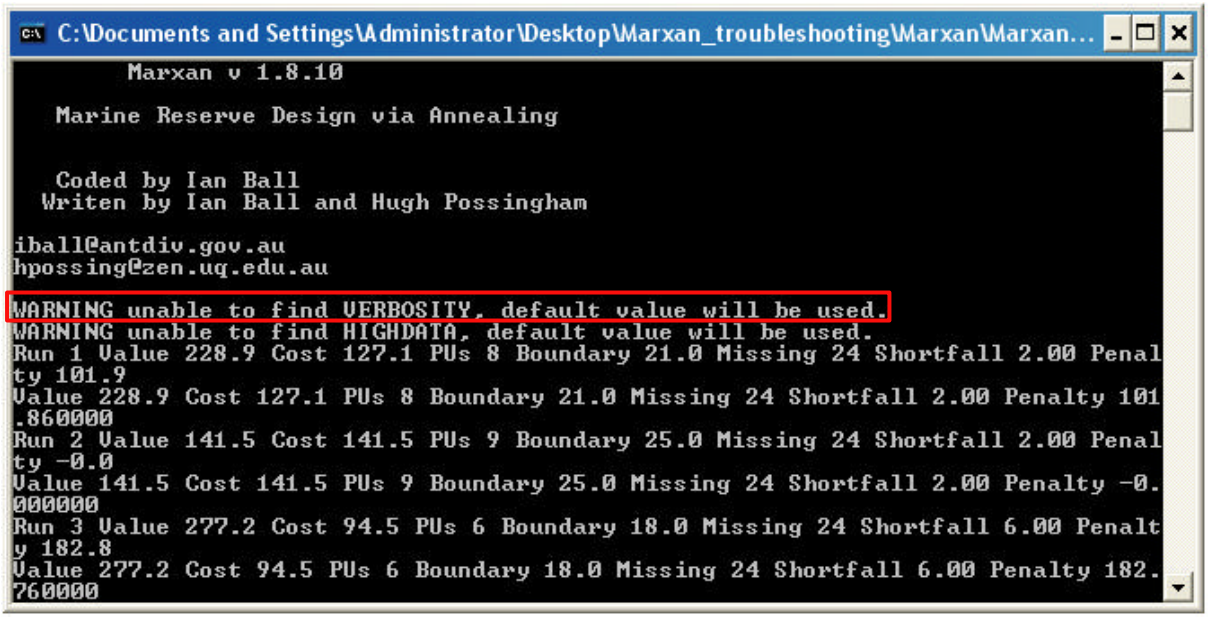


1. The message identifying that Marxan is unable to save the desired output.
2. The name of the directory Marxan is trying to save the output files in (in the example the error is because of a space between the words ‘out’ and ‘put’).
3. The name of the file Marxan is trying to solve when it encounters the error.

What to do.

1. Check that the name of the directory you have established to save the output files in is the same as the name given in the **Input Parameter File** under the variable, Output Directory (‘OUTPUTDIR’).
2. Check that this directory is either located within the same directory as the Marxan executable (Marxan.exe) or the location has been correctly specified in the **Input Parameter File** (This is easily done using **Inedit**).
3. Check that the name of the file given in the **Input Parameter File** exactly matches the name in the input file directory.

A-4. Marxan runs but warns you it is unable to find a particular variable



Marxan will report this warning when it is unable to read the value for some parameter in the **Input Parameter File**. This may either be because a value was not specified, or there was a formatting error. Marxan will still run normally but will use the default value for that parameter.

What to do.

1. If you intentionally did not state a value and are happy for the default value to be used, do nothing. Marxan will run as normal.
2. If you do not wish to use the default value, check the following:
    - That the **Input Parameter File** (input.dat) contains a value for the missing parameter.
    - That the parameter is correctly identified in the **Input Parameter File**. Its name must be in all capitals with no spaces or additional characters.
    - Check that there is a hard return at the end of the **Input Parameter File**.

Note: Do NOT worry about the line that reads “WARNING unable to find HIGHDATA”, this refers to a now defunct parameter in Marxan. Marxan will run correctly and as intended even when this warning is displayed. This error message will not show up in future versions of Marxan.

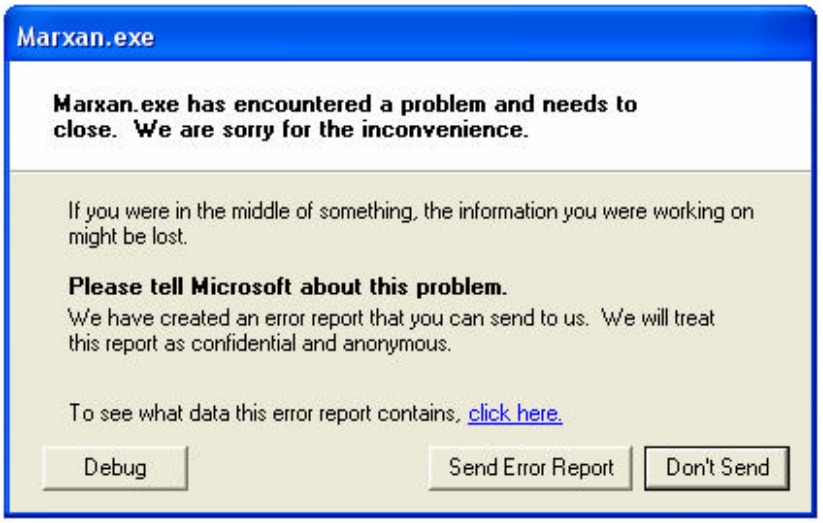
A-5. No outputs are being saved in the output directory

What to do.

1. Check that an output directory has actually been specified in the **Input Parameter File**. If no directory is specified then Marxan will not save any outputs.
2. Check that you have selected the outputs that should be saved. This can be done by looking directly at the **Input Parameter File**; all output to be saved should have the value '1' next to them. Outputs you don't want saved should have the value '0' next to them.

A-6. Marxan crashes as soon as it is executed and the Marxan screen closes

This may happen for a variety of reasons but most typically if there is a value missing from the input data files. If you are running Marxan on a Windows machine this may also be accompanied by the following error message:



If this message appears, please feel free to let Microsoft know. We very much doubt, however, that they will do anything about it. Marxan is not a big money earner for them.

What to do.

1. Check that there are values for all variables in each of the input data files. This includes the three required files (**Conservation Feature File**, **Planning Unit File** and the **Conservation Feature versus Planning Unit File**), and the two optional files (the **Boundary Length File** and the **Block Definition File**). For each of these files there must be a complete set of values for all required variables and any of the non-required variables that are being used.
2. Check that there is not more than a single hard return (blank lines) at the end of each of your input file.

(This page intentionally blank)

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. It was briefly introduced in Section 1.5. This section provides information about how each of the components in the objective function are calculated. The objective function in Marxan is designed so that the lower the value the better. It can potentially include the following components:

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

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** (see Section 3.2.3). 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. See Section 3.2.3.2 for a discussion about planning unit cost.

B-1.2 Boundary and Boundary Length Modifier (BLM)

'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** (see Section 3.3.1). 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. Section 3.3.1.2 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 Penalty and Species Penalty Factor (SPF)

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:

I. 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.

II. Select the planning unit with the lowest cost-per-hectare.
Add its cost to the running cost total and the level of representation for the conservation feature to the representation level total.

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

IV. 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[10]. 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.

[10] See Section B-2.3.1 of this appendix for a description of greedy heuristics.

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 Conservation Feature Penalty Factor, see Section 3.2.2.4). 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-1.3.1 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 (see Section 3.2.2.5), and a minimum separation distance required between multiple occurrences of the same feature (see Section 3.2.2.9). Both of these can be specified in the **Conservation Feature File** (see Section 3.2.2).

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 (see Section 3.2.2.5). 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 (see Section 3.2.1.7.3). 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

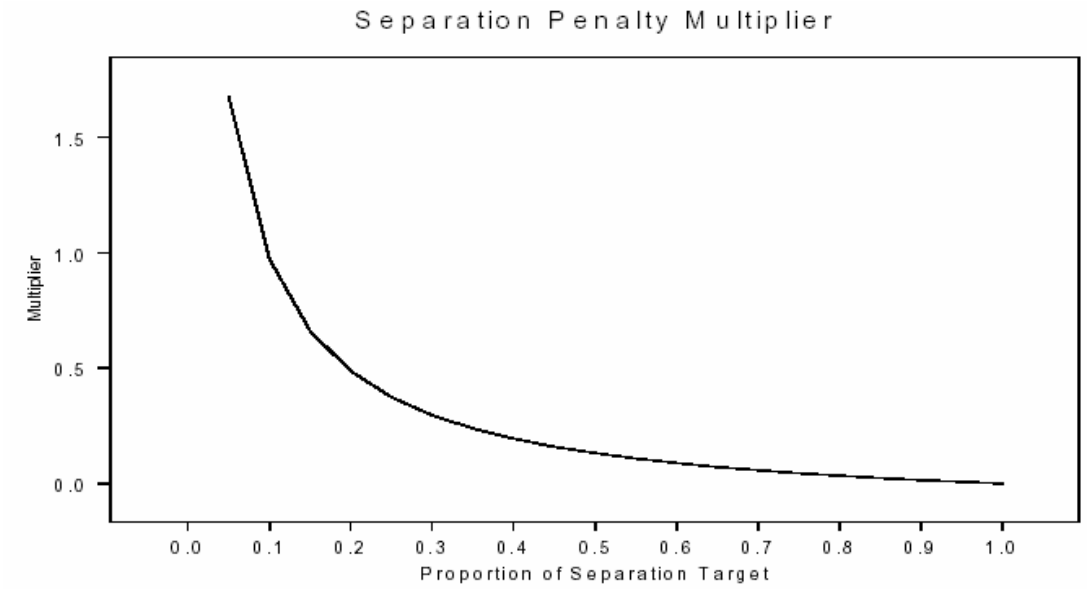
conservation features of large size). They could be used where clumping is to be encouraged but the minimum clump size is not as rigid as a minimum viable population size.

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 (see Section 3.2.2.9). 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:

$$penalty = \frac{1}{7 * C_p + 0.2} - \frac{1}{7.2} \quad .$$

Here *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. As the separation count increases from 1/target separation count to 1 this penalty looks like:



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-1.4 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 doesn't 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:

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

Here *t* is the time during the run which must be between 0 (start of the run) and 1 (end of the run). *A* and *b* are control parameters. *b* controls the shape control of the penalty curve, in other words how gradually the penalty is applied (if it is set high, the penalty will vary little until late in the run). *A* controls the size of the penalty. If this is set high, exceeding the threshold will be penalized very heavily. A lower value for *A* might allow the threshold to be slightly exceeded. The penalty always starts at 0 when *t* is zero. The value for both *A* and *b* will require some experimentation to set appropriately and both can be modified on the 'Cost Threshold' tab of **Inedit** (see Section 3.2.1.6). In **Inedit**, *A* and *b* are listed as 'Penalty Factor A' and 'Penalty Factor B'. 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. More information about which optimization method you might use under different circumstances can be found in Section 3.2.1.2.2. 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:

$$Tinitial = MinChange + 0.1 \text{ x } \left(MaxChange - MinChange \right)$$

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 simply by ticking the 'Adaptive Annealing' box on the **Inedit** tab, 'Annealing' (see Section 3.2.1.3), or 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.

B-2.1.3 Setting a Fixed Annealing Schedule

First, set the "verbosity" of the algorithm to "Detailed Progress" (see Section 3.2.1.5.1) so that you can see the simulated annealing at work.

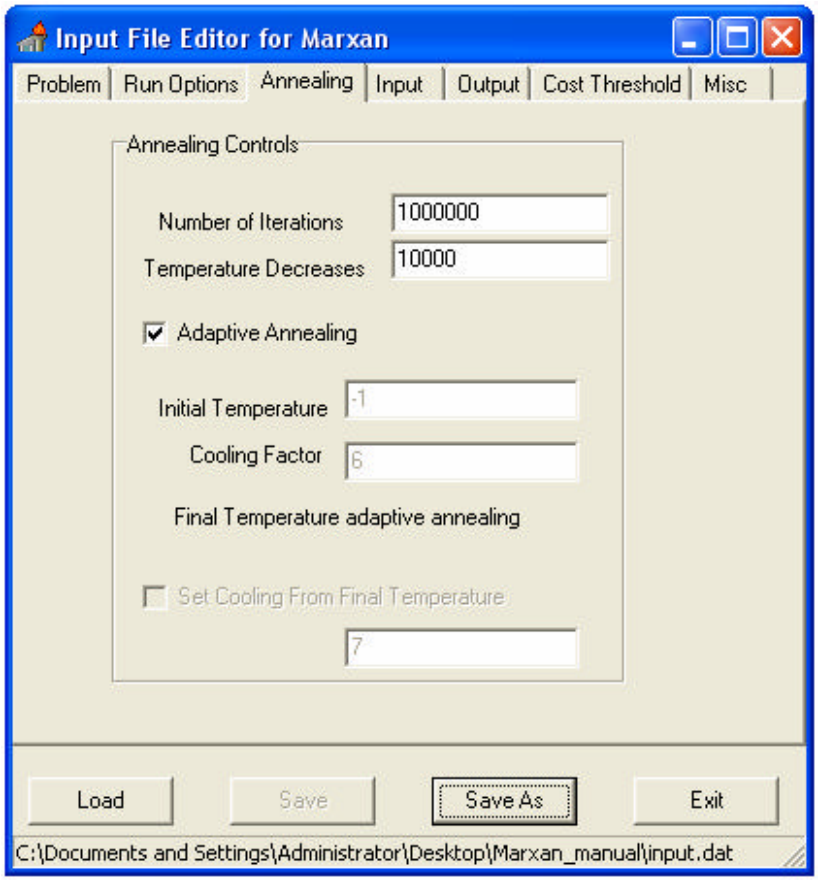
When setting a fixed annealing schedule the two parameters that need to be set are the initial and final temperature. The final temperature is set by choosing an appropriate value for the cooling function. If the final temperature is too low then the algorithm will spend a lot of time "stuck" at a local 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 also 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 on the **Inedit** tab labeled 'Annealing' (see below). Alternatively they can be set directly in the 'input.dat' file using the variable names, 'STARTTEMP' and 'COOLFAC' (see Section 3.2.1.3.1).



The 'Annealing' tab of the program **Inedit**. The different parameters that control the annealing schedule can be set here.

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 either be selected on the 'Run Options' page of **Inedit** (see Section 3.2.1.2), or specified directly in the 'input.dat' file using the variable 'ITIMTYPE' (see Section 3.2.1.2.2).

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 (Pressey et al. 1993). 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 a few of main types of heuristics upon which the others are variations. 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.

B-2.3.1.1 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.

B-2.3.1.2 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 have been titled; 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.

B-2.3.2.1 Maximum Rarity

This method scores each planning unit according to the formula:

$$\frac{\text{Effective Abundance}}{(\text{Rarity x Planning Unit Cost})} .$$

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 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.

B-2.3.2.2 Best Rarity

Best rarity is very similar to the maximum rarity heuristic described above. The same formula is used:

$$\frac{\text{Effective Abundance}}{(\text{Rarity x Planning Unit Cost})} .$$

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.

B-2.3.2.3 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:

$$\frac{\sum_{cons.\,features} \frac{\text{Effective Abundance}}{\text{Rarity}}}{\text{Cost of Planning Unit}}$$

B-2.3.2.4 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:

$$\frac{\sum_{cons.\,features} \frac{\text{Effective Abundance}}{\text{Rarity}}}{\text{Cost x Number of Conservation Features}}$$

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 for a 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:

$$\textit{Irreplaceability}\begin{cases}\dfrac{\text{Buffer - Effective Abundance}}{\text{Buffer}}, & \text{Buffer} > 0\\ 0, & \text{Buffer} = 0\end{cases}$$

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 (see below).

B-2.3.3.1 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.

B-2.3.3.2 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.

(This page intentionally blank)

Appendix C – Advice on developing Marxan input files and displaying results in GIS

There are many ways of developing the 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. Our aim here is to point you towards useful tools and resources rather than provide step by step advice. Most of the advice is based on ESRI ArcView 3 and ArcGIS 8 and 9, and the user friendly interfaces for Marxan; Conservation Land-Use Zoning (CLUZ) and P.A.N.D.A.

C-1 Resources

C-1.1 Software

You will need at minimum a spreadsheet or textpad type program to develop, read and manipulate the required input text 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. Another useful program for manipulating input files is the C-Plan table editor, which is freely available when C-Plan is downloaded from http://www.uq.edu.au/~uqmwatts/cplan.html.

For spatial analysis Geographic Information Systems (GIS) software is required. The most popular are the ESRI family of software (www.esri.com). This includes ArcView 3 and ArcGIS 8 or 9. Due to the popularity of these programs there are many extensions that can be downloaded to expand their functionality and automate functions for more user friendly manipulation of data. We recommend a few as we describe methods of creating the Marxan input files below. Note that there are many excellent free open source GIS software packages available for download such as MapWindow (www.mapwindow.com), Quantum GIS (www.qgis.org), and GRASS (http://grass.itc.it/). Another free alternative is DIVA (http://www.diva-gis.org/). Currently they do not have the same functionality as the ESRI ones, but various programmers are working on scripts to expand their functionality including the tools necessary for developing Marxan input files. Later versions of this manual will include updates on this situation.

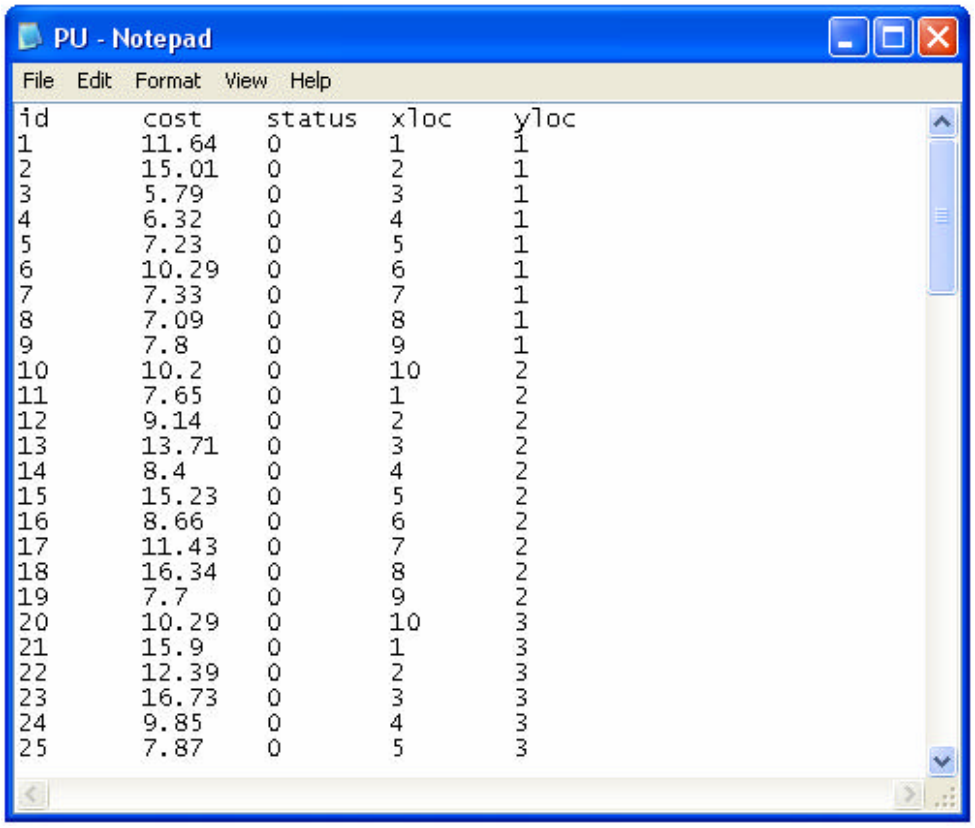
Also, there are some free software tools that simplify running Marxan and creating the Marxan input files. Three of these are: CLUZ that operates through ArcView 3

(http://www.mosaic-conservation.org/cluz/index.html), C-Plan that also operates through ArcView 3 (http://www.uq.edu.au/~uqmwatts/cplan.html) and P.A.N.D.A that requires ArcGIS 8 and 9 (http://www.mappamondogis.it/panda_en.htm). There are future plans for similar software to work through free GIS software. The Nature Conservancy has developed "Protected Area GAP Decision-Support System for ArcGIS 9.1" that facilitates Marxan analysis, available from ftp://cerp:cerppassword@ftp.tnc.org/CDSS/Protected_Area_Gap_DSS.zip Lastly there is Vista that operates through ArcGIS 9 that is not free, for free trial see http://www.natureserve.org/prodServices/vista/overview.jsp.

C-1.2 Courses and tutorials

The University of Queensland offers courses on the use of Marxan and systematic conservation planning. The details can be found at the Marxan website http://www.ecology.uq.edu.au/index.html?page=27710 also see http://www.aeda.edu.au/events. If you are not able to attend, the course materials, including the tutorials, are available to download. This includes step by step advice on developing input files and running Marxan. There are also often a short-course on conservation planning and software such as Marxan as part of the Society for Conservation Biology annual conference http://www.conbio.org/. For information on past short-courses (including presentations), software demonstrations (including downloadable data and instructions on using Marxan) see http://www.aeda.edu.au/events. A useful tutorial on using the CLUZ interface for Marxan that can be downloaded at http://www.mosaic-conservation.org/cluz/tutorial.html and a step-by-step guide for running conservation assessments using CLUZ and Marxan is available from http://www.mosaic-conservation.org/cluz/steps.html.

C-2 Creating the planning unit file



id	cost	status	xloc	yloc
1	11.64	0	1	1
2	15.01	0	2	1
3	5.79	0	3	1
4	6.32	0	4	1
5	7.23	0	5	1
6	10.29	0	6	1
7	7.33	0	7	1
8	7.09	0	8	1
9	7.8	0	9	1
10	10.2	0	10	2
11	7.65	0	1	2
12	9.14	0	2	2
13	13.71	0	3	2
14	8.4	0	4	2
15	15.23	0	5	2
16	8.66	0	6	2
17	11.43	0	7	2
18	16.34	0	8	2
19	7.7	0	9	2
20	10.29	0	10	3
21	15.9	0	1	3
22	12.39	0	2	3
23	16.73	0	3	3
24	9.85	0	4	3
25	7.87	0	5	3

An example of the **Planning Unit File**.

When producing the input files, it is first necessary to have a spatial layer of planning units. Planning units are not, however, constrained to being square, any division is acceptable. Other examples include hexagons, cadastral parcels and hydrological units. Some thought should be put into determining the most appropriate type and size of planning unit for each specific application see the references section for advice. To create a planning unit theme that represents entities such as cadastral boundaries or watersheds just use an existing spatial layer that provides the boundaries of these polygons , otherwise it will be necessary to produce your own layer.

For ArcView 3 users:

1. Either use existing planning units if available or develop new ones. If developing new ones a useful extension is http://arcscripts.esri.com/details.asp?dbid=14329. This extension generates an array of repeating shapes over a user-specified area. These shapes can be hexagons, squares, triangles, circles or points, and they can be generated with any directional orientation. Alternatively ET GeoWizards has a function to create grids www.ian-ko.com/
2. *Use your existing protected areas theme and add a field called "ID" to the protected areas theme table. Give each protected area a unique identifier number. Either:
    - Use the “update polygon theme” in Xtools http://arcscripts.esri.com/details.asp?dbid=11526
    - Use the “erase” function in ET geowizards www.ian-ko.com/ to erase planning units where protected areas are located. Then “union” the protected areas layer with the erased planning units.
    - Use ArcView's “union” option in the Geoprocessing Wizard to combine the planning units and the protected areas into a new theme. The new theme will divide each protected area into a number of whole regular polygons and some fragments. Use the “dissolve” option in the Geoprocessing Wizard to merge all these pieces into one polygon for each protected area. In the dialog box, for the Select an attribute to dissolve input box choose the " ID" field.
3. Ensure that each planning unit has a unique identifier that can be used for the “id” field.
4. Add a field “status” and use the protected areas layer to “select by theme” planning units that are protected. After selecting already protected planning units, in the theme table add “2” in the “status” field for these planning units. This process can be repeated for planning units if you want particular areas to be excluded. In the theme table add “3” in the “status” field.
5. To calculate the cost field you need at a minimum a spatial cost surface.
6. The next step is to calculate the cost of each planning unit.
    - Set view properties. The analysis extent should be set to your planning unit layer, and the analysis grid size should be set small enough to allow accurate results whilst being large enough to be tractable to run (eg. 10m or 100m depending on regions size)
    - If it is a vector layer convert to a raster**

- Use ‘summarise zones’ with the cost layer and planning units. Join the output table with the planning unit layer. Or, zonal statistics*** can be applied using various user-created extensions such as “Mila Grid Utilities” http://www.mila.ucl.ac.be/logistique/sig/sig-tools/milagrid/index.html

7. To create the X and Y fields, Xtool **** http://arcscripts.esri.com/details.asp?dbid=14329 has a function to do this.
8. Your theme table should contain all necessary fields to create the Marxan **Planning Unit File**. Export table and open in notepad or a spreadsheet program to clean-up, check and save as a text file with the “.dat” extension.

CLUZ users will need to follow steps 1-2 shown above to create the planning unit theme but the remaining steps can be undertaken using the “Create unit theme from shapefile”, “Calculate % of unit that falls within PAs” and “Create unit.dat” functions.

For ArcMap 8 and 9 users:

1. Either use existing planning units if available or develop new ones. If developing new ones a useful extension for creating a grid is ET geowizards http://www.ian-ko.com/
2. *Use your existing protected areas theme and add a field called "ID" to the protected areas theme table. Give each protected area a unique identifier number. Either:
   - Use the “update polygon layer” function in Xtools Pro **** http://www.xtoolspro.com/index.html
   - Use the “erase” function in ET geowizards www.ian-ko.com/ to erase planning units where protected areas are located then “union” the protected areas layer with the erased planning units.
   - Use ArcMap “union” function to combine the planning units and the protected areas into a new theme. The new theme will divide each protected area into a number of whole regular polygons and some fragments. Use the “dissolve” function to merge all these pieces into one polygon for each protected area. Dissolve based on the " ID" field.
3. Ensure that each planning unit has a unique identifier that can be used for the “id” field.
4. Add a field “status” to the planning unit theme. Use the protected areas layer to “select by location” planning units that are protected. In the theme table add “2” in the “status” field for these planning units. This process can be repeated for

planning units that want to be excluded. In the theme table add “3” in the “status” field.

5. To calculate the cost field you need at minimum a spatial cost surface.
    - If it is a vector layer convert to a raster**
    - Use “zonal statistics” *** with the cost layer and planning units. Join the output table to the planning unit layer.
6. To create the X and Y fields, Xtool Pro**** http://www.xtoolspro.com/index.html has a function to do this. Otherwise i) convert feature to points, ii) add XY coordinates and iii) join back.
7. Your theme table should contain all necessary fields to create the Marxan **Planning Unit File**. Export table and open in notepad or a spreadsheet program to clean-up, check and save as a text file with the “.dat” extension.

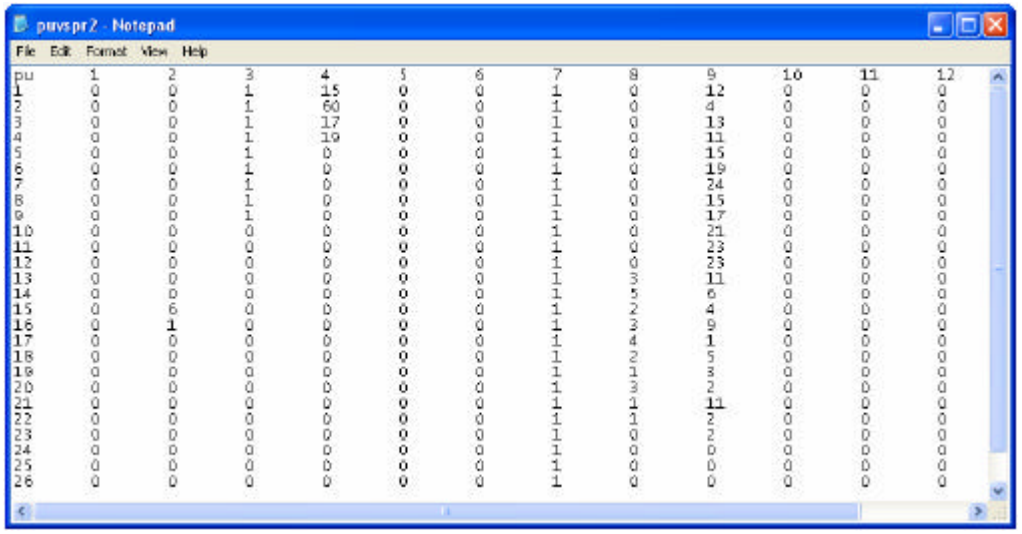
*Some people do not include protected areas as whole planning units but classify their planning units as protected or not based on the proportion of a planning unit already conserved.

**Note that Spatial Analyst extension is required for ArcMap and ArcView 3 to create and use raster files.

***There is often a computational limit to using zonal statistics.

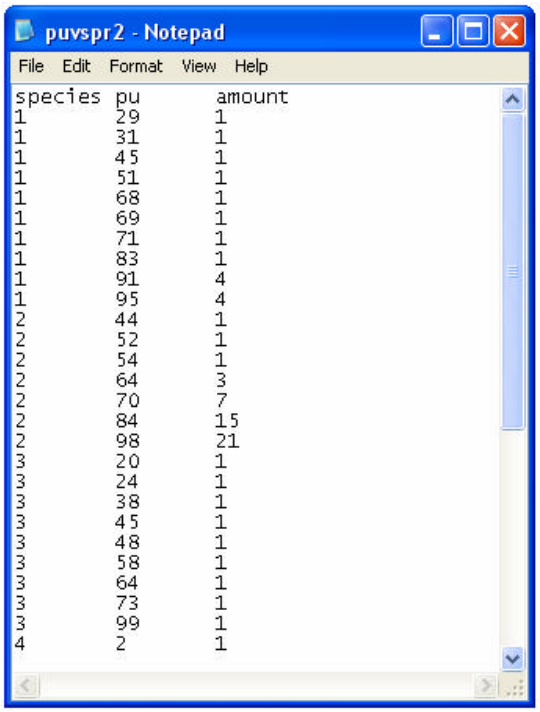
****These function are free to use, others are not.

C-3 Creating the planning unit versus conservation feature file



pu	1	2	3	4	5	6	7	8	9	10	11	12
1	0	0	1	15	0	0	1	0	12	0	0	0
2	0	0	1	60	0	0	1	0	4	0	0	0
3	0	0	1	17	0	0	1	0	13	0	0	0
4	0	0	1	19	0	0	1	0	11	0	0	0
5	0	0	1	0	0	0	1	0	15	0	0	0
6	0	0	1	0	0	0	1	0	19	0	0	0
7	0	0	1	0	0	0	1	0	24	0	0	0
8	0	0	1	0	0	0	1	0	15	0	0	0
9	0	0	1	0	0	0	1	0	17	0	0	0
10	0	0	0	0	0	0	1	0	21	0	0	0
11	0	0	0	0	0	0	1	0	23	0	0	0
12	0	0	0	0	0	0	1	0	23	0	0	0
13	0	0	0	0	0	0	1	3	11	0	0	0
14	0	0	0	0	0	0	1	5	6	0	0	0
15	0	6	0	0	0	0	1	2	4	0	0	0
16	0	1	0	0	0	0	1	3	9	0	0	0
17	0	0	0	0	0	0	1	4	1	0	0	0
18	0	0	0	0	0	0	1	2	5	0	0	0
19	0	0	0	0	0	0	1	1	3	0	0	0
20	0	0	0	0	0	0	1	3	2	0	0	0
21	0	0	0	0	0	0	1	1	11	0	0	0
22	0	0	0	0	0	0	1	1	2	0	0	0
23	0	0	0	0	0	0	1	0	2	0	0	0
24	0	0	0	0	0	0	1	0	0	0	0	0
25	0	0	0	0	0	0	1	0	0	0	0	0
26	0	0	0	0	0	0	1	0	0	0	0	0

An example of the tabular form of the **Planning Unit versus Conservation Feature File**



species	pu	amount
1	29	1
1	31	1
1	45	1
1	51	1
1	68	1
1	69	1
1	71	1
1	83	1
1	91	4
1	95	4
2	44	1
2	52	1
2	54	1
2	64	3
2	70	7
2	84	15
2	98	21
3	20	1
3	24	1
3	38	1
3	45	1
3	48	1
3	58	1
3	64	1
3	73	1
3	99	1
4	2	1

An example of the relational form of the **Planning unit versus conservation feature file**

There are two formats of the **Planning Unit versus Conservation Feature File**: tabular and relational. The relational format is necessary for newer versions of Marxan and allows for faster calculations. If using relational, we recommend first creating a tabular version of the file then converting this file to relational using either the program that is included when you download newer versions of Marxan from: http://www.ecology.uq.edu.au/index.html?page=27710 or the C-Plan Table Editor: http://www.uq.edu.au/~uqmwatts/cplan.html which allows files to be imported (.csv or .dbf so they may need to be converted to these) then exported to relational using the "save as Marxan file" command.

These steps will need to be continually applied to each of the conservation features used in the analysis.

For ArcView 3 Users

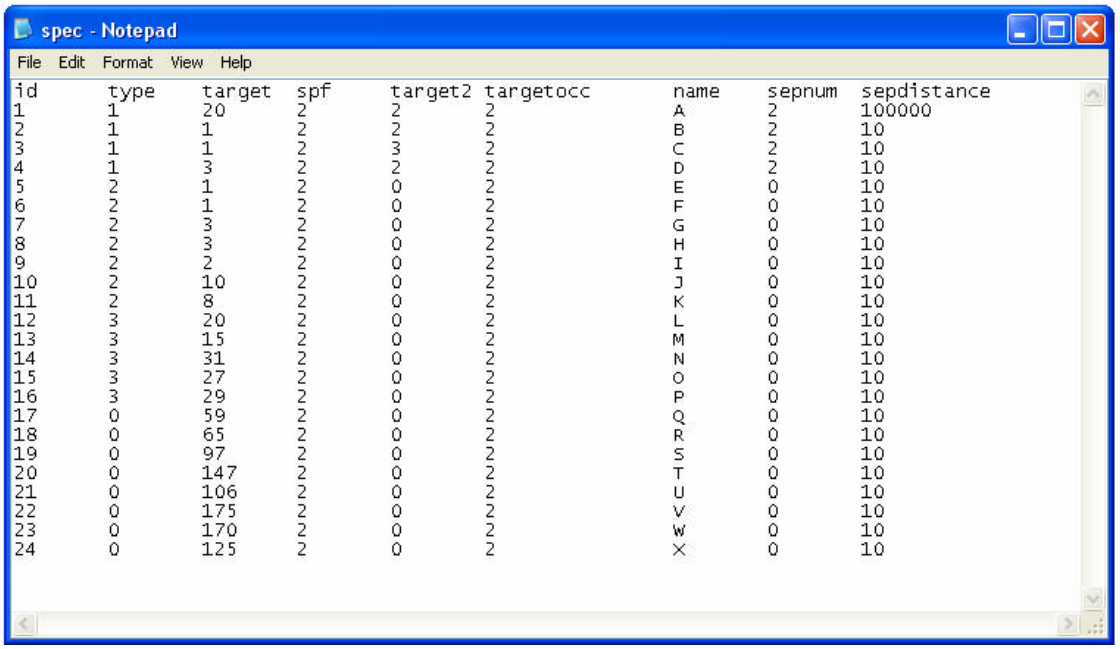
For CLUZ users there are functions that create a new planning unit v conservation feature table, import data from existing tables or shapefiles and export these data in the correct Marxan format, Otherwise:

1. Make a copy of the planning units layer and delete all fields except ID. You now want to tabulate any features you have.
2. Set view properties. The analysis extent should be set to your planning unit layer, and the analysis grid size should be set small enough to allow accurate results whilst being large enough to be tractable to run (eg. 10m or 100m depending on regions size)
3. If it is a vector layer convert to a raster.
4. Use the "tabulate areas" function to calculate the amount of each feature in each planning unit. Join the table that is produced to the copied planning units layer. Repeat this step for all features.
5. Once this has been repeated for all features. Open attribute table and export the table. Open in notepad or a spreadsheet program to check. For example make sure the heading includes "pu" and note that the name of each feature needs to be a number. Save as a text file with the ".dat" extension.

For ArcMap users

1. Make a copy of the planning units layer and delete all fields except ID. You need to tabulate any features you have. Repeat the following steps for all features.
2. Use the "tabulate areas" function to calculate the amount of each feature in each planning unit. Join the table that is produced to the copied planning units layer. Repeat this step for all features.
3. Once this has been repeated for all features. Open attribute table and export the table. Open in notepad or a spreadsheet program to check. For example, make sure the heading includes "pu" and note that the name of each feature needs to be a number. Save as a text file with the ".dat" extension.
4. Refer to the PANDA user manual for details on using the **Planning Unit File** with PANDA.

C-4 Conservation feature file



id	type	target	spf	target2	targetocc	name	sepnum	sepdistance
1	1	20	2	2	2	A	2	100000
2	1	1	2	2	2	B	2	10
3	1	1	2	3	2	C	2	10
4	1	3	2	2	2	D	2	10
5	2	1	2	0	2	E	0	10
6	2	1	2	0	2	F	0	10
7	2	3	2	0	2	G	0	10
8	2	3	2	0	2	H	0	10
9	2	2	2	0	2	I	0	10
10	2	10	2	0	2	J	0	10
11	2	8	2	0	2	K	0	10
12	3	20	2	0	2	L	0	10
13	3	15	2	0	2	M	0	10
14	3	31	2	0	2	N	0	10
15	3	27	2	0	2	O	0	10
16	3	29	2	0	2	P	0	10
17	0	59	2	0	2	Q	0	10
18	0	65	2	0	2	R	0	10
19	0	97	2	0	2	S	0	10
20	0	147	2	0	2	T	0	10
21	0	106	2	0	2	U	0	10
22	0	175	2	0	2	V	0	10
23	0	170	2	0	2	W	0	10
24	0	125	2	0	2	X	0	10

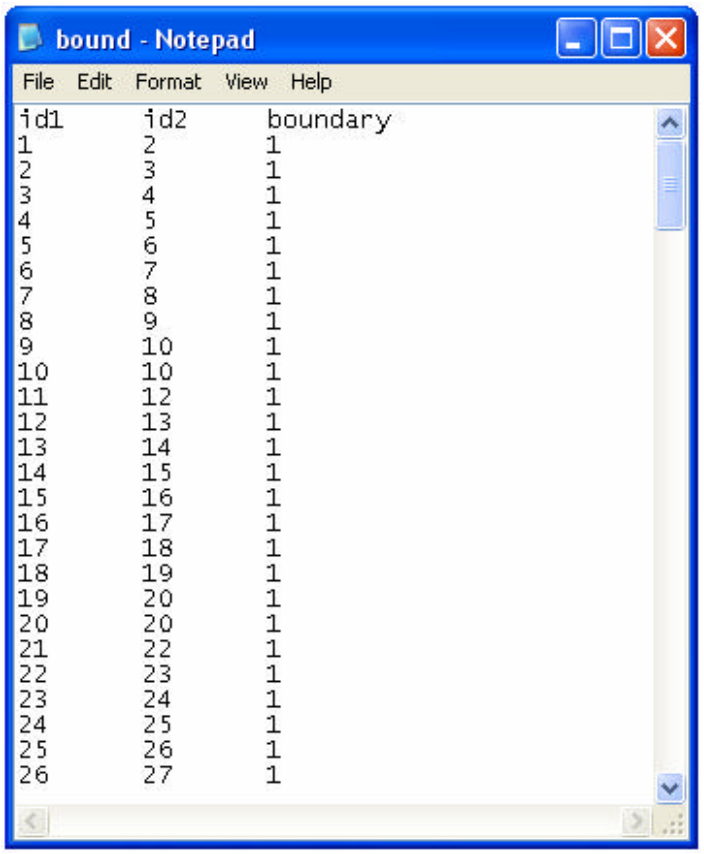
An example of the **Conservation Feature File** (spec.dat) used in Marxan.

The **Conservation Feature File** contains information about each feature being considered within the analysis. Most of the data required for developing this file will be knowledge about the natural history of each feature, threatening processes and the spatial requirements for its persistence. This will probably involve an extensive search of the literature and expert input. The file will most likely be developed in a spreadsheet program to enable formulas to be applied and saved as a text file with the ".dat" extension.

If similar targets are going to be set across particular features we recommend the use of a **Block Definition File** (See Section 3.3.2).

For CLUZ users there are several functions that create a new target table, allow the data to be inputted and export these data in the correct Marxan format. For P.A.N.D.A. users refer to the PANDA user manual.

C-5 Creating the Boundary Length File



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

The Marxan **Boundary Length File** contains the costs of the boundaries between planning units. The following two extensions produce a Marxan **Boundary Length File** in which the cost is the length of the shared boundary between adjacent planning units (the most common choice), however, we direct more advanced users to the MGPH for other techniques. Note that the planning unit layer should be topologically correct. That is, there must be no gaps or overlaps between neighboring planning units and no dangling arcs. Topology tools within ArcGIS or various scripts can be used to inspect and correct these errors. It is often possible to create a **Boundary Length File** from a planning unit layer that has topological errors. However, the boundary data will include the errors and cause Marxan to produce strange results (e.g. by making neighbouring planning units appear to not be neighbours).

For ArcView 3 users an extension is available from http://www.mosaic-conservation.org/gis/boundary.html. This function is also included in the CLUZ extension.

For ArcGIS the JNCC ArcGIS extensions are available from
http://www.ecology.uq.edu.au/index.html?page=30009&pid=29778

PANDA also provides a tool to create the **Boundary Length File** under the "Marxan Advanced" menu.

C-6 Linking output files with ArcGIS

It is very useful to display the output files from Marxan for visual interpretation of results, publishing and for decision support. There are user friendly interfaces that allow analysis and instant display of results in GIS. These includes CLUZ, P.A.N.D.A. and C-Plan. There are also plans for other new interfaces that will operate through open-source GIS software. If you are not using any of these interfaces you can manually link the output files to both ArcView 3 and ArcGIS 8 and 9 by:

1. Opening the files in a spreadsheet program, auto-fit the columns and save as a .dbf or .txt file.
2. Open in GIS and join the table with the spatial layer of your planning units.
3. It is often useful to open copies of your planning unit file so that you can view and compare different results.