Copyright

by

Michael Ian Ciarleglio

2008

The Dissertation Committee for Michael Ian Ciarleglio certifies that this is the approved version of the following dissertation:

Modular Abstract Self-Learning Tabu Search (MASTS)

Metaheuristic Search Theory and Practice

Committee:

J. Wesley Barnes, Supervisor

Clint Dawson

Chris Margules

Suzanne Pierce

William Press

Sahotra Sarkar

Modular Abstract Self-Learning Tabu Search (MASTS)

Metaheuristic Search Theory and Practice

by

Michael Ian Ciarleglio, B.E; M.E.

Dissertation

Presented to the Faculty of the Graduate School of
The University of Texas at Austin
in Partial Fulfillment
of the Requirements
for the Degree of

Doctor of Philosophy

The University of Texas at Austin
May 2008

Dedication

To my community.

Acknowledgements

The research described in this dissertation was made possible by generous funding from the National Science Foundation (NSF) and the Air Force Office of Scientific Research (AFOSR).

I am fortunate to have an amazing set of advisors from several disciplines. Special thanks to my advisor Wes Barnes, for patiently focusing my energy and thoughts into a coherent dissertation, and for supporting an unconventional approach. Also, Sahotra Sarkar for his commitment to the global community and expertise in a number of different fields. And Suzanne Pierce, her Big Ideas, and the extraordinary ability to unite people to make them happen.

I also had the great company of many talented peers. The inimitable Mrs. Novie Darmawan-Gupta, thank you for your companionship and mathematical precision. I'd also like to thank Will Cain for his professional software advice and commitment to the aforementioned Big Ideas. Trevon Fuller and Alexander Moffett and others in the Biodiversity and Biocultural Conservation Lab, thank you on many accounts for data and technical assistance. Finally thanks to Marcel Dulay and Erica Aliss for making large problems in groundwater seem manageable.

Modular Abstract Self-Learning Tabu Search (MASTS)

Metaheuristic Search Theory and Practice

Publication No. _______________

Michael Ian Ciarleglio, Ph.D.
The University of Texas at Austin, 2008

Supervisor: J. Wesley Barnes

MASTS is an extensible, feature rich, software architecture based on tabu search (TS), a metaheuristic that relies on memory structures to intelligently organize and navigate the search space. MASTS introduces a new methodology of rule based objectives (RBOs), in which the search objective is replaced with a binary comparison operator more capable of expressing a variety of preferences. In addition, MASTS supports a new meta-strategy, dynamic neighborhood selection (DNS), which "learns" about the search landscape to implement an adaptive intensification-diversification strategy. DNS can improve search performance by directing the search to promising regions and reducing the number of required evaluations. To demonstrate the flexibility and range of capabilities, MASTS is applied to two complex decision problems in conservation planning and groundwater management.

As an extension of MASTS, ConsNet addresses the spatial conservation area network design problem (SCANP) in conservation biology. Given a set of possible geographic reserve sites, the goal is to select which sites to place under conservation to preserve unique elements of biodiversity. Structurally, this problem resembles the NP-hard set cover problem, but also considers additional spatial criteria including compactness, connectivity, and replication. Modeling the conservation network as a graph, ConsNet uses novel techniques to quickly compute these spatial criteria, exceeding the capabilities of classical optimization methods and prior planning software.

In the arena of groundwater planning, MASTS demonstrates extraordinary flexibility as both an advanced search engine and a decision aid. In House Bill 1763, the Texas state legislature mandates that individual Groundwater Conservation Districts (GCDs) must work together to set specific management goals for the future condition of regional groundwater resources. This complex multi-agent multi-criteria decision problem involves finding the best way to meet these goals considering a host of decision variables such as pumping locations, groundwater extraction rates, and drought management policies. In two separate projects, MASTS has shaped planning decisions in the Barton Springs/Edwards Aquifer Conservation District and Groundwater Management Area 9 (GMA9). The software has been an invaluable decision support tool for planners, stakeholders, and scientists alike, allowing users to explore the problem from a multi-criteria perspective.

TABLE OF CONTENTS

1 Introduction

The goal of the research documented in this dissertation was to merge powerful metaheuristic search techniques with adaptive self-learning algorithms, decision analysis, and a feature-rich software architecture. This goal has been achieved, as demonstrated by the highly successful application of the MASTS software to two complex decision problems in conservation planning (Ciarleglio, Barnes, & Sarkar, 2007; C. R. Margules, Nicholls, & Pressey, 1988; C. R. Margules & Sarkar, 2007) and groundwater management (Cain et al., 2008; Eaton, Schwarz, & Sharp, 2007; Pierce, 2006). MASTS is based on tabu search (TS), a metaheuristic that relies on memory structures to intelligently organize and navigate the search space (Barnes & Chambers, 1995; Battiti & Tecchiolli, 1994; Glover & Laguna, 1997). Significantly advancing the state of the art in TS methodology, I have developed and implemented, for the *first* time, a methodology using rule based objectives (RBOs), where the one-dimensional numeric objective function is replaced with a binary comparison operator which is much more capable of expressing exact preferences. In addition, I have formalized the meta-strategy, dynamic neighborhood selection (DNS), which "learns" about the search landscape to implement an adaptive intensification-diversification strategy. DNS can improve search performance by directing the search to promising regions and reducing the number of required evaluations (Harwig, Barnes, & Moore, 2006; McKinzie & Barnes, 2006; Porter, Larsen, Barnes, & Howell, 2006).

The advanced capabilities of MASTS have provided greatly superior solutions when applied to the spatial conservation area network design problem (SCANP) in conservation biology (Ciarleglio et al., 2007). Given a set of possible geographic reserve sites, the goal is to select which sites to place under conservation to preserve unique elements of biodiversity (C. R. Margules & Sarkar, 2007). Structurally, this problem resembles the classical set cover problem (SCP) and the related maximal cover problem (MCP), both known to be NP-hard (Camm, Polasky, Solow, & Csuti, 1996; Papadimitriou & Steiglitz, 1998). Modeling the conservation area as a graph, I have developed and implemented novel and efficient techniques to account for spatial criteria such as compactness, connectivity, and replication. These spatial criteria are integral to the planning process, but have been largely ignored due to the computational and conceptual difficulties of applying classical mathematical optimization methods such as integer programming (Wolsey, 1998) and nonlinear programming (Bazaraa, Sherali, & Shetty, 2006). MASTS has been applied to a spectrum of SCANPs, some massive in size, efficiently generating spatially coherent solutions *well beyond* the capacity of both previous exact and heuristic methods. The superior capabilities demonstrated by this extension of the MASTS software, ConsNet, will place it at the forefront of national and international planning efforts.

In the arena of groundwater planning, MASTS has demonstrated extraordinary flexibility as both an advanced search engine and a decision aid (Cain et al., 2008; Pierce, 2006). In House Bill 1763, the Texas state legislature mandated that individual Groundwater Conservation Districts (GCDs) must work together to set specific management goals for the future condition of regional groundwater resources (Mace, Petrossian, Bradley, & Mullican, 2006). This complex multi-agent multi-criteria decision problem involves finding the best way to both set and then meet these goals using a groundwater availability model (GAM) while considering a host of decision variables such as pumping locations, groundwater extraction rates, and drought management policies (2007; Pierce, 2006). In two separate projects, MASTS has shaped planning decisions in the Barton Springs/Edwards Aquifer Conservation District (Pierce, 2006) and Groundwater Management Area 9 (GMA9) (Eaton et al., 2007). The software has been an invaluable decision support tool for planners, stakeholders, and scientists alike. MASTS encourages problem exploration through a dynamic graphical user interface (GUI), allowing users to guide the search with their own multi-criteria objectives, accumulating a portfolio of preferred solutions. The ability to interact with the groundwater model has brought structure to live negotiations, expediting the planning process. As planners in the Texas Hill Country move toward consensus, MASTS can rapidly identify feasible policies through its unique ability to archive and explore the results of thousands of model executions.

In addition to the applications and techniques presented above, this research contains novel contributions to each of the three CES Ph.D. concentration areas:

AREA C: Application Areas

As described above, MASTS has already made significant contributions to complex problems in conservation planning (Ciarleglio et al., 2007) and groundwater management (Cain et al., 2008; Eaton et al., 2007; Pierce, 2006). MASTS is the product of careful software design, a template that can be applied to a wide variety of complex decision problems. The extensible interface allows clients to rapidly apply MASTS to new problems, while receiving benefits such as state of the art tabu search methodologies, multi-threading, a solution cache/archive, and an interactive GUI. MASTS is on the frontier of interactive metaheuristic search techniques; users will learn about their problems through dynamic feedback, and direct the search in accordance with their subject matter knowledge and personal preferences.

AREA B: Scientific Computing

The tabu search algorithm admits a coarse grained parallelization to distribute function evaluations to multiple processors. Parallel programming techniques such as mutual exclusion, thread local variables, and atomic operations must be used to protect data structures and reduce parallel overhead (Goetz, 2006).

In ConsNet, computing the spatial properties of the conservation network requires efficient data structures and novel update algorithms. As cells are added and removed from the conservation area, an update algorithm quickly re-computes local changes to the compactness, connectivity, and replication with a partial graph traversal. The computational benefit of this update algorithm allows ConsNet to address essential spatial criteria that have previously been impossible to consider effectively.

Finally, to provide initial starting points for ConsNet, new heuristic algorithms have been developed for the SCANP. By judiciously employing fast sorting techniques, these heuristics have demonstrated significant superiority when compared to previous techniques both in speed and in solution quality.

AREA A: Applicable Mathematics

Extensive experimental results have shown that intransitive comparison operators can lead to significant improvements in search performance for the SCANP (Ciarleglio et al., 2007). Although counterintuitive, these results and logical analogies strongly suggest that intransitive search behavior is fundamental to the success of a metaheuristic search technique searching for near optimal solutions to complex problems. Modeling the search as an iterated decision sequence, we can find parallels in decision theory that justify intransitive preferences in a search environment (Fishburn, 1991; Mandler, 2005).

The next suggested step in future research would be to analyze the complicated interaction between the solution space *X*, the objective function *f*, and the neighborhoods $\mathcal{N}$ within the descriptive framework of landscape theory (Barnes, Dimova, Dokov, & Solomon, 2003). While ConsNet contains highly effective choices for *X*, *f*, and $\mathcal{N}$, additional research could lead to performance improvements.

2 MASTS System Description

This section describes how MASTS embodies the familiar elements of a tabu search (TS) within an abstract object-oriented framework. The power of MASTS arises from its flexible abstract design that captures the elementary behavior of the search elements and algorithms. While this section does not present a detailed software specification, it does describe the basic components and their roles, including how these components work together to create some of the more advanced features of MASTS.

2.1 Design Framework and Philosophy

MASTS fully embraces the principles of object-oriented programming (OOP). The result is that MASTS embodies a cleaner, more intuitive design with greater capabilities than previous software, providing a flexible, extensible programming framework with reusable components to aid in rapid development of advanced TS applications.

Proper OOP follows a fundamental set of design principles based on inheritance, modularity, polymorphism, and encapsulation and exploits established design patterns to solve common problems. The following guidelines proved invaluable in the design and implementation of MASTS (Freeman, 2004; Sierra & Bates, 2003):

- every object should have a single purpose
- prefer composition over inheritance
- avoid tight coupling
- design to an interface
- identify and apply design patterns, avoid anti-patterns

While good practice, these rules are insufficient to produce a fully functional program. MASTS was created with specific design goals, which guided the shape of the final product from the early stages. As discussed below, these goals included:

- proper abstraction

- multi-threaded functionality
- model view controller pattern
- persistence
- reusability

2.1.1 Proper Abstraction

To the highest degree possible, every element of the search is represented with an abstract object or interface. The design was guided by simplicity and proper division of labor, producing simple objects whose responsibilities and collaborations align with the conventional conceptual framework. Figure 2.1 shows the layers of abstraction that were considered in the design.

MASTS presents definitions for all of the common abstract search elements. This layer is foundational because these elements provide a “common language” to build more complex search strategies and search engines. MASTS goes well beyond other software in providing a simple but powerful model for these search elements and their interactions. This model provides superior descriptive power, both in formulating the problem and creating a metaheuristic search engine. The framework is sufficient to model TS or other metaheuristic techniques such as simulated annealing (SA) (Kirkpatrick, Gelatt, & Vecchi, 1983), genetic algorithms (GA) (Goldberg, 1989) or greedy randomized adaptive search (GRASP) (Feo & Resende, 1995).

The structure and behavior of the MASTS search engine is broadly defined to allow flexibility when creating a search engine. Since existing metaheuristic approaches are inherently different, no attempt is made to create a general metaheuristic search procedure that has “common” steps. As discussed below, previous attempts to construct such a common framework have proven to be overly restrictive. Thus, the MASTS contract for the abstract search engine does not specifically embrace lower level components (such as “generate neighbors”, “evaluate”, and “select neighbor”).

Within the MASTS search engine, however, it is fully expected that different strategies may be applied. In TS, these strategies include dynamic neighborhood selection and reactive tabu search. Such components should be properly abstracted so that different strategies can be used interchangeably.

2.1.2 Multi-Threaded Functionality

With the advent of multi-core (dual-core, quad-core, and eight-core) processors, multi-threading holds great promise to increase software performance. It also represents an extreme challenge to software programmers. Since single-threaded software cannot benefit from multi-core processors, improvements in software performance are now heavily reliant on multi-threading. Unfortunately, multi-threading is considerably more difficult than sequential programming. Dan Appleman (2003) has the following comments:

> “It's a useful technology—one that has the potential to improve your application's real (or perceived) performance. But it is the software equivalent of a nuclear device because if it is used incorrectly, it can blow up in your face. No—worse than that—used incorrectly, it can destroy your reputation and your business because it has nearly infinite potential to increase your testing and debugging costs.”

One reason that multi-threading is so difficult is because it is a relatively new paradigm within professional code development. Many languages have just recently added support for multi-threading. While Java has built in support for multi-threading, there were serious bugs with the memory model and compiler as recently as 2004 (Pugh, 2004). These issues were fixed with the release of Java 1.5 (Tiger) in accordance with Java Specification Request (JSR) 133.

Multi-threading is not a simple endeavor. Some programmers are not accustomed to dealing with the exigencies of concurrent threads, where events may not happen in a specific order, and interruptions can happen at any time. Even worse, it is often impossible to retrofit existing code with multi-threaded capabilities. Thus, the best advice is to design the applications for multi-threading from the ground up. Dan Appleman (2003) continues:

> "Again, I stress, design your applications correctly. If your design is incorrect, it will be virtually impossible to patch up the problems later. And again, the potential cost to fix threading problems has no upper limit."

Multi-threading weighs heavily in the design of MASTS. Most importantly, it forces us to embrace simple, immutable objects with limited responsibilities (Goetz, 2006). The dangers of multi-threading provide an extra incentive to make sure that the design is as simple as possible.

In addition, multi-threading adds extra complications when an interface is exposed. Ideally, clients can write adaptors for MASTS without having to worry about multiple threads. Pushing multi-threading onto the client side creates more problems than it solves. MASTS is particularly useful because the multi-threading capability is transparent to MASTS clients, i.e., MASTS clients receive the multi-threading capability for "free."

2.1.3 Model View Controller Pattern (MVC)

Figure 2.1 presents the components (objects) of the MASTS "model". Each object has specific responsibilities and collaborates with other objects. The model-view-controller (MVC) pattern recommends additional control objects to interact with the model. These controllers issue commands and request data from the model. As pictured in Figure 2.2, each controller is designed to operate independently, handling requests and performing tasks in a multi-threaded environment.

The graphical user interface (GUI) completes the MVC pattern by providing the "view". It allows users to interact with the search, monitor progress, and quickly process the results. The GUI provided with MASTS is 100% reusable for every problem. On some panels, clients may optionally provide GUI components specific to their problem. The GUI influenced the design process because it requires event driven programming. MASTS has its own event notification system, which notifies the GUI (and other *listeners*) when key events occur.

2.1.4 Persistence

Persistence refers to the survival of data and/or to the program state after execution has ended. In MASTS, persistence allows users to save their progress, including elite solutions and search history. While providing persistence is useful, it demands careful and challenging programming (a task that most programmers would rather avoid). A simple and fast version of persistence can be achieved with Java serialization. In serialization, a "flattened" version of the object is written to the hard drive as serial data. When requested, the object can be reanimated (provided that the class definition has not changed).

MASTS uses persistence to save the following items (which essentially comprise the whole program): (1) problem definition, (2) solutions, (3) objectives, (4) search status and (5) user preferences.

The extensive use of persistence governed some of the design choices in MASTS. In particular, it forced a clear division of objects that contain data from objects that act on data. Designing for persistence requires the adoption of simple, clean designs.

2.1.5 Reusability

While the concept of reusable metaheuristic software is not new, previous attempts (discussed in Section 2.2) lack the structure, flexibility, and features to take full advantage of "reusability". First, a design that fails to reflect an intuitive understanding of metaheuristics will discourage most potential clients from using it. Other existing software lacks the proper abstract design, making it difficult to include special behavior or new strategies. Finally, some current software simply lacks useful features, i.e., there is no perceptible advantage for its reuse. For research purposes, simple features like saving a set of best solutions, working with multiple objectives, and logging the search activity are invaluable. Other important features include ease of use, quick access to the results, and the ability to use multiple processors. Software that does not contain all or most of these features is not likely to be reused. The key to reusable software is that *people have to want to reuse it.*

This begs the question, “What MASTS design will maximize its reusability?” This question involves design psychology as much as computer science. We know that code reuse eliminates bugs and simplifies software maintenance, but these concepts rarely appeal to novice or intermediate level programmers. Often the most important question is, “If I use MASTS, what do I get for free?” When the answer includes (1) a free GUI, (2) free solution archiving, (2) free persistence, and (4) free multi-threading, we have an initially appealing software package even before we discuss MASTS’ advanced search capabilities. MASTS’ approach is radically different from earlier software packages, where the design efforts were exclusively focused on the search algorithms and ignored overall reusability.

While it is important that search algorithms be reusable, how much does this really help software developers?

Harder (2001) states ”You can define your problem and your ideal tabu search and leave the grunt work to OpenTS.” Unfortunately, OpenTS provides only a lightweight TS engine. Harder (2001) incorrectly perceives the “grunt work” to be the construction of the search itself. On the contrary, the majority of the grunt work (for almost any program) actually lies in the data management, the GUI development, interface design, and workflow control. This statement is validated by the fact that the core of the MASTS TS module is about 400 lines of code, out of an estimated 50,000 for the entire MASTS framework. This indicates that the search engine is only a small part of what may be reused. Given a choice, most metaheuristic experts (the most likely users of MASTS) would reuse the 49,600 lines of MASTS and write their own search engine rather than taking the 400 lines of tabu search code and supplying the entire supporting framework. Few developers have the time (or inclination) to write and test GUIs, caches, and multi-threaded controllers.

MASTS is reusable precisely because it provides convenience and functionality that would otherwise require excessive resources and effort to develop. Clients of MASTS will *want* to reuse it not only for the search capability, but for all of the added functionality.

By providing a graphical user interface, sorted tables, simple visualization, data imports and exports, and other organizational tools, MASTS greatly enhances the understanding of the problem itself and the interaction between the search and the solution space. For the conservation network design problem (discussed in Chapter 6), the best strategies to date have resulted from "observing" the search evolve solutions. MASTS provides a wealth of feedback that can be used to suggest new strategies to improve performance.

2.2 Previous Software

There are a few general purpose optimization software packages that focus on metaheuristic methods. All of them provide an interface that allows them to adapt to different problems. Software packages in this category include: Coin-OR OTS, HotFrame, EasyLocal++, and OptTek. This section presents a critical review of these metaheuristic software packages.

2.2.1 OpenTS (OTS)

OpenTS (Harder, 2001), part of the COIN-OR (Computational Infrastructure for Operations Research), is very limited in the problems it can address and does not provide built-in support for advanced tabu search methods.

The OpenTS interface is simplistic. The abstractions of the major search components are either absent or not sufficiently generic. For example, OpenTS provides no way to take advantage of hashing solutions and studying the search trajectory. The OpenTS interface fails to define a library of common reusable elements, forcing all clients to develop and validate their own implementations of major objects such as decision variables, neighborhoods, moves, and tabu memory structures.

Finally, OpenTS claims to support dynamic tabu search, but provides no default implementation or framework, leaving the required additional programming to the client.

2.2.2 HotFrame

HotFrame (Fink & Voß, 2002) is a software library with reusable components that incorporates several different search algorithms, including steepest descent, simulated annealing, and tabu search. The object oriented design focuses on the commonalities in metaheuristic methods and demonstrates how common search components like neighborhoods can be reused. Unfortunately, several design aspects of the software limit its potential for widespread application or adoption.

First, the software and the documentation are clearly not developed for readability. The naming conventions make the documentation absolutely unintelligible. Naming a module `StopCriteriaInterface` is clearly superior to the non-descriptive `OmegaInterface` used by HotFrame. Variable names like `X_P`, `X_S`, `X_S_N`, `X_S_I`, and `X_S_A` only exacerbate the difficulty of understanding this software library.

The HotFrame creators appear to be aware of the benefits of object oriented design. They establish workflow and collaborations, identify objects that can be reused, and make effective use of the strategy pattern. In particular, they have realized that it is useful to define abstract search elements such as a neighborhood. Unfortunately, they largely fail to create useful representations for other abstract search elements. For example, the decision variables and all of the methods to evaluate and modify solutions are contained in a *single* structure called the solution space interface S.

> "The solution space interface defines the basic functionality that must be implemented by problem-specific classes: construction of a solution (given a problem instance), objective function computation, computation of the evaluation of a given move, modification of the solution according to a given move." (pg 28)

Allocating this much functionality into one class is a fundamental violation of OOP.

MASTS purposefully separates these tasks and interrelationships. Less tightly coupled search components allow much greater flexibility in the use of decision variables, neighborhoods, and objective functions while greatly reducing the risk of unintended adverse interaction effects .

The designers of HotFrame address several search techniques and create *some* re-usable components, but fail to provide a feature rich search environment. HotFrame is ultimately limited by the failure to define key search elements, enforce a proper division of labor, and use composition (rather than inheritance) to create flexible designs.

2.2.3 EasyLocal++

EasyLocal++ (Di Gaspero & Schaerf, 2003) claims to provide a "neat conceptual scheme" for creating a variety of hybrid local search techniques. EasyLocal++ is marred by a very convoluted construction and a description that it is difficult to follow. Gaspero and Schaerf (2003) perhaps "aimed too high" by trying to create a generic framework that applies to all local search techniques. In this quest, they fail to create reasonable proxies for fundamental search elements such as decision variables, neighborhoods, and objectives. Without these, the benefits conferred by attacking a problem using EasyLocal++ are not worth the investment. At best, you will obtain a mediocre implementation of simulated annealing, a primitive version of tabu search, and a log file to help you figure out why it doesn't work.

2.2.4 OptTek's OptQuest Engine

The OptQuest Engine (OptTek Systems, Inc.) provides some search options well beyond the scope of the MASTS software, which focuses exclusively on tabu search. The OptQuest interface is presented online at:

http://www.opttek.com/documentation/v62engine/OptQuest%20Engine%20API/index.html

In OptQuest, we see a strong software implementation of decision variables, objectives, and constraints similar to MASTS. However, from a software design perspective, OptQuest wraps too much functionality into the actual optimization engine. MASTS avoids this by having separate controllers that handle solution generation, objective evaluation and storage, and memory cleanup. In addition, MASTS defines crucial objects such as neighborhoods and moves, which allows a much cleaner interface for dynamic neighborhood selection and reactive tabu strategies. Finally, while MASTS separately abstracts the key pieces of the search algorithm, allowing different implementations to be used interchangeably, OptQuest relies on a huge set of "search parameters" to attempt to achieve the same goals.

There are some other key performance contexts missing from the OptQuest interface that further differentiate it from MASTS:

- Multi-threading – the “services” that work together in MASTS are multi-threaded. In the presence of multiple processors, costly tasks such as solution construction and function evaluations are performed in parallel with high efficiency.
- GUI design – each service in MASTS may be easily adapted for GUI interaction and control
- Event management – an event manager notifies listeners of key events. Clients writing extensions for the code can listen for these events.
- Solution archiving – MASTS assigns a hash code (a unique serial number) to each solution and stores it in a memory managed cache. This cache is backed by a customizable archive that allows long term storage of solutions. In a simulation-optimization setting, where function evaluations take seconds or minutes, this archive may save considerable time. It also permits the search to produce quick answers, perhaps during live negotiations when there is no time to run additional scenarios.
- Rule based objectives – this original innovation is unique to MASTS. In essence, it allows users to provide objectives in the form of a comparison operator which compares alternatives according to a set of rules, rather than a numeric score.

2.3 Decision Variables

Decision variables are at the heart of any optimization problem, and make an excellent candidate for an abstract data type. In general, the DV (decision variable) class contains the value (which may be an object of any type) of the decision variable as well as all of the methods that allow us to properly change the value of the decision variable. These methods are protected to prevent the value from inadvertently changing. Two common decision variables, discrete and boolean (on/off), are presented in Figure 2.3. They are represented in MASTS with the DiscreteDv and BooleanDv classes.

However, in an abstract environment, *anything* that fulfills the contract of the DV class can be considered a decision variable. This makes the DV class more of a meta-variable, with capabilities beyond the usual definition of a decision variable. For instance, in the aquifer management problem, one goal is to determine the best way to allocate extra pumping to different pumping centers. This problem can be phrased as a "percentage partition" variable, demonstrated in Figure 2.4. The problem now becomes assigning discrete units of extra pumping into bins represented by the pumping stations.

There are many justifications for representing decision variables in this manner, as opposed to a more conventional approach involving several discrete or continuous variables. First, this approach naturally captures the constraint that reductions to one pumping center must be balanced by increases at another. This constraint is easily satisfied if the move set consists of "picking up" a block of allocated pumping and moving it into another bin. Moreover, each pumping center may have an upper limit on additional pumping. This constraint is efficiently handled if the move set simply excludes moves that would overload a pumping center. There is no need for added complexity when incorporating the two major constraints on the problem.

One last major advantage to this approach is that we can easily adjust the size of the search space, by changing the size of the blocks in the partition. Figure 2.4 breaks the additional pumping into 100 increments of 1%, but we could easily consider 20 increments of 5%. In many problems, the model is not accurate enough to justify increments of 1%; time may be saved by using 5% chunks instead.

There is no limit to how "rich" you can make a decision variable as long as you adhere to the contract described in Table 2.1. The decision variable may be any type of object and can assume any type of value. Such a flexible definition allows unlimited creativity when formulating a problem. While other software packages, like OptQuest, have abstract decision variables, their contract is significantly more restrictive. As discussed in Section 2.4, the key to the extra flexibility is the abstraction of the concepts of neighborhoods and moves.

2.4 Neighborhoods and Moves

A decision variable by itself is inoperative until we describe how to change the value with moves and organize these moves into neighborhoods. Both moves and neighborhoods are represented as abstract objects, which may be customized for a specific problem or application. The software treats decision variables, neighborhoods, and moves as a tightly coupled triplet; these structures work closely together to create the neighborhood topology within the search landscape (Barnes et al., 2003). The generality of this approach enables dynamic neighborhood selection, making it easier to study how different neighborhood and move configurations impact the search progress. It also provides a general context for implementing any type of metaheuristic search.

Because the DV class represents an abstract value, one must also define moves and neighborhoods that collaborate with the DV class. The Move class represents an abstract move on a decision variable, and it contains methods that safely change the value of that decision variable. The Nbhd class does not physically contain neighboring solutions. It contains the rules for generating moves around an incumbent solution that will take us to neighboring solutions. The rules may be static or may incorporate specific knowledge about the problem. When a Nbhd uses problem specific knowledge to eliminate some possible moves, this is equivalent to a candidate list procedure (Rangaswamy, Jain, & Glover, 1998).

These data structures are tightly coupled because they are dependent on each other and require privileged access to some protected fields. For example, with a discrete decision variable, a proper corresponding Move and Nbhd are required. Care has been taken to design these structures so that they work together without exposing sensitive fields and methods to external elements of the search. Programmers should follow the general rule that moves are the only "legal" way to change the value of a decision variable.

Figure 2.5 shows the coupling between decision variables, moves, and neighborhoods. This diagram can be extended to arbitrarily complex situations. For instance, it may be necessary to incorporate multiple move types and neighborhoods. Some of these potential situations are shown in Figure 2.6. MASTS permits all of these configurations, allowing users to attach any number of neighborhoods to a decision variable. Setting up the neighborhood and move structure in code is quite straightforward.

For clarification, consider the simple example of interactions between decision variables, moves, and neighborhoods presented in Figure 2.7. More complicated structures have been created for two specific problems, the groundwater management problem (Pierce, 2006) and the species conservation problem (Ciarleglio et al., 2007).

Any complicated problem is likely to have multiple neighborhoods for each decision variable. Multiple neighborhoods introduce a new type of decision: Which neighborhood should be used at each iteration?

Chapter 4 discusses dynamic neighborhood selection and a full example is provided with ConsNet (see Section 6.5).

2.5 Tabu Memory Structures

At each iteration, the TS explores a number of moves (transformations of the current solution) to neighboring solutions. A tabu memory structure embodies a logical set of rules that declares a subset of these neighbors to be forbidden (tabu). After accepting a move to a new incumbent solution, the tabu memory prevents the search from returning to the previous solution (or other specified features) for a stipulated number of iterations, the *tabu tenure*. This keeps the search moving forward and prevents short term cycling behavior.

A tabu memory structure is based on tabu attributes. Typically, these attributes are based on some property of the move (a *move based attribute*) or the neighboring solution (a *solution based attribute*). Once the search performs a move away from an incumbent solution with a specific attribute, that attribute remains *active* for the number of iterations stipulated by the current tabu tenure. Since a static tabu tenure often results in cycling, adaptive or reactive strategies improve the search performance by strategically changing the tabu tenure (Battiti & Tecchiolli, 1994; Dell'Amico & Trubian, 1993; Harwig et al., 2006).

Tabu memory structures are specific to tabu search, but they fit quite naturally into the "decision variable-neighborhood-move" triplet defined in Figure 2.8. Within MASTS, the TabuList is the most basic type of tabu memory structure. A TabuList is attached to a specific decision variable, and only activates when the search performs a move that the TabuList is equipped to manage. As shown in Figure 2.9, a TabuList may track one or more move types. The TabuList declares which types of moves it is listening for, and receives both the move and the associated neighbor solution to determine whether that solution is tabu. As detailed in Table 2.2, each TabuList keeps track of its tabu tenure and provides public accessor methods which allow modification of the tabu tenure.

For a complex problem, there can be several decision variables, with one or more TabuLists attached to multiple move types for each decision variable. A higher level class called the TabuStructure organizes the individual lists. For a specific move and solution, the TabuStructure identifies the applicable TabuLists. If *any* of these lists determine that a particular solution is tabu, then that solution is considered tabu by the search. When the search selects the new incumbent from the neighboring solutions, all of the associated TabuLists are notified to declare the appropriate attributes as tabu.

The TabuReactor, which organizes the lists and implements dynamic tabu tenure strategies, has three primary responsibilities:

- serves as a single point of contact to work directly with the search engine
- coordinates the actions of the different TabuLists (via an internal TabuStructure)
- implements the dynamic tabu strategy by changing the tabu tenure on some or all of the lists

The methods for the TabuReactor are presented in Table 2.3. The react() method is called after the search has determined the new incumbent, but before setTabu(). Thus, the tabu tenure adjustments occur *before* the tabu lists mark a certain attribute as tabu. The IterationSummary sent to the react() method contains all of the results from the iteration, granting the TabuReactor access to a wide variety of information such as:

- trajectory information, the number of recognized solutions
- the number of improving/disimproving moves
- the number of consecutive improving/non-improving moves and
- the number of tabu moves

2.6 Solution Representation and Hashing

In the previous sections, we have explicitly discussed only individual decision variables. Most problems have many associated decision variables. One strength of the MASTS framework is that clients may use any number of decision variables *of any type*. This capability allows decision problems of arbitrary complexity to be described within the MASTS framework.

In the presence of multiple decision variables, the solution is represented as an array that is wrapped in a DvConfiguration class (see Figure 2.10). A fresh (either new or recycled) instance of DvConfiguration is created for each solution that is visited. The DvConfiguration contains the state of all decision variables, but none of the computed results. The order of the decision variables is the same in every DvConfiguration.

A hash code is used by the tabu search to track individual solutions and detect cycling. Each DV possesses a method that converts the current value into a series of bytes, which are fed into the byte buffer for the MessageDigest class. MASTS employs the MD5 algorithm embedded in Java to create 128 bit hash codes. This hash code is immediately converted to a hexadecimal string to facilitate the database applications used in MASTS.

When solution archiving is required, DvConfigurations are written to the hard drive in a serialized state. Upon restoring the DvConfiguration, it is important to verify that the problem definition has not changed in a way that invalidates the DvConfiguration. Thus, additional data is included in the hash code (this is sometimes called salting the hash). The hash explicitly considers the following items:

- the number of decision variables,
- the types of decision variables
- the names of the decision variables,
- the ordering of the decision variables,
- the unique problemID provided by the user

The hash is re-computed when restoring a DvConfiguration from the archive and checked against the original value. If any of the items listed above have changed, then an exception is thrown. The goal is to avoid an “accident” where a DvConfiguration from a different problem (or an earlier version of the same problem) is mistakenly loaded.

Table 2.4 indicates that the hash codes generated by the MD5 algorithm will almost certainly be unique. If 820 trillion configurations are explored, the chance of collision is only slightly greater than one in a billion. If MD5 proves unsuitable for this application, it is easy to move to a more secure hash algorithm.

Some problems have non-unique elements that can create identical solutions even though the configurations may be different. For instance, in a multi-vehicle pickup and delivery problem, one approach is to assume that all the vehicles are identical. In this case, a solution yields the same outcome regardless of which vehicle takes each route. Even though these solutions look different, they are essentially *identical* because the vehicles belong to the same equivalence class. The hashing algorithms employed by MASTS can recognize duplicate arrangements in problems with non-unique permutative elements. Since the problems to be addressed in the dissertation do not require this ability, we will not discuss it further.

Because computing the hash is a time consuming process, it is done only once for each configuration. After the hash has been computed, it is permanently attached to the DvConfiguration and further changes to the DvConfiguration are forbidden.

2.7 Solution Construction and Archiving

One of the major MASTS contributions is the highly configurable solution archive at the core of the program. The archive is an integral part of MASTS, adding value to the program in the following ways:

- allows the user to save progress by storing solutions
- allows users to collect portfolios of preferred solutions, and analyze them from the perspective of different objectives
- the search can save time by looking up archived results
- large archives can be assembled and stored to save time on model evaluations, especially during live negotiations
- users gain insights about the nature of the problem and the search dynamics by examining archived solutions

Despite its obvious influence in the program, the archive must remain transparent to the user, performing its duties unobtrusively in the background. Moreover, the archive behavior must be suitable for a wide variety of different problem types. In some cases, this model may be expensive to evaluate (*extended evaluation)*, such as with a groundwater management model. In other cases, the model may be very rapidly evaluated (*quick evaluation)*.

The approach to solution archive management depends on the effort required for solution evaluation. For extended evaluation problems, *significant* computation time can be saved by storing as many solutions as possible either in memory or on a hard disk. For quick evaluation problems, it may be useful to store solutions in memory, but using the hard disk as auxiliary storage could be more expensive than simply re-evaluating the solution. Both types of problems can be addressed using a configurable cache backed by a hard disk archive. A ResourceService monitors both the cache and the available memory, performing adjustments when memory becomes scarce.

This section discusses the individual pieces of the solution archive, and provides an overview of how they work together.

2.7.1 The Soln and Results Classes

The function evaluation in MASTS is assumed to be an abstract "black box" model. The inputs to the model are contained in the DvConfiguration, and the outputs are contained in an abstract data type called Results. A Soln is simply a class that binds a DvConfiguration with a set of Results (see Figure 2.11).

The ResultsBuilder is the "black box" model in the process. Its primary function is to evaluate and create new Results for a specific DvConfiguration. As presented in Table 2.5, the buildResults() method also receives the previous move and the previous solution. This allows the ResultsBuilder to use "shortcuts" when appropriate. For some problems it is faster to update a solution based on the previous solution rather than re-computing everything from scratch.

Because the buildResults() method may be called simultaneously from different threads, it must not store temporary or mutable information in shared class variables. All references that need to be shielded from thread interference should be held in local or ThreadLocal variables.

The Results should contain all of the data that analysts would want to examine for a particular model. While some models generate great amounts of data, it is usually possible to condense the outputs into representative measures that adequately capture the performance.

Ideally, the Results contain only raw data, free from interpretation and subjective opinion. Judgments regarding the quality of the results are determined separately by an objective function class. In the presence of multiple objectives, a particular set of Results may rank highly with one objective and poorly with another. Separating the raw output data from the subjective interpretation is crucial to the proper maintenance of both the software and the solution archive. The reason is that subjective interpretations can be subject to *frequent* change. During the life of a project, analysts will often decide that they want to interpret the output data in a slightly different way, using different weights and different methods. By comparison, the Results from the model are relatively static and final; once the Results class has been properly defined (containing all the data that people would want when making decisions), the definition is not likely to change. This makes the Results ideal for database archiving, perhaps through serialization. In addition, the constancy of the Results class makes the software easier to maintain. When creating new ways to interpret the data, there is no need to modify the Results class, which could cause harmful interference with others that have to interact with that class.

2.7.2 The SolnService and Archiving

The SolnService is a high level program controller that houses the archive, providing methods to build and register solutions, query the archive, and moderate the archive size. The archive consists of two layers: a memory cache backed by hard disk storage. Since access requests may simultaneously arrive from several threads, proper synchronization is mandatory in this cache/archive (see Figure 2.12).

The Archiver is the portion of the archive that writes solutions to the hard drive, and then restores them to memory upon request. The details of this transaction are left largely to the implementing class. MASTS supplies a default Archiver through serialization. However, if the class definition changes while the object is serialized, then the stored object can be restored to memory only with additional special coding effort. Thus, it is important that the Results class has reached a stable structure before large archives are developed.

Since MASTS clients can supply their own Archiver, one possibility is to store the results in a MySQL database, on a per-problem basis, allowing greater accessibility to the results. The CacheArchiver is the layer of the archive that controls the solutions resident in memory. The CacheArchiver wraps around the Archiver, handling the interface between the clients, the cache, and the underlying archive. The cache's upper limit can be dynamically set by clients. If the cache limit is exceeded, overflow is written to the underlying archive. This is one potential mechanism for handling memory shortages.

2.7.3 ResourceService

The ResourceService contains automated tools to manage the size and behavior of the cache to prevent memory shortages. For quick evaluation problems in particular, the ResourceService can be configured to automatically purge the archive. During a purge, the ResourceService keeps only the top performing solutions, or those that have been explicitly saved by the user. The goal is to prevent cache overflow and the associated hard drive transactions.

When enabled, the automatic purges may be triggered if memory becomes tight or when the archive grows past a certain fixed size. The purge should occur before the archive reaches the maximum cache size, at which point overflow solutions will be written to the archive (slowing down the search). Users may specify how many of the top solutions for each objective should be retained during a purge event. At any time, users may also mark specific solutions that should be saved.

Extended evaluation problems generate fewer solutions, and it may be more effective to hold these solutions in the archive instead of purging. In this case, automatic purges should be disabled. As the archive grows, it is possible to save time by looking up previously evaluated results rather than performing a lengthy computation.

2.7.4 Search Modes and Solution Construction

For quick evaluation problems, it may be too costly to cycle every solution through the archive. Each solution that goes into the archive requires extra processor time in the following areas:

- Creating a **new** DvConfiguration and Results object (re-using objects that have been published in the archive is not advisable)
- Computing the hash code for the DvConfiguration
- Obtaining the proper locks on the synchronized archive
- The possibility of writing the solution to the hard drive (if the cache is not large enough)
- The ResourceService will have to purge more frequently (invoking the garbage collector)

In this section, we discuss different "search modes" that can bypass any of these steps (depending on the policy for a specific problem). The *search mode* appears as an option on the GUI, and is implemented in code as a CandidateFactory. The search engine will use the CandidateFactory to generate and explore the candidate solutions in a neighborhood. The candidate factory can use a variety of multi-threading and archiving policies. Below, we discuss the advantages and disadvantages of two such search modes: full archiving and minimal archiving. These two approaches are sufficient for both the groundwater and conservation problems, but other CandidateFactories can be created if different behaviors are required.

Search Mode: Full Archiving

This search mode commits every solution to the archive. This approach is ideal for an extended evaluation problem, where the overhead of archiving is a small fraction of the computational effort (and the archive can be used to save future evaluations). Full archiving allows users to see all of the solutions generated by the search which may lead to better understanding of the search behavior. Finally, because a hash code is computed for every solution, full archiving can identify repeated solutions and detect basins of attraction. This ability is available in other modes, but to a lesser degree. Unfortunately full archiving requires computational overhead for every solution (described above) which can slow the search down, particularly for quick evaluation problems.

Search Mode: Minimal Archiving

At the very least, The search must archive the single incumbent solution from each iteration. Other solutions can be explored without passing through the archive. This raises the possibility of skipping the archiving steps described above. For unarchived solutions, no attempt is made to generate the hash code or contact the archive, saving a *substantial* amount of overhead in processor time and thread contention. Also, instead of creating new DvConfiguration and Results objects, a thread local version can be re-used without having to reallocate memory, saving a significant time if these objects have a large memory footprint (see Section 2.9.1). However, because these variables are constantly overwritten, the Results for the new incumbent solution are lost by the end of the iteration, requiring one extra evaluation to recover the Results for the new incumbent solution.

With the savings in memory and thread contention, minimal archiving is the most efficient method for quick evaluation problems in a multi-processor environment. Obviously, the savings in overhead must outweigh the cost of the extra evaluation (which is performed in a serial part of the code). In addition, because the search does not query the archive for previous results, this method can not benefit by looking up previous solutions (which is not a significant loss if the search rarely visits the same solution).

In this mode, the search hashes and tracks only the incumbent solutions. Even though slightly less information is available to detect cycles and basins of attraction, it is still possible to detect solutions that have appeared on the trajectory more than once. In many cases, as shown by Battiti and Techiolli (Battiti & Tecchiolli, 1994), this is satisfactory.

2.7.5 Advantages of Solution Management

Centralized solution management is a *very* important feature of MASTS. In addition to computational savings on extended evaluation problems, the archive is instrumental in tracking multiple objectives. When new objectives are created, the archive can supply good initial solutions. Behind the scenes, a centralized archive simplifies the software maintenance and memory resource management. Finally, the archive is a key part of the user experience, allowing users to save results and monitor the search and exploration process.

Storing solutions in a centralized archive allows rapid search initialization with new objectives. By assessing previously evaluated solutions the solution archive, the search with the new objective acquires a “head start”. The search can quickly begin at the best known solution for that specific objective. This has already proven quite valuable in both the groundwater management problem and the conservation network design problem. The rapid initialization feature could also be valuable in a real-time setting where users cannot wait for additional function evaluations to be performed, such as during negotiations or dispute resolution.

In addition, software maintenance is easier because the computation and storage of the results is cleanly separated from the objective functions. This arrangement allows a “lightweight” objective structure with less responsibility. Although each objective score may store *some* properties about the solution, it does not have to store the *entire* solution (just the serial number that can be used to locate the Soln in the archive). This schema requires less memory when multiple objectives over a common set of solutions are being considered. Furthermore, if the solutions were stored with each objective rather than in a central repository, it would be more difficult to lookup previously evaluated solutions. Finally, one goal of the MASTS software design is to make it simple for clients to write their own objectives. If the objective class had to manage its own archive, then there is an added risk that an implementing class could accidentally break this system. More details about objective function management are presented in Section 3.6.

Finally, the multi-threaded solution archive has proven most valuable because it allows clients to view, save, load, examine, and interact with solutions in real time. Several insights leading to massive improvements in the conservation network design problem have arisen directly from MASTS capability to efficiently display the solutions currently loaded into memory.

Once again, clients of MASTS receive advanced solution management (including archiving) without writing *any* extra code. This allows programmers to rapidly proto-type new problems without having to worry about the semantics of multi-threading, caching, or a persistence enabled archive.

2.8 Multiprocessor Management

With the introduction of low cost dual core processors, many desktops (and laptops) will soon have multiple processors. As discussed in Section 2.1.2, software must be specially written to take advantage of this extra computing power. MASTS is written to exploit multiple processors in the following ways:

- to parallelize function evaluations during the search
- to perform batch type tasks (such as calculating many objective scores at once)
- to maintain GUI responsiveness

Simultaneous tasks are carried out using multiple threads in Java. The thread scheduler in the Java Virtual Machine (JVM) transparently handles the issues of thread and memory management across multiple processors. If more than one processor is available, the JVM distributes threads across these processors allowing for simultaneous execution. If only one processor is available, the threads share time on the processor as the thread scheduler sees fit. The convenient result is that a multi-threaded code can run on a machine with one or many processors.

This feature of Java is limited to a machine that has multiple CPUs with shared memory (such as a dual core, quad core, or eight core processor). The JVM relies on the hardware to handle data staleness and coherence protocols. In a multi-processor environment with distributed memory, messages must be explicitly passed between processors to keep track of data, and the JVM does not (and should not) handle this automatically.

The primary difficulties in multi-threaded code involve:

- protecting data from corruption caused by interleaved operations in critical sections of code
- sharing data between threads reliably
- avoiding deadlock of threads

- reducing the hardware time spent on lock contention

Multi-threading is usually required for GUI development as well. In Java, the Swing dispatch thread runs the GUI. If you overload this thread with computations, then the GUI becomes unresponsive. The solution is usually to spawn cumbersome tasks onto worker threads.

TS algorithms inherently allows course grained parallelization. For each iteration, the search generates a set of *n* neighbors. Unless the solution is already in the archive, each neighbor requires a function evaluation. These evaluations may be distributed across several processors. The search cannot continue until all of these evaluations are complete. In MASTS, these tasks are allocated to a thread pool with the number of worker threads equal to the number of CPUs.

2.9 Performance Enhancement and Profiling

One critical aspect of software maintenance and improvement is benchmarking and profiling the code. ConsNet has a built in “benchmark mode” which can run automated tests for different data sets and report the required computation time. The benchmark has been useful in verifying how recent changes in the software have greatly improved the run time, and will continue to play an important role in assessing minor changes to the software and program performance.

Another useful tool has been the profiler built into the Java virtual machine. It records how long the program spends in each method. From this information, it is possible to understand where the program is spending most of the execution time. Unfortunately, the profile does not fully capture the processor time lost due to thread contention or memory allocation and garbage collection. A more sophisticated tool for this type of analysis is the Java Heap Profiler (HPROF) introduced as part of the HotSpot Virtual Machine in J2SE 5.0 (O'Hair, 2004).

Using both benchmarks and profiling, it was possible to identify the promising code subsets that could benefit from code optimization. Restructuring the code and removing some of the unnecessary archiving operations boosted program speed by a factor of 50 compared to the earliest benchmarks. The critical sections that have been updated include (1) memory allocation and de-allocation, (2) hashing, and (3) concurrent thread management.

2.9.1 Memory Allocation and De-allocation

Despite Java's poor reputation in the late 1990's, the modern HotSpot Java Virtual Machine (JVM) is very well-tuned for memory allocation and de-allocation, competitive with similar approaches in C and C++ (Goetz, 2005). The details of the memory heap and garbage collection strategies in the HotSpot JVM are briefly described in a white paper from the Sun Microsystems (Memory Management in the HotSpot Virtual Machine, 2006).

Regardless of the semantics of the garbage collector, one inescapable truth remains: the less garbage you create, the less time you spend cleaning it up. If a program repeatedly creates and disposes of objects with large memory footprints, then it can be much more efficient to re-use them.

But recycling objects in a multi-threaded Java application is not a light endeavor. The danger lies in two areas:

- Since the lifecycle of the object must be properly managed, additional methods to set and clear the fields introduce new opportunities for error
- State changes made by one thread may not be visible to another thread without proper synchronization, which raises the possibility that one thread will observe a *stale* value in a recycled object (Goetz, 2006)

The second bullet should be taken quite seriously, since errors of this type can be incredibly difficult to detect and debug. Moreover, this approach requires an explicit (and perhaps costly) synchronization between threads. A better approach is to create *thread local* instances of the objects that can be re-used. A thread local variable will create one instance of an object for each thread that accesses the variable. This instance of the object is created, accessed, and in essence "lives" on a specific thread. Because this thread is the only one that can access this object, it is guaranteed that the instructions currently running on this thread will see the object in a consistent state. The proper use of a thread local variable mandates that a reference to the thread local object is never published to another thread. In addition, a proper design will limit the number of unique threads that request and hold thread local instances of a variable.

Several sections of the code, including hashing, solution evaluation, and the spatial update algorithms, can benefit from thread local variables. For a large problem (the *worldmpa jun2007* data set with 176,093 cells and 1038 surrogates), consider that the DvConfiguration requires about 22kB of storage and the Results require about 9kB (with integer representation and replication). We use these estimates in the following discussion to describe how much memory allocation can be saved with thread local variables.

Hashing requires the value of the decision variables to be placed in a byte buffer. Without thread local variables, this byte buffer (and the message digest object) would have to be re-allocated every time a solution is hashed. Thus, with each thread local hash, we conserve about 22kB.

If the search is evaluating solutions without archiving the results, then it is possible to re-use both the DvConfiguration and Results objects in a thread local context, saving about 31kB per evaluation. First, proper values are stored in the thread local DvConfiguration and Results, which are then sent to the objective function for evaluation. Since the objective does not maintain a reference to the either object *and* we are not committing the solution to the archive, no reference to the thread local variables escapes, and they are safe to re-use for the next evaluation (as long as any old values are properly overwritten). However, since neither the Results nor the DvConfiguration for any of the solutions were stored in that iteration, it is necessary to perform one extra evaluation at the end of the iteration to obtain a fresh set of Results and DvConfiguration for the new incumbent solution. (Since these are stored in the archive, they cannot be created with the thread local variable.) Over the course of 1,000 iterations (13,500,000 evaluations), this strategy prevents the allocation and de-allocation of about 400GB for the cost of 1000 extra evaluations.

Finally, extra savings can be achieved with thread local versions of the data structures that compute the spatial characteristics of each solution. These tracking data structures (stacks, arrays, and BitSets) require a minimum of 25kB of memory and require significantly more memory if the clusters in the solution are large. Without thread local variables, they would have to be reallocated with every evaluation. Again, over the course of 1,000 iterations, re-using these data structures can prevent the allocation and de-allocation of hundreds of gigabytes.

It is estimated that improved memory management with thread local variables is responsible for about 70% of the improvements reported since the earliest benchmarks.

2.9.2 Hashing

Early profiling indicated that hashing was a bottleneck in terms of computational effort, requiring more time than the function evaluation itself. Thus hashing should be avoided if possible; indeed, the program only needs to hash the solutions that will be stored in the archive. If the program archives just the solutions that are on the search trajectory, then hashing the other candidates is not absolutely required (although the search will not be able to determine if these solutions have previously been visited). In ConsNet, it has been demonstrated that a solution is rarely (if ever) encountered twice, and thus hashing *every* solution is an unnecessary computational burden. The search mode called “minimal archiving” saves considerable time by hashing only the solutions placed in the archive.

This design choice is correct for ConsNet, but it is not a general rejection of hash based methodologies. ConsNet still carefully tracks solution identity for those solutions that appear on the search trajectory, and can implement reactive strategies based on this information. Moreover, there are other search modes that can perform different levels of hashing and archiving, and these may be better suited to problems that have smaller solutions spaces or longer evaluation times.

The hashing speed itself was greatly improved by using a thread local *direct byte buffer* (instead of a *non-direct byte buffer*) to collect and store the hash bytes from the decision variables. A direct byte buffer is capable of taking advantage of some of the native I/O operations (below the virtual machine level).

The decision to hash only the incumbent solutions, along with the faster hash techniques accounts for about 20% of the improvement since the earliest benchmarks.

2.9.3 Multithreaded Management

Because the evaluation times in ConsNet can be extremely short, it can be difficult to construct an efficient multi-threaded approach to parallelize a large batch of function evaluations. The main problem is that the overhead associated with creating the tasks and coordinating the worker threads can be more costly than the function evaluation itself. One way to evaluate a multi-threaded design is to compare the performance of a serial version of the code against a version that distributes the tasks to multiple worker threads (one thread for each processor). Ideally, on a two processor system the multi-threaded evaluations will occur twice as fast, but in reality the overhead cuts into this performance.

One way to reduce the overhead is to eliminate any portions of the parallel code where threads have to wait on a common synchronization lock. Since both the solution archive and the objective archives are synchronized, one logical step is to avoid accessing these archives. Thus, the decision to use "minimal archiving" removed a major source of thread contention. However, even after a careful revamping of the concurrent structures in the MASTS code, there was still some noticeable thread contention, which was even more pronounced on a four processor machine. A profile showed that the threads were being forced to wait at the work queue. All four worker threads were pulling tasks form a single synchronized queue, and they were quite literally starving as they waited.

One potential correction is to provide each worker thread with its own queue. Tasks are distributed evenly among the queues. If one worker thread finishes all of its assigned tasks before the others, it is allowed to steal a unit of work from a thread that has not yet finished. This approach to the multi-threaded management is responsible for about 10% of the improvement since the earliest benchmark.

2.10 Conclusions

MASTS was designed as a multi-purpose optimization platform. During development, it was simultaneously tailored to two very different decision problems, demonstrating that the interface is suitably generic for a wide variety of problems. MASTS goes well beyond optimization to function as a decision aid, driven by the principle of ongoing interactive analysis. The features and interface are an excellent complement for large planning exercises. Users are able to build and manage portfolios of preferred solutions and save their progress. Using multiple objectives, the problem can be explored from a variety of individual or group perspectives. The GUI provides real-time feedback and enhances understanding of the problem. In addition, the solution archive can provide significant savings in time and effort when new objectives are introduced. One of the most useful features of MASTS, the rule based objective, is discussed thoroughly in the next chapter.

2.11 Figures

Figure 2.1 The layers of abstraction within the MASTS framework. The most important layer is the abstract search elements. A proper design here creates a common language that we can use while designing the abstract search strategies and search engines.

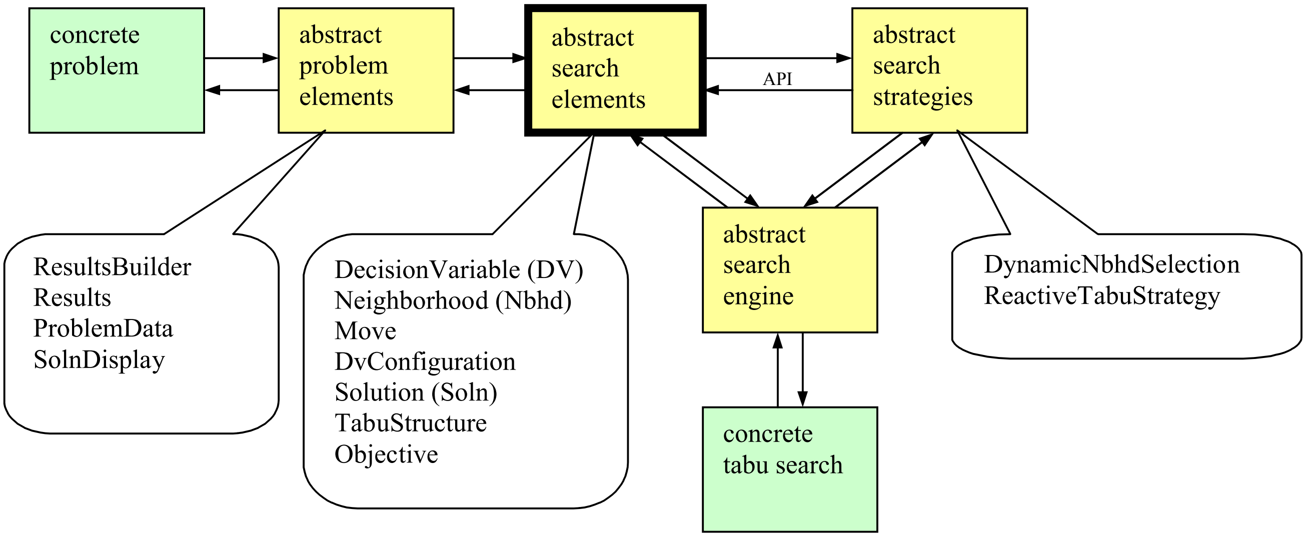


Figure 2.2 MASTS consists of several top level controllers (or "services") that issue commands and request data from the program components. Each service is designed to operate independently, handling requests and performing tasks in a multi-threaded environment. These services constitute the "controller" in the model-view-controller pattern.

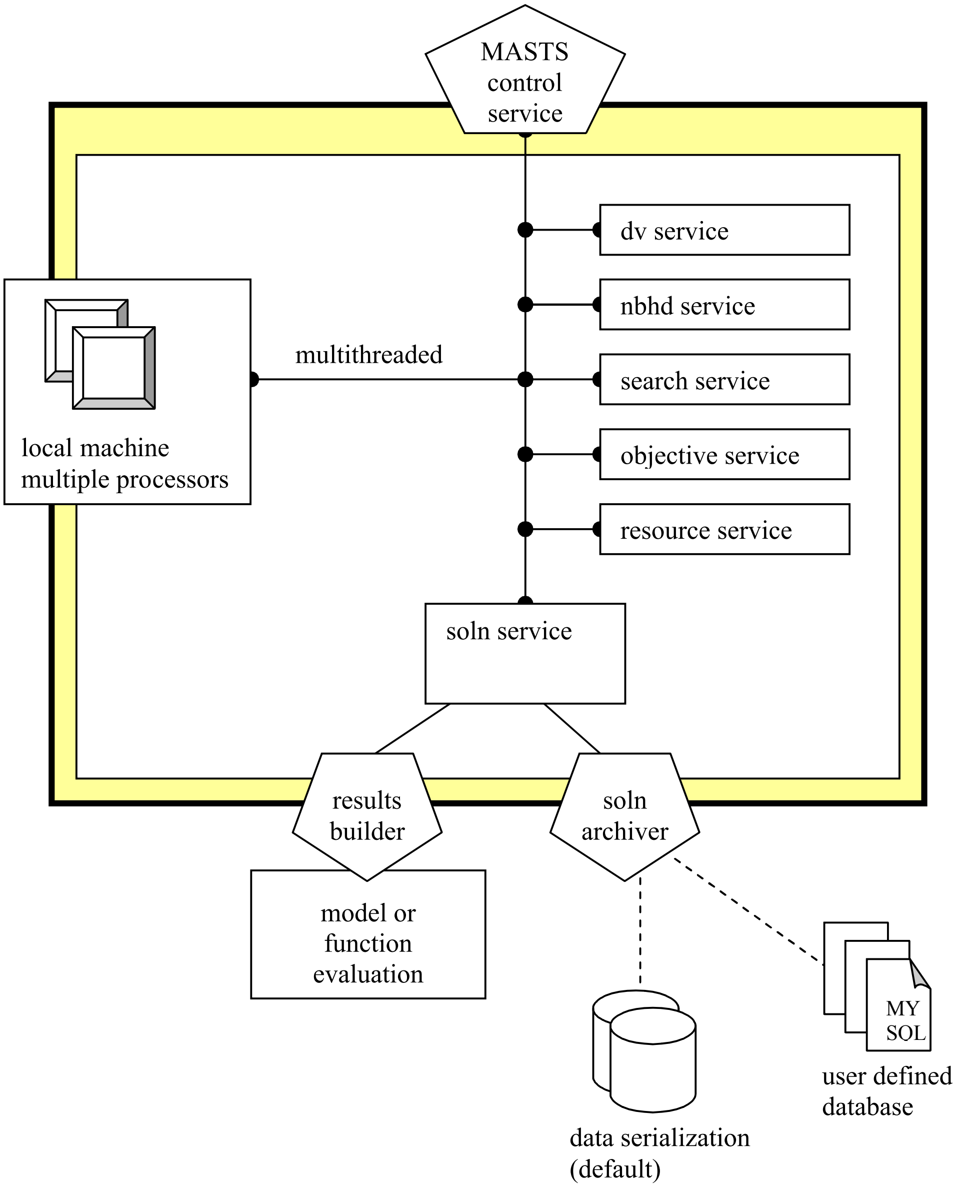


Figure 2.3 Two common types of decision variables.

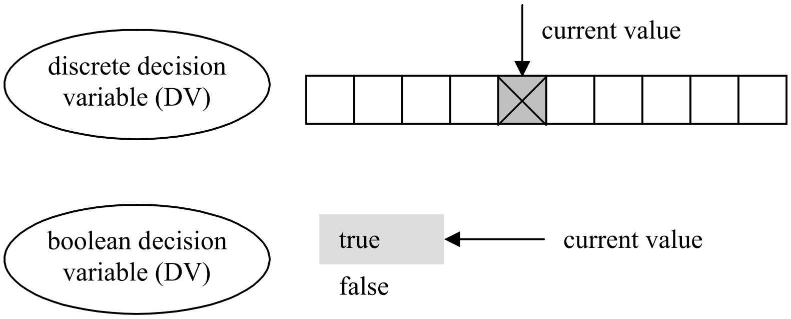

current value
discrete decision variable (DV)
boolean decision variable (DV)
true
false
current value

Figure 2.4 An example of a complex "partition" decision variable: the pumping allocation problem as a partition among bins. The bins are different sizes, indicating that the pumping centers have different available capacities. All of the extra pumping must be allocated.

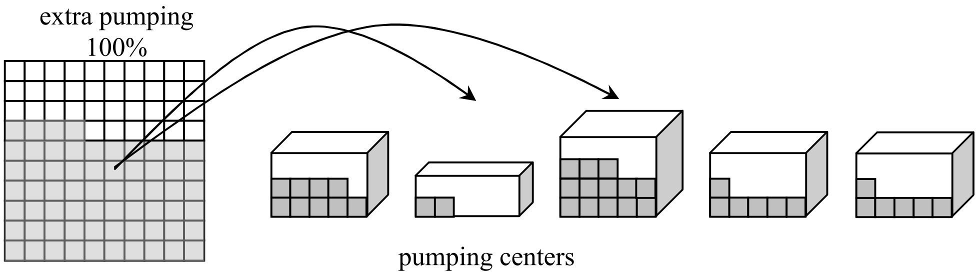


Table 2.1 An abridged summary of the DV class. Note that the "value" of the decision variable is given by a generic V. This means that any object may serve as the "value" of the decision variable. In addition, the decision variable must provide a unique byte array that represents the value. This byte array can be used to create a hash.

Method Summary	
`String`	`getName()`
`V`	`getValue()` Get the value of the decision variable, returning null if the decision variable is inactive.
`boolean`	`isAllowableValue()` Checks to see if the currently set value is an allowable value.
`abstract boolean`	`isAllowableValue(Object value)` Checks the following: value is the proper class and has a legal value.
`protected void`	`setValue(V value)` Sets the value of the decision variable, without checking the value.
`(package private) void`	`setValueObject(Object o)` A method used when we are loading objects from an archive, and we are uncertain if they are still valid.
`abstract byte[]`	`valueToByteArray()` Used in hashing, the MD5 hash algorithm reads byte arrays.

Figure 2.5 The interactions between decision variables, moves, and neighborhoods can be modeled as a tightly coupled triplet. This general structure can be extended to arbitrarily complex situations.

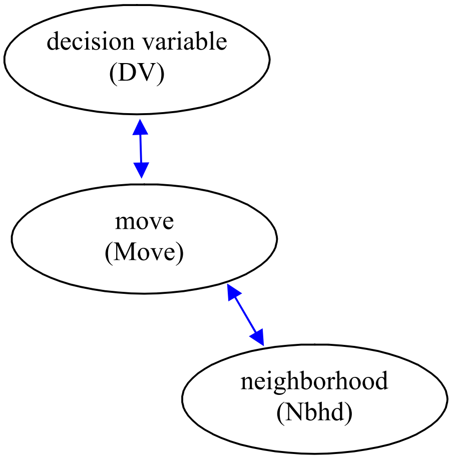
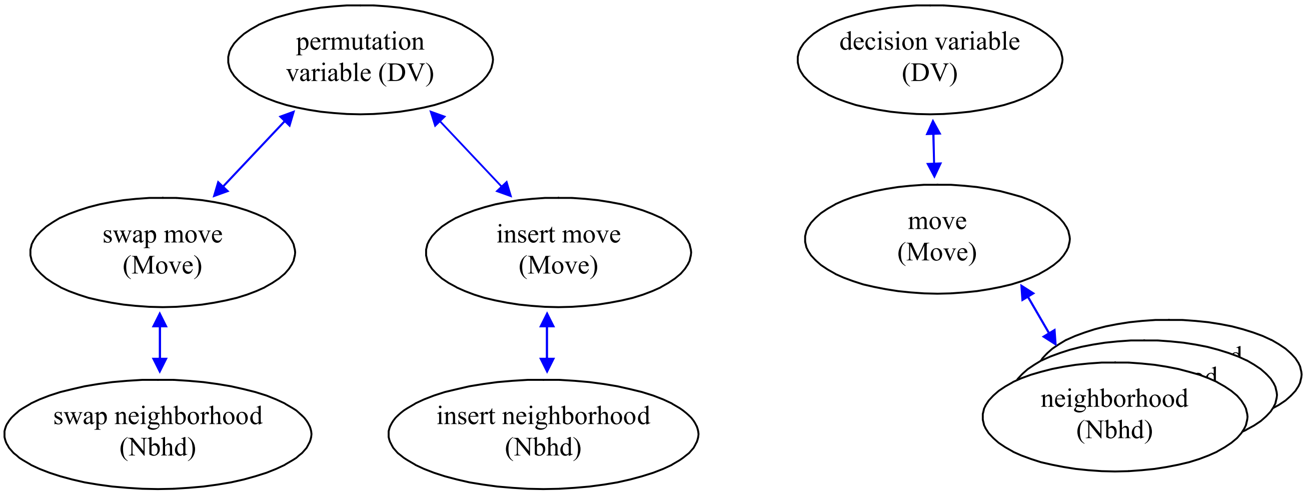


Figure 2.6 More complicated arrangements for the interaction between decision variables, moves, and neighborhoods.



Multiple move types may be applicable for a decision variable. Here, different neighborhoods generate different move types for the same decision variable.

Multiple neighborhoods. Each generates the same type of move, but uses different rules to create a set of potential moves.

Figure 2.7 One option for creating a neighborhood and moves for a discrete decision variable. Here, the moves incrementally adjust the value of the discrete variable. The neighborhood creates an “annulus” of moves centered at the original value. If the inner radius $r_i = 2$ and the outer radius $r_o = 4$, then the neighborhood will generate 6 potential moves. When executed, each move will change the value of the decision variable.

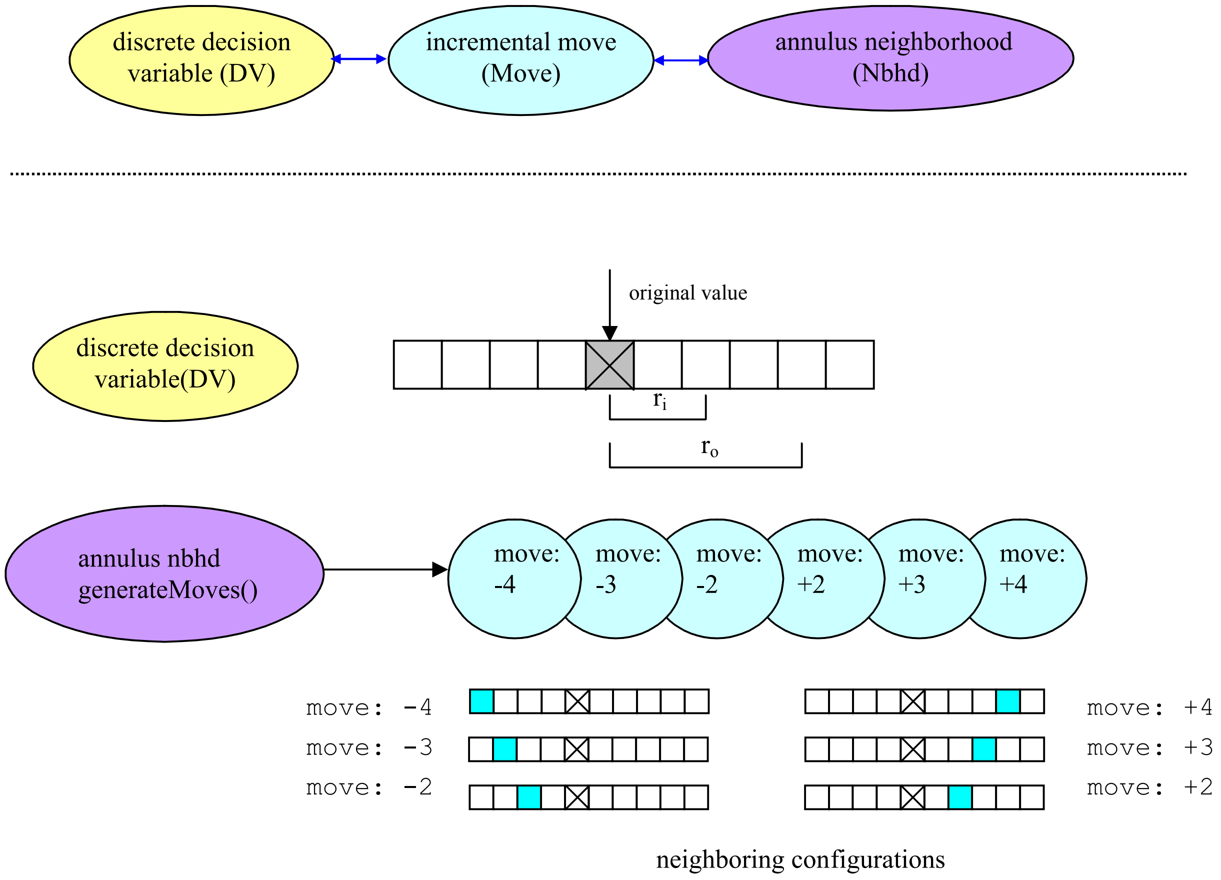


Figure 2.8 A tabu list can be attached to a specific move class for each decision variable.

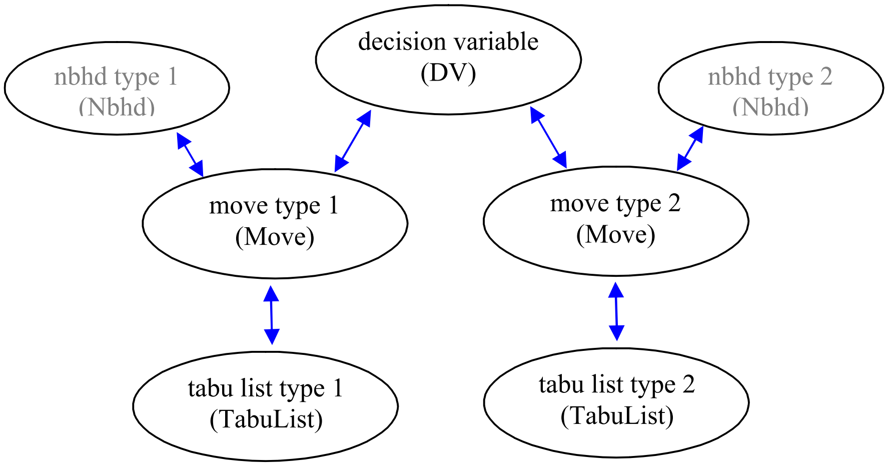


Figure 2.9 A tabu list can work with more than one type of move, if the moves extend the same class (or implement the same interface). Thus, one tabu list can track both inserts and swaps.

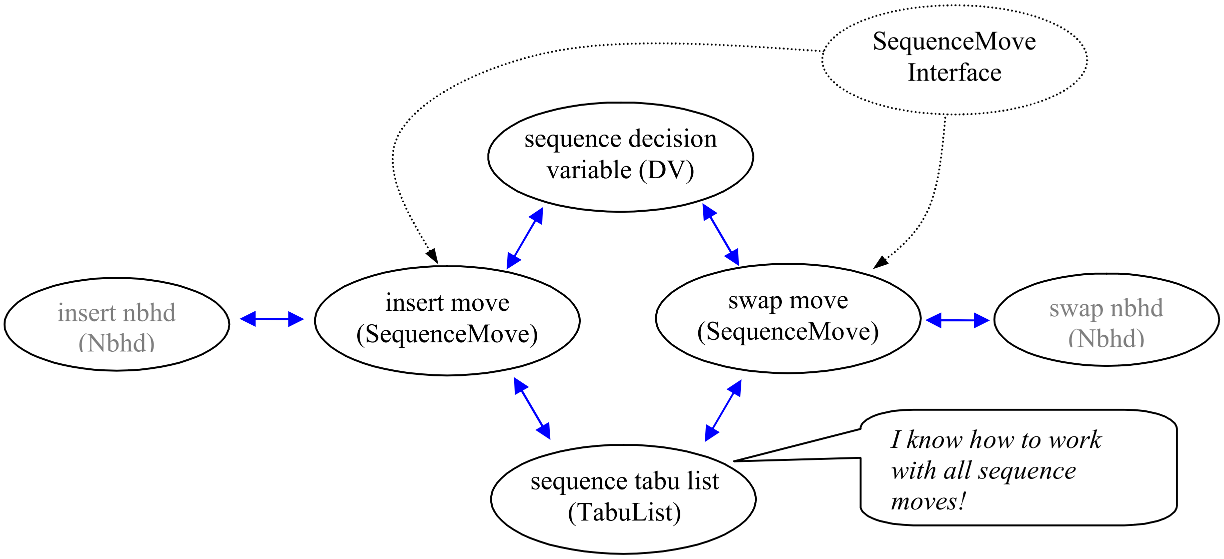


Table 2.2 A quick summary of the methods in the abstract TabuList class. Each tabu list knows which decision variable it belongs to and which moves it may operate on.

Method Summary	
`int`	`getCurrentTenure()` Get the tenure that is currently being assigned when a tabu restriction is declared.
`int`	`getDvIndex()` The dvIndex for the decision variable that this list operates on.
`int`	`getInitialTenure()` The tenure used when the list is first created.
`int`	`getMaxTenure()` The maximum allowable tenure.
`int`	`getMinTenure()` The minimum allowable tenure.
`Class<M>`	`getMoveClass()` Declares which types of moves this tabu list operates on.
`TabuList<M>`	`getNewInstance()` Gets a fresh instance of this tabu list, properly initialized.
`boolean`	`isTabu(M move, int iteration, Soln newSoln)` Check to see if the move/solution is tabu.
`boolean`	`setCurrentTenure(int tenure)` Set the tenure that will be assigned within this list when a move/soln is declared tabu.
`void`	`setInitialTenure(int initialTenure)` Specifies the tenure to be used when the list is first created (the set method is available so that different types of algorithms may use different initial tenures).
`void`	`setMaxTenure(int maxTenure)` Specifies the maximum tabu tenure that is allowed (the set method is available so that different types of algorithms can use different min and max tenures).
`void`	`setMinTenure(int minTenure)` Specifies the minimum tabu tenure that is allowed (the set method is available so that different types of algorithms may use different min and max tenures).
`void`	`setTabu(M move, int iteration, Soln newIncumbentSoln)` Set the tabu status based on the move/solution/iteration.

Table 2.3 The methods in the TabuReactor interface. The tabu reactor houses the individual tabu lists to keep them organized. In addition, it implements the reactive tabu search. The react() method gives the tabu reactor a chance to change the tenure on some or all of the lists.

Method Summary	
`TabuReactor`	`getNewInstance()` Get a new instance of this tabu reactor.
`String`	`getUniqueName()` Tabu reactors must provide a unique name.
`void`	`intitialize(TabuStructure tabuStructure, TabuListService tabuService, ProblemData problemData)` Initialize the taub reactor with problem data.
`boolean`	`isTabu(Move move, int iteration, Soln soln)` The search calls this method to see if the move/soln is tabu; iteration is the current iteration.
`void`	`react(IterationSummary iSummary)` After the search has evaluated the candidates for one iteration, the tabu reactor is given a chance to react.
`void`	`setTabu(Move move, int iteration, Soln soln)` After the search decides which move it is going to accept, this function examines that move and finds attributes to set as tabu.

Figure 2.10 A solution to the optimization problem can be described as an array of decision variables. The DvConfiguration class contains this array. New configurations are created using the guardedClone() method.

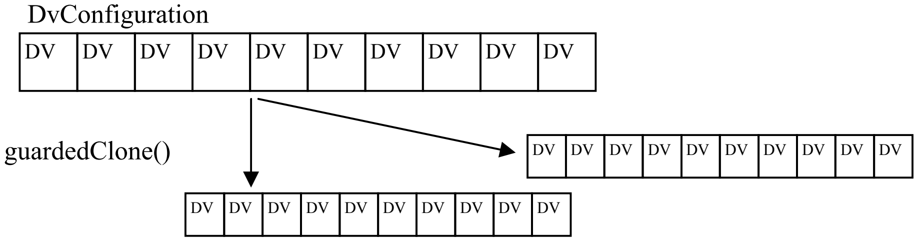

DvConfiguration
DV DV DV DV DV DV DV DV DV DV
guardedClone()
DV DV DV DV DV DV DV DV DV DV
DV DV DV DV DV DV DV DV DV DV

Table 2.4 The number of hashes needed to generate a collision with specified probability, assuming all hashes are equally likely.

Bits	**Possible outputs**	**Desired probability of random collision**							
		10^{-12}	**10^{-9}**	**10^{-6}**	**0.1%**	**1%**	**25%**	**50%**	**75%**
64	1.8×10^{19}	6.1×10^{3}	1.9×10^{5}	6.1×10^{6}	1.9×10^{8}	6.1×10^{8}	3.3×10^{9}	5.1×10^{9}	7.2×10^{9}
128	3.4×10^{38}	2.6×10^{13}	8.2×10^{14}	2.6×10^{16}	8.3×10^{17}	2.6×10^{18}	1.4×10^{19}	2.2×10^{19}	3.1×10^{19}
256	1.2×10^{77}	4.8×10^{32}	1.5×10^{34}	4.8×10^{35}	1.5×10^{37}	4.8×10^{37}	2.6×10^{38}	4.0×10^{38}	5.7×10^{38}
384	3.9×10^{115}	8.9×10^{51}	2.8×10^{53}	8.9×10^{54}	2.8×10^{56}	8.9×10^{56}	4.8×10^{57}	7.4×10^{57}	1.0×10^{58}
512	1.3×10^{154}	1.6×10^{71}	5.2×10^{72}	1.6×10^{74}	5.2×10^{75}	1.6×10^{76}	8.8×10^{76}	1.4×10^{77}	1.9×10^{77}

Source: Wikipedia: “Birthday Attack”

Figure 2.11 MASTS assumes that a black box model can evaluate a DvConfiguration and return a set of meaningful results. A class implementing the ResultsBuilder interface provides the specific functionality of the black box. The model may be a quick evaluation or an extended evaluation. Once the results are computed, the DvConfiguration and Results are packaged together to form a Soln.

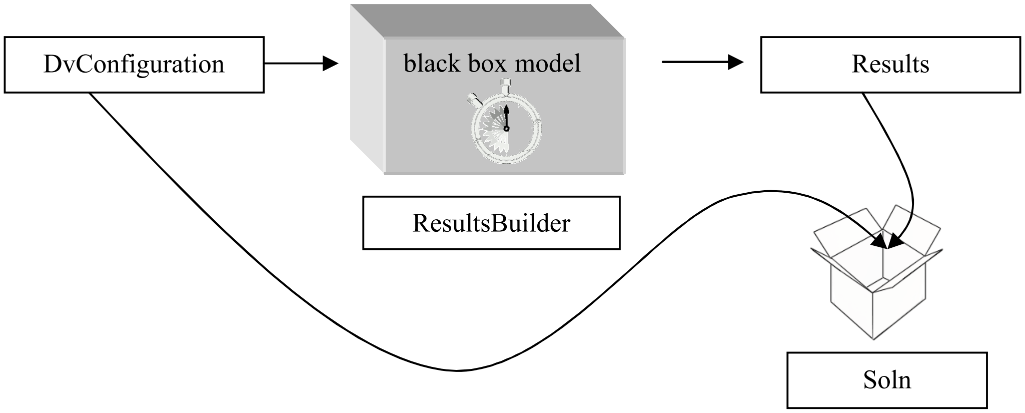
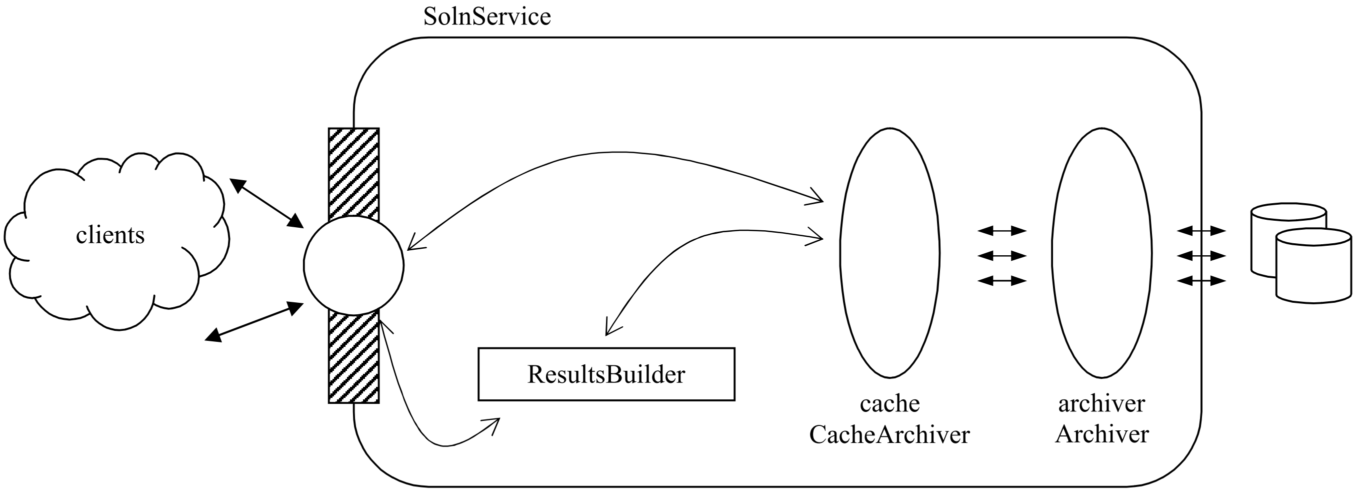


Table 2.5 The ResultsBuilder interface has only two methods. The buildResults method should be designed to run simultaneously on multiple threads.

Method Summary	
`Results`	`buildResults(DvConfiguration dvConfig, Move previousMove, Soln previousSoln)`
`void`	`startup(ProblemData problemData)` Gives initial access to the problem data object.

Figure 2.12 The SolnService manages the creation and storage of solutions. It must be capable of handling simultaneous requests from different threads. The cache holds recently evaluated solutions for quick access, and the archiver transfers solutions from memory to hard drive storage.



3 Objectives in MASTS

Objectives are the most flexible part of MASTS. In order to address the formidable complexities of real problems, the objective interface within MASTS must encompass a wide range of possible decision rules. Since the objective function drives the search, its exact phrasing is absolutely critical. MASTS introduces Rule Based Objectives (RBO) as a flexible and precise way to rank solutions using pairwise comparison operators rather than numeric scores. With rule based objectives, MASTS becomes a sophisticated decision analysis tool, capable of modeling complex preferences even in situations when the RBO defines an ordering that is not linear or changes over time.

3.1 Rule Based Objectives

MASTS deals with search objectives using a new, significantly more powerful convention. Classical objective functions return real-valued numeric scores, and in the case of minimization, the solutions with lower scores are deemed superior. In MASTS, the user does not have to provide an objective score in the form of a single number (or vector of numbers). Instead, the user is required to provide a binary comparison operator, $\preceq$, that considers two different solutions and assesses if the first is superior, equivalent, or inferior to the other. Score based objectives are a subset of RBOs where $\preceq$ assumes the natural ordering for real numbers to determine the superiority (or equivalence) between two solutions.

An RBO is sufficient to drive the search toward optimality because TS requires only an ordinal ranking of solutions; at each iteration, TS selects the best solution that is not tabu. In contrast, simulated annealing requires an interval scale (to quantify differences in two solutions) and some genetic algorithms require a ratio scale (to compare fitness and determine likelihood of reproduction).

RBOs can create extremely precise and detailed orderings of alternatives compared to conventional score based objectives. Classical optimization methods usually rely on a single numeric score, which is often a weighted composite of different attributes of the solution, frequently constructed in an arbitrary manner. As the number of different attributes grows, it can difficult to rationally transform them into a single number in a manner that is consistent with the user's intentions. Rather, it may be advantageous to replace numeric scores with a logical set of rules that impose order on the solution space.

In addition, there are multi-criteria situations where it may be undesirable or difficult to work exclusively with numeric scores. When two criteria represent different value systems, it may be impossible to aggregate such criteria scores in a meaningful manner (Sarkar, 2005). For example, how does one compare preserving endangered species with fostering economic development? Insisting on a single valued objective score leads to inevitable aggregation of disparate quantities, requiring assumptions that many decision makers find intolerable. RBOs have the flexibility to incorporate established methods from the field of multi-criteria analysis (MCA) (Figueira, Greco, & Ehrgott, 2005) and are well poised to take advantage of MCA methods that provide ordinal rankings of alternatives.

Finally, since the comparison is performed with two alternatives at a time, a fairly sophisticated RBO can make decisions based on the relative differences of two alternatives. Many MCA methods are also founded on pairwise comparison operators, such as Regime (Hinloopen, Nijkamp, & Rietveld, 1983) and the multiplicative AHP (Lootsma, 1993). Loomes and Sugden (1982) propose a model of regret and rejoice that is based on examining two alternatives at a time. In contrast, conventional numeric scores are assigned individually, without reference to any other solution. In essence, the pairwise comparison admits an "extra dimension" of information that can be used to make a decision; although taking advantage of this information could generate a non-linear ordering (discussed in the next section).

3.2 Assumption and Relaxation of Linear Ordering

Let A be a set of known alternatives, and $\preceq$ be a relationship defined on that set. The $\preceq$ operator yields a linear (or total) ordering of A if by definition:

i) $\forall$ x $\in$ A   x $\preceq$ x (reflexive)

ii) $\forall$ x,y $\in$ A x $\preceq$ y and y $\preceq$ x $\Rightarrow$ y = x (antisymmetric)

iii) $\forall$ x,y,z $\in$ A x $\preceq$ y and y $\preceq$ z $\Rightarrow$ x $\preceq$ z (transitive)

iv) $\forall$ x,y $\in$ A x $\preceq$ y or y $\preceq$ x (completeness)

In software, the requirements of a linear order guarantee that the elements of A may be arranged in a consistent, repeatable order, an important programmatic consideration when dealing with sorted collections. However, a linear ordering can be an overly strict requirement when dealing with preferences. Fishburn (1991) surveys decision models where intransitivity may be admissible; particularly if decisions are made under risk or uncertainty. In addition, "weak" forms of intransitivity have surfaced in revealed preference theory (Kim, 1987). Mandler (2005) shows that intransitivity of choice may in fact be "rational" in some sequential decision environments. Ciarleglio et al. (2007) present empirical evidence that intransitive orderings can be used by a tabu search to find better descent paths. To investigate further the potential of intransitive orderings, and to find better decision models to drive the search, we will build an analogy between metaheuristics and *revealed preference theory* (Samuelson, 1938; Sen, 1971).

Another difficulty may arise if the comparison operator changes over time, which could happen for a variety of reasons. Whenever the definition of $\preceq$ changes, it is possible that a previous ranking created with $\preceq$ is no longer valid. This introduces the possibility of *rank reversal*, where the ordinal ranking of two alternatives can be inverted when $\preceq$ changes. For the particular case when $\preceq$ depends on the set of known alternatives A, it is possible that the act of adding or removing alternatives from A can cause a rank reversal. In the extreme case, the comparison operator may change every time a new alternative is introduced.

In the sections that follow, we find evidence that intransitivity and rank reversal are likely to occur in practical decision contexts. The flexibility of the rule based objective approach can easily accommodate these unconventional orderings.

3.3 Revealed Preference in Metaheuristics

For the following discussion, we anthropomorphize the search and treat it as a rational agent making a sequence of decisions. Serving as a proxy agent for the user; the search emulates the user's preferences and acts on the user's behalf. Following Mandler (2005), we distinguish between the agent's *psychological preferences* and its *choice behavior*. The psychological preferences involve judgments about the agent's long term welfare or well-being, based on a rationally constructed world view. Psychological preferences encapsulate an ideal outcome, which may or may not be attainable. Since the search does not have its own psychology, we will call these preferences *outcome preferences*. In the context of metaheuristic search, the goal is to find a dominating (or at least non-dominated) alternative with respect to the outcome preferences.

Now imagine a case where the outcome preferences are undisclosed, poorly understood, or incomplete. In this setting, it may be possible to characterize the outcome preferences by observing choice behavior. In repeated trials, the agent must select from a given, non-empty set of alternatives (the choice set). A *choice function* is a general type of function that examines the choice set and returns a subset of those alternatives which are chosen. The decisions made at this level are known as the choice behavior, and these choices uncover a set of *revealed preferences*. Ideally, the revealed preferences will be *outcome rational;* the choice sequences made by the agent will not lead to a dominated outcome when appraised by the outcome preferences (Mandler, 2005).

Outcome preferences, built on careful consideration and economic rationality, are assumed to be transitive. However, the demands of rationality do not necessarily require that the outcome preferences are complete. In contrast, the revealed preferences are fundamentally complete; the agent *must* make a decision no matter what alternatives are presented. It seems logical that revealed preferences should also be transitive, but this is not necessarily the case. It may be possible to achieve outcome rationality with intransitive revealed preferences (see Section 3.4).

Whether using a conventional numeric score or a rule based objective, it should be clear that the objective is a proxy for the outcome preferences of the agent, but may not be a perfect model for the revealed preferences; the ordering specified by the revealed preferences may not always agree with the ordering specified by the outcome preferences. *When presented with a choice set, any search agent that chooses something other than a steepest descent alternative is exercising a revealed preference that is different from the outcome preference.* If the revealed preference were congruent to the outcome preference, then the search would always choose a steepest descent alternative, and would almost certainly be trapped in a local optimum (except in rare circumstance where steepest descent terminates at the global optimum). Since local optima are often dominated by other solutions, this situation does not fit the definition of outcome rationality.

At this point, we may suppose that two closely related but different orderings exist ($\preceq_R$ for the revealed preferences, and $\preceq_O$ for the outcome preferences). When the revealed preference structure is not congruent with the outcome preference structure, the revealed preference *overrides* the outcome preference. Advanced search methods occasionally override the outcome preference by choosing something other than a steepest descent alternative. At some point in the search, it is possible that $a_i \preceq_R a_j$ while $a_j \preceq_O a_i$. A situation where $a_i \preceq_R a_k \preceq_R a_j \preceq_O a_i$ does not imply intransitivity because $\preceq_R$ and $\preceq_O$ are different comparison operators; taken separately, both operators could be transitive.

In practice, a common assumption is that the objective function is a perfect proxy for the user's outcome preference. If the subscript S represents the *search* and the subscript U represents the *user*, then $\preceq_{O,S}$ is identical to $\preceq_{O,U}$, i.e., there is only one set of outcome preferences $\preceq_O$. This naturally leads to the question of how and when the user's revealed preferences are defined. In typical search methods (both metaheuristic and classical), the user is not explicitly asked to define a choice function which would expose the revealed preferences. Instead, a search method uses a predefined choice function as a proxy for $\preceq_{R,U}$. Tabu search, simulated annealing, and genetic algorithms all use *different* choice functions built around $\preceq_O$. This choice function makes decisions at each iteration and yields a revealed preference $\preceq_{R,S}$ for the search, presumably different from that of the user.

Since the revealed preference of the search acts on behalf of the human agent and will occasionally override the outcome preference, a natural question is "Would a human agent make similar choices?" A variety of different arguments strongly indicate that the answer is yes. Since expert *human* problem solvers created metaheuristics, occasionally overriding the outcome preference is demonstrably a human trait (as well as a viable approach for decision making strategies). Further justification is intertwined with the definition of an *extended choice function* (Mandler, 2005), where the current choice set presented to the user depends on prior choice sets and decisions, i.e., the current decision not only affects your holdings, it may also affect your future choices. This linkage admits the use of extended search strategies that unfold over several iterations. In most practical problems, there is at least an intuitive understanding of how the current decision can influence future choice sets. Thus, forward thinkers will often identify potential *gambits*, choice sequences that may lead to better overall solutions by accepting a less desirable solution in the current choice set. Experienced and effective problem solvers employ such gambits intuitively. Thus, in a direct search methodology, a revealed preference overriding an outcome preference is not an aberration of the algorithm and its agent. Rather, similar behavior is known to be exercised by human agents.

Properly executed models of revealed preference can significantly improve the search. For example, rule based objectives provide an opportunity to embed *problem specific* gambits into $\precsim_{R,S}$ which can lead to critical improvements in the search performance; this approach was used in (Ciarleglio et al., 2007). By accepting something other than the steepest descent alternative, gambits are particularly effective at evading nonproductive sub-regions of the search space.

In a MASTS rule based objective, the revealed preferences are modeled as a binary comparison operator, and separate definitions can be used for outcome preferences ($\precsim_{O}$) and revealed preferences ($\precsim_{R,U}$). Allowing users to provide a separate comparison operator for the revealed preferences provides greater control over the choice behavior of the search, enabling gambit strategies that are safely segregated from the outcome preferences. In practice, $\precsim_{O}$ is used to rank and display the stored solutions, and $\precsim_{R,S}$ is used when the search is making decisions at each iteration (see Figure 3.1).

3.4 Intransitivity in Metaheuristics

Prior to considering intransitive decision operators, we examine $\precsim_{R,tabu}$, the revealed preferences assumed by a basic tabu search which employs a linear preference ordering for $\precsim_{O}$ (analogous to a numeric objective function). For a single iteration, the choices made by the tabu search reveal linear preferences $\precsim_{R,tabu}$ which follow a strict hierarchy, presented in Figure 3.2. First, a solution that is not tabu is always preferred over a solution that is tabu. Given two solutions that are not tabu (or both tabu), the search will choose the solution that is preferred by $\precsim_{O}$. Thus, if *all* of the solutions are tabu, the search chooses the solution preferred by $\precsim_{O}$. The logic inherent in the tabu memory structure can be interpreted as a gambit. Simulated annealing uses a much simpler gambit in its revealed preferences, i.e., randomly selecting disimproving moves. *Calculated, purposeful gambits can improve the search performance*. The gains can be even more impressive when the gambits are tailored for a specific problem, as demonstrated by the rule based objectives in ConsNet.

The model presented above for $\precsim_{\mathrm{R,tabu}}$ is simplistic. The tabu memory structure (and thus the revealed preferences) change at each iteration. Including aspiration criteria further complicates the theoretical analysis of the search because the resulting comparison also depends on the set of known alternatives (in particular, the best known solution). The overall goal of these strategies is to create "intelligent" revealed preferences based on the main objective $\precsim_{\mathrm{O}}$. However, $\precsim_{\mathrm{O}}$ may not be the most effective way to make decisions at each iteration, particularly if problem specific gambits can be leveraged. Clients of MASTS can provide a user defined revealed preference operator $\precsim_{\mathrm{R,U}}$ that will be used by $\precsim_{\mathrm{R,tabu}}$ to make comparisons and select the next incumbent solution (see Figure 3.2).

During the initial development of RBOs, logical but intransitive comparison operators for $\precsim_{\mathrm{R,U}}$ were proposed. As discussed in Section 6.6, these operators embodied intuitive gambits exploiting the problem structure. Since intransitivity is widely perceived as irrational, the following concerns arose:

- Could the intransitive behavior be justified as a reasonable alternative to transitivity?
- Would intransitivity adversely affect the outcome of the search?

While these questions cannot be answered conclusively, the use of intransitive preference can be justified in many scenarios. Fishburn (1991) and Anand (1993) challenge the assumption of transitivity as a cornerstone of rationality, presenting many scenarios where intransitivity may be more appropriate. In this particular application, the intransitive comparison operators appear to "make sense" to experienced problem solvers of proven capability, and outperformed similar transitive comparison operators (Ciarleglio et al., 2007). Thus, sufficient evidence (both theoretical and empirical) suggests that intransitivity may be admissible and defensible.

The more difficult question is whether intransitivity will adversely affect the outcome of the search. To address this topic, the different types of revealed preferences that one could define programmatically must be specified:

- static revealed preferences – the comparison operator $\precsim_{\mathrm{R,U}}$ does not change over time or with the addition of new alternatives

- dynamic revealed preferences – the comparison operator $\precsim_{\mathrm{R,U}}$ changes over time or with the addition of new alternatives
- stochastic revealed preferences – the comparison operator $\precsim_{\mathrm{R,U}}$ contains a stochastic element, and sometimes makes random decisions

Only static comparison operators $\precsim_{\mathrm{R,U}}$ have been studied in this research. If $\precsim_{\mathrm{R,U}}$ happens to be intransitive, it is possible to observe (within a single iteration) alternatives *x*, *y*, and *z* such that $x \precsim_{\mathrm{R,U}} y \precsim_{\mathrm{R,U}} z \precsim_{\mathrm{R,U}} x$ (performed as pairwise comparisons). Because $\precsim_{\mathrm{R,U}}$ is static, this intransitivity is not a result of changes over time; the comparison operator itself is fundamentally intransitive.

Even if $\precsim_{\mathrm{R,U}}$ is static, the revealed preferences of the search $\precsim_{\mathrm{R,tabu}}$ (which are based on $\precsim_{\mathrm{R,U}}$) are dynamic because the tabu memory structure changes over time. Dynamic preference operators can appear to be intransitive over time. When the comparison operator is updated, rank reversals may occur. Consequently, comparisons performed at different times using the altered comparison operators can exhibit *temporal intransitivity*. Stochastic comparison procedures, such as those used by simulated annealing and genetic algorithms, will also exhibit temporal intransitivity. Thus, temporal intransitivity is already a fundamental part of metaheuristics. In proposing a static intransitive comparison operator for $\precsim_{\mathrm{R,U}}$, it is necessary to consider the implications of intransitivity that is *not* the result of changing preferences, but rather a fixed property of the comparison operator.

In a similar (but not precisely parallel) situation, Mandler (2004) shows that these two types of intransitivity may be closely related. According to Mandler, the following conditions are equivalent explanations for the preferences exhibited by a decision maker seeking to maintain the status quo:

- Require transitivity and completeness, but drop the assumption of constancy, i.e., the preferences may change over time, allowing temporal intransitivity.
- Require constancy and completeness, but drop the assumption of transitivity, i.e., the preferences may be intransitive but do not change over time (static).

Thus, the temporal intransitivity that arises from a changing comparison operator may not be significantly different from the intransitivity that arises from a single static intransitive comparison operator.

Finally, Mandler (2005) proves that in some circumstances, there exists a set of intransitive revealed preferences that can lead to non-dominated outcomes. It should be noted that simulated annealing loosely qualifies as one such example. The stochastic nature of the revealed preferences is blatantly intransitive (at least in the temporal sense), but simulated annealing is known to asymptotically converge to a global optimum (Mitra, Romeo, & Sangiovanni-Vincentelli, 1986). Mandler offers this explanation:

> "A willingness sometimes to choose *x* over *y* and sometimes to choose the reverse can help an agent who cannot rank *x* and *y* to avoid manipulation and achieve outcome rationality."

The key to Mandler's proof is that the agent's psychological (outcome) preferences must be *incomplete*, meaning that there exist two alternatives x and y such that neither x $\preceq$ y nor y $\preceq$ x. Many multi-criteria decision problems exhibit incompleteness, particularly when the comparison between two alternatives cannot be reduced to a common value system. If incompleteness does hold, then Mandler's theorem shows:

> "… that intransitivity of choice is then consistent with outcome rationality: if an agent's psychological preferences are transitive and incomplete there exist extended choice functions that never lead to dominated outcomes but that generate intransitive revealed preferences. Hence, although one can use revealed preferences to assemble a complete ordering from an incomplete psychological preference relation, the ordering can be intransitive—but not irrational in the sense of leading to dominated outcomes."

While this proof does not show how to construct intransitive preferences, it strengthens the argument that intransitive preferences are not illogical, and in fact may lead to non-dominated solutions. The strong result from this dissertation, which seems to corroborate Mandler's theory, is that intransitive comparison operators are capable of finding optimal solutions (Ciarleglio et al., 2007). Moreover, search performance may be significantly improved if the intransitivity occurs as the result of strategic gambits.

3.5 Rank Reversal

When using a comparison operator that depends on the set of known alternatives, it is possible that the revealed preferences and/or the outcome preferences can be susceptible to rank reversal. Rank reversal is a feature in some decision models where the addition or deletion of an alternative from the choice set A can invert the ordinal ranking of two previously compared alternatives. There is a healthy debate over whether rank reversal should be admitted into a formal decision process. The question boils down to how seriously one takes the axiom of independence of irrelevant alternatives (IIA) (Arrow, 1951). This axiom is commonly interpreted as follows: if a_1 is preferred to a_2, then the introduction of another alternative to A must not make a_2 preferred to a_1. While this logic is appealing, it is perhaps inconsistent with real world decision making situations (Tversky & Kahneman, 1981). Let us consider the conversations given below that introduce different situations that question this axiom.

Conversation 1 *One situation where rank reversal seems unjustified.*

On a used car lot, you are presented with the same model car, either in red or blue. Preferring red over blue, you are ready to sign the paperwork and drive away in the red car. While in the office, another car (the same model but green) arrives on the lot. Presented with three options, you decide that perhaps you would prefer the blue car after all.

Conversation 2 *In this situation, rank reversal makes a little more sense.*

Marc works on Wall Street. Relying on an extended network of cronies and informants, Marc has made millions through insider trading. Every morning, he calculates which stocks are going to make the most money, and his list is never wrong or out of order. Before lunch, Marc always purchases the second most profitable stock. He's not stupid; if he bought the best stock every day, his success would draw suspicion and the SEC would probably launch an investigation. Marc's preference for stocks goes

$a_2 > a_1 > a_3 > a_4 > a_5$

Where a_1 is the most profitable stock, a_2 is the second most profitable and so on.
Today, at 10am Marc learns about a couple of new hot initial public offerings. According to his insider sources, one of them (a*) will be the big winner of the day. Realizing that a_1 is now the second most profitable stock, Marc updates his preferences in the following way:

$a_1 > a^* > a_2 > a_3 > a_4 > a_5$

The introduction of the new alternative has reversed the rank between a_1 and a_2.

In general, rank reversal can occur whenever new information becomes available. Conversation 2 demonstrates a type of rank reversal that occurs because of a preference based on the *order statistics* (such as the min, max, or second best) of the alternatives within the set A. Rank reversal can also occur when the decision model contains other self-referential measures (such as the mean, mode, sum, or cardinality) of the alternatives within set A. Rank reversal occurs in the Analytic Hierarchy Process (AHP) precisely because criteria are "normalized" over the alternatives within A (Saaty, 1980). This self-referential quality opens the possibility that the ordering could change every time an alternative is added. Since individuals often make decisions relative to the best or worst case scenario, there is nothing particularly objectionable about a comparison operator that allows rank reversal. For this reason, MASTS will be designed to accommodate rank reversal.

3.6 Objective Structure

The MASTS objective system relies on four core data types that interact within a top level manager called the ObjectiveService. Each data type is defined as an interface, permitting different implementations for most behaviors. One of these abstract types, the ObjectiveScore *object* does not necessarily represent a simple numeric score. The ObjectiveScore is an object which holds information (numbers and other data) that can be used to perform comparisons in our RBO framework. For this discussion, the word *score* will be "overloaded" to refer to an instance of this ObjectiveScore class. Each separate objective in the program requires an ObjectiveMaster to manage the sorting and evaluation of the objective scores. Since memory may be an issue if multiple objectives are loaded, the various components discussed below support managed memory reductions, including purges and caching.

ObjectiveService. The ObjectiveService manages a possibly dynamic collection of objectives. Clients may use this service to look up an objective by name and evaluate objective scores. The ObjectiveService creates and manages the ObjectiveMasters, one for each objective. When the program is closed, it serializes all objectives to the hard drive and then reloads them whenever the program restarts. The entire ObjectiveMaster (including the sorted scores) is saved as one unit.

Storing too many scores or defining too many objectives could create memory problems. The ResourceService can be configured to purge the objectives, removing all but the top 1,000 scores (or other specified amount) in each objective tree. This purge requires synchronized coordination between all loaded objectives and the solution archive. For this reason, the GUI services are temporarily suspended to prevent any errant clicking that may interfere with the purge. The ResourceService and ObjectiveMaster facilitate these purges in a thread safe manner.

ObjectiveMaster. The ObjectiveMaster keeps track of the essential components in each objective, including the ObjectiveKernel, ObjectiveScore, ObjectiveScoreComparators, and a sorted tree of scores that represents the current rankings. Access to the sorted tree must be properly synchronized. The ObjectiveMaster and all of its fields implement the Serializable interface, which allows the entire structure to be serialized to the hard drive. When memory becomes an issue, the ObjectiveMaster contains methods that purge the lowest ranking alternatives.

ObjectiveScore. One ObjectiveScore is created for each solution. The ObjectiveScore object holds the *bare minimum* of data that is required to compare this solution to others. For a simple problem, this data may be a single number. For a complicated problem, the score may consist of several data types. Since the number of scores could be large, each score should contain only the data which is needed for immediate comparisons. An ObjectiveScore has *no* reference to the Soln or Results. Instead, each score contains the serial number (or hash string) that can be used to locate the solution.

The ObjectiveScore has an evaluate() method that should be used to compute/store any necessary values that will be used during the comparison. No computations should be performed during the actual comparison process since it executes so often.

ObjectiveScoreComparator. This object contains the rules for comparing two different objective scores (which represent two different solutions). As described above, each objective may contain two different comparison operators: $\preceq_O$ which defines the preferred outcome (the true objective) of the search, and $\preceq_R$ which is the revealed preference used by the search to choose the new incumbent solution. This interface extends the Java Comparator with the usual behavior assigned to the compare(ObjectiveScore score1, ObjectiveScore score2) function. This function returns a negative integer if score1 is *superior to* score2, zero if they are *equal*, and a positive integer if score1 is *inferior to* score2.

Since the comparison function may run millions of times, it should avoid performing any calculations that could be pre-computed. The ObjectiveComparator is used as the ordering for the sorted score tree, and to determine the best neighboring solution at each iteration in the search.

ObjectiveKernel. The ObjectiveKernel is the “common” data that is accessible to each of the scores and the comparator. It contains data that is required for comparing objective scores. Each score can access this kernel to look up relevant data.

3.7 Robustness Despite Nonlinear Orderings

In the sections above, I have discussed two types of nonlinear orderings related to intransitive comparison operators and rank reversal. These orderings can have potentially adverse affects on the data structures used to store and rank the objectives. A comparison operator that violates linear ordering or changes over time can create problems in the following areas:

(a) losing data because access relies on a consistent sorted ordering

(b) maintaining the structure of the sorted objective tree

(c) the alternatives may be out of order, and the best may not be at the top of the list

(d) the search may fail to choose the best alternative at the end of the iteration

(e) the ordering of sorted data becomes stale once the comparison operator changes

Items (a) and (b) are unlikely to cause problems, since the collections are sorted with $\precsim_O$ which is likely to be linear. Still, programmers may want to display the ordering supplied by $\precsim_R$, which may be nonlinear. Problem (a) can be avoided if the lookups for individual scores are not performed using the comparison operator explicitly. Instead, a separate hash table is used to store the objective using the keys assigned by the solution archive. This method guarantees that we can locate any specific objective score, without depending on the order.

To address item (b), we use a red-black tree to store the sorted objectives. It is likely that a red-black tree maintains sufficient order even with a defective comparison operator, because it re-arranges nodes based on redness, blackness, and the existing structure of the tree with minimal use of the comparison operator (Cormen, Leiserson, Rivest, & Stein, 2001).

In regard to items (c) and (d), a tabu search will continue to function properly even if something other than the "best" neighbor is selected as the new incumbent solution. In fact, TS may frequently accept less than the best neighboring solution *on purpose* as a calculated gambit or as part of its diversification and navigation strategy. Thus, TS is resilient against intransitive orders, and has a demonstrated ability to recover from inefficient choices (Harwig et al., 2006; McKinzie & Barnes, 2006; Porter et al., 2006).

Item (e) relates primarily to rank reversal situations. Both $\precsim_O$ and $\precsim_R$ may be subject to rank reversal. From a programming standpoint, the primary difficulty with rank reversal is that the ordering defined by the comparator can change at any time, since new alternatives are constantly being discovered (and old ones removed). This can cause the sorted objective tree (managed by $\precsim_O$) to go "stale"; that is, the current ordering of the alternatives in memory may not reflect the true ordering. Fortunately, this does not break the tree. Insertions and removals can still be carried out even with an intransitive comparison operator, due to the way that this data is stored and accessed. Periodically, the tree can be re-sorted so that it reflects the new ordering.

Each objective keeps track of whether or not a re-sort is required. If necessary, re-sorts are performed before we purge the tree (so that we preserve the best solutions). Re-sorts may also be requested from the GUI panel at any time, when it is possible that rank reversal may have occurred.

MASTS is effectively immune from the effects of stale ordering because it does not use the sorted objective tree to make its decisions. Instead, it uses fresh pairwise comparisons with $\precsim_R$ to determine the new incumbent solution. When appropriate, the comparison operator is updated as the objectives are evaluated. *After* the objective scores are evaluated, the search iterates over the candidate solutions, using the pairwise comparison operator to select the best one. Thus, the search is making decisions based on the most current information about the ordering.

Finally, since the search does not rely on the sorted score tree, re-sorts are not necessary every time the ordering changes. In fact, they are only required before a purge or when a user requests the best solutions. Thus, minimal CPU time is spent combating the problem of stale ordering associated with rank reversal.

3.8 Penalties

There are two types of classical penalties commonly used with TS methods that deal with numeric objective scores:

- those that express a preference about properties of a solution, such as Langrangian penalties for constraint violations (Wilde & Beightler, 1967)

- those that manipulate the trajectory of the search, such as strategic oscillation (Glover & Laguna, 1997)

The first category is easily addressed by rule based objectives. MASTS allows the use of numeric Lagrangian penalties. However, MASTS' RBOs are more effective at dealing with such penalty structures because they allow greater precision when describing the allowable tradeoffs in marginally infeasible situations. For example, a company may be willing to accept a reduction in the number of service vehicles only if fewer than three customers will be inconvenienced and none of the other vehicles have to work more than 15 minutes extra. Writing this rule is straightforward; defining a classical numeric penalty term that accurately enforces this rule is much more difficult. The second category of penalty terms can be avoided by using more direct methods of controlling the search behavior, such as RBOs and dynamic neighborhood selection, including dynamic neighborhood selection. In particular, methods that *alter* the objective score to influence the search trajectory are easily modeled as a revealed preference (without interfering with the preferred outcome of the search).

3.9 Conclusions

Rule based objectives can replace numeric scores in a tabu search, using a binary comparison operator $\precsim_O$ to determine the overall rankings. The comparison operators used by RBOs allow users to specify precise ordinal rankings, particularly in a multi-criteria setting. In addition, it may be advantageous (although not required) to use a slightly different comparison operator $\precsim_R$ to decide how the search chooses the next incumbent solution. This dual preference structure is highly analogous to revealed preference theory.

Using a separate comparison operator $\precsim_R$ to govern the choice behavior of the search allows users to embed gambits which can improve search performance by simultaneously exploiting problem specific features and any inherent structure in the iterated sequence of decisions. These gambits could make $\precsim_R$ intransitive, an unusual prospect for a supposedly rational search procedure. Both empirical and theoretical results suggest that the intransitivity is defensible in this situation and does not necessarily prevent the search from finding globally optimal solutions. In fact, empirical results show that intransitivity actually enhances the search performance by evading non-productive sub-regions and superfluous local optima in the search space. Further research may find additional ways to take advantage of this link between metaheuristics and revealed preference theory.

3.10 Figures

Figure 3.1 A rule based objective may consist of two comparison operators which compare alternative solutions. The more familiar operator, $\precsim_O$, describes the overall objective of the search. Another comparison operator, $\precsim_R$, is used to make the decisions at each iteration in the search. This system is similar to the structure of revealed preference theory. Here, $\precsim_R$, represents the revealed preferences of the search agent.

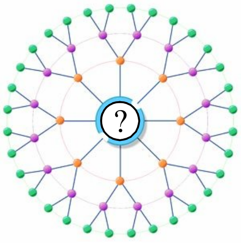
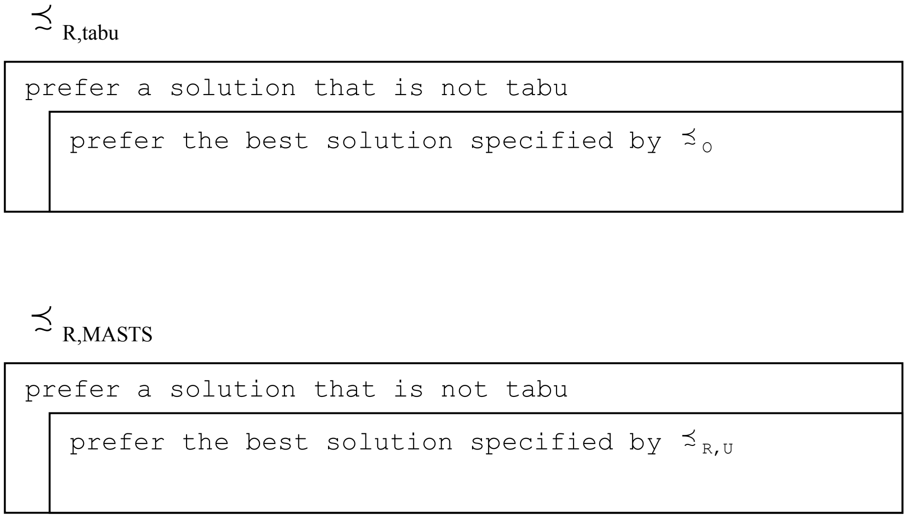
$\precsim_R$  The revealed preferences are used by search to make decisions at each iteration. Frequently, these preferences override the ordering specified by the outcome preference.	$\precsim_O$ The outcome preferences are used to rank and display the solutions. This ordering is consistent with the overall objective of the search, and the psychological preferences of the user.

Figure 3.2 The revealed preferences of a tabu search are typically built around the preferences of the overall objective $\precsim_O$ (*top*). In MASTS, users may supply an alternate definition of the revealed preferences $\precsim_{R,U}$, which will be used in lieu of $\precsim_O$ while making decisions for each iteration (*bottom*).



$\precsim_{R,tabu}$

prefer a solution that is not tabu

prefer the best solution specified by $\precsim_O$

$\precsim_{R,MASTS}$

prefer a solution that is not tabu

prefer the best solution specified by $\precsim_{R,U}$

4 Dynamic Neighborhood Selection

In TS, we are often faced with an immense number of available moves at each iteration. Dynamic neighborhood selection (DNS) can help identify the most promising subset of moves for the next iteration. In this research, a flexible new approach has been developed that incorporates various intensification/diversification strategies (IDS) which learn about the search landscape, focusing on information from the objective function f rather than specific details of the solution space, X. In many problems, it is both possible and highly advantageous to leverage domain specific knowledge to create an intuitive DNS strategy. Using the descriptive framework of landscape theory (Barnes et al., 2003; Dimova, Barnes, & Popova, 2005; Solomon, Barnes, Dokov, & Acevedo, 2003), my research has extended this approach to provide direction when the neighborhood structure for the problem is poorly understood. By defining the components of a generic DNS strategy, it is hoped that problem independent implementations can be developed for a variety of contexts.

4.1 Problem Definition and Background

At each iteration, MASTS examines a set of neighboring solutions, a *neighborhood*, around the current incumbent solution. A complicated optimization problem may involve many neighborhoods for different types of variables and moves, and deciding which neighborhood to use for the next iteration becomes a dominant consideration. DNS is the process of strategically choosing the "best" neighborhood for the next iteration.

While many problems implicitly require selecting a neighborhood at each iteration, little attention has been given to reactive strategies that improve search performance. Traditionally, neighborhood selection is used to restrict excessively large neighborhoods associated with exponentially large solution spaces. My previous and current research has shown conclusively that DNS can also improve search performance in two major areas:

- Aiding in strategic intensification and diversification (Harwig et al., 2006; Porter et al., 2006) and
- Preventing the search from being trapped in a basin of attraction (Battiti & Tecchiolli, 1994).

The generic neighborhood selection problem can be stated as follows. The search begins with a finite set of neighborhoods $\mathcal{N}$ = {N_1, N_2, ... , N_k}. The problem solution space may embrace several different types of decision variables. Commonly occurring variable types include discrete, boolean, and permutation variables. Neighborhoods may be simple or complex; a *simple neighborhood* generates moves for only one decision variable, a *complex neighborhood* could generate moves on two or more decision variables. One or more neighborhoods may act on each decision variable and may generate different types of moves.

At each iteration, the tabu search must select *one* neighborhood from $\mathcal{N}$. Depending on the incumbent solution x*, the selected neighborhood N_i, generates a set of neighbors about x*, denoted N_i(x*) = {x_1, x_2, …, x_p}. Some of these neighbors may be equivalent solutions (either precisely the same structure in X, or equivalent features on X, such as non-unique permutative elements). When evaluating neighbors, it is prudent to discard truly redundant equivalent solutions, evaluating only a single member of such an equivalent set. Thus, the *size of the neighborhood* (the number of unique solutions that require evaluation) may be less than the number of solutions actually generated.

Both $\mathcal{N}$ and its members N_i may be dynamically defined, i.e., new neighborhoods may be introduced at any time during the search, and neighborhoods may be dynamically redefined. Harnessing this capability into a coherent DNS strategy should only be considered after more fundamental strategies are in place.

4.1.1 Origins of the Problem

In most complex problems, we are faced with possible neighborhoods of overwhelming size. It is inefficient to define one large neighborhood that generates all such moves. If the search must examine a large neighborhood at each iteration, progress will be unnecessarily slow. A logical approach is to define several smaller neighborhoods and then select one of the neighborhoods at each iteration.

Classical tabu search (Glover & Laguna, 1997) has provisions for narrowing down large neighborhoods by considering smaller "candidate subsets" of the entire neighborhood. This process simply yields a new neighborhood of a smaller size. However, a strategic approach to neighborhood selection has not been adequately defined. The literature contains a large variety of different approaches to neighborhood reduction, but few that offer strategic insights. Duong and Dien (2003) state, " … since the size of the neighborhood increases rapidly with the problem size, it may become unreasonable to scan the whole neighborhood to identify the best neighboring solution". To deal with this, they use the large neighborhood but place an upper limit on the number of neighbors they investigate at each step. Another common approach is to randomly select a few elements. While these methods are possibly acceptable within the broadest context of TS, they have little strategic value.

4.1.2 Previous DNS Success

There exist successful DNS strategies that have improved the performance of a TS approach by exploiting the underlying structure of the problem to define intensification and diversification neighborhoods. Porter et al. (2006) use domain specific knowledge in a thermodynamic design problem to identify high influence moves, and use these moves to diversify the search (described in Section 4.2). For the bin packing problem, Harwig et al. (2006) define a more intricate web of conditions that examine the current solution and recent search trajectory to decide which moves to use next. Lambert et al. (2007) creates different search "phases" that use different neighborhoods. He strategically alternates between the phases, and uses dynamic strategies to engage in intensification, diversification, and "superdiversification." While these strategies are problem dependent and widely varied, they may provide insight into a unified framework.

4.1.3 Variable Neighborhood Search (VNS)

Variable neighborhood search (VNS) is a metaheuristic technique that incorporates a systematic method to explore and change neighborhoods (P. Hansen & Mladenovic, 2003). Conceptually, the goals of neighborhood selection in VNS and DNS are the same, although there are major differences in the methodology employed in the two techniques. First, the rules for neighborhood selection in VNS are either static or incorporate randomization. The dynamic DNS strategies proposed below place more emphasis on learning about the neighborhoods and making informed decisions about which one to use next. Another major difference is that VNS does not employ the sophisticated memory and decision structures prevalent in advanced TS approaches, which limits VNS' ability to guide the search as effectively. Most variants on the basic VNS strategy reject non-improving moves and automatically take the best neighboring solution as the new search incumbent (steepest descent) while relying on random moves or search restarts to escape local optima (P. Hansen & Mladenovic, 2003). Since basic VNS is ineffective, increasingly complicated variants have been developed in an attempt to overcome the shortcomings of basic VNS.

Both MASTS and VNS are capable of using multiple neighborhoods. Individuals looking to VNS for insights on intelligent ways to select neighborhoods will be disappointed. Unfortunately, the majority of VNS strategies surveyed in (P. Hansen & Mladenovic, 2003) do not use dynamic rules to select neighborhoods, opting instead to cycle through the neighborhoods or choose one at random. However, Hansen et al. (2004) augmented basic VNS with a strategic neighborhood selection for solving the maximum clique problem. They describe three types of neighborhoods which involve adding, dropping, or interchanging nodes in the graph and specify that the add neighborhoods should be used for solution construction, the drop neighborhood should be used for "shaking" (escape from local minima), and the interchange neighborhood should be used if the incumbent solution has a cardinality that matches the best known solution.

Braeysy (2003) builds a multi-stage heuristic that combines a VNS approach with dynamic neighborhoods. As the search cycles through four custom neighborhoods, the neighborhood size gradually increases, allowing the search to explore with greater depth. This strategy is a type of neighborhood selection that we will later classify as a neighborhood update.

Unfortunately, VNS does not embody intelligent ways to select or organize neighborhoods, and provides no guidance for those who wish to add this capability; it was simply not the focus of the initial algorithm. VNS is therefore fundamentally different from DNS.

4.1.4 Landscape Theory

Landscape theory (Barnes et al., 2003) is a sophisticated way to view the interaction between the solution space, *X*, a neighborhood (or neighborhood structure) N, and the objective function, *f*, often denoted by the landscape triplet, $\mathcal{L}$ (*X*, *f*, N). Each element of $\mathcal{L}$ is essential to the understanding of the behavior of the search. Naturally, we prefer landscapes where the interaction between *X*, *f*, and $\mathcal{N}$ is predictable. I will call this type of landscape *harmonious*, because *X*, *f*, and $\mathcal{N}$ work together in a way that we can understand and anticipate. The intuitive simplicity of a harmonious landscape can be exploited to create effective DNS strategies. Experienced search architects often leverage problem specific knowledge when designing $\mathcal{N}$ and *f* to create harmonious landscapes.

Unfortunately, it is not always possible to create a harmonious landscape. Perhaps *f* behaves erratically or there is no simple way to organize neighborhoods so that they cooperate with *f*. In these *inharmonious* landscapes, our perceptions about how the search behaves in different regions of *X* are either absent or insufficient, making it difficult to create an effective DNS strategy. Landscape theory may suggest different ways to classify landscape behavior and analyze neighborhood performance. Since some or all aspects of $\mathcal{L}$ change from problem to problem, a successful general DNS strategy must be cognizant of the different types of landscapes that may be encountered. The work presented here builds on the powerful descriptive framework provided by landscape theory, examining the implications that different types of landscapes can have on general search strategies.

First, we define some descriptive terms that generalize the specific landscape definitions presented by Barnes et al (2003). A *smooth landscape* can be thought of as a rolling surface where the local minima occur at the bottom of large basins, and a neighborhood around a good solution often contains other good solutions. The neighborhoods that characterize this landscape contain solutions of similar quality. In contrast, a *rugged landscape* is one where the neighbors of good solutions are arbitrarily bad, and poor solutions are directly surrounded by above average solutions. Often, a local minima will be surrounded by worse than average solutions, creating numerous, deep point-like depressions. A *mixed landscape* may contain regions in *X* that behave like a smooth landscape and others that behave like a rugged landscape.

Barnes et al (2003) provide more rigorous analysis by examining *elementary landscapes* with a single neighborhood*,* in which the digraph associated with the neighborhood satisfies Grover's wave equation (Grover, 1992). They discover two fundamental types of elementary landscapes which have surprising properties, placing bounds on the *average* performance of a the neighborhood. In a *smooth-elementary* landscape, we can expect that a neighborhood around a good solution will contain (on average) other good solutions (where good is defined as superior to the global average). In a *rugged-elementary* landscape, neighborhoods around good solutions necessarily contain solutions that are inferior to the global average. This type of information can be leveraged to guide the construction of $\boldsymbol{\mathcal{N}}$ and the associated DNS strategy.

Since most practical problems have exponentially large solution spaces, the early work on landscape theory provided no constructive methods for establishing whether a landscape is elementary. Fortunately, Dimova et al. (2005) established a link between arbitrary elementary landscapes and autoregressive processes of order one. A landscape will be elementary if and only if the autocorrelation function generated by a random walk on $\boldsymbol{\mathcal{L}}$ is consistent with an AR(1) time series. Empirically, we can test whether a landscape is elementary using a Box-Jenkins analysis (Box, Jenkins, & Reinsel, 1994).

For an inharmonious landscape, we are uncertain whether the landscape will be smooth, rugged, or mixed. This research indicates that a DNS strategy designed for this general scenario should be able to categorize the landscape type and respond accordingly.

4.2 Different Perspectives on IDS

DNS is one way to implement an intensification diversification strategy (IDS). In the commonly accepted definitions, search intensification refers to any procedure used to explore a local search space region in detail. Conversely, diversification involves escaping to explore different regions and enhance a global perspective. In this sense, intensification and diversification rely on the properties of the solution space *X*, specifically the notion of distance. My research has resulted in new developments in dynamic TS which suggest an alternate but intuitive definition that includes the objective function. The following sections clarify the characteristics that separate this alternate definition from standard IDS and compare these two different perspectives on IDS, exploring the similarities, assumptions, and consequences behind each. By considering the objective function as a critical component of intensification and diversification, we can create more general IDS strategies that can be applied to a wide variety of solution spaces and landscapes.

IDS strategies in TS can be implemented by judiciously alternating between intensifying and diversifying moves. In the standard approach to IDS, an intensifying move is synonymous with a relatively small change in X, while diversifying moves involve larger changes. Even if *X* is not metrizable, we can usually characterize "large" and "small" moves associated with the decision variables, i.e., the components of *X*. A small move remains "close" to the incumbent solution, while large moves travel to relatively distant locations in the search space. This weaker concept of distance is referred to as a pseudometric or semimetric (depending on which properties of a metric are not present).

IDS strategies that intensify using small moves and diversify using large moves are predicated on the assumption of a smooth landscape. In this situation, the search uses small moves to thoroughly explore a local minimum, and larger moves to travel to another basin, escape a chaotic attractor basin, or otherwise "shakeup" the solution structure. Henceforth, we refer to this strategy as XIDS to emphasize that it depends on some measure of distance in *X*.

The logic embedded in XIDS may not work for all problems. In a rugged landscape, the local minima may be isolated in countless point-like depressions, which precludes intensification within a local region of the space. Relatively small moves could create large changes in the objective function. This raises serious questions about the efficacy of using a measure of distance on *X* to control the intensification and diversification properties of the search. In a non-elementary rugged landscape, small and large moves in *X* may have seemingly unexpected or unpredictable influence on the objective function, which can cause an XIDS strategy to be ineffective.

Another approach to IDS is to consider the objective function as a direct measure of intensifying and diversifying moves. Porter et al. (2006) demonstrated that some engineers have intuitively adopted this alternate definition, and applied it to create a successful IDS strategy. The goal was to select optimal placement of heat lamps to achieve a desired temperature profile on a metal sheet. To intensify and diversify the search, they utilize low influence and high influence moves.

> "The coarse neighborhood identifies moves of *high influence*. When performed, high influence moves have a strong relative effect upon the objective function value. The heaters most likely to strongly influence the design surface element of interest were selected first."

In addition, they use a "fine neighborhood" which generates low influence moves, further refining and improving a particular solution. These low influence moves involved the lamps that were least likely to have a large effect on the objective function.

What if we formally extend this concept, treating it as a new framework for IDS? In a fIDS (objective function intensification diversification strategy) we re-define the concepts of intensification and diversification to align with the objective *f* rather than the solution space *X*. Many IDS strategies are predicated on this logic; but they do not explicitly highlight the significance of this modified approach. This failure to differentiate has created some ambiguity in the terminology, so I will introduce and define new terms when necessary.

In the fIDS framework, moves that create a relatively small change in the objective function score (as defined in Section 3.6) are classified as *f-intensifying moves*. Likewise, moves that create a large change in the objective function can be considered *f-diversifying moves*. With this definition, it is impossible to classify a move as intensifying or diversifying until *after* it has been evaluated. Thus, implementing fIDS requires a predictive mechanism to make informed guesses about how the moves will impact the objective score. In the heat transfer problem described above, the authors used engineering system knowledge as their predictive mechanism (Porter et al., 2006). They identify a *fixed* set of the most influential heaters to use as high influence moves.

If the design components within the problem had been slightly less predictable, the strategy would not have been so easily formulated. Suppose, for instance, that we did not know the thermal properties of the design surface. In this case, we cannot identify precisely which heaters will have the most effect on the objective score. It may not always be possible to predict which moves will be f-intensifying moves or f-diversifying moves.

Nevertheless, fIDS requires some predictive mechanism when deciding how to intensify or diversify. To provide this insight in a general setting, a logical approach is to collect statistics for each neighborhood as moves are generated and evaluated. As the search proceeds, some neighborhoods will emerge as strong *f-intensifying neighborhoods* and some will emerge as strong *f-diversifying neighborhoods* (and others will lie somewhere in between). The search can use this data to roughly predict how different moves and different neighborhoods may impact the objective. The terminology for the different IDS approaches is clarified in Table 4.1.

Table 4.1 The terminology and differences between the two IDS strategies are detailed in this table.

XIDS	X-intensifying move	moves a small distance within the solution space X
	X-diversifying move	moves a large distance within the solution space X, shakes up the structure of the solution
	X-intensifying neighborhood	generates small moves in X
	X-diversifying neighborhood	generates large moves in X with the intent of shaking up the structure of the solution
fIDS	f-intensifying move	has only a small impact on the objective function f (ranking or score)
	f-diversifying move	has a large impact on the objective function f (ranking or score)
	f-intensifying neighborhood	a neighborhood that is expected to generate moves that have a small impact on the objective function f (ranking or score)
	f-diversifying neighborhood	a neighborhood that is expected to generate moves that have a large impact on the objective function f

Clearly, the adaptive approach embodied by fIDS is more difficult to implement than XIDS, but the potential advantages are compelling for large practical problems that embody non-intuitive landscape structures. For easy problems with smooth landscapes or intuitive elements, we should use the simplest appropriate IDS strategy. To conclude this section, we discuss the similarities and differences between fIDS and XIDS, and describe the features of fIDS that make it an attractive strategy for a generalized search program.

Parallels

This reasoning behind fIDS is compatible and parallel with XIDS. Suppose the landscape is smooth and the search must intensify to explore a local minimum. XIDS would choose an X-intensifying neighborhood that generates solutions close to the incumbent in X. XIDS implicitly expects that such solutions will have similar and hopefully higher quality (as measured by f). When fIDS intensifies, it will also choose a neighborhood that is expected to yield solutions similar in their values of f. The only difference is that fIDS adaptively decides which neighborhoods might have this property. In a smooth elementary landscape, fIDS and XIDS would likely select the same neighborhoods to be used for intensification.

Departures

The advantage of fIDS becomes apparent in a mixed or rugged landscape, where XIDS breaks down. Suppose the search has located an interesting local optimum, prompting intensification. While XIDS looks for marginal improvements at solutions proximal in X (a useless approach in a rugged landscape), fIDS uses historical data to propose moves that maintain the solution quality. These may yield neighbors with similar structure to the current elite solution, or neighbors with markedly different solution structures. fIDS intensification is driven only by the supposition that the objective score will be similar to the current elite solution. Given a proper neighborhood definition, fIDS has the potential to discover and connect distant portions of the search space based on objective values. The trajectory of such a search in X could appear pathological to those who insist that an organized search must explore local regions and then move on to distant places. Unfortunately, this entrenched logic simply fails in the presence of a rugged landscape. fIDS organizes moves and neighborhoods by their performance in f, rather than their locality in X.

As a result, diversification is conceptually different within fIDS, also raising some concerns for those who think of diversification only as a property within X. One common concern is since fIDS focuses on the objective function and ignores structure on X, it may not visit sufficiently different regions of X to ensure a diverse sampling of the space. This concern is at least partially allayed by two facts. First, fIDS is not independent of the solution space; it incorporates information about X indirectly through the neighborhood structure. As long as some neighborhoods are capable of making large changes within X, then a properly designed fIDS strategy will achieve diverse movement within X. Second, solution diversity is also largely controlled by the tabu memory structures, which prohibit quick returns to solutions with attributes that have been recently observed.

Advantages

The fIDS strategy was developed to complement the generic MASTS framework and is sufficiently general to apply to a wide variety of problems with different structures, providing a significant advantage over XIDS. First, we note that fIDS does not depend directly on the distance measure or other properties within X. This allows a mixture of different types of decision variables, without having to categorize moves based on their size in X. Suppose a problem has discrete variables and an ordering sequence that must be jointly optimized. Without more information, it is impossible to decide which is better for intensification: an adjacent insert move on the sequence, or a small adjustment to a discrete variable. The best way to differentiate between these moves is by their impact on the objective score. In this respect, the objective function serves as the common measure of the intensification level for the different move types that may arise from a complicated solution space. Since it works exclusively with the objective function, fIDS does not require any specific details about the structure of X (other than the neighborhood definition). This independence from X allows fIDS to generalize to any problem.

Also, the adaptive learning and neighborhood selection in fIDS is essential when we do not have foreknowledge of the objective function. In MASTS, users are able to dynamically construct their multi-criteria objective functions during the search performance. There may be two users who are optimizing for completely different criteria. In each case, the neighborhoods used for intensification and diversification will be different; the search must identify which neighborhoods complement each objective function. This adaptive approach allows DNS to succeed in an environment where the objective functions are dynamically defined. MASTS allows users to dynamically create new objectives which reflect the user's increased knowledge of the problem and change of preferences during a particular execution of the search.

Finally, fIDS requires fewer assumptions about the landscape of the problem, and this strategy can work even when it is not manifestly clear how the neighborhoods interact with the objective function; for example, when dealing with a mixture of smooth and rugged landscapes. By monitoring the evolving properties of each neighborhood, it is possible to identify which neighborhoods will be most suitable for intensification and diversification. In essence, the goal is to transform a landscape of unknown structure and character into a smoother incarnation through dynamic neighborhood selection.

4.3 Components of a DNS Strategy

A good DNS should improve search performance in the following areas:

- **Avoid basins of attraction.** Battiti and Tecchiolli (1994) concluded that hashing was an efficient strategy for detecting both cycling and chaotic attractors. This reaction strategy can be implmeneted in DNS.
- **Implement an intensification/diversification strategy (IDS).** DNS provides an extensible medium for the more general ideas contained in the intensification/diversification strategy. Through proper design, the many advantages of fIDS can be transferred into the neighborhood selection strategy.
- **Save time with appropriate neighborhood size.** With an effective DNS, it becomes possible to more confidently rely on smaller neighborhoods. Small neighborhoods can save time because they require fewer function evaluations. Even with small neighborhoods, it is possible to exploit structure in the problem that can accelerate the search process.

The rest of this section presents the basic components of a general DNS methodology based on the principles of fIDS. Figure 4.1 shows the conceptual model which forms the strategic building blocks of DNS. This model provides a common organizational framework both for currently used strategies and for more sophisticated strategies yet to be developed. There are many ways to implement each component in the overall DNS strategy, but common patterns tend to re-appear in problem specific implementations. While general strategies are discussed, no attempt is made to develop a DNS strategy that will be effective works for all problem types. Rather, the goal is to provide an abstract foundation for complex strategies that can be supported by MASTS software. A proof of concept is demonstrated by the DNS used in the conservation planning problem (see Section 6.5).

4.3.1 Neighborhood Definition (DEF)

No matter how sophisticated the algorithm, DNS requires neighborhoods that allow intelligent behavior to emerge. As indicated in Figure 4.1, this is where the user's intuition and domain knowledge is integrated into the search process. The user does not have to predict the exact behavior of each neighborhood, but the neighborhoods should be distinctly organized in ways that will be detectable by the objective function. Ideally, the user should take advantage of special structures on X. In addition, users should try to break down large neighborhoods into smaller logical neighborhoods. We refer to this strategy as *neighborhood decomposition*.

In the conservation area network design problem, the decision variable is a vector of bits, indicating which cells are selected for inclusion in the reserve network. A neighborhood that flips each bit could be prohibitively large, so we use a neighborhood decomposition. The cells can be grouped into subsets based on several criteria such as relative spatial location (i.e., clustering and connectivity), the distribution of species, or secondary costs/benefits associated with each cell. Neighborhoods defined in this fashion will have identifiably different behaviors at different stages of the search. An intelligent DNS strategy will be able to capitalize on these differences.

MASTS provides standard neighborhood definitions for all decision variables. Users may expand or alter these definitions to accommodate particular problem specific structures.

4.3.2 Search Mode Selection (SMS)

SMS does not choose the neighborhoods; rather, it tries to capture the search's preferred mode of intensification or diversification. The term "search mode" is intended to represent an ensemble of different IDS behaviors; it is *not* simply intensify or diversify. For example, there could be a spectrum of different intensification levels. There may also be special modes that are invoked in specific circumstances. The primary responsibility of the SMS is to monitor search progress and provide information about which mode might be beneficial. It is up to the NCS to translate this information into the neighborhood that is most likely to satisfy this mode. In that respect, the SMS and NCS must be integrated to function smoothly together.

Simple problems do not require a complicated SMS. Consider a problem that requires only two neighborhoods, one for normal operation and one that is used to escape from the current solution space context. One simple SMS approach is to trigger the escape neighborhood after a specified number of iterations has passed without finding a new best solution.

A more extensive SMS system will be justified when the number of neighborhoods is *large*. In such a complicated problem, the search will very likely have several different desired modes of operation. *General modes* will apply to the normal operation of the search, making minor changes to the intensification and diversification levels using *general use* neighborhoods. S*pecific modes* will apply to specific circumstances, playing a role in escape or other special maneuvers that require a *specific use* neighborhood. A strong programmatic framework will be required to coordinate these actions, and developing this framework will require varying amounts of experimentation.

In general, the SMS monitors various aspects of the search progress to decide which mode might be most appropriate. Figure 4.2 presents a prototype SMS where the general modes are represented by an incremental slider, with intensification and diversification are located at opposite extremes. The idsMode variable holds this information. After each iteration, the SMS would review the search progress to decide whether to increase or decrease the idsMode, or perhaps to invoke a specific search mode. An example of a specific mode is *stagnation*, triggered by an unacceptable number of search iterations without finding a new best solution. In addition to problem specific information, the following aspects of search performance may be useful in determining the search mode:

- the number (or proportion) of solutions in the previous neighborhood that have already been visited by the search (as incumbent solutions)
- whether or not the most recent incumbent solution has previously been visited by the search
- the number of consecutive improving or non-improving moves
- the number of consecutive moves without a new best solution
- the discovery of a new best known solution

4.3.3 Neighborhood Performance Assessment (NPA)

When we do not know which neighborhoods are suitable for intensification or diversification, we must collect data about each neighborhood to assist in this decision. Neighborhood performance assessment (NPA) refers to the process of analyzing the moves and solutions for each neighborhood after it has been evaluated, and maintaining a key set of statistics that will help in the NCS process. This data collection and analysis represents one of the self-learning components of MASTS, and the first decision that must be made is what types of data should be collected.

According to fIDS strategy, we must examine the neighborhood's impact on the objective score in order to decide whether the neighborhood is f-intensifying or f-diversifying. The objective score is the common measure of performance between all moves and neighborhoods and serves well as the basis of comparison in an IDS strategy.

Since NPA requires information from the objective function, the assessment must occur *after* the solutions have been evaluated. When the search finishes an iteration using a single neighborhood, the proper statistics are gathered (specific to the current objective). The following list contains several statistics that may be considered. The values will change as the search visits different regions of the solution space. For each item, we may consider such things as the most recent value, a historical running average (over recent iterations), or the min and max values.

The first two items below are appropriate when a real-valued univariate objective function is available. The other measures are appropriate to the *score* context provided by rule based objectives.

Average Neighborhood Move Value. This score attempts to characterize the quality of the solutions in the neighborhood compared to the incumbent x*. The *move value* from incumbent x* to a new solution x is:

$$d(f, x^*, x) = f(x) - f(x^*)$$

Suppose a negative move value indicates an improving move. For a specific incumbent solution, x*, with *p* neighbors (denoted $x_1 \ldots x_p$), the *average neighborhood move value* is defined as:

$$\Delta_N(f,x^*) = \frac{1}{p}\sum_{i=1}^{p}\left[f(x_i) - f(x^*)\right] = \left[\frac{1}{p}\sum_{i=1}^{p} f(x_i)\right] - f(x^*)$$

A negative score for Δ_N indicates that the neighborhood has produced solutions that are on average better than the incumbent. Given a choice between two neighborhoods identical in other respects, we might choose the one that has shown it can find better solutions.

Average Neighborhood Impact. This score characterizes how closely distributed the neighboring solutions are to the incumbent solution (in terms of objective score). This measure is a sample estimate of the variance about $f(x^*)$:

$$\sigma_N^2 = \frac{1}{p}\sum_{i=1}^{p}\left(f(x_i) - f(x^*)\right)^2$$

If the *average neighborhood impact* (σ_N) is large, then the neighborhood may be suitable for diversification according to the fIDS strategy. Neighborhoods with small σ_N are suitable for intensification.

Proportion of Improving, Non-Improving, and Equivalent Moves. When using a RBO, a comparison operator $\preceq$ specifies whether the first alternative is better than ($\prec$), worse than ($\succ$), or equivalent (=) to the second alternative. During a single iteration, we can count the number of *improving* moves ($x_i \prec x^*$), *non-improving* moves ($x_i \succ x^*$), and *equivalent* moves ($x_i = x^*$) generated by the neighborhood.

$$\text{nImproving} = \#\{x \in N(x^*) \mid x < x^*\}$$
$$\text{nNonImproving} = \#\{x \in N(x^*) \mid x > x^*\}$$
$$\text{nEquivalent} = \#\{x \in N(x^*) \mid x = x^*\}$$

Given a choice between two neighborhoods identical in other respects, we prefer the neighborhood that exhibited more improving moves.

Long Term Performance. Count the number of times that this neighborhood (when selected) has returned an improving move, a non-improving move, or an equivalent move.

Number of Moves. The number of moves generated by a neighborhood can factor into its intensification/diversification properties. The expectation is that a neighborhood with more moves has greater foresight when selecting the next best solution.

Most Recent Use. It is necessary to consider recency in any DNS strategy with a large number of neighborhoods. This ensures that every neighborhood gets a turn so that the space is adequately sampled. As part of the NPA, we should keep track of the last iteration that each neighborhood was used.

Total Usage. It may be useful to count the number of times the neighborhood has been used. If nothing else, this information can be used to analyze the DNS.

Runs and Droughts. A *run* is the number of recent consecutive turns that the neighborhood (when selected) has returned an improving move. A drought is the number of recent consecutive turns that the neighborhood (when selected) has returned a non-improving or equivalent move.

Runs should be treated with caution in terms of strategic value. Just because a neighborhood is making continual improvements, this doesn't exclude the possibility that better gains can be found with another neighborhood. Many VNS strategies will stick with a single neighborhood as long as it returns improving solutions (until it reaches a local minimum). This myopic strategy performs abysmally in the conservation network problem, which indicates that it may be of limited strategic value.

After the information is collected by the NPA, the next step is to interpret the nature and behavior of the neighborhood and decide when to use it.

4.3.4 Neighborhood Classification and Selection (NCS)

Neighborhood classification and selection (NCS), at the heart of the DNS strategy, must synthesize information from SMS and NPA to decide which neighborhood to select at the next iteration. The SMS provides guidance about whether the search should intensify or diversify. In addition, the NCS has access to a variety of neighborhood performance statistics that have been collected. Since the number of potential strategies is limitless, we will consider an approach that is a suitable generalization for the types of strategies that are commonly employed.

The first step is to assign each neighborhood a *diversity score*, **d**. This score, based on the data from the NPA, should loosely reflect which neighborhoods are suitable for intensification and diversification. Again, specific problem domain knowledge can replace the role of NPA for simple problems. Just as fIDS prescribes, neighborhoods with a large impact on the objective function should be considered diversifying, those with a small impact will be intensifying. This is a first pass at the neighborhood selection.

The next step is to consider the recommendation from the SMS, which describes the preferred idsMode of the search. In Figure 4.3, the idsMode is 1. This gives an approximate estimate of which neighborhoods to use (the ones closer to the intensification end of the spectrum). From this subset of neighborhoods, we must select one based on secondary criteria. For example, choose the neighborhood that has not been used recently, or choose the neighborhood that provided high quality solutions the last time it was used.

Unlike the fairly static rules in VNS that tend to focus on one neighborhood at a time, this approach dynamically controls neighborhood use. Moreover, with three reactive rule sets controlling the action (from the SMS, the NPA, and the NCS) the search is unlikely to repeat its behavior.

This NCS strategy is clearly meant for a general situation with many neighborhoods. Whenever possible, simpler strategies are certainly preferred. In Porter et al. (2006), the authors define essentially two modes of search behavior; one mode for intensification and the other mode for diversification. The trigger for alternating between these modes was based on the number of non-improving moves. This basic rule set formed their SMS strategy. They did not require an NPA strategy because they already understood the essential behavior of the neighborhoods. When it was time to select a neighborhood (NCS), the decision was based entirely on the search mode. If the search was intensifying, they would use the fine neighborhood. If it was diversifying they would use the coarse neighborhood. Because there were only two neighborhoods and two search modes, the DNS process is greatly simplified, but still fits into this general framework.

A more adaptive approach, which also fits this generic framework, was used to organize several neighborhoods for the conservation planning problem (see Section 6.5).

4.3.5 Neighborhood Update and Restructuring (NUR)

It may be beneficial in some DNS implementations to make slight changes to existing neighborhoods (updates), or create new neighborhoods by recombining the current neighborhoods (restructuring). For example, suppose a swap neighborhood in the traveling salesman problem generates all the swap moves between cities that are within 3 positions in the current sequence. If the search stops finding new solutions, we might consider looking at swaps within 4 positions. A similar approach is used to change the radius of a Euclidean neighborhood in (Kovacevic-Vujcic, Cangalovic, Asic, Ivanovic, & Drazic, 1999). During a neighborhood update, a parameter defining the extent of the neighborhood may change, but the fundamental structure of the neighborhood remains unchanged.

Restructuring is a process by which new neighborhoods can be created by combining currently existing neighborhoods. For instance, neighborhoods can be joined together in a *union* or *composite*. The resulting neighborhood will produce all of the moves from the member neighborhoods. A union of this type increases the number of moves that will be considered during the search iteration, providing more opportunities to find improving moves, at the cost of more computational effort.

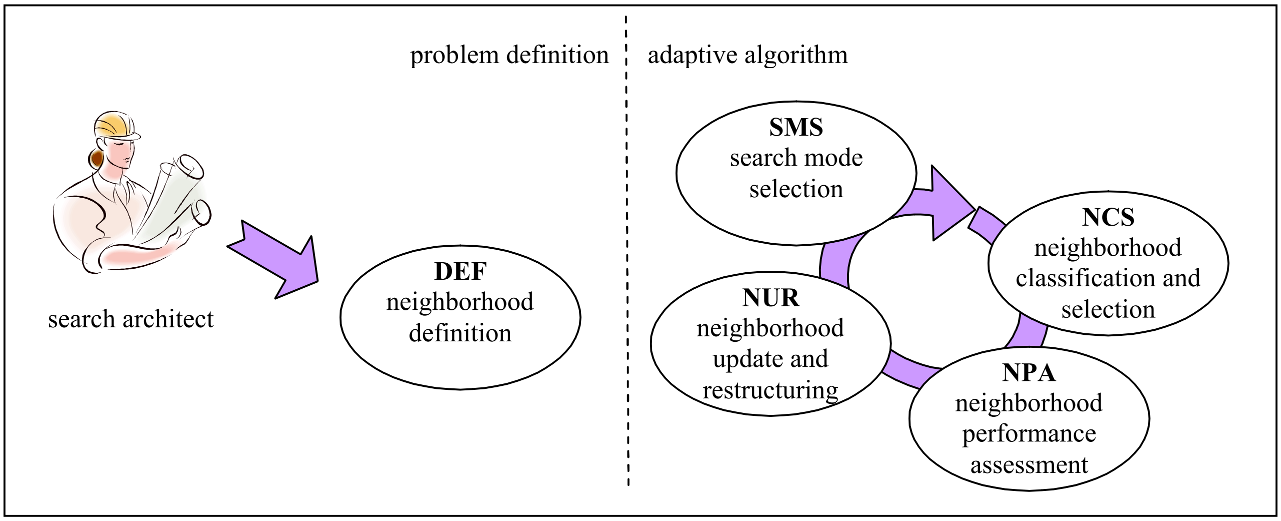
4.4 Conclusions

Elements of DNS appear in numerous studies because the concept is an intuitive and effective strategy for guiding the intensification and diversification of a search. A versatile IDS strategy must consider the objective function f (and not just the solution space X) to gain better understanding of the search landscape $\mathcal{L}\,(X, f, \mathrm{N})$. Neighborhood selection strategies can be unified into a common descriptive framework, which is the first step towards reusable software solutions. DNS greatly improves the search performance in

ConsNet, as discussed in Section 6.5, allowing the search to reach solutions that are not available with simpler techniques or random neighborhood selection.

4.5 Figures

Figure 4.1 The components of a dynamic neighborhood selection (DNS) algorithm.



DEF neighborhood definition	The search architect provides a neighborhood definition that incorporates intuition about how moves can be organized to facilitate an intensification/diversification strategy.
SMS search mode selection	The search decides based on an ensemble of dynamic contextual rules whether it should intensify or diversify.
NCS neighborhood classification and selection	The search must select a neighborhood appropriate to the *search mode*. To this end, neighborhoods must be classified as intensifying or diversifying (or somewhere in between).
NPA neighborhood performance assessment	In order to classify neighborhoods as intensifying or diversifying, we must collect information about the results when each neighborhood is used.
NUR neighborhood update and restructuring	Dynamically defined neighborhoods will change as the search proceeds. This stage updates those neighborhoods and gives an opportunity to restructure existing neighborhoods

Figure 4.2 The modes proposed for a generalized search mode selector (SMS) algorithm.

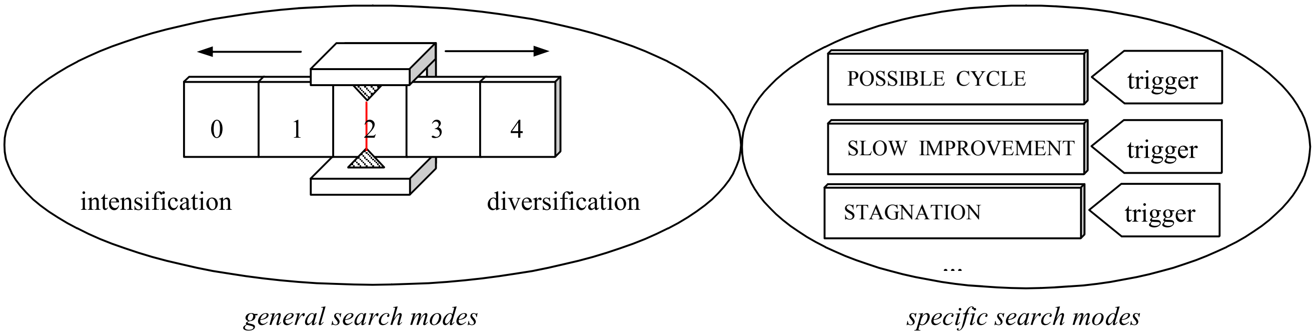
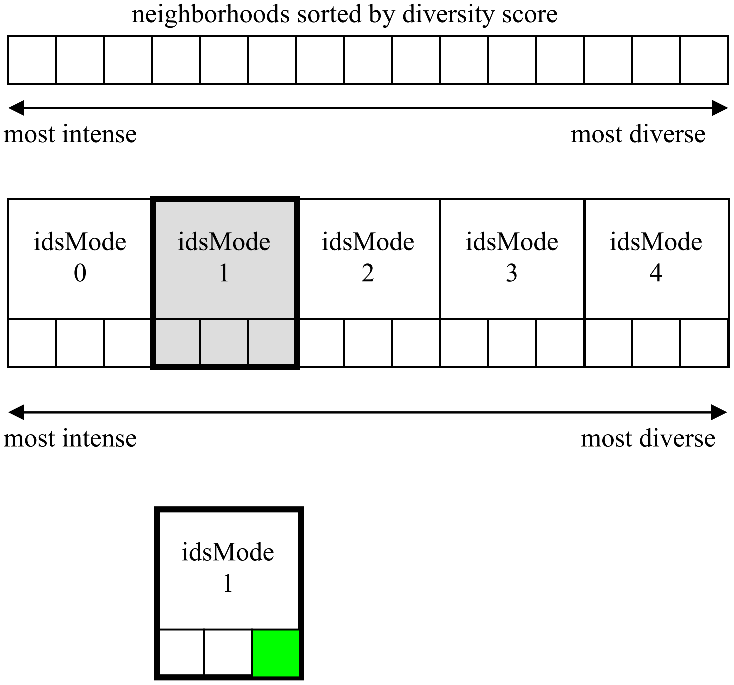


Figure 4.3 A general outline for a NCS of a neighborhood classification and selection.



Step 1. Sort the neighborhoods using an informed guess or data collected about their intensification-diversification properties.

Step 2. The SMS provides guidance about whether the search should intensify or diversify (the idsMode). This allows us to identify a subset of neighborhoods that we might choose from.

Step 3. Use secondary criteria to select the most desirable neighborhood from the subset.

5 Groundwater Planning

5.1 Introduction: Groundwater Management in Texas

Groundwater resources will become increasingly stressed as water demand continues to grow throughout Texas. Recognizing the need for localized planning, the state legislature passed House Bill 1763 (HB1763) in September 2005, overhauling the procedures for statewide groundwater planning, placing greater control in the hands of regional planning units called Groundwater Management Areas (GMA). Each GMA is composed of one or more Groundwater Conservation Districts (GCD), which *must* work together to propose "desired future conditions" (DFC) that describe a specific *management goal* for the future state of local groundwater resources (such as water level, flow, quality, or volume). The Texas Water Development Board (TWDB) then uses the best available state approved models to determine the managed available groundwater (MAG), i.e., the volume of groundwater that can developed without violating the desired future conditions (Mace, Chowdhury, Anaya, & Way, 2001).

This new process for the certification of management plans shifts major responsibilities to the local managers of the GCDs and GMAs. In addition to surveying the local water usage, projections, and availability, managers must now work closely with other districts in their GMA to set a DFC. Before settling on a plan, the GMA must be reasonably certain that their DFCs are:

- physically consistent among the GCDs within the management area
- physically consistent with their estimated water demands, and
- achievable within the models used for certification by the TWDB.

The model used by the TWDB is crucial to the evaluation of the water management plan. Thus, the GMA has a vested interest in obtaining and understanding this model. While the TWDB can *assist* with this analysis, supplying training and performing individual model runs, the GMA will inevitably need to perform a more rigorous analysis to identify consistent DFCs and management plans. An operational guideline is that each GMA should be reasonably certain that the plan will be accepted *before* submitting it. Rejections can create lengthy delays; the TWDB recommends submission before December 2007 so that the updated information can appear in the next Regional Water Plan (2011) (Petrossian, 2007). State law requires submission by September 1, 2010.

This new procedure is plagued by a seemingly insurmountable information gap, a lack of usable scientific tools, and stakeholder conflicts. This situation creates near stagnation in the planning process. Some of the major obstacles include:

- Many local groundwater planners are unfamiliar with the TWDB models, or disagree with the construction of the model.
- Groundwater managers create plans based on estimated groundwater needs, but have trouble relating this to a desired future condition. Suitable tools to illustrate this connection do not exist.
- The DFCs are often inconsistent with actual water demands in the area; there is no simple scientific tool to check whether a specific DFC is feasible, or to search for a feasible DFC.
- The Groundwater Availability Models (GAMs) mandated by law do not always address stakeholder concerns.
- Competing stakeholder interests can create an impasse.
- The TWDB can verify whether a particular management policy will satisfy the DFC, but the process to *find* policies that meet these conditions is still manual, and does not consider optimizing ancillary goals.

The above observations come from my active participation in two landmark projects in Texas; one involving a portion of GMA10 (Pierce, 2006) and one for GMA9 (Eaton et al., 2007). The regions of study are pictured on the map presented in Figure 5.1 and Figure 5.2. Working directly with stakeholders and managers, these interdisciplinary studies have shown that significant progress can be made by implementing a decision support system (DSS). The software behind such a system is an interactive forum that converts large volumes of raw model data into a more a succinct and useful representation, creating discussion and understanding (Kersten, Mikolajuk, & Yeh, 2000). In both studies, it was found that the effectiveness of a DSS can be greatly enhanced by linking the model to an optimization engine.

Optimization provides new insights that may not be discovered with manual interaction. More importantly, optimization can suggest feasible or near-feasible alternatives that satisfy the often conflicting targets and goals proposed by stakeholders.

MASTS was used as the optimization engine in both programs because of its flexibility to address different types of decision problems. The concerns and model parameters in the Barton Springs segment of the Edwards Aquifer (GMA10) are quite different from the more rural Hill Country portion of the Trinity Aquifer (GMA9). In both projects, MASTS has demonstrated that it is an indispensable addition to a DSS. MASTS handles multiple criteria and multiple objectives with ease, allowing users to (1) design their own searches, (2) analyze the collective results, and (3) build portfolios of preferred solutions. The solution archive saves time by storing thousands of model runs, which can be quickly re-assessed as new objectives are established.

Planning for the groundwater components of MASTS began in May 2005, and the design was guided by the needs of stakeholders, managers, and analysts. By Fall 2006, with programming assistance provided by Will Cain, MASTS was integrated into the GWDSS to assist Suzanne Pierce with her work for the Barton Springs/Edwards Aquifer Conservation District (BSEACD) in GMA10.

A second project also began during Fall 2006, when a Policy Research Project with the LBJ School of Public Affairs began open collaboration with managers and stakeholders in GMA9. Students in this PRP conducted numerous interviews with local water users, observed and presented at public meetings, and collected data to help GMA9 identify appropriate DFCs. The modeling team consisted of myself and fellow student, Erica Allis. We were charged with reviewing the model used by the TWDB and providing technical data for potential DFCs. Realizing the need for both interaction and optimization, I integrated the model into the MASTS framework, creating the MASTS-GMA9 software.

In July 2007, after two semesters of collaborative planning with members of the class, a highly capable and interactive version of MASTS-GMA9 was released directly to planners in GMA9. Members of the TWDB and GMA9 have praised the software as an invaluable tool for analyzing desired future conditions, eliminating many major obstacles in the planning process. The training session on June 29, 2007 was highly successful; the software helped the group focus on specific metrics, resulting in the first constructive group discussion of realistic DFCs since the project began. The GCD managers requested a larger training session for their personnel, conducted on July 9 by Marcel Dulay (a dissertation student from the LBJ School who specializes in water management policy). The MASTS GMA-9 software will play an integral role in helping GMA9 establish their desired future conditions.

The remainder of this section discusses how MASTS was applied to the two groundwater problems mentioned above. For GMA10, the optimization in MASTS was used primarily as a scientific tool to study the impact of increased pumping in different areas, with an auxiliary role in identifying optimal drought management plans. In GMA9, MASTS assumes a more central role in the planning process on a more regional scale.

5.2 GMA10 – Barton Springs Segment of the Edwards Aquifer

The Barton Springs segment of the Edwards Aquifer lies in a rapidly urbanizing area of the city of Austin. Narrative elicitations from various stakeholder groups show that the core concerns are water reliability, preserving sensitive environmental features (Barton Springs and the recharge zone), and the desire for economic growth and urban expansion. These high level concerns have fueled *spirited* debates over more tangible issues such as impervious cover, endangered species protection, changes in land use, drought planning, and economic impacts (Pierce, 2006).

To address these concerns, it was necessary to expand on the Barton Springs Edwards Aquifer GAM provided by the TWDB (Scanlon et al., 2001). The Groundwater Decision Support System (GWDSS) developed by Pierce (2006) added several sub-process models to augment the capabilities of the existing GAM. First, the revised model included reactive drought management to analyze the impacts of water conservation and cutbacks during periods of drought. Second, the model was altered to account for changes in recharge related to impervious cover by using a spatial database to link into the city's zoning maps for precise control. Finally, the model was coupled with a lumped parameter systems dynamic model to provide fast approximations of available water budgets.

Although still under consideration, the DFC for this segment of the Edwards Aquifer will most likely stipulate a minimum required flow level for Barton Springs and required water levels at key indicator wells.

Ultimately, the TWDB can verify whether a particular policy will satisfy the DFC, but they cannot perform an exhaustive search for the *best* way to meet these conditions (this process is still manual). Members of the BSEACD have indicated a number of secondary goals beyond simply meeting the DFC, including:

- maximize total pumping (the groundwater available for use and permitting)
- maximize average storage (a measure of reliability)
- minimize the number of cells that fall below a target saturated thickness (a measure of reliability)
- maximize the minimum spring flow (Barton Springs)
- maximize the average spring flow (Barton Springs)

Both the project managers and stakeholders were reluctant to rush for optimization, especially since the multi-criteria objective is so vaguely defined and no agreement has been reached regarding the DFCs. In addition, optimization cannot be bluntly introduced without the explicit approval of stakeholders; lest it be interpreted as a decision maker rather than a decision aid. Still, MASTS is being used behind the scenes to provide valuable scientific insights into the behavior of the aquifer system. In addition, planners at the BSEACD have been using the optimization to identify ideal drought management strategies.

The GWDSS uses four decision variables to explore the spatial allocation of pumping and drought management policy. One decision variable controls the amount of extra pumping that will be introduced to the entire model. The second decision variable controls how the extra pumping is distributed throughout 11 different zones. This "partitioning" decision variable is actually a meta-variable described in Section 2.3 (MASTS allows decision variables to be more complicated than a simple number). The third decision variable specifies the percent reduction in pumping that will be mandated during an *alarm stage drought*. The fourth decision variable specifies the percent reduction in pumping that will be mandated in a more serious *critical stage drought*. Alarm stage and critical stage droughts are declared when key monitor wells drop below a certain level.

Pierce (2006) ran several searches and conducted experiments to address the five basic objectives most relevant to the BSEACD (listed above in bullets). These experiments made a counter-intuitive discovery that additional pumping could be placed in a zone proximal to the outlet of Barton Springs without adversely affecting the springflow, indicating that this zone might be hydraulically isolated from the Barton Springs flow system.

In addition, Pierce concluded that alarm stage and critical stage droughts did not occur if additional pumping is kept below a threshold value. But this information may be obscured by the fact that pumping estimates throughout the model are considerably lower than the current reported usage (Smith & Hunt, 2004). Thus, Brian Smith (senior hydrologist for the BSEACD) is currently using the GWDSS and MASTS to perform an ongoing analysis of the drought reductions in a revised model that includes extra pumping.

5.3 GMA9 – The Hill Country

The policy research project (PRP) with GMA9 built on the methods established by Pierce (2006) in her work with BSEACD, but evolved in a completely different direction. The stakeholder issues, science, and scope of the GMA9 project were fundamentally different, creating a *complex* new challenge. In particular, the size of the study area, the number of different parties involved, and a non-comprehensive model have made it difficult to hold the productive scientific dialog that must occur in order to establish the DFC. The MASTS-GMA9 software developed for this project will play a central role in opening this dialog, providing analysis tools and optimization to guide the group toward consensus on a DFC.

The key stakeholder concerns were determined through interviews and public forums: water availability (dry wells), reduced springflow, contaminated water supplies, urban encroachment, and growth management. The major barriers to action include limited resources (money, data, staff) complicated by a region-wide resistance to change and regulation.

Scientific analysis and modeling is complicated by the fairly restrictive limitations in the Trinity Hill Country GAM. It has a coarse spatial and temporal resolution, mathematical instability, and it fails to adequately address all of the stakeholder concerns. In addition, it does not account for all of the major groundwater sources. The Llano Uplift Aquifers (Hickory, Ellenburger-San Saba, and Marble Falls Aquifers) are handled in a separate model, and the Lower Trinity has not yet been incorporated into the

Trinity Hill Country GAM. Early in the PRP, it was decided that there is no way to improve or extend the model without major changes to the current GAM, which would require extensive validation to comply with TWDB standards and state law. To meet the December 1 deadline and secure the approval of the TWDB, it was decided that the Trinity Hill Country GAM shall be used *as is*, which has drawn criticism from some GCD managers who feel that this model does not accurately represent their concerns.

One final complicating factor for the GMA9 project: the sheer size of the study area means that any cooperative planning activity is a massive undertaking. This project covers portions of 9 conservation districts, 8 counties, and 4 river authorities. In a group this large, the sciences of facilitation, communication, and negotiation are equally important to the science behind the groundwater availability model. In particular, communicating scientific details and results requires careful moderation because conversations about modeling quickly result in arguments about accuracy and discussion of overly specific details. Overall, the scientific dialog suffers from imprecise models, a failure to focus on the regional picture, and a lack of usable tools to interact with the model. This impedes progress towards a DFC, which must take the form of specific numbers based on the scientific model from the state.

As we introduced MASTS-GMA9, some of these barriers started to disappear. Primarily, the software has bridged a large portion of the information gap, allowing planners to analyze the impact of pumping changes (both increase and decreases) throughout the aquifer on a county by county basis. This has created a more focused discussion about the DFC. This section will describe some of the features and limitations of this model in more detail an will provide a description of the software.

5.3.1 The Trinity Hill Country GAM

The Trinity Hill Country Aquifer Groundwater Availability Model (GAM) was developed by the TWDB to simulate regional water levels and availability over a 50 year planning horizon. Three aquifers are included in this model as separate layers: the Edwards Group of the Edwards-Trinity plateau, the Upper Trinity, and the Middle Trinity. Groundwater pumping in the model increases over time based on projections from the Regional Water Planning Groups (RWPG) (Regional Water Plans, 2001). The model also includes a repeat of the drought of record at the end of the 50 year run. The steady state model is calibrated to observed water levels from winter 1975-1976, and the transient model is calibrated to observed well levels during a 24 month period (1996-1997).

The TWDB has created a thorough report of this model (Mace et al., 2001). This report provides detailed information on the model structure, inputs such as pumping rates and recharge, and the calibration procedures. The report also contains a wide range of figures that summarize the model results including saturated thickness, drawdown, and water budget analysis. However, more model runs with different input parameters are required to provide insight into the aquifer response and potential desired future conditions.

The Trinity Hill Country Aquifer GAM achieves its purpose, describing groundwater availability on a regional scale, but has its fair share of flaws and critics. It is important to communicate the shortcomings of the model, so that planners can avoid making decisions on flawed data. The managers in GMA9 *should* hold a healthy skepticism; over-reliance on any model would be ill-advised. At the same time, dwelling on these flaws can be highly destructive to the negotiation process. Many constituents of GMA9 have voiced concern about the scope and accuracy of the Trinity Hill Country Aquifer GAM. The flaws in the model quickly become bludgeons when participants don't see a way to get what they want. To quiet this contentious crowd, expert facilitator David Eaton has one answer:

> "This model is currently the only tool endorsed by state law for groundwater planning in the Trinity-Hill Country Aquifer for GMA9."

And while this answer is blunt, it exposes the consequences of HB1763: the model at its core is very much a political tool.

This central role highlights the need for a simple software utility that helps stakeholders and managers interact with the model. The MASTS-GMA9 software fulfills that role, allowing users to explore the impacts of increased or decreased pumping in the Hill Country area. This tool can be used to draw scientific insight and understanding from the model. More importantly, the software enables all parties to enter the discussion on common ground, finding ways to interpret the model to support their viewpoint. At the very least, MASTS-GMA9 will allow any discontented parties to quantify their objections to the Trinity Hill Country Aquifer GAM, which is the first step to arranging a solution through negotiation.

Before describing the role of software, we introduce the main features of the model, including some the potential weaknesses about which planners should be aware.

Baseline Model Performance

The model shows the impact of projected pumping through the year 2050, and includes a repeat of the recharge conditions experienced during the drought of record (1950-1956). Average recharge conditions, based on the average precipitation from 1960 to 1990, are applied from 1998 to 2043. The drought of record occurs from 2043 to 2050, at the end of the model simulation. At the peak of this drought, recharge from precipitation dropped to half of the average (non-drought) values. This study has the drought positioned in the last seven years of the model simulation because it is the most conservative scenario (the drought occurs during the highest water demand phase of the model).

Figure 5.3 shows the average drawdown in the Middle Trinity Aquifer for each county in the model. In 2040, before the onset of the drought, Kendall, Comal, and Hays Counties have experienced an average drawdown of about 30 feet (compared to the hydraulic head in December 1997). The drought pushes this decline to about 65 feet in 2050. Bandera and Kerr Counties experience declines of about 30 feet by 2050. The region in northern Bexar western Comal, and southern Kendall Counties is the hardest hit, with drawdowns of more than 100 feet. The contour plot shown in Figure 5.4 shows the drawdown for the Middle Trinity Aquifer in the year 2050.

Coarse Spatial and Temporal Resolution

Each cell in the model is one square mile in area. The coarse spatial resolution is adequate for making regional inferences about water level trends, but may not provide highly localized information. For example, it is unlikely that the water level in a specific well could be predicted using the model. The model reports an average water level for each cell, but the water table can vary drastically over the course of a mile based upon topography. Moreover, during the predictive phase, the model considers the averages over a whole year, and thus it does not include the seasonal variations that can impact local wells. The model does, however, allow users to identify regional behavior which may be used to indicate trends in hydraulic head levels.

Calibration

Although the model is calibrated, it remains a generalization of a very detailed hydraulic system. Predicted and observed water levels in the steady state calibration differ by as much as 100 feet in the north-eastern portion of the aquifer, and the root mean squared (RMS) error is reported as 56 feet. The calibration of the transient model was a manual procedure, with similar margins of error. By comparison, the saturated thickness of the aquifer in 1997 is about 100 feet along the northern boundary and 500 feet along the southern boundary. The magnitude of this error is understandable given the sparsity of hydrogeologic data for the region.

Limited Applicability to Springs and Rivers

The model has limited applicability to springs. Nineteen different springs are represented in the model with the DRAIN package. The GAM report indicates that the steady state model accurately simulates flow to 16 of the 19 springs. Given the scale of the model, accurate springflow (in this case) means the correct order of magnitude. This is "good enough" for a regional model with coarse temporal and spatial resolution.

From the stakeholder perspective, extreme caution should be exercised when interpreting springflow values from the model. To understand how rough the estimate is, one must examine the methods used to calculate springflow data. The predicted springflow is dependent on both the water level and an associated drain conductance for the spring. The steady state model was calibrated to the 1975 water level, and has an associated RMS error of 56 feet. The TWDB then used estimates of springflow to estimate proper values for the conductance. This procedure can introduce additional error.

In this particular model, the errors associated with the drain conductance may be significant. The estimates for springflow that were used to find appropriate values for the conductance (in the 1975 steady state model) are taken from *single* springflow measurements collected in a variety of different years (1940, 1960, 1966, 1967, 1970, 1973, 1975, 1976, 1988, and 1991) (Table 2, pg 46) (Mace et al., 2001). In addition, these measurements were taken at different times of the year, some during January, March, April, June, July, August, and December. The TWDB modelers were aiming for the proper order of magnitude, rather than a precise springflow representation. Springflow is secondary information from this model, subject to both estimation errors in the conductance and the error in the calibrated water levels. This extra uncertainty should be considered before using springflow from this model as a desired future condition.

The drainage (baseflow) to the rivers is also subject to secondary errors associated with the drain conductance. The conductance is calculated based on the estimated width of the stream, an assumed length of 1 mile in the actual cell, and a fixed hydraulic conductivity (the same for every river). Even with these rough assumptions, the simulated baseflow to the Guadalupe, Medina, and Blanco Rivers is within 25% of estimates. This is better agreement than with many of the springs. Thus, it may be more desirable to use baseflow to rivers as part of a desired future condition, except during drought conditions.

The trends in the model show that during a drought of record, baseflow to rivers may decrease 60 to 65 percent, and springflows may decrease by 55% (averaged over a year). It would be a useful exercise to see how this compares with historical data from the drought of record. Report 353 provides no indication that the model is accurately reproducing the aquifer stress experienced during this drought. Specifically, the assumption that rivers continue to gain during the drought is *suspect*. The drought plays an important role in the model to reinforce conservative planning, but care should be taken when phrasing a desired future condition that is tied exclusively to the drought; this portion of the model may have higher errors.

Convergence Issues

The UT GMA9 class varied the pumping in the model to observe how water levels might be affected with changes in pumping. The model did not always converge for the each of the pumping test scenarios (even with minor changes). The lack of convergence is marked by the presence of oscillating cells in the model, This could be caused by the coarse spatial resolution, thin layers, and the large time steps used in the model (such factors are known sources of instability in finite difference equations). The problem appears more frequently during the drought portion of the model.

5.3.2 MASTS-GMA9 Software

In addition to optimization services, the MASTS-GMA9 software itself is a user-friendly wrapper for the model provided by the TWDB. The software provides a graphical user interface (GUI) which allows users to make changes to the pumping in different zones and layers of the model. The results for each scenario are saved, and the software provides a graphing utility to investigate and compare different pumping scenarios. The ease of use makes it a viable utility for live discussion sessions. By March 2007, a stand alone application had been developed specifically for GMA9.

Model Zones

Modflow inputs and outputs are given on a cell-by-cell basis. The data are easier to manage if the model is divided into different zones. This helps organize the input data, and provides a simple and meaningful way to report the outputs with aggregated regional data. These zones could encompass cities, counties, hydrogeologic units, expected growth corridors, or other regions relevant to planners. For each zone, the software gathers measurements to assess the model performance, including head levels, drawdown, and detailed budget information from the cell by cell budget file.

Since Modflow itself does not contain spatially indexed data, GIS was used to overlay the model grid onto a map of GMA9 to determine the precise location of each cell in the model. In this manner, the modeling group has created several zones for consideration in the model. The zone definitions are located in the zone.dat file (the format is straightforward, but different from the ZONEBUDGET program).

First, a zone has been created for each of the eleven counties within the model (2 counties are completely outside GMA9). This will allow a regional analysis of the model outputs. Another set of zones has been created for the cities, including any cells within two miles of an urban area. The ten city zones allow a user to control the pumping and examine model outputs for a specific city. Figure 5.5 and Figure 5.6 show the model grid superimposed on the region, with cities and counties labeled.

Zone Based Pumping Scenarios

The MASTS-GMA9 software allows users to edit the pumping in each zone using slider bars. For simplicity, the changes to pumping are defined relative to the baseline GAM model via a pumping factor. A pumping factor of 1.1 corresponds to a 10% increase in the pumping specified in the baseline GAM model. This increase is applied to every cell in the zone for the predictive phase of the model (stress periods 27-79). A pumping factor of 0.9 indicates a reduction to 90% of the pumping in the baseline GAM model.

It is important to remember that the baseline GAM (the original model provided by the TWDB) already contains projected increases in the pumping. Users should review these projections, and decide whether they are still accurate or should be increased or decreased. The TWDB notes that the projections from the RWPG (included in the model) are for dry conditions, and may be somewhat higher than actual demand (perhaps by 2-20%).

Pumping factors can be used to scale the pumping rates up or down, but they cannot change the spatial distribution of pumping within a zone. Thus, this approach relies on the current distribution of pumping. Figure 5.7 shows that in the Middle Trinity, significant pumping is evenly distributed in every cell of Bandera, Bexar, Comal, Hays, Kerr, and Travis County.

Moreover, most counties contain “pumping centers”, cells that have high pumping rates. Changing the pumping factor affects these pumping centers as well as the distributed pumping.

There are two potential pitfalls when changing the pumping factor. Increasing the pumping factor for large pumping centers can place local stress on the model. This is particularly relevant in Bexar and southern Kendall County, where the model already experiences a large number of dry cells in 2050. Also, users should remember that pumping factors cannot introduce new pumping in a cell where there is currently no pumping. This warning is less applicable in areas where pumping is distributed across the region.

Despite these potential drawbacks, pumping factors were chosen because they are simple to use. The software is designed to support real time negotiations, and users will not have time (or the proper information) to edit pumping cell by cell. The goal of the pumping factors is to allow stakeholders quickly explore the impact of increased or decreased pumping, and the calculation does not have to be 100% precise.

Zone Based Reporting Tool

The software can read all of the primary Modflow outputs contained in the HEAD and BUDGET files. These files contain cell-by-cell data for every layer for every time step for every stress period. The HEAD file is approximately 27MB for this particular model and the BUDGET file is approximately 60MB. Users may only be interested in some of these data, such as the head in an individual cell, the average drawdown over a specific zone, the total change in storage over the entire aquifer, or the discharge to a specific river. The zones created for this software can be used to organize and analyze Modflow outputs. The HBA (head and budget aspects) file contains the instructions for which data to collect. By manipulating the HBA file, users can request data (average, sum, max, min, contours) for any cell, zone, or layer defined in the model, for any component of the head and budget files. For the Trinity Hill Country model, the HBA file has already been set to collect the information relevant to planners in GMA9, but additions can easily be made.

For example, suppose Johnson City is concerned about how localized pumping affects head levels in the Middle Trinity Aquifer. To obtain this information at stress period 30 time step 1, the user would include the following lines in the HBA file.

```
HEAD; AVG; 30 1; Johnson City AND LAYER3

WELLS; SUM; 30 1; Johnson City AND LAYER3
```

This instructs the Modflow post-processor to fetch data from the proper output files. The Johnson City zone contains two layers. In this case, the “AND” operator indicates that the program should take the intersection of the Johnson City zone and LAYER3; computing the average head level those cells in both Johnson City *and* LAYER3. In the second line, WELLS refers to a field in the cell by cell budget file. Other potential fields include that may be used are STORAGE, RECHARGE, and DRAINS, and DRAWDOWN. Readers interested in more detailed options may examine the HBA file included with the software.

For every model run, the items listed in the HBA file are collected and archived to the hard drive for future analysis. This is essentially a condensed summary of the key results from the model run. The graphing utility in the program interprets and presents the data as time series plots. When the summary spreadsheet is created, each of the items in the HBA file appears on a separate line. This data can then be imported into Microsoft Excel or Access. Overall, the HBA file allows users to create a customizable summary for any model.

Software Verification

The modeling group has verified that the code managing the pumping profile, zones, and phases is working properly. Project staff validated the code by cross checking outputs from the baseline model in PMWIN with those generated using the software. Hydraulic head, well discharge, and drain discharge were parameters used in the cross check exercise.

Basic Software Capabilities

The MASTS-GMA9 software ties together the zone-based analysis tools described above with a friendly graphical user interface,acting as a "wrapper" for the currently existing GAM model developed by the TWDB. As described above, users are allowed to change the pumping in any zone for the predictive phase of the model (stress periods 27-79). The only input file that changes is the WEL file, which contains the pumping definitions. All other model files remain untouched.

In the basic version of the software, users create a pumping scenario by adjusting the sliding bars that control the pumping factor for each zone. The model can be executed with the new settings with just the push of one button. This scenario can be saved with a descriptive name to the hard drive, and will be available when the program restarts.

On the SOLN tab, users may create graphs or contour plots for any of the saved scenarios. In particular, Figure 5.8 illustrates how users can create graphs that compare a number of different scenarios. The data used to create these charts can be written to an Excel spreadsheet as a table using the "write table" button. In addition, as shown in Figure 5.9, the graphs can be *detached* into separate windows so that many graphs can be viewed at once. Finally, the graphs can be saved as pictures using the right click context menu.

Finally, by request, users are allowed to run the 50 year simulation with or without the drought conditions. The software is designed so that it is not possible to directly compare a drought scenario with a non-drought scenario.

Using the Software for Analysis

Installation instructions are provided with the software distribution in a "read me" document. Once installed, users should keep in mind the following important points while interpreting the results:

- The model will not converge for all settings. Users cannot save the results if the model does not converge. A large red button and error message appear when the model does not converge, but it is okay to continue using the program. The button and error messages can be hidden from view again.

- The program does not track the units for the reported values. Table 5.1 contains a summary of the reported values, providing details about the computation and the proper units.
- Dry cells can create unexpected results while increasing the pumping. Since Modflow turns off the pumping in dry cells, the actual withdrawals from the model may not match the values specified in the WEL file. As pumping increases, the number of dry cell increases, and the amount of water withdrawn from the aquifer could decrease. Users can check the actual withdrawals via the graphing utility.
- Dry cells can create problems while interpreting the results. Neither the averages nor the sums include dry cells in the model. When dry cells are present, the results may be slightly skewed. For example, beyond a certain point, it is possible that the increased pumping may not exhibit a corresponding decrease in head levels. The most likely cause is a large number of dry cells. The graphing utility allows users to see the number of dry cells.
- The averages are county wide. Larger counties will have greater variation within their borders.
- It is not easy to measure the absolute influence of pumping in one county in another target county. The impact in the target county is a measure of both the *proximity* of the two counties, and the size of the target county. If the target county is small, then it is possible that pumping in a neighboring county can have a strong influence throughout the entire target county. If the target county is large, then only a fraction of it may be affected, having a limited impact on the countywide average.

5.3.3 Recent Developments

In the months prior to the December deadline, the GMA9 planning process stalled on a few major issues. First, the Hays Trinity Conservation District management plan specifies that aquifer discharge to springs and rivers should be included in the desired future conditions (Campbell et al., 2005). This feature was not included in the original software because the various drain cells in the model were not clearly categorized. Working with Ali Chowdhury, this data is now clearly documented (Chowdhury, 2007) and reported in the MASTS-GMA9 software.

Some planners also wanted to eliminate the drought simulation from the model. With all of the uncertainty in the model, the drought could obfuscate the planning process. The position and duration of the drought are speculative, and the drought causes much of the numerical instability in the model. In addition, the model does not include reactive measures (such as mandatory pumping reductions) that would certainly reduce water usage during a drought. By request, the MASTS-GMA9 software can now be run through 2050 without a simulated drought.

Finally, after examining the model using MASTS-GMA9, it was discovered that the pumping projections from the Regional Water Planning Groups (Regional Water Plans, 2001) are no longer accurate. Table 5.2 compares the pumping in the baseline GAM with the updated estimates for 2007. For the Trinity Glen Rose GCD, the current pumping estimates are 2.6 times higher than the projected pumping that appears in the baseline GAM. This region of the model (San Antonio and northern Bexar County) is already under significant stress, with an average drawdown of close to 100 feet (and many dry cells are already present). The model fails to converge if extra pumping is added. In other regions, the discrepancy between current pumping and the baseline model is surmountable, but it requires careful group decisions about how to best address the problem. Since that is a fairly major step, the MASTS-GMA9 software still refers to the pumping projections in the original baseline GAM until a more refined estimate can be produced.

5.3.4 Search Capabilities with MASTS

Although the constituents of GMA9 are still in the process of joint planning, it is likely that each GCD will specify the maximum acceptable drawdown (averaged across the GCD) that they are willing to accept in the year 2050. Some GCDs may also specify a minimum acceptable discharge to springs. The MASTS-GMA9 software can help managers establish these limits. However, drawdown estimates may be optimistic unless each district considers the affects of increased pumping in neighboring districts. This inter-connectedness and the need for collaborative planning were discussed heavily at the training sessions (UT GMA 9 Groundwater Management Class, June 27, 2007).

The goal of the search is to adjust pumping rates (via pumping factors, as described in Section 5.3.2) until every GCD meets or is safely in excess of the desired future conditions, subject to the additional constraint that they also have enough water to meet anticipated demand. The precise form of the search objective is a matter for the planners to decide (with the help of a facilitator familiar with their goals). It may not be possible to find a scenario that satisfies these constraints for each GCD. In this case, the search will look for the closest match. As it explores this infeasible region of space, planners will learn more about what *is* physically possible. Most likely, both the drawdown and water demands are negotiable.

In the event that a satisficing scenario (or scenarios) can be found, the search will try to minimize the average drawdown within each GCD (while meeting the water demand constraint). The search will not attempt to maximize the amount of water pumped from the aquifer. Given the uncertainties in the model, the antiquated water laws in Texas, and the critical importance of water resources, it seems prudent to take a conservative approach to determining the managed available groundwater (MAG). If we *overestimate* the MAG, then GCDs may be *required* to permit water use up to the MAG . Thus, it will be satisfactory to meet but not exceed the water demands for each GCD.

A preliminary *rule based objective* that conforms to the requirements above has been developed and tested with the Trinity Hill Country model. Since GMA9 has not agreed on desired future conditions or the exact phrasing of the objective, the results of this exercise are highly speculative. However, it has validated that the advanced features of MASTS are working as intended. The search was run on six processors (3 computers) for about two days, building an archive of 7000 model runs. The archive was combined onto one computer, and consumed about 14GB.

It is possible to rapidly search through the archive with different objective functions. To test this capability, all objective scores were cleared, and the archive was flushed completely to the hard drive so that no solutions were stored in memory. Five minutes were required to restore and evaluate 7000 solutions from the archive. (This time requirement is governed largely by the efficiency of the hard drive used.) By comparison, it took about 14,000 minutes to originally obtain the 7000 solutions. Given the opportunity to evaluate and archive solutions in advance, this MASTS capability will, for the first time, allow real time search analysis and rapid dispute resolution in many domains where such a process was heretofore impossible.

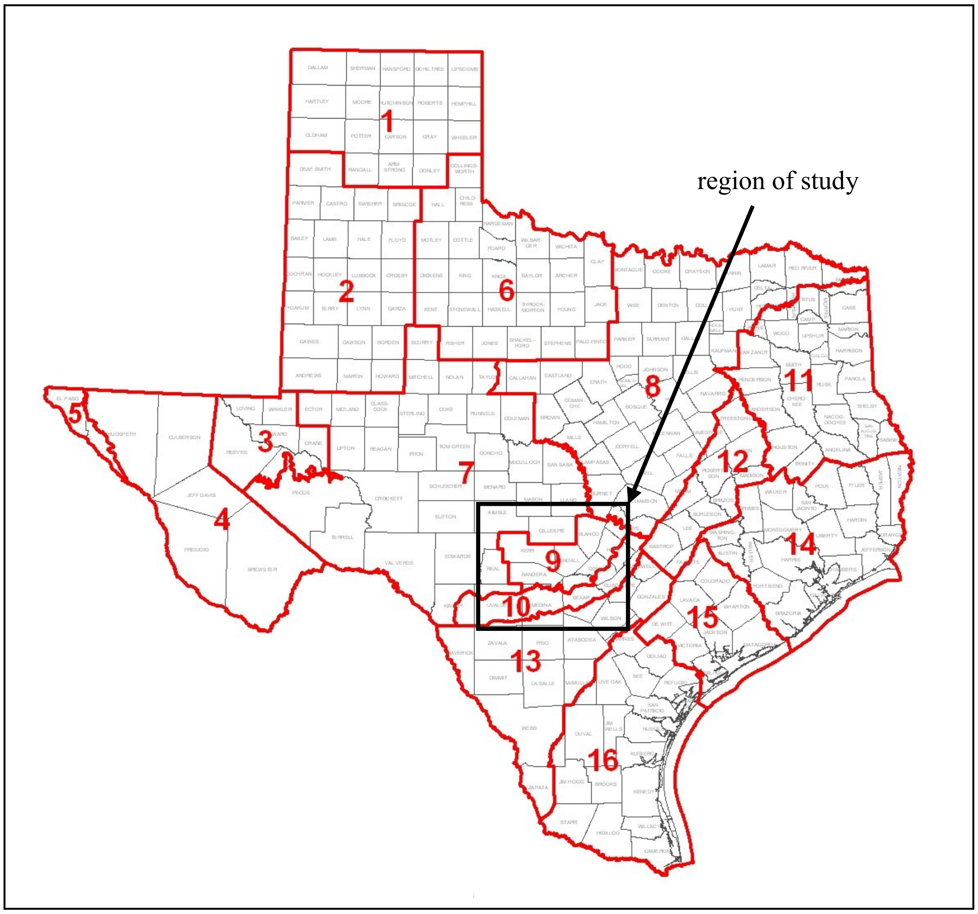
5.4 Conclusions

In two separate groundwater planning projects, MASTS has been vital as a tool for discussion, analysis, and planning. In the Barton Springs Segment of the Edwards Aquifer, MASTS illustrated the impacts of different spatial configurations of pumping, and is being used by the BSEACD to explore different drought policies. In GMA9, MASTS has invigorated the planning process by providing a user-friendly interface to the model and results. Powerful optimization tools including rule based objectives and solution archiving can be used to search for a scenario that best satisfies their desired future conditions.

The success and flexibility that MASTS has demonstrated with groundwater decision problems has led Sandia National Labs to adopt a preliminary version in their Computer Aided Dispute Resolution (CADRE) software (Cain et al., 2008).

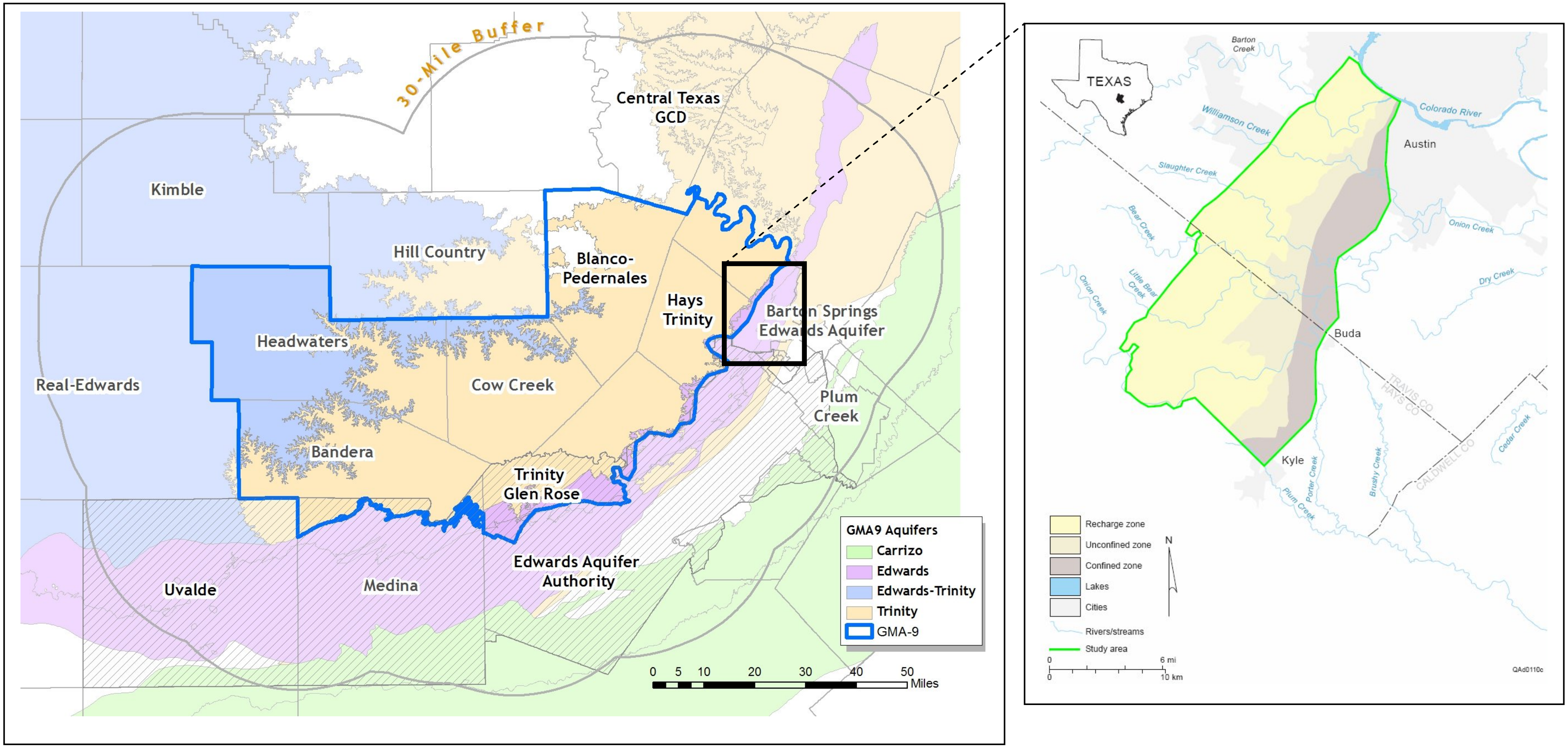
5.5 Figures

Figure 5.1 House Bill 1763 has placed groundwater planning under the regional control of 16 groundwater management areas (GMA).



Source: Texas Water Development Board, http://www.twdb.state.tx.us/GwRD/GMA/gmahome.htm

Figure 5.2 Two landmark projects examined groundwater issues in different Texas groundwater management areas.



Source: Figure developed by the UT GMA 9 Groundwater Management Class 2007

The policy research project conducted by the LBJ school encompassed all of GMA9, including portions of 9 groundwater conservation districts (GCDs) labeled above. The groundwater availability model for this region simulates the Trinity Aquifer (shown in peach).

Source: Texas Water Development Board

The dissertation from Suzanne Pierce involved the Barton Springs segment of the Edwards Aquifer, in the Barton Springs /Edwards Aquifer Conservation District (BSEACD), part of GMA10.

Figure 5.3 The average drawdowns in the Middle Trinity Aquifer for the counties in the Trinity Hill Country GAM. In the last ten years (2040-2050), the drought increases the rate of head level decline. The average drawdown is computed over all the cells that reside within each county (ignoring dry cells).

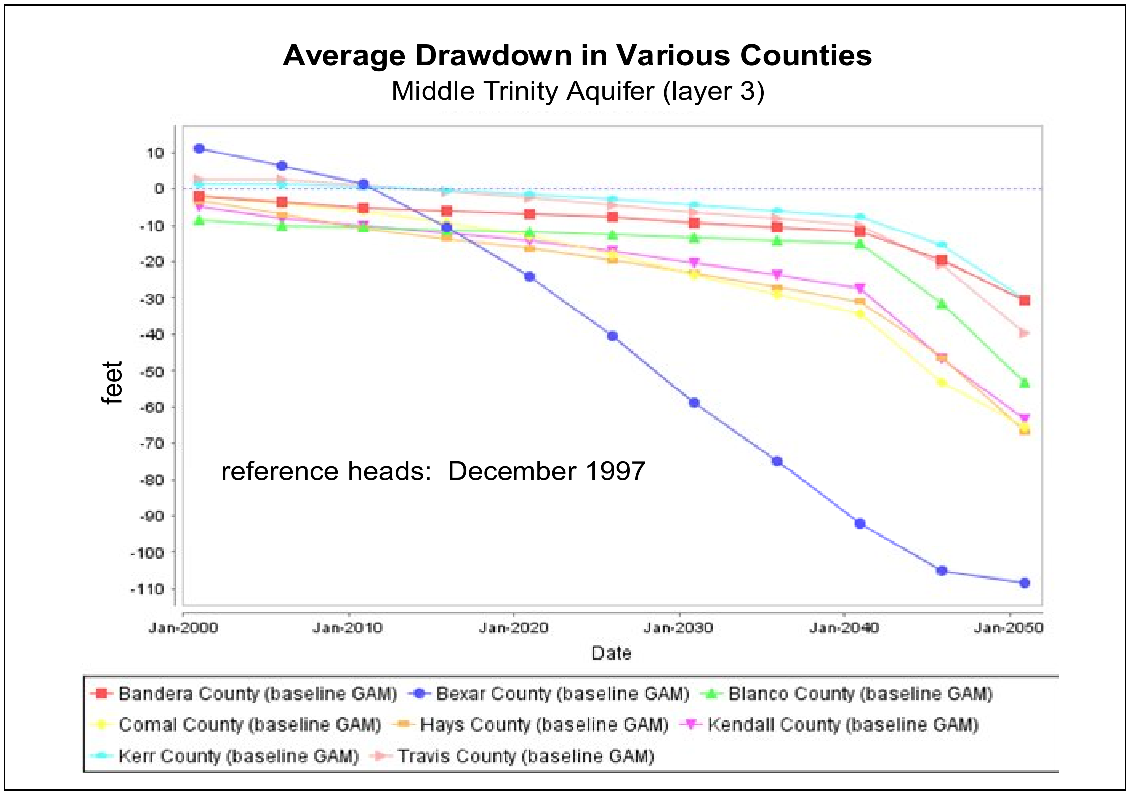


Figure 5.4 A contour plot of the drawdown in the Middle Trinity Aquifer at the end of the simulation (the year 2050 after a repeat of the drought of record). The drawdown is computed relative to the head levels in December 1997. White cells represent constant head boundaries, lakes, or dry cells.

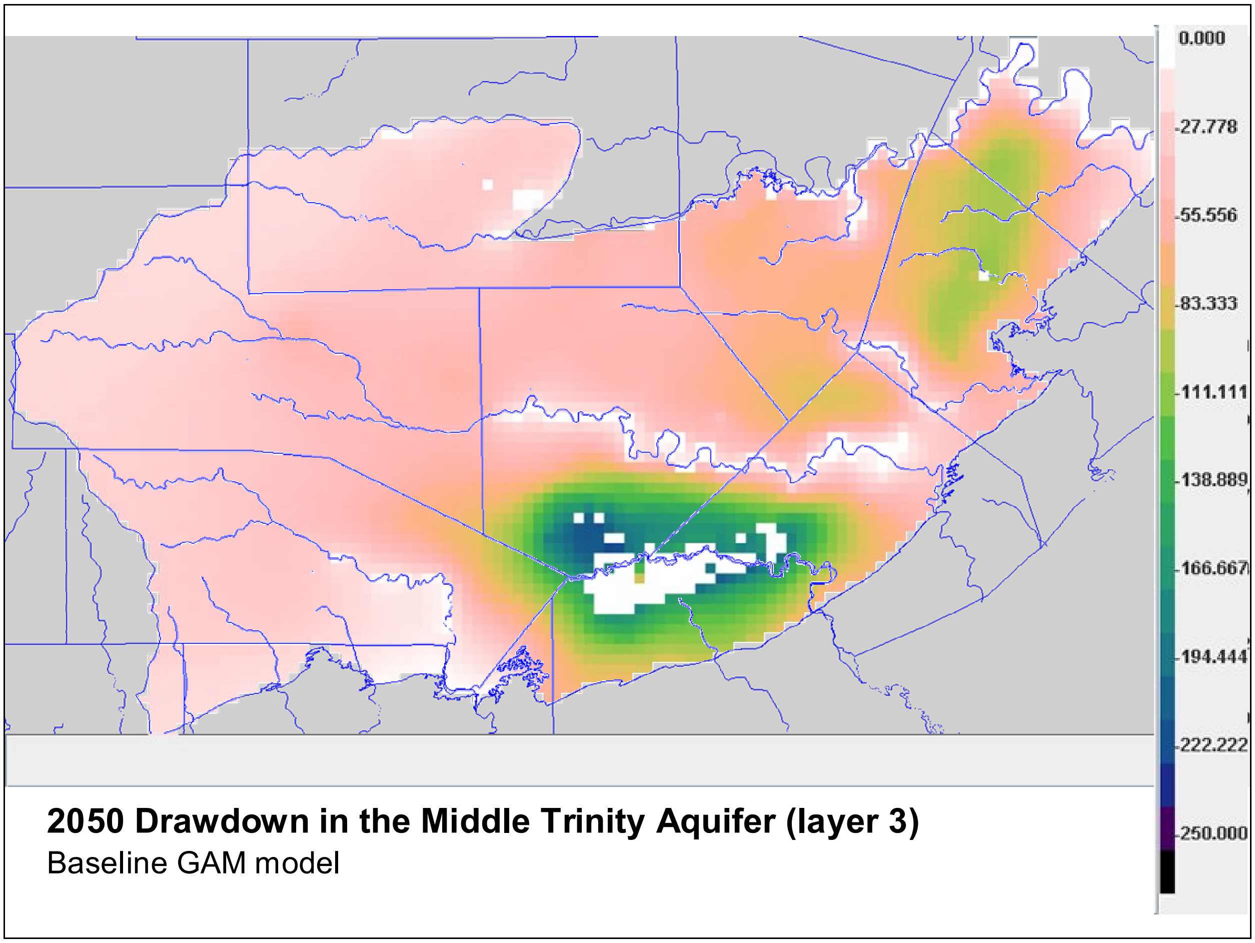


Figure 5.5 The model grid for the Trinity Hill Country GAM as it overlays the counties in GMA9.

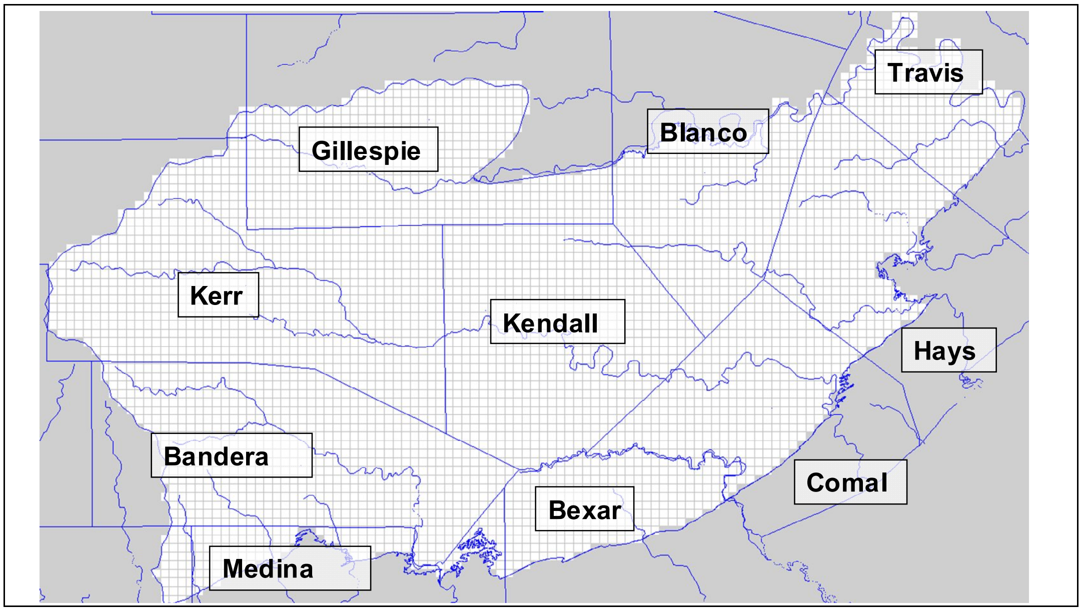


Figure 5.6 The model grid as it overlays the cities in GMA9.

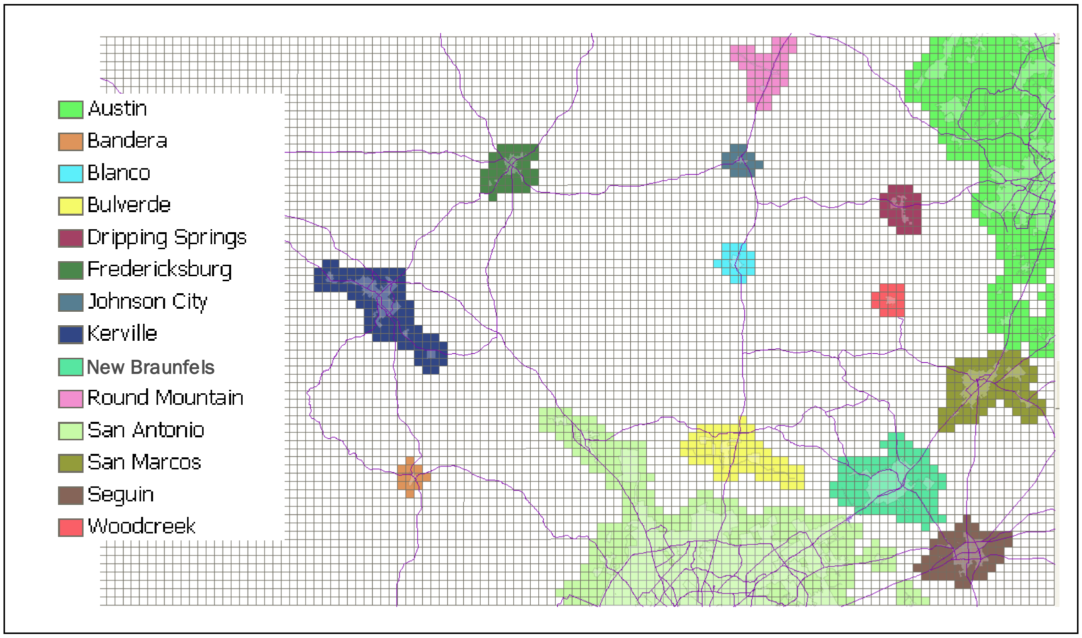


Figure 5.7 A contour plot showing the pumping for layer 3 (the Middle Trinity) in the year 2050. The units are ft^3/day. Most counties have distributed pumping along with some major pumping centers. Cells shown in black are the major pumping centers, pumping 15,000 ft^3/day or more. White cells indicate little or no pumping (some may be dry cells or lakes).

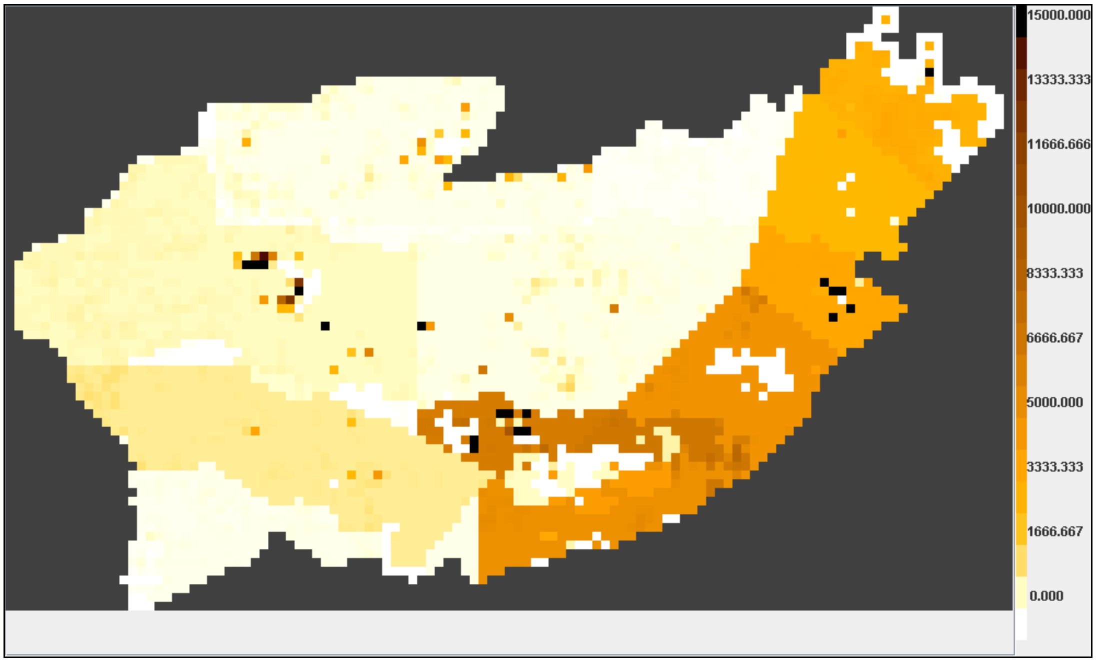
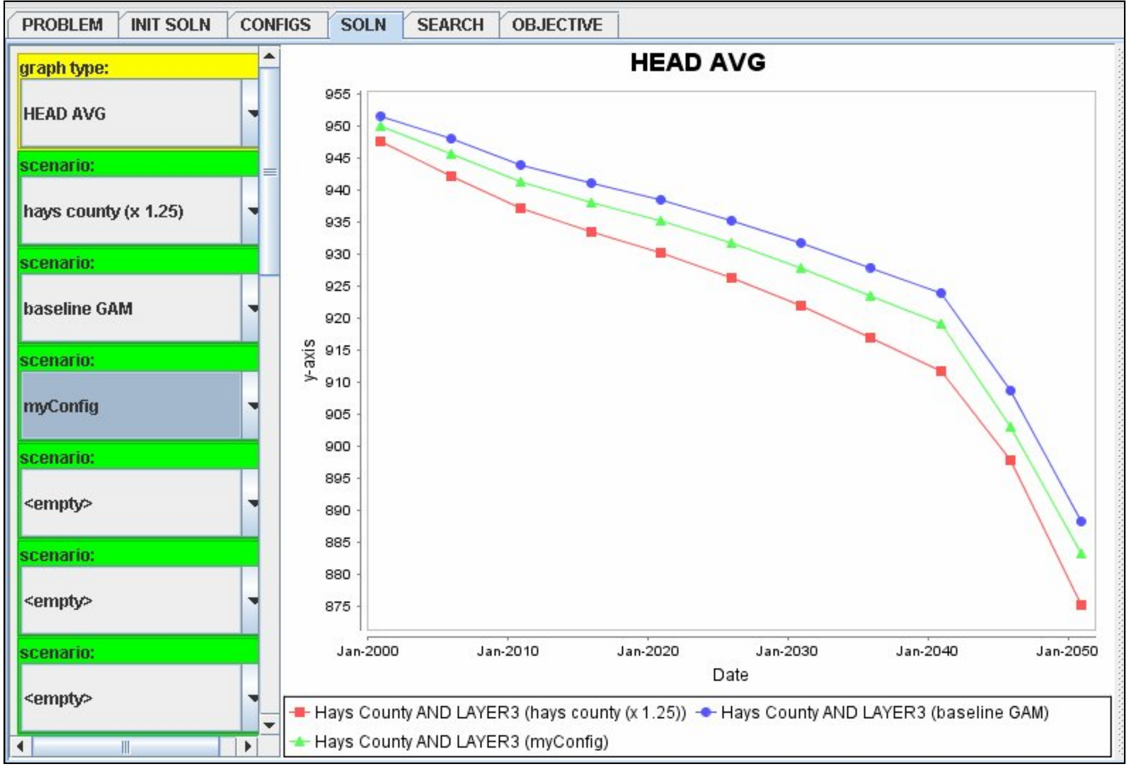


Figure 5.8 A screen shot showing the graphing capabilities of the MASTS-GMA9 software.



graph type: HEAD AVG

scenario: hays county (x 1.25)

scenario: baseline GAM

scenario: myConfig

scenario: <empty>

scenario: <empty>

scenario: <empty>

Figure 5.9 Users may create and compare several types of graphs

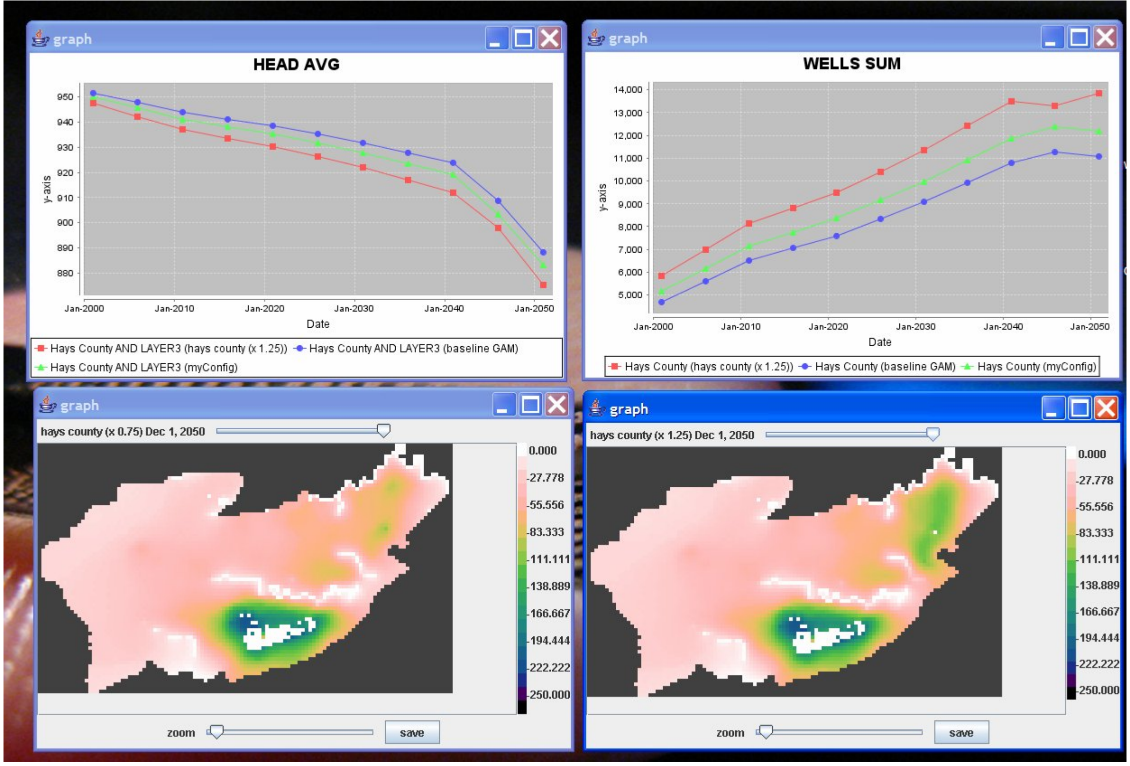


Table 5.1 The quantities and units reported by the software.

Value Name	Units	Notes
HEAD AVG (average head)	feet	Averaged over all the cells in the zone (excluding dry cells). The average head is only computed for one specific layer at a time (never averaged across layers).
DRAWDOWN AVG (average drawdown)	feet	The drawdown is referenced to the head levels in the model in December 1997 (stress period 26). The values are averaged over all the cells in the zone (excluding dry cells). The average drawdown is only computed for one specific layer at a time (never averaged across layers). Negative drawdown indicates that water levels have dropped compared to 1997.
DRAINS SUM (total drainage for an entire layer)	ft^3/day	Sums up the DRAINS entry in the cell by cell water budget file, for every cell in the specified zone. This information is only calculated for each layer. In addition to rivers and springs, the model contains drain cells that allow water to flow from one layer to another. The inclusion of these drain cells means that the drain sum is much larger than *just* the baseflow and springflow. A negative value indicates water is leaving the aquifer through the drains.
RECHARGE SUM (total recharge for a specific zone)	ft^3/day	Sums up the RECHARGE entry in the cell by cell water budget file, for every cell in the specified zone. A positive value indicates that water is entering the aquifer.
WELLS SUM (total pumping for a specific zone)	acre-ft /year	Sums up the WELLS entry in the cell by cell water budget file, for every cell in the specified zone. A positive value indicates water is being pumped from the aquifer. The units are presented in acre-ft/year.
DRAINS SUM (total drainage for a specific zone)	acre-ft /year	Sums up the DRAINS from the budget file. A positive value indicates water discharging from the aquifer through the drain feature.
HEAD DRYCELLS (number of dry cells)	--	The number of dry cells found in a specific zone.
Contour Name	**Units**	**Notes**
DRAWDOWN CONTOUR	feet	Shows the drawdown since 1997. A negative value indicates a water level decline. Dry cells are shown in white.
HEAD CONTOUR	feet	Shows the head level in the aquifer. Dry cells are shown in white.

Table 5.2 A comparison of the pumping in the baseline GAM with more current estimates for the year 2007.

GCD	2007 pumping in baseline GAM	updated 2007 estimates*
Blanco-Pedernales	624	1,592
Cow Creek	5,428	6,000
Hays-Trinity	6,027	4,837
Headwaters	5,605	8,693
Trinity Glen Rose	6,896	18,100
Medina	235	365
Bandera	4,368	3,328

*source: Ron Feiseler, General Manager GMA9

6 Conservation Area Network (CAN) Planning

6.1 Overview

The extinction of a species is an irreversible act. In recent decades, the threats to biodiversity from anthropomorphic encroachment, resource extraction, and habitat change have become more imminent (C. R. Margules & Sarkar, 2007; Pimm, Russell, Gittleman, & Brooks, 1995). In response, the science of conservation biology emerged with the goal of preserving biodiversity, primarily through the establishment of conservation areas. The central problem of *how* to design effective conservation area networks has matured from its theoretical origins into a structured, data driven, practicable framework called systematic conservation planning (C. R. Margules & Sarkar, 2007).

An effective conservation area network must represent unique elements of biodiversity and provide for their persistence. The persistence of biota depends on many criteria, including the spatial configuration of conservation areas, other biological features, and a variety of economic and socio-political factors (C. R. Margules & Sarkar, 2007). The synthesis of these criteria into an economically viable conservation plan often involves complex optimization problems which necessitate the use of software as a planning aid.

ConsNet, a comprehensive software package for systematic conservation planning, contains powerful new techniques for building conservation area networks while taking into account spatial requirements such as size, compactness, connectivity, and replication. These spatial criteria are integral to the planning process, but have been largely ignored due to the computational and modeling difficulties of considering them (C. R. Margules & Sarkar, 2007).

ConsNet is the next generation in conservation planning software, representing a substantial improvement over similar software packages in utility, performance, and ease of use. Built on MASTS, ConsNet uses innovative tabu search strategies such as rule based objectives (Chapter 3) and dynamic neighborhood selection (Chapter 4) to improve search performance remarkably. Beyond optimization, ConsNet is designed to be a planning tool and decision aid; planners design their own searches, analyze the results in real time, create portfolios of preferred solutions, and save their progress. This chapter discusses the advanced strategies employed by ConsNet, ending with selected results from three sample data sets.

6.1.1 Systematic Conservation Planning

Protected conservation areas are the foundation for modern conservation efforts. The International Union for Conservation of Nature (IUCN) has designated seven categories of protected areas, based on the level of protection afforded by local laws (http://www.iucn.org). Conservation areas fulfill the goals of a conservation plan in two ways. First, they may contain a broad *representation* of biodiversity. Second, they protect biodiversity from outside threats (*persistence*) (C. R. Margules & Pressey, 2000).

While preserving biodiversity is the primary goal of conservation efforts, the definition of biodiversity is extremely broad. A conventional definition encompasses diversity in genes, species, and ecosystems (Raven, 1992). This definition may not be inclusive enough, compared to the preferred "variety of living features and processes at all levels of structural, taxonomic, and functional organization" (C. R. Margules & Sarkar, 2007). To reduce the vagueness, the implementation of a systematic conservation plan depends on an operational definition of biodiversity that places the focus on a set of biodiversity surrogates (Sarkar & Margules, 2002). Even then, conservation efforts can differ greatly depending on the stated specific goals, and there is much debate over what exactly should be conserved (Sarkar, 2005).

Semantics aside, biodiversity is an irreplaceable resource and asset. Worldwide, fisheries depend on wild species and intact ecosystems. Humans depend on pharmaceutical medicines that were first found in nature and on traditional medicines from plants and animals. For some countries, biodiversity yields significant income from recreation and tourism. Genetic biodiversity teaches us how some species overcome pests and diseases, and we routinely take advantage of adaptations in species, for instance, the cultivation of crops that can survive in harsh conditions (Raven, 1992). The full potential of biodiversity is unknown and unexplored (Sarkar, 2005).

Conservation efforts must be strategically focused for two dominant reasons: (1) habitat destruction may lead to increased species extinction rates (Pimm et al., 1995) and (2) the resources and funds available for conservation efforts are very limited, requiring prioritization. In the past, reserves have been chosen in areas that have cultural or scenic values, or areas that are unsuited for agricultural or economic development (C. R. Margules & Pressey, 2000). As such, existing conservation area networks may not be appropriate for preserving biodiversity. Recent research reveals another source of urgency. Failure to create effective conservation areas *now* can lead to much higher costs to achieve similar goals in the future (Fuller, Sanchez-Cordero, Illoldi-Rangel, Linaje, & Sarkar, 2007).

Recognizing the need for systematic planning, Margules and Pressey (2000) describe a dynamic six step system which lays the framework for biodiversity preservation. This process is expanded to eleven comprehensive steps in Margules and Sarkar (2007). We present an *abbreviated* description of the steps that are particularly relevant to the planning software:

1. Compile, assess, and refine biodiversity and socio-economic data for the region

This stage is arguably the most important because the quality of the collected data directly impacts the quality of the results. Typically, data for biotic and environmental parameters are compiled from existing sources, climate databases, remote sensing, and targeted biological surveys (if resources allow). This stage also includes data treatment which often involves niche modeling or other methods to predict the distribution of species. Finally, planners should collect data regarding economic and social issues that may factor into the final conservation plan.

2. Identify Biodiversity Surrogates for the Region:

Since it is often impractical to gather data about all of the individual species, we often use *surrogates* that provide a partial estimate of biodiversity. Rare, endemic, or iconic species are commonly chosen as surrogates, but planners may also consider taxa subsets and species assemblages. One type of surrogate is an indicator species whose presence can either imply the presence of other species or indicate the presence of pollution or otherwise hostile environments. Lindenmayer et al. (2000) explain seven different usages of indicator species as surrogates. In some cases, it is possible to use environmental factors to create surrogates for biodiversity. Sarkar et al. (2005) show that readily available information such as elevation, annual mean temperature, precipitation, and soil type provide some indication of the biological diversity.

Margules and Sarkar (2007) differentiate between "true surrogates" and "estimator surrogates." A true surrogate is based on direct observations of biodiversity. An estimator surrogate relies on a biological model to interpret and augment these direct observations. In practice, ideal surrogates should be both quantifiable and estimable.

3. Set Explicit Goals

Planners must set explicit goals for the conservation area network. This involves setting quantitative targets for surrogate preservation while establishing constraints based on costs or area. Planners also need to consider what spatial characteristics might provide the most benefit to the conservation area network.

4. Review the Existing Conservation Area Network

Existing protected areas may help meet some of the biodiversity goals. It may be beneficial to place new conservation areas next to currently protected areas, since this arrangement may have a preferred spatial layout.

5. Select Additional Conservation Areas

The region of study contains a number of potential conservation areas (or cells). The decision maker must choose which sites to place under conservation to meet a variety of goals. Because the number of potential sites can be quite large, software often assists with this step.

6.1.2 Complementarity and Measures of Biodiversity

There are three primary types of biodiversity metrics: α-diversity measures the diversity within a single site, β-diversity considers the diversity between several sites, and γ-diversity describes the total diversity of a region (Whittaker, 1975). When selecting optimal networks, the goal is to maximize the γ-diversity, while simultaneously ensuring that every surrogate is adequately represented.

A geographic cell is *rich* in biodiversity if it contains many of the surrogates that we are studying. Cell by cell *richness* is a very myopic measure of α-diversity. Given cells that are equally rich in biodiversity, it is preferable to select the one that contains surrogates not found elsewhere in the conservation network, to increase the β-diversity.

In this regard, *complementarity* goes one step beyond richness to describe both biodiversity and the extent to which all surrogates are represented. Most complementarity formulations measure the degree to which a new cell can "complement" the others by adding elements of biodiversity that are not already represented in the network. This idea was first implemented in a reserve selection problem by Margules et al. (1988). Complementarity has proven particularly effective in building heuristic solutions (Pressey, Possingham, & Day, 1997). Sarkar (2003) defines 12 different complementarity measures.

Rarity measures the uniqueness of a biological surrogate. For presence-absence (binary) data, surrogates that appear in the fewest cells are rare. Another approach is to consider the expected presence of a surrogate in each cell. Surrogates with small expected values (summed across all cells) are considered rare.

6.1.3 Spatial Characteristics

The spatial characteristics of the reserve network are important for a variety of biological reasons. The equilibrium theory of island biogeography expresses a preference for larger conservation areas which are circular in shape (compact) and well-connected (C. R. Margules & Pressey, 2000). Even though this theory is not universally accepted (Sarkar, 2005) and can be misleading because land between conservation areas is not analogous to oceans between islands (C. Margules, Higgs, & Rafe, 1982), these three criteria still play an important role in conservation area network design.

Current software does not give planners precise control over the spatial configuration, and is not capable of considering spatial configuration at the individual species level of detail. In fact, current methods typically address only a single spatial aspect, *compactness*. Compactness is commonly measured using the perimeter-to-area ratio of the land selected in the conservation area network. Planners often wish to minimize the exposed perimeter of the conservation area network, while maximizing the enclosed area. Among the many biological reasons to minimize the perimeter and related edge effects are: (1) some species may experience higher mortality rates near the edges, (2) an edge may cause an uneven distribution in the species, (3) more favorable conditions on one edge spark asymmetric migrations and (4) an edge can serve as a pathway for invasive species and/or allow unwanted forms of inter-species contact (Fagan, Cantrell, & Cosner, 1999).

An algorithm that prefers a low perimeter to area ratio will naturally select circles over other shapes (compactness is sometimes called *shape*). In addition, it will reject a fragmented network in favor of one that contains a minimal number of clusters. Thus, minimizing the perimeter to area ratio enforces simultaneous preference for both shape and connectivity, making it difficult to address each issue individually. Compactness by itself does not provide a sufficient description of the network geometry. The search can be guided more effectively by simultaneously considering other spatial measures such as connectivity and replication.

Connectivity measures how the conservation areas are interconnected. One direct measure of connectivity is the number of isolated conservation areas (*clusters*) in a network, i.e., separated by a region that is not part of the conservation network. Low connectivity is characterized by the presence of many clusters. Figure 6.1 (image on right) contains an example of a well connected network with few clusters. Connectivity can be improved with the incorporation of *corridors*, i.e., thin segments of protected land that connect isolated clusters.

Well-connected networks are preferred because: (1) some species that thrive on a cycle of migration and repopulation experience population declines in poorly connected reserves (C. R. Margules & Pressey, 2000), (2) poor connectivity reduces the heterogeneity of the habitat available to resident species; leading to their decline over time (C. Margules et al., 1982) and (3) reduced connectivity can affect pollination and alter the composition of local plant populations (Fagan et al., 1999).

In contrast to connectivity, *replication* indicates how many disconnected populations of each surrogate are present in the network. Depending on the circumstances, these populations may not be entirely independent due to other vectors of exchange, such as wind, water, and migration routes, but they are physically separated conservation areas. The lack of connection can serve to limit the spread of contagion or blight, and may prevent network wide extinction if a natural disaster were to devastate one region of the network. Anthropogenic degradation of the environment is another threat that must be considered. Disconnected (and perhaps independent) populations create redundancy within the conservation network, an essential part of risk mitigation.

ConsNet is equipped to handle compactness, connectivity, and replication. By allowing user management of such characteristics, it provides a level of spatial control not previously available.

6.1.4 Economic, Agricultural, and Social Costs

There are a multitude of other factors that can affect the planning process, either as costs or benefits. ConsNet uses a general method for incorporating them into the formulation. In most cases, these additional factors, such as human population, can be assigned on a "per cell" basis. These factors are assumed to be additive, so that the total presence can be computed as the sum across all of the selected cells. This information can be used as part of the objective function to drive the search in favorable directions.

It is important to consider these auxiliary costs, especially those that would disrupt local economies, agriculture, and traditions. Disenfranchising local populations can create resentment which hinders conservation efforts. Even worse, displacing large populations can exacerbate a growing humanitarian crisis of conservation refugees and create civil conflict (Dowie, 2005).

It is equally important to consider the benefits acquired from the conserved areas. Chan et. al (2006) show that it is reasonable to choose a conservation network that maximizes ecosystem services. In their study, they examine multiple properties of the conservation area network, including carbon storage, crop pollination, flood control, forage production, recreation, and water provision.

6.2 Problem Statement and Definition

The conservation area network design problem (CANP) has many variations which depend on the goals of the planners, the format of the surrogate representation data, the structure of the objective function, physical and budgetary constraints, and whether network geometry is considered. The flexible, object-oriented programming in ConsNet and MASTS can easily accommodate these different problem types. ConsNet focuses on an open-ended approach designed to adapt to different situations, a vital characteristic for CANP algorithms (Sarkar et al., 2006). This enables ConsNet to address new types of problems, and to revisit more classical formulations with novel techniques that improve performance or solution quality. Let us first describe the essential components of the conservation network design problem in MASTS.

All CANPs share the following basic characteristics. The study region is partitioned into "cells" or "sites." For each site, there are data on the distribution of appropriate biodiversity surrogates (Lindenmayer et al., 2000; C. R. Margules & Sarkar, 2007) and the potential costs (or benefits) of placing each site under conservation. ConsNet can manage different physical arrangements of cells as illustrated in Figure 6.2. Most often, the region of study is divided into regularly spaced “square” cells which allows savings in computation and memory. Irregular cells require the data set to supply additional information about the cells’ adjacencies, areas, and shared boundaries. A problem’s size is described with the number of cells (n) and the number of surrogates (m). The list of cells is Σ ($\sigma_j \in \Sigma$, j = 1,2,…,n), the list of surrogates is Λ ($\lambda_i \in \Lambda$, i = 1,2,...m). The user may require any subset of cells to be permanently included or excluded from the network, effectively removing these cells from the decision problem. In this case, the problem size is described by n_{active}, the number of cells whose status is not permanently fixed by the user.

For each cell, p_{ij} describes the representation of surrogate λ_i in cell σ_j. In the classical SCP, p_{ij} is a binary *presence/absence* variable, with 0 indicating absence and 1 indicating presence. Alternatively, p_{ij} can be a floating point value representing a probabilistic expectation (Sarkar, Pappas, Garson, Aggarwal, & Cameron, 2004). In this case, the sum of the p_{ij} (over all cells) represents the expected occurrences of the i^{th} surrogate. The representation targets for each surrogate are given by a vector τ_i , i = 1,2,...m.

Each of the possible finite set of solutions, Γ (γ_k, k = 1,2,…, 2^n), can be represented as a boolean vector of length n where selected cells have value 1. For a specific solution, γ_k, ConsNet stores the solution as a bit set (with bit level manipulation). This condensed storage is preferred, and often required, since MASTS may hold several thousands of solutions in memory.

The most common problem statements fall into one of two basic categories: a set cover problem (SCP) or a maximal cover problem (MCP). In the classical SCP (Daskin, 1995) with presence/absence data, the objective is to determine the smallest set of cells such that each surrogate is represented once in the network. When the surrogate data appears as probabilistic expectations, the constraints and objective function are no longer necessarily integer valued, and the related problem is known as the Expected Surrogate Set Cover Problem (ESSCP) (Sarkar et al., 2004). Defining the variable $y_{jk} = 1$ if $\sigma_j \in \gamma_k$ and 0 otherwise, and given a representation target for each surrogate, τ_i, the mathematical programming formulation of the general set cover problem is:

$$\min_{\gamma_k \in \Gamma} \quad \left|\gamma_k\right| = \sum_{j=1}^{n} y_{jk} \qquad \text{subject to} \quad \forall \text{i:} \ \sum_{\text{j}=1}^{n} y_{jk} p_{ij} \geq \tau_i$$

Alternately, the problem can be phrased as a maximal cover problem (MCP), where the goal is to maximize surrogate representation in the presence of auxiliary constraint that limit the number of cells that may be selected. In the most basic formulation of the problem (Church, Stoms, & Davis, 1996), the extent and quality of the cover is measured by the number of surrogates which have met their target representation. Both the SCP and MCP are known to be NP-hard (Nemhauser & Wolsey, 1988).

This dissertation focuses on spatial variants of the basic SCP, trying to find a minimal set cover and simultaneously optimize spatial characteristics of the network. This general class of problems is known as the Spatial Conservation Area Network Problem (SCANP).

The number of variations on these problems grows quickly when considering these extensions:

1. Using different measures of complementarity to assess the extent and quality of the cover

2. Incorporating multiple costs into the statement of the problem

3. Potential tradeoffs between the cost constraints and representation targets

4. Incorporating preferences about the spatial configuration of the network

The size and complex variations of these problems quickly exceed the purview of classical methods. Specifically, it is doubtful that the spatial properties of clustering and replication can be tractably formulated within a classical mathematical programming formulation. Onal and Briers (2003) presented a method that incorporated boundary length into a linear integer program, but this method does not generalize to the clustering and replication problem. Alagador and Cerdiera (2007) propose a linear integer program to find solutions with clusters aggregated around mandatory cells in a proposed conservation area network. This approach is applicable only for a very specific type of problem, and is demonstrated only on a very small problem possessing 496 cells.

From a pragmatic viewpoint, expressing the problem as a precise mathematical program (if one exists) is unnecessary and overly restrictive. Tabu search does not require such a formulation, and makes no distinction between linear or non-linear, integer or mixed-integer problems. Moreover, the heuristic strategies used by the tabu search often incorporate auxiliary information not easily expressed in a classical mathematical programming context. Indeed, some rule based objectives have no numerical equivalent, placing them outside the framework of classical mathematical optimization. For these reasons, the search variables and components are best described in the context of a computer program.

As each solution is evaluated, the raw results are stored in a data object that implements the Results interface. When available and enabled, the following attributes are computed and stored for every solution:

Table 6.1 The attributes included in the Results data structure.

name	type	description
selectedCells	BitSet	a bit level structure that tells which cells indices are selected
nSelectedCells	int	the number of selected cells (cardinality of the solution)
perimeter	double	the total perimeter of the network
area	double	the total area of the network
nClusters	int	the number of geographically disconnected clusters in the network
representationArray	int[] or double[]†	the total representation for each surrogate
totalRepresentation	int or double†	the sum of the representation array
replicationArray	int[]	the replication (number of geographically disconnected instances) of each surrogate
totalReplication	int	the sum of the replication array
costsArray	double[]	the costs (or benefits) summed over the selected cells

†integers are used when the surrogate data are presence/absence, or floating point values rounded into an integer.

Innovative techniques have been developed to make these computations more tractable. Efficient analysis of the spatial configuration requires specialized data structures. To identify clusters, every cell σ_j must have a complete list of its adjacent neighbors (i.e., all other cells sharing any contiguous part of the boundary defining σ_j of length > 0, i.e., shared points or corners are not sufficient). ConsNet stores this information in a doubly linked structure, in which pointers identify adjacent neighbors for each cell.

While area and perimeter are easy to compute, other spatial attributes can be computationally costly. Since the ConsNet neighborhoods contain solutions that result from adding or removing a single cell from the incumbent solution, ConsNet employs fast updates to compute the representation, area, perimeter, costs, clustering, and replication of a neighboring solution. These updates are a critical part of the search, allowing solutions to be evaluated several orders of magnitude faster than building each solution from scratch, with larger problems receiving even more benefit (see Section 6.3).

The attributes presented in Table 6.1 serve as a common foundation for objective evaluations. As specified by the MASTS API, these data are uninterpreted. The objective functions will subsequently interpret these data, extracting the minimum needed for comparing two solutions. Thus far, the problem has been defined without explicitly stating the objective function. The objective function can take many different forms and new capabilities can be added just by creating new objective functions (see Section 6.6).

6.3 Quick Spatial Updates

When changing one cell at a time, spatial properties such as area, perimeter, clustering, and replication can be quickly updated using the previous solution as a reference point. In many cases, the shortcuts can be performed by visiting only a couple of cells rather than re-evaluating the whole network, saving several orders of computational effort. The update strategy depends on which spatial properties are being examined as well as the structure of the conservation area network. For example, better shortcuts may be available when considering clustering exclusively, as opposed to both clustering and replication. Moreover, a rectangular grid structure may permit more efficient shortcuts compared to problems with irregularly shaped cells. Finally, the nature of the update will differ when cells are added or removed.

6.3.1 Area and perimeter

The updates to area and perimeter are identical, regardless of the shape of each individual cell. When a cell is added, its area is simply added to the total area of the network. The perimeter requires additional considerations. Let B_{ij} represent the boundary between any two neighboring cells σ_i and σ_j. Let B_i represent the total perimeter for cell σ_i. Adding a single cell will effectively delete the boundaries between the added cell and its *selected* neighbors. But adding the cell will also create new boundaries between the added cell and its *unselected* neighbors, as well as any boundary that is shared with the exterior of the network. Accounting for all of these factors, the update for the perimeter when adding a cell (starting with the perimeter of the previous solution) can be written in pseudocode:

```
perimeter += Bi
for each selected neighbor j
    perimeter -= 2*Bij
```

It is written this way to reduce the number of lookups for B_{ij}, which may involve a linked list. Since most of the network is usually *unselected*, this approach involves fewer lookups for B_{ij} compared to other equivalent logic. The perimeter update for removing a cell is similar to the approach above.

6.3.2 Clustering Structure and the Number of Clusters

Updating the number of clusters (cluster number) occasionally requires a partial graph traversal of the *implied graph* associated with the current network to find the extent of one or more clusters. Because this traversal can be costly (especially when the clusters are large), traversal should be used only as a last resort. In many cases, the change in clustering can be determined simply by examining the immediate neighbors of the single cell that has just been selected or deselected. Depending on which neighboring cells are selected, simple geometric arguments can be used to determine the new cluster number. For a square grid in particular, this type of shortcut could eliminate a huge number of potential graph traversals. This section describes the logic used to determine whether a shortcut may be used.

First, consider a situation where a partial graph traversal is required to determine the extent of a cluster and properly update the cluster number. The following example uses a regular square grid, but the concept is also valid for cells with irregular shapes. On a square grid, the neighbors of a given cell are those cells that share a non-zero boundary length in the cardinal directions north, south, east, and west. Figure 6.3 shows a *connector cell* on a square grid (cell x) joining two darkened selected cells. Cells marked with a *?* may or may not be currently selected. Removing cell x could create two clusters, and adding cell x could fuse two clusters into one. However, the two selected neighbors may also be connected by an alternate path. To determine if cell x will alter the clustering, a graph traversal is required to find the extent of the two clusters. In Figure 6.3, the traversal could start with the western neighbor of cell x and identify all neighboring cells that are selected. For each selected neighbor, the traversal will recursively visit any selected neighbors that have not been previously visited until the cluster of selected connected cells has been fully mapped. If this traversal includes the *eastern* neighbor, then the eastern and western neighbors reside in the same cluster regardless of the selected status of cell x.

In Figure 6.4, cell x is a connector cell for up to 4 clusters. It could require three partial graph traversals to determine if all 4 neighbors of cell x are connected through other paths. If traversing the clusters associated with three of the neighbors does not find a connection to the fourth neighbor, then the fourth neighbor is not connected to the three other neighbors.

Consider a subgraph $G(\sigma_{ik})$ created by the selected neighbors of connector cell σ_i for a given solution Γ_k. A connector cell is identical to a cut vertex (or articulation point) in graph theory. In a general setting, adding or removing a connector cell will require a graph traversal to determine how to update the clustering structure. On a square grid, if two or more neighbors of a cell x are selected, then cell σ_i could be a connector cell if those neighbors are not otherwise connected. Luckily, a square grid's regular structure provides quick tests to determine alternate connectivity. In non-square tessellations, such tests may not be available.

If a cell is not a connector cell, then it must be one of two other types. If the neighbor subgraph $G(\sigma_{ik})$ is empty, then cell σ_i is an i*solated* cell with no selected neighbors. Otherwise, the cell is an *annex* to a currently existing cluster. Figure 6.5 illustrates disconnected, annex, and connector cells for an arbitrary set of tessellating cells.

Toggling the status of an isolated cell will increment or decrement the cluster number. Toggling an annex cell changes the size of a cluster but does not change the cluster number. Neither of these two cases require graph traversals. Toggling a connector cell may require a graph traversal to determine how the clustering structure changes.

For arbitrary tessellating cells, determining the character of a specific cell is complicated by the possibility that cells may touch at a single point (such cells are not considered adjacent). In a regular square grid, the single point boundary locations are known and shortcuts are readily available. In an arbitrary non-square grid, the location of zero boundary vertices is unpredictable, so shortcuts may be limited. This decision about when to use a shortcut is part of the ConservationGrid interface, so that different types of grids may implement different shortcuts whenever they are available.

When designing a shortcut, the following facts should be noted:

- In most applications, the graph will contain more white cells (unselected) than black cells (selected).
- If all the neighbors are unselected, then the cell is automatically an isolated cell, regardless of cell geometry.
- If exactly one neighbor is selected, then the cell is automatically an annex cell, regardless of cell geometry.
- In most applications, compactness of selected cells is preferred. The most frequent case is that all of the neighbors will be unselected (an isolated cell), and the second most frequent case is that all neighbors will be selected. Annex cells are the third most frequently encountered case, and the rarest case is that the cell is a connector.

On a square grid, it is easy to classify cells as isolated, annex, or connector by considering at the 8 cells that surround the central cell (N, NE, E, SE, S, SW, W, NW). This expanded subgraph $G'(\sigma_{ik})$ will consist of selected cells that are neighbors and selected cells that share a zero boundary vertex (*improper neighbors*). In this sense, $G(\sigma_{ik})$ is a subgraph of $G'(\sigma_{ik})$. Expanding G in this manner slightly changes the definition of a connector cell; σ_i is a connector cell if the graph $G'(\sigma_{ik})$ contains at least two neighbors of σ_i but the set of selected neighbors is not connected in $G'(\sigma_{ik})$. In essence, by examining a larger extent of the local graph structure, some cells that would have required graph traversals can now be categorized as annex cells. Figure 6.6 shows a few different arrangements on a square grid, and classifies the cells as isolated, annex, or connector.

For a square grid, there are 256 possible combinations of on/off settings for the 8 surrounding cells. An exhaustive enumeration shows that 16 of these cases lead to isolated cells and 117 lead to annex cells. Thus, more than half of the 256 combinations are patterns where the change in clustering is easily calculated. As mentioned previously, some of these configurations will be more common than others. When compactness is rewarded by the objective function, the resulting networks are characterized by large areas of empty white space, and large clusters with clean boundaries. In a network like this, virtually 100% of the cells can be classified as isolated cells or annex cells, meaning that cluster traversals are rarely required.

In summary, the fast update algorithm for clustering involves the following steps:

```
nSelected = the number of selected neighbors of $\sigma_i$ (obtained while updating the
perimeter)

if(nSelected == 0)
        an isolated cell; the clustering increases by one if the cell is added,
        decreases by one if the cell is removed

else if(nSelected == 1)
        an annex cell; clustering does not change.

else
        let the abstract grid type decide if the cell is annex cell or a connector
        cell.  if the cell is an annex cell, the clustering does not change.
        otherwise, proceed to deal with the connector cell.

// the cell is a connector cell and grid traversal is required

nSeparateNeighboringClusters = 0

for each selected neighbor j {
        if(already visited j)
                continue
        else
                traverse the cluster starting at cell j
                marking which cells are “already visited”
                nSeparateNeighboringClusters++
} // end for each selected neighbor

// adding a connector cell?  that joins one or more clusters
nClusters = previousNClusters – nSeparateNeighboringClusters + 1

// removing a connector cell? that creates more clusters
nClusters = previousNClusters + nSeparateNeighboringClusters - 1
```

If cell σ_i is a connector cell that joins three separate (“unique”) clusters, then turning it off will create three new clusters where previously there was only one. So the number of clusters actually changes by two. This explains the plus or minus one on the line where the number of clusters is updated.

In most cases, the clustering can be updated with little or no effort. In the case of a square grid, traversals can be almost eliminated. However, all of these shortcuts are invalid once we have to consider replication.

6.3.3 Updating Surrogate Replication

Updating the replication requires recording both the number of clusters and which surrogates are contained in each cluster. The replication of a surrogate is the (integer) number of separate clusters in which that surrogate can be found with non-zero expectation. Without special data structures to track the contents of each individual cluster, there is no general way to avoid a partial graph traversal when updating the surrogate replication. The memory required for such data structures would be prohibitively expensive and cumbersome considering the number of different alternatives that are visited during the search.

Updating the replication is straightforward but time-consuming. While traversing the surrounding clusters, the search must also look at the surrogate representation in each cell. If replication information is not required for every surrogate, then it is possible to save computational effort by performing the analysis on a specified subset of the surrogates.

During the update, the concept of isolated, annex, and connector cells still applies, but the number of available shortcuts is greatly reduced. Updating replication for an isolated cell (the most common type of update) is trivial where the replication increases (decreases) by one unit for each surrogate that is present in the newly selected (unselected) cell.

An annex cell is added or removed to a currently existing cluster. While adding an annex cell does not change the clustering, it could change the replication depending on what surrogates exist in the rest of the cluster. Since these data are not stored, a traversal of the cluster will be required. When adding a connector cell, one or more traversals could be required.

The first step of the update is to identify the number and extent of separate neighboring clusters adjoining the central cell σ_i. For an annex cell, there is only one such cluster. For a connector cell, nSeparateNeighboringClusters is the number of clusters that will be fused (created) if the cell is added (removed). This part of the algorithm is identical to the technique used to update the number of clusters.

Suppose that a connector cell serves as the connection between three separate clusters. Removing this cell will increase the number of clusters by two; one cluster is "destroyed" and three are created. The replication for some species could increase by one or two units. If the only instance of a surrogate is in the connector cell, the replication would *decrease* by one. The algorithm will update the clusters and the replication simultaneously (since the quantities are closely linked).

The following algorithm handles isolated, annex, and connector cells.

```
nSeparateNeighboringClusters = 0
int[] newReplication  // (0->nSurrogates)
int[] oldReplication  // (0->nSurrogates)
int[] localReplication = new int[nSurrogates]  // all zeros
int[] trackTheseSurrogates = pre-defined from the user
int[] myClusterIndices = null  // temp variable to define clusters
i = index of the cell that is being turned on or off

for each selected neighbor j {
        if(already visited j)
                continue
        else
                myClusterIndices = traverse the cluster starting at cell j
                (also marks which cells are "already visited")

                updateLocalReplication(myClusterIndices, localReplication)

                nSeparateNeighboringClusters++
} // end for each selected neighbor

// adding a cell?  use this code
nClusters = previousNClusters – nSeparateNeighboringClusters + 1
for(int k : trackTheseSurrogates) {

        newReplication[k] = oldReplication[k] – localReplication[k]

        // adjustment!!
        if(localReplication[k] > 0 || grid.getRepresentation(i,k) > 0)
                newReplication[k]++
}

// removing a cell?  use this code
nClusters = previousNClusters + nSeparateNeighboringClusters - 1
for(int k : trackTheseSurrogates) {

        newReplication[k] = oldReplication[k] + localReplication[k]

        // adjustment!!
        if(localReplication[k]>0 || grid.getRepresentation(i,k)> 0)
                newReplication[k]--
}
```

The localReplication array keeps track of the replication in the separate neighboring clusters. The k^{th} entry corresponds to the k^{th} surrogate. The values will be between 0 and the nSeparateNeighboringClusters (the number of replications is always less than or equal to the number of clusters). For each neighboring cluster, the updateLocalReplication() method will increment localReplication[k] if the kth surrogate is contained in the cluster. This method only examines the surrogates listed in the trackTheseSurrogates array. It returns early if and when it finds a replication for every surrogate.

As noted above, the actual change in replication is going to be one less than the value in the localReplcation array. Also, the localReplication does not account for any contribution to replication made exclusively by the central cell. The "adjustment" line accounts for both of these cases. The bookkeeping is somewhat intricate, but the replication has been properly updated.

6.3.4 Using ThreadLocal Variables

Several of these algorithms mention auxiliary data structures that track cluster indices, surrogate replication, and which cells have been visited in a traversal. These data structures can require a lot of memory. One way to cut down on the overhead of memory allocation is to re-use thread local variables. These variables include:

- the traversal stack – a stack of cell indices used during the recursive graph traversal
- the tracking bit set – used to track which cells are currently selected, and which cells have already been visited
- the current cluster indices – an array containing the indices of member cells as a cluster is mapped out
- the local replication – an array that keeps track of the replication for each surrogate

The use of thread local variables is responsible for a major increase in performance (see Section 2.9.1).

6.4 Basic Tabu Search Features

Integrating MASTS with ConsNet required definitions for the decision variable, moves, neighborhoods, tabu lists, and objective functions. As a client of MASTS, ConsNet can also take advantage of powerful strategies such as adaptive tabu search (ATS), dynamic neighborhood selection (DNS), and rule based objectives (RBOs). This section describes the basic components of the tabu search. An extensive description of DNS is presented in Section 6.5, and the various rule based objectives are described in Section 6.6. Throughout this discussion, a number of heuristic parameters have been set by careful observation and experimentation.

6.4.1 Decision Variable and Moves

In MASTS, the decision variable(s) can take on a variety of forms. For ConsNet, the decision variable consists of a bit vector of size *n*. The Java BitSet class was used to implement the bitwise storage. The hash code is generated from all of the bits.

The basic move consists of flipping a the status single bit, either including or removing a cell from the conservation area network. Moves that consider flipping more than one bit are occasionally used to diversify the search. Because the entire set of bit-flip moves is potentially huge, the moves are strategically organized into neighborhoods of reasonable size, as discussed in Section 6.5.

6.4.2 Reactive and Adaptive Tabu Search

The tabu attributes are defined for individual cells. When the search performs a toggle move for a specific cell, any move that alters this cell is considered tabu for the number of iterations specified by the tabu tenure. The tenure is allowed to vary between an upper and lower bound based on the number of active cells in the problem (n_{active}). Experiments have shown that a lower bound of $0.001*n_{active}$ and an upper bound of $0.003*n_{active}$ work best in a variety of situations. This means that between 0.1% and 0.3% of the possible single bit-flip moves will be considered tabu during normal search operation (regardless of how these moves are organized into neighborhoods).

Adaptive tabu search (ATS) and reactive tabu search (RTS) both prescribe methods to vary the tabu tenure to intensify and diversify the search. RTS (Battiti & Tecchiolli, 1994) manages the tabu tenure by detecting repeats of solutions during the search. However, in the practical sized conservation network problems, observing just *one* solution repetition using ConsNet is extremely unlikely given the size of the solution space and the shuffling behavior of the DNS. Thus, RTS is not used.

Instead, ConsNet uses ATS (Dell'Amico & Trubian, 1993), a technique still highly favored (Lambert et al., 2007). ATS increments or decrements the tabu tenure depending on whether the new incumbent solution represents an disimproving or improving move, respectively. However, a single unit change in the tabu tenure would be too small considering the longer tenures being used by ConsNet. Consequently, ConsNet modifies the tenures in the following way: (a) when the search accepts a solution that is superior to the previous incumbent, the tenure is decreased by the number of *consecutive* improvements the search has accepted just prior to the current iteration; (b) if the search accepts a non-improving solution, the tenure is increased by the number of *consecutive* non-improvements that have transpired prior to the current iteration; and (c) no changes are made to the tabu tenure if the search accepts a solution that is equivalent to the previous incumbent. This cumulative adjustment to the tenure encourages appropriate periods of intensification and diversification to occur.

6.5 Neighborhoods and DNS in ConsNet

6.5.1 Neighborhood Definition

The "atomic" move in ConsNet toggles a single bit in the incumbent solution, effectively adding or removing a single cell from the network. A neighborhood that examines the whole network would create a number of moves/neighbors equal to the number of active cells, n_{active}. In problems of practical size, the search progress would be untenably slow if all neighbors were evaluated at each iteration; thus the cells are reorganized into smaller management groups called "supercells". These smaller supercells serve as the general use neighborhoods (in the context of large neighborhood decomposition).

In ConsNet, users are allowed to include or exclude specific cells permanently, effectively reducing the size of the problem. The moves created by neighborhoods are not allowed to alter these "inactive" cells. When describing the problem size, $\mathrm{n}_{\mathrm{active}}$ refers to the number of active cells, which may be less than or equal to the number of cells in the entire network. The neighborhood strategies below are based on the number of active cells in the problem.

The number of supercells (and the size of each supercell) depends on the number of active cells. As expected, there will be *many* search related properties that depend on problem size. The majority of these properties are explicitly defined and explained throughout this section. Table 6.15 (on page 179) shows how all of these properties vary with problem size. This table presents the "big picture" and each property will be fully discussed below. This information appears in one table so that the behavior of the search for different problem sizes can be quickly ascertained.

For example, a problem with 50,000 active cells has a *target supercell size* (*S*) of 5,000 cells. Each supercell will form a single *general use* neighborhood that generates *S* atomic moves. These general use neighborhoods will be used in most iterations. The target supercell size can greatly affect the search performance. Large supercells require more evaluations per iteration, possibly slowing search progress. However, using small neighborhoods will inevitably cause myopic, inefficient choices to be made. As the search proceeds, these errors can accumulate and adversely affect solution quality. There are several mechanisms that can offset this search myopia. First, dynamic neighborhood selection can improve search performance by making more informed choices and occasionally performing intensifying moves that examine larger neighborhoods. It has also been established that rule based objectives can be used to overcome the damage of earlier bad choices (Ciarleglio et al., 2007). Thus, it is possible to secure performance benefits from small neighborhoods without sacrificing solution quality.

Two types of supercell neighborhoods will be created:

(1) geographic supercells (group contiguous cells)

(2) richness supercells (group cells with similar richness)

Neighborhoods organized in this manner will provide measurably different behavior for most common objective functions. The DNS strategy will be able to detect and exploit this behavior. The geographic supercells are likely to capitalize on the spatial correlation that is present in most features of the problem. For instance, cells containing a specific surrogate or cost feature are likely to be near other cells containing that same feature. If the objective function rewards this feature, then some supercells will emerge as particularly influential. The richness supercells behave in a similar manner, and may be preferred when total richness is a large component of the objective function.

The geographic and richness supercells will be the "general use" neighborhoods. In addition, we define some "special use" neighborhoods to use when the search progress begins to stagnate. The details for these neighborhoods are provided in this section.

6.5.2 General Use Neighborhoods

The general use neighborhoods are supercells organized by geographic contiguity and richness. All supercells will be the same size S, based on the number of active cells n_{active}:

$$S(n_{active}) = \begin{cases} n_{active}/5 & n_{active} \leq 5000 \\ n_{active}/10 & 5000 < n_{active} \leq 100,000 \\ n_{active}/20 + 5000 & 100,000 < n_{active} \leq 200,000 \\ n_{active}/30 + 8334 & 200,000 < n_{active} \end{cases}$$

The supercells are required to be the same size because this simplifies planning for the dynamic parts of the search, which rely implicitly on neighborhoods of uniform size. For example, declaring an attribute tabu in a neighborhood that contains 10 neighboring solutions has different implications when compared to a neighborhood of 1000 solutions. Similarly, if the neighborhoods have different sizes, then some will be inherently more myopic than others, and measures of neighborhood performance would have to account for differences in size. Finally, from a user perspective, neighborhoods of the same size create a sense of predictability.

Notice that the supercells become a proportionally smaller part of the entire network as the problem size grows. This reflects a belief that for larger problems, the computational benefits conferred by small neighborhoods outweigh the difficulties associated with myopic neighborhoods.

By design, supercells will not include any cells that have been permanently included or excluded from the conservation area network. These cells are considered inactive. Neighborhoods are not allowed to create moves that alter the status of inactive cells.

The fixed supercell size means that *at least* B supercells will be required to cover all of the active cells in the network, where B is defined:

$$B(n_{active}) = \left\lceil \frac{n_{active}}{S(n_{active})} \right\rceil$$

Since the search chooses one supercell neighborhood at each iteration, B should be interpreted as the *minimum number of iterations required to visit each cell*. This interpretation guides the timing used in the dynamic neighborhood selection. The spatial supercells will be arranged so that exactly B spatial supercells cover all of the active cells. The richness supercells will be built using only cells with richness greater than zero. Since the richness supercells are also size S, there may be fewer than B richness supercells in the problem, and the total number of supercells will be 2B or less. Below, we discuss constructing both types of supercells, effectively creating two neighborhood systems.

The construction of spatial (contiguous) supercells is somewhat complex. Since the supercells are required to be the same size, it is possible that a supercell may not be completely connected and some cells may appear in more than one supercell (overlap). However, no cell may appear in more than two spatial supercells. The construction technique starts with an active cell that does not belong to any supercells. The "boundary" of the supercell grows one layer at a time by including contiguous cells that do not appear in another supercell. If the algorithm runs out of such connected boundary cells before the supercell achieves the required size, then it selects a disconnected free cell, and the supercell growth continues. If all cells are covered, usually near the end of the construction process, then nearby contiguous cells that overlap with no more than one other spatial supercell are added. Finally, in the rare cases that no such contiguous cells are available, cells that appear in no more than one other supercell are added. The construction process is designed to manage cases when the study region is composed of disconnected regions. It is also guaranteed to create exactly B supercells.

Another independent procedure creates the richness based supercells. Like the spatial supercells, the richness supercells are required to be of size S. The richness supercells are constructed using only the cells with richness greater than zero. As a result, there may be fewer richness supercells than spatial supercells. The cells with nonzero richness are sorted and arranged into supercells of size S, so that the richest cells appear in the first supercell, and the poorest cells appear in the last supercell. To meet the size requirement, some cells are included in more than one richness supercell, but the number of overlapping elements in any one supercell are kept to a minimum.

6.5.3 Special Use Neighborhoods

The general use neighborhoods described above are limited to moves that change only one cell at a time. Although the search can make significant gains using these moves, progress will eventually stall out. Special use neighborhoods can aid in both intensification and diversification. For intensification, a single large neighborhood containing *all* single cell moves may help overcome the myopia of the supercell approach. For diversification, different neighborhoods can supply a variety of "shake-up" moves designed to drive the search to regions in the solution space. These diversification neighborhoods could create moves based on cell richness or the spatial characteristics of the solution. Figure 6.7 presents the hierarchical organization of special use neighborhoods. For the spatial rearrangement neighborhoods, examples are shown on a square grid, but the definitions directly extend to networks of arbitrary shape and connectivity.

Although the DNS automatically chooses when to use the special use neighborhoods, users may override the DNS through the GUI and force any of these neighborhoods to be used while the search is running. Moreover, users can deactivate specific neighborhoods so that the search is not permitted to use them. This provides an important capability for expert intervention in the search. Ordinarily, this level of control might be considered "too advanced" for casual users, but the special neighborhoods defined for this problem are intuitive and have predictable effects on the search. Users familiar with the special use neighborhoods can observe the search and decide which neighborhoods might be appropriate. None of the special use neighborhoods described below will alter cells that are permanently included or excluded from the conservation area network.

One Large Neighborhood. This neighborhood considers toggling the status of every cell individually, creating one move per active cell. This neighborhood contains all of the possible bit flip moves, and is *the* natural choice for a general intensification neighborhood. This neighborhood can overcome the myopia associated with supercells, but with a high evaluation cost. If repeated use of this neighborhood does not find better solutions, then the search may have reached a point where single cell moves are no longer effective. Although the DNS will use occasionally use this neighborhood, users may request this neighborhood anytime they feel that the search can benefit from intensification.

Cluster Deletion Neighborhood. This neighborhood creates several moves by individually deleting clusters that contain more than one cell, creating one move per cluster. This neighborhood may be useful when the objective is to reduce the number of clusters. Perhaps the deleted cells will be added into other clusters.

Cluster Shrink Neighborhood. This neighborhood creates several moves by "shrinking" all clusters that contain more than one cell (one move per cluster). The shrink operation involves removing the boundary cells of the cluster. A boundary cell is a cell that belongs to the cluster and:

- has a neighbor that is not included in the conservation area network, or
- shares and edge with the boundary of the study region

Figure 6.8 illustrates the cluster shrink neighborhood. For large compact clusters, this neighborhood simply removes the outer layer of the cluster, shrinking the boundaries. For a small cluster that has lots of holes and crevices, the shrink operation could remove a large portion of the cluster. This neighborhood will only produce moves that affect *at least* two cells. If the cluster shrink deletes the entire cluster, then no move is produced for that cluster. Since the removed cells will be marked as tabu, it is likely that they cells will be relocated to another cluster.

Burr Removal Neighborhood. As illustrated in Figure 6.9, this neighborhood creates one move that considers the simultaneous deletion of all burrs. A burr is any cell that is in the network that has *exactly* one neighbor also included in the network. Removing the burrs can improve the compactness of the current conservation area network. This neighborhood will only generate a move if one or more cells is classified as a burr.

Singlet Removal Neighborhood. As illustrated in Figure 6.10, this neighborhood creates one move that removes all single-cell clusters (“singlets”). If there are no such cells in the current solution, then this neighborhood does not generate any moves.

Cluster Expand Neighborhood. As illustrated in Figure 6.11, this neighborhood creates several moves by expanding the boundary of each cluster in the network (one move per cluster). The expansion move will add cells that neighbor the cluster but are not currently selected. Moves will only be generated for clusters that contain two or more cells, and each move must change the status of two or more cells. Otherwise, no move is generated for that cluster.

Hole Filler Neighborhood. As illustrated in Figure 6.12, this neighborhood generates one move by filling all of the “holes” in the network. A hole is a single cell that is not selected, but whose neighbors are all selected. This does not include cells along the boundary of the study region.

Crevice Filler Neighborhood. As illustrated in Figure 6.13, this neighborhood generates one move by filling all of the “crevices” in the network. A crevice is a single cell that is not selected, but has three at least three neighbors selected and at least one neighboring cell that is either not selected or belongs to the exterior of the study region.

Singlet Expand Neighborhood. As illustrated in Figure 6.14, this neighborhood generates one move by expanding the single cell clusters (“singlets”). This move toggles all the neighbors of the single cell cluster so that they are included in the network.

Spatial Composite Neighborhood – Add Cells. This composite neighborhood will generate all of the moves described by the cluster expand, hole filler, crevice filler, and single expand neighborhood. These neighborhoods are grouped together because they all *add* cells to the network.

Spatial Composite Neighborhood – Remove Cells. This composite neighborhood will generate all of the moves described by the cluster deletion, cluster shrink, burr removal, and singlet removal neighborhoods. These neighborhoods are grouped together because they all *remove* cells from the network.

Spatial Composite Neighborhood – Full Rearrangement. This composite neighborhood will generate all of the moves described in the neighborhoods above, creating some moves that add cells and some moves that remove cells.

Richness Swap Neighborhood. Unlike the neighborhoods listed above, this special use neighborhood does not involve the spatial properties of the solution. This neighborhood generates several moves that remove rich cells from the network, and replaces them with rich cells that were not in the network. Each move consists of several such swaps; the number of swaps, $n_{swaps,}$ is equal to a problem size property, *min size shakeup move*. This property gives an indication of how large a shakeup move should be, and a proper formula appears in Section 6.5.4.2.

The swap moves are assembled in the following manner, and the number of moves depends on how many cells are currently selected. Iterating through the cells (starting with the richest cells), generate two lists ordered by decreasing richness. The first list contains the cells that are currently selected and the other list has cells that are unselected. Swaps are created by pairing the last element from the selected list with the first element from the unselected list, in effect swapping the poorest selected cell for the richest unselected cell. As swaps are generated, the elements are removed from both lists. The process continues until n_{swaps} pairs are collected or until no more pairs are available.

6.5.4 ConsNet DNS Strategies

As presented in Table 6.2, seven different neighborhood selection strategies are available in ConsNet. All of them use the same basic procedure for selecting general neighborhoods. They differ in the way that they employ the special neighborhoods to escape local optima. The simplest strategies (*basic*) have no escape mechanisms and rely on the general use neighborhoods or the single large neighborhood. The *standard* strategy uses the general neighborhoods for most iterations, but intensifies the search using the single large neighborhood. The *standard (escape enabled)* uses the large neighborhood to intensify, but occasionally uses the richness swap neighborhood to escape local optima. The *advanced* strategies use the spatial rearrangement neighborhoods to escape local optima. There are three advanced options so that users can better control the nature of the shakeup moves. For any of these strategies, users may override and force the use of a specific neighborhood, or disable certain neighborhoods from being automatically selected.

Table 6.2 Available neighborhood selection strategies

DNS Strategy Name	Description
basic	Select from among the general neighborhoods based on quality and recency of use.
basic (use large nbhd only)	Use nothing but the "one large neighborhood," especially useful for intensification
standard	Select from the general neighborhoods, but occasionally use the large neighborhood to intensify the search
standard (escape enabled)	Select from the general neighborhoods, occasionally use the large neighborhood to intensify, and the richness swap neighborhood to diversify
advanced (escape with spatial nbhds – remove cells)	Similar to the standard, but use the spatial neighborhoods as the primary escape mechanism. Only spatial moves which *remove* cells will be considered
advanced (escape with spatial nbhds – add cells)	Similar to the standard, but use the spatial neighborhoods as the primary escape mechanism. Only spatial moves which *add* cells will be considered.
advanced (escape with spatial nbhds – add and remove)	Similar to the standard, but use the spatial neighborhoods as the primary escape mechanism.

In the following sections, the rules for selecting the general neighborhoods and special use neighborhoods will be discussed in detail.

6.5.4.1 General Neighborhood Selection

All of the DNS strategies use the same basic method to determine which general neighborhood will be used during the next iteration of the search. This general DNS strategy selects neighborhoods based on quality and recency. The goal is to assure that the search occasionally visits each neighborhood, favoring those that have recently yielded high quality solutions. Each neighborhood receives a desirability score based on two properties: (1) a moving average of the number of non-tabu improving moves that have been observed in the neighborhood and (2) how recently the neighborhood has been used.

In addition to balancing quality and recency, we must choose how heavily to weigh the historical performance of the neighborhood in the moving average. The historic weight ($w_{historic}$) is a number between 0 and 1; values close to one place more emphasis on the historical performance and have more inertia against changes in the average value. Values close to zero mean that the moving average is largely determined by the performance on the most recent iteration. Each time neighborhood i is used, the performance index p_i for the neighborhood is updated as follows:

$$p_{i,new} = w_{historic} p_{i,previous} + (1 - w_{historic})\ nImprovingMovesNotTabu$$

Here, *nImprovingMovesNotTabu* is the number of improving moves that were not tabu when the neighborhood was last used. We do not count improving moves that were tabu; in some situations, this can encourage the search to repeatedly use a neighborhood where all of the improving moves are tabu, in which case the search will have to accept a series of disimproving moves. A value of 0.1 was chosen for $w_{historic}$, so that the performance index can change significantly each time it is updated, but retains some information about previous performance. It has been determined that the search performance is somewhat sensitive to $w_{historic}$. However, the value of 0.1 seems to work well for a variety of problems.

The desirability score assigned to each neighborhood also depends on how recently the neighborhood has been used. If there are k general neighborhoods, then one heuristic approach might consider visiting each neighborhood at least once every 3*k iterations. Suppose r_i is the number of iterations that have passed since neighborhood *i* was last used. We define the desirability of neighborhood *i* as follows:

$$d_i = \frac{p_i}{\max\limits_i(p_i)} + \frac{r_i}{3k}$$

The first term represents the quality of the neighborhood. It is normalized by the highest performance index among the neighborhoods, and ranges from 0 to 1. The second component represents the recency, and ranges, approximately, from 0 to 1. The neighborhood with the highest desirability score will be chosen for the next iteration.

6.5.4.2 Special Use Neighborhood Selection

Special neighborhoods can be used for both intensification and diversification. Since it is incredibly rare that solutions are visited twice, the diversification process should not rely on methods that use repeated solutions to detect chaotic attractor basins. Instead, the escape methods will be invoked when the search has reached a stagnation point. A measure of stagnation that has proven effective in a wide variety of situations is the number of consecutive non-aspiring iterations, i.e., the number of iterations that have passed since the best solution has been improved. The timing of the escape reactions should depend on the problem size. Schedules can be used to coordinate the usage of different neighborhoods.

Each schedule consists of two parameters, a threshold and a recovery (see Figure 6.15**)**. The threshold is the number of consecutive non-aspiring iterations that will trigger a response, causing the DNS to select a specific neighborhood. Once a response has occurred, the recovery is the number of iterations that must pass before the search can use this neighborhood again. The values for the threshold and recovery are adjusted with the problem size; both are specified as multiples of B, the minimum number of iterations required to visit every cell in the conservation network using the general neighborhoods. In this system, the timing of escape events is based on the number of opportunities each cell has had to change its on/off status. After 40B iterations, we can estimate that each cell may have had 40 opportunities to change status (assuming the neighborhoods are given equal preference).

The program will organize the special use neighborhoods on four schedules, shown in Table 6.3.

Table 6.3 Timing for the various schedules used by the DNS.

schedule	**threshold**	**recovery**
1	40*B	42*B
2	100*B	200*B
3	200*B	400*B
4	400*B	1200*B

Schedule 1 is associated with the very large neighborhood, which will be used fairly often to help intensify the search. This neighborhood does not disrupt the solution, and thus does not require significant recovery time. When triggered, the DNS repeats the large neighborhood several times in order to thoroughly explore the current location. The number of repeats, *nRepeatsLargeNbhd*, is assigned to be 2*B (yielding the recovery of 22*B). If the search is in a promising basin, this procedure usually finds a new best solution, resetting the counter for all of the schedules. While new best solutions continue to be found, the search will not progress to schedule 2.

The threshold on schedule 2 permits two schedule 1 events to occur before an event can be triggered on schedule 2. Once an event is triggered on schedule 2, the recovery period allows two cycles to pass on schedule 1 before triggering another schedule 2 event. If a new best solution is found during this recovery period, then it could be even longer before another schedule 2 response occurs. The threshold and recovery on schedule 3 are twice that of schedule 2. The long recovery time on schedule 4 makes it ideal for major shakeup events. Once a schedule 4 event occurs, the recovery time allows four cycles on schedule 2 before triggering another schedule 4 event, and two cycles could occur on schedule 3 before triggering another schedule 4 event.

With the schedules and neighborhoods defined, a strategy can be completely defined by associating special use neighborhoods with specific schedules, demonstrated in Table 6.4. It is not required that a special neighborhood be assigned to every schedule. For instance, in the advanced strategies, it is assumed that schedule 3 is too early for the richness swap neighborhood, given that schedule 2 contains a fairly significant shakeup move.

Table 6.4 The DNS strategies assign different neighborhoods to the various schedules. Entries containing a “--" do not have a special use neighborhood assigned to that schedule.

DNS Strategy	schedule 1 nbhd	schedule 2 nbhd	schedule 3 nbhd	schedule 4 nbhd
basic	--	--	--	--
basic (use large nbhd only)	--	--	--	--
standard	one large nbhd	--	--	--
standard (escape enabled)	one large nbhd	--	--	richness swap
advanced (escape with spatial nbhds – remove cells)	one large nbhd	--	spatial composite remove cells	--
advanced (escape with spatial nbhds – add cells)	one large nbhd	--	spatial composite add cells	--
advanced (escape with spatial nbhds – add and remove)	one large nbhd	--	spatial composite add and remove	--

The three advanced strategies differ only in the types of spatial rearrangement moves that they create. Depending on the constraints in the objective, users may not want to consider removing (or adding) large groups of cells because it could force the search into an infeasible region. Thus, several options are provided.

The spatial neighborhoods described above produce a variety of moves, some of which affect only a couple of cells. However, the DNS selects these neighborhoods with the intention of shaking up the solution. To force the search to accept a large move, the DNS will filter out all moves that are smaller than the *min size shakeup move*. Here the size of the move is the number of cells that change status. The *min size shakeup move* is a property of the problem size (listed in Table 6.15 on page 179), defined as:

$$min\ size\ shakeup\ move = 5 + \left\lfloor \frac{n_{active}}{3000} \right\rfloor$$

In Section 6.8.6, we will examine the performance of the various neighborhood selection strategies for a variety of common objectives, and ascertain when each one might be most effective.

6.6 Problem Objectives

Two different types of objectives have emerged in ConsNet. The first category consists of pre-defined objectives that have been developed to target specific problems such as the SCP and some of its spatial variants. These pre-defined objectives take full advantage of the flexibility of rule based objectives. They may contain intricate comparison rules, perhaps using locally intransitive behavior to maximize performance on the given problem. Pre-defined objectives are powerful but singular in their purpose. They are easy to set up and use, but they are not highly configurable.

To provide more flexibility, ConsNet also allows users to define simple multi-criteria objectives through the GUI. These user-defined objectives are usually less complex (and perhaps less powerful) than the pre-defined objectives, but allow more precise guidance of the search. The final version of ConsNet will have a suite of pre-defined and user-defined objectives. Experienced users will find it convenient and advantageous to switch between objectives during the search in order to improve different solution attributes. This interactive optimization has been amply demonstrated in previous efforts and has clearly shown itself to be one of the most effective current features in ConsNet.

Objectives are usually defined relative to some representation target τ_t. When writing a rule based objective in code, the programmer has unlimited flexibility to create a scoring/ranking system. However, when creating a new objective through the GUI, users are limited to the following attributes:

Table 6.5 The attributes available to construct user-defined objectives for a specific target set τ.

name	**description**
nSelectedCells	the number of selected cells (cardinality of the solution)
perimeter	the total perimeter of the network
area	the total area of the network
nClusters	the number of geographically disconnected clusters in the network
shape (perimeter to area ratio)	perimeter divided by area
size (average cluster size)	the area divided by the number of clusters
totalRepresentation	the sum total of surrogate representation in the network
totalReplication	the sum total of surrogate replications
replication for surrogate i	the replication for a specific surrogate
totalSurplus for τ	the sum of all surrogates exceeding the target (positive)
totalDeficit for τ	the sum of all the deficient surrogates (positive)
surplusSlack for τ	a measurement of target surplus, the sum of surrogate representation exceeding the target, but only counting the first unit of surplus
nDeficientSurrogates for τ	the number of surrogates that have not met the targets
nSatisfiedSurrogates for τ	the number of surrogates that have satisfied the targets
largest deficit for τ	the magnitude (positive) of the largest deficit
total cost [k]	the sum of the k^{th} cost for this network (perhaps many costs)
average cost [k]	the average cost per cell for the k_{th} cost

One attribute, the *surplus slack*, has proven especially useful in solving the SCP problem and its many variants. It is used extensively in the pre-defined objectives, and greatly improves the performance of the search. The surplus slack for a given solution is the sum of surrogate representation exceeding the target, counting only the first unit of surplus for each surrogate (Ciarleglio et al., 2007). By maximizing the surplus slack, we increase the number of surrogates that have achieved a "critical" surplus of one unit, increasing the likelihood of finding a cell that can be dropped from the current solution. A related quantity, the *secondary slack,* looks one step further. It sums any surplus greater than one unit, counting only the second unit of surplus. Maximizing the secondary slack increases the likelihood that the search can remove two cells.

6.6.1 Pre-Defined Rule Based Objectives

ConsNet provides two particularly useful pre-defined objectives (with more under development). The MDS-C objective (Most Deficient Surrogate - version C) addresses the basic SCP: find the minimum number of cells required to meet a specific representation target. The ITS (Intransitive Shape) objective performs a similar function while aggressively optimizing the shape (compactness) of the network. Both of these objectives address the set cover problem, and in this context, a solution is considered *feasible* if it meets the specified representation targets.

6.6.1.1 MDS-C Objective

In Ciarleglio et al. (2007), we introduce a RBO designed to solve the basic SCP for a given target set. In addition to finding a minimal cardinality cover, the rules also address auxiliary goals to maximize clustering and the total surrogate representation.

Recall that a RBO is a comparison operator that examines two alternatives (or solutions) a_i and a_j, and returns "a_i superior to a_j", "a_i inferior to a_j" or "a_i is equal to a_j". We show that an intransitive comparison operator can actually improve search performance by making local decisions other than the simple steepest descent. This allows the search to detect and fix inefficient structures in the solution as it is built.

The intransitive RBO in this study was called MDS-C, based on the Most Deficient Surrogate heuristic, which pursues the surrogates with the largest target deficits. As presented in Figure 6.16, the MDS-C comparison operator can be broken into three different cases. In *surplus mode*, both alternatives have satisfied the targets. In *deficit mode*, both alternatives fail to meet the targets, and in *borderline mode*, only one of the alternatives has met all targets.

In borderline mode, the MDS-C always prefers a feasible alternative (one that meets the representation target) over an infeasible alternative. As presented in Figure 6.17, the comparison rules for the surplus mode are straightforward. This figure should be interpreted as a tie-breaking hierarchy. The first criterion is the number of cells. When a_i and a_j meet the targets, we always prefer a solution with fewer cells. If no preference is indicated, we move to the next nested decision based on the surplusSlack. The rules within surplus mode represent a linear ordering (because of the strict hierarchy).

Figure 6.18 (following the nested hierarchical format introduced in Figure 6.17) details the MDS-C in deficit mode. If one alternative has fewer nDeficientSurrogates, it is preferred. If they both have the same nDeficientSurrogates *and* the same nCells, the decision is made based on the largestDeficit, then the remainingDeficit, and so on (following the hierarchy presented). However, the *decision branch* (the **else if** statement) is invoked when both alternatives have the same nDeficientSurrogates and have *different* nCells. This condition interrupts the strict nested hierarchy.

If the two alternatives possess the same nDeficientSurrogates but differing nCells, the alternative with lesser nCells may be preferred, especially if there is no accompanying increase in either the largestDeficit or the remainingDeficit. In this case, the solution with fewer cells is preferred because it does an equivalent job of meeting the representation targets, but with one less cell. Going one step further, it may be admissible to prefer a solution with one fewer cell as long as the remainingDeficit increases by only one unit. Unselected cells that contribute only one unit of representation are usually abundant; dropping these cells opens the possibility of finding a better cell (with perhaps two units of added representation) somewhere else in the solution space. This *gambit* allows the search to accept strategically disimproving moves.

More often, we are comparing solutions that differ by a cardinality of two due the neighborhood structure used in the search. Extending the logic above, we may prefer a solution that has two fewer cells as long as the remainingDeficit does not increase by more than 2 units. This 2 unit difference could be caused (a) by two nearly redundant cells or (b) by one redundant cell *and* a cell that is only contributing two units of representation. In both cases, the solution with fewer cells is preferable since it permits the search to find a more efficient solution.

Figure 6.18 shows that MDS-C will prefer the solution with fewer cells as long as both conditions (a) and (b), stated below, are true:

(a) its largest deficit is no greater than the other solution's largest deficit

AND

(b) either

{its remaining deficit is no greater than the other solution's remaining deficit }
OR gambit:
{the two solutions differ by no more than two cells and
its remaining deficit is no greater than 2 units more than the other solution's
remaining deficit }

Both types of deficits are considered to prevent the search from preferring a solution that maintains the remainingDeficit but *increases* the largestDeficit (which would violate the most deficient surrogate heuristic). The search can invoke this gambit while comparing alternatives within a single neighborhood because these solutions are either +1 or -1 in cardinality (different by no more than two). As clearly illustrated in the example given in Figure 6.19, this rule can cause MDS-C to be intransitive under certain conditions. In deficit mode, the search routinely encounters situations where this intransitivity directly affects the trajectory of the search, improving search performance dramatically (see Ciarleglio et. al (2007) for more details).

This example shows that rule based objectives can be highly tailored to improve the search performance. The gambit used by MDS-C allows the search to "override" the steepest descent direction if it detects structures in the solution that may be preferable in the long run. Although, this gambit leads to an intransitive comparison operator, there is ample evidence that some intransitivity is allowable when considering the sequential decisions made by the search (see Section 3.4). The MDS-C comparison operator significantly outperforms its transitive equivalent.

6.6.1.2 ITS Objective

The goal of the intransitive shape objective (ITS) is to minimize the cardinality of the set cover and simultaneously improve the shape (compactness) of the conservation area network. Balancing these two criteria is not simple. ITS uses specialized gambits that accept opportunistic tradeoffs between the criteria taking calculated risks which aggressively advance the Pareto frontier. Empirically, this intransitive approach has significantly outperformed similar transitive objectives.

The ITS objective uses different comparison operators to represent the *revealed preferences* and *outcome preferences* of the search agent (see Section 3.3). The revealed preferences $\preceq_{R,ITS}$ are used by the search to make decisions at each iteration. This comparison operator contains several gambits that make it intransitive. Unlike MDS-C, the ITS comparison operator exhibits intransitivity *in feasible space, including near the local optima*. This pervasive intransitivity makes it difficult to describe the outcome preferences generated by $\preceq_{R,ITS}$ with regard to a fixed tradeoff between shape and cardinality. In addition, it is possible no maximal element exists, i.e., for *any* solution, a_i , there exists at least one solution, a_j, such that $a_i \preceq_{R,ITS} a_j$.

The structure embedded in $\preceq_{R,ITS}$ creates ambiguity in the link between $\preceq_{R,ITS}$ and $\preceq_{O,ITS}$. Careful examination of the comparison rules in $\preceq_{R,ITS}$ and observations of the associated long term search behavior indicate that the search generally prefers solutions with lower cardinality compared to solutions with superior shape. However, because $\preceq_{R,ITS}$ also includes auxiliary preferences, it is important to note that the *exact* outcome preference exerted by $\preceq_{R,ITS}$ is unknown. Hence, $\preceq_{R,ITS}$ yields an approximation to the stated intent of obtaining solutions with minimal cardinality and preferred shape. To simplify the issue, it is assumed that the $\preceq_{O,ITS}$ follows this convention: feasible alternatives are sorted first by the number of cells (with fewer cells preferred), and then by shape (compactness is preferred). These unambiguous outcome preferences are used to rank and display solutions. However, despite the complex mechanics of the revealed preferences, the search using $\preceq_{R,ITS}$ behaves in a very rational and predictable manner.

Like MDS-C in the previous section, $\preceq_{R,ITS}$ is broken into three cases: *surplus mode*, *deficit mode*, and *borderline mode*. In deficit mode, ITS is identical to MDS-C, except the preference for clusters is replaced with a preference for shape. In borderline mode, ITS always prefers a feasible solution over an infeasible solution. In surplus mode, where the search spends most of its time, $\preceq_{R,ITS}$ considers four main criteria: the number of selected cells, the shape, the number of clusters, and the surplus slack, as well as two auxiliary criteria: the secondary slack and the total representation. The criteria are considered in the following hierarchical order:

(1) Since one primary goal is to find minimal cardinality set covers and it is the *hardest* criteria to manage, minimizing the **number of cells** is placed first.

(2) Using shape (a floating point value) as the second criteria could terminate the decision process before the other important criteria are considered because ties would be rare. Instead, the **number of clusters** is used as a reasonable surrogate for shape. The assumed correlation between the number of clusters and the shape is only valid when the alternatives under comparison differ by one or two cells *and* the cells are fairly uniform in shape (such as a regular grid).

(3) The **surplus slack** is placed third because it plays a critical role in allowing the search to drop cells and find new solutions of minimal cardinality.

(4) Having considered the other three main criteria, **shape** is placed fourth, followed by the auxiliary criteria .

(5) In the rare event of a tie, sort by the **secondary slack**.

(6) Finally, sort by **total representation**.

The above logic yields the comparison operator, $\precsim_{\text{R,ITS}}$, presented in Figure 6.20. As in MDS-C, gambits embedded in $\precsim_{\text{R,ITS}}$ improve the search performance. These gambits (G1-G4) alter the decision rules used in the first four levels of the comparison operator. G1 causes the search to prefer a solution with *more cells* only if it offers a substantial improvement in the surplusSlack while maintaining the shape and clustering. If the number of cells is a tie, G2 causes the search to prefer a solution with *more clusters* only if it offers a substantial improvement in surplusSlack. If the number of clusters is tied, G3 causes the search to prefer a solution with *less surplusSlack* as long as the difference in surplus slack is below a specified margin and the other solution has better or equivalent shape. Finally G4, a new type of gambit, is designed to manage inexact floating point values. If the floating point values are within a specified tolerance, they are considered identical, and the comparison operation continues to the next level. A detailed pseudocode, including the values chosen for the margins and thresholds, appears in Figure 6.21.

G1 through G4 are *essential* to the performance of the ITS comparison operator. Without them, the search can improve shape but has difficulties maintaining a minimal cover. It is likely that G2, which promotes the importance of surplus slack (*particularly* in presence-absence problems), endows ITS with the ability to pursue minimal cardinality solutions. The full implications of the other gambits are not fully understood.

Direct performance comparisons with other conventional objective functions is difficult because of the unique structure and character of ITS. However, a *large* number of cardinality/shape objectives have been explored during the research documented here, including a variety of linear methods such as strict preference hierarchies and weighted score combinations. Compared to *combined* output of these methods, *ITS found significantly superior solutions for every data set*.

Ostensibly, this suggests that ITS is somehow superior to linear objective functions at driving the search, despite its relative inability to decide which solution is technically "best". We hypothesize that the intransitivity may enable the search to avoid or escape local optima in a way that linear objectives cannot. This is consistent with the assertion from Mandler (2005) that "A willingness sometimes to choose *x* over *y* and sometimes to choose the reverse can help an agent who cannot rank *x* and *y* to avoid manipulation and achieve outcome rationality." In a sense, the intransitive comparison operator is "less gullible" than a linear operator, which always pursues the steepest descent directions. Thus, ITS may be evading the non-productive local optima that characterize this search landscape.

Overall, it appears that ITS is highly effective at exploring the Pareto frontier between shape and cardinality. Several results from the ITS objective are presented in Section 6.8.

6.6.2 User Defined Multi-Criteria Objectives

The user interface in MASTS provides several different options for combining different criteria into a multi-criteria objective. One approach that has been successful in previous planning exercises is the modified analytic hierarchy process (mAHP) (Moffett, Dyer, & Sarkar, 2006b). Due to the flexibility of rule based objectives, it is likely that other multi-criteria analysis (MCA) techniques (particularly outranking methods) can also be integrated into ConsNet (Moffett, Dyer, & Sarkar, 2006a).

The general multi-criteria anlaysis (gMCA) used in ConsNet is closely related to the mAHP. It should be noted that neither approach resembles the original AHP except in the way that weights are assigned to the individual criteria (Saaty, 1980). The criteria that may be considered are shown in Table 6.5. Adopting the notation from (Moffett et al., 2006a), the set of criteria under consideration will be designated $K = \{\kappa_1, \kappa_2, ... \kappa_n\}$where κ_i represents a single criterion. The set of alternatives under consideration will be designated $A = \{\alpha_1,\ldots\alpha_m\}$, where α_j represents a single alternative. The set of alternatives under consideration will by necessity be a miniscule subset of Γ, the set of all possible alternatives. For this discussion, *n* is the number of criteria and *m* is the number of alternatives.

The criteria (such as the number of cells, the target surplus, etc) are defined in such a way that we can assign an unambiguous quantitative value v_{ij} which describes the performance of α_j with respect to κ_i. Additional assumptions about the criteria and these values are required before this technique can be applied (Moffett et al., 2006a). First, we must be able to define a linearization function for each criterion (ν_i) which scales all of the values to the same interval. In the gMCA, users specify whether the criterion should be maximized or minimized and then provide a fixed min ($v_{i,min}$) and max ($v_{i,max}$) value for the criterion. The function ν_i normalizes the criteria scores v_{ij} between the max and min values, with the convention that larger scores are preferred:

$$v_i(\mathrm{v}_{ij}) = \begin{cases} 0 & \mathrm{v}_{ij} \leq \mathrm{v}_{i,\min} \\ \dfrac{\mathrm{v}_{ij} - \mathrm{v}_{i,\min}}{\mathrm{v}_{i,\max} - \mathrm{v}_{i,\min}} & \mathrm{v}_{i,\min} < \mathrm{v}_{ij} < \mathrm{v}_{i,\max} \\ 1 & \mathrm{v}_{i,\max} \leq \mathrm{v}_{ij} \end{cases} \qquad \text{if } \kappa_{\mathrm{i}} \text{ is maximized}$$

$$v_i(\mathrm{v}_{ij}) = \begin{cases} 1 & \mathrm{v}_{ij} \leq \mathrm{v}_{i,\min} \\ \dfrac{\mathrm{v}_{i,\max} - \mathrm{v}_{ij}}{\mathrm{v}_{i,\max} - \mathrm{v}_{i,\min}} & \mathrm{v}_{i,\min} < \mathrm{v}_{ij} < \mathrm{v}_{i,\max} \\ 0 & \mathrm{v}_{i,\max} \leq \mathrm{v}_{ij} \end{cases} \qquad \text{if } \kappa_{\mathrm{i}} \text{ is minimized}$$

Another assumption required for this technique is that the relative importance of the criteria can be evaluated on a ratio scale. This allows weights ω_i to be assigned to each criterion, identical to the method prescribed by the mAHP. The score for each alternative is calculated as the linear combination of the weights and individual criterion scores, with the convention that larger scores are preferred:

$$f(\alpha_j) = \sum_{i=1}^{n} \omega_i v_i(\mathrm{v}_{ij})$$

Because the weights are normalized, the maximum value of this score is 1. This score is most accurate if the criteria are mutually difference independent (Dyer, 2005), which is sometimes difficult to establish. In addition, some knowledge of the anticipated values is required to assign the min and max values for each criterion, and the precise interpretation of these values is not fully understood. Finally, the gMCA was developed to rank *static collections* of alternatives, and it may not be ideal for driving a dynamic search. Despite these drawbacks, the gMCA demonstrates consistent and predictable behavior as a ranking method.

By itself, the gMCA method is not highly effective at driving the search toward optimal solutions. This is because *the preference structure used to rank the absolute best alternatives is not necessarily the best preference structure to choose the next incumbent solution during an iterative search*. For instance, when the gMCA is applied to a multi-criteria SCP, the search flounders unless the surplusSlack is part of the objective. Although the surplusSlack greatly improves the search performance by making it easier to drop cells and discover new solutions, it seems unreasonable to ask planners to rate the importance of the surplusSlack, which has no bearing on the quality of the conservation area network. This quandary led to the development of a gMCA variant that uses separate models for the revealed preference and outcome preference.

This variant, gMCA-SCP, uses the same scoring system as the gMCA to rank solutions (i.e. both methods have identical outcome preferences). However, the revealed preference of the gMCA-SCP is modified to include a preference for surplusSlack:

$$f_{REVEALED}(\alpha_j) = \sum_{i=1}^{n} \omega_i v_i(\mathrm{v}_{ij}) + 0.05\left(\frac{surplusSlack}{nSurrogates}\right)$$

This adjusted score is used to determine the next incumbent solution. Note that the maximum value of the surplusSlack term is 0.05, compared to a maximum value of 1 for the remaining terms. This extra term is essential to search performance but transparent to the user. It does not alter the final rankings and users are not forced to express a preference regarding the surplusSlack. In the software, this objective is called the “multi-criteria minimum area problem”. This example highlights the flexibility of the dual preference structure to simplify difficult multi-criteria situations.

6.7 New Heuristic Approaches

A new family of heuristic methods, cell signature sorting (CSS), has been developed to construct fast initial solutions for the basic set cover problem associated with ConsNet with a specified representation target τ. These algorithms are significantly faster than ResNet, previously the fastest heuristic for this problem (Garson, Aggarwal, & Sarkar, 2002), and provide solutions of comparable or higher quality. There are a wide variety of heuristic methods in the literature. Iterative heuristics that incorporate complementarity are the most successful (Justus & Sarkar, 2002). Pressey, Possingham, & Day (1997) examine 30 algorithms based on rarity and complementarity, but did not find an overall “best” strategy, later concluding that heuristic performance is highly problem specific (Pressey et al., 1999). Sarkar (2003) defines 12 different types of complementarity for use in such algorithms. The CSS algorithms favor the measures of target-based complementarity (T-complementarity) and richness (F-complementarity). T-complementarity is the total representation that a cell can contribute towards the targets that are not already satisfied. F-complementarity is measured as the total representation within a cell.

The data sets presented below demonstrate that heuristic solutions may be far from optimal, casting doubt on the overall usefulness of simpler heuristic approaches. The tabu search methodology embedded in ConsNet is a much more robust approach that offers drastic improvements over simple heuristic solutions for a variety of objectives. Given ConsNet's capabilities, considering the subtle differences in the quality of heuristic algorithms is perhaps irrelevant. However, the construction of an initial solution should be *fast*, which is the primary reason the new CSS algorithms have been developed.

This speed is due to advanced data structure management and the *cell signature*, a large bitwise integer that captures the presence/absence of surrogates for each cell, giving priority to certain surrogates. CSS methods can be customized for different strategies involving rarity and complementarity. Two strategies were found to be particularly effective. The Most Deficient Surrogates strategy (MDS) focuses on reducing the largest target deficit. The Rarity First strategy (RF) pursues cells that reduce the deficit for the rarest surrogates. The algorithms are described in this section, and results are presented in Section 6.8.1.

6.7.1 Algorithm Components

First, an overview of the mechanisms used in this algorithm is provided. One fundamental concept is the *surrogate priority index array*. As the name suggests, this array lists the surrogates in order of priority, starting with the lowest priority in position 0. In a rarity first algorithm, the "high priority" surrogates must be the rarest. In the MDS algorithm, the high priority surrogates are those with the largest deficits. The ability to assign different priorities gives the algorithm considerable flexibility. The *cell signature* is created from the surrogate priority index array. Each set bit (a bit containing 1) in the cell signature represents the presence of a specific surrogate. The higher order bits correspond to the high priority surrogates (see Figure 6.22).

This cell signature provides a quick and detailed mechanism for breaking ties. If many cells contain the highest priority surrogate, then the decision about which cell to select may proceed to the second highest priority surrogate. If there are several cells that contain this surrogate, then the decision may continue to the next surrogate, and so on. With a cell signature, this type of logic can be performed as an integer comparison, which is extremely efficient to execute.

The source code for the CSS family contains other supporting data structures and concepts:

Grid Tracker. As the heuristic solution is built, the grid tracker monitors which cells are selected, which cells are available, and the surplus/deficit for each surrogate. The grid tracker manages the addition and removal of cells, and serves as an oracle for those who wish to know the state of the partially constructed network. Also, the grid tracker enforces the requirements on permanently included and excluded cells.

Virtual Cell. The virtual cell represents the actual cell, with extra data that will be used during the heuristic. In particular, the virtual cell keeps track of the cell signature, target based complementarity (T-complementarity), and richness (F-complementarity).

Virtual Cell Pool. The virtual cell pool strategically organizes the virtual cells that are available for insertion (or deletion). It assigns, updates, and sorts cell signatures for each of the virtual cells. The virtual cell pool maintains a sorted set of virtual cells in such a manner that the next most desirable cell is always “on top”. Thus, a search is not required to find the next best cell, and multiple “good” cells can quickly be added by iterating over the elements of this sorted set. This arrangement saves considerable computational effort compared to looking through the whole set at each iteration. Operations on this sorted set are kept to a minimum.

Virtual Cell Comparator. The virtual cell comparator is the component within the virtual cell pool that defines how to sort the cells. Currently, the comparator can base decisions on F-complementarity, T-complementarity, and cell signatures.

Surrogate Re-prioritization Event. As cells are added or removed, it is possible that the surrogate priorities may change. In the MDS strategy, the remaining deficits will change as cells are added, forcing us to reassess the “most deficient surrogate” (along with the other surrogate priorities). When the priority index array changes, then all of the cell signatures in the cell pool will have to be re-indexed, and the set must be freshly sorted. Since this takes considerable effort, surrogate re-prioritizations are done sparingly or not at all (depending on the algorithm currently being employed).

6.7.2 Algorithm Execution

The algorithm components are designed to allow different strategies to be chained together or interleaved. In both the MDS and RF algorithm, a forward strategy adds cells until a feasible solution is found, and then a backwards strategy removes redundant cells. Both the forward and backwards algorithm use the same components, initialized in a slightly different way. The pseudo-code listed below shows the general steps in the algorithm, and the following discussion fills in the specific details for the MDS and RF algorithms.

```
FORWARD ALGORITHM
> initialize grid tracker with no cells selected (except for the permanently included cells)
> initialize virtual cell pool
      this pool keeps track of cells available for inclusion
      generates a priority index array for the surrogates
      creates a virtual cell comparator (defining preferences for cell selection)
      builds a sorted set of virtual cells, the most desirable come first
> continue to select the first cell recommended by virtual cell pool until a target is met;
  as cells are selected, they are removed from the virtual cell pool
> when a surrogate meets its target, the virtual cell pool adjusts the cell signatures and
  complementarity values.  For every cell containing this surrogate, the bit representing
  that surrogate is set to zero (effectively removing this surrogate from the cell
  signature).  The cell pool removes and re-inserts only the affected cells into the sorted
  set.
> surrogate re-prioritization:  periodically, the algorithm may re-assess the priority index
  array.  If the priorities change, the cell pool re-sorts the remaining set of virtual
  cells.
> forward algorithm ends when all targets are met
BACKWARDS ALGORITHM
> initialize the BW algorithm with the grid tracker from the forward algorithm
> the BW strategy initializes a new virtual cell pool
      this pool keeps track of cells that are currently selected
      generates a priority index array for the surrogates
      creates a virtual cell comparator (defining preferences for cell removal)
      builds sorted set of virtual cells, the most desirable come first
> iterate over the sorted set of cells, for each cell:
      if the cell can be removed without violating the targets, remove the cell from the
      grid tracker
      remove the cell from the cell pool
> surrogate re-prioritization:  periodically, the algorithm may re-assess the priority index
  array.  If the priorities change, the cell pool re-sorts the remaining set of virtual
  cells.
> BW algorithm ends when we there are no cells remaining in the cell pool.
```

6.7.2.1 Most Deficient Surrogates (MDS) Heuristic

The MDS strategy selects cells that have high T-complementarity, giving secondary preference to the surrogates that have the largest remaining deficits. The motivation for this approach came from experimental observation. When the algorithm selects cells with high T-complementarity, the total deficit gets smaller, but not necessarily in a uniform manner. In the "endgame", there could be one remaining surrogate with a huge deficit. Satisfying the target for this surrogate requires adding cells for the singular purpose of meeting the last remaining target. This situation can be avoided by minimizing the largest deficit in earlier iterations, ensuring that all of the deficits are of comparable magnitude in the endgame. Instead of one huge deficit, for example, there could be three small ones. In some cases there could be available cells that simultaneously satisfy these three surrogates. In any event, this operational strategy enhances the likelihood of finishing with a tightly packed solution. This reasoning is summarized in Figure 6.23.

To define the MDS heuristic fully, we must specify the rules used in the forward and backward algorithm (i.e. the surrogate priority index array, the virtual cell comparator, and the re-prioritization criteria). In the forward algorithm, the surrogate priority index array is sorted so that the most deficient surrogates occupy the most significant positions (representing the most significant bits in the cell signature). The virtual cell comparator operates according to the rules in Figure 6.24. This figure should be interpreted as a tie-breaking hierarchy. Cells with higher T-complementarity are preferred. If there is a tie on the T-complementary, then the cell with the larger cell signature is preferred (which favors cells that contain the highest priority surrogates). If two cells have the same cell signature, then the cell with greater overall richness is preferred.

Since the most deficient surrogate changes over time, the surrogate priority index array is occasionally updated according to the current deficits within the partially constructed network. Reprioritization occurs every time a specific number of surrogates meet their respective targets. This reprioritization frequency depends on the number of surrogates:

$$reprioritizationFrequency = \begin{cases} 10 & nSurrogates < 100 \\ \lfloor 10 \cdot (\log[nSurrogates] - 1) \rfloor & nSurrogates \geq 100 \end{cases}$$

With this scheme, if there are 1000 surrogates, then a surrogate reprioritization will occur every time 20 surrogates reach their specified targets. If the frequency is too low, then the algorithm will run unnecessarily slow because of the time required to re-sort the cells. If the frequency is too high, then the algorithm could spend too much time focusing on surrogates that are no longer considered “high priority.”

Finally, every time the target for a specific surrogate is satisfied, the algorithm will update the T-complementarity for each cell that contains that surrogate. In addition, the bit corresponding to that surrogate in the cell signature is set to zero. Because each cell signature will contain a zero in this bit, the surrogate is effectively removed from the decision process that compares cell signatures.

Once the forward algorithm has added enough cells to satisfy all the targets, the backwards algorithm takes control. The priority index array is now defined so that the surrogates with the *smallest surplus* occupy the most significant positions (representing the most significant bits in the cell signature). Now cells that have a *small* cell signature are favored, i.e., cells that contain only high surplus surrogates are preferred (see Figure 6.25). Since the surpluses change as cells are removed from the network, a surrogate prioritization occurs every time the total surplus decreases by 1% from its original value (after the backward algorithm is invoked).

6.7.2.2 Rarity First (RF) Heuristic

The rarity first heuristic focuses on satisfying the targets for the rarest surrogates, similar to ResNet (Garson et al., 2002). The surrogate priority index array is ordered so that the rarest surrogates occupy the high priority positions. The virtual cell comparator behaves according to the rules in Figure 6.26. Unlike the MDS heuristic, the cell signature takes precedence over the T-complementarity, effectively requiring that rare surrogates are satisfied before cells are chosen on the basis of complementarity or richness.

Since the rarity of the surrogates does not change over time, the surrogate priority index array does not need to be updated, making this algorithm slightly faster than its MDS counterpart. When the targets for a specific surrogate are satisfied, the algorithm will update the T-complementarity for each cell containing that surrogate. It will also set the corresponding bit in the cell signature to zero, effectively removing that surrogate from the decision process. Once all targets are met, the same backwards algorithm used by MDS is used to remove redundant cells.

6.8 ConsNet Results Overview and Analysis

Table 6.6 details the practical data sets to which ConsNet has been successfully applied in the research documented here. Specific detailed results for each data set (with the exception of the *unpublished* World Marine Protected Areas (MPA) data set) will be presented in the sections that follow. The detailed findings from the extensive analyses of the *massive* World MPA data set will be released in the near future as a collaborative effort with Louisa Wood at the University of British Columbia.

Table 6.6 Summary of the data sets examined in this study.

data set	nCells (n)	nSurrogates (m)	problemSize (n*m)	representationType
Mexico[1]	71,248	86	6,127,328	presence/absence
West Virginia[2,3]	94,771	323	30,611,033	presence/absence
Indoburma[4]	294,830	184	54,248,720	expected value (rounded 0.01)
World MPA[5]	176,093	1,038	182,784,534	expected value (rounded 0.01)

[1] Sanchez-Cordero, V., Illoldi-Rangel, P., Linaje, M., Sarkar, S., & Peterson, A. T. (2005). Deforestation and Extant Distributions of Mexican Endemic Mammals. Biological Conservation, 126, 465-473.

[2] Justus, J., Fuller, T., & Sarkar, S. (2007). The Influence of Representation Targets on the Total Area of Conservation Area Networks. [in press] Conservation Biology.

[3] Strager, J. M., & Yuill, C. B. (2002). The West Virginia GAP analysis project final report: U. S. Geological Survey, Morgantown, West Virginia.

[4] Pawar, S., Koo, M. S., Kelley, C., Ahmed, M. F., & Sarkar, S. (2007). Conservation Assessment and Prioritization of Areas in Northeast India: Priorities for Amphibians and Reptiles. Biological Conservation, 136, 346-361.

[5] Acknowledgements: L. Wood, D. Pauly and others of the Sea Around Us Project, University of British Columbia for providing access to the input dataset. The Sea Around Us Project is an activity initiated and funded by the Pew Charitable Trusts. Marine mammal distribution data was provided by K. Kaschner, FMAP project, Dalhousie University & Institut fuer Biologie, Abtl. Evolutionsbiologie & Oekologie.

6.8.1 Heuristic Algorithm Results

Table 6.7 presents both the performance of the MDS2 and RF4 heuristics on each data set and the best known minimal set cover solutions (all found using ConsNet). Despite their algorithmic complexity, MDS2 and RF4 are much faster than ResNet (Garson et al., 2002), the most comparable heuristic approach. The increase in speed is due to the use of the cell signature comparisons and strategic management of the virtual cell pool. Table 6.7 indicates that the computational efficiency of MDS2 and RF4 is *critical* on larger problems and that heuristic solutions may be arbitrarily poor for large data sets. Although the true optimum is unknown for the two larger problems, the MDS2 and RF4 solutions for the World MPA data set are inferior by over 14% and 24%, respectively, to the best known ConsNet solution of 12,675 cells. It has been observed that RF4 outperforms MDS2 on presence-absence data sets that contain dense surrogates, such as the West Virginia data set.

Table 6.7 Performance of the MDS and RF heuristics on selected data sets, solving the basic SCP for a 10% representation target.

	Mexico		**West Virginia**		**Indoburma**		**World MPA**	
heuristic name	nCells	time	nCells	time	nCells	time	nCells	time
MDS2*	4116	6 s	9530	139 s	24114	186 s	14472	180 s
RF4*	4275	4.2 s	9477	110 s	27025	151 s	15808	145 s
ResNet	4233	15 m	--	--	--	--	14836	8 hr (est)
best known simple set cover solution	**4105**†		**9477**†		**22001**		**12675**	

*The times reported for the heuristic solutions do not include file IO (input/output) because of the unique way ConsNet loads and stores problem profiles. This IO is a *one time* setup cost, requiring less than a minute for large problems.
†known optimal simple set cover solutions

6.8.2 Benchmarks and Computational Effort

The solutions that will be presented in the following sections are the result of an extended analysis. While all of them were obtained in less than one day of computation (some requiring just a couple of hours), they were not carefully timed. Timing solutions is not straightforward, because ConsNet is not a “one click optimize” program. These solutions were discovered with user intervention using a variety of search techniques. It is difficult to report a time without also reporting every step along the way.

Instead of reporting times to arbitrary solutions in an uncontrolled setting, it is much more effective to measure performance with a fixed set of repeatable benchmarks. The benchmark history plays a critical role in tracking and improving program performance. These benchmarks are described below, along with a summary of the most relevant performance results for each data set.

The number of control parameters for the benchmarks is fairly large, including settings for ConsNet and the Java virtual machine. Every benchmark uses the same settings for the dynamic neighborhood selection: *basic DNS (general nbhds only)*, and the same tabu reaction strategy:

adaptive tabu reactor 1. All benchmarks are performed with the default settings on the garbage collector, and the maximum amount of memory that can be allocated on the specific computer.

The speed of the function evaluations depends heavily on the structure of the solution. Also, the evaluations will take significantly longer if replication is included. An abbreviated list of the benchmark settings are presented in Table 6.8. The hardware and software specifications of the computers used to perform the benchmarks are described in Table 6.9 and Table 6.10. The benchmarks have been run for a variety of settings and system architectures, but the significant results can be summarized with just a handful of tests.

Each benchmark was run multiple times. The first trial typically took longer because of the time required for the JVM to load the classes and perform dynamic compilations to optimize the code. The unpredictable timing of garbage collection also causes variation in benchmark speeds. The best of three benchmark trials is reported. If a benchmark run is labeled multi-threaded (MT), then the evaluation tasks are distributed to a thread pool containing one thread per processor (only on multi-processor machines). To evaluate the processor utilization efficiency, another identical benchmark is run in single-threaded (ST) mode, where all evaluations execute serially on a single thread.

Table 6.8 An abbreviated list of settings for the various benchmarks.

benchmark name	starting point	objective	iterations	spatial analysis
mexico2	MDS-2 heuristic	MDS-C 10% target	3000	clusters and shape
mexico3	ALL cells selected	MDS-C 10% target	3000	clusters and shape
mexico_rep	MDS-2 heuristic	MDS-C 10% target	3000	replication (all surrogates)
westvirginia2	MDS-2 heuristic	MDS-C 10% target	3000	clusters and shape
westvirginia_shape	best known ITS soln*	ITS 10% target	3000	clusters and shape
indoburma2	MDS-2 heuristic	MDS-C 10% target	3000	clusters and shape
worldmpa2	MDS-2 heuristic	MDS-C 10% target	3000	clusters and shape

*the solution shown in Figure 6.33

Table 6.9 The hardware specifications of the various computers used in the benchmarks.

Computer Name	Chipset	CPU (GHz)	FSB (MHz)	nCPUs	bits	L2 cache	visible RAM*	RAM (MHz)
Frisbee (Dell Inspiron 6400)	Intel Core Duo T2400	1.83	663	x 2	32	2 MB per core	1.4 GB	533 DDR2 SDRAM
Simplex32	Intel Core2 Extreme Q6950	3.0	1333	x 4	32	4 MB per core	1.4 GB	667 DDR2 SDRAM

*the JVM cannot use more than 1.4 GB of RAM on a 32-bit system

Table 6.10 The operating system and Java Virtual Machine installed on each computer.

Computer Name	OS	JVM
Frisbee	WindowsXP Pro SP2	Java HotSpot (build 1.6.0_01-b06, mixed mode)
Simplex32	WindowsXP Pro SP2	Java HotSpot (build 1.6.0_03-b05, mixed mode)

Table 6.11 The benchmarks are performed in pairs, first using multi-threaded (MT) code that takes advantage of all processors, and then executing on a single thread (ST). By comparing the two lines, it is possible to estimate the CPU efficiency for the parallel evaluations. Results are reported in evaluations per second.

line	computer	benchmark	threading	JVM	evals/second (best of three)	approximate time for 100,000 iterations (minutes)	CPU efficiency actual/max
1	Frisbee	mexico2	MT 2	server	350,453	33	1.63 / 2
2	Frisbee	mexico2	ST	server	214,559	55	
3	Frisbee	mexico2	MT 2	client	246,142	48	1.34 / 2
4	Frisbee	mexico2	ST	client	183,353	65	
5	Frisbee	mexico3	MT 2	server	19,967	N/A	1.98 / 2
6	Frisbee	mexico3	ST	server	10,078	N/A	
7	Frisbee	westvirginia2	MT 2	server	141,690	111	1.84 / 2
8	Frisbee	westvirginia2	ST	server	77,088	205	
9	Simplex32	mexico2	MT 4	server	1,092,526	11	2.11 / 4
10	Simplex32	mexico2	ST	server	517,319	23	
11	Simplex32	mexico_rep	MT4	server	416,000	29	3.24 / 4
12	Simplex32	mexico_rep	ST	server	128,372	92	
13	Simplex32	westvirginia2	MT 4	server	488,174	32	2.98 / 4
14	Simplex32	westvirginia2	ST	server	163,781	96	
15	Simplex32	westvirginia_shape	MT 4	server	677,687	23	2.66 / 4
16	Simplex32	westvirginia_shape	ST	server	254,061	62	
17	Simplex32	indoburma2	MT 4	server	437,232	69	2.91 / 4
18	Simplex32	indoburma2	ST	server	150,091	202	
19	Simplex32	worldmpa2	MT 4	server	277,758	83	3.51 / 4
20	Simplex32	worldmpa2	ST	server	78,913	292	

In Table 6.11, lines 1-4 demonstrate the difference between the *server* and *client* virtual machine. The server VM is more aggressive at code optimization, dynamic compilation, and garbage collection, and runs 20-50% faster than the client VM for most benchmarks in ConsNet.

In lines 5 and 6, the search starts with *all cells selected*. This particular benchmark is very slow because the search must occasionally traverse a cluster which contains *all* of the selected cells. In practice, this behavior is rarely seen because (a) most practical solutions will not have all cells selected and (b) users normally start the search from a reasonable initial solution. However, this benchmark demonstrates that different parts of the solution space can require vastly different evaluation times.

In fact, ConsNet is optimized to run faster when the solutions contain compact reserves with only a fraction of the available cells selected (precisely the types of solutions that planners will prefer, and where the search will spend most of its time). The *westvirginia_shape* (lines 15-16) benchmark evaluates solutions with preferred shape. Compared to the *westvirginia2* benchmark (lines 13-14), it is clear that ConsNet runs as much 50% faster (single-threaded) for solutions that are structurally compact.

Including replication in the analysis requires additional computational effort. The *mexico_rep* benchmark (lines 11-12) computes replication for all surrogates. Comparing this to the *mexico2* benchmark (lines 9-10), it appears that evaluations with replication take about 4 times longer than evaluations with clustering alone (single threaded). Overall, the time required to compute replication is highly dependent on the number of surrogates and the structure of the solution.

The CPU efficiency is reported as the ratio of execution time between the MT and ST benchmarks. Ideally, this ratio will be close to the number of processors, indicating that every processor is being fully utilized. However, this ideal ratio cannot be achieved because (a) some processors may be idle during the serial portions of the code and (b) the computational overhead required to manage multiple threads can offset gains in parallel efficiency.

For evaluations that are extremely quick, thread contention can be a major loss. To reduce thread contention, the thread pool uses one work queue *per thread* and work stealing to balance the workloads. Although fairly efficient for these tests, slight adjustments can be made to increase multi-threaded performance and scalability. Table 6.11 shows a clear trend that processor efficiency improves for larger problems.

6.8.3 Mexico's Endemic Mammals

The Mexico data set contains the modeled distributions of 86 endemic mammal species (Sanchez-Cordero et al., 2005). Mexico is divided into 71,248 "square" cells, each 0.05° by 0.05°, containing presence-absence data on the modeled species. The total species richness is illustrated in Figure 6.27). The following analysis is based on a 10% conservation target for all species.

One unexpected feature of this problem is the *overwhelming* number of feasible minimal set cover solutions (meeting the 10% representation target with 4105 selected cells). Millions of such solutions are readily available through ConsNet and they can be radically different with selected pairs sharing only 1500 cells. This revelation is important to planners. Often, the problem is presented as a "tradeoff" between shape and cardinality, i.e., to improve the shape, we may have to accept a solution with more cells. The Mexico data set demonstrates that such a compromise is not always necessary; the abundance of minimal cover solutions allows aggressive attempts to improve the shape *while maintaining a minimal cover solution*. The ITS objective is well-suited for this task.

ConsNet can produce a huge variety of solutions for users to interactively explore. Only a few *key* results are presented here. Figure 6.28 presents the inferior MDS2 heuristic solution, a convenient starting point for the search. The ITS objective outperformed all other objectives producing the highly superior solution presented in Figure 6.29. This highly refined solution was produced in about 700,000 iterations over (~1.25 hours on Simplex32, timing discussed in Section 6.8.2), but excellent similar solutions were found within the first 100,000 iterations.

Table 6.12 Summary of solutions presented for the Mexico data set.

configName	selected cells	area (km^2)	perimeter (km)	clusters	total representation	shape
MDS2 heuristic solution	4,116	116,402	40,067	639	41,917	0.344
best known ITS solution	4,105	115,905	17,094	101	36,784	0.147

6.8.4 West Virginia

The West Virginia data set contains the modeled distributions of 323 species (birds, mammals, amphibians, and reptiles). This presence-absence data set, created by Chris Kelley, is based on the results of the West Virginia GAP Analysis Project (Strager & Yuill, 2002). West Virginia is divided into 94,771 cells, each 0.00833° by 0.00833°. The following results are based on a 10% conservation target.

Figure 6.30 shows that this data set is quite dense with 196 of the surrogates are present in more than half of the cells. In general, a minimal cover for this type of problem is easily obtained. As illustrated in Figure 6.31, the RF4 heuristic algorithm found a minimal cover. Other key solutions are summarized in Table 6.13.

Table 6.13 Summary of solutions presented for the West Virginia data set

configName	selected cells	area (km^2)	perimeter (km)	clusters	total representation	shape
RF4 heuristic solution	9477	6377	14,387	1,453	2,109,132	2.256
MDS2 heuristic solution	9530	6403	14,470	1,442	2,174,763	2.260
ITS starting from RF4 solution	9477	6363	1,822	23	1,991,459	0.286
ITS starting from MDS2 soln	9477	6356	1,449	17	1,965,631	0.228

Using the ITS objective, the search quickly consolidated the RF4 heuristic solution while maintaining the minimal set cover. Within 10,000 iterations, the solution contained less than 200 clusters. An excellent solution was obtained within 100,000 iterations. The result for 500,000 iterations is shown in Figure 6.32. Figure 6.33 shows the superior result for 500,000 iterations when the search was started at the MDS2 solution. The time required for computation is discussed in Section 6.8.2.

6.8.5 Reptiles and Amphibians in Indoburma

This enormous data set (Pawar, Koo, Kelley, Ahmed, & Sarkar, 2007), illustrated in Figure 6.34, covers portions of Burma (Myanmar) and seven states in India. The region was divided into 294,830 cells (0.0166° by 0.0166°). Maxent software (Phillips, Dudik, & Schapire, 2004) was used to model the distributions of 184 herpelogical species (63 amphibian, 121 reptile), and the surrogate representation is provided as expected values. The solutions presented here are for a 10% representation target.

Simple heuristic approaches like MDS2 are not effective on problems of this size. Presented in Figure 6.35, the MDS2 heuristic solution contains 2,100 cells (7,000 km^2) more than the best known minimal cover. Unlike the previous data sets, however, the MDS2 solution is not highly scattered. Since Maxent uses environmental variables to make predictions, the spatial layout of the surrogate distributions often highlight natural features and eco-regions.

Figure 6.36 presents the best known minimal cardinality solution, discovered using the MDS-C objective starting from the MDS2 heuristic solution. The MDS-C solution is highly scattered but preserves some of the regions of interest found in the heuristic solution even when the search starts from a variety of different initial solutions. This MDS-C solution with 22,001 cells was obtained after an extended search (~750,000 iterations, ~9 hours on Simplex 32, timing discussed in Section 6.8.2).

Figure 6.37 shows the solution obtained from a similar experiment performed with the ITS objective. The much more compact ITS solution is within 0.7% of the number of cells present in the MDS-C solution (in Figure 6.36). For the Indoburma expected value data set, it appears that a compromise between solution cardinality and shape may be unavoidable. To explore a range of options, the solution presented in Figure 6.38 was generated with a user defined gMCA objective (see Section 6.6.2) which placed more emphasis on shape. The gMCA solution improved the shape and connectivity considerably, at the cost of an extra 1,267 km^2. A comparison of the four discussed solutions appears in Table 6.14.

Table 6.14 Summary of solutions presented for the Indoburma data set

configName	selected cells	area (km^2)	perimeter (km)	clusters	total representation	shape
MDS2 heuristic solution	24,114	77,154	22,546	530	3.02E+07	0.292
best known MDSC solution	22,001	70,260	31,420	1,274	2.48E+07	0.447
best known ITS solution	22,160	70,772	11,542	93	2.48E+07	0.163
gMCA custom solution	22,549	72,039	8,921	52	2.66E+07	0.124

6.8.6 DNS – Selected Results

The effectiveness of the various DNS strategies will be demonstrated in two steps. First, the *basic* DNS strategy used to select general neighborhoods is shown to outperform random neighborhood selection. Since this basic strategy is the foundation for more advanced strategies, verification that it functions reasonably well for a variety of different objective functions and different problem types is important. The *standard* and *advanced* DNS methods extend the basic strategy, using different neighborhoods to escape when search progress stagnates. Initial results show that the basic neighborhood selection algorithm is superior to random neighborhood selection. Combined with the tabu memory structure, the basic strategy is already capable of escaping from local optima. Thus, escape reactions in the standard and advanced strategies should occur infrequently and later in the search process.

The basic DNS strategy works well on both presence-absence *and* expected value data sets. First, consider the SCP using the MDS-C objective described in Section 6.6.1.1. Figure 6.39 shows that basic DNS vastly outperforms random neighborhood selection for the Mexico data set. The search begins with no cells selected, builds up to a feasible solution, and then removes redundant cells to reach a minimal cardinality solution. The basic DNS strategy was quite superior to random neighborhood selection in both stages of the search, and appears to be particularly efficient at removing cells. Basic DNS discovered a minimal cardinality solution of 4105 cells at iteration 6,527. By comparison, random neighborhood selection *failed* to produce a minimal cardinality cover in 15,000 iterations. Since the neighborhoods are all the same size, every test involved the same number of function evaluations.

The next goal is to simultaneously minimize the cardinality of the set cover while improving the shape of the solution. The ITS objective described in Section 6.6.1.2 is well suited for this optimization. Figure 6.40 shows the Pareto frontier explored by the search using the various DNS strategies. Each trial was allowed a maximum of 1.7 billion evaluations (~250,000 iterations). The basic DNS strategy found dominating solutions compared to the three random trials. The standard and advanced strategies pushed the Pareto frontier even further. The advanced strategy used spatial rearrangement moves that added cells (no removals allowed). The performance of the standard and advanced strategies is comparable, but the advanced strategy spent quite a bit of time exploring solutions with higher cardinality as it tried different descent directions. For extended searches or larger problems, the advanced strategy is preferred because of its ability to diversify.

Similar tests were performed on the Indoburma data set, which uses expected values for surrogate representation. When searching for a minimal set cover with the MDS-C objective, Figure 6.41 shows again that basic DNS dominates random neighborhood selection, finding higher quality solutions considerably faster. Each trial was allowed a maximum of 2.7 billion evaluations (~150,000 iterations). The standard DNS strategy offers marginal improvements over basic DNS, but this margin represents a lot of saved effort. By the end of the search, basic DNS is trailing standard DNS by as much as 200,000,000 evaluations.

Continuing analysis on the Indoburma data set, we attempt to optimize both the cardinality and the shape of the solution using the ITS objective. Each trial was allowed a maximum 4.5 billion evaluations (~250,000 iterations). Figure 6.42 shows the Pareto frontier explored by the search using the various DNS strategies. While basic DNS does not “dominate” random neighborhood selection in the Pareto sense, it does a much better job reducing the cardinality of the set cover, which is by far the more difficult of the two criteria (and more likely to be of concern to decision makers). Advanced DNS (which is recommended on spatial problems) completely dominates the other methods.

6.9 Conclusions

ConsNet is the next generation in conservation area network design software. ConsNet provides novel and powerful algorithms which allow the user to address spatial characteristics such as connectivity and clustering. These capabilities are not available in other conservation area network design software. The extra features of MASTS, such as multi-criteria objectives, the graphical user interface, and the solution archive, allow users to thoroughly investigate their problem in an ongoing interactive analysis.

6.10 Figures

Figure 6.1 Clustering for a sample problem in Namibia (1250 cells, 33 species, 10% target) where two solutions use 41 cells (the minimal number of cells) to satisfy all surrogate targets. On the left, 29 isolated clusters are present (diagonal cells are considered isolated clusters). On the right, 9 clusters are present.

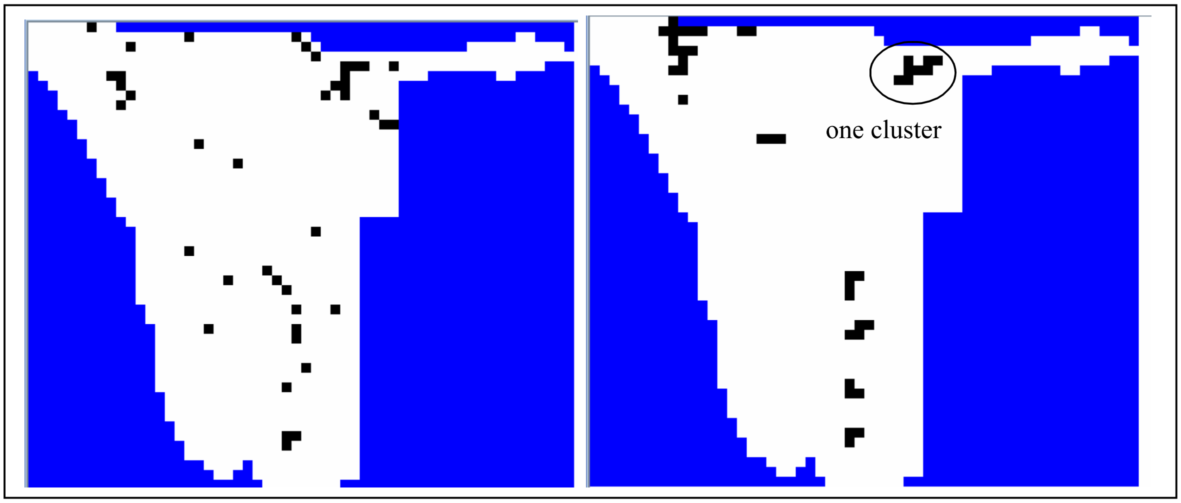


Figure 6.2 ConsNet can handle both regular grids (left) and irregular grids (right). The data structures are encapsulated in an interface so that the program works the same regardless of the grid type.

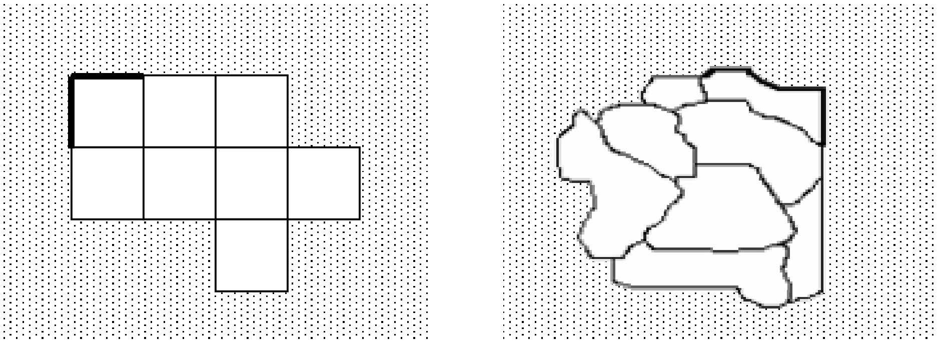
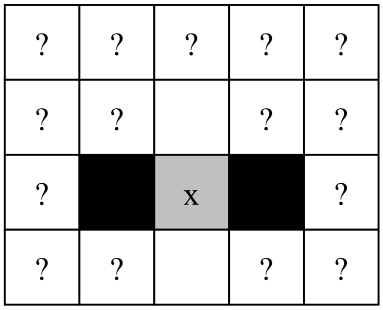
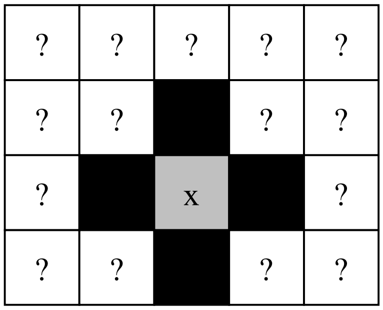


Figure 6.3 An example of a connector cell on a square grid. The status of the cells marked with a question mark is unknown. Changing the status of cell x may or may not change the number of clusters, depending on whether the two selected neighbors are connected by an alternate path through the ? cells.



?	?	?	?	?
?	?		?	?
?	■	x	■	?
?	?		?	?

Figure 6.4 An example of a connector cell on a square grid. Even when cell x is surrounded by four selected neighbors, the number of clusters that could be created or destroyed is unknown. The selected neighbors may be connected through alternate paths. However, checking the status of the SW, NW, SE, or NE neighbors may reveal whether all of the primary neighbors are connected.



?	?	?	?	?
?	?	■	?	?
?	■	x	■	?
?	?	■	?	?

Figure 6.5 An example of a isolated, annex, and connector cells on a grid with arbitrary shapes.

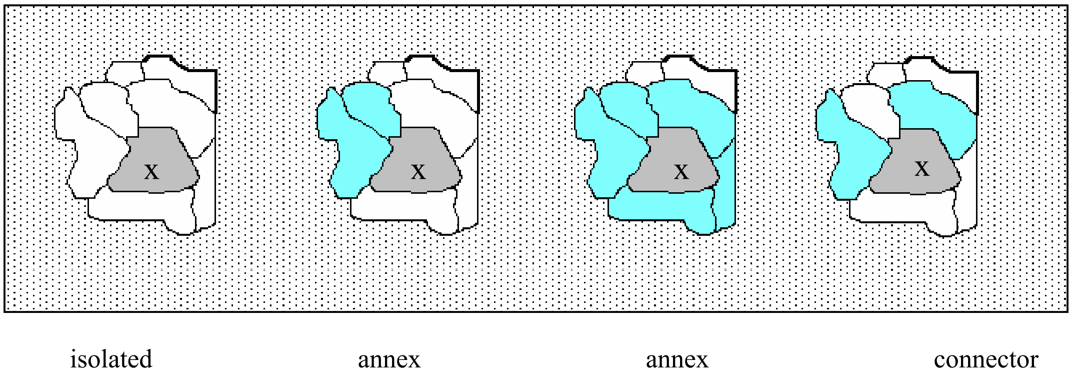
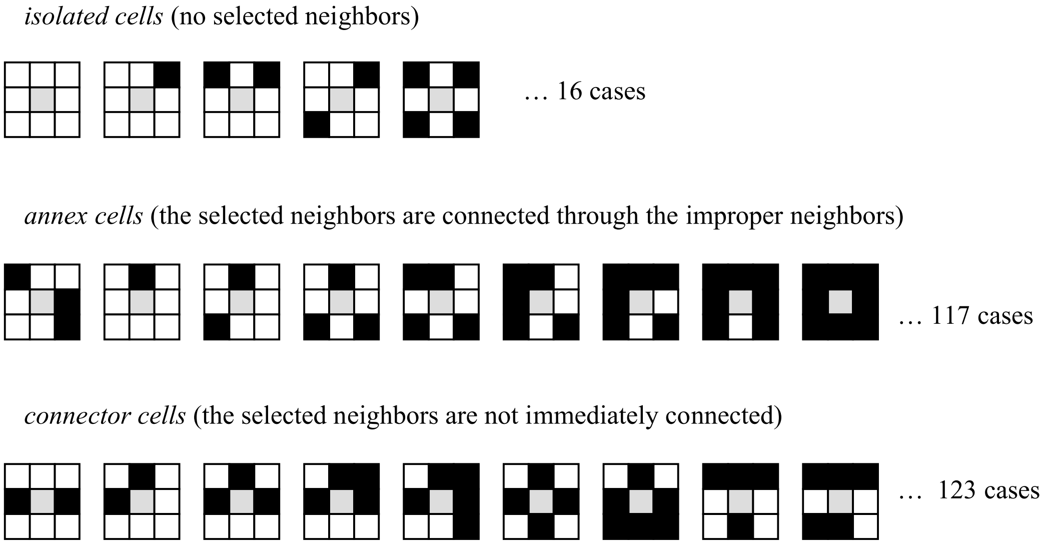


Figure 6.6 Examples of isolated, annex, and connector cells on a square grid.



isolated cells (no selected neighbors)

… 16 cases

annex cells (the selected neighbors are connected through the improper neighbors)

… 117 cases

connector cells (the selected neighbors are not immediately connected)

… 123 cases

Table 6.15 Properties that depend on problem size. The tabu tenure and neighborhood selection strategies are based on the number of active cells.

problem size (active cells)	target supercell size (S)	base # of super-cells (B)	min Tenure (T_{min})	max Tenure (T_{max})	min size shakeup move	nRepeats LargeNbhd	schedule1		schedule2		schedule3		schedule4	
							threshold	recovery	threshold	recovery	threshold	recovery	threshold	recovery
1000	200	5	5	15	5	10	200	210	500	1000	1000	2000	2000	6000
2000	400	5	5	15	5	10	200	210	500	1000	1000	2000	2000	6000
3000	600	5	5	15	6	10	200	210	500	1000	1000	2000	2000	6000
4000	800	5	5	15	6	10	200	210	500	1000	1000	2000	2000	6000
5000	1000	5	5	15	6	10	200	210	500	1000	1000	2000	2000	6000
10000	1000	10	10	30	8	20	400	420	1000	2000	2000	4000	4000	12000
20000	2000	10	20	60	11	20	400	420	1000	2000	2000	4000	4000	12000
30000	3000	10	30	90	15	20	400	420	1000	2000	2000	4000	4000	12000
40000	4000	10	40	120	18	20	400	420	1000	2000	2000	4000	4000	12000
50000	5000	10	50	150	21	20	400	420	1000	2000	2000	4000	4000	12000
60000	6000	10	60	180	25	20	400	420	1000	2000	2000	4000	4000	12000
70000	7000	10	70	210	28	20	400	420	1000	2000	2000	4000	4000	12000
80000	8000	10	80	240	31	20	400	420	1000	2000	2000	4000	4000	12000
90000	9000	10	90	270	35	20	400	420	1000	2000	2000	4000	4000	12000
100000	10000	10	100	300	38	20	400	420	1000	2000	2000	4000	4000	12000
110000	10500	11	110	330	41	22	440	462	1100	2200	2200	4400	4400	13200
120000	11000	11	120	360	45	22	440	462	1100	2200	2200	4400	4400	13200
130000	11500	12	130	390	48	24	480	504	1200	2400	2400	4800	4800	14400
140000	12000	12	140	420	51	24	480	504	1200	2400	2400	4800	4800	14400
150000	12500	12	150	450	55	24	480	504	1200	2400	2400	4800	4800	14400
160000	13000	13	160	480	58	26	520	546	1300	2600	2600	5200	5200	15600
170000	13500	13	170	510	61	26	520	546	1300	2600	2600	5200	5200	15600
180000	14000	13	180	540	65	26	520	546	1300	2600	2600	5200	5200	15600
190000	14500	14	190	570	68	28	560	588	1400	2800	2800	5600	5600	16800
200000	15001	14	200	600	71	28	560	588	1400	2800	2800	5600	5600	16800
210000	15334	14	210	630	75	28	560	588	1400	2800	2800	5600	5600	16800
220000	15667	15	220	660	78	30	600	630	1500	3000	3000	6000	6000	18000
230000	16001	15	230	690	81	30	600	630	1500	3000	3000	6000	6000	18000

problem size (active cells)	target supercell size (S)	base # of super-cells (B)	min Tenure (T_{min})	max Tenure (T_{max})	min size shakeup move	nRepeats LargeNbhd	schedule1		schedule2		schedule3		schedule4	
							threshold	recovery	threshold	recovery	threshold	recovery	threshold	recovery
240000	16334	15	240	720	85	30	600	630	1500	3000	3000	6000	6000	18000
250000	16667	15	250	750	88	30	600	630	1500	3000	3000	6000	6000	18000
260000	17001	16	260	780	91	32	640	672	1600	3200	3200	6400	6400	19200
270000	17334	16	270	810	95	32	640	672	1600	3200	3200	6400	6400	19200
280000	17667	16	280	840	98	32	640	672	1600	3200	3200	6400	6400	19200
290000	18001	17	290	870	101	34	680	714	1700	3400	3400	6800	6800	20400
300000	18334	17	300	900	105	34	680	714	1700	3400	3400	6800	6800	20400
350000	20001	18	350	1050	121	36	720	756	1800	3600	3600	7200	7200	21600
400000	21667	19	400	1200	138	38	760	798	1900	3800	3800	7600	7600	22800
450000	23334	20	450	1350	155	40	800	840	2000	4000	4000	8000	8000	24000
500000	25001	20	500	1500	171	40	800	840	2000	4000	4000	8000	8000	24000
550000	26667	21	550	1650	188	42	840	882	2100	4200	4200	8400	8400	25200
600000	28334	22	600	1800	205	44	880	924	2200	4400	4400	8800	8800	26400
650000	30001	22	650	1950	221	44	880	924	2200	4400	4400	8800	8800	26400
700000	31667	23	700	2100	238	46	920	966	2300	4600	4600	9200	9200	27600
750000	33334	23	750	2250	255	46	920	966	2300	4600	4600	9200	9200	27600
800000	35001	23	800	2400	271	46	920	966	2300	4600	4600	9200	9200	27600
850000	36667	24	850	2550	288	48	960	1008	2400	4800	4800	9600	9600	28800
900000	38334	24	900	2700	305	48	960	1008	2400	4800	4800	9600	9600	28800
950000	40001	24	950	2850	321	48	960	1008	2400	4800	4800	9600	9600	28800
1000000	41667	24	1000	3000	338	48	960	1008	2400	4800	4800	9600	9600	28800

Figure 6.7 The hierarchy of ConsNet neighborhoods

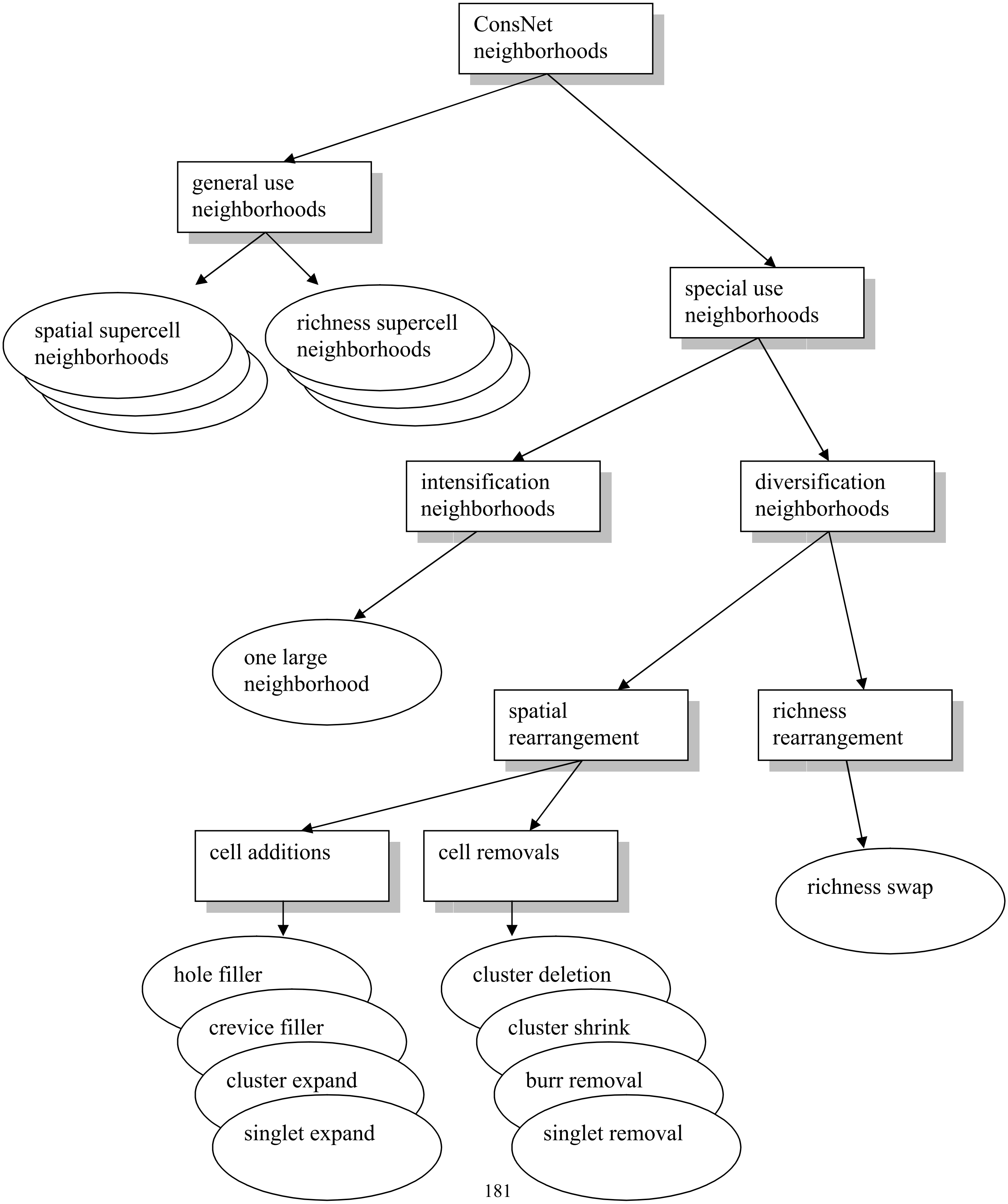


Figure 6.8 The cluster shrink neighborhood.

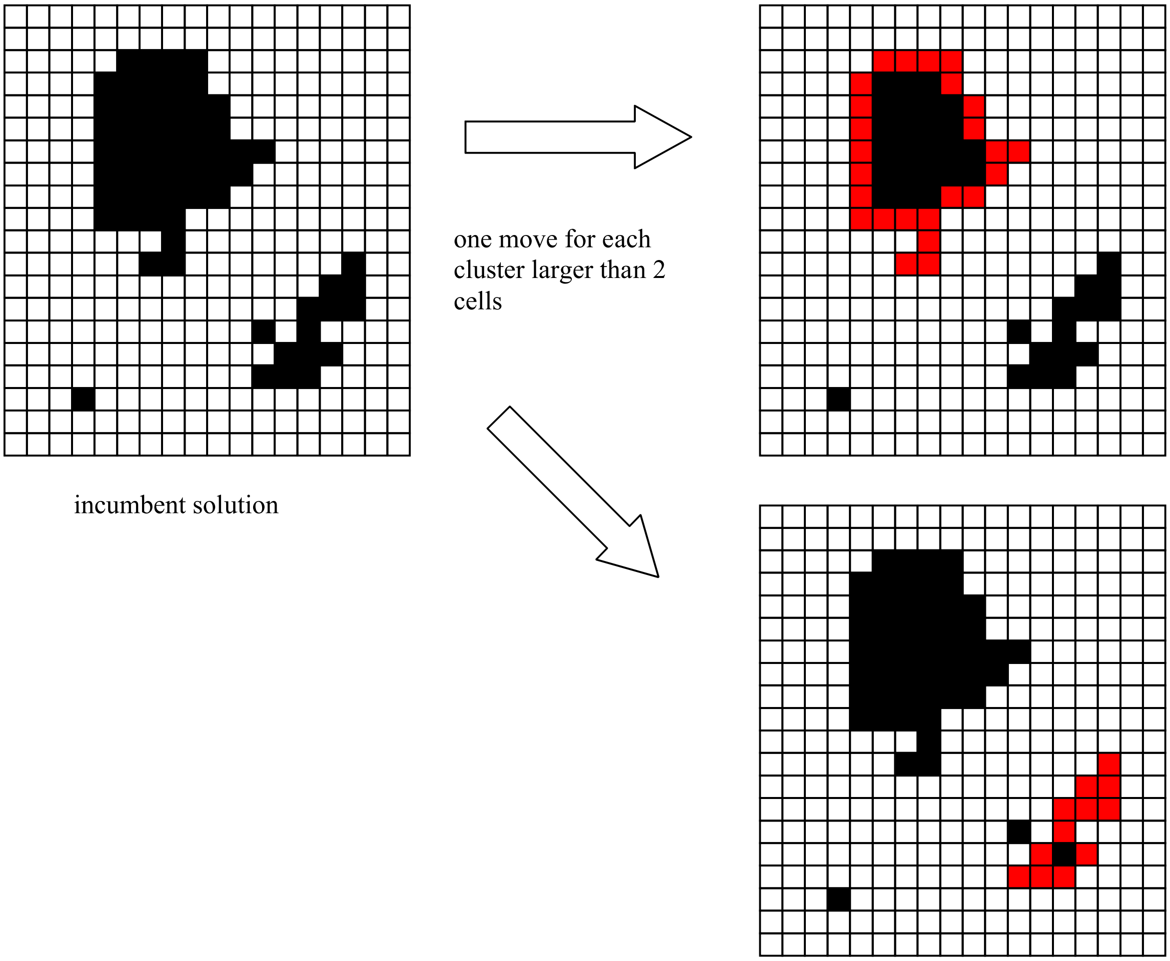
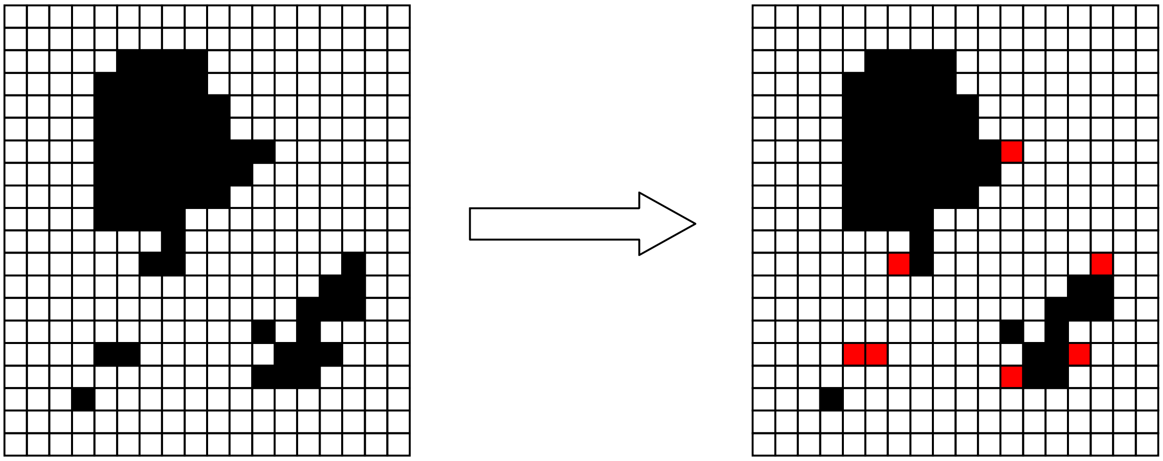
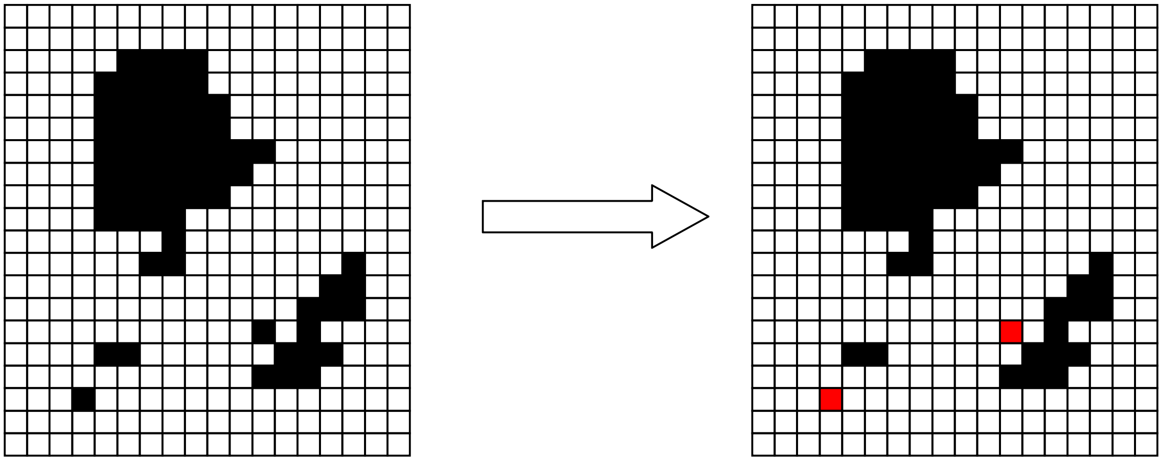


Figure 6.9 The burr removal neighborhood



incumbent solution

Figure 6.10 The singlet removal neighborhood.



incumbent solution

Figure 6.11 The cluster expand neighborhood.

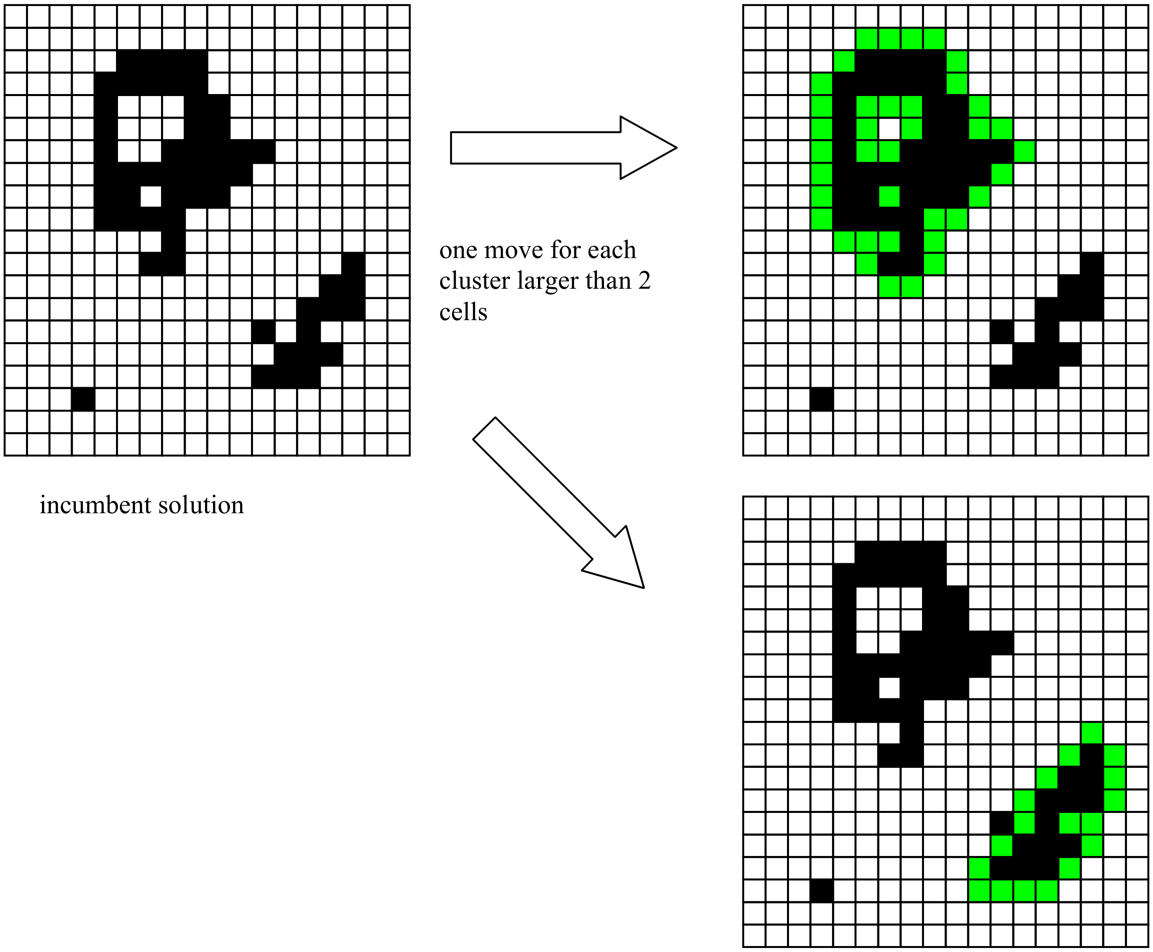


Figure 6.12 The hole filler neighborhood.

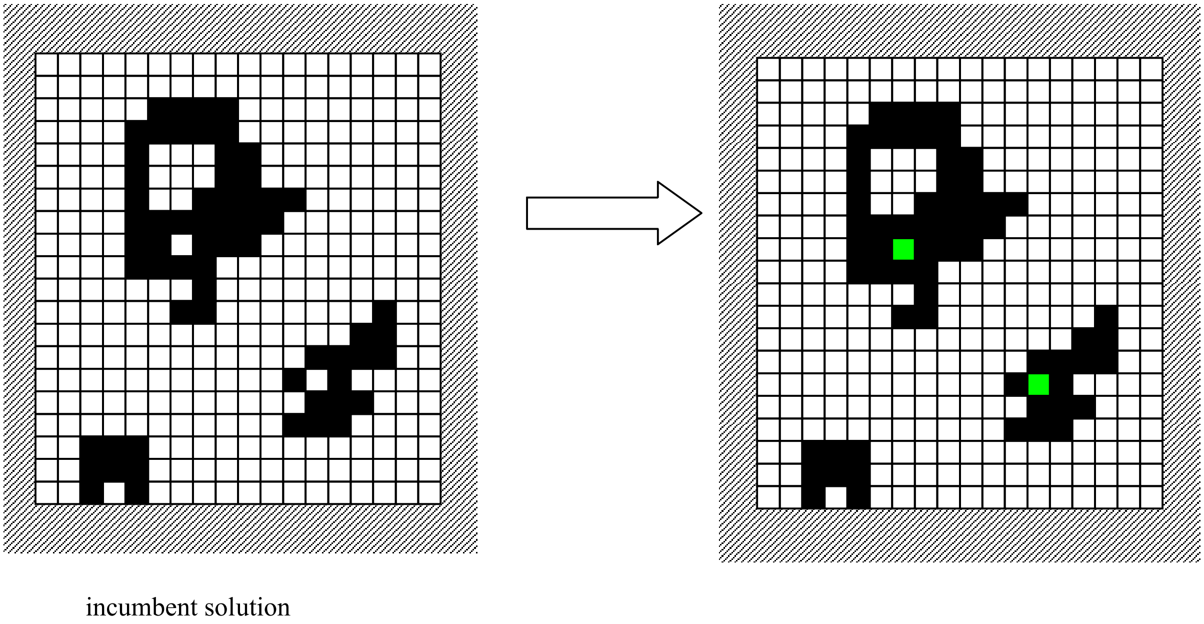


Figure 6.13 The crevice filler neighborhood.

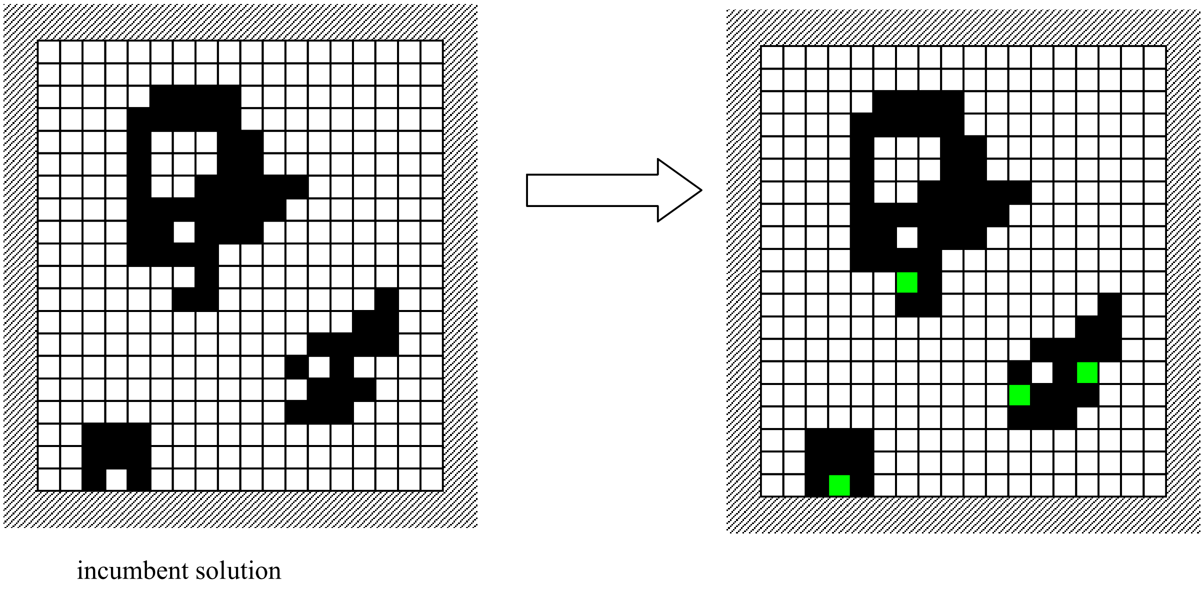


Figure 6.14 The singlet expand neighborhood

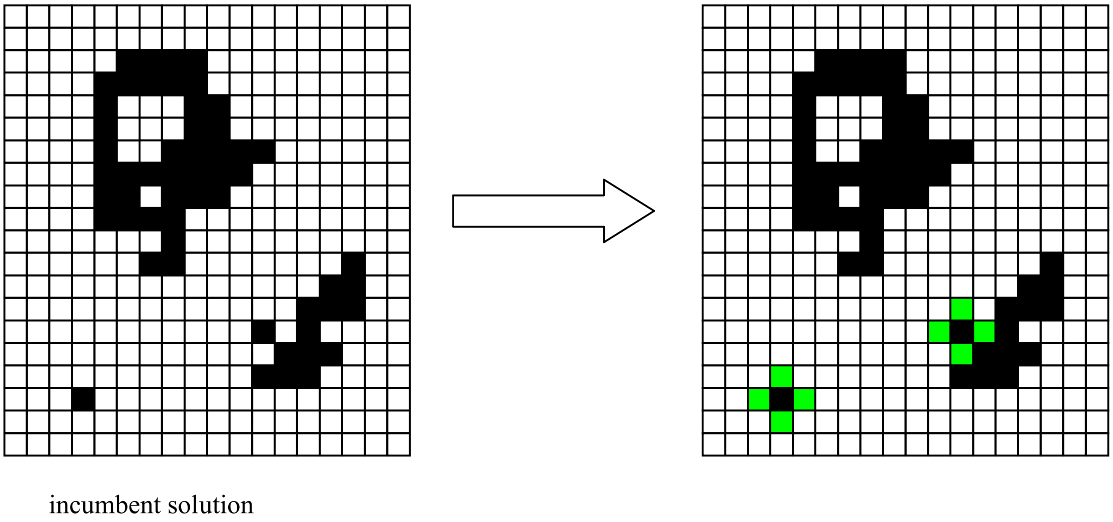
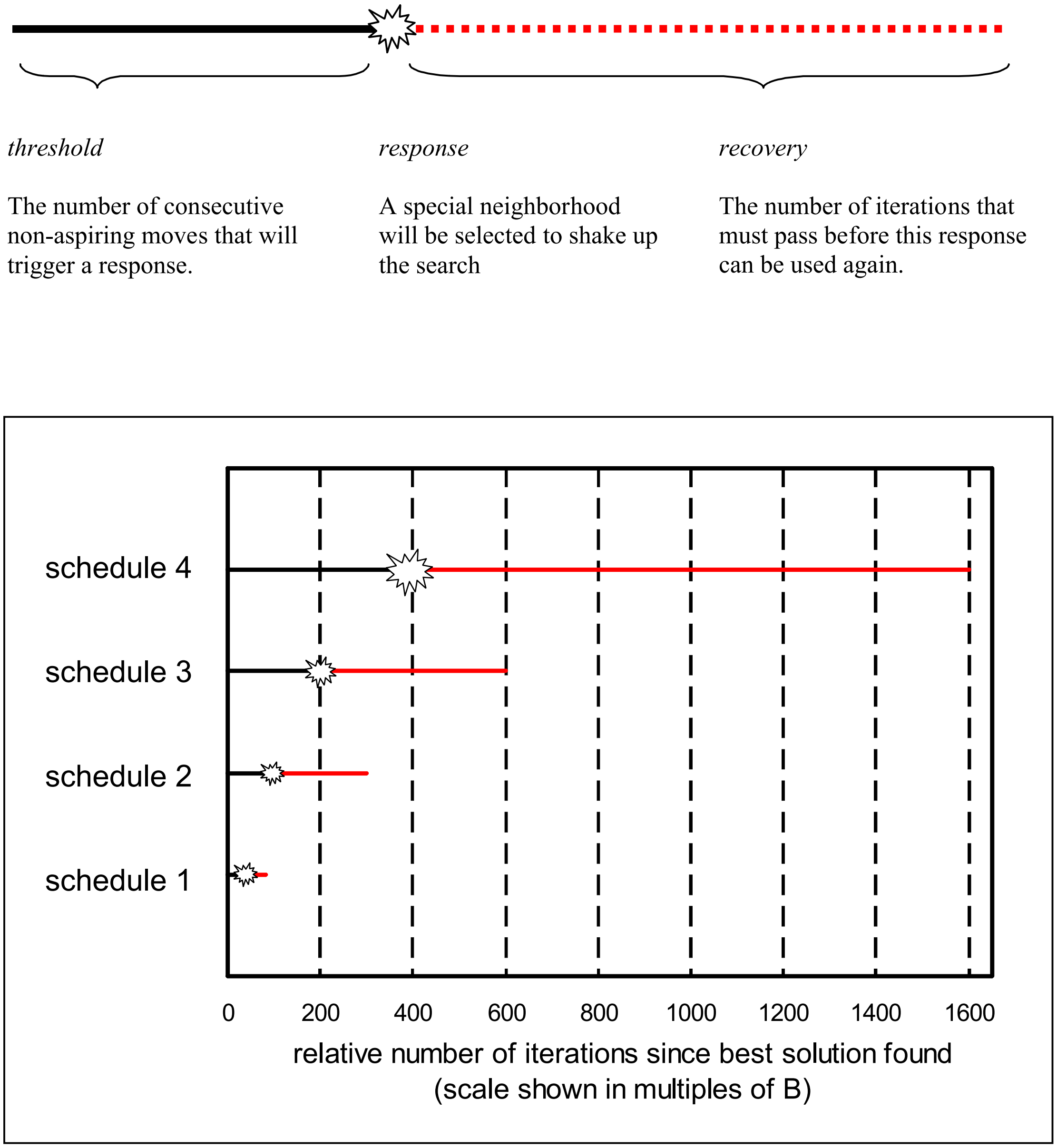


Figure 6.15 The schedules used by the dynamic neighborhood selection strategies consist of a threshold, response, and recovery. The size of the threshold and recovery are scaled with the problem size.



threshold

The number of consecutive non-aspiring moves that will trigger a response.

response

A special neighborhood will be selected to shake up the search

recovery

The number of iterations that must pass before this response can be used again.

Figure 6.16 The comparison for this rule based objective is broken into three different cases.

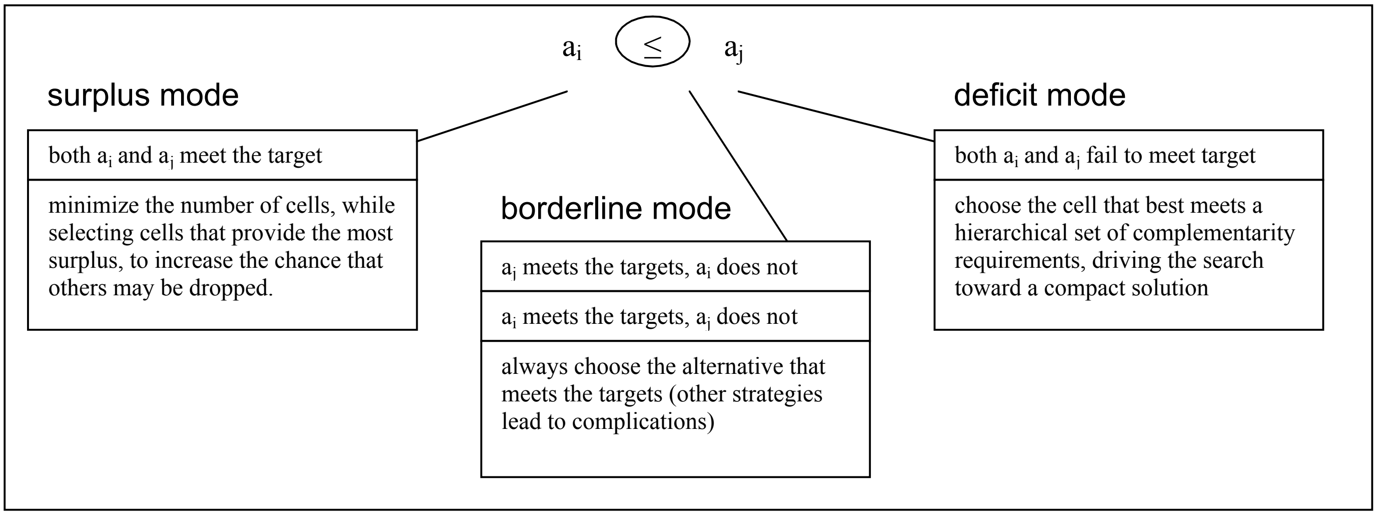

$a_i \leq a_j$
surplus mode
both a_i and a_j meet the target
minimize the number of cells, while selecting cells that provide the most surplus, to increase the chance that others may be dropped.
borderline mode
a_j meets the targets, a_i does not
a_i meets the targets, a_j does not
always choose the alternative that meets the targets (other strategies lead to complications)
deficit mode
both a_i and a_j fail to meet target
choose the cell that best meets a hierarchical set of complementarity requirements, driving the search toward a compact solution

Figure 6.17 Comparison rules for surplus mode in MDS-C.

```
prefer the alternative with fewer cells (nCells)
    prefer the alternative with more surplusSlack
        prefer the alternative with greater secondarySlack
            prefer the alternative with greater totalRepresentation
                prefer the alternative with fewer clusters (nClusters)
                     if there is a tie in the number of clusters, return a tie
```

Figure 6.18 Comparison rules for deficit mode in MDS-C.

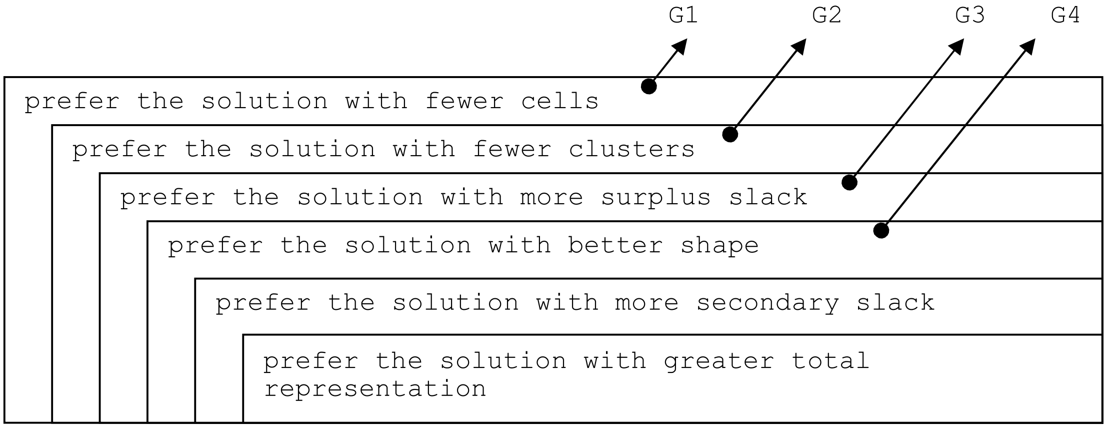
```
prefer the alternative with fewer deficient surrogates (nDeficientSurrogates)
    if both alternatives have the same number of cells (nCells)
        prefer the alternative with the smaller largest deficit (largestDeficit)
            prefer the alternative with the smallest remaining deficit (remainingDeficit)
                prefer the alternative with more surplus slack
                    prefer the alternative with the greater population (totalRepresentation)
                        prefer the alternative with fewer clusters (nClusters)
                            if there is a tie on the number of clusters, return tie

    else if ai has fewer cells than aj

    Q: when would we prefer an alternative with fewer cells
    A: as long as it doesn’t increase the largest deficit, AND it either improves the
    remaining deficit OR it is within two cells and does not increase the remaining deficit by
    more than 1 unit (a tradeoff).

    if{[ai.largestDeficit <= aj.largestDeficit] &&
            [(ai.remainingDeficit <= aj.remainingDeficit) ||
                    (aj.nSelectedCells - ai.nSelectedCells <= 2) &&
                    (ai.remainingDeficit – 2.0 <= aj.remainingDeficit)] }
                            prefer ai;
    else
            prefer aj;

    else aj has fewer cells than ai (symmetric to above)
```

Figure 6.19 An example of intransitivity in the rule based comparison operator MDS-C.

ID	nDeficientSpecies	nCells	largestDeficit	remainingDeficit
a_1	6	8	4	23
a_2	6	9	4	21
a_3	6	10	4	20

- a_1 is strictly preferred to a_2 because of the gambit rule
- a_2 is strictly preferred to a_3 because of the gambit rule
- a_3 is strictly preferred to a_1 because the difference in the remaining deficits is too large to invoke the gambit rule

Figure 6.20 The ITS comparison operator in surplus mode. Both alternatives have met the targets. Gambits are embedded in each layer of the hierarchy.


G1
G2
G3
G4
prefer the solution with fewer cells
prefer the solution with fewer clusters
prefer the solution with more surplus slack
prefer the solution with better shape
prefer the solution with more secondary slack
prefer the solution with greater total representation

G1	prefer the solution with fewer cells *except* when the other solution has: -better shape *and* -fewer or the same number of clusters *and* -and at least two units of surplus slack for each extra cell
G2	prefer the solution with fewer clusters *except* when the other solution has: -at least some margin of additional surplus slack
G3	prefer the solution with more surplus slack *except* when: -the difference in surplus slack is small (less than a defined margin) and -the shape of the other solution is verifiably better
G4	prefer the solution with better shape except when: -the difference is on the order of floating point error, this results in a tie

Figure 6.21 The *surplus mode* comparison operator for the intransitive shape objective (ITS). Source code located in `objective.shape.implement;`

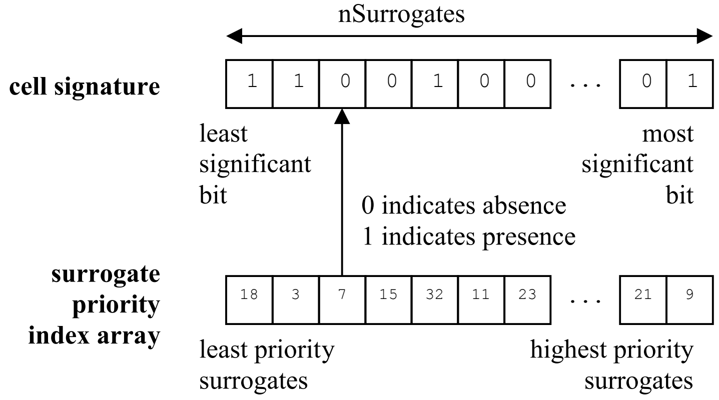
```
// define variables slackUnit, slackMargin, twoSlackUnits

if internal representation is integers between 0 and 100
        slackUnit = 50;
        slackMargin = 10;
else if internal representation is floating point
        slackUnit = 0.5;
        slackMargin = 0.1;
else if internal representation is presence/absence
        slackUnit = 1;
        slackMargin = 1;

twoSlackUnits = slackUnit*2;

// define variables shapeGap and shapeTolerance
// use estimates of the average values that may be encountered

totalPerimeter = 0;
for(int i = 0, n = grid.getNCells(); i < n; i++) {
        totalPerimeter += grid.getAbsolutePerimeter(i);
}
shapeGap = (totalPerimeter/grid.getTotalArea())*1e-4;
shapeTolerance = (totalPerimeter/grid.getTotalArea())*1e-5;
```

```
// method that performs comparison (only surplus mode is shown)

public int compare(GeneralShapeScore ai, GeneralShapeScore aj) {

if(ai.nDeficientSpecies == 0 && aj.nDeficientSpecies == 0) {
// both solutions meet the targets
// prefer the solution with fewer cells
        if(ai.nSelectedCells < aj.nSelectedCells) {
        // G1:  when would we accept a solution with more cells?
        // G1:  if it has a better shape, similar or better clustering, and boosts the surplus slack by two units per added cell
                if(aj.perimeterAreaRatio + shapeGap < ai.perimeterAreaRatio &&
                   aj.nClusters <= ai.nClusters &&
                   aj.surplusSlack-ai.surplusSlack >= (aj.nSelectedCells-ai.nSelectedCells)*twoSlackUnits)
                return AJ;
                else
                        return AI;
        }
```

```
else if( aj.nSelectedCells < ai.nSelectedCells) {
        // G1:  symmetric case
        if(     ai.perimeterAreaRatio + shapeGap < aj.perimeterAreaRatio &&
                ai.nClusters <= aj.nClusters &&
                ai.surplusSlack-aj.surplusSlack >= (ai.nSelectedCells-aj.nSelectedCells)*twoSlackUnits)
                return AI;
        else
                return AJ;
}
else {
        // tie on the number of cells, prefer the solution with fewer clusters
        if(ai.nClusters < aj.nClusters) {
                // G2:  when would I be willing to prefer a solution with more clusters?
                // G2:  if it gives any extra slack (above the margin)
                if(aj.surplusSlack - ai.surplusSlack >= slackMargin)
                        return AJ;
                else
                        return AI;
        }
        else if(aj.nClusters < ai.nClusters) {
                // G2:  symmetric case
                if(ai.surplusSlack - aj.surplusSlack >= slackMargin)
                        return AI;
                else
                        return AJ;
        }
        else {
                // tie on the number of clusters, fairly common because it is an integer
                // prefer the solution with more surplus slack
                if(ai.surplusSlack > aj.surplusSlack) {
                        // G3:  when would we prefer a solution with less surplus slack?
                        // G3:  if the difference in slack is small and the shape of the other solution is better
                        if(  ai.surplusSlack - aj.surplusSlack < slackMargin &&
                             aj.perimeterAreaRatio + shapeTolerance < ai.perimeterAreaRatio)
                                return AJ;
                        else
                                return AI;
                }
```

```
            else if(ai.surplusSlack < aj.surplusSlack) {
                // G3:  symmetric case
                if(  aj.surplusSlack - ai.surplusSlack < slackMargin &&
                     ai.perimeterAreaRatio + shapeTolerance < aj.perimeterAreaRatio)
                    return AI;
                else
                    return AJ;
            }
            else {
                // tie on surplus slack, prefer the solution with demonstrably better shape
                // G4:  they have to be different by an amount greater than the tolerance
                if(ai.perimeterAreaRatio + shapeTolerance < aj.perimeterAreaRatio)
                    return AI;
                else if(ai.perimeterAreaRatio > aj.perimeterAreaRatio + shapeTolerance)
                    return AJ;
                else {
                    // tie on shape, prefer more secondary slack
                    // secondary slack is contained in variable “critical deficit”
                    if(ai.criticalDeficit < aj.criticalDeficit)
                        return AJ;
                    else if(ai.criticalDeficit > aj.criticalDeficit)
                        return AI;
                    else {
                        // tie on secondary slack, optimize total population
                        if(ai.totalPopulation < aj.totalPopulation)
                            return AJ;
                        else if(ai.totalPopulation > aj.totalPopulation)
                            return AI;
                        else
                            return TIE;
                    }
                }
            }
        }
    }
} // end surplus mode comparison operator
```

Figure 6.22 The creation of cell signatures from a surrogate priority index array.



The cell signature is a binary BigInteger with one bit for every surrogate. This integer creates a score for the cell that can be used to compare it to other cells.

The index array contains the indices of the species sorted for some criteria (perhaps rarity). The 0th position will represent 2^0 and the ith position represents 2^i. Thus, the surrogates at the end of the array represent the most significant bits.

Figure 6.23 The rationale for the most deficient surrogate heuristic.

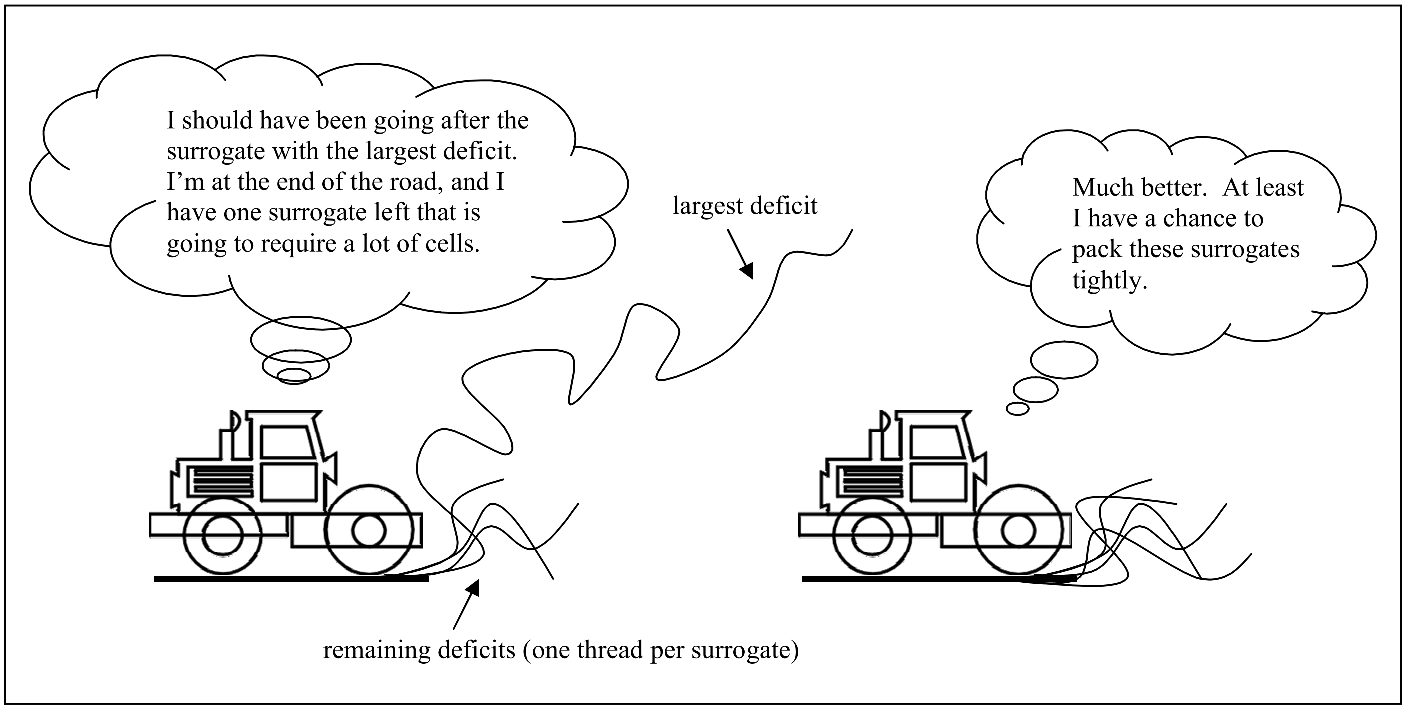

I should have been going after the surrogate with the largest deficit. I'm at the end of the road, and I have one surrogate left that is going to require a lot of cells.
largest deficit
Much better. At least I have a chance to pack these surrogates tightly.
remaining deficits (one thread per surrogate)

Figure 6.24 The virtual cell comparison rules for the forward part of the MDS heuristic algorithm.

```
prefer the cell with the largest T-complementarity (target based complementarity)
    prefer the cell with the largest cell signature (contains most deficient species)
        prefer the cell with the greatest F-complementarity (total richness)
            prefer the cell with the smaller ID (arbitrary lexical ordering)
```

Figure 6.25 The virtual cell comparison rules for the backward portion of the MDS heuristic algorithm.

```
prefer the cell with the smaller cell signature (contains only high surplus surrogates)
    prefer the cell with the smaller total richness
        prefer the cell with the smaller ID (arbitrary lexical ordering)
```

Figure 6.26 The virtual cell comparison rules for the forward portion of the RF heuristic algorithm.

```
prefer the cell with the largest cell signature (favors rare surrogates)
    prefer the cell with the largest T-complementarity (target based complementarity)
        prefer the cell with the greatest F-complementarity (total richness)
            prefer the cell with the smaller ID (arbitrary lexical ordering)
```

Figure 6.27 Total richness plot for the Mexico data set. The richest cells, shown in white, contain about 27 surrogates (out of 86).

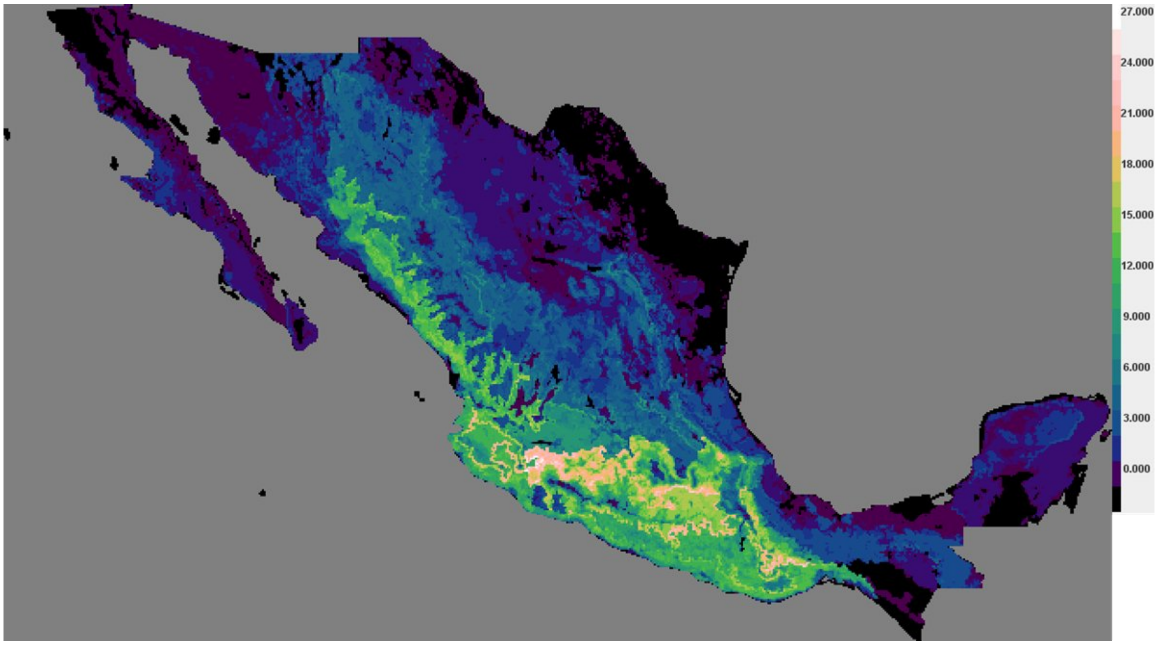


Figure 6.28 The MDS2 heuristic solution contains 4116 cells and 639 clusters. This initial solution takes about 5 seconds to build, and serves as a good starting point for the tabu search.

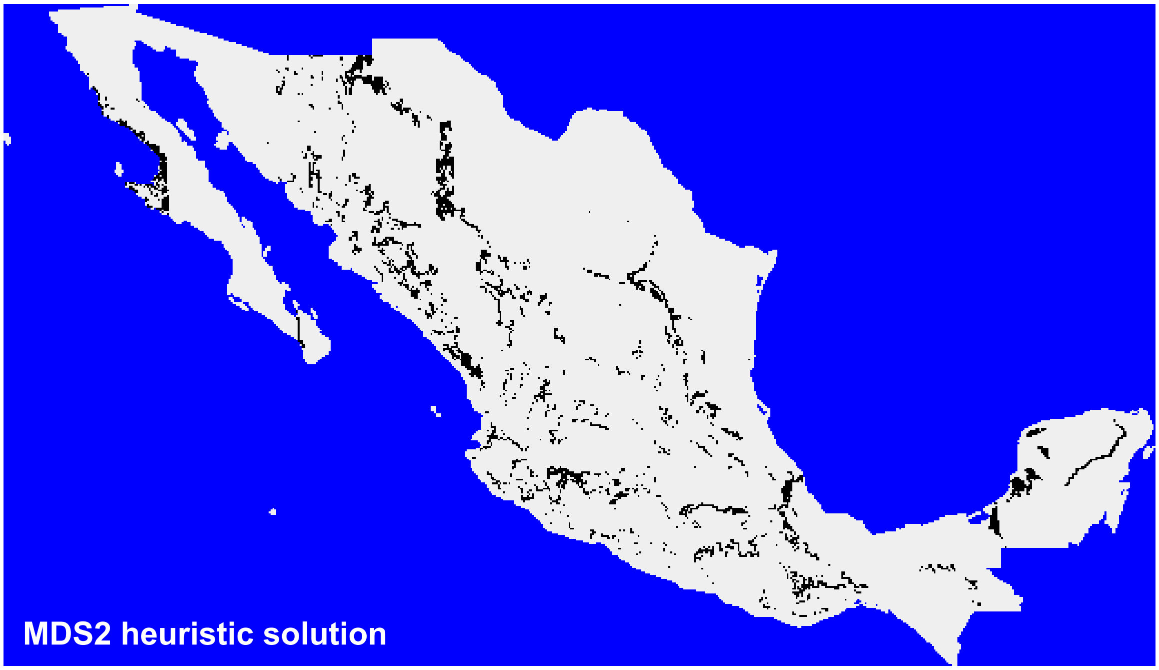
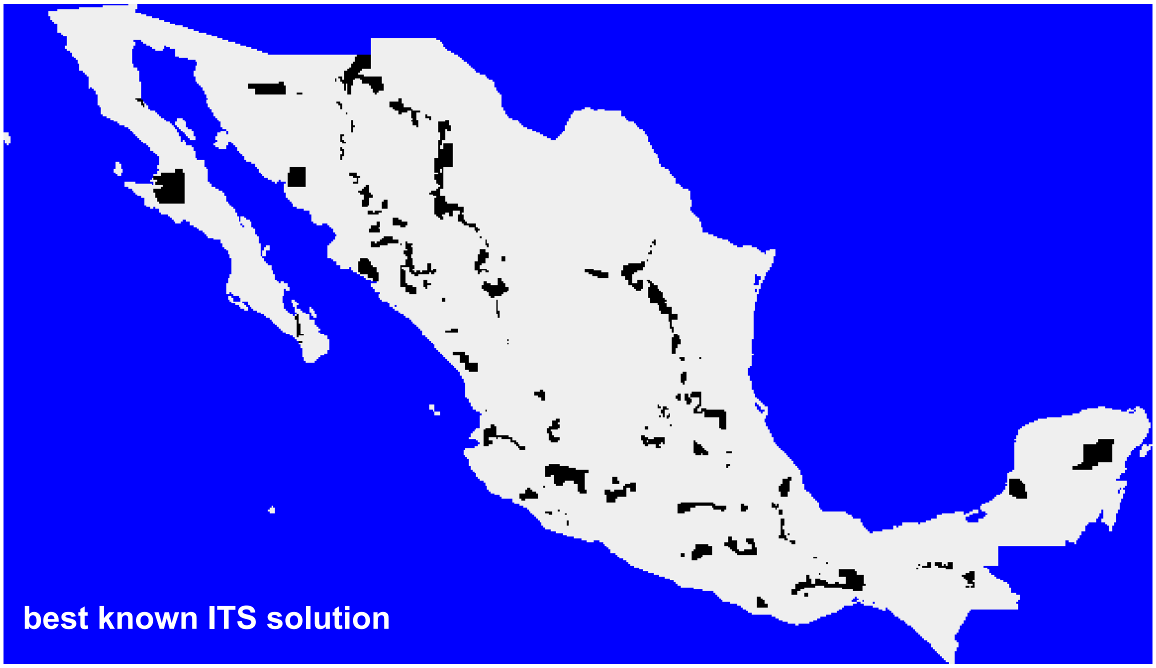


Figure 6.29 *(top)* This solution was obtained using the ITS (intransitive shape) objective. With 4105 selected cells, it is a minimal set cover. In addition, it has an excellent perimeter to area ratio (0.147). The search began with the MDS2 heuristic solution. This highly refined solution required about 700,000 iterations, but excellent solutions were found in under 100,000 iterations. *(bottom)* A comparison between the best known ITS solution (green) and the MDS2 heuristic solution where the search started (pink). Cells in common to both solutions are shown in black.



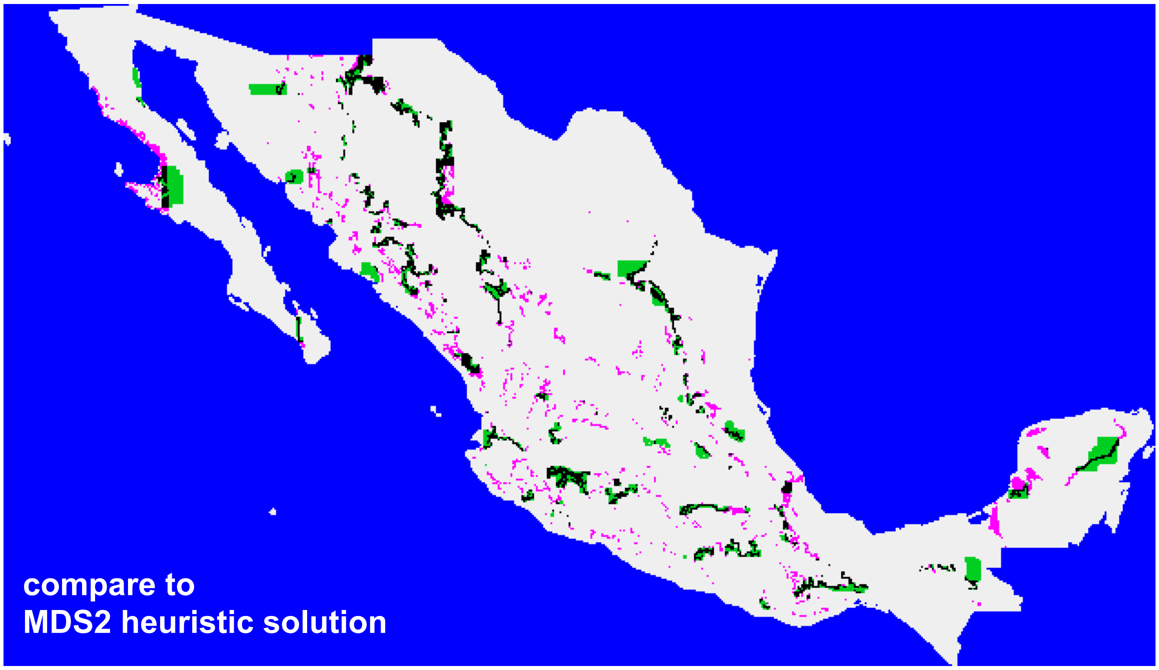


Figure 6.30 Total richness plot for West Virginia. The richest cells, shown in white, contain about 281 surrogates (out of 353).

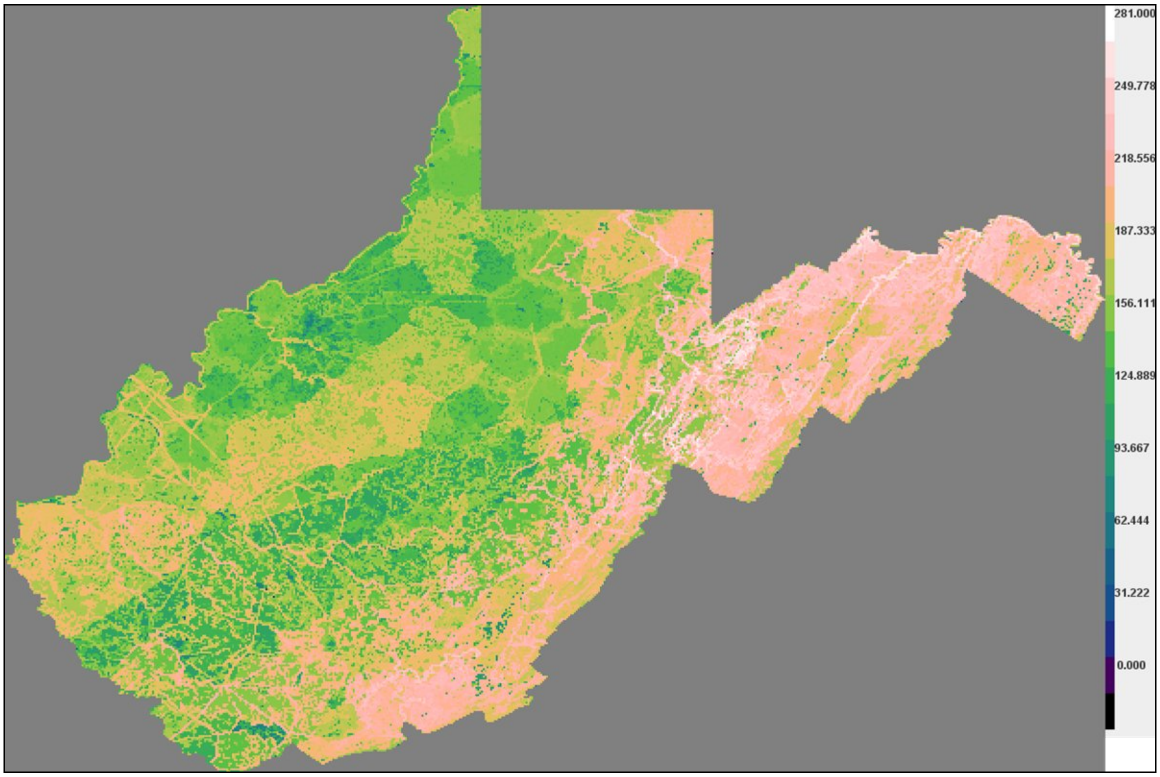
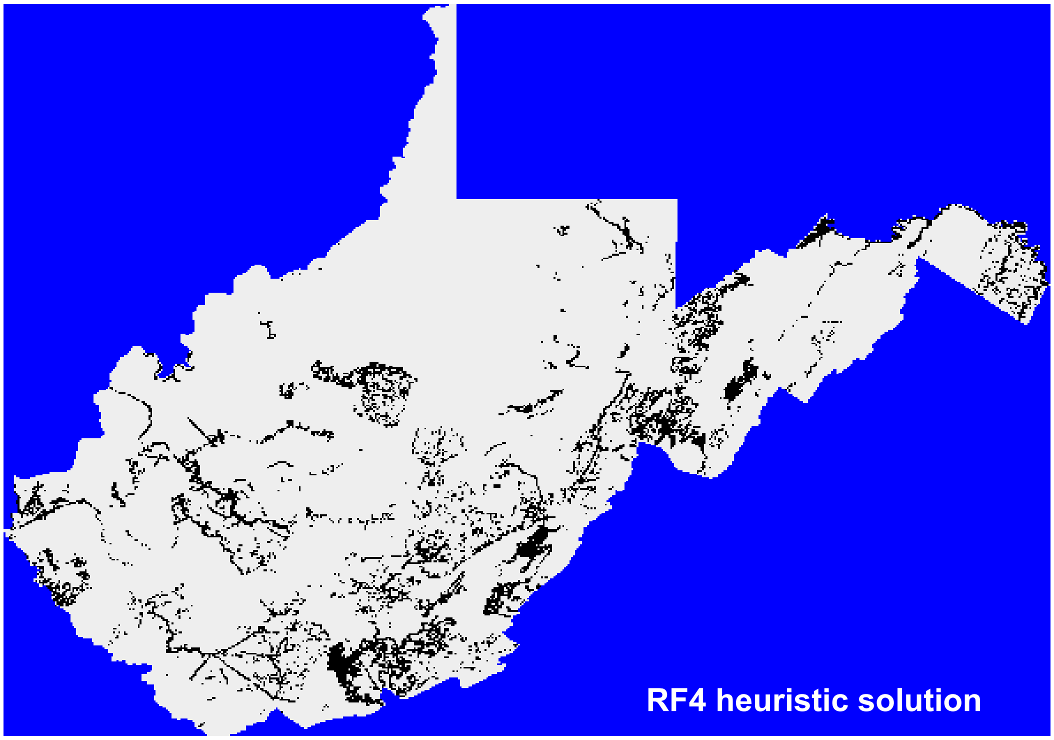
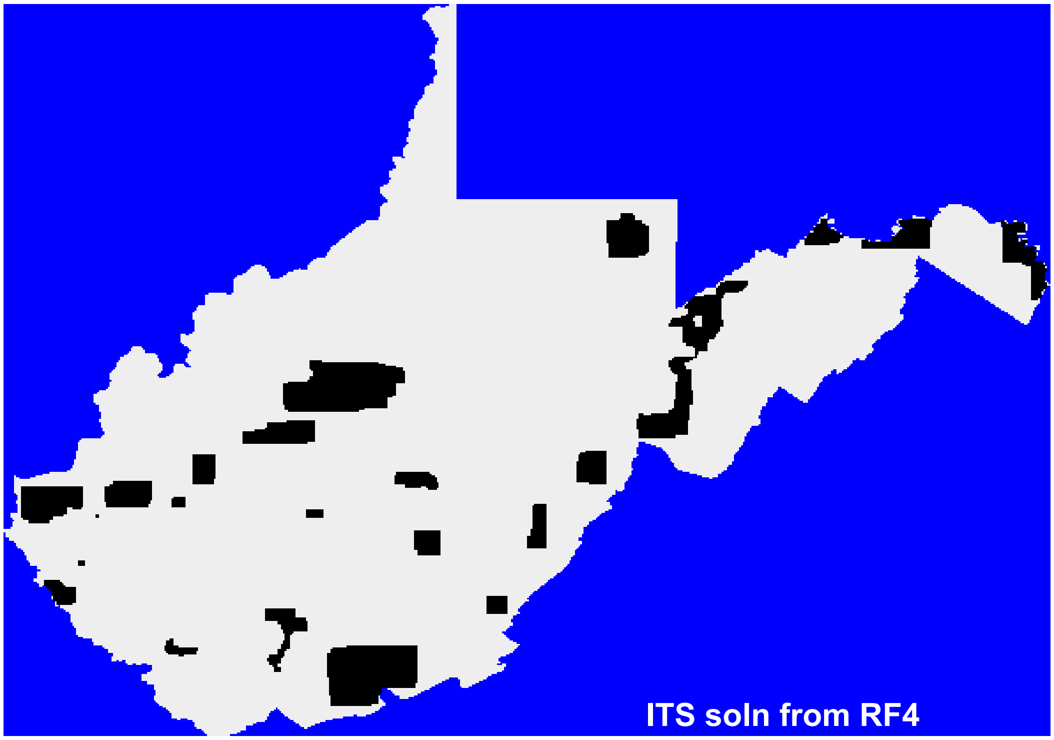


Figure 6.31 The RF4 heuristic found a minimal set cover (9477 cells), but the solution is not compact.





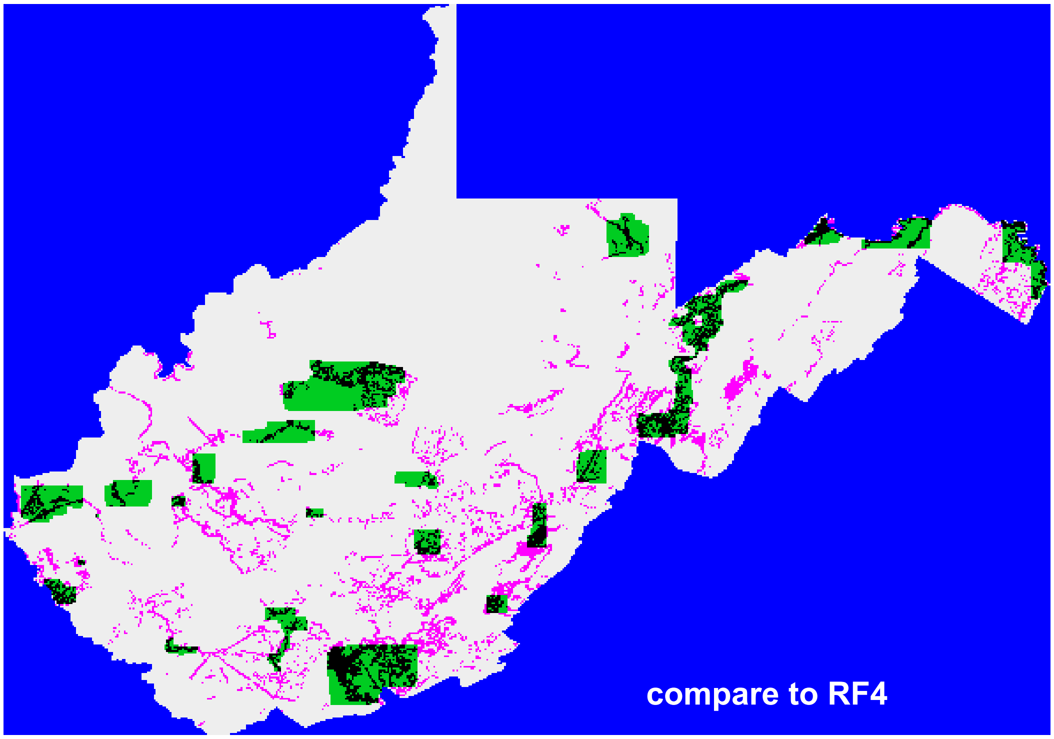
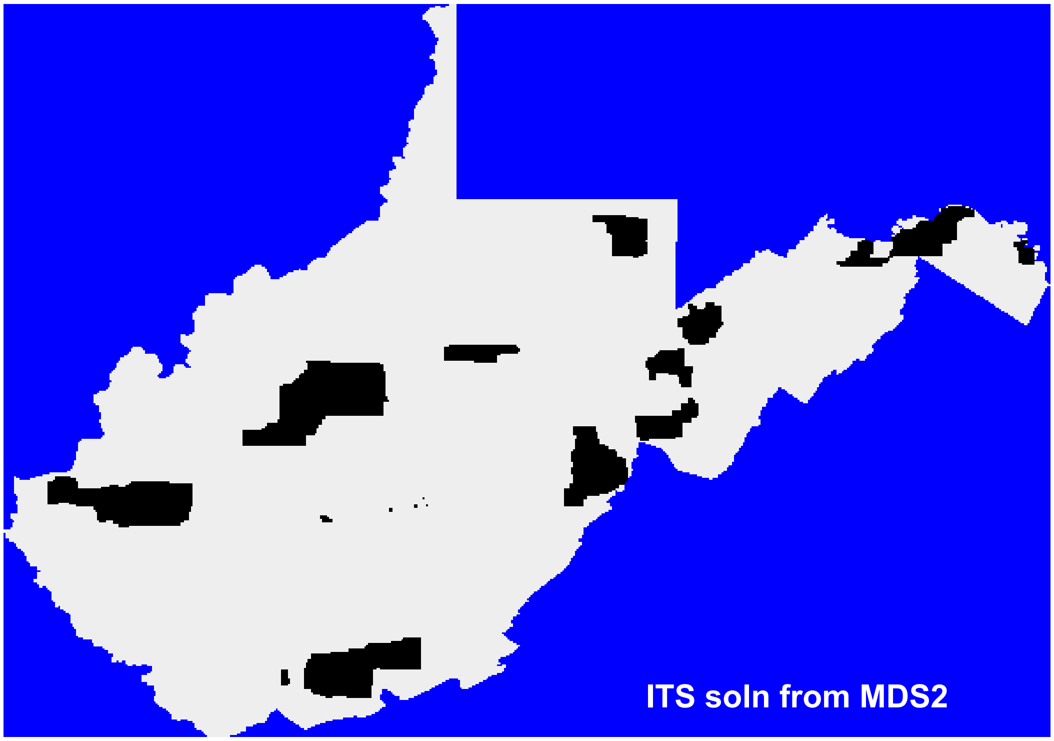


Figure 6.32 The ITS (min cells and shape) objective improved the compactness considerably, while maintaining a minimal set cover of 9477 cells. The comparison below shows the starting point of the search (shown in pink) and the final solution (in green). Overlapping cells are shown in black.



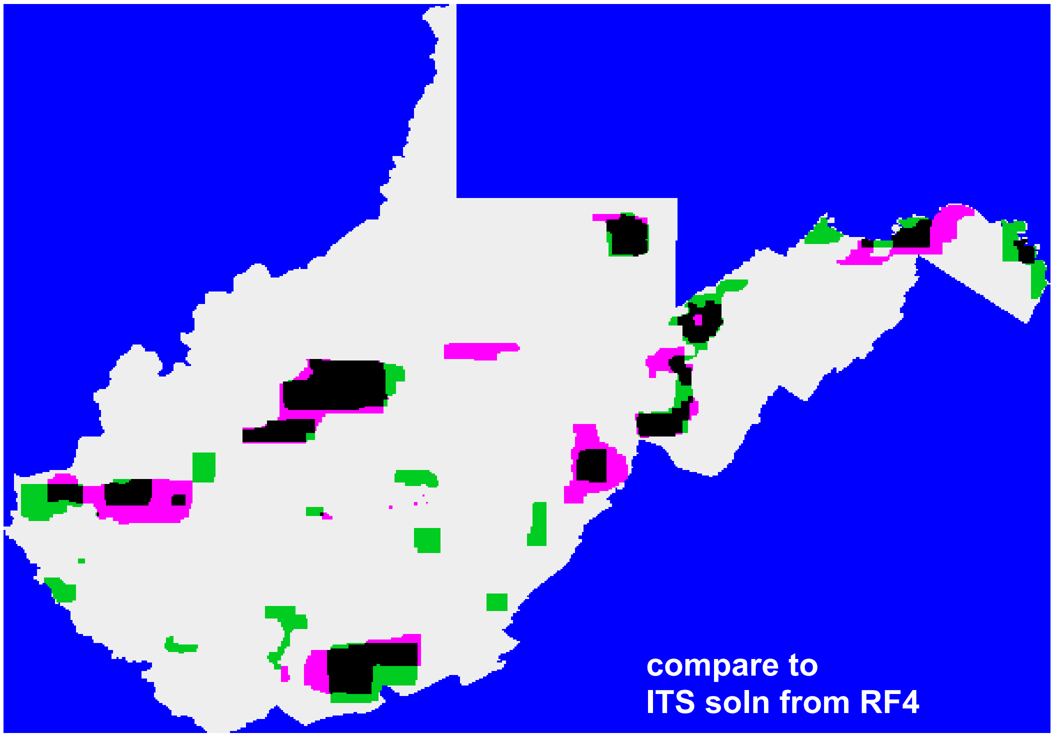


Figure 6.33 The ITS objective was also run starting from the MDS2 heuristic solution, leading to a conservation network with slightly better shape. The "ITS soln from MDS2" is shown in pink, and the "ITS soln from RF4" is shown in green. Cells common to both solutions are shown in black.

Figure 6.34 Total richness plot for the Indoburma data set. The surrogate representation is provided as an expected value. The white warm colors indicate cells with high richness. The data set contains 294,830 cells and 184 surrogates.

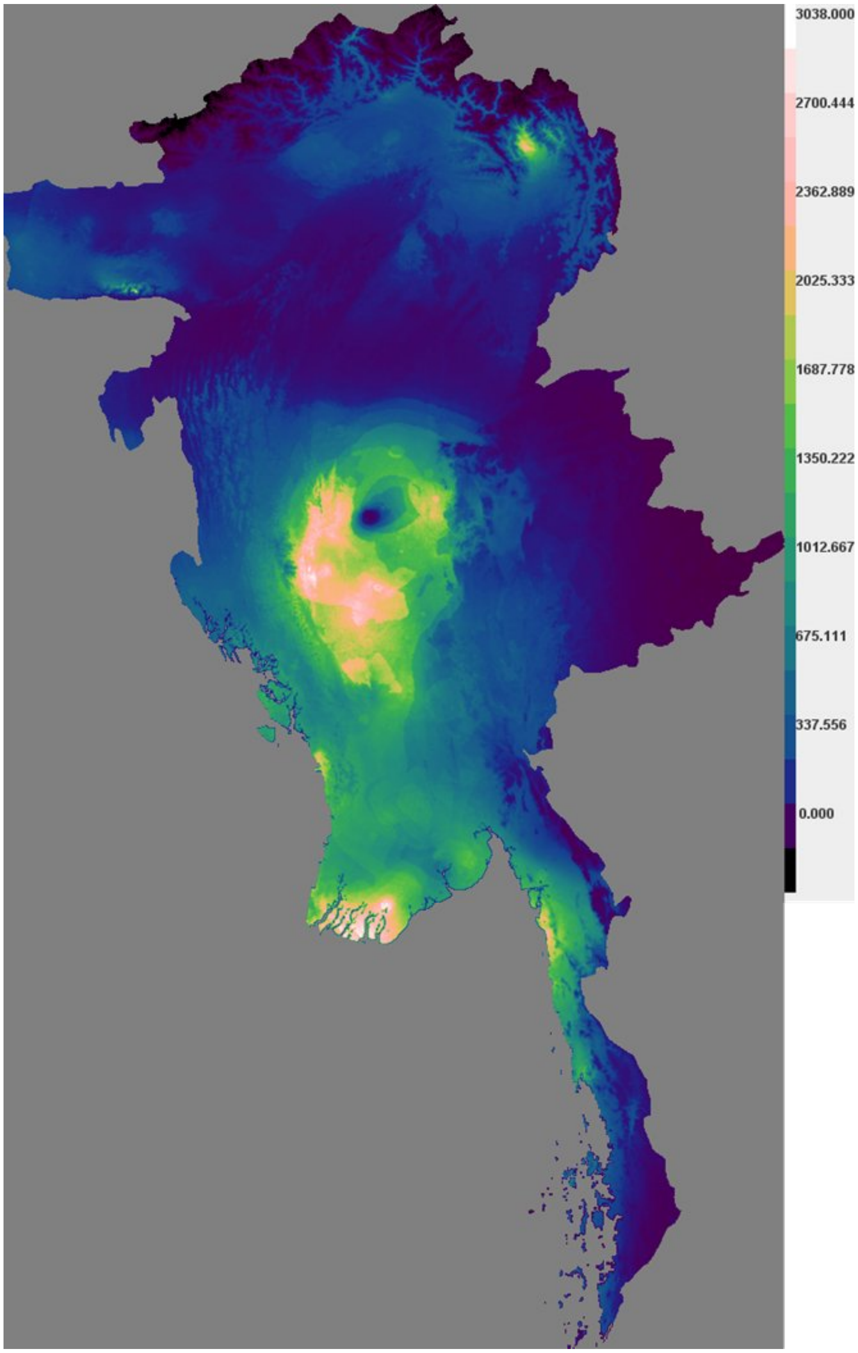


Figure 6.35 The MDS2 heuristic solution quickly identifies some regions of interest, but the solution is inferior.

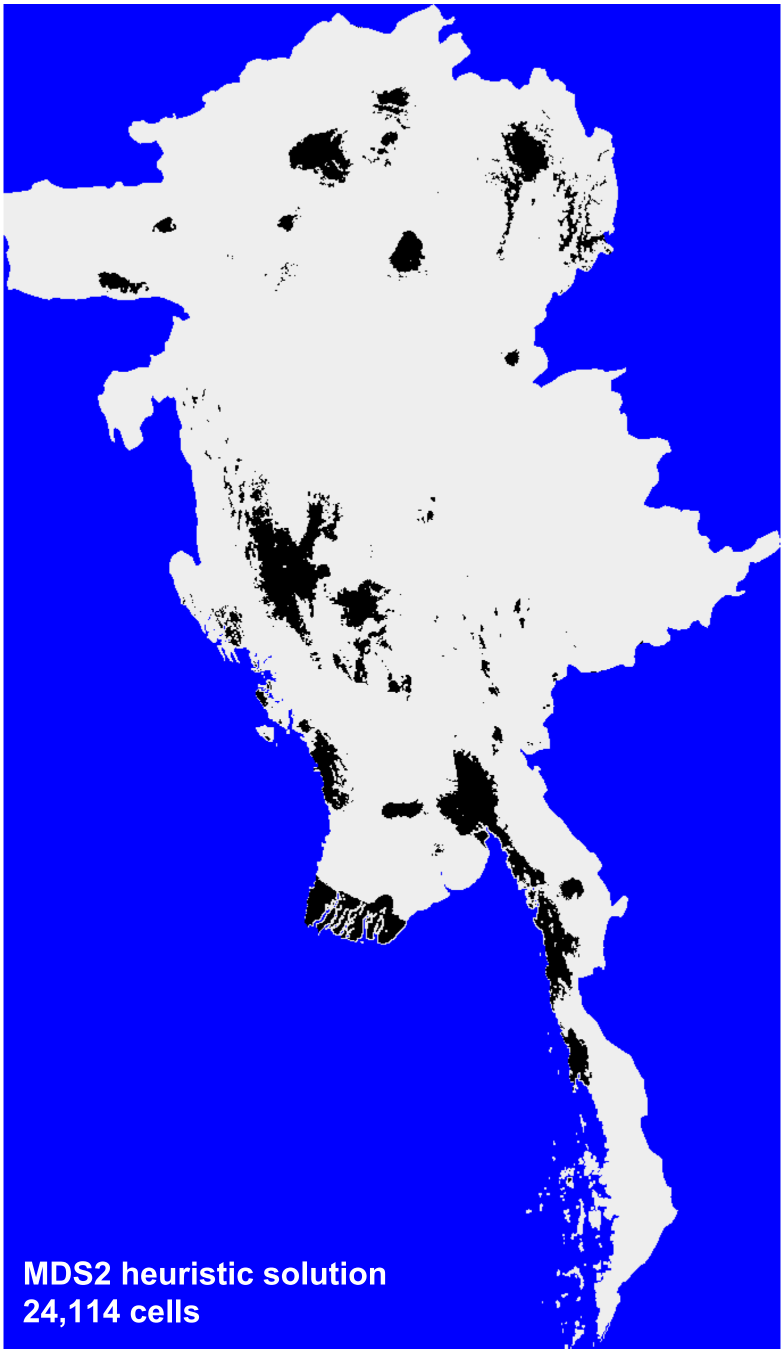


Figure 6.36 Starting from the MDS2 heuristic solution, the MDSC objective guided the search to a solution with 22,001 cells (an 8.7% reduction in the number of cells). However, this solution is highly scattered because the MDSC objective does not respect the shape of the network.

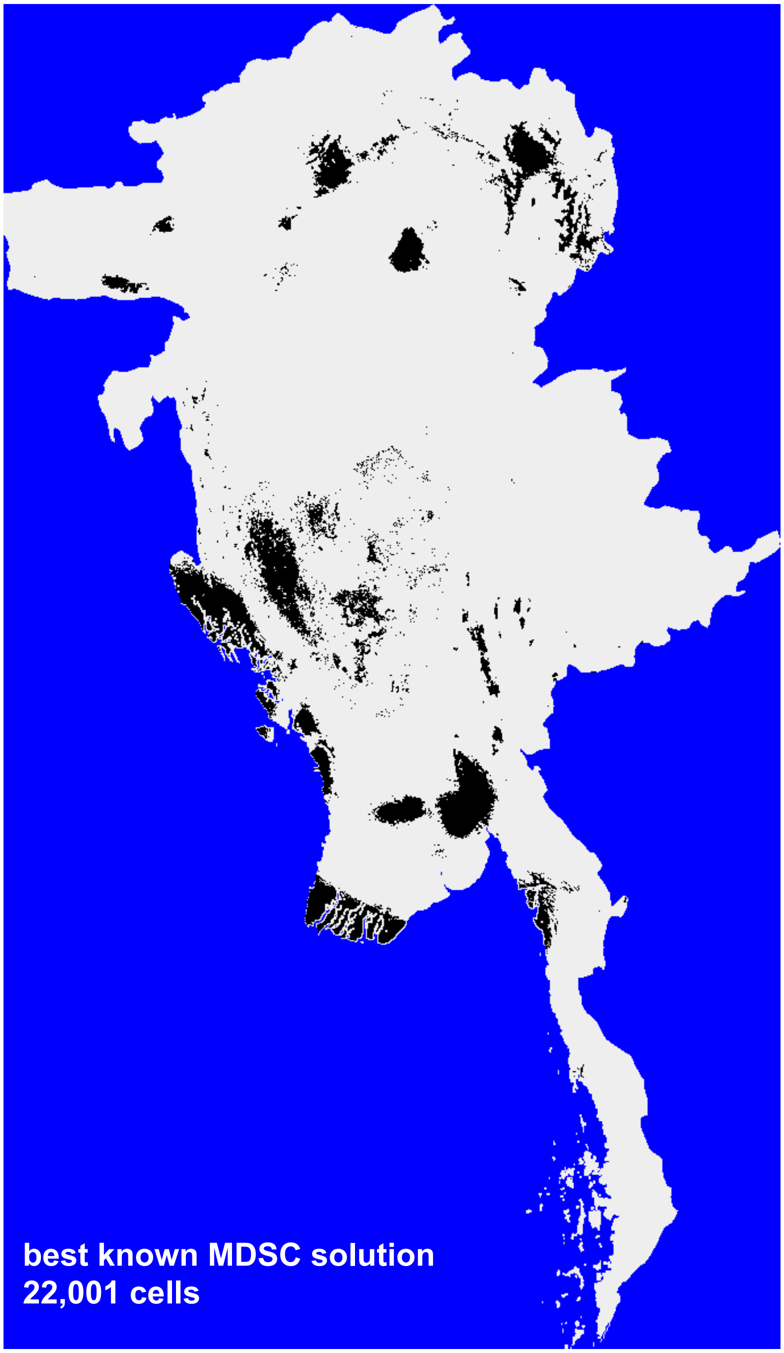


Figure 6.37 Starting from the MDS2 heuristic solution, the ITS objective guided the search to a solution with 22,160 cells (an 8.1% reduction in the number of cells). Although it contains 159 more cells than the best known set cover, it is much more compact.

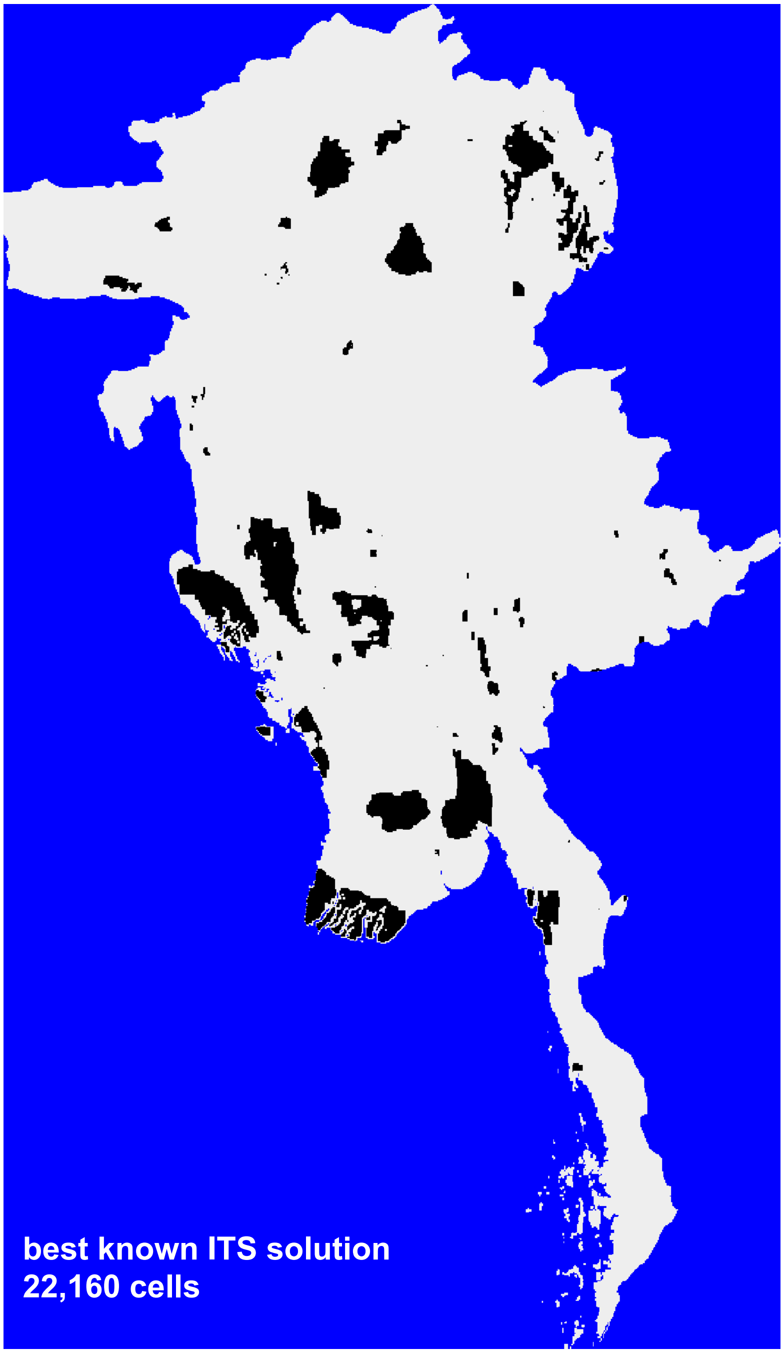


Figure 6.38 A user defined gMCA objective function led to this solution. The objective addressed many criteria, including the number of cells, the shape, the number of clusters, the surplus slack, and the total representation.

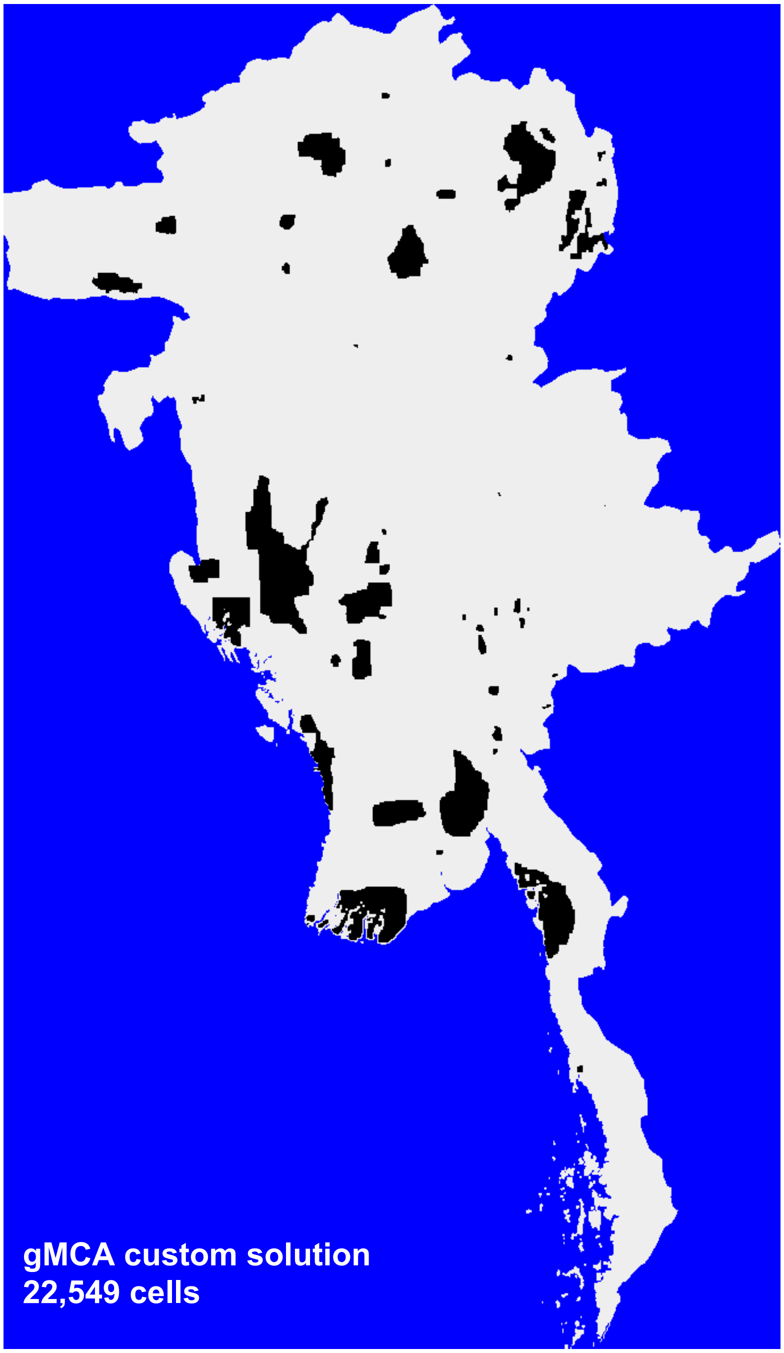


Figure 6.39 The basic DNS strategy outperforms random neighborhood selection when building a minimal set cover for the mexicoT0 data set. The search starts with no cells selected, builds up to a feasible solution, and then attempts to find a minimal cover by removing cells. The basic strategy found a minimal cover at iteration 6,527. The random approach was not very effective at removing cells, and did not locate a minimal cardinality solution.

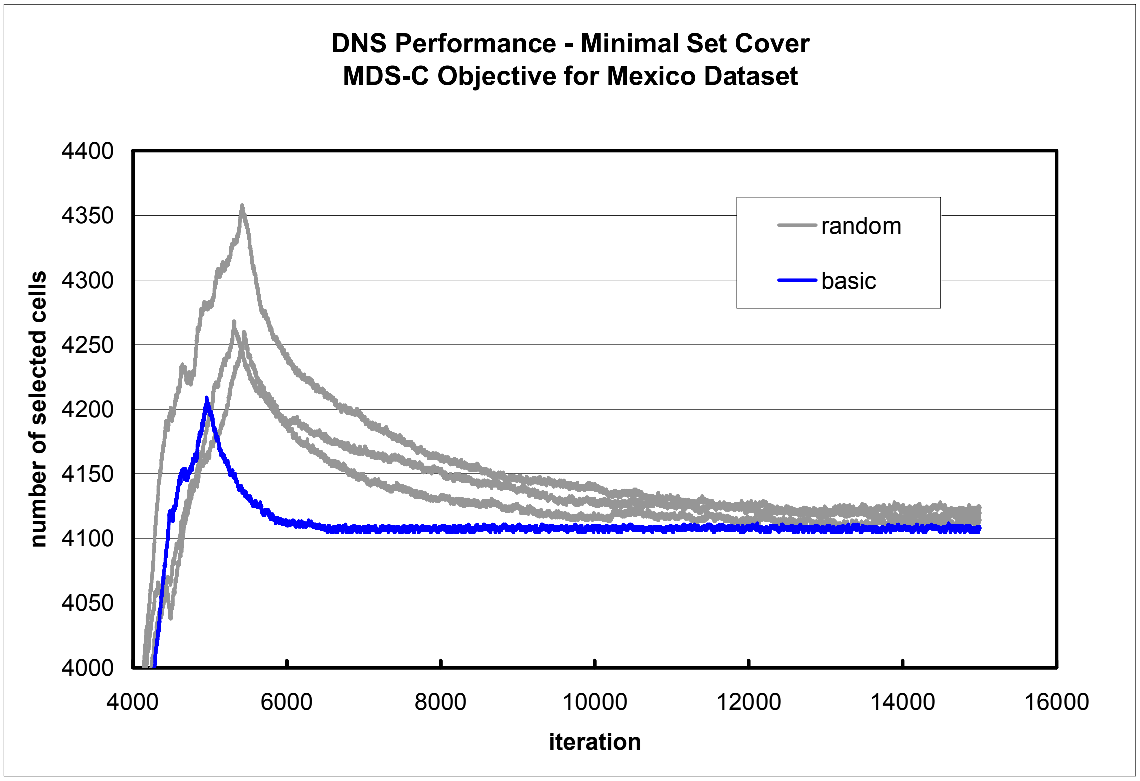


Figure 6.40 The ITS objective attempts to simultaneously minimize the shape of the conservation area network and the number of selected cells. Shown below is the Pareto frontier explored by the search using a variety of DNS approaches. Each trial was allowed to run for 1.7 billion evaluations (~250,000 iterations). Basic DNS dominates random neighborhood selection. The standard and advanced methods expand the Pareto frontier even further.

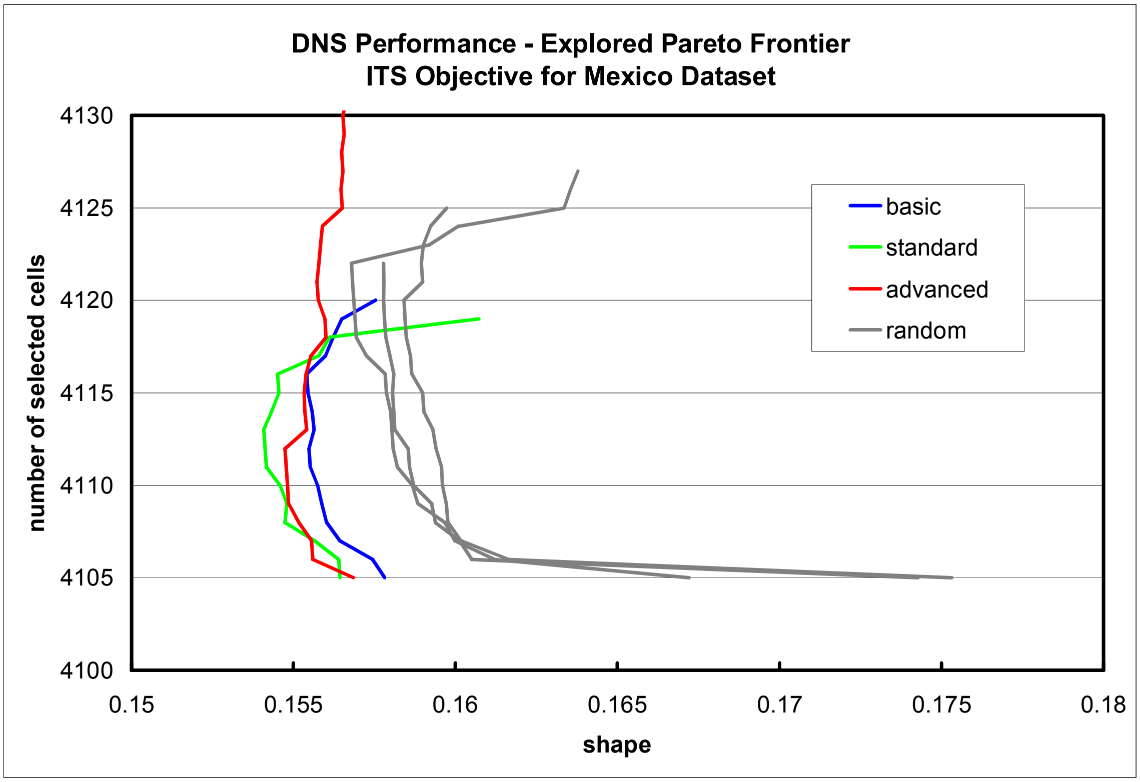


Figure 6.41 Basic DNS outperforms random neighborhood selection when attempting to find a minimal set cover, finding higher quality solutions considerably faster. Standard DNS exhibits marginal improvements over basic DNS.

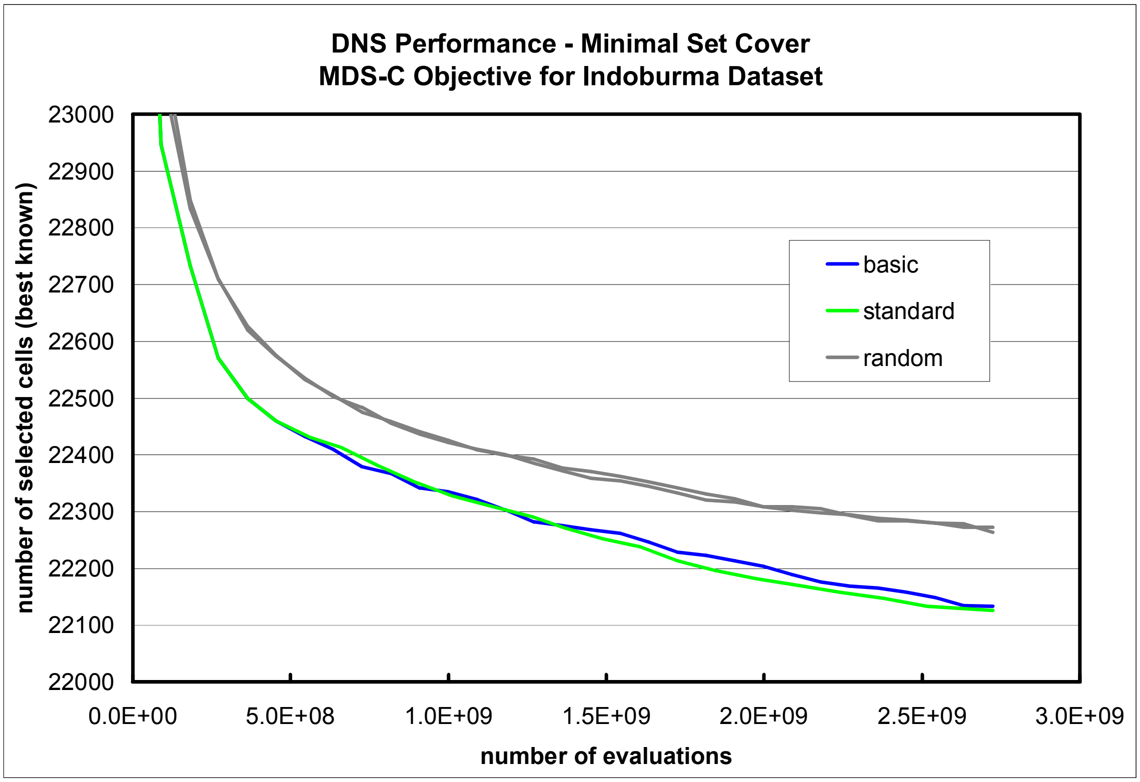
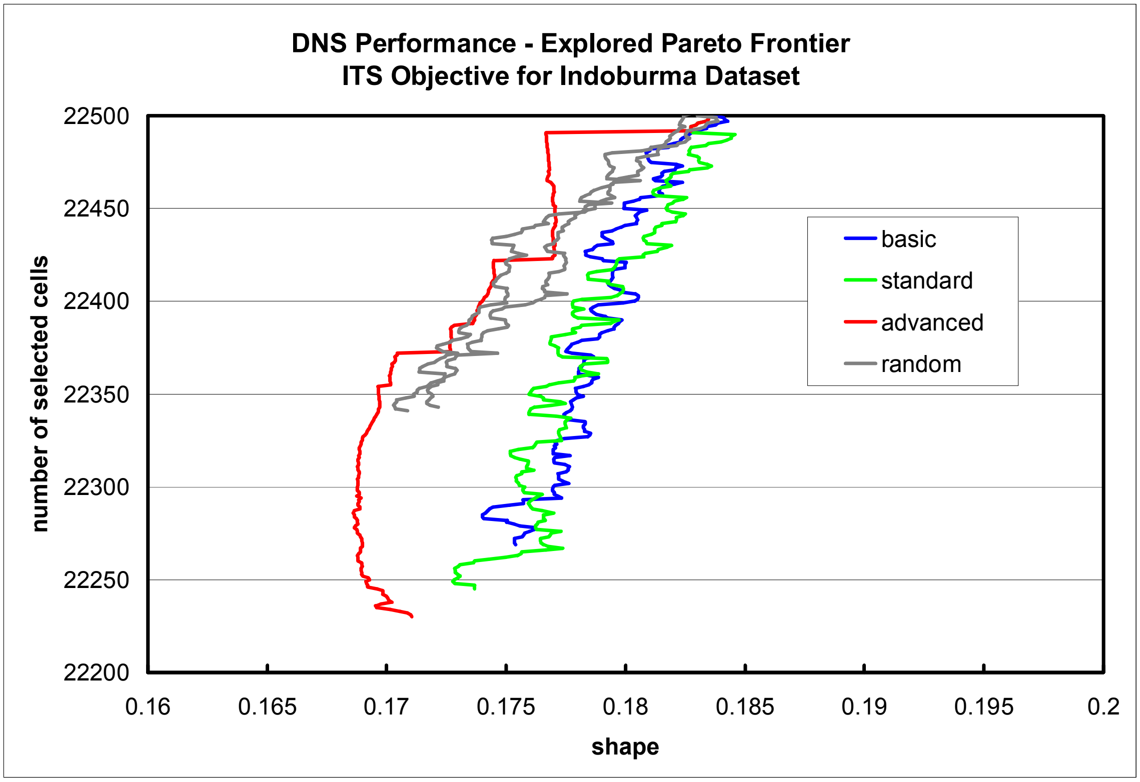


Figure 6.42 The ITS objective attempts to simultaneously minimize the shape of the conservation area network and the number of selected cells. Shown below is the Pareto frontier explored by the search using a variety of DNS approaches. Each trial was allowed to run for 4.5 billion evaluations (~250,000 iterations). Compared to random neighborhood selection, the basic and standard DNS strategies find solutions with fewer cells, which is considerably harder than minimizing the shape. The advanced DNS strategy dominates the other methods.



7 Conclusions

The reusable MASTS software documented in this dissertation can be applied to a wide variety of problems, as demonstrated by the two diverse examples presented in Chapters 5 and 6. The API is sufficiently generic to describe complex decision models as well as other search components such as moves, neighborhoods, and objectives. Clients who extend MASTS also receive many rich features, including a detailed GUI, multi-threading, multi-objective management, and solution archiving. With this "built in" support, MASTS can be used to rapidly prototype and deploy advanced search methodologies.

MASTS advances the state of the art with two new strategies: rule based objectives (RBO) and dynamic neighborhood selection (DNS). RBOs use binary comparison operators (rather than traditional numeric scores) to rank solutions and make decisions at each iteration in the search. RBOs allow users to specify precise ordinal rankings, particularly in a multi-criteria setting. In addition, defining a custom comparison operator $\preceq_R$ to govern the choice behavior of the search allows users to embed gambits which can improve search performance by exploiting problem specific features. These gambits could make $\preceq_R$ intransitive and empirical results show that structured intransitivity actually enhances the search performance by evading superfluous local optima in the search space.

Another novel strategy, DNS, when properly implemented, greatly improves the search performance by directing the search to promising regions and economizing the number of evaluations required during exploration. An effective DNS strategy will "learn" about the search landscape and implement an adaptive intensification-diversification strategy. The strategy must consider both the objective function f and the solution space X, to gain better understanding of the search landscape $\mathcal{L}$ (X, f, N). Empirical results in ConsNet show that fairly simple DNS strategies allow the search to find solutions that are not easily obtained with simpler techniques such as random neighborhood selection.

MASTS demonstrates extraordinary flexibility as both an advanced search engine and a decision aid. The highly integrated software, including the advanced graphical user interface, is designed to support ongoing interactive analysis, allowing users to analyze results, build portfolios of preferred solutions, and save their progress. The ability to examine the problem from different perspectives generates dialogue and enhances problem understanding.

In two separate groundwater planning projects, MASTS has been a focal point for discussion, analysis, and planning. In the Barton Springs Segment of the Edwards Aquifer, MASTS illustrated the impacts of different spatial configurations of pumping, and is being used by the BSEACD to explore different drought policies. In GMA9 (the Texas Hill Country), MASTS software plays a critical role in helping district managers investigate their future water resources. While a firm objective has not been established, extended tests using a solution archive (about 7000 model runs) validate the possibility of real time search to support rapid dispute resolution.

MASTS has also demonstrated *unprecedented* success in the spatially coherent design of conservation area networks, surpassing the capabilities of previous exact and heuristic methods. A combination of novel algorithms, dynamic neighborhood selection, and highly effective rule based objectives allow the search to *rapidly* identify efficient solutions. The superior capabilities demonstrated by this extension of the MASTS software, ConsNet, will place it at the forefront of national and international planning efforts.

References

Alagador, D., & Cerdeira, J. O. (2007). Designing spatially-explicit reserve networks in the presence of mandatory sites. *Biological Conservation, 137*(2), 254-262.

Anand, P. (1993). The Philosophy of Intransitive Preference. *The Economic Journal, 203*(417), 337-346.

Appleman, D. (2003). *Moving to VB. NET: Strategies, Concepts, and Code*: Apress.

Arrow, K. J. (1951). *Social choice and individual values*. New York, London: Wiley, Chapman & Hall.

Barnes, J. W., & Chambers, J. B. (1995). Solving the job shop scheduling problem with tabu search. *IIE Transactions, 27*(2), 257-263.

Barnes, J. W., Dimova, B., Dokov, S. P., & Solomon, A. (2003). The theory of elementary landscapes. *Applied Mathematics Letters, 16*(3), 337-343.

Battiti, R., & Tecchiolli, G. (1994). The Reactive Tabu Search. *ORSA Journal on Computing, 6*(2), 126.

Bazaraa, M. S., Sherali, H. D., & Shetty, C. M. (2006). *Nonlinear programming : theory and algorithms* (3rd ed.). Hoboken, N.J.: Wiley-Interscience.

Box, G. E. P., Jenkins, G. N., & Reinsel, G. C. (1994). *Time series analysis : forecasting and control* (3rd ed.). Englewood Cliffs, N.J.: Prentice Hall.

Braeysy, O. (2003). A Reactive Variable Neighborhood Search for the Vehicle-Routing Problem with Time Windows. *INFORMS Journal on Computing, 15*(4), 347-368.

Cain, W., Barnes, J. W., Ciarleglio, M., Lowry, T. S., Pierce, S. A., Sharp, J. M., et al. (2008, April 14-17 2008). *CADRE: A negotiation support system for sustainable water management.* Paper presented at the Water Down Under 2008, 31st Hydrology and Water Resources Symposium, 4th International Conference on Water Resources and Environment Research (ICWRER), Adelaide, Australia

Camm, J. D., Polasky, S., Solow, A., & Csuti, B. (1996). A note on optimal algorithms for reserve site selection. *Biological Conservation, 78*(3), 353-355.

Campbell, S., Backus, A., Wierman, D., Hemingway, M., Day, J., & Hollan, J. (2005). *Hays Trinity Groundwater Conseration District Groundwater Management Plan [retrieved from http://haysgroundwater.org/files/ManagementPlan-FinalAdoptedVersion.pdf]*.

Chan, K. M. A., Shaw, M. R., Cameron, D. R., Underwood, E. C., & Daily, G. C. (2006). Conservation Planning for Ecosystem Services. *PLoS Biology, 4*(11), 2138-2152.

Chowdhury, A. H. (2007). *GAM Run 07-18 [available at http://www.twdb.state.tx.us/gam/GAMruns/GR07-18.pdf]:* Texas Water Development Board.

Church, R. L., Stoms, D. M., & Davis, F. W. (1996). Reserve selection as a maximal covering location problem. *Biological Conservation, 76*(2), 105-112.

Ciarleglio, M., Barnes, J. W., & Sarkar, S. (2007). A Tabu Search Approach to the Spatially Coherent Conservation Area Network Design Problem *[editor invited paper, in review, Journal of Heuristics]*. Graduate Program in Operations Research and Industrial Engineering - University of Texas at Austin.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2001). *Introduction to algorithms* (2nd ed.). Cambridge, Mass.: MIT Press.

Daskin, M. S. (1995). *Network and discrete location : models, algorithms, and applications*. New York: Wiley.

Dell'Amico, M., & Trubian, M. (1993). Applying tabu search to the job-shop scheduling problem. *Annals of Operations Research, 41*(3), 231-252.

Di Gaspero, L., & Schaerf, A. (2003). EASYLOCAL++: an object-oriented framework for the flexible design of local-search algorithms. *Softw. Pract. Exp., 33*(8), 733-765.

Dimova, B., Barnes, J. W., & Popova, E. (2005). Arbitrary elementary landscapes & AR(1) processes. *Applied Mathematics Letters, 18*(3), 287-292.

Dowie, M. (2005). Conservation Refugees. *Orion, November-December*, 16-27.

Duong, D. X., & Dien, P. H. (2003). Tabu Search Approach to the Solution of the General Lectures Scheduling Problem. *Vietnam Journal of Mathematics, 31*(4), 437-448.

Dyer, J. S. (2005). MAUT – Multiattribute utility theory. In J. Figueira, S. Greco & M. Ehrgott (Eds.), *Multiple Criteria Decision Making: State of the Art Surveys* (pp. 265-295).

Eaton, D., Schwarz, S., & Sharp, J. M. (2007). *Groundwater Management in Texas [in preparation]*: Lyndon B. Johnson School of Public Affairs.

Fagan, W. F., Cantrell, R. S., & Cosner, C. (1999). How Habitat Edges Change Species Interactions. *The American Naturalist, 153*(2), 165-182.

Feo, T. A., & Resende, M. G. C. (1995). Greedy Randomized Adaptive Search Procedures. *Journal of Global Optimization, 6*(2), 109-1333.

Figueira, J., Greco, S., & Ehrgott, M. (Eds.). (2005). *Multiple criteria decision analysis: state of the art surveys*: Springer.

Fink, A., & Voß, S. (2002). HotFrame: A Heuristic Optimization Framework. In S. Voß & D. L. Woodruff (Eds.), *Optimization Software Class Libraries* (pp. 81–154). Boston: Kluwer.

Fishburn, P. C. (1991). Nontransitive preferences in decision theory. *Journal of Risk and Uncertainty, 4*(2), 113-134.

Freeman, E. (2004). *Head First Design Patterns*: O'Reilly.

Fuller, T., Sanchez-Cordero, V., Illoldi-Rangel, P., Linaje, M., & Sarkar, S. (2007). The cost of postponing biodiversity conservation in Mexico. *Biological Conservation, 134*(4), 593-600.

Garson, J., Aggarwal, A., & Sarkar, S. (2002). *Resnet Manual v1.2*. Austin, Texas: Biodiversity and Biocultural Conservation Laboratory at the University of Texas.

Glover, F., & Laguna, M. (1997). *Tabu search*. Boston: Kluwer Academic Publishers.

Goetz, B. (2005). Java theory and practice: Urban performance legends, revisited. from http://www-128.ibm.com/developerworks/java/library/j-jtp09275.html

Goetz, B. (2006). *Java concurrency in practice*. Upper Saddle River, NJ: Addison-Wesley.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*: Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA.

Grover, L. K. (1992). Local search and the local structure of NP-complete problems. *Operations Research Letters, 12*(4), 235-243.

Hansen, P., & Mladenovic, N. (2003). Variable neighborhood search. In F. Glover & G. A. Kochenberger (Eds.), *Handbook of Metaheuristics* (pp. 145–184): Springer.

Hansen, P., Mladenovic, N., & Urosevic, D. (2004). Variable neighborhood search for the maximum clique. *Discrete Applied Mathematics, 145*(1), 117-125.

Harder, R. (2001). OpenTS—Java tabu search. from http://www.coin-or.org/Ots/index.html

Harwig, J., Barnes, J. W., & Moore, J. (2006). An Adaptive Tabu Search Approach for 2–Dimensional Orthogonal Packing Problems. *Military Operations Research, 11*(2), 1-34.

Hinloopen, E., Nijkamp, P., & Rietveld, P. (1983). The regime method: a new multicriteria method. In P. Hansen (Ed.), *Essays and surveys on multiple criteria decision making: proceedings of the Fifth International Conference on Multiple Criteria Decision Making, Mons, Belgium, August 9-13, 1982* (pp. 146-155). Berlin ; New York: Springer-Verlag.

Justus, J., & Sarkar, S. (2002). The principle of complementarity in the design of reserve networks to conserve biodiversity: a preliminary history. *Journal of Biosciences, 27*(4), 421-435.

Kersten, G. E., Mikolajuk, Z., & Yeh, A. G. O. (2000). *Decision support systems for sustainable development : a resource book of methods and applications*. Boston: Kluwer Academic.

Kim, T. (1987). Intransitive Indifference and Revealed Preference Theory. *Econometrica, 55*(1), 163-167.

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by Simulated Annealing. *Science, 220*(4598), 671-680.

Kovacevic-Vujcic, V. V., Cangalovic, M. M., Asic, M. D., Ivanovic, L., & Drazic, M. (1999). TABU search methodology in global optimization. *Computers & Mathematics with Applications, 37*(4-5), 125-133.

Lambert, G., Barnes, J. W., & Veldhuizen, D. V. (2007). A Tabu Search Approach to the Strategic Airlift Problem. *Journal of Military Operations Research [in press]*.

Lindenmayer, D. B., Margules, C. R., & Botkin, D. B. (2000). Indicators of Biodiversity for Ecologically Sustainable Forest Management. *Conservation Biology, 14*(4), 941-950.

Loomes, G., & Sugden, R. (1982). Regret Theory: An Alternative Theory of Rational Choice Under Uncertainty. *The Economic Journal, 92*(368), 805-824.

Lootsma, F. A. (1993). Scale Sensitivity in the Multipiicative AHP and SMART. *Journal of Multi-Criteria Decision Analysis, 2*, 87-110.

Mace, R. E., Chowdhury, A. H., Anaya, R., & Way, S. C. T. (2001). *Report 353 Groundwater Availability of the Trinity Aquifer Hill Country Area, Texas: Numerical Simulations Through 2050*: Texas Water Development Board.

Mace, R. E., Petrossian, R., Bradley, R., & Mullican, W. F. (2006). *A Streetcar Named Desired Future Conditions*. Paper presented at the 7th annual: The Changing Face of Water Rights in Texas. from http://www.twdb.state.tx.us/gam/03-1_mace.pdf.

Mandler, M. (2004). Status quo maintenance reconsidered: changing or incomplete preferences? *The Economic Journal, 114*, 518-535.

Mandler, M. (2005). Incomplete preferences and rational intransitivity of choice. *Games & Economic Behavior, 50*(2), 255-277.

Margules, C., Higgs, A. J., & Rafe, R. W. (1982). Modern biogeographic theory: are there any lessons for nature reserve design. *Biological Conservation, 24*, 115-128.

Margules, C. R., Nicholls, A. O., & Pressey, R. L. (1988). Selecting networks of reserves to maximize biological diversity. *Biological Conservation, 43*(11), 63-76.

Margules, C. R., & Pressey, R. L. (2000). Systematic conservation planning. *Nature, 405*(6783), 243.

Margules, C. R., & Sarkar, S. (2007). *Systematic Conservation Planning*. Cambridge, UK: Cambridge University Press.

McKinzie, K., & Barnes, J. W. (2006). A Tabu Search Approach to the Strategic Mobility Mode Selection Problem. *Air Force Journal of Logistics, 30*(3), 51-62.

Memory Management in the HotSpot Virtual Machine. (2006). *Sun Developer Network: White Papers*, from http://java.sun.com/j2se/reference/whitepapers/memorymanagement_whitepaper.pdf

Mitra, D., Romeo, F., & Sangiovanni-Vincentelli, A. (1986). Convergence and Finite-Time Behavior of Simulated Annealing. *Advances in Applied Probability, 18*(3), 747-771.

Moffett, A., Dyer, J. S., & Sarkar, S. (2006a). Incorporating multiple criteria into the design of conservation area networks: a minireview with recommendations. *Diversity & Distributions, 12*(2), 125-137.

Moffett, A., Dyer, J. S., & Sarkar, S. (2006b). Integrating biodiversity representation with multiple criteria in North-Central Namibia using non-dominated alternatives and a modified analytic hierarchy process. *Biological Conservation, 129*(2), 181-191.

Nemhauser, G. L., & Wolsey, L. A. (1988). *Integer and combinatorial optimization*. New York: Wiley.

O'Hair, K. (2004). HPROF: A Heap/CPU Profiling Tool in J2SE 5.0. *Sun Developer Network: Technical Articles and Tips*, from http://java.sun.com/developer/technicalArticles/Programming/HPROF.html

Onal, H., & Briers, R. A. (2003). Selection of a minimum-boundary reserve network using integer programming. *Proceedings: Biological Sciences, 270*(1523), 1487-1491.

OptTek Systems, Inc. Retrieved July 31, 2007, from http://www.opttek.com/

Papadimitriou, C. H., & Steiglitz, K. (1998). *Combinatorial optimization : algorithms and complexity*. Mineola, N.Y.: Dover Publications.

Pawar, S., Koo, M. S., Kelley, C., Ahmed, M. F., & Sarkar, S. (2007). Conservation Assessment and Prioritization of Areas in Northeast India: Priorities for Amphibians and Reptiles. *Biological Conservation, 136*, 346-361.

Petrossian, R. (2007). Groundwater Management Areas: Making Decisions about the Future of Groundwater [*public presentation to Johnson City*]. Texas Water Development Board.

Phillips, S. J., Dudik, M., & Schapire, R. E. (2004). *A Maximum Entropy Approach to Species Distribution Modeling*. New York: ACM Press.

Pierce, S. A. (2006). *Groundwater decision support : an integrated assessment linking causal narratives, numerical models, and combinatorial search techniques to determine available yield for an aquifer system.* Unpublished Thesis (Ph. D.), University of Texas at Austin, 2006.

Pimm, S. L., Russell, G. J., Gittleman, J. L., & Brooks, T. M. (1995). The Future of Biodiversity. *Science, 269*(5222), 347-350.

Porter, J. M., Larsen, M. E., Barnes, J. W., & Howell, J. R. (2006). Metaheuristic Optimization of a Discrete Array of Radiant Heaters. *Journal of Heat Transfer, 128*(10), 1-30.

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

Pressey, R. L., Possingham, H. P., Logan, V. S., Day, J. R., Williams, P. H., & Pressey, B. (1999). Effects of data characteristics on the results of reserve selection algorithms. *Journal of Biogeography, 26*(1), 179-191.

Pugh, W. (2004). JSR-133: Java Memory Model and Thread Specification Revision. from http://jcp.org/en/jsr/detail?id=133

Rangaswamy, B., Jain, A. S., & Glover, F. (1998). Tabu search candidate list strategies in scheduling. In *Advances in computational and stochastic optimization, logic programming, and heuristic search: interfaces in computer science and operations research* (pp. 215-233): Kluwer Academic Publishers.

Raven, P. (1992). The Nature and Value of Biodiversity. In *Global Biodiversity Strategy: Guidelines for action to save, study and use Earth's biotic wealth sustainably and equitably*: World Resources Institute, IUCN-The World Conservation Union, United Nations Environment Programme (UNEP) in consultation with the Food and Agriculture Organization (FAO) and the United Nations Education, Scientific and Cultural Organization (UNESCO).

Regional Water Plans, 2001. *Texas Water Development Board.* Retrieved December 29, 2007, from http://www.twdb.state.tx.us/RWPG/planning_page.asp

Saaty, T. L. (1980). *The analytic hierarchy process : planning, setting priorities, resource allocation*. New York ; London: McGraw-Hill International Book Co.

Samuelson, P. A. (1938). A Note on the Pure Theory of Consumers' Behaviour. *Economica, New Series, 5*(17), 61-71.

Sarkar, S. (2003). Technical Note No 5: Definitions of Complementarity. Biodiversity and Biocultural Conservation Laboratory at The University of Texas.

Sarkar, S. (2005). *Biodiversity and environmental philosophy : an introduction*. Cambridge, UK ; New York: Cambridge University Press.

Sarkar, S., Justus, J., Fuller, T., Kelley, C., Garson, J., & Mayfield, M. (2005). Effectiveness of Environmental Surrogates for the Selection of Conservation Area Networks. *Conservation Biology, 19*(3), 815-825.

Sarkar, S., & Margules, C. R. (2002). Operationalizing biodiversity for conservation planning. *Journal of Biosciences, 27*(S2), 299-308.

Sarkar, S., Pappas, C., Garson, J., Aggarwal, A., & Cameron, S. (2004). Place prioritization for biodiversity conservation using probabilistic surrogate distribution data. *Diversity & Distributions, 10*(2), 125-133.

Sarkar, S., Pressey, R. L., Faith, D. P., Margueles, C. R., Fuller, T., Stoms, D. M., et al. (2006). Biodiversity Conservation Planning Tools: Present Status and Challenges for the Future. *Annual Review of Environment & Resources, 31*(1), 123-159.

Scanlon, B. R., Mace, R. E., Smith, B., Hovorka, S., Dutton, A., & Reedy, R. (2001). *Groundwater Availability of the Barton Springs Segment of the Edwards Aquifer, Texas: Numerical Simulations Through 2050.*: Texas Water Development Board.

Sen, A. K. (1971). Choice Functions and Revealed Preference. *The Review of Economic Studies, 38*(3), 307-317.

Sierra, K., & Bates, B. (2003). *Head First Java*: O'Reilly.

Smith, B. A., & Hunt, B. B. (2004). *Evaluation of sustainable yield of the Barton Springs segment of the Edwards aquifer, Hays and Travis counties, central Texas*: Barton Springs/Edwards Aquifer Conservation District.

Solomon, A., Barnes, J. W., Dokov, S. P., & Acevedo, R. (2003). Weakly symmetric graphs, elementary landscapes, and the TSP. *Applied Mathematics Letters, 16*(3), 401.

Strager, J. M., & Yuill, C. B. (2002). *The West Virginia GAP analysis project final report*: U. S. Geological Survey, Morgantown, West Virginia. Available from http://www.nrac.wvu.edu/projects/gap/pub.htm (accessed December 2007).

Tversky, A., & Kahneman, D. (1981). The framing of decisions and the psychology of choice. *Science, 211*(4481), 453.

UT GMA 9 Groundwater Management Class. (June 27, 2007). Training Session for the Interactive Trinity Aquifer Model. Austin, TX.

Whittaker, R. H. (1975). *Communities and ecosystems* (2d ed.). New York: Macmillan.

Wilde, D. J., & Beightler, C. S. (1967). *Foundations of optimization*. Englewood Cliffs, N.J.,: Prentice-Hall.

Wolsey, L. A. (1998). *Integer programming*. New York: J. Wiley.

Vita

Michael Ian Ciarleglio was born September 14, 1979, the son of Judy and Peter Ciarleglio. He graduated from Brentwood High School in Brentwood, Tennessee, in 1997. He attended Vanderbilt University from 1997 to 2001, receiving a Bachelor of Engineering degree in Mechanical Engineering (and Math).

Before returning to graduate school, he worked for 18 months as a systems engineer for a missile defense project, specializing in infrared sensors, data acquisition, and field deployment. Michael entered the University of Texas at Austin in Fall 2003, supported by a Grant for the Vertical Integration of Research and Education (VIGRE), funded by the National Science Foundation. He has been a teaching assistant for differential equations, and an assistant instructor for pre-calculus.

Permanent Address: 3180 Boxley Valley Road, Franklin, TN 37064

This dissertation was typed by the author.