Use-Driven Concept Formation

by

Jennifer M. Roberts

S.M., Electrical Engineering and Computer Science,
Massachusetts Institute of Technology (2006)
B.S., Electrical Engineering, and B.S., Computer Science,
University of Maryland, College Park (2004)

Submitted to the Department of Electrical Engineering
and Computer Science
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in Computer Science and Engineering
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
September 2010

Author ....
Department of Electrical Engineering
and Computer Science
September 3, 2010

Certified by ....
Patrick H. Winston
Ford Professor of Artificial Intelligence and Computer Science
Thesis Supervisor

Certified by ....
Randall Davis
Professor
Thesis Supervisor

Accepted by ....
Terry P. Orlando
Chairman, Department Committee on Graduate Theses

Use-Driven Concept Formation

by

Jennifer M. Roberts

Submitted to the Department of Electrical Engineering
and Computer Science
on September 3, 2010, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Computer Science and Engineering

Abstract

When faced with a complex task, humans often identify domain-specific concepts that make the task more tractable. In this thesis, I investigate the formation of domain-specific concepts of this sort. I propose a set of principles for formulating domain-specific concepts, including a new inductive bias that I call the equivalence class principle. I then use the domain of two-player, perfect-information games to test and refine those principles. I show how the principles can be applied in a semi-automated fashion to identify strategically-important visual concepts, discover high-level structure in a game's state space, create human-interpretable descriptions of tactics, and uncover both offensive and defensive strategies within five deterministic, perfect-information games that have up to forty-two million states apiece.

I introduce a visualization technique for networks that discovers a new strategy for exploiting an opponent's mistakes in lose tic-tac-toe; discovers the optimal defensive strategies in five and six men's morris; discovers the optimal offensive strategies in pong hau k'i, tic-tac-toe, and lose tic-tac-toe; simplifies state spaces by up to two orders of magnitude; and creates a hierarchical depiction of a game's state space that allows the user to explore the space at multiple levels of granularity. I also introduce the equivalence class principle, an inductive bias that identifies concepts by building connections between two representations in the same domain. I demonstrate how this principle can be used to rediscover visual concepts that would help a person learn to play a game, propose a procedure for using such concepts to create succinct, human-interpretable descriptions of offensive and defensive tactics, and show that these tactics can compress important information in the five men's morris state space by two orders of magnitude.

Thesis Supervisor: Patrick H. Winston
Title: Ford Professor of Artificial Intelligence and Computer Science

Thesis Supervisor: Randall Davis
Title: Professor

Acknowledgments

First, I would like to thank my two advisors Patrick Winston and Randy Davis. Patrick gave me the freedom to pursue my interests and encouraged me to explore my ideas and see where they led. He also taught me how to showcase my work at all stages of development. Patrick has an aptitude for not only identifying the parts of a narrative that I had been spending the most time refining, but also for proposing a cohesive framework for outlining my work, providing the appropriate level of detail, and highlighting my contributions.

Randy is exceptional at asking probing questions that helped me to refine my research approach, identify the theoretical underpinning of my work, and challenge my background assumptions. He has a rare ability to critically dissect a piece of work in a way that is both encouraging and insightful. My discussions with Randy truly helped me to develop scientifically.

I would also like to thank my committee members, Susan Epstein and Sanjoy Mahajan. Susan's research served as a foundation for much of my work. She has helped to refine my work on games, coined the term "use-driven concepts," and offered numerous pointers on how to write about scientific work in a clear, strong, cohesive manner. Sanjoy provided a unique perspective on learning and encouraged me to study sets of representations in a more principled manner.

I would like to thank the Fannie and John Hertz Foundation, the National Science Foundation, the Google Anita Borg Scholarship program, and the Marilyn Pierce EECS Departmental Fellowship committee for their generous financial support. These organizations allowed me to pursue research on topics that I found both interesting and rewarding. I would especially like to thank Tom Weaver and everyone else at the Fannie and John Hertz Foundation for the excellent mentorship programs that they have initiated since I entered the program. Their professional development and networking opportunities gave me a much broader perspective on the many ways in which we can contribute to society as scientists. Members of the Anita Borg Scholarship program also introduced me to a remarkable, empowering community of

female researchers.

Mark Finlayson has been a good officemate and lent me the hardware for my experiments on five and six men's morris. Jake Beal has been an excellent collaborator, and I have enjoyed our thought-provoking discussions on computational models, cognitive science, and the future of cognitively-inspired artificial intelligence. My friends from the Graduate Student's Association (Paul, Nicole, Stephen, and others) helped me to improve the student experience within the EECS department and made my time in graduate school much more enjoyable. Nicole, Vanessa, Jonathan, Mireille, and David made MIT feel more like home, and I love that I can always count on Laurie to provide a positive outlook, regardless of the situation.

Most of all, I would like to thank my husband, Chen. Chen has been remarkably supportive through this whole process, acting as my chief confident, research collaborator, editor, cheerleader, and best friend. He has helped me through every step of the process. I would also like to thank my mom, dad, and sister, Becca, who have been amazingly supportive and encouraging and expressed supreme confidence in my ability to accomplish large amounts of work in short periods of time.

Contents

List of Figures

List of Tables

Chapter 1

Principles of Concept Formation

Theories of the known ... may be equivalent in all their predictions and are hence scientifically indistinguishable. However, they are not psychologically identical when trying to move from that base into the unknown.

– Richard Feynman, Nobel Lecture, 1965

Domain-specific concepts often make complex tasks more tractable. For example, light can be viewed as a wave or as a particle, and each of these two concepts make it easier for people to solve a class of physics problems. While viewing light as a wave is an oversimplification that does not accurately describe the true nature of light, the concept of a wave is still useful within that domain because it often enables someone to predict how light will interact with its surroundings.

In this thesis, I explore techniques for identifying *use-driven concepts*, which are human-interpretable, domain-specific concepts that make a complex task more tractable. Ultimately, I would like to create computational tools that start with only domain-general knowledge, use that domain-general knowledge to study a novel domain, and identify concepts that would enable a person to perform tasks within the domain of interest. Toward that end, I explore two types of concept formation, one that involves identifying conceptual patterns in networks and one that involves building connections between two representations in order to identify use-driven concepts. The representations that I am interested in provide two perspectives on the same domain in the way that a Cartesian plot and an equation provide two perspectives within mathematics. In this chapter, I propose a principles for forming use-driven concepts, and throughout the rest of the thesis, I use the domain of two-player, perfect-information games to test and refine those principles.

The primary contribution of this work is a semi-automated process for identifying use-driven concepts by building connections between two representations that describe the same domain. Sets of representations within the same domain not only provide different perspectives, but also provide a mechanism for identifying domain-specific concepts. Assuming that you have two representations that give different perspectives on the same set of entities, the computer can classify the entities using features from the first representation. When the entities in the second representation are grouped based on their classification in the first representation, members of the induced classes often exhibit a pattern that humans can detect. This process highlights features in the second representation that correlate with the classification boundaries from the first representation and provides an inductive bias called the *equivalence class principle* for identifying noteworthy features in the second representation. Patterns exposed by the classification in the second representation form a set of use-driven concepts.

The equivalence class principle relies on having a pair of representations that describe the same set of entities. Its ability to expose connections between the representations depends on the extent to which they provide different perspectives on the same information. When information that is explicitly encoded in the first representation is at least implicitly encoded in the second, or when both representations implicitly encode the same information in different ways, the equivalence class principle can highlight connections between the representations.

Often, important features in the second representation form patterns that humans find perceptually salient. The degree to which such patterns are perceptually salient may depend on how explicitly the second representation encodes the features of interest. While implicitly-encoded features may still be detectable, recognizing them

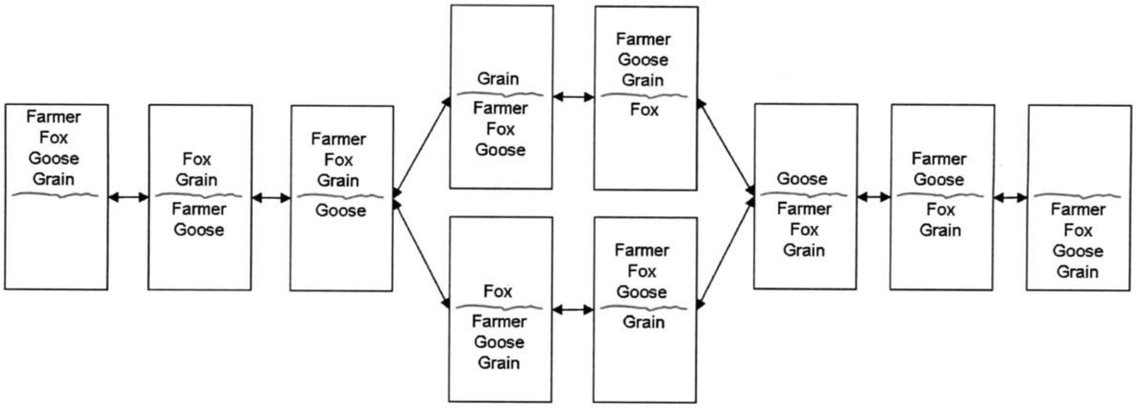


Figure 1-1: A network representation of the famous farmer, goose, and fox problem. In this problem, the farmer must take his fox, goose, and grain from one side of the river to the other in a boat that only holds himself and one other item. While crossing the river, he cannot leave the fox and the goose alone or the goose and grain alone because, given the opportunity, the fox will eat the goose and the goose will eat the grain. Although most people think the problem has one solution, this network diagram reveals two. (Adapted from Winston, 1993.)

will generally require more processing.

In the next section, I provide an example of how the equivalence class principle can reveal mathematical concepts, while in the remainder of the thesis, I demonstrate how it can reveal game-related concepts. More specifically, I use the principle to identify use-driven visual concepts that provide a mechanism for succinctly describing winning sequences of moves. Developing a fully-automated algorithm for applying the equivalence class principle would lead to a new type of machine learning algorithm.

The second major contribution of this work is a general-purpose approach for simplifying networks. Networks such as the one pictured in Figure 1-1 can provide deep insight into the nature of complex problems. However, even when they have small numbers of nodes, networks such as the one pictured in Figure 1-2a become difficult for humans to interpret. A good layout may reveal symmetries, as in Figure 1-2b, but the high-level structure necessary for understanding the topology and the nature of the problem still remains hidden.

My network simplification algorithms identify patterns and high-level structure in networks, resulting in simplifications such as the one in Figure 1-2c. The algorithms proceed by identifying subnetworks that share the same topology and or share

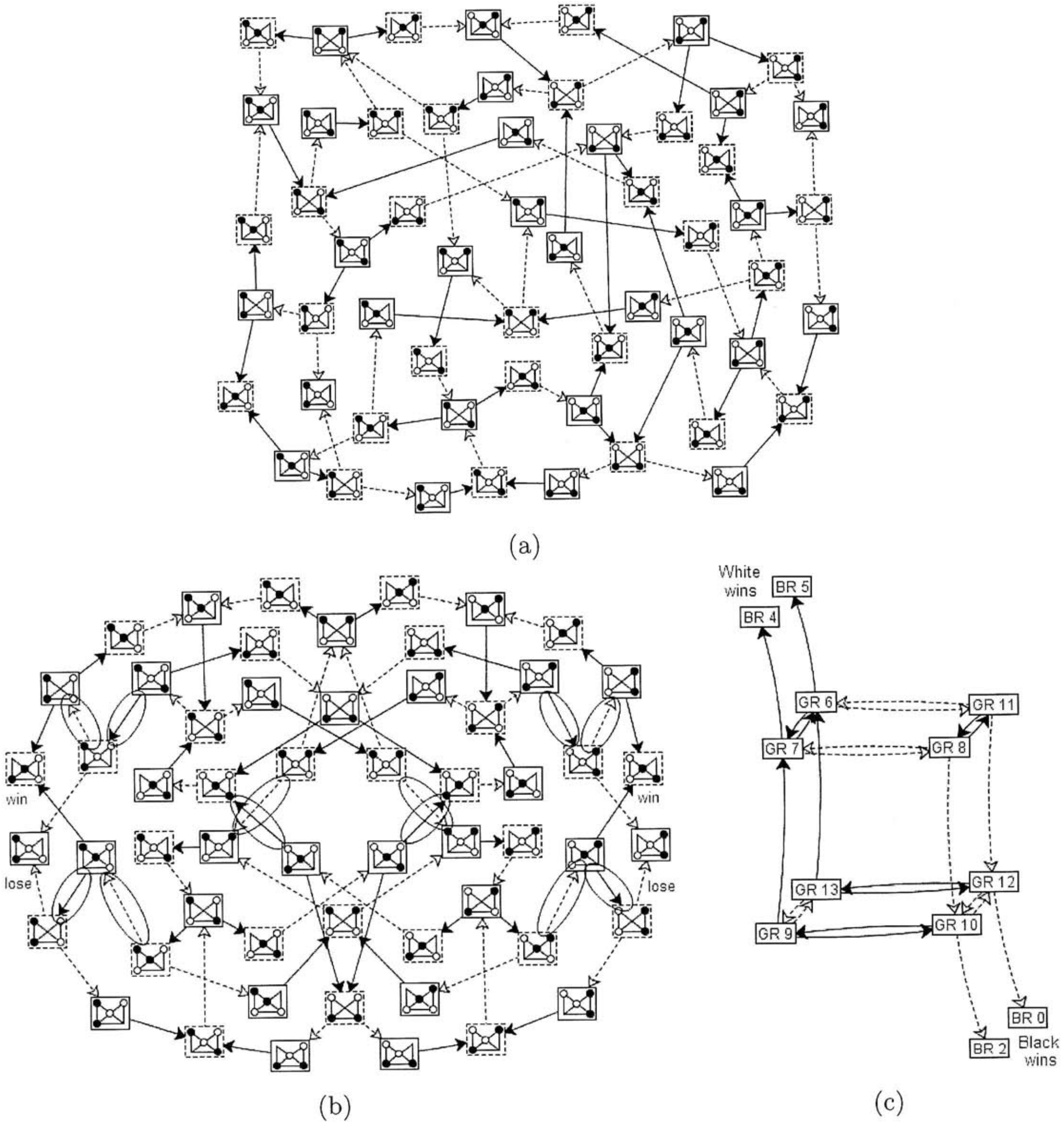


Figure 1-2: (a) State space of pong hau k'i, a game with 56 states (Zaslavsky, 1982). See Chapter 4 for a full description of the game. Each game state consists of a board configuration paired with an indication of which player moves next. Dotted outlines and arrows indicate when white moves next, and solid outlines and arrows indicate when black moves next. (b) A symmetric layout of the same state space. The four circled moves near the center of the diagram are important offensive moves, while the rest of the circled moves are important defensive moves. (c) A simplification of the state space diagram for pong hau k'i. Non-leaf nodes represent loops in the graph. This diagram highlights the high-level structure of the game, exposing the key offensive moves. The key offensive moves for black move from cycle G9 to cycle G10 or from cycle G13 to cycle G12. Once in cycle G10 or G12, black can win as soon as white mistakenly moves to BR0 or BR2. The key offensive moves for white follow a similar pattern.

patterns involving both topology and node attributes, where a node attribute might indicate whether a state in a game is a win or a loss. When a pattern is identified, it is replaced by a single node, converting the pattern into a labeled entity that can then become part of a more complex pattern. This grouping mechanism hierarchically organizes the network in a way that emphasizes high level structure, exposes portions of the network containing nodes that have not been grouped into a pattern, reveals high-level patterns that would not otherwise be visible, and provides a mechanism for visualizing a network at multiple levels of granularity. Selectively grouping some patterns and not others can highlight different types of structure within the graph. When the grouping mechanisms expose high-level patterns that have meaning within the domain of interest, these patterns become new domain-specific concepts.

1.1 Principles for Forming Use-Driven Concepts

As noted, use-driven concepts are human-interpretable, domain-specific concepts that make complex tasks more tractable. In this thesis, I explore two modes of use-driven concept formation, one that forms concepts within networks, and one that forms concepts within a broader set of representations. In this section, I describe the principles behind each type of concept formation and describe how broadly the principles may be applied. In the next section, I describe how to use these principles to analyze games.

1.1.1 Principles for Identifying Use-Driven Concepts in Networks

Concepts in networks are represented as subgraphs that share node and edge attributes. For example, in a game state graph, a subtree with leaves that are all draw states represents a set of paths that lead to a draw. A draw tree is an example of an *annotated topological pattern* (*ATP*), a set of constraints on a subgraph's topology and on its node and edge attributes that define a pattern in a network. Complex concepts can be identified by hierarchically grouping instances of ATPs to incrementally build upon known information.

To facilitate pattern recognition and formation, each instance of an ATP within a network is collapsed into a labeled node, called a *collapsed subgraph.*[1] Collapsed subgraphs can then become part of higher-order ATPs. When collapsed subgraphs with different topologies receive the same label, this collapsing procedure can abstract away unnecessary information and, with proper visualization tools, make it easier for people to identify high-level patterns. The iterative collapsing procedure also organizes concepts hierarchically in a manner that facilitates visualization at multiple levels of granularity.

Within this work, ATPs generally build upon previously-acquired knowledge by incorporating both collapsed subgraphs and regular nodes. This ensures that complex ATPs build on simpler ones in a straightforward manner, thus using simple concepts to create complex concepts. Pattern formation could also proceed in a less compositional fashion, but in the majority of this thesis, I focus on a compositional mode of pattern formation.

1.1.2 The Equivalence Class Principle for Forming Concepts Using Sets of Representations

> *...[D]ifferent views suggest different kinds of modifications which might be made and hence are not equivalent in the hypotheses one generates from them in one's attempt to understand what is not yet understood.*
>
> – Richard Feynman, Nobel Lecture, 1965

While the last section described a mode of concept formation intended for network representations, this section describes a principle that can be used to form

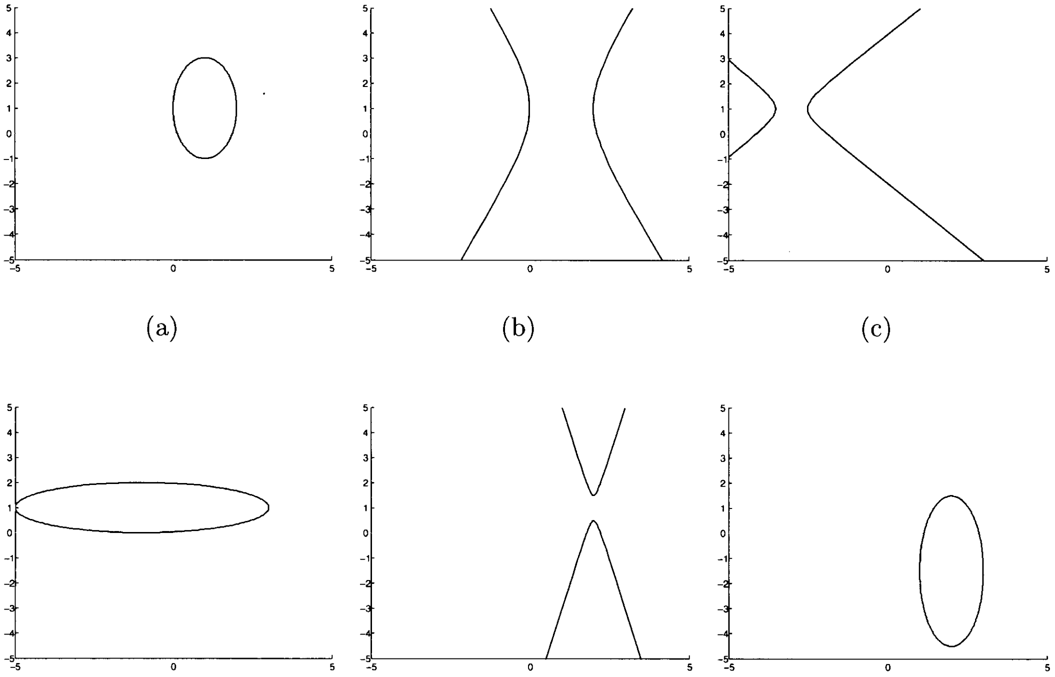
[1] The idea for collapsing subgraphs into nodes comes from a case study in which a human subject collapsed pairs of states within a state space (Epstein & Keibel, 2002; Epstein, 2005).

concepts within any set of representations that describe the same group of entities in different ways. Multiple representations within the same domain not only provide different viewpoints of the same entities, but also provide a mechanism for identifying important domain-specific concepts. In this section, I illustrate this principle using examples from mathematics, and throughout the rest of the thesis, I show how it can be applied to the domain of games.

In mathematics, equations and Cartesian plots provide two different perspectives on lines and curves. Given a large number of plots such as the ones shown in Figure 1-3, one way to classify the plots is based on shape. While shape provides a perceptually-salient cue for separating the ellipses from the hyperbolas, the corresponding equations in Figure 1-4 seem relatively similar to one another. Equations of this sort could be classified based on a number of features, such as the number of minus signs, the magnitude of the x^2 term coefficients, or the set of coefficients in the equation, but no feature necessarily seems more compelling than any other.

Within a multi-representational setting, the *equivalence class principle*, which was introduced in the last section, provides a mechanism for deciding which features are important. The principle states that features in one representation are important when they correlate with classification boundaries in another representation. Applying this principle to the equations in Figure 1-4, equations (a), (d), and (f) fall into one class because they all describe ellipses and equations (b), (c), and (e) fall into another class because they all describe hyperbolas. When a large number of equations are grouped in this way, the sign agreement between the x^2 and y^2 terms becomes one of the few features that correlates with the classification boundary. This exposes the fact that an equation of this form corresponds to an ellipse when the signs of the squared terms are the same, and it corresponds to a hyperbola when the signs of the squared terms are different. The fact that the sign agreement of the squared terms can predict whether a plot is an ellipse or a hyperbola makes this feature an important concept within this domain.

Using the equivalence class principle to identify important concepts involves the



(a) (b) (c)

(d) (e) (f)

Figure 1-3: Visually, these plots form two classes, ellipses and hyperbolas.

$$x^2 - 2x + 1 + \frac{y^2}{4} - \frac{y}{2} + \frac{1}{4} = 1 \tag{a}$$

$$x^2 - 2x + 1 - \frac{y^2}{4} + \frac{y}{2} - \frac{1}{4} = 1 \tag{b}$$

$$4x^2 + 24x + 36 - 4y^2 + 8y - 4 = 1 \tag{c}$$

$$\frac{x^2}{16} + \frac{x}{8} + \frac{1}{16} + y^2 - 2y + 1 = 1 \tag{d}$$

$$-64x^2 + 256x - 256 + 4y^2 - 8y + 4 = 1 \tag{e}$$

$$x^2 - 4x + 4 + \frac{y^2}{9} + \frac{y}{3} + \frac{1}{4} = 1 \tag{f}$$

Figure 1-4: Equations for the plots shown in Figure 1-3. Considered in isolation, these equations provide no obvious criteria for classification. The equations could be classified based on whether or not they contain fractions, the number of coefficients with positive terms, or magnitude of the largest coefficient. Considered in conjunction with the Cartesian plots, however, correspondences between the equations and plots reveal that equations with x^2 and y^2 terms that share the same sign correspond to ellipses, and equations where these two signs differ correspond to hyperbola.

Equivalence Class Concept Formation

1. **Classification:** Classify entities in representation one.
2. **Exemplar-Based Concept Formation:** Create exemplar-based concept descriptions in representation 2 by using the representation one classification to group entities in representation two.
3. **Concept Redescription:** Identify patterns within the sets of exemplars and use those patterns to concisely redescribe the concepts.

Figure 1-5: Equivalence class concept formation involves these three steps.

three steps delineated in Figure 1-5.[2] First, entities are classified in one representation, the way that the ellipses and hyperbolas were classified in the Cartesian plane. Then, entities within a second representation are grouped based on their classification in the first representation to form exemplar-based concept descriptions. This corresponds to grouping the equations based on the classification of their plot, and using a large set of ellipse equations to form an exemplar-based description of the concept "equation of an ellipse." The final step involves analyzing the exemplar-based description of the concept to extract important features and succinctly redescribe the exemplars. This corresponds to taking a large number of ellipse equations and developing an redescribed definition such as "an equation with x^2 and y^2 terms that share the same sign."

1.2 Using the Modes of Use-Driven Concept Formation to Analyze Games

To apply the concept formation principles to games, I use two game-related representations: a state space representation and a visual representation of the board. In the state space representation, a state consists of a *board configuration* and an indication

[2] The terms used to describe the different types of concepts are adapted from exemplar theory (Murphy, 2002) and the theory of representational redescription (Karmiloff-Smith, 1995).

of which player plays next. The board configuration stores the positions of all of the pieces on the board, but these positions need not be stored in a way that facilitates visual pattern recognition. The visual representation consists of a picture of the board with the pieces in a particular configuration, often paired with an indication of which player plays next. A picture of the board that corresponds to a particular state is called a *board diagram.*

When applied to a game's state space, the technique for analyzing networks from Section 1.1.1 can be applied interactively or non-interactively to serve different purposes. Recall that an *annotated topological pattern* is a subgraph in the state space that exhibits a topological pattern and contains nodes and edges with labels that meet certain criteria. A *pattern grouping algorithm* searches a state space, identifies instances of either a particular annotated topological pattern or a set of related annotated topological patterns, and collapses each instance into a labeled node. Collapsed nodes of this type are called *collapsed subgraphs* or *collapsed subtrees*, depending on their topology. The label that a collapsed subgraph receives depends on the annotated topological pattern it exhibits and depends on the labels attached to nodes in its subgraph. Collapsed subgraphs with the same label form a class of collapsed subgraphs. The pattern grouping algorithms can identify instances of game-general concepts such as forced move sequences and paths that lead to a draw.

The pattern grouping algorithms can support an interactive mode of computer-aided discovery, whereby the computer applies a set of pattern grouping algorithms, provides the user with a hierarchical visualization of the collapsed state space, and allows the user to define new annotated topological patterns. The new annotated topological patterns can be seen as complex, task-specific concepts. Using a new pattern grouping algorithm to collapse the user-defined annotated topological pattern will reveal another level of structure within the state space. The user can then proceed to iteratively develop additional annotated topological patterns and pattern grouping algorithms. This mode of interactive computer-aided discovery is explored briefly in Chapter 3 and more thoroughly in Appendix A.

The pattern grouping algorithms also support a non-interactive mode of computer-aided discovery, which I use throughout the majority of the thesis. In this mode, the algorithms collapse instances of annotated topological patterns that correspond to a set of game-general concepts, thus identifying which game-general concepts can be used to analyze the game of interest.

When used non-interactively, the pattern grouping algorithms serve two purposes. First, they hierarchically organize a game's state space in a way that exposes high-level structure and *strategies*. For example, in Chapter 5, I show how the pattern grouping algorithms rediscover X's best strategy, i.e., X's best long-term plan, in tic-tac-toe. Using this strategy, X opens by placing a mark in a corner square. In Chapter 3, I introduce a set of pattern grouping algorithms that can theoretically identify the optimal defensive strategy within any two-player perfect-information draw game by highlighting all states in which a player can select moves that either continue to force a draw or potentially lead to a win for the opponent.[3] In Chapters 4 through 6, I describe how these algorithms have identified the optimal defensive strategies in pong hau k'i (56 states), tic-tac-toe (765 states), lose tic-tac-toe (765 states), five men's morris (18,000 states), and six men's morris (42,000 states).[4]

In addition to exposing high-level structure and strategies in games, the pattern grouping algorithms perform the first step, i.e., the classification step, of the equivalence class concept formation process. This first step involves identifying a class of states within a game's state space. The second step involves using the equivalence class principle to build connections between the state space representation and the visual representation in order to create an exemplar-based description of a strategically-significant *visual concept*. The third step involves extracting a succinct description of the visual concept.

For example, the first step might extract all states within the state space that are one move away from a win, the second step would collect board diagrams that fit this description, and the third step would entail extracting and succinctly describing the visual pattern shared by the set of board diagrams. This visual pattern becomes a

[3] If a player can force a win or a loss from the initial state, this set of algorithms would detect the paths from the initial state to the win or loss states.

[4] For a full description of these games, please refer to Chapters 4 through 6.

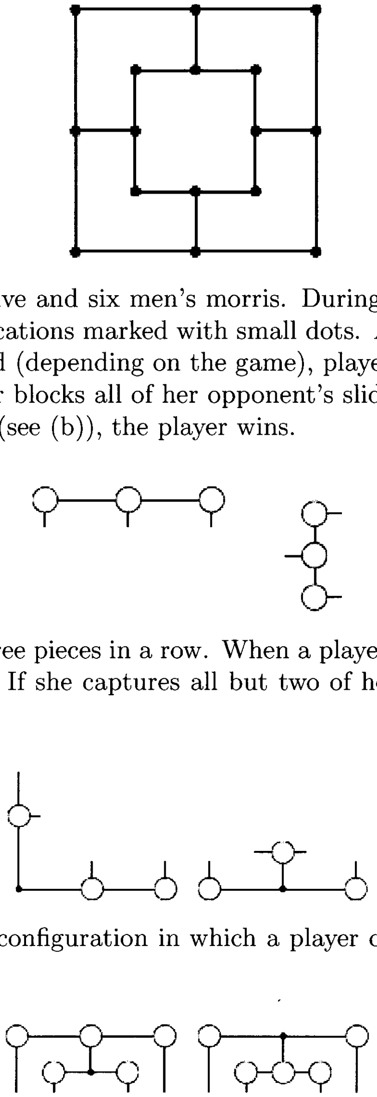
use-driven visual concept because it can be used to predict when a player is one move away from a win and used to concisely describe a winning *tactic*, i.e., a short-term plan that leads to a win. In this work, the computer performs the first two steps and I perform the last step, so the current implementation is an example of computer-aided concept discovery.

By using the pattern grouping algorithms combined with the equivalence class principle, general knowledge about two-player, perfect-information games can be used to identify strategies, tactics, and visual concepts. Again, visual concepts, such as those shown in Figure 1-6, are strategically-important visual patterns that have been identified via the equivalence class principle and can be used to concisely describe tactics. For example, the visual concept in Figure 1-6d can describe a tactic in which white slides its center piece up and down to capture a black piece on each turn and thus win the game. In Chapter 6, I describe how using visual concepts to concisely describe tactics in a human-interpretable manner can compress the important information in a game's state space by two orders of magnitude.

1.3 Significance

In this thesis, I test the claim that multiple representations in the same domain provide a mechanism for identifying important domain-specific concepts by creating connections between representations. The equivalence class principle is the inductive bias that makes this process possible. Theoretically, the equivalence class principle should enable connections to be formed when two representations encode the same information in different ways, and should fail to detect patterns when the two representations describe disjoint pieces of information. By using the equivalence class principle to search for patterns in five deterministic, two-player, perfect-information games, I provide evidence to support these claims. In the final chapter, I describe how the equivalence class principle can be used to develop a new type of machine learning algorithm.

Using the equivalence class principle to analyze games is analogous to searching



(a) The board used to play five and six men's morris. During the first phase of the games, players place pieces on the locations marked with small dots. After each player places either five or six pieces on the board (depending on the game), players take turns sliding pieces to adjacent dots. When a player blocks all of her opponent's sliding moves or captures all but two of her opponent's pieces (see (b)), the player wins.

(b) Mill: A mill consists of three pieces in a row. When a player forms a mill, she can capture one of her opponent's pieces. If she captures all but two of her opponent's pieces, she wins the game.

(c) One-from-a-mill: A piece configuration in which a player can slide a piece into a mill on her next turn.

(d) Trapezoidal double mill: A piece configuration in which a player can slide the center piece up and down to capture one of her opponent's pieces on each turn.

Figure 1-6: (a) The board used to play five and six men's morris. For a complete description of the rules, please see Chapter 6. (b-d) Examples of visual concepts rediscovered by applying the equivalence class principle to five men's morris.

for patterns in game databases. Gasser (1996) used brute-force methods to solve nine men's morris, a variation on two of the games that I analyze in this thesis, but his work does not provide human-interpretable descriptions of optimal tactics. In this thesis, I exploit a method for developing human-interpretable descriptions of tactics, building on the action sequence descriptions developed by Lock and Epstein (2004). While the preliminary efforts that I describe in this thesis do not determine how much of the five men's morris state space can be described in a human-interpretable fashion, they provide evidence that human-interpretable descriptions comprised of use-driven concepts can compress portions of the space by several orders of magnitude. Although the procedure for performing this compression is not fully automated, the last chapter describes a procedure for gradually automating the process.

1.4 Using Games to Test and Refine the Modes of Use-Driven Concept Formation

In the rest of the thesis, I apply the principles outlined in this chapter to five games of increasing complexity and show how the principles can be used to identify structure in a game's state space and to discover visual concepts, strategies, and tactics. In Chapter 3, I briefly describe how the pattern grouping algorithms can support an interactive mode of computer-aided discovery that enables the identification of new state space concepts, and in Appendix A, I explain how I used this interactive mode to develop pattern grouping algorithms that identify game-general concepts. In the remainder of the thesis, I describe work that applies the pattern grouping algorithms non-interactively to hierarchically organize a game's state space and to create exemplar-based descriptions of visual concepts.

In Chapter 2, I explain how my work relates to other research in machine learning, computer-aided discovery, concept formation, and game analysis. In Chapter 3, I describe a set of pattern grouping algorithms that can be applied to any deterministic, perfect-information, two-player game. I use examples from tic-tac-toe to explain how these algorithms simplify a state space and create exemplar-based concepts. In Chapter 4, I describe additional pattern grouping algorithms that simplify cyclic state spaces with a high number of forced moves and demonstrate how these algorithms discover optimal offensive and defensive strategies within a simple 56-state game called pong hau k'i.

In the rest of the chapters, I summarize the results obtained when the algorithms described in Chapter 3 are applied to tic-tac-toe, lose tic-tac-toe, five men's morris, and six men's morris. Using these algorithms, I show why playing tic-tac-toe requires less computational effort than playing lose tic-tac-toe, discover a difficult-to-find offensive strategy for lose tic-tac-toe, outline optimal defensive strategies for five and six men's morris, reveal visual concepts in tic-tac-toe and five men's morris, and show how human-interpretable tactics expressed in terms of visual concepts can compress state space information by several orders of magnitude. I outline the results for the tic-tac-toe games and the morris games in Chapters 5 and 6, respectively. In Chapter 7, I conclude with a discussion of future work and contributions.

In Appendix A, I describe how I developed some of the pattern grouping algorithms from Chapter 3. In Appendix B, I describe some of my previous work on sets of representations, work that inspired me to focus on the equivalence class principle. In this appendix, I propose three computational properties that make sets of interconnected representations powerful.

Chapter 2

Related Work

In this chapter, I describe how the work in this thesis relates to work in machine learning, computer-aided discovery, concept formation, and game analysis.

2.1 Inductive Biases

A well known result in machine learning called the *No Free Lunch Theorem* states that no learning algorithm can succeed in all contexts. Even when an algorithm performs well in some contexts, there will always be another context where the algorithm performs no better than chance (Wolpert, 1996; Wolpert & Macready, 1997). Mitchell introduced the term "inductive bias" to describe the assumptions that cause a learning algorithm to favor one interpretation of a data set over another (T. M. Mitchell, 1980).

Occam's razor is one of the most famous inductive biases, stating that the simplest explanation for a phenomenon should always be favored. Bayesian model comparison uses Occam's razor as its inductive bias (MacKay, 2003), while other machine learning algorithms favor other types of biases. For example, nearest neighbors clustering algorithms build on the assumption that data points that are close to one another in a feature space belong to the same group, support vector machines build on the assumption that the maximum margin classifier provides the best separation between data clusters, and inductive logic programming builds on the assumption that descriptions of a group of entities can be expressed in first order logic and such descriptions can be built using inverse resolution. Other machine learning paradigms, such as neural networks, have inductive biases that are harder to clearly define. Cognitive scientists have discovered a number of inductive biases used by humans, such as the shape bias for word learning (Landau, Smith, & Jones, 1988) and the assumption that objects tend to follow smooth trajectories for naive physics (Spelke, 1990) (see Kemp, 2007 for a review).

This thesis explores an inductive bias for learning in a multi-representational context. The only other paradigm for multi-representational learning that I am aware of is cross-modal clustering (Coen, 2005). Cross-modal clustering detects correlations between two feature spaces that describe the same events. For example, a cross-modal clustering model of how songbirds learn to sing creates connections between a motor feature space and an auditory feature space (Coen, 2007). Cross-modal clustering has also been used to learn to recognize vowel sounds using a visual feature space that captures lip position and an auditory feature space that stores information about formants (Coen, 2006). This approach uses two feature spaces to learn how to distinguish between and recognize entities such as vowels or sequences of notes. In contrast, my approach uses entities that have already been classified in one representation to expose patterns in another representation.

While inductive biases are usually encoded into a machine learning algorithm, the one tested in this thesis has not yet been encoded in that way. In this thesis, I simply test whether the inductive bias known as the equivalence class principle can be used to expose concepts. Ultimately, the equivalence class principle could be used to build a machine learning algorithm that works like a clustering algorithm in reverse. Instead of starting with data in a feature space, identifying clusters, and learning decision boundaries that separate the clusters, the reverse clustering algorithm would start with a set of decision boundaries and learn which features correlate with that decision boundary. More specifically, a classification in one representation would create decision boundaries in a second representation. The learning algorithm would then identify features in the second representation that could predict the position of the decision boundary within the first representation.

If the algorithm cannot find features that correlate with the decision boundary, it either means that (1) the decision boundary does not create a meaningful separation of the data in the second representation or (2) the algorithm is considering the wrong set of features. In general, distinguishing between these two cases would be non-trivial. The first alternative would mean that the equivalence class principle does not apply in the situation of interest. Based on the *No Free Lunch Theorem*, the principle should fail to yield meaningful results at least some of the time. The second alternative would mean that the learning algorithm's hypothesis space does not contain the correct hypotheses.

2.2 Computer-Aided Discovery

While computer-aided discovery has probably occurred in one form or another since the beginning of artificial intelligence, in this section, I describe examples of computer-aided discovery in which computers help humans develop concepts, models, or theories (see Langley, 1998 for a review). My focus will be on describing the types of interactions between humans and computers that enable the discovery process. This review is not meant to be exhaustive. Instead, I present several examples of computer-aided discovery to explore the types of human-computer interactions that occur within these systems.

Early efforts to perform discovery using computers attempted to recreate historical scientific results in mathematics (Lenat, 1977), physics (Langley, 1981), chemistry (Zytkow & Simon, 1986), and biology (Kulkarni & Simon, 1990). While researchers at that time hoped to create systems that worked completely without human intervention, in reality humans aided the discovery process. For example, Lenat's program AM (Lenat, 1977) derived concepts from discrete mathematics. In addition to having heuristic algorithms that automatically directed the discovery process, Lenat also periodically influenced which lines of inquiry the computer pursued by indicating which concepts seemed interesting (or coincided with known discoveries). In much of this work, the human intervention is not well-documented or not applied in a principled manner.

Humans can interact with a computer-aided discovery system during the pre-processing stage, the post-processing stage, and/or interactively during application of the discovery algorithms. Most systems require some type of human interaction during the pre-processing stage, such as tuning the data to the learning algorithm, removing noise from the data, constraining the output representation, or tuning the hypothesis space.

Bayesian methods require humans to tune the hypothesis space, which constrains what the algorithm can learn and often ensures that a search through the space remains tractable. For example, in a study that trained a Bayesian network to use mammographic data to estimate breast cancer risk, the researchers constrained the network model to search for dependencies between approximately thirty-five features and limited the number and type of dependencies that the discovery algorithm could form. The network discovered a dependency between breast cancer risk and mass stability that experts had not previously detected and accessed breast cancer risk more accurately than the human radiologists (Burnside et al., 2009).

During the pre-processing stage, many systems allow researchers to provide background knowledge that guides the discovery process. For example, Progol, an inductive logic programming system, allows users to provide background knowledge that the program uses to constrain its search. This system has successfully identified parts of a molecule that cause genetic mutations (King et al., 1996).

Most systems also allow researchers to tune the data to guide discovery. For example, researchers supply GRAFFITI, a system that creates conjectures in discrete mathematics and graph theory, with a diverse set of graphs, so that the system generates conjectures that generalize to many situations (Fajtlowicz, 1988). This has allowed the program to identify conjectures that have led to a number of proofs, at least one of which was important enough to include in a math journal (Langley, 1998).

The computer-aided discovery work in this thesis requires essentially no pre-processing of the data, because it uses a game's rules to build a state space and then uses the state space to classify visual data. As mentioned in the last section, the classified visual data could be used as input to a machine learning algorithm. In this way, much of my work pre-processes data into classes so that a human or learning algorithm can then search for patterns within the classes. The pre-processing stage of my computer-aided discovery work only requires that a human supply background information that contains the rules of the game, a procedure for using those rules to build the game's state space, and a procedure for distinguishing between win, lose, and draw states.

The DaViCCAND system, a system for learning quantitative models from data, allows user to interactively guide search by giving the users multiple points at which they can select variables for analysis, focus analysis on certain regions of data, use qualitative descriptions to identify important subsets of data, and identify points as outliers. DaViCCAND has successfully been used to develop better quantitative models for removing impurities from slag, a process necessary for creating iron and steel (F. Mitchell et al., 1997).

MECHEM, a system for formulating reaction pathways for complex chemical reactions, also allows users to interactively affect its search process, this time by letting users periodically suggest which pieces of background knowledge MECHEM should take into consideration (Valds-Prez, 1995). At the end of its analysis, MECHEM provides users with a set of possible pathways, and users perform post-processing by analyzing the results and running experiments to determine which of MECHEM's outputs are genuine. Thus, MECHEM provides humans with a constrained search space, and humans perform the final stage of analysis, which so far has resulted in at least three publications in chemistry journals (Langley, 1998).

Throughout the majority of the thesis, the computer provided a constrained search space, and I identified visual concepts and tactics during the post-processing stage. Switching to an interactive mode of computer-aided discovery would allow the computer to perform a larger portion of the analysis, an idea that I explore in more detail in chapters 6 and 7. Appendix A also describes how the pattern grouping algorithms can be used interactively to identify concepts in networks.

2.3 Concept Formation

A good deal of research in cognitive science has been devoted toward ascertaining the nature of concepts and how they are represented within the human mind. While I do not address the issue of how the human mind represents concepts in this thesis, examining various attempts to characterize the nature of concepts will help to ground our exploration of use-driven concepts. Unless otherwise noted, the information described in this section comes from the first few chapters and the last chapter of Murphy (2002).

Cognitive science researchers have developed four theories of concepts, the first of which has essentially been disproven. The first theory states that concepts consist of definitions that clearly differentiate between entities that are examples of the concept and entities that are not. This formulation of concepts has proven to be too brittle to account for most empirical results. The three theories of concepts still under consideration are the prototype theory, the exemplar theory, and the theory theory. Prototype theorists have proposed several approaches for representing concepts, including weighted feature-based representations and schema or frame-based representations. Supporters of the exemplar theory argue that concepts are simply represented by a set of examples. Proponents of the theory theory argue that concepts are stored in conjunction with general knowledge about the world, but do not seem to support any particular representation for concepts.

In this thesis, the term concept will generally refer to ideas that can be used to express human-interpretable descriptions of positions and tactics. Throughout the thesis, I loosely adopt ideas from exemplar theory and the theory theory. In chapters 5 and 6, I use an exemplar version of concepts to extract useful visual concepts. In Chapter 6, I outline a procedure for identifying use-driven visual concepts by searching for concepts that can be used to succinctly describe a set of tactics. This procedure implicitly builds on the idea that concepts can only be understood when considered in relation to domain knowledge. However, the cognitive theory most relevant to this work is Karmiloff-Smith's theory of representational redesciption. This theory suggests that learning involves iteratively redescribing information into more and more reusable units (Karmiloff-Smith, 1995). An automated version of the procedure for identifying use-driven visual concepts could provide a new computational model for representational redescription.

2.4 Games

In this section, I describe four thrusts in game research relevant to this thesis: game solving, extracting information from endgame databases, identifying tactical move sequences, and identifying patterns within a game's state space or board diagrams.

2.4.1 Solving Games

To date, researchers have solved a number of two-player perfect information games with state spaces of varying complexity, the most noteworthy being connect four with 10^{14} states (Allis, 1988; Allen, 1989), awari with 10^{12} states (Romein & Bal, 2003), qubic (4x4x4 tic-tac-toe) with 10^{30} states (Allis, 1994), go moku with 10^{105} states (Allis, 1994), nine men's morris with 10^{10} states (Gasser, 1996), and checkers with 10^{21} states (Schaeffer et al., 2007). Qubic and go moku rely on knowledge-based methods that filter out bad moves to reduce the complexity of the search space, an approach that only works in games where bad moves can be filtered out easily (Allis, van den Herik, & Herschberg, 1991). Connect four was solved both by using knowledge about potential threats (Allis, 1988) and by applying a brute-force search (Allen, 1989).

Brute force is by far the most popular approach taken by researchers. It has been used to solve all but two of the games listed above (all except for qubic and go muku). These solutions rely on either a full delineation of the state space or a combination of retrograde analysis and forward search. Retrograde analysis searches backwards from end states to build an endgame database that stores the game theoretical value of each state, i.e., whether the state is a win, lose, or draw when players play perfectly. These endgame databases also store the number of moves necessary to reach the nearest win, lose, or draw state. Because storing databases for the entire state space is not always possible due to space constraints, a forward search is used to determine the game-theoretical value of the start state by searching from the start state to states in the endgame databases. Van den Herik et al (2002) contains a thorough review of game solving approaches and successes.

2.4.2 Extracting Information from Endgame Databases

To solve nine men's morris, Gasser (1996) created endgame databases with 10^{10} states. Although he extracted statistical information about how frequently certain classes of states lead to a win and the longest paths leading to a win, he concluded that identifying "interesting positions" and determining which states were likely occur in real games was a daunting task. He also made no efforts to identify tactical patterns. Five and six men's morris, the two games analyzed in Chapter 6, are variations on nine men's morris. In this thesis, I address how to identify important positions, propose a method for determining which states are likely to occur during a normal game, and extract tactical patterns from the five men's morris state space.

Retrograde analysis has also been used to build endgame databases for other games, such as chinese chess (e.g., Fang, Hsu, & Hsu, 2002) and chess (e.g., Thompson, 1996), games with 10^{48} states and 10^{46} states, respectively (van den Herik, Uiterwijk, & van Rijswijck, 2002). Furnkranz (2001) reviews efforts to extract information from chess endgame databases. Early efforts focused on learning to classify whether a position was won, but did not produce human-interpretable results. Other efforts semi-autonomously acquired rules for classifying states, but required extensive interaction from experts, relying on them to decompose the endgame into subproblems, to refine the rules created by the computer, or minimally to indicate which computer-generated concepts were meaningful. Unlike work that focuses on classifying states, the work in this thesis uses classification in the state space to identify offensive and defensive tactics.

Bain and Srinivasan (1995) used inductive logic programming to classify states based on the number of moves necessary to achieve a win, which in theory would encode enough information for optimal play. While they created accurate classifiers for states in the KRK endgame that led to wins in less than six moves, these classifiers required too many rules for a human to interpret. The classifiers were also unable to distinguish between states that led to wins in six or more moves, so the approached failed to learn an optimal KRK strategy. In contrast, human players often use simple but suboptimal strategies to win the KRK endgame (Fürnkranz, 2001).

Sadikov and Bratko (2006) developed a method for separating chess endgames into stages that are characterized by different sets of goals. They sought to create a human-interpretable description of the strategy used in each stage using decision trees. Their approach was successful in identifying long term strategies, but could not compete with forward search as an approach for short-term planning. The authors noted that strategies identified were often too vague.

Guid et al. (2006) developed an interactive computer-aided system that helps experts develop human-interpretable subgoals and rules for playing endgames and tested it using the challenging KBNK endgame. Within the system, the expert and the computer interactively create a list of subgoals that lead to a win and develop rules that describe when and how to achieve each subgoal. In addition to proposing rules, the computer uses search to determine when a rule allows suboptimal play (play that does not follow the shortest path to a win) and asks the expert to determine whether the suboptimal routes still make sufficient progress toward the ultimate goal.

Instead of identifying stages within an endgame or lists of subgoals for winning a game, I use game databases to identify visual concepts and sets of offensive and defensive tactics that lead to wins or block an opponent's path to a win. While the tactics that I identify are optimal, my emphasis has been on developing human-interpretable descriptions of optimal tactics.

2.4.3 Identifying Move Sequences

Macro operators can facilitate problem solving and planning efforts by consolidating frequently-used sequences of basic operations. Early efforts to learn macro operators centered around problems that used an evaluation function to facilitate search. A good macro operator would smooth out the search space by creating operations that allowed the search algorithm "jump" over valleys in the search space. Researchers explored heuristics for creating helpful macro operators, such as identifying sequences of operations that led from one peak in the search space to another (Iba, 1989) or identifying sequences of operations that led from a peak to an even better evaluation function value (Finkelstein & Markovitch, 1998).

Korf (1985) created methods for identifying macro operators in problems, such as the Rubik's cube and the Towers of Hannoi, where an evaluation function cannot accurately predict the distance to the goal. Korf's approach involves learning macros that temporarily dismantle subgoals, provided that the macros reassemble all dismantled subgoals and achieve an additional subgoal by the time they finish executing. McGovern (2002) explored macro operator construction in a reinforcement learning environment using Markov decision processes, and Lock and Epstein (2004) developed an approach for learning macros for two-player perfect-information games by observing expert play.

Unlike the efforts described above, in this thesis, macro building does not involve an evaluation function, subgoals, or reinforcement learning. Instead, the pattern grouping algorithms that I describe in Chapter 3 identify and classify all action sequences (i.e., macros) of interest. Lock and Epstein developed methods for consolidating related action sequences, and I build on their work by using visual concepts to further compress sets of action sequences.

As with many pattern learning endeavors, macro learning suffers from the utility problem, namely that it is difficult to determine which macros will improve problem solving capabilities (Minton, 1990; Finkelstein & Markovitch, 1998). In the evaluation function setting, adding macros connects distant parts of the search space, but also increases the number of operations that must be considered at each step, so adding an arbitrarily high number of macros will actually impede search progress. My methods address the utility problem by identifying only the action sequences that lead directly from states reachable via perfect play to goal states. This automatically causes macro formation efforts to focus on the most useful action sequences. In this case, macro descriptions that compress the largest amount of information are the most useful.

2.4.4 Identifying Patterns in Game State Spaces and on Game Boards

In this thesis, I identify patterns in a game's state space and on the game board. In a case study, a human subject solved the game of pong hau k'i by consolidating nodes in the game's state space when they shared the same configuration of pieces on the game board (Epstein & Keibel, 2002; Epstein, 2005). Essentially, the human subject collapsed states with the same board diagram into a single node, regardless of who moved next. Using this compressed version of the state space, the subject identified offensive and defensive moves.

My pattern grouping algorithms expand on the idea of compressing nodes in a game's state space. The pattern grouping algorithms introduced in Chapter 4 compress the pong hau k'i state space in a way that makes it easier to extract important offensive and defensive moves. The pattern grouping algorithms introduced in Chapter 3 can identify deep forks by iteratively collapsing topological patterns in a game's state space, which relates to other efforts for identifying forks in games (e.g., Allis, 1992; Epstein, 1991; Yee, Saxena, Utgoff, & Barto, 1990; Collins, 1987; Minton, 1984).

Human subject experiments show that experts appear to use pattern recognition to select moves (see Gobet & Charness, 2006 for a review). Lock and Epstein (2004) define *context* as a visual pattern on a game board for identifying when to use an action sequence. They create contexts by labeling every location on a board as occupied by black, occupied by white, open, or unrestricted. Buro (1999, 2003) uses conjunctions of simple boolean features to automatically generate complex patterns, and these patterns enable his Othello game-playing program to win against any human player. Kaneko, Yamaguchi, and Kawai (2003) have expanded upon Buro's method in order to automatically generate complex patterns from logical formula. Levinson and Snyder (1991) used graphs to encode patterns in chess and capture tactical relationships between pieces.

While many state-of-the-art Go programs rely on human-generated patterns, efforts have also been made to automatically generate patterns (e.g., Kojima, Ueda, & Nagano, 1997; Kojima, 1998; Kojima, Ueda, & Nagano, 2000; Sei & Kawashima, 1998; Stoutamire, 1991) (see Muller, 2002 for a review). Cazenave (2001) uses a visual pattern paired with a set of conditions to describe a set of positions in Go. My work builds on Cazenave's patterns with external conditions by using visual concepts and sets of conditions to describe the context in which action sequences can be applied. The process through which I create these redescribed contexts builds on Lock and Epstein's absolute contexts (Lock & Epstein, 2004).

Chapter 3

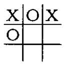
Pattern Grouping Algorithms

In this chapter, I introduce pattern grouping algorithms that serve two purposes. First, they perform the classification step of the equivalence class concept formation process by identifying instances of *annotated topological patterns* within a game's state space. As noted in Chapter 1, an annotated topological pattern is a subgraph or set of subgraphs that satisfies both topological constraints and constraints on the subgraph's node and edge attributes. The pattern grouping algorithms introduced in this chapter locate instances of annotated topological patterns that correspond to game-general concepts.

In addition to performing the first step of the equivalence class concept formation process, these pattern grouping algorithms hierarchically organize a game's state space by collapsing instances of annotated topological patterns. The collapsing operation results in depictions of a game's state space that expose high-level structure. Applying multiple pattern grouping algorithms in succession results in a set of state space diagrams at varying levels of granularity.

In this chapter, I describe the four pattern grouping algorithms that I use to analyze the tic-tac-toe games and morris games in Chapters 5 and 6, respectively. In Chapter 4, I describe several more algorithms designed specifically for the game of pong hau k'i. In the first section of this chapter, I introduce basic versions of the four pattern grouping algorithms. I explore how each algorithm simplifies a small state space, works in conjunction with the equivalence class principle to expose visual concepts, and highlights additional annotated topological patterns. In Section 3.2, I explain how the algorithms from the first section can be generalized and applied recursively to process all parts of a game's state space that lead to a win, lose, or draw state. I also describe the relative order in which the pattern grouping algorithms must be applied. In Section 3, I discuss the benefits of hierarchical pattern grouping algorithms.

3.1 Four Pattern Grouping Algorithms that Expose Visual Concepts and Simplify Game State Spaces

In this section, I introduce the four pattern grouping algorithms that I use throughout the majority of the thesis by demonstrating how these algorithms simplify the portion of the tic-tac-toe state space shown in Figure 3-1, i.e., the portion of the state space beginning with the state , where X moves next. When introducing each algorithm, I describe the algorithm, show how it simplifies the state space, and describe variations that I use throughout the thesis. I also show how some of the algorithms expose annotated topological patterns.

3.1.1 One-Move-to-a-Goal Grouping Algorithm

The first pattern grouping algorithm identifies regions of the state space in which a player can win in one move. Applying this algorithm reveals which states and moves remain reachable if both players take winning moves whenever they have the opportunity. This corresponds to assuming that each player can calculate only the outcome of his or her next move (i.e., has *one-step lookahead*) and that both players play perfectly given this limited computational ability. Identifying regions of this sort may reveal a visual concept that allows a player to recognize when she can win in one move. Variations on this pattern grouping algorithm can reveal regions of the graph

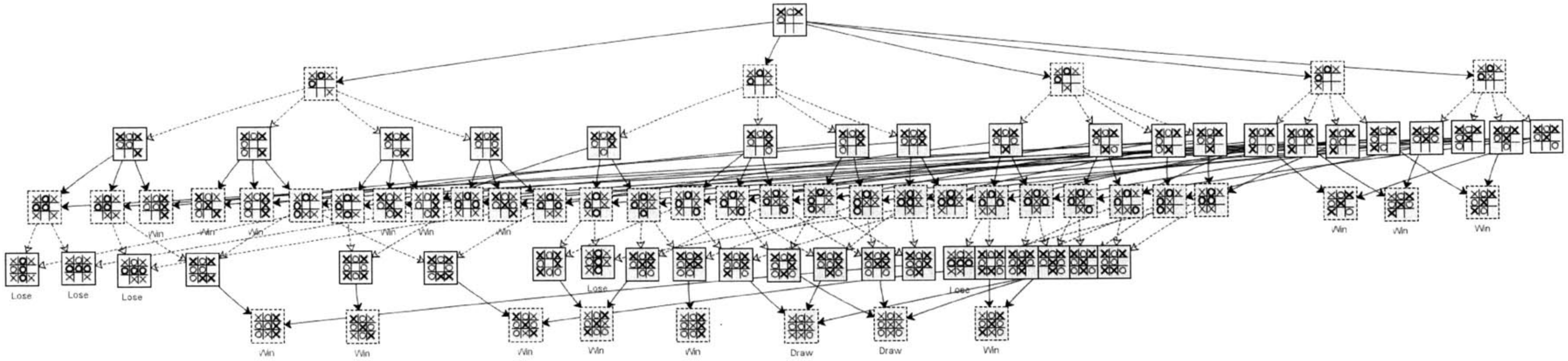


Figure 3-1: A subtree of the tic-tac-toe state space with 85 vertices and 143 edges. States are labeled from X's perspective, so a "win" state correspond to a win for X, while a "lose" state correspond to a win for O. Solid outlines and arrows indicate states where X moves next, while dotted outlines and arrows indicate states where O moves next.

that lead to other types of goals.

The most basic version of the algorithm proceeds by selecting a state where one player wins and where the winning player made the last move. In tic-tac-toe, this includes all states in which X has three in a row and X moved last, or O has three in a row and O moved last. Given a win state, the algorithm selects the parent of the winning state and all states that descend exclusively from the parent, as shown in Figure 3-2b. Figure 3-2c shows how the algorithm then collapses the selected states into a single leaf node, called a *collapsed subtree.* The collapsed subtree stores the selected subtree so that paths to the win state are retained for later use. Making the collapsed subtree a leaf node removes paths that become unreachable when players take any winning move available to them. It also highlights the fact that the collapsed subtree can be treated as an end state based on this limited perfect-play assumption. Applying this algorithm to the state space in Figure 3-1 results in the simplified state space shown in Figure 3-3.

Examining the roots of all collapsed subtrees, i.e., all *subtree roots*, that lead to winning and losing states reveals two visual concepts. These visual concepts involve lines that share a set of attributes and can be used to recognize states that are one move from a win. Namely, X can win in one move when the board contains a line with two X's and a blank spot, and O can win in one move when the board contains a line with two O's and a blank spot. Table 3.1 shows how these patterns are obtained using the equivalence class principle. While the visual concepts identified in this example seem simple, throughout the rest of the thesis, I will show how this technique can reveal more complex concepts.

I use several variations of the one-move-to-a-goal algorithm to analyze the tic-tac-toe and morris games, and I have identified additional variations could be used to analyze other games. For example, variations on the one-move-to-a-goal grouping algorithm can identify paths leading to other types of goals and states several moves away from a goal. Within these variations, the *goal-setter* is the player who wants to achieve the goal of interest. In our previous example, X was the goal-setter and X's goal was to win in one move. The general form of this pattern grouping algorithm

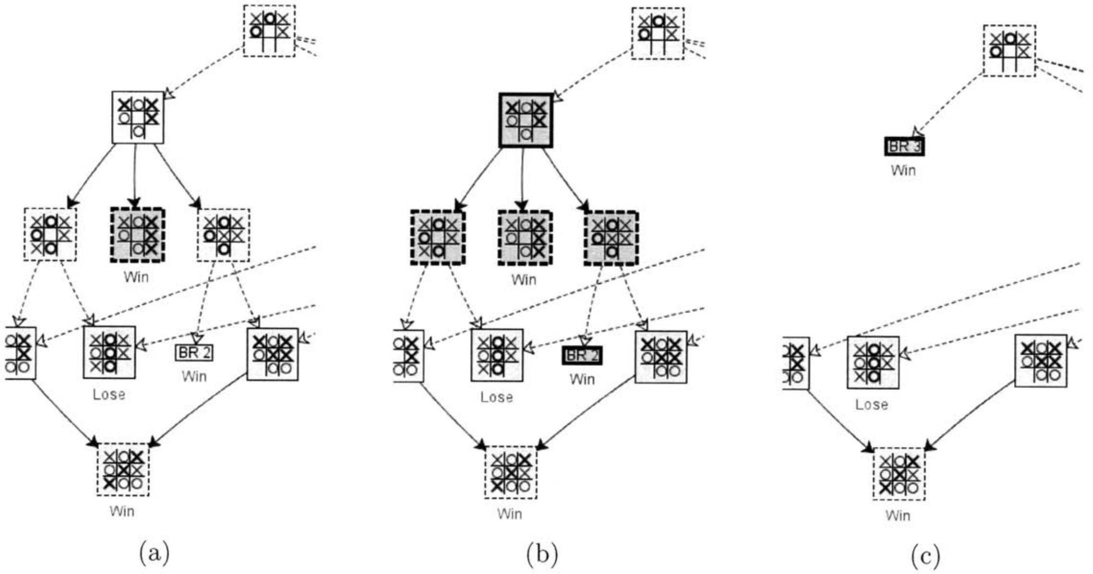


Figure 3-2: (a) The algorithm starts by selecting a win or lose state. In this example, it selects a state in which X wins. (b) The algorithm then selects the win (or lose) state's parent, P, and all descendants of P with parents that all descend from P. (c) The algorithm collapses the selected subtree and converts it to a leaf node. The new leaf node retains all of P's incoming connections. If the win or lose state has a parent that does not descend from P, a copy of the win or lose state should be left in the graph. This allows subsequent applications of the one-move-to-a-goal grouping algorithm to collapse any additional paths to this state. This operation can be seen as pruning the tree while retaining a copy of the path to the win state.

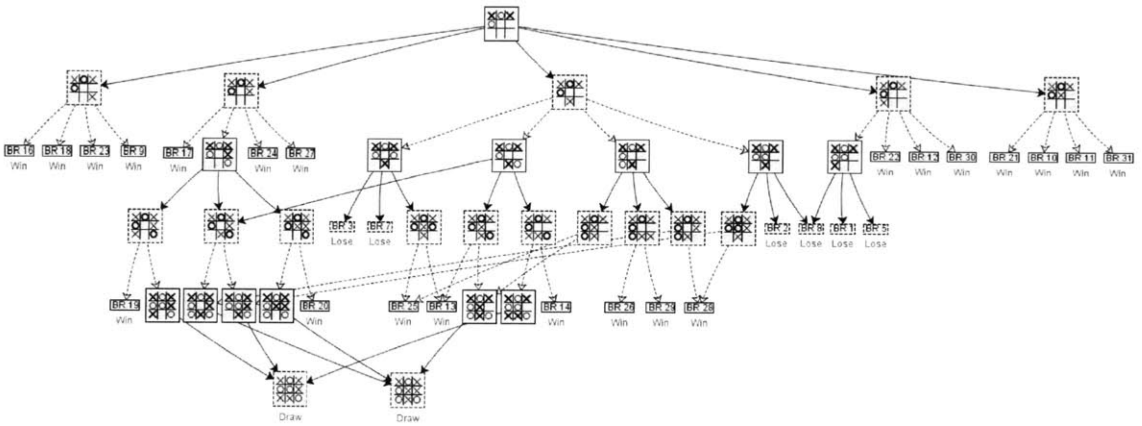


Figure 3-3: The state space from Figure 3-1 after the one-move-to-goal grouping algorithm has collapsed all regions that lead to a win for X or O. Collapsed subtrees leading to a win for X are labeled "win," while collapsed subtrees leading to a win for O are labeled "lose." Applying this grouping operation reduces the number of vertices in the state space from 85 to 58, and the number of edges from 143 to 69.

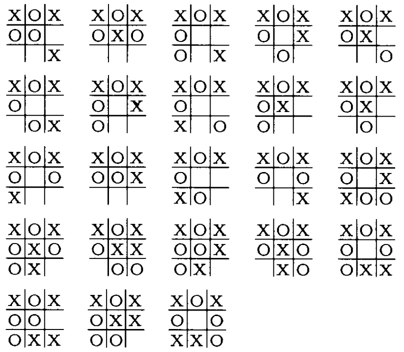
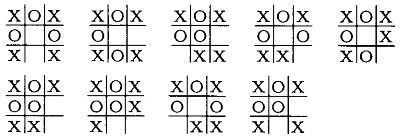
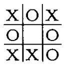
Annotation	Exemplar-Based Concept	Redescribed Concept
One move to a win for X, X moves next		Boards that contain a line with two X's and a blank spot.
One move to a win for O, O moves next		Boards that contain a line with two O's and a blank spot.

Table 3.1: A simple example that uses the equivalence class principle to identify visual concepts. The first column contains labels for the collapsed subtrees identified by the one-move-to-a-goal grouping algorithm. The second column contains the subtree roots that received the label from column one. The set of subtree roots form an exemplar-based description of a visual concept. The third column contains redescribed versions of the visual concepts from the center column.

starts by identifying a state or collapsed subtree that achieves a goal of interest, be it a piece capture, a win, a state where the player could win in three moves, or a particular configuration of the pieces on the board. It then locates regions in the state space where the goal-setter can make a single move that reaches the goal of interest.

Although the procedure depicted in Figure 3-2 selects and stores all nodes that descend exclusively from the goal node's parent, an alternate variation can select and store only the parent and the set of children that achieve the goal of interest. Any nodes that become unreachable after the parent and its goal-achieving children have been collapsed are subsequently removed from the graph, thus pruning the state space. Because this variation is more tractable for cyclic state spaces, I use it when analyzing the morris games.

While the labels used in Table 3.1 distinguish only between collapsed subtrees that lead to a win for X and collapsed subtrees that lead to a win for O, variations on this algorithm can distinguish between collapsed subtrees where the root is a *decision point* or a *fixed-choice point.* A subtree root is a decision-point root when the goal-setter has some moves that lead to its goal and some moves that do not. The subtree highlighted in Figure 3-2b has a decision-point root because only a subset of the root's children lead to the goal. In contrast, a fixed-choice root corresponds to situations where all of the goal-setter's moves lead to the goal. The tic-tac-toe state  is an example of a fixed choice root because X's only available move leads to a win. I use the fixed choice vs. decision point variation when analyzing both the tic-tac-toe and morris games in subsequent chapters.

The simplified state space shown in Figure 3-3 reveals two strategically-important annotated topological patterns that will be explored in the next two sections. First, the simplified state space contains several subtrees that consist of a root with a set of leaf children that all have the same label. The state space contains another set of subtrees where all but one of the children are leaf nodes that share the same label. These annotated topological patterns correspond to forks and forced avoidances, respectively.

3.1.2 Fork Grouping Algorithm

The fork grouping algorithm captures collapsed subtrees where the *opponent* moves next, i.e., the player opposing the goal-setter moves next, and all of the opponent's moves lead to a goal that the opponent would prefer to avoid. In tic-tac-toe, this corresponds to states where O moves next and all of O's moves lead to a win for X, or X moves next and all of X's moves lead to a win for O. Using this algorithm to collapse subtrees in Figure 3-3 results in a simplified state space that more succinctly highlights which paths lead to a win or a loss when each player takes a winning move whenever one is available. Applying the equivalence class principle reveals visual concepts that allow X or O to win on their next turn regardless of which move their opponent makes in the meantime. This algorithm can also be generalized to identify forks that lead to other goals such as piece captures and paths that lead to a goal in

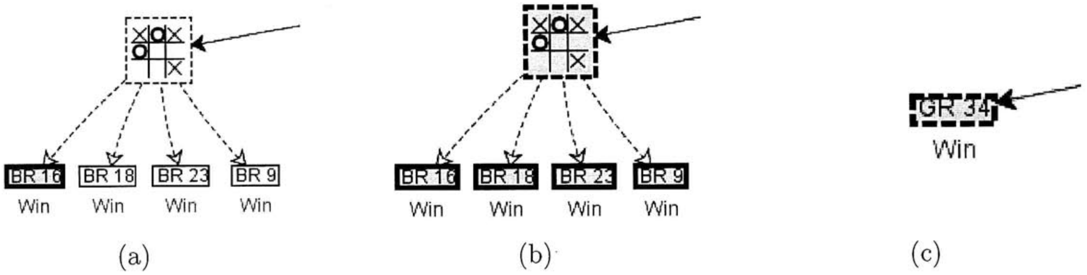


Figure 3-4: (a) The algorithm starts by selecting a win or lose state or a collapsed subtree that leads to a win or a loss with perfect play. In this example, it selects a collapsed subtree that leads to a win for X in one move. (b) The algorithm then selects the collapsed subtree's parent, P, and all of P's children. All of the children must lead to the same goal (here a win for X) and the parent must be a state where the opponent moves next in order for the subtree to be a fork. (c) The algorithm collapses the selected subtree and converts it to a leaf node. The new leaf node retains all of P's incoming connections.

an arbitrary number of moves.

This algorithm starts by identifying a state or collapsed subtree that results in a win and is reached via a move made by the opponent, i.e., the player that could lose the game by selecting the move in question (see Figure 3-4a for an example). Because the one-move-to-a-goal algorithm creates collapsed subtrees that fit this description, the fork grouping algorithm is often applied directly after the one-move-to-a-goal algorithm. The fork grouping algorithm proceeds by first selecting a state or collapsed subtree that meets this description. Then, it selects the collapsed subtrees's parent, P, (a state where the opponent plays next), and examines P's children's labels to determine whether all of its children achieve the same goal. If all the children do achieve the same goal, as in Figure 3-4b, the subtree corresponds to a fork and the parent and children are selected, otherwise, the pattern grouping algorithm does nothing. If the subtree is a fork, the tree is replaced by a leaf node, and the leaf node retains the parent's incoming connections, as shown in Figure 3-4c. Again, this leaf node stores a copy of the subtree so that all paths from the parent node to a goal state can be reconstructed. Figure 3-5 shows how the fork grouping algorithm simplifies the state space from Figure 3-3.

As in the last section, the subtree roots exhibit two visual concepts that can be

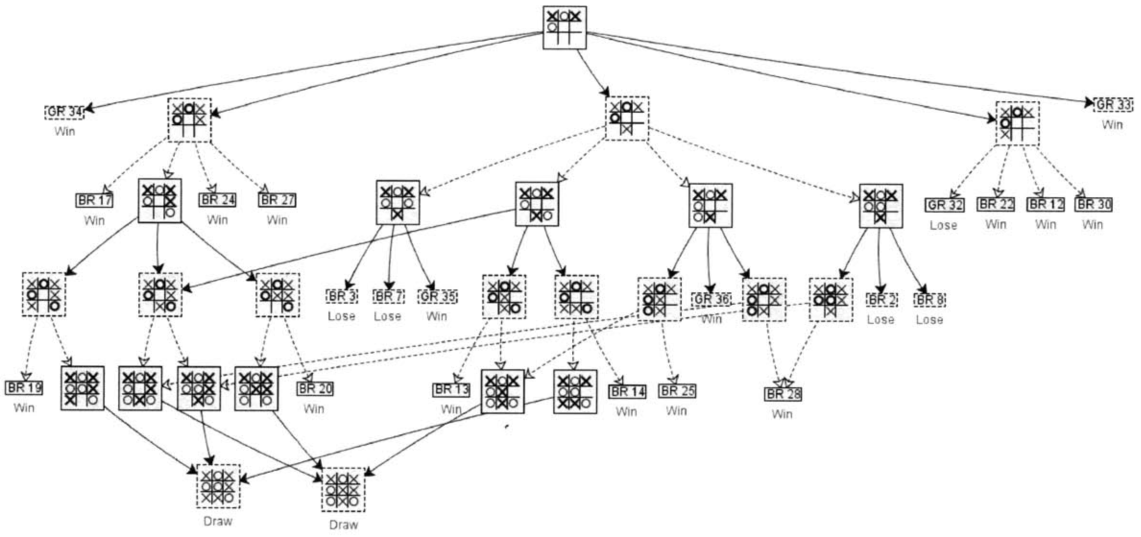


Figure 3-5: The state space from Figure 3-3 after the fork grouping algorithm has collapsed all regions where X and O cannot prevent their opponent from winning. Again, collapsed subtrees that result in a win for X, given perfect play, are labeled "win," while collapsed subtrees that result in a win for O, given perfect play, are labeled "lose." Applying this grouping operation reduces the number of vertices in the state space from 58 to 46, and the number of edges from 69 to 54.

used describe states that lead to a win: (1) X can win in one move regardless of what O does when the board contains two intersecting lines with two X's and a blank spot; (2) O can win in one move regardless of what X does when the board contains two intersecting lines with two O's and a blank spot. In this case, the roots shown in Table 3.2 may not seem like enough evidence to create the redescribed concepts described in the third column, but running this algorithm on the whole state space identifies a larger number of roots that all conform to the patterns described in the table. This example is simply intended to illustrate how the equivalence class principle can extract states that exhibit a strategically-significant visual concept. The principle will be applied more broadly in subsequent chapters.

Variations on the fork grouping algorithm can identify forks to other goals such as piece captures and forks to paths that lead to wins and losses after several moves of perfect play. The pattern grouping algorithm can distinguish between subtrees with multiple children (called *forks*) and subtrees with one child (called *forced moves*). The variation I use to analyze the tic-tac-toe and morris games distinguishes between

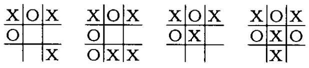
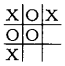
Annotation	Exemplar-Based Concept	Redescribed Concept
Fork to a win for X in one move, O moves next		Boards containing two lines that both contain two Xs and a blank spot and intersect at an X.
Fork to a win for O in one move, X moves next		Boards containing two lines that both contain two Os and a blank spot and intersect at an O.

Table 3.2: Another example showing how the equivalence class principle can identify strategically-significant visual concepts, this time using the fork grouping algorithm. The first column lists the labels applied by the fork grouping algorithm, the second column lists the roots of the subtrees that received the labels, and the third column contains a succinct description of the visual concept. Running this algorithm on the whole state space provides more evidence that the third column description is correct.

forks and forced moves and identifies collapsed subtrees that lead to wins and losses after many moves of perfect play.

3.1.3 Avoidance Grouping Algorithm

The avoidance grouping algorithm collects regions where the goal-setter's opponent can prevent the her from achieving her goal. This algorithm effectively removes paths from the state space that the opponent would prefer not to take. Running this algorithm after the ones described in the previous sections results in the state space that would be obtained if each player could predict only the outcome of both her next move and her opponent's next move, and both players play perfectly given this limited computational ability.

Like the fork grouping algorithm, this algorithm starts by identifying a state or collapsed subtree that results in a win and is reached via a move made by the opponent. Because the one-move-to-a-goal grouping algorithm described in Section 3.1.1 creates collapsed subtrees of this type, both the fork and the avoidance grouping algorithms are generally applied after the one-move-to-a-goal grouping algorithm. Again, the avoidance algorithm starts by selecting a state or collapsed subtree that leads to a win. In this case, the algorithm selects only the parent (a state where the opponent

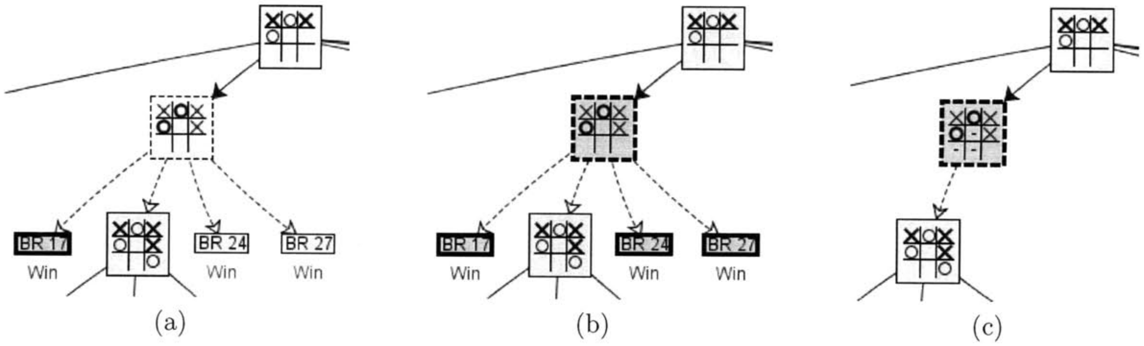


Figure 3-6: (a) The algorithm starts by selecting either a win or lose state or a collapsed subtree that leads to a win or a loss with perfect play. In this example, it selects a collapsed subtree that leads to a win for X in one move. (b) The algorithm then selects the collapsed subtree's parent, P, and all of P's children that lead to a win for X in one move. (c) The algorithm collapses the selected subtree and stores it in a node that retains all of P's other incoming and outgoing connections.

moves next) and a subset of the children: the subset of children that achieve a goal that the opponent wishes to avoid. The selected set of nodes are then stored within a new node that retains all of the parents other connections. Intuitively, this collapses the unwanted child nodes into the parent node and retains only the child nodes that the opponent would choose to move to, thus using the process depicted in Figure 3-6 to prune the tree. Applying this algorithm to the state space in Figure 3-5 results in the state space shown in Figure 3-7.

Applying the equivalence class principle again reveals strategically-important visual concepts, in this case lines containing two X's and a blank spot, where O must fill the blank spot on its next move to prevent X from winning. Table 3.3 shows the labels and roots collected using this algorithm.

This grouping algorithm can detect both *forced avoidances* and *multi-choice avoidances*. When the opponent only has one move that prevents the goal-setter from achieving his or her goal, the subtree root is called a forced avoidance. However, when the opponent has multiple moves that avoid the unwanted goal, the subtree root is instead referred to as a multi-choice avoidance.

After applying the three pattern grouping algorithms described so far to obtain the state space shown in Figure 3-7, the vast majority of the paths through the state space lead to draw states. This suggests that collapsing subtrees in which all paths

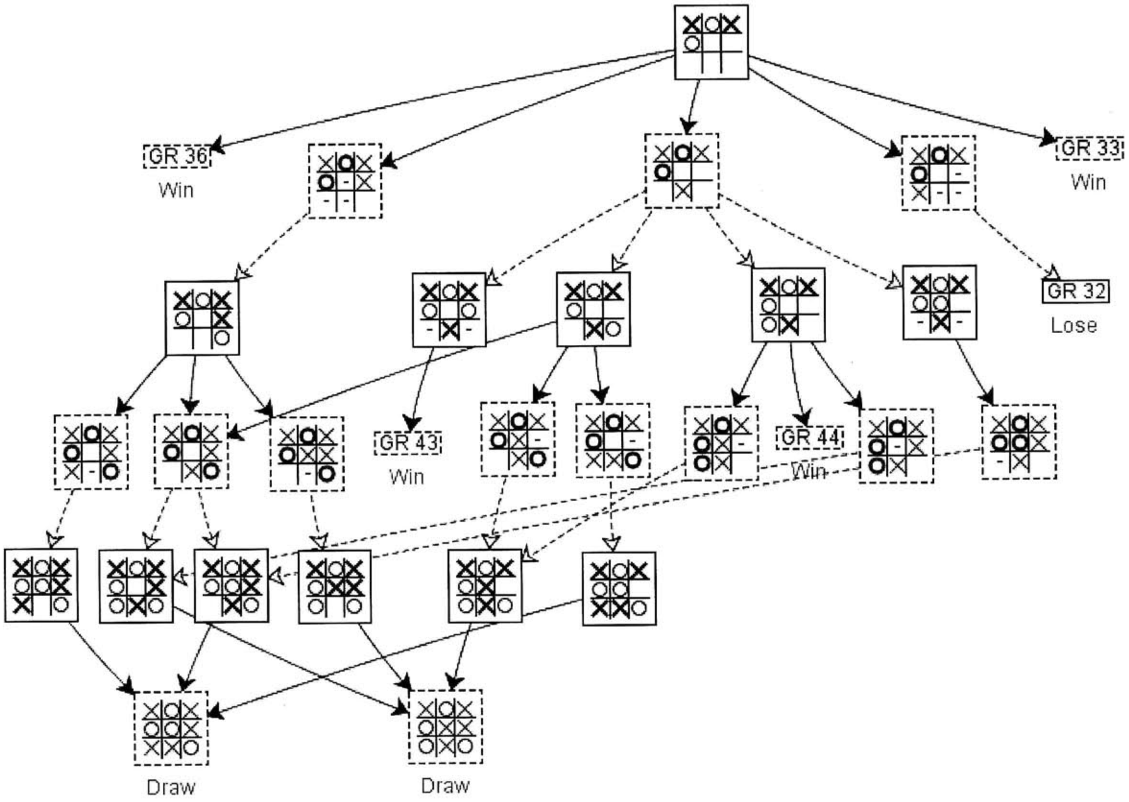


Figure 3-7: The state space from Figure 3-5 after the avoidance grouping algorithm has hidden all moves that X and O would not select because they lead to a win for their opponent. The collapsed subtrees are depicted using a tic-tac-toe board in which the suboptimal moves are marked with a dash. States where X moves next have a solid outline, while states where O moves next have a dotted one. The nodes labeled "win" and "lose" are forks identified by the fork grouping algorithm from the last section, where win indicates a win for X and lose indicates a loss for X. Applying this grouping operation reduces the number of vertices in the state space from 46 to 30, and the number of edges from 54 to 37.

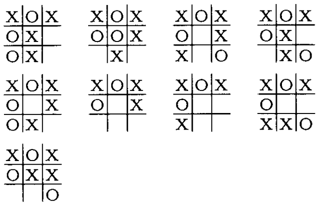
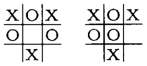
Annotation	Exemplar-Based Concept	Redescribed Concept
Forced avoidance of a 1-move win for X, O moves next		Boards with a line containing two Xs and a blank spot, where O must fill the blank spot.
Forced avoidance of a 1-move win for O, X moves next		Boards with a line containing two Os and a blank spot, where X must fill the blank spot.

Table 3.3: The avoidance algorithm also identifies strategically-significant visual concepts when used in conjunction with the equivalence class principle. The first column contains labels applied by the avoidance grouping algorithm, the second column contains subtree roots, and the third column contains succinct visual concept descriptions. The label "forced avoidance" indicates that the opponent has only one move that will keep the goal-setter from winning. Running this algorithm on the whole state space confirms the accuracy of the third column generalizations.

lead to the same result would further simplify the space.

3.1.4 Draw Tree Grouping Algorithm

The final pattern grouping algorithm collapses trees that lead to draw states. Because the avoidance grouping algorithm effectively removes paths in the state space that players are unlikely to take, the avoidance grouping algorithm often creates new draw trees within the state space. As the trees are identified, the draw tree algorithm determines how many moves of perfect play are necessary for the paths in the tree to lead to a draw. This number depends directly on the number of moves of perfect play, i.e., the amount of lookahead, required to form the avoidance collapsed subtrees within a particular draw tree.

The algorithm proceeds by identifying a draw state and searching up the tree to identify the highest root node whose descendants are all interior nodes or draw states (see Figure 3-8). The algorithm collapses the selected tree and labels it as a path to a draw state. Searching from the leaves up the tree allows the algorithm to more easily

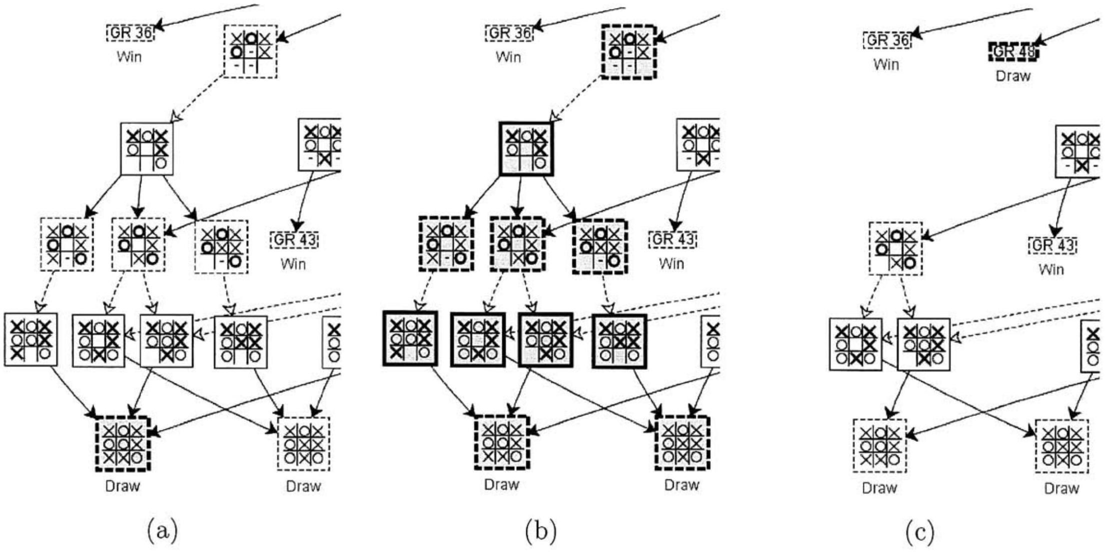


Figure 3-8: (a) The algorithm starts by selecting a draw state. (b) The algorithm then recursively searches up the tree until it identifies the largest subtree in which all paths lead to a draw state. (c) The algorithm collapses the selected subtree into a leaf node that retains the root's incoming connections. During the collapse, any nodes in the subtree that have ancestors that do not descend from the root node are not removed from the graph.

process extremely large state spaces.

As the algorithm searches up the tree, it examines all avoidance collapsed subtrees to see how many of the goal-setter's moves the opponent must anticipate in order for the avoidance collapsed subtrees to be created. In this case, all of the avoidance collapsed subtrees require 1-move lookahead, because the opponent only needs to anticipate one of the goal-setter's moves in order to keep the goal-setter from winning.

Applying this algorithm results in the simplified state space shown in Figure 3-9, but does not appear to reveal any strategically-significant visual concepts. Thus, this grouping algorithm makes the state space easier to analyze without identifying any visual concepts that predict whether a state leads to a draw. Table 3.4 lists the subtree roots obtained by applying the equivalence class principle. The lack of a salient patterns suggests that the equivalence class principle may not always facilitate the creation of visual concepts, but if the principle reveals visual concepts a significant portion of the time, it is still a valuable tool.

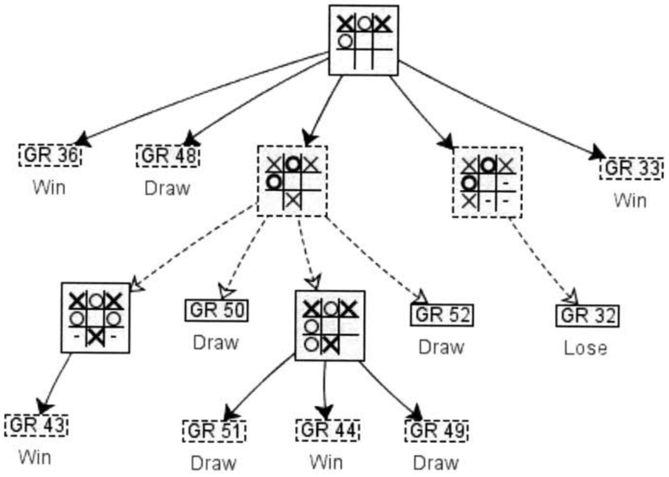


Figure 3-9: The state space from Figure 3-7 after the draw tree grouping algorithm has collapsed paths leading to draw trees. Applying this grouping operation reduces the number of vertices from 30 to 15, and the number of edges from 37 to 14.

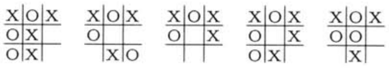
Annotation	Exemplar-Based Concept	Redescribed Concept
Path to a draw with 1-move lookahead		No obvious visual pattern

Table 3.4: The roots identified by the draw tree grouping algorithm do not reveal any obvious visual concepts that distinguish them from other states in the original state space. Because all of the states considered in this section contain XOX on the top row and the O in the leftmost square of the second row, the presence of this pattern does not distinguish the exemplars from other states in the state space from Figure 3-1.

3.2 Combining the Pattern Grouping Algorithms

The pattern grouping algorithms introduced in the last section lend themselves to a particular order of application. Recall that each of the pattern grouping algorithms collapses instances of a particular annotated topological pattern. If an algorithm cannot find instances of its topological pattern, the algorithm does nothing. To avoid extraneous applications of the algorithms, a pattern grouping algorithm should only be applied when instances of its annotated topological pattern are likely to be in the state space.

The pattern grouping algorithms from the last section often collapse subtrees in a way that creates new instances of another algorithm's annotated topological pattern. This suggests an optimal order for applying the algorithms. For instance, the one-move-to-a-goal algorithm collapses subtrees that form the leaves of forks and avoidances. The fork grouping algorithm then creates collapsed subtrees that become leaves in the one-move-to-a-goal subtrees. These relationships occur because the players take turns and the one-move-to-a-goal grouping algorithm looks for roots where the goal-setter moves next, while the fork and avoidance pattern look for roots where the opponent moves next. Similarly, the avoidance grouping algorithm creates instances of draw trees by collapsing paths that players will probably avoid.

In this section, I describe the default sequence of pattern grouping algorithms, and in Chapters 5 and 6, I describe the variations on this sequence used to analyze the tic-tac-toe games and morris games, respectively. The default sequence collapses all parts of a game's state space that lead to a win, lose, or draw state when the players have unlimited computational capabilities and play perfectly. This effect can be achieved by recursively applying the pattern grouping algorithms from the last section.

Each recursive step consists of three stages, a stage that collapses subtrees in which the goal-setter moves next (using the one-move-to-a-goal grouping algorithm), a stage that collapses subtrees in which the opponent moves next (using the fork

Step	Algorithm(s)	Sample Annotation
1a	One-move-to-a-goal	One move to a win, goal-setter moves next
1b	Fork and Avoidance	Fork to or avoidance of a 1-move win, opponent moves next
1c	Draw	Draw tree with 1-move lookahead
2a	One-move-to-a-goal	Two moves to a win, goal-setter moves next
2b	Fork and Avoidance	Fork to or avoidance of a 2-move win, opponent moves next
2c	Draw	Draw tree with 2-move lookahead
Na	One-move-to-a-goal	N moves to a win, goal-setter moves next
Nb	Fork and Avoidance	Fork to or avoidance of an N-move win, opponent moves next
Nc	Draw	Draw tree with N-move lookahead

Table 3.5: The one-move-to-goal, fork and avoidance, and draw tree grouping algorithms are applied cyclically until an application of the one-move-to-a-goal pattern grouping algorithm fails to detect any paths that lead to a win. This occurs when the entire state space has been reduced to a single collapsed draw tree or all paths to wins have been absorbed into avoidance nodes. See Chapters 5 and 6 for examples.

and avoidance grouping algorithms), and a stage that collapses subtrees that lead to draw states (using the draw tree grouping algorithm). Each stage processes parts of the state space that lead to a win, lose, or draw state, given that the players have limited computational abilities and they play perfectly given the information available to them. As the algorithm proceeds, the players' computational abilities increase incrementally until the players have enough information to play perfectly. Throughout the recursion, the labels used to annotate collapsed subtrees change to reflect this increase in computational abilities. Table 3.5 shows the list of pattern grouping algorithms and labels.

Initially, the segmentation algorithm starts with the set of win, lose, and draw states. During the first step of recursion, the one-move-to-a-goal algorithm from Section 3.1.1 collapses all subtrees that lead to a win or lose state in one move. The collapsed subtrees created during this stage become starting points for the pattern grouping algorithms applied during the next stage. The fork and avoidance algorithms from Sections 3.1.2 and 3.1.3 use the new set of start points to identify all forks that lead to a win or loss in one move and all regions where a player can avoid a one-move win or loss. The draw tree grouping algorithm from Section 3.1.4 then starts with the draw states and collapses any tree that leads to these states. Draw states that remain in the tree and any draw trees created during this step are stored to become starting points for the next application of the draw tree algorithm. Collapsed subtrees created during this step of recursion receive one of the following labels: one move to a win/loss with a decision point root, one move to a win/loss with a fixed-choice root, fork to a win/loss in one move, forced move to a win/loss in one move, forced avoidance of a one-move win/loss, multi-choice avoidance of a one-move win/loss, or draw trees with one-move lookahead.

During the second step of recursion, the one-move-to-a-goal algorithm again processes subtrees in which the goal-setter makes the next move, but this time it uses the fork and forced move collapsed subtrees created during the previous step as starting points. Thus, the algorithm now identifies subtrees that lead to a goal, assuming that the goal-setter makes two perfect moves. These collapsed subtrees serve as starting points for the second application of the fork and avoidance grouping algorithms, and the draw states and collapsed subtrees from the previous iteration serve as starting points for the second application of the draw tree grouping algorithm. This recursive step creates collapsed subtrees with the following labels: two moves to a win/loss with a decision point/fixed-choice/forced-avoidance root, fork to a win/loss in two moves, fork with a forced avoidance root to a win/loss in two moves, forced move to a win/loss in two moves, forced avoidance of a two-move win/loss, multi-choice avoidance of a two-move win/loss, or draw trees with two-move lookahead.

In general, the fork and forced move collapsed subtrees created during the $N - 1^{st}$ step serve as a starting point for the N^{th} application of the one-move-to-a-goal algorithm, while the N-moves-to-a-goal collapsed subtrees serve as a starting point for the N^{th} application of the fork and avoidance grouping algorithms. All draw states and draw trees that remain in the simplified state space serve as starting points for the N^{th} application of the draw tree grouping operation. The recursion concludes when no more one-move-to-a-goal collapsed subtrees can be formed. At this point, all N-moves-to-a-goal collapsed subtrees have been absorbed by avoidance collapsed subtrees and the states that remain in the simplified state space are avoidance collapsed subtrees, draw states, draw trees, and states that do not lead to a win, lose, or draw state due to loops in the state space.

The collapsed subtrees created by the N^{th} application of the one-move-to-a-goal grouping algorithm are labeled as N-move-to-a-goal collapsed subtrees. The N^{th} application of the fork grouping algorithm creates forks to a goal in N moves or forced moves to a goal in N moves, while the N^{th} application of the avoidance grouping algorithm creates forced avoidances of N-move goals and multi-choice avoidances of N-move goals. Table 3.6 shows the detailed labels created by the one-move-to-a-goal, fork, and avoidance pattern grouping algorithms. In general, a collapsed subtree's label contains information about the subtree's root and the number of moves from the root to the goal.

The level and type of detail used to differentiate between different types of collapsed subtrees is an experimental parameter designed to separate board diagrams in a way that enables the equivalence class principle to highlight visual concepts. For example, distinguishing between forced avoidances and multi-choice avoidances in the tic-tac-toe games emphasizes the fact that tic-tac-toe contains a large number of forced avoidances, while lose tic-tac-toe contains a similarly large number of multi-choice avoidances. A forced avoidance in tic-tac-toe has strategic significance because it allows a player to create two marks in a row and thus force her opponent to make a move that blocks her from getting three in a row. This ability makes it possible to force a win more quickly in tic-tac-toe than in lose tic-tac-toe. If tic-tac-toe did not have such a large number of forced avoidances, the separation between forced and multi-choice avoidances might not be as meaningful. Ideally, the set of annotations

Algorithm	Annotation	Group Type
One-move-to-a-goal	N-move path to a goal with a *groupType* root, goal-setter moves next	Decision point root (subtree where only some children lead to the goal) Fixed choice root (subtree where all children lead to the goal and the root is not a forced avoidance collapsed subtree) Forced avoidance root (subtree with one ungrouped child where the subtree root is a forced avoidance collapsed subtree)
Fork	*GroupType* to a N-move win, opponent moves next	Fork (subtree with more than one child where all children lead to a goal that the opponent cannot avoid) Forced move (subtree with only one child where the root is not a forced avoidance collapsed subtree) Fork with a forced avoidance root (subtree with one ungrouped child where the subtree root is a forced avoidance collapsed subtree)
Avoidance	*GroupType* of a N-move win, opponent moves next	Forced avoidance (avoidance collapsed subtree where the opponent has only one move that does not lead to the other player's goal) Multi-choice avoidance (avoidance collapsed subtree where the opponent has more than one move that does not lead to the other player's goal)

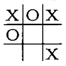
Table 3.6: The one-move-to-goal, fork, and avoidance grouping algorithms label collapsed subtrees in different ways based on the type of root in the subtree and how many children the subtree has. In general, the labels capture whether players have more than one option and whether players options are limited by needing to avoid paths to a win for their opponent.

should be tuned to create the clearest separation of visual concepts.

By examining uncollapsed nodes during different steps of the recursive process, one can explore the parts of a state space that remain reachable when players have only limited computational abilities. For example, applying the first four steps of the recursive process reveals the parts of the state space that remain reachable when each player can anticipate the next four of her opponent's moves and plays perfectly, given four-move lookahead. Because human players rarely achieve perfect play in a real game, analyzing the portions of a state space reachable via computationally-limited play provides a mechanism for identifying visual concepts and tactics that might become important in a real game.

3.3 Benefits of Hierarchical Grouping

The pattern grouping algorithms create a hierarchical organization of a state space by iteratively collapsing subtrees. When the one-move-to-a-goal grouping algorithm runs the first time, it creates a node that stores a subtree that contains paths to a win or lose state. This node is then stored as part of another subtree in a fork or avoidance node. The fork or avoidance node may be stored inside of another subtree the next time the one-move-to-a-goal grouping algorithm runs. These nested subtrees break the state space into manageably-sized segments that can be explored at multiple levels of granularity.

Figure 3-10 shows how a node in the simplified state space constructed in Section 3.1 stores all six paths leading from  to a win state (along with a path to a loss that can only be reached if X makes a mistake). Analyzing trees such as this one can yield insight into what moves a player must make to reach a win state and why a fork leads to a win regardless of what the opponent does. In Chapter 6, I show how the action sequences stored within these subtrees can be used to extract tactical patterns that lead to goals.

The hierarchial organization created by the pattern grouping algorithms provides a mechanism for visually exploring large state spaces. Meaningful sections of the state

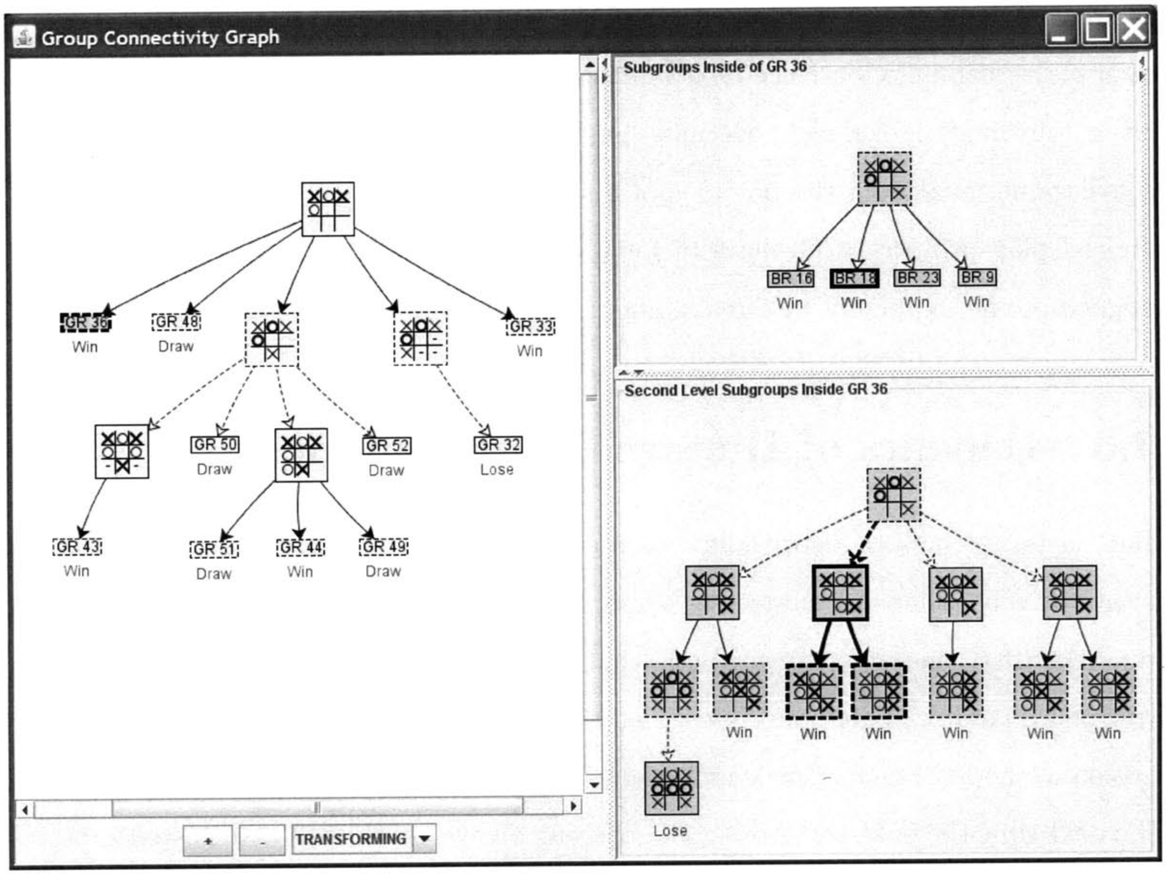


Figure 3-10: The nodes of the segmented state space store sections of the original state space. The node highlighted in the left panel (GR 36) contains the subtrees shown in the top right panel. When the leaf nodes in the top right panel are expanded, they create the subtree shown in the bottom right panel. The node highlighted in the top right panel (BR 18) expands into the highlighted subtree in the bottom right panel. Because the one-move-to-goal grouping algorithm from Section 3.1.1 only collapses child nodes that lead to a win or loss and child nodes that will be disconnected from the graph, the subtree in the bottom right panel only contains a subset of its root node's descendants.

space can be explored in isolation at whatever level of granularity makes sense, given the size of the state space. In theory, the segmented state space structure created by the pattern grouping algorithms could easily form the basis for a system that allows the user to explore complex state spaces by zooming in and out of areas of interest.

Chapter 4

Pattern Grouping Algorithms for Identifying Offensive Strategies

The pattern grouping algorithms in the previous chapter analyze portions of a state space that lead to a win, lose, or draw state. In this chapter, I describe several pattern grouping algorithms that can be used to analyze parts of a state space where players can force a draw in simple games that do not have draw states. In these games, players force a draw by circling around the parts of the state space where neither player can force a win.

Pong hau k'i, an ancient game from eastern Asia, is a fifty-six state game that meets this description (Zaslavsky, 1982; Bell, 1979). In this stand alone chapter, I describe a human case study involving pong hau k'i that inspired the pattern grouping algorithm approach that I use throughout the thesis. I also describe how I expanded on the results of the human case study to create pattern grouping algorithms that highlight offensive and defensive moves in pong hau k'i.

Pong hau k'i is played on the board shown in Figure 4-1. Play starts in the state shown in Figure 4-1a, and black moves first. During each turn, a player slides one of her pieces to an adjacent node. Play continues until one of the players can no longer make a sliding move. Thus, the goal of the game is to block the opponent's pieces so that the opponent cannot slide to an adjacent node. An example of a winning state for black is shown in Figure 4-1b.

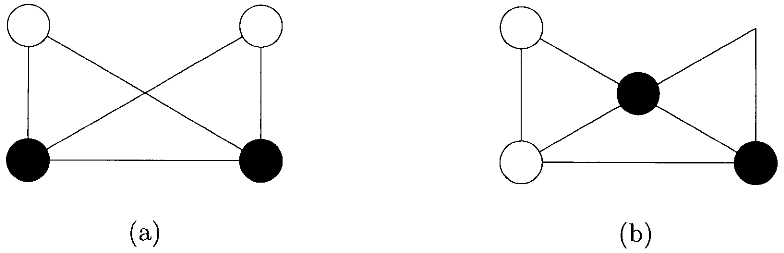


Figure 4-1: Two examples of pong hau k'i states. The pong hau k'i game board has five nodes where players can place pieces. On each turn, a player slides one of his or her pieces to an adjacent node, and play ends when one of the players has no legal move. (a) The starting state. Black moves first. (b) An end state in which black wins. White cannot slide either of its pieces because black blocks all adjacent nodes.

Figure 4-2a shows the entire state space for pong hau k'i. Even with only fifty-six states, this state space diagram is difficult for humans to interpret. While the layout in 4-2b reveals symmetry, it still hides the structure necessary for telling which moves are important. In this chapter, I describe how to create the simplified diagram in 4-2c, which makes it easy for a human to identify the strategically-significant offensive and defensive moves.

4.1 A Human Subject Simplifies the Pong Hau K'i State Space to Discover Offensive and Defensive Moves

A human subject, given only the rules of game and instructed to learn how to play, created the simplified state space diagram shown in Figure 4-3 (Epstein & Keibel, 2002; Epstein, 2005). The subject created this simplified diagram by collapsing any states with the same board diagram into a single node. Thus, each node in the simplified state space diagram represents two states, one in which black moves next and one in which white moves next. Due to this compression, the arrows in the state space diagram are bidirectional, and only certain paths through the state space are legal in a real game. In a real game, players must alternate turns, so legal paths alternate between solid and dotted arrows, which represent black's and white's moves, respectively.

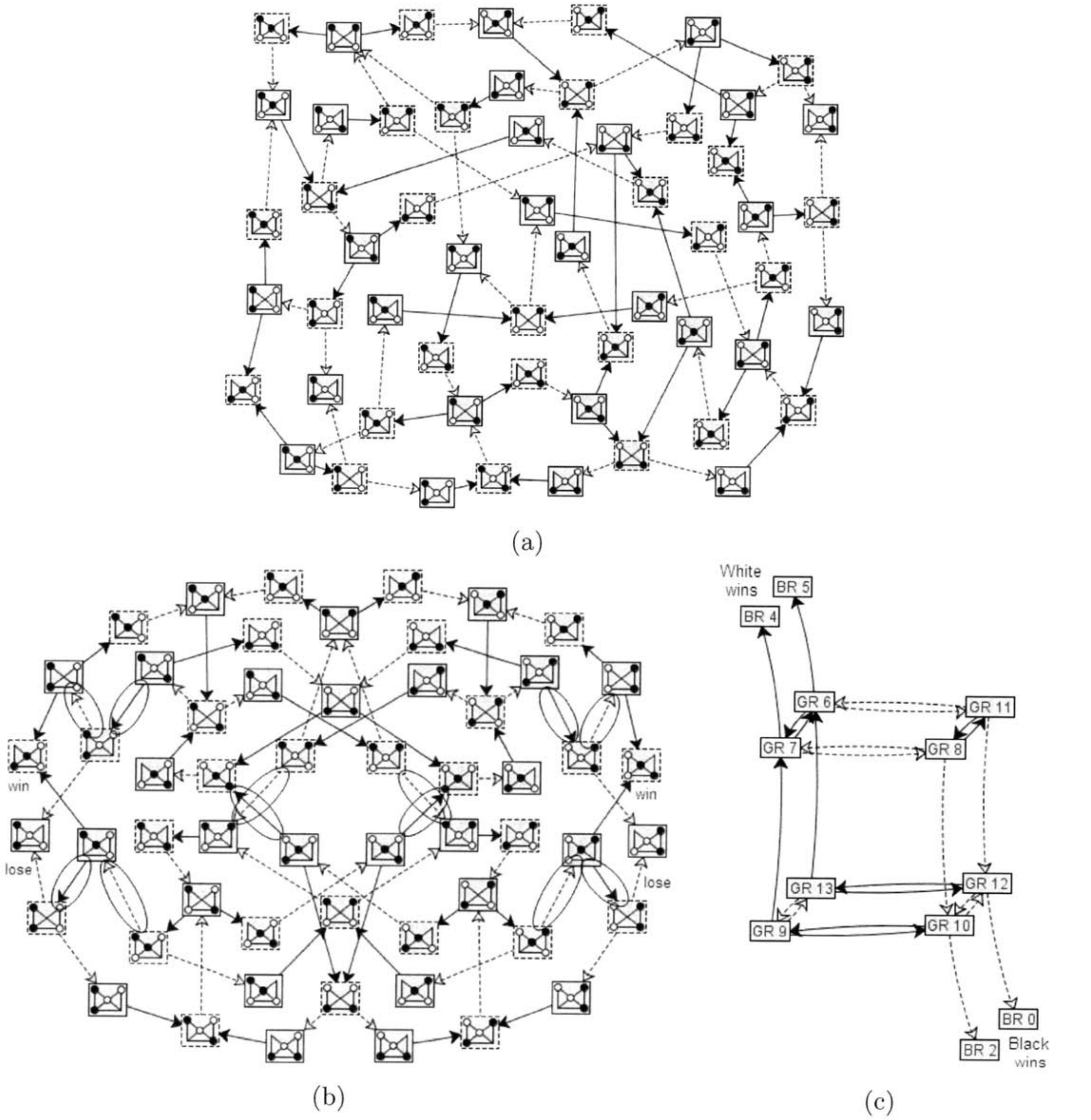


Figure 4-2: This figure is a reproduction of Figure 1-2. (a) The state space of pong hau k'i. Each game state consists of a board configuration paired with an indication of which player moves next. Dotted outlines and arrows indicate when white moves next, and solid outlines and arrows indicate when black moves next. (b) A symmetric layout of the same state space. The four circled moves near the center of the diagram are important offensive moves, while the rest of the circled moves are important defensive moves. (c) A simplification of the state space diagram for pong hau k'i. Non-leaf nodes represent loops in the graph. This diagram highlights the high-level structure of the game, exposing the key offensive moves.

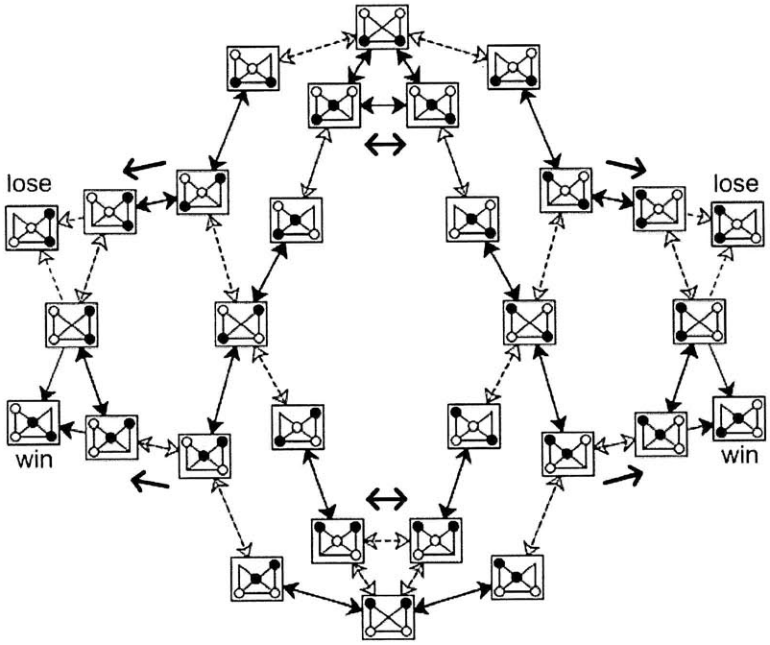


Figure 4-3: A reproduction of the collapsed pong hau k'i state space diagram created by the human subject. With the exception of the win and lose states, each node represents two states, one in which black moves next and one in which white moves next. Arrows with white heads represent moves made by white, while arrows with black heads represent moves made by black. Legal paths through the state space iterate between following a black arrow and following a white arrow. "Win" indicates a win for black and "lose" indicates a win for white. Strategically-important moves are highlighted using arrows that are not connected to graph. Players must avoid the highlighted moves on the right and left to keep from losing the game. Players must avoid the highlighted moves in the center of the diagram to keep from transitioning from offense to defense. (Adapted from Epstein, 2005.)

This simplified state space allowed the human subject to discover the important offensive and defensive moves within pong hau k'i. The discovery of offensive moves is particularly noteworthy because the existence of an offensive strategy for this game was hitherto unknown by members of the community. Prior to the subject's discovery, it appeared that the best a player could do would be simply to avoid losing the game. Figure 4-3 highlights the key offensive and defensive moves discovered by the subject, but using this diagram to identify the offensive moves is non-trivial.

4.2 Using Pattern Grouping Algorithms to Expose High-Level Structure in Pong Hau K'i

The approach used by the subject in the case study can be expanded upon to create a simplified diagram that highlights the offensive and defensive moves in pong hau k'i. In each of the following subsections, I describe a pattern grouping algorithm that performs part of this simplification.

4.2.1 Collapsing Paths to a Win

The first pattern grouping algorithm applied to pong hau k'i is the one-move-to-a-goal pattern grouping algorithm from Section 3.1.1. The resulting state space diagram is shown in Figure 4-4. Instead of running the avoidance grouping algorithm next, these leaf nodes are simply ignored during subsequent steps of the state space simplification process. The avoidance pattern grouping algorithm would collapse the leaf nodes into their parents, thus hiding the positions of the win and lose states. Instead, these leaf

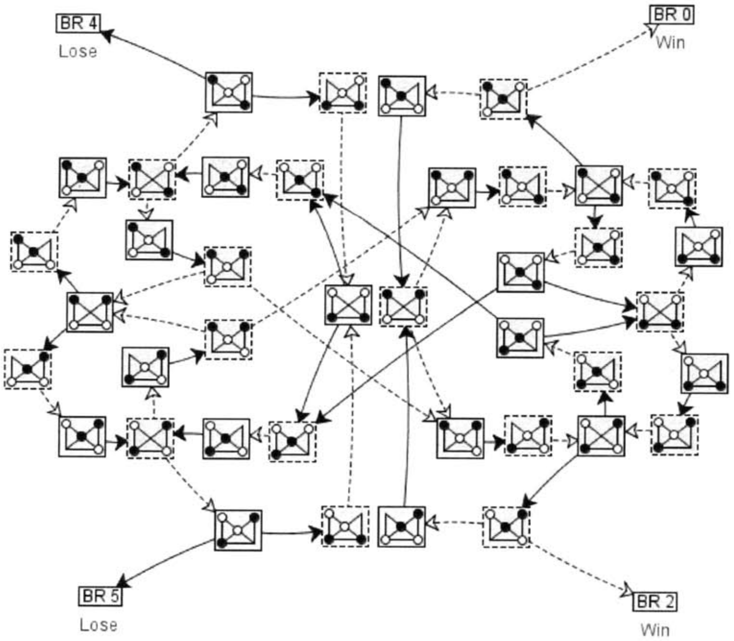


Figure 4-4: Pong hau k'i state space after applying the one-move-to-a-goal pattern grouping algorithm.

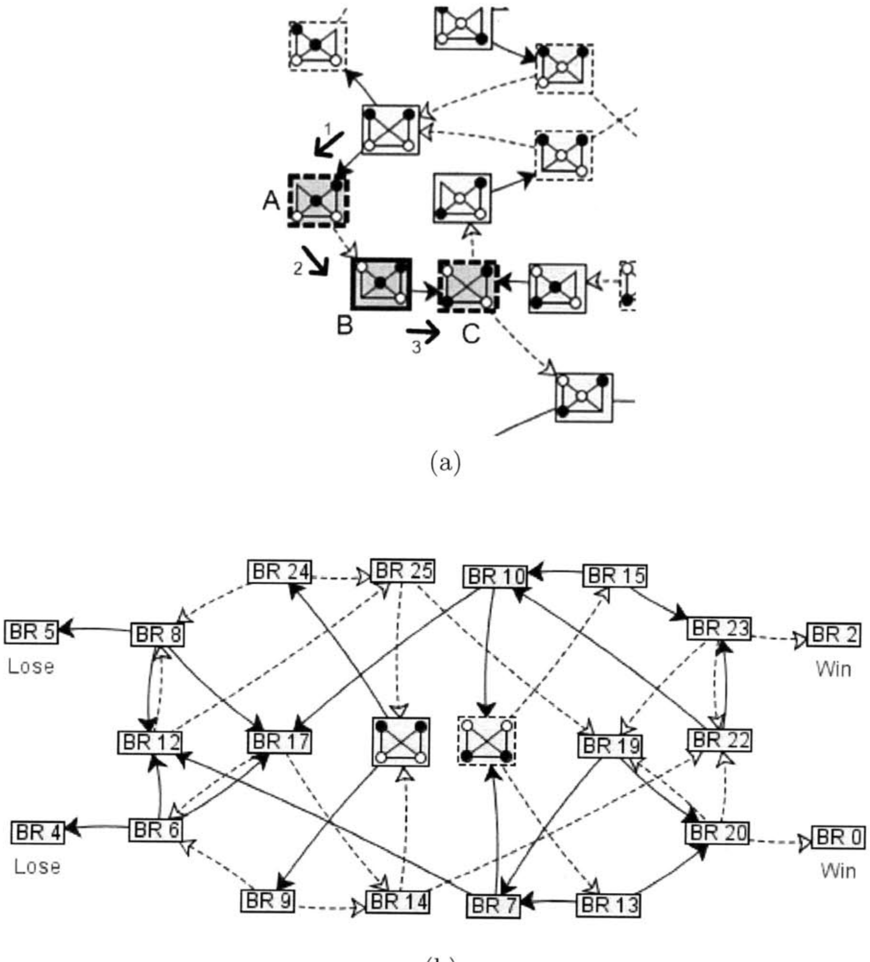
nodes are ignored when the remaining pattern grouping algorithms are applied, but they are left in the simplified state space diagram to make it easy to see the location of the win and lose states.

4.2.2 Collapsing Forced Move Sequences

After applying the one-move-to-a-goal pattern grouping algorithm, I applied a pattern grouping algorithm that identifies and collapses *forced move sequences* within the non-leaf portions of the state space. A forced move sequence is a sequence of moves where each player has only one move that they can select. Again, because leaves in the state space are ignored, a node with two children, one that is a leaf and one that is not, is treated as if it had only the non-leaf child during this stage of processing. A forced move sequence is collapsed into a single node that retains the incoming connections of the first state in the sequence and the outgoing connections of the last state. Figure 4-5 shows an example of a forced move sequence and shows the simplified state space obtained using this algorithm.

4.2.3 Collapsing Move Sequences that Exhibit the Same Motion Pattern

Most of the forced move sequences identified by the algorithm in the last section exhibit a visual motion pattern similar to the ones shown in Figure 4-6. Two sequences of moves, such as the sequences in 4-6a and 4-6b, exhibit the same *motion pattern* when the pieces slide along the same segments of the board in the same order, regardless of which piece makes which move. The pattern grouping algorithm in this section collapses adjacent nodes that exhibit the same motion pattern by starting with a forced move sequence node, extracting the node's motion pattern, and searching adjacent areas of the state space to find other nodes or groups of nodes that exhibit the same pattern. When a set of nodes exhibiting the same motion pattern is identified, it is collapsed into a single collapsed subgraph that retains all of the collapsed subgraph's incoming and outgoing connections. When a state is part of multiple mo-



(a)

(b)

Figure 4-5: (a) An example of a forced move sequence. The three highlighted edges and three highlighted nodes are all part of the forced moving sequence, which consists of a subgraph with an incoming edge. When the forced sequence is collapsed, all three nodes and three edges are stored within the collapsed subgraph. The state space retains a copy of edge 1 to connect the collapsed subgraph to edge 1's source. The collapsed subgraph is also connected to all of node C's incoming and outgoing edges, except for the incoming edges that is part of the forced sequence. (b) The pong hau k'i state space after the forced move sequence pattern grouping algorithm has been applied.

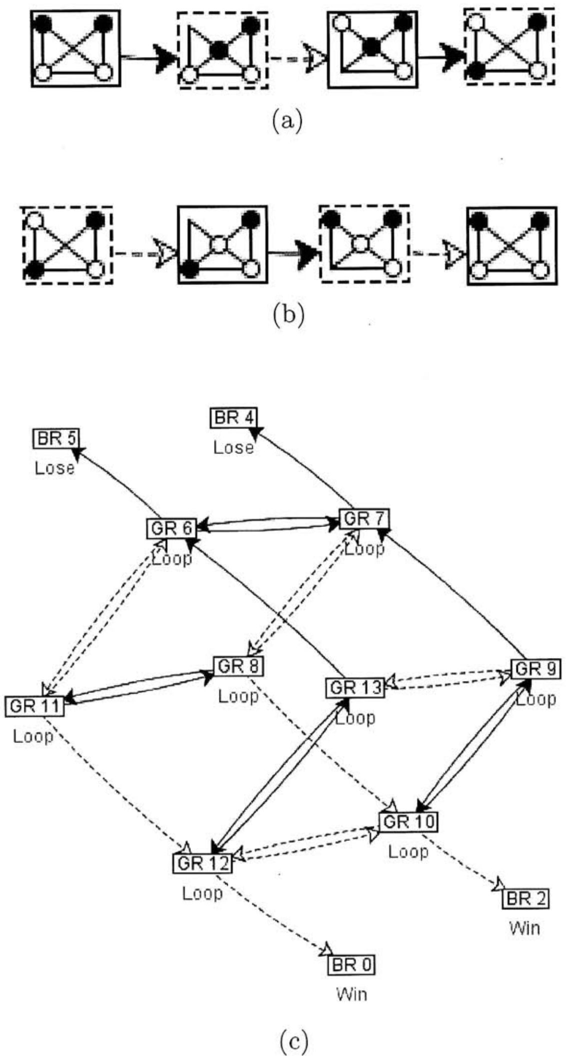


Figure 4-6: (a and b) Examples of two motion sequences that exhibit the same motion pattern. (c) The pong hau k'i state space after applying the motion pattern grouping algorithm.

tion patterns, multiple collapsed subgraphs retain copies of the state. Applying this algorithm to the state space diagram in Figure 4-5b results in the simplified state space diagram shown in Figure 4-6c.

Each of the newly formed collapsed subgraphs stores a loop from the original state space in which two of the pieces repeatedly chase one another around either the leftmost or the rightmost triangle of the board, while the remaining two pieces stay in a particular stationary configuration on the opposite side of the board. Because the players can chase one another around each of the two triangles in a clockwise and

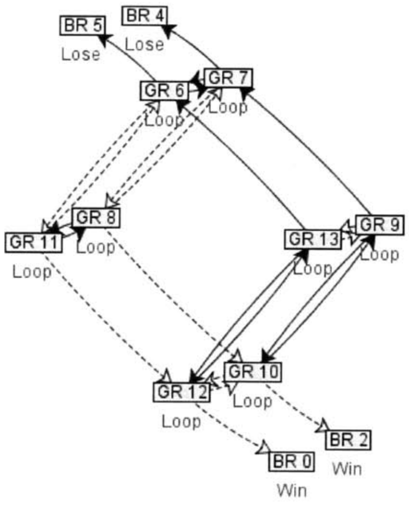


Figure 4-7: A reproduction of the simplification from Figure 4-6 in which the nodes are grouped based on symmetry.

counterclockwise direction, and because the remaining two pieces can be in one of two configurations during each of these chasing sequences, the state space contains eight different instances of this type of pattern. Each of the eight collapsed subgraphs encapsulate one such instance.

4.2.4 Grouping Collapsed Subtrees Based on Symmetry

The state space diagram from Figure 4-6 can be further simplified by noting symmetry within the board diagrams and within the topological structure of the collapsed subgraphs in the simplified diagram. Figure 4-7 groups collapsed subgraphs that correspond to symmetric parts of the original state space. For each pair, the collapsed subgraph represented by the first node can be obtained from its partner by horizontally reflecting each board diagram within the second collapsed subgraph.

4.2.5 Interpreting the Simplified State Space Diagram

The simplified state space diagram shown in Figure 4-7 exposes optimal offensive and defensive moves. Because the leaf nodes at the bottom of the diagram (BR0 and BR2) correspond to wins for black, white wants to avoid the two moves that lead to those nodes. At the same time, black wants to remain as close to BR0 and BR2 as possible, so black ideally wants to remain in GR10 and GR12 to maximize the likelihood that white will accidentally move to BR0 or BR2. This means that black can occasionally move to GR9 and GR13 and back to GR10 and GR12 to add variety to the game, but black should not move from GR9 and GR13 to GR6 and GR7, because doing so will move black from an offensive position to a defensive position. White's optimal strategy follows the same logic.

The pattern grouping algorithms that I introduced in this chapter highlight important offensive and defensive moves in pong hau k'i. While the drawn part of the state space could have been parsed in a different manner to highlight different loops in the state space, my technique identifies loops that exhibit a simple visual pattern that humans would theoretically find easy to remember. Thus, this technique provides a mechanism for finding structure in the drawn portion of a simple state space by exploiting salient patterns in a visual representation of the board.

Applying the equivalence class principle to the key offensive and defensive moves reveals that all important moves involve a piece moving across the bottom edge of the board. This can be seen by reexamining Figure 4-3. If for example, white slides a piece along the bottom edge of the board into position where its pieces form a vertical line, white will lose before its next turn, provided that black plays correctly. If white slides a piece along the bottom edge of the board while black's pieces both remain positioned at the top of the board, white will move from offensive to defensive. The only other moves in which white moves along the bottom edge of the board are unlikely to occur in a real game. Thus, using the equivalence class principle to extract visual patterns yields the succinct description of the optimal offensive and defensive strategies: players should avoid moving pieces along the bottom edge of the board.

Chapter 5

Extracting Strategic Information from Tic-Tac-Toe and Lose Tic-Tac-Toe

When applied to tic-tac-toe and lose tic-tac-toe, two 765-state games, the sequence of pattern grouping algorithms from Section 3.2 reveals optimal strategies and computational differences between the games. Combined with the equivalence class principle, these algorithms uncover strategically-important visual concepts that can be used to describe tactics. In this chapter, I present these results and highlight computational evidence that lose tic-tac-toe is more difficult than tic-tac-toe, show how the algorithms discovered a new strategy for exploiting mistakes in lose tic-tac-toe, and demonstrate how the visual concepts identified using the equivalence class principle can be used to create human-interpretable descriptions of winning tactics.

In the rest of this section, I introduce the tic-tac-toe games and describe the sequence of algorithms that I use to analyze these games. Lose tic-tac-toe is a variation on the well-known game of tic-tac-toe in which players lose when they get three in a row instead of winning when they get three in a row. Thus, the object of lose tic-tac-toe is to force your opponent to get three in a row. X moves first in both games.

In contrast to most games where players make winning moves, in lose tic-tac-toe players can make moves that cause them to lose. This affects the sequence of grouping algorithms used to simplify the lose tic-tac-toe state space. In tic-tac-toe, the parent of a state where X wins is a state where X moves next. Thus, the one-move-to-a-goal algorithm will find instances of its annotated topological pattern within the tic-tac-toe state space. In contrast, the parent of a state where X wins in lose tic-tac-toe is a state where O moves next. This follows because O must make a losing move and create three Os in a row in order for X to win. Because of this, the one-move-to-a-goal grouping algorithm will not find any instances of its annotated topological pattern within the original lose tic-tac-toe state space, but the fork and avoidance grouping algorithms will find instances of their respective patterns.

Thus, in lose tic-tac-toe, the first algorithm must detect situations where a player should avoid making a move that will cause her to lose. Because of this, I apply the fork and avoidance grouping algorithms first, followed by the draw tree grouping algorithm, to create a shortened first recursive step. Table 5.1 summarizes the sequence of grouping algorithms used to analyze the tic-tac-toe games. Table 5.2 (a reproduction of Table 3.6) shows how the collapsed subtrees are labeled.

The draw tree grouping algorithm could be run before any of the other grouping algorithms to identify paths that lead to a draw regardless of whether the players play intelligently. This application of the draw tree grouping algorithm would occur before the first application of the one-move-to-a-goal algorithm in tic-tac-toe and before the first application of the fork and avoidance algorithms in lose tic-tac-toe. However, preliminary analysis indicated that only two draw trees of this sort exist, providing only a minimal compression of the state space, so this grouping has been omitted from the rest of the analysis.

While tic-tac-toe and lose tic-tac-toe both technically have 5478 states, where a state is defined as a configuration of X and O marks on the board paired with an indication of who moves next, only 765 of these states are symmetrically distinct. In this chapter, I use sets of symmetrically-identical states when performing the state space analysis, so each state pictured in the state space diagrams actually represents all reflections and rotations of the pictured state. This creates an artifact in the state

Grouping Number	Algorithm(s)	Sample Annotation
1	Fork and Avoidance*	Fork to or avoidance of a 0-move win, opponent moves next
2	Draw*	Draw tree with 0-move lookahead
3	One-move-to-goal	One move to a win, goal-setter moves next
4	Fork and Avoidance	Fork to or avoidance of a 1-move win, opponent moves next
5	Draw	Draw tree with 1-move lookahead
6	One-move-to-goal	Two moves to a win, goal-setter moves next
7	Fork and Avoidance	Fork to or avoidance of a 2-move win, opponent moves next
8	Draw	Draw tree with 2-move lookahead
$3N$	One-move-to-goal	N moves to a win, goal-setter moves next
$3N + 1$	Fork and Avoidance	Fork to or avoidance of an N-move win, opponent moves next
$3N + 2$	Draw	Draw tree with N-move lookahead

Table 5.1: *The first two grouping operations only apply to lose tic-tac-toe. The one-move-to-goal, fork and avoidance, and draw grouping cycle repeats until no more one-move-to-goal collapsed subtrees can be created.

Algorithm	Annotation	Group Type
One-move-to-a-goal	N-move path to a goal with a *grouptype* root, goal-setter moves next	Decision point root (subtree where only some children lead to the goal) Fixed choice root (subtree where all children lead to the goal and the root is not a forced avoidance collapsed subtree) Forced avoidance root (subtree with one ungrouped child where the subtree root is a forced avoidance collapsed subtree)
Fork	*Grouptype* to a N-move win, opponent moves next	Fork (subtree with more than one child where all children lead to a goal that the opponent cannot avoid) Forced move (subtree with only one child where the root is not a forced avoidance collapsed subtree) Fork with a forced avoidance root (subtree with one ungrouped child where the subtree root is a forced avoidance collapsed subtree)
Avoidance	*Grouptype* of a N-move win, opponent moves next	Forced avoidance (avoidance collapsed subtree where the opponent has only one move that does not lead to the other player's goal) Multi-choice avoidance (avoidance collapsed subtree where the opponent has more than one move that does not lead to the other player's goal)

Table 5.2: The one-move-to-goal, fork, and avoidance grouping algorithms label collapsed subtrees in different ways based on the type of root in the subtree and how many children the subtree has. In general, the labels capture whether players have more than one option and whether players options are limited by needing to avoid paths to a win for their opponent.

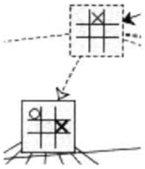


Figure 5-1: Each state in the state spaces represents all reflections and rotations of a board configuration. The board configurations used to represent the sets along a path may differ in orientation, resulting paths through the state space where the board appears to flip. I have made efforts to reduce the occurrence of this artifact.

space diagrams, shown in Figure 5-1, which causes boards to appear to flip along certain paths through the state space.

In the rest of the chapter, I explore the results obtained when the sequence of algorithms described above are applied to tic-tac-toe and lose tic-tac-toe. In the next section, I show how the algorithms collapse the tic-tac-toe and lose tic-tac-toe state spaces in a manner that highlights computational differences between the games and discovers optimal strategies. Humans find lose tic-tac-toe more difficult than tic-tac-toe (Ratterman & Epstein, 1995; Cohen, 1972), and the segmentation results presented in the next section suggest several reasons why. In the final section, I show how the equivalence class principle enables the computer-aided discovery of strategically-important visual concepts.

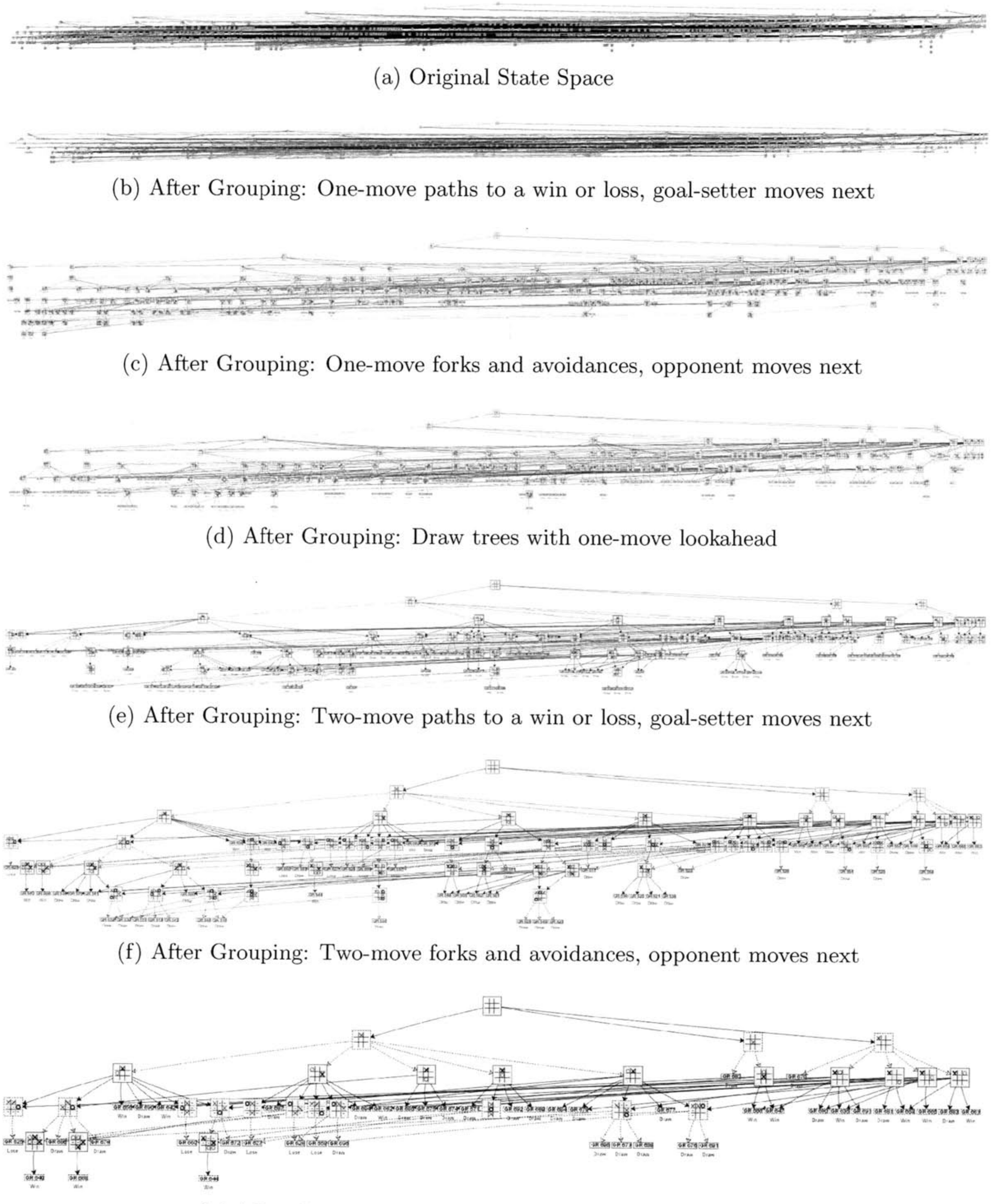
5.1 Segmentation Exposes Optimal Opening Moves, Strategies, Structure at Multiple Levels of Granularity, and Computational Differences between Games

Figure 5-2 shows how the pattern grouping algorithms reveal structure in the tic-tac-toe state space at multiple levels of granularity. Ultimately, tic-tac-toe reduces to a single collapsed subtree that leads to a draw, but in general, players must have 3-move lookahead in order to force a draw. Figure 5-2h shows a simplification of the state space that contains collapsed subtrees that lead to a draw, given two-move lookahead, and collapsed subtrees that lead to win or lose states, given 3-move lookahead. Figures 5-2h and 5-2i indicate that X's best strategy is to move into a corner, because unless O moves into the center, X can force a win in three moves. X can also win by initially moving to the center if O moves to a side. Otherwise, X can force a draw. Assuming both players can only anticipate their opponent's next two moves, O only has a chance of winning if X initially moves to a side. From there, O's best bet is to move to a corner. If players wish to force a draw, Figures 5-2h and 5-2i show that forcing a draw requires less lookahead if X moves into a corner or into the center than if X moves to a side.

Lose tic-tac-toe also reduces to a single collapsed subtree that leads to a draw (see Figure 5-3), but lose tic-tac-toe requires five extra grouping operations to do so (two in the beginning and three at the end). Figure 5-5 shows a comparison of the grouping speeds. The first two operations applied to tic-tac-toe, the one-move-to-a-win and the one-move fork and avoidance operations, cut the number of edges in the state space in half and in half again, and the one-move fork and avoidance operation drastically reduces the number of vertices in the graph. This indicates that most losses in tic-tac-toe can be avoided by a simple one-move lookahead, while avoiding a loss in lose tic-tac-toe generally requires more computational effort.

The extra grouping operations required for lose tic-tac-toe indicate that paths to a win or lose state are longer than in tic-tac-toe. In lose tic-tac-toe, players must always fill in every spot on the board to force their opponent to lose, while in tic-tac-toe, players can force a win or a loss more quickly, sometimes after only seven marks have been placed on the board. This suggests that lose tic-tac-toe requires more computational resources to play well, which agrees with empirical observations that people have more difficulty playing lose tic-tac-toe than tic-tac-toe (Ratterman & Epstein, 1995; Cohen, 1972).

The lose tic-tac-toe segmentation also shows that lose tic-tac-toe poses a particular challenge for the player who plays X. The simplified state space in Figure 5-3j (enlarged in Figure 5-4) shows collapsed subtrees that lead to win and lose states in



(a) Original State Space

(b) After Grouping: One-move paths to a win or loss, goal-setter moves next

(c) After Grouping: One-move forks and avoidances, opponent moves next

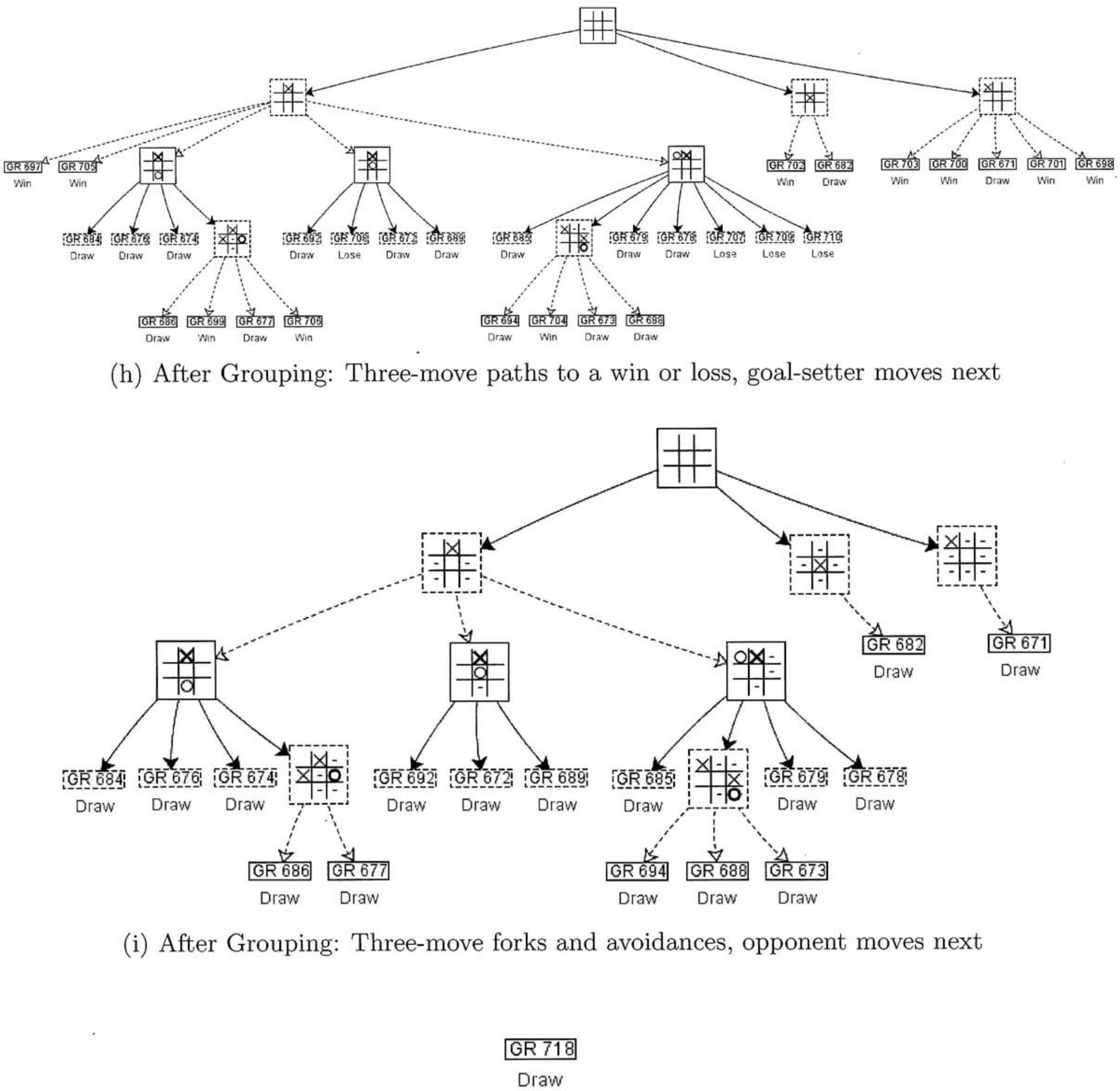
(d) After Grouping: Draw trees with one-move lookahead

(e) After Grouping: Two-move paths to a win or loss, goal-setter moves next

(f) After Grouping: Two-move forks and avoidances, opponent moves next

(g) After Grouping: Draw trees with two-move lookahead

Figure 5-2: Tic-tac-toe segmentation results.



(h) After Grouping: Three-move paths to a win or loss, goal-setter moves next

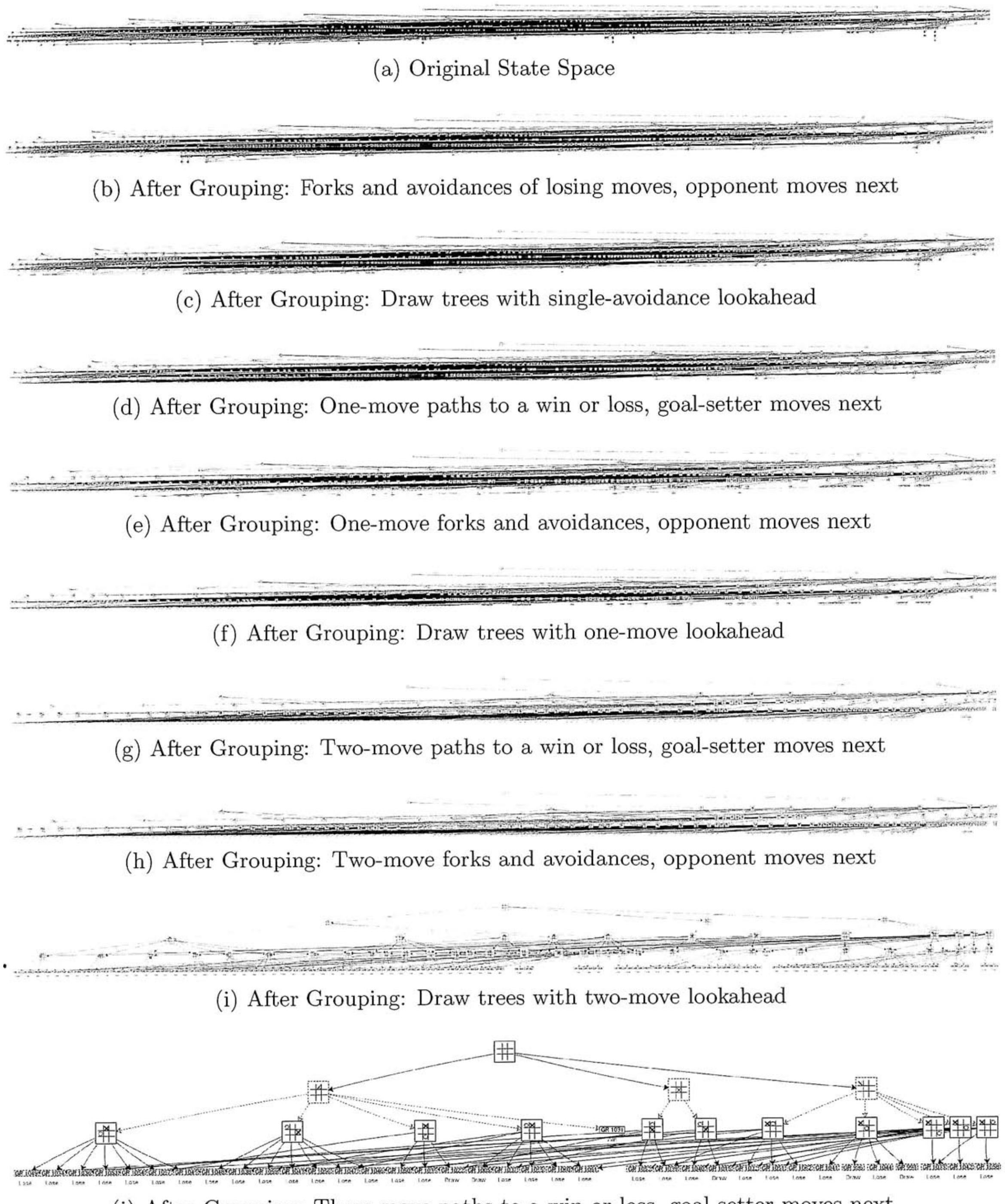
(i) After Grouping: Three-move forks and avoidances, opponent moves next

(j) After Grouping: Draw trees with three-move lookahead

Figure 5-2: Tic-tac-toe segmentation results (continued). Collapsed subtrees labeled win are a win for X, and collapsed subtrees labeled lose are a win for O. Boards with a horizontal line through certain positions are avoidance collapsed subtrees, where the horizontal lines indicate the moves that must be avoided to keep the opponent from winning. States where X moves next have a solid outline, while states where O moves next have a dotted outline.

three moves and collapsed subtrees that lead to draws, given two-move lookahead. The vast majority of paths lead to a win for O, because essentially all of the leaf nodes in the bottom row of the tree are a lose for X. More specifically, thirty-one of the leaf nodes are collapsed subtrees that lead to a win for O in three moves (the leaves on the bottom row of the tree that are labeled lose), while five of the leaves are collapsed subtrees that lead to a draw, given two-move lookahead, (the remaining leaves in the bottom row of the tree), and one leaf is a collapsed subtree that leads to a win for X (the only collapsed subtree in the third row of the tree). In contrast to Figure 5-2h, the corresponding segmentation for tic-tac-toe, the ratio of paths to wins versus losses shows that it is much harder for X to win lose tic-tac-toe than it is for O to win tic-tac-toe.

Selective application of the grouping algorithms exposes a difficult-to-discover winning strategy for X and strategy that allows X to exploit O's mistakes. The segmentation shown in Figure 5-6 shows a winning strategy for X that humans often fail to discover (Ratterman & Epstein, 1995), in which X reflects O's moves through the center. This segmentation also reveals a strategy for exploiting O's mistakes in which X forms a 2x2 square of X's. Despite computational and human subject studies of lose tic-tac-toe, the strategy for exploiting O's mistakes had not been discovered by other researchers (Ratterman & Epstein, 1995; Epstein, June 18, 2010). This example demonstrates that the segmentation approach can reveal strategies that would be difficult for a human to discover on her own.



(a) Original State Space

(b) After Grouping: Forks and avoidances of losing moves, opponent moves next

(c) After Grouping: Draw trees with single-avoidance lookahead

(d) After Grouping: One-move paths to a win or loss, goal-setter moves next

(e) After Grouping: One-move forks and avoidances, opponent moves next

(f) After Grouping: Draw trees with one-move lookahead

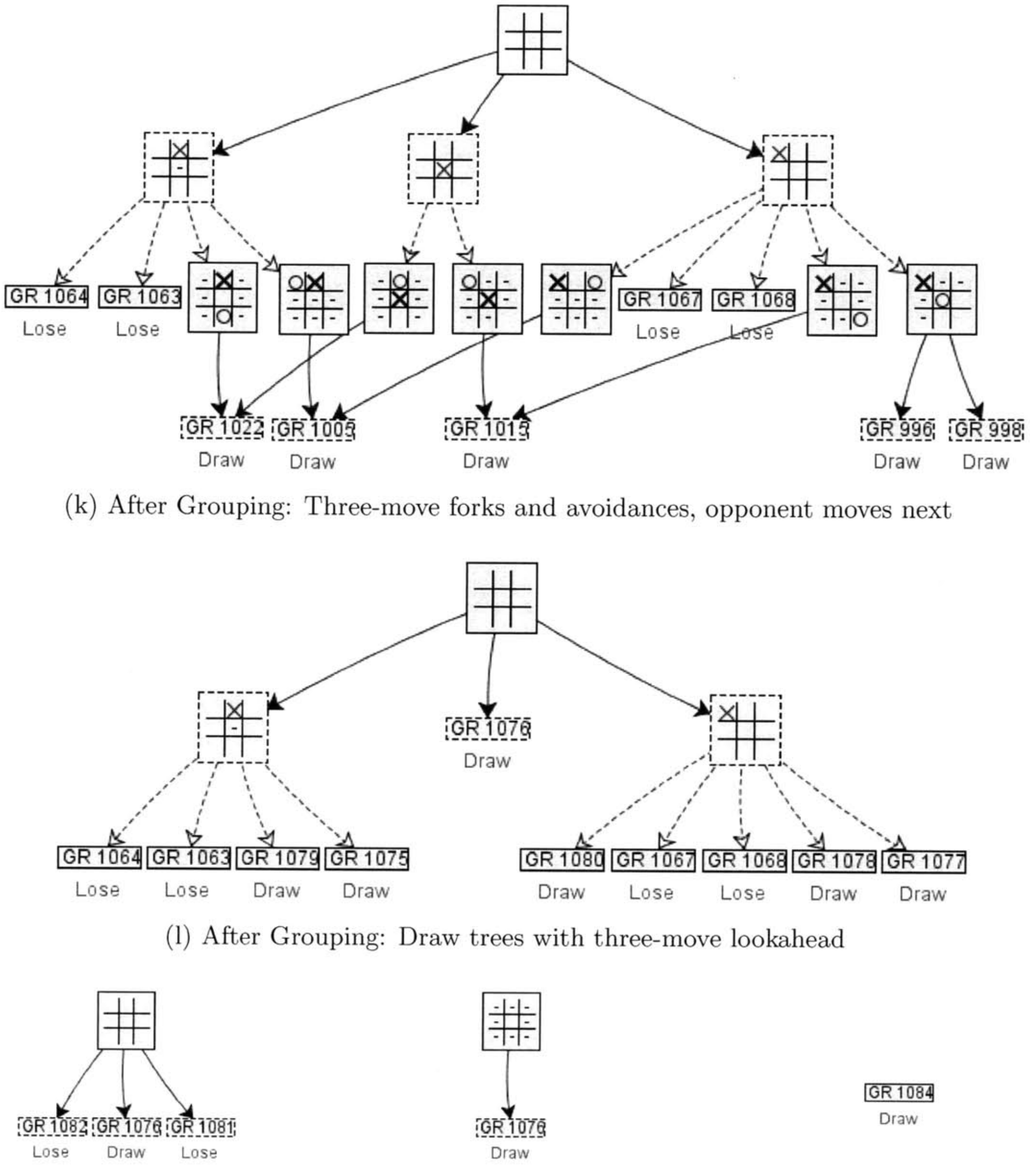
(g) After Grouping: Two-move paths to a win or loss, goal-setter moves next

(h) After Grouping: Two-move forks and avoidances, opponent moves next

(i) After Grouping: Draw trees with two-move lookahead

(j) After Grouping: Three-move paths to a win or loss, goal-setter moves next

Figure 5-3: Lose tic-tac-toe segmentation results.



(k) After Grouping: Three-move forks and avoidances, opponent moves next

(l) After Grouping: Draw trees with three-move lookahead

(m) After Grouping: Four-move paths to a win or loss, goal-setter moves next

(n) After Grouping: Four-move forks and avoidances, opponent moves next

(o) After Grouping: Draw trees with four-move lookahead

Figure 5-3: Lose tic-tac-toe segmentation results (continued). (a-h) Compared with Figure 5-2 (a-b), the lose tic-tac-toe state space requires six more grouping operations before the state space visibly looks cleaner. Note that the first two grouping operations apply only to lose tic-tac-toe. Simplifications (k-m) no longer contain subtrees that lead to a win for X, indicating that if O has three-move lookahead, O can force a win or a draw. Again, collapsed subtrees labeled win are a win for X (where a win for X means that O has gotten three in a row), and collapsed subtrees labeled lose are a win for O (where X has gotten three in a row). Subfigure j is enlarged and discussed in Figure 5-4.

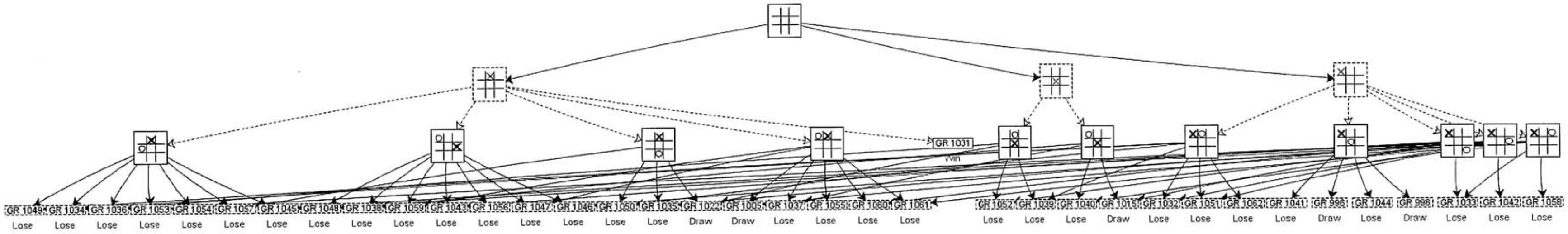
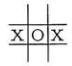
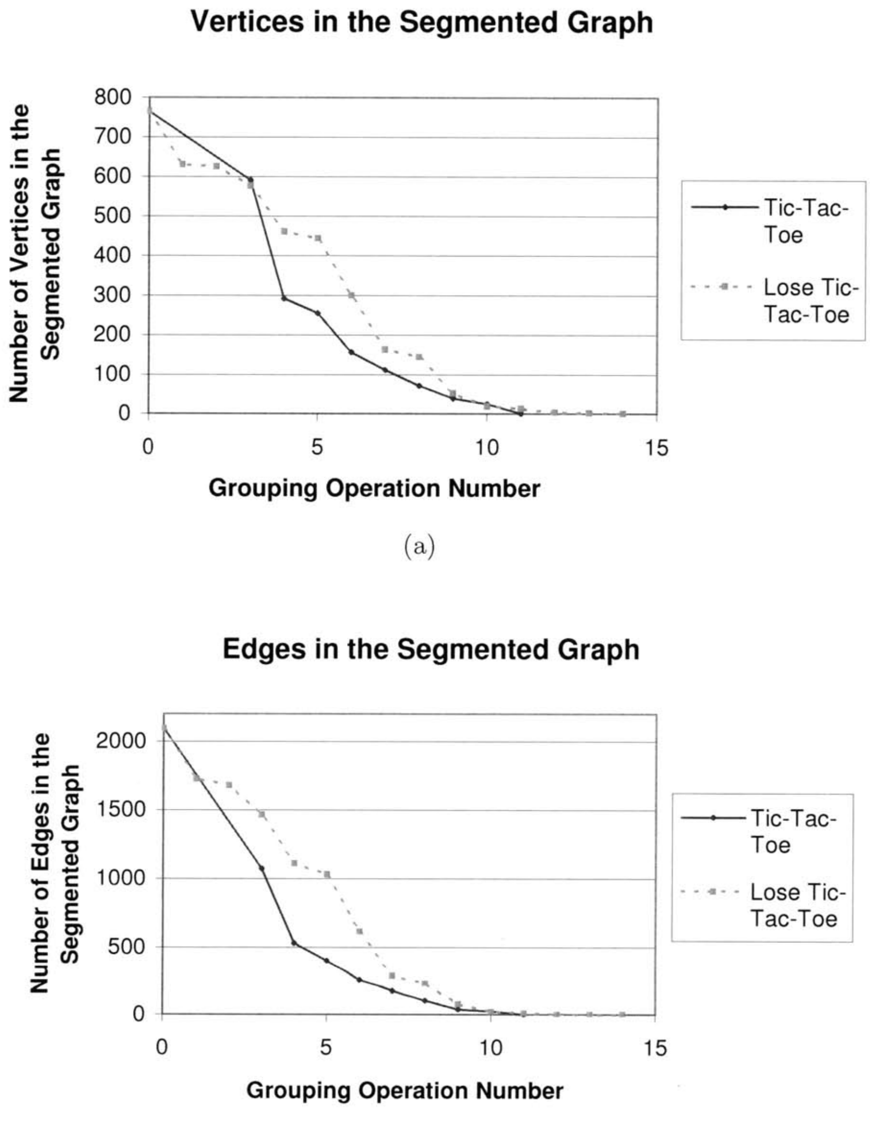


Figure 5-4: An enlarged version of Figure 5-3j. This shows all paths through the state space assuming that each player can anticipate the next three of their opponents moves (i.e., has three-move lookahead), and plans accordingly. Note that when players have three-move lookahead, X can only win via one path, and this path that involves reaching the state  . From this state, X can win the game regardless of where O moves next.



(a)

(b)

Figure 5-5: The number of vertices and edges in the segmented state space after each grouping operation. The grouping operation number corresponds to the numbers listed in Table 5.1. Operations three and four correspond to the one-move-to-a-win step and the one-move fork and avoidance operations step, respectively.

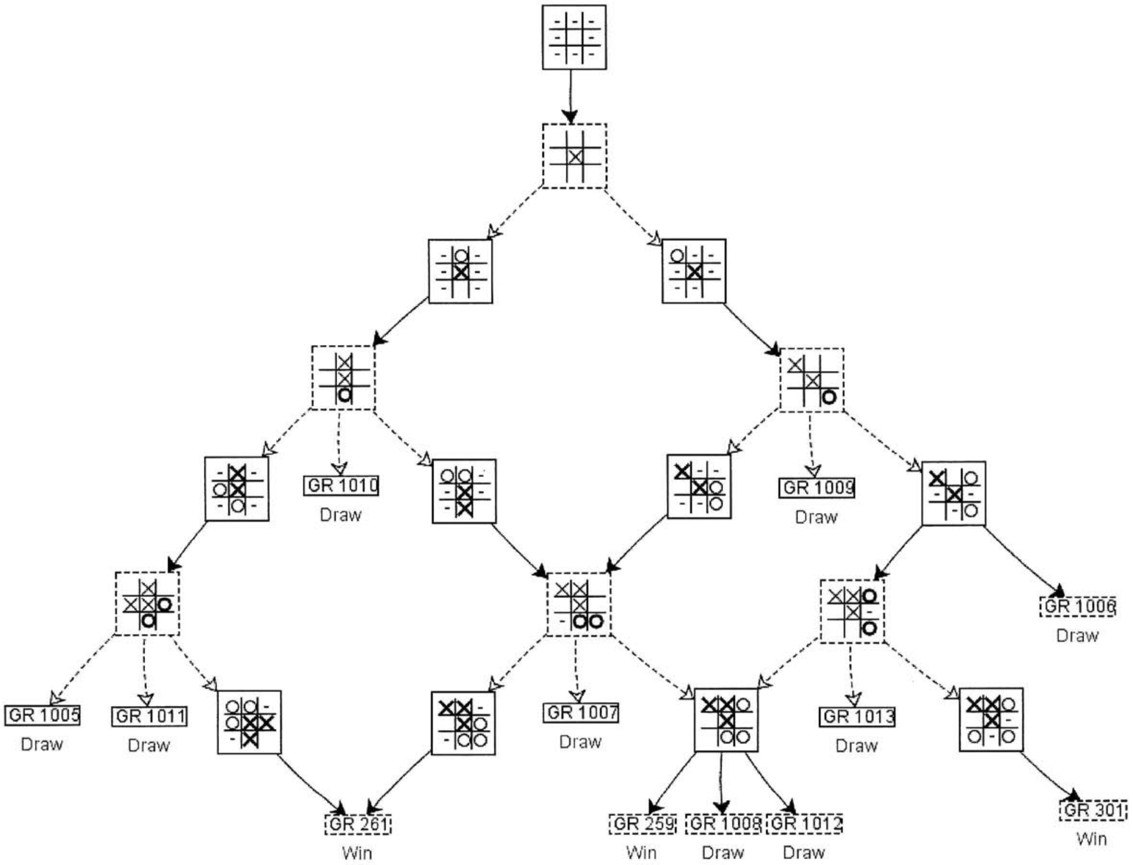


Figure 5-6: Selective application of the pattern grouping algorithms reveals an optimal lose tic-tac-toe strategy in which X reflects O's moves through the center and a strategy that allows X to exploit O's mistakes by forming a 2x2 square of X's. I created this figure by applying the first five grouping operations listed in Table 5.1 and then modifying all subsequent applications of the one-move-to-a-goal grouping algorithm to collapse only paths to a win for O, leaving paths to a win for X untouched.

5.2 The Equivalence Class Principle Exposes Visual Concepts

Applying the equivalence class principle to tic-tac-toe exposes visual concepts that can be used to succinctly describe optimal tactics. For instance, collapsed subtrees labeled “one move to a win for X, X moves next” all have at least one line with two X’s and a blank spot, where X must fill the blank spot to win. Thus, the visual concept “line with two X’s and a blank” can be used to succinctly describe the optimal tactic for the 206 states labeled as “one move to a win for X, X moves next” by the pattern grouping algorithm.[1] This collapsed subtree will be referred to as a {X,X,-} win, when X moves next, and a {O,O,-} win, when O moves next, where the {} notation denotes an unordered set of marks and the ’-’ denotes a blank spot.

The same visual concept can also describe the collapsed subtrees labeled “forced avoidance of a one-move win for X, O moves next.” The 74 instances of this collapsed subtree feature exactly one line with two X’s and a blank spot, but this time O must block the blank spot to keep X from winning. This concept will be referred to as a {X,X,-} avoidance.

Table 5.3 shows how the equivalence class principle starts to expose more complex concepts as the distance from the winning and losing states increases. In these collapsed subtrees, X creates two {X,X,-} lines or O creates two {O,O,-} lines so that their opponent cannot block both lines in one turn. Then, X or O completes whichever line has not been blocked during their next move. This type of fork will be referred to as a {-,X,X,X,-} or {-,O,O,O,-} fork. More complex tactics build on the one-move-to-a-win, avoidance, and fork concepts described above, so the names assigned to these three types of concepts are summarized in Figure 5-7.

Tables 5.4 to 5.9 show all of the exemplar-based concepts collected by the grouping algorithms. Most one-move-to-a-goal, fork, and avoidance collapsed subtrees yield

[1] Tic-tac-toe may contain more than 206 unique states that contain a line with two X’s and a blank spot, but the grouping algorithm only labels states that it believes may be reachable via computationally-limited perfect play at processing time. Thus, if a path through the state space passes through several states where X can win in one move, the states further along the path may not be reachable if X takes the first opportunity it has to win.

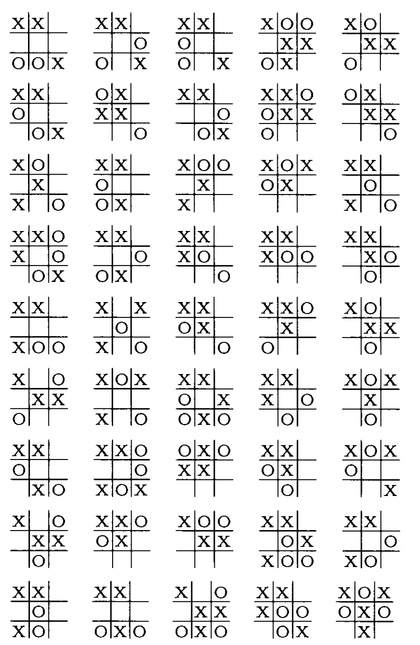
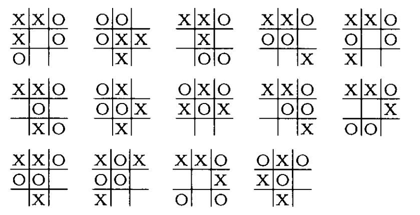
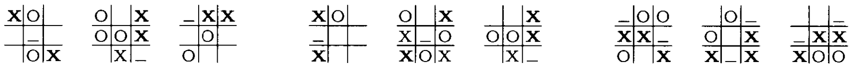
Annotation	Exemplar-Based Concept	Redescribed Concept
Fork to a one-move win for X, O moves next		Two lines that both contain two X's and a blank spot and intersect at an X. On its next turn, X must complete one of the lines.
Fork to a one-move win for O, X moves next		Two lines that both contain two O's and a blank spot and intersect at an O. On its next turn, O must complete one of the lines.

Table 5.3: Using the equivalence class principle to identify strategically-important concepts for collapsed subtrees with the label "fork to a one-move win." The second column contains all subtree roots identified by the fork grouping algorithm. Each board diagram represents all rotations and reflections of the board shown.



(a) {X,X,-} Win (b) {X,X,-} Avoidance (c) {-,X,X,X,-} Fork

Figure 5-7: Labels for some common tic-tac-toe concepts. (a) Roots from one-move-to-a-win subtrees where X moves next. (b) Roots from forced avoidances of a one-move win for X, where O moves next. (c) Roots from forks leading to a one-move win for X, where O moves next. These concepts will be referred to as {X,X,-} wins, {X,X,-} avoidances, and {-,X,X,X,-} forks, respectively.

visual concepts that provide succinct explanations of winning tactics. These visual concepts can be used to define visual routines such as "find the {X,X,-} line and fill the blank spot during your next turn." As the number of moves from the goal increases, the concepts become more and more complex, and the distinctions between collapsed subtree types, such as separating forced avoidances from multi-choice avoidances and separating forced avoidance roots from decision point roots and fixed choice roots, often provide insight into how the collapsed subtrees lead to a win or a loss.

Sometimes the concepts become complex enough and the number of exemplars becomes small enough that the exemplars are better described as special cases. This increase in visual complexity correlates with an increase in the computational processing required to identify the corresponding set of states in the state space. In other words, states further from a win require more processing to identify, indicating that information about how far such states are from a win is more implicitly encoded within the state space, and this implicit encoding in the state space correlates with the presence of more complex visual concepts within the board diagram.

While some collapsed subtrees, such as most of the ones listed in Tables 5-2i and 5.9, do not yield obvious visual patterns, more often than not, the collapsed subtrees created via the equivalence class principle do aid in the construction of visual concepts that can be used to describe what move a player should make next. This supports the hypothesis that the equivalence class principle is a useful inductive bias.

The pattern grouping algorithms and the equivalence class principle also aid in the identification of optimal tactics for lose tic-tac-toe. Figure 5.10 shows examples of roots and concepts obtained when the same analysis is performed on lose tic-tac-toe. The fact that the technique generalizes to lose tic-tac-toe provides further support for the equivalence class principle.

In addition to tactically-significant concepts, the pattern grouping algorithms and equivalence class principle reveal interesting tic-tac-toe trivia. For instance, only two symmetrically-distinct states (listed in row two of Table 5.9) lead to a draw regardless of how the players play. Similarly, only one symmetrically-distinct tic-tac-toe state leads to a win for O in three moves (see row three of Table 5.7) regardless of O's next move, but this state is unlikely to occur in an actual game.

To use the tactics described in the tables, one must be sure to recognize when a state matches more than one concept. For example, when it is X's turn in tic-tac-toe and X has the opportunity to create a {-,X,X,X,-} fork, X must make sure that the board does not also contain a {O,O,-} line that it must block by moving to a different position. Players must always pay attention to the visual concept on the board that corresponds to the shortest path to a goal and responds to that concept.

In this chapter, I have shown that the pattern grouping algorithms and equivalence class principle enable the computer-aided rediscovery of tactics and related visual concepts in tic-tac-toe and lose tic-tac-toe. In the next chapter, I show how the algorithms reveal visual concepts and tactics in games with tens of millions of states.

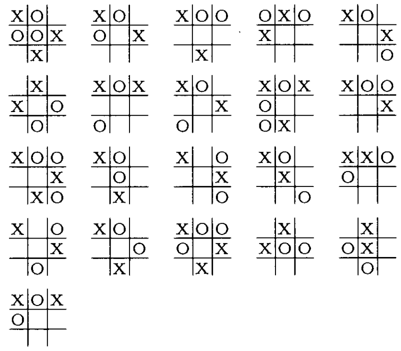
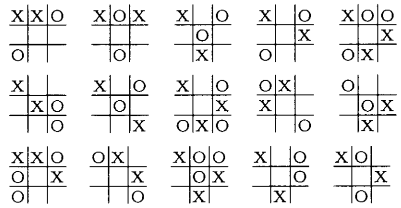
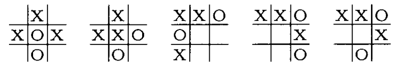
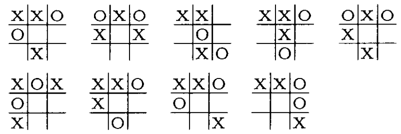
Annotation	Exemplar-Based Concept	Redescribed Concept
Two moves to a win for X, decision point root, X moves next		Two {-,-,X} lines that intersect at a blank spot. X must fill the intersection point on its next turn to create a {-,X,X,X,-} fork.
Two moves to a win for X, forced avoidance root, X moves next		Two {-,-,X} lines and a {O,O,-} line that intersect at a blank spot. X must fill the intersection point on its next turn to block O from winning and to create a {-,X,X,X,-} fork.
Two moves to a win for O, decision point root, O moves next		Two {-,-,O} lines that intersect at a blank spot. O must fill the intersection point on its next turn to create a {-,O,O,O,-} fork.
Two moves to a win for O, forced avoidance root, O moves next		Two {-,-,O} lines and a {X,X,-} line that intersect at a blank spot. O must fill the intersection point on its next turn to block X from winning and to create a {-,O,O,O,-} fork.

Table 5.4: Visual concepts for states that are two moves from a win or loss.

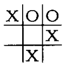
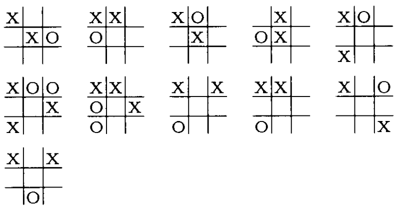
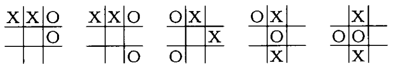
Annotation	Exemplar-Based Concept	Redescribed Concept
Fork to a 2 move win for X, O moves next		Two sets of intersecting {-,-,X} lines. Regardless of where O moves, X can create a {-,X,X,X,-} fork on the next turn.
Fork with a forced avoidance root to a 2 move win for X, O moves next		A {X,X,-} line that forces O to move to a position that allows X to create a {-,X,X,X,-} fork during its next turn.
Fork with a forced avoidance root to a 2 move win for O, X moves next		A {O,O,-} line that forces X to move to a position that allows O to create a {-,O,O,O,-} fork during its next turn.

Table 5.5: Visual concepts for collapsed subtrees that fork to a win or loss in two moves.

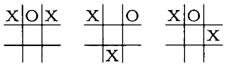
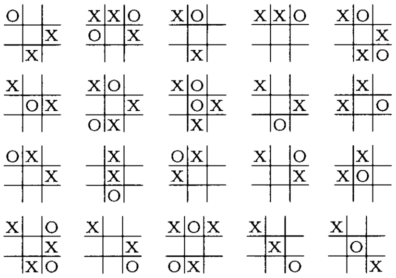
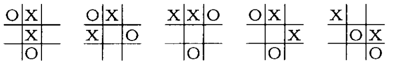
Annotation	Exemplar-Based Concept	Redescribed Concept
Forced avoidance of a two-move win for X, O moves next		X has at least one set of intersecting {-,-,X} lines that could be used to form a {-,X,X,X,-} fork. O has one {-,-,O} with one blank spot at a position that will not enable X {-,X,X,X,-} fork. O must form a {O,O,-} line that forces X to move to a position that does not form a {-,X,X,X,-} fork.
Multi-choice avoidance of a two-move win for X, O moves next		X has at least one set of intersecting {-,-,X} lines that could be used to form a {-,X,X,X,-} fork. O must either create a {O,O,-} line that forces X to move to a position where it cannot create a {-,X,X,X,-} fork or, if X only has one set of intersecting {-,-,X} lines, O must block the position that allows X to create a {-,X,X,X,-} fork.
Multi-choice avoidance of a two-move win for O, X moves next		Same as above, with X blocking O from creating a {-,X,X,X,-}. Visually, these states either contain horizontal or vertical line with the sequence "XXO" or a two X's that are not part of a {X,X,-} line.

Table 5.6: Avoidances of two-move paths to a win or loss. The second column contains all subtree roots found by the avoidance grouping algorithm, but the algorithm only identifies roots that are reachable, given that each player can anticipate its opponent's next two moves and plays perfectly based on this limited computational ability.

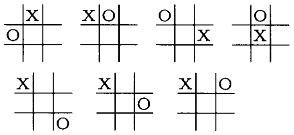
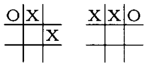
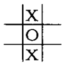
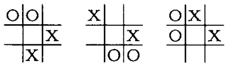
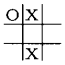
Annotation	Exemplar-Based Concept	Redescribed Concept
3 moves to a win for X, X moves next		X has a {-,-,X} line that can be used to force O into a position where X can create a {-,X,X,X,-} fork on its next move.
3 moves to a win for O, O moves next		O has a {-,-,O} line that can be used to force X into a position where X cannot interfere with O creating a {-,O,O,O,-} fork on O's next move.
3 moves to a win for O regardless of O's next move, O moves next		OXO on a horizontal or vertical line through the center. Wherever O moves on its next turn, O will be able to create a {-,O,O,O,-} fork in two turns.
Forced avoidance leads to a 3 move win for X, X moves next		A {O,O,-} line forces X to move to a position where it can create a {-,X,X,X,-} fork on its next move without interference from O. O cannot interfere either because X creates a {X,X,-} line that forces O's next move, or because X moves to a position that creates two sets of intersecting {-,-,X} lines (which creates two positions where X can form a {-,X,X,X,-} fork.
Forced avoidance leads to a 3 move win for O, O moves next		A {X,X,-} line forces O to create a {O,O,-} that forces X to create another {X,X,-} line that forces O to create a {-,O,O,O,-} fork.

Table 5.7: Three-move paths to a win or loss. Again, each board pictured represents the set of boards that can be obtained via rotations and reflections. Thus, the single board pictured on the third row represents the boards with XOX in either a horizontal or vertical line through the center.

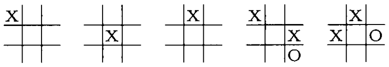
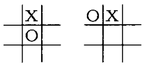
Annotation	Exemplar-Based Concept	Redescribed Concept
Avoidance of a 3-move win for X, O moves next		No obvious pattern, just the set of states where O can avoid creating the states in Table 5.7.
Avoidance of a 3-move win for O, X moves next		No obvious pattern, just the set of states where X can avoid creating the states in Table 5.7.

Table 5.8: Avoidances of three-move paths to a win or loss. The board with only an X in the top corner is the only collapsed subtree that is a forced avoidance, because O can only move in the center if it wants to force a draw. At this level, it becomes more feasible to remember optimal move patterns for particular states. For example, if X moves in the center, O must move to a corner to force a draw. Figure 5-2i, the segmented state space diagram obtained after grouping avoidances of three-move paths, shows the optimal moves for each state in column 2.

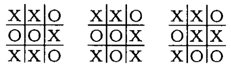
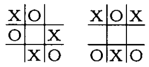
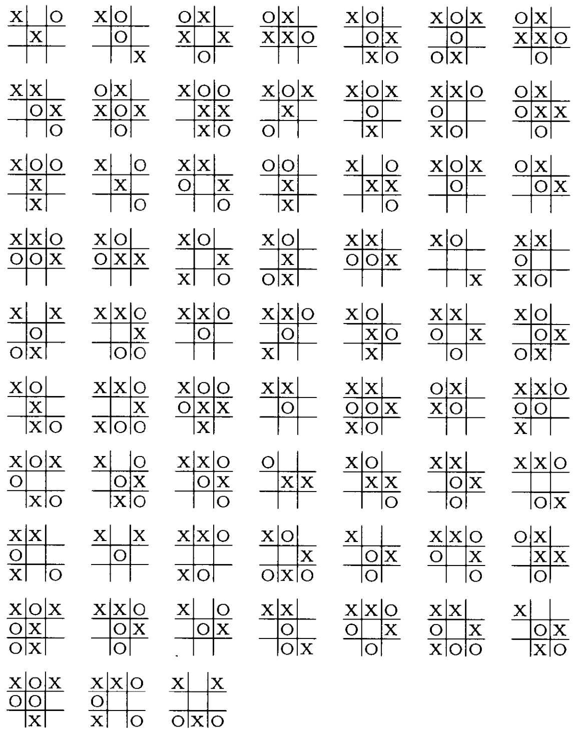
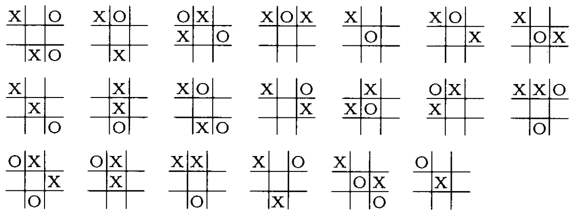
Annotation	Exemplar-Based Concept	Redescribed Concept
Draw states		No lines with three in a row.
Draw tree, no lookahead		Boards that block all but one line with an X and O.
Draw tree, one move lookahead		No obvious pattern.
Draw tree, two move lookahead		No obvious pattern

Table 5.9: While many of the one-move-to-goal, fork, and avoidance collapsed subtrees exposed strategically-important visual concepts, the roots of the draw trees generally do not exhibit an obvious pattern. The draw states in the first row are included for reference. The states in the second row always lead to a draw because players must take turns filling the last open line.

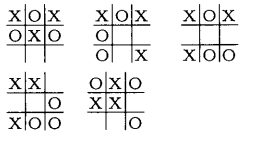
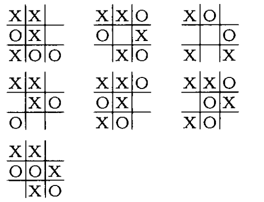
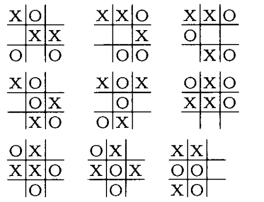
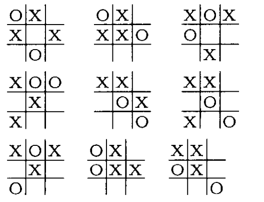
Annotation	Exemplar-Based Concept	Redescribed Concept
Forced avoidance of an immediate win for O, X moves next		Only one blank spot on the board that is not part of a {X,X,-} line. X must move into the blank spot to avoid losing.
One move to a win for O, O moves next		Only one blank spot on the board that is not part of a {X,X,-} line. O must move into this spot to force X to make a losing move.
Forced avoidance of a one-move win for O, X moves next		Boards contain three blank spots. One that is part of a {X,X,-} line, one that is part of an {O,O,-} line, and one that is at the intersection of two {X,O,-} lines. X must move into the intersection of the {X,O,-} lines to avoid losing on its next turn.
Two moves to a win for O, O moves next		Boards contain four blank spots. Three of the blank spots are either in a {X,X,-} line and a {X,-,-} line that intersect at an X or at the three intersections of three {X,-,-} lines. Thus, X can only fill one blank spot without creating a {-,X,X,X,-} fork from tic-tac-toe. O must block this blank spot to force X to create a {-,X,X,X,-} fork.

Table 5.10: Examples of strategic collapsed subtrees and concepts in lose tic-tac-toe. Unlike the tic-tac-toe tables, some of the rows in this table contain only a subset of the exemplar-based concepts identified by the grouping algorithms to give the reader a sense of what concepts the grouping algorithms expose.

Chapter 6

Extracting Strategic Information from Five Men's Morris and Six Men's Morris

To test their generality, I have applied the pattern grouping algorithms to two games with moderately large state spaces: five men's morris, with 1.8×10^7 states, and six men's morris, with 4.2×10^7 states. For both games, the pattern grouping algorithms identify strategically-important sets of states, identify the optimal defensive strategy, and identify the subset of the original state space that is reachable during perfect play. The pattern grouping algorithms reveal that both games are draw games because players can avoid parts of the state space that allow their opponent to win, causing the players to circle around in a reduced portion of the state space. By applying the equivalence class principle, I also rediscovered visual concepts and used them to succinctly describe both offensive and defensive tactics, providing a compressed representation of information from the state space.

Five and six men's morris are two variations on the same game, and both are played on the board shown in Figure 6-1. In the first phase of play, players place one piece on the board during each turn until each player has placed either five pieces for five men's morris or six pieces for six men's morris. I assume that black plays first. After pieces have been placed, players start sliding pieces to any adjacent empty

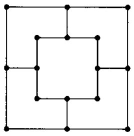


Figure 6-1: Board used to play five and six men's morris. Pieces are played on the locations marked with small dots.

intersection. During either phase of the game, if a player positions three of her pieces along the same line, thus creating what is referred to as a *mill*, the player can remove one of her opponent's pieces. If her opponent's pieces also form a mill, she must remove her opponent's pieces that are not in the mill before removing a piece that will break her opponent's mill. A player wins by either removing all but two of her opponent's pieces from the board or by blocking her opponent's pieces so that her opponent cannot make any sliding moves.

A popular variation of the game allows players to "fly" or jump their pieces to any open position on the board when they only have three pieces left, but I have not used this variation. Morris game enthusiasts also disagree about whether a player should remove one or two of her opponent's pieces when the player creates two mills at once (this can happen when a player places a piece in one of the corners). In this analysis, I assume that players only remove one piece, regardless of whether they form a single or a double mill.

Unlike the tic-tac-toe games, the morris games do not have any draw end states, so the draw grouping algorithm cannot be applied to these games.[1] Because of this, I used the modified grouping algorithm sequence shown in Table 6.1. Essentially, every recursive step simply skips the draw tree grouping phase. Because the morris games can be won via a piece capture or via blocking all of the opponent's pieces, the pattern grouping algorithms label paths to a win based on what type of win the paths lead to. Like the tic-tac-toe games, the morris games do exhibit symmetry, but in this case, grouping all symmetric states becomes confusing during the sliding move phase of the game. Thus, the pattern grouping algorithms have been applied to the

[1] If players agreed on a move limit, the states could be modified to incorporate the number of moves made so far. Then, the draw grouping algorithm could be applied to the modified state space.

Grouping Number	Algorithm(s)	Sample Annotation
1	One-move-to-goal	One move to a win, goal-setter moves next
2	Fork and Avoidance	Fork to or avoidance of a 1-move win, opponent moves next
3	One-move-to-goal	Two moves to a win, goal-setter moves next
4	Fork and Avoidance	Fork to or avoidance of a 2-move win, opponent moves next
$2N-1$	One-move-to-goal	N moves to a win, goal-setter moves next
$2N$	Fork and Avoidance	Fork to or avoidance of an N-move win, opponent moves next

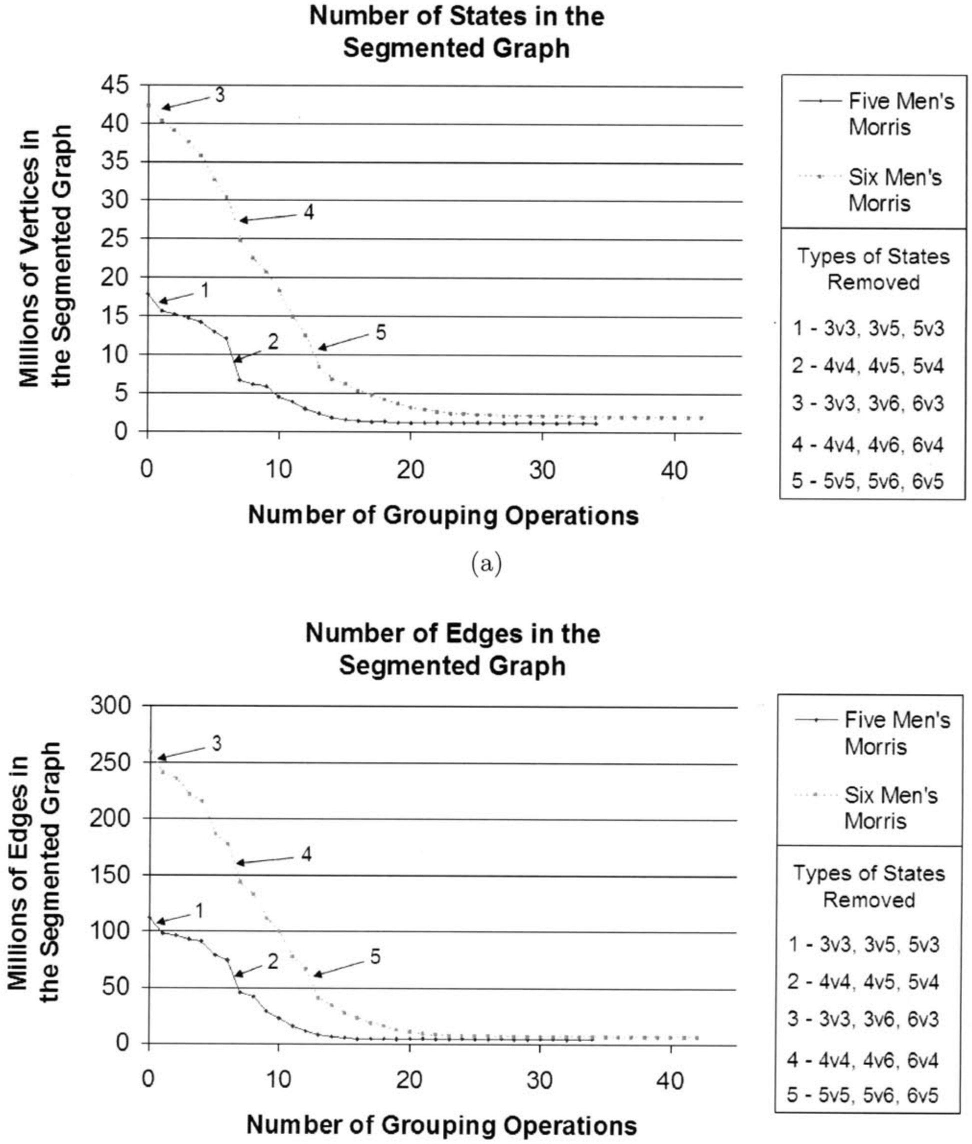
Table 6.1: The sequence of pattern grouping algorithms used to analyze the morris games. Grouping continues until all top-level collapsed subtrees are avoidance collapsed subtrees (see Section 3.2 for more details).

complete state space instead of a version that accounts for symmetry.

6.1 Segmentation of the Morris State Spaces Exposes High-Level Structure and Optimal Defensive Strategies

Figure 6-2 shows how the state spaces collapse as the pattern grouping algorithms are applied. Segmentation causes the number of states to decrease by more than an order of magnitude for both games, reducing five men's morris from 18 million states to 1 million states and reducing six men's morris from 42 million states to 2 million states. The states that remain in the state space are all reachable via perfect play, and as long as players remain in this part of the state space, the players can force a draw.

Only certain types of states are reachable via perfect play (see Table 6.2 for a summary). For instance, the vast majority of 3v3 sliding move states, i.e., sliding



(a)

(b)

Figure 6-2: The number of states and edges in the segmented state space after each pattern grouping operation. The arrows highlight pattern grouping operations where a large number of states become unreachable, given a certain level of computationally-limited perfect play. The boxes to the right show what types of endgame states become unreachable, where AvB indicates the number of black pieces on the board (A) versus the number of white pieces on the board (B). Other types of states may become unreachable at these points as well, but not in significant quantities.

	Number of Reachable Endgame States			
State Type	Five Men's Morris		Six Men's Morris	
	Initial	Final	Initial	Final
3vN, Nv3, 4v4	6.6M	0	10M	0
4v5, 5v4	5.4M	36K	5.8M	31K
5v5	3.0M	520K	4.0M	7.4K
4v6, 6v4	-	-	6.3M	0
5v6, 6v5	-	-	6.9M	130K
6v6	-	-	1.8M	440K

Table 6.2: A comparison of the number of endgame states in the original state space versus the number reachable via perfect play. AvB refers to the number of pieces on the board, where A is the number of black pieces and B is the number of white pieces. Most states where a player has four or fewer pieces on the board are not reachable during perfect play. For the state types that do remain reachable, the number of reachable states is one to three orders of magnitude smaller than the number of states in the original state space.

move states with three black pieces and three white pieces on the board, become unreachable early in the segmentation process, indicating that most of these states are not reachable when players have even limited lookahead capabilities. This suggests why the flying-move variation of the games is popular. Without flying, most 3v3 states will never occur in a normal game, but with flying, it becomes more difficult for a player to force a win from a 3v3 state, so more of the 3v3 state space becomes reachable during a normal game, making the game more challenging.

The final segmentations contain only unprocessed states, forced avoidance collapsed subtrees, and multi-choice avoidance collapsed subtrees. The avoidance collapsed subtrees show what moves players need to make to force a draw, providing players with the perfect defensive strategy. In five men's morris, players must avoid paths that lead to a win in anywhere from 1 through 17 moves. Most short paths lead to a blocked move win, while longer paths lead to a win via piece capture. Of the paths that force a win via piece capture, the longest one requires 15 moves. In six men's morris, players must avoid even longer paths, paths that lead to wins after up to 21 moves of perfect play.

6.2 Equivalence Class Principle Exposes Visual Concepts and Symmetries

6.2.1 Selecting a Set of Collapsed Subtrees for Further Analysis

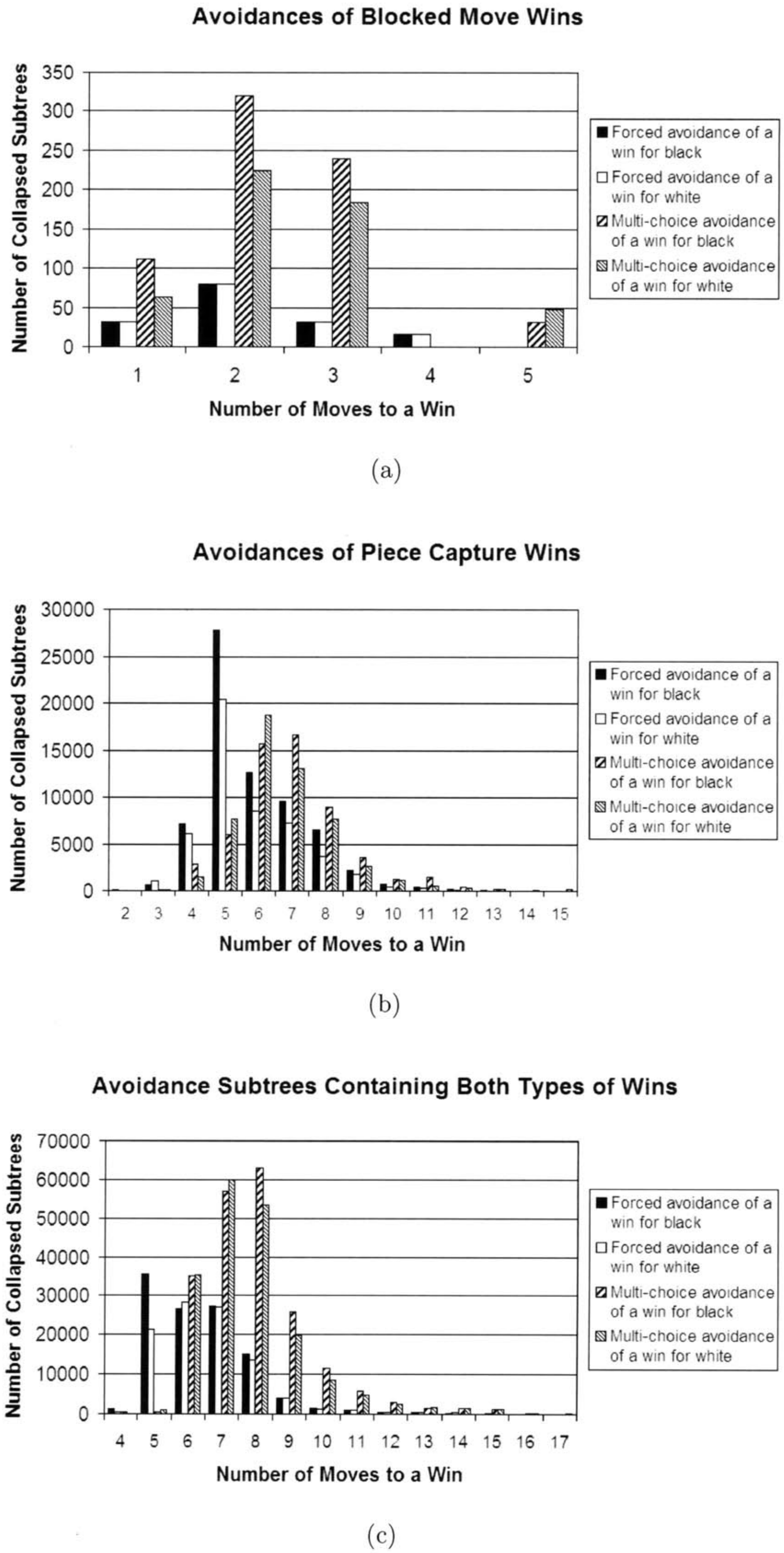
To test whether the equivalence class principle exposes strategically-useful concepts, I selected a subset of the avoidance subtrees identified within five men's morris. The final segmentation of the state space contains 839,912 avoidance subtrees, which can be subdivided based on whether the avoidance is forced or multi-choice, what kind of win is being avoided, which player would win, and the number of moves to the win. Based on the kind of win, the collapsed subtrees can be separated into three groups: collapsed subtrees that contain only paths that lead to a piece capture win, collapsed subtrees that contain only paths that lead to a blocked move win, and collapsed subtrees that contain both types of paths. Figure 6-3 shows how the number of collapsed subtrees depend on win type, avoidance type, winner, and number of moves to the win. The collapsed subtrees can be separated into 119 groups based on these four features.

The analysis in the remainder of this chapter will focus on forced avoidance collapsed subtrees that lead to piece capture wins for white. This subset, which contains 49,704 collapsed subtrees (6% of the subtrees), was selected for the following reasons. As shown in Figure 6-3, most of the blocked move collapsed subtrees have relatively short paths to a win, while the collapsed subtrees containing piece capture wins or both types of wins display a larger range of path lengths. As explained shortly, the tactics used to achieve or avoid a win play an important part in the analysis of the collapsed subtrees. Because the tactics displayed within collapsed subtrees that contain both types of wins probably depend on the tactics used in subtrees with only piece capture wins, the piece capture wins were selected for the first stage of analysis. The decision to focus on white versus black was fairly arbitrary, because in the sliding move portion of the game, the tactics for both sides should be the same. Based on the analysis of the tic-tac-toe games, multi-choice avoidances often exhibit the same tactics as forced choice avoidances, so the forced choice avoidances were chosen for simplicity. Analyzing the single move necessary to thwart a win is simpler than analyzing the set of moves necessary to achieve the same purpose.

6.2.2 Preliminary Analysis of the Forced Avoidance Collapsed Subtrees Reveals Inside-Out Symmetry

The remainder of this analysis focuses on the 49,704 collapsed subtrees that describe forced avoidances of piece capture wins for white. Because these collapsed subtrees come from the final segmentation of the five men's morris state space, each subtree root is a state that is reachable during perfect play. Analyzing how well this set of roots exposes strategically-significant concepts in the visual space will provide data on how well the equivalence class principle applies to the morris games when used in conjunction with the pattern grouping algorithms from Section 3.1.

These collapsed subtrees can be separated into 12 categories based on the number of moves white must make to win, which ranges from 3 to 14. Examining the roots for the forced avoidances of 13- and 14-move paths shows that the morris games exhibit a special type of symmetry called inside-out symmetry (see Figure 6-4). The inside and outside squares of the board can switch positions without affecting the outcome of the game, because the abilities to form mills, to slide pieces, and to block an opponent's moves are not affected by this transformation. By accounting for rotational and reflectional symmetry, the number of distinct collapsed subtrees can generally be reduced by a factor of eight. Accounting for inside-out symmetry reduces the number of subtrees by an additional factor of two, yielding the collapsed subtree counts shown in Table 6.3.



(a)

(b)

(c)

Figure 6-3: Number of forced avoidance collapsed subtrees, organized by category. The presence of a path length on the x axis indicates that at least one of the bars has a non-zero value.

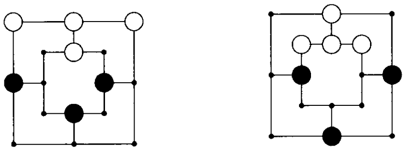


Figure 6-4: The roots of all sixteen of the collapsed subtrees with the label "forced avoidance of a 14-move piece-capture win for white" have board diagrams that are rotationally or reflectively symmetric to the board diagrams pictured above. These two board diagrams display a special type of symmetry called "inside-out" symmetry, which applies to the morris games. The first board can be transformed into the second by flipping the inside and outside squares.

Path Length	Number of Subtrees	Number of Subtrees, Accounting for Simple Symmetry	Number of Subtrees, Accounting for All Symmetries
3	1,008	128	64
4	6,088	762	381
5	20,464	2,564	1,282
6	8,496	1,064	532
7	7,248	906	453
8	3,696	464	232
9	1,776	224	112
10	416	52	26
11	368	46	23
12	96	12	6
13	32	4	2
14	16	2	1
Total	49,704	6,228	3,114

Table 6.3: Number of collapsed subtrees for each of the 12 types of forced avoidances where black must avoid a piece capture win for white. Path length indicates the number of moves white must make to win. Simple symmetry refers to all rotations and reflections, while all symmetry accounts for inside-out transformations as well.

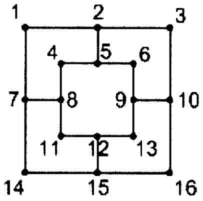


Figure 6-5: Numbered morris board used to define absolute action sequences.

6.2.3 Procedure for Identifying Visual Concepts and Using Them to Redescribe Tactics

Ideally, the forced avoidance categories would separate the collapsed subtrees based on both the tactic white uses to force a win and the tactic black uses to block white from winning. To test whether this occurs, I developed the following procedure for using visual concepts to succinctly describe tactics and *visual patterns* on board diagrams. Here, a visual pattern refers to one visual concept or a combination of visual concepts on a board, such as a white mill and a black one-from-a-mill on the same board. This procedure involves developing *redescribed action sequences*, which are action sequences expressed in terms of visual concepts.

Redescribed action sequences build on *absolute action sequences*, a type of action sequence introduced in Lock and Epstein (2004). An absolute action sequence is defined using the board numbering scheme shown in Figure 6-5. Table 6.4 shows two absolute action sequences that lead to piece captures for white. When an action sequence describes a path where white can force a win, every opportunity for black to move will fit the description "B anywhere." To eliminate this redundancy, I will use <> brackets to denote *full action sequences*, which list both black's and white's moves, and <<>> brackets to denote *abbreviated action sequences*, which list only white's moves, with the implicit assumption that a "B anywhere" move should be inserted between each pair of white moves in the sequence.

Instead of defining moves relative to locations on the board, redescribed action sequences define them relative to visual concepts, such as mills or double mills. One redescribed action sequence can describe a set of absolute action sequences. The redescribed action sequence <W slides out of its mill, B anywhere, W slides back into

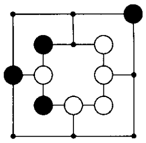
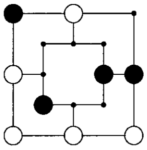
Context	Absolute Action Sequence	Abbreviated Absolute Action Sequence
	<W slides 6 to 5, B anywhere, W slides 5 to 6 and captures 11>	<<W slides 6 to 5, W slides 5 to 6 and captures 11>>
	<W slides 15 to 12, B anywhere, W slides 12 to 15 and captures 1>	<<W slides 15 to 12, W slides 12 to 15 and captures 1>>

Table 6.4: Two examples of absolute action sequences. The first column describes the *context*, or start state, and the next two columns describe the same action sequence using the full and abbreviated notations, respectively. Each action sequence results in white capturing one of black's pieces.

its mill and captures the B blocking its one-from an L-shaped mill> uses concepts from Table 6.5 to describe both of the absolute action sequences from Table 6.4 (as well as a number of other absolute action sequences).

Redescribed action sequences can be used to identify tactics within forced avoidance collapsed subtrees by applying the following procedure. Starting with a set of N-move forced avoidance subtrees, identify all N-move paths that lead to a win for white and express them as absolute action sequences. Identify the visual concepts that can be used to compress the set of absolute action sequences into one or more redescribed action sequences. Use the visual concepts to redescribe the absolute action sequences to obtain a compressed version of the tactics white uses to win. Repeat this procedure to obtain redescribed action sequences for the moves black makes to prevent white from winning.

The redescribed action sequences are used to classify the board diagrams that correspond to collapsed subtree roots. (The term *root board diagram* will be used to refer to the board diagram of a collapsed subtree root.) Each root board diagram is paired with the redescribed action sequences used within its collapsed subtree. More specifically, each root board diagram is paired with two redescribed action sequences: one that depicts white's winning tactic and one that depicts black's avoidance tactic.

Root board diagrams that share the same set of tactics belong to the same *tactical category.* Visual patterns shared by the root board diagrams in a tactical category can be described using the visual concepts from the redescribed action sequences, creating what is called the *redescribed context* for the action sequences.

Identifying the visual concepts used to form the redescribed action sequences is the least refined step of this process. Often, the board diagrams for the subtree roots (obtained using the equivalence class principle) contain visual patterns that form the basis for these concepts. The players' moves relative to fixed pieces on the board also provide visual cues. The issue of identifying these visual concepts will be addressed again in the discussion section.

6.2.4 Analysis: Using Visual Concepts to Identify Tactical Categories

The analysis in this section attempts to quantify how well the 3-move, 4-move, and 5-move forced avoidance categories from Table 6.3 expose tactics and their associated visual concepts. Under ideal conditions, the forced avoidance categories would separate the collapsed subtrees based on both the tactic white uses to force a win and the tactic black uses to keep white from winning. In other words, each forced avoidance category would ideally correspond to a single tactical category. In actuality, each forced avoidance category contains multiple tactical categories.

As a step toward separating the tactical categories, I ran scripts to classify the root board diagrams based on two features: how many pieces have been placed on the board, which determines whether the next move will be a sliding move or a placing move, and how many pieces remain in play. By separating them based on these two features, the scripts classified the root board diagrams into *forced avoidance subcategories.*

I identified tactical categories by applying the procedure outlined in the last section to each of the forced avoidance subcategories. I analyzed the 3-move forced avoidances first, and then proceeded to the 4-move and 5-move forced avoidances in

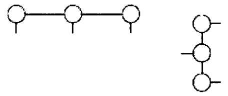
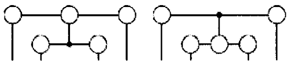
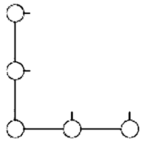
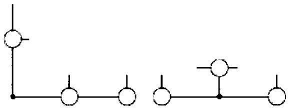
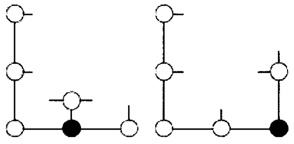
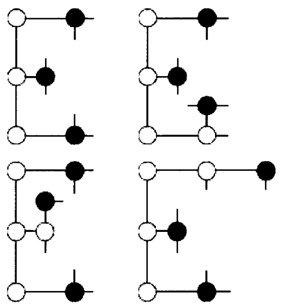
Visual Concept	Examples	Use
Mill		Three in a row. White takes one of black's pieces when the mill is formed. White can then slide a piece out of and back into the mill to take another one of black's pieces after two moves.
Trapezoidal double mill		White can slide its center piece up and down to create a new mill each move. Thus, white can use this piece formation to remove one of black's pieces each move.
L-shaped double mill		Another way to create two mills with five pieces. This structure gives white more flexibility in moving pieces in and out of mills because it is harder for black to block this than for black to block a single mill.
One-from-a-mill		White can slide into a mill in one move.
Black blocking a one-from an L-shaped double mill		White can move in and out of its complete mill to remove the black piece that is blocking its one-from-a-mill. Then, white can create another mill in one move.
Blocked mill		White cannot move in and out of its mill because all adjacent positions are occupied. Any positions that are occupied by a white piece have an adjacent black piece that can reblock the position if the white piece decides to move.

Table 6.5: Visual concepts for sliding move strategies.

that order. As the number of moves increased, the action sequence redescription process became more time consuming, so I only identified tactical categories for the 3, 4, and 5-move sets. These sets account for approximately half of the forced avoidance collapsed subtrees that lead to a piece capture win for white.

As expected, I found that many of the tactics used in the 4-move and 5-move collapsed subtrees build on tactics from the 3-move collapsed subtrees. Table 6.5 shows some of the visual concepts that reoccur in many of the redescribed action sequences and describes how they are used during the endgame when pieces slide from one position to another.

Overall, the forced avoidance subcategories made the action sequence redescription process more manageable to perform by hand, because the subcategories divide the root board diagrams into smaller groups. For the most part, each forced avoidance subcategory contains less than 100 members (accounting for all types of symmetry), although a few subcategories contain 200 or 300 members and one contains 1065 members. These statistics can be compared with the original forced avoidance category sizes from Table 6.3.

Although a forced avoidance subcategory occasionally corresponds to a single tactical category, forced avoidance subcategories are more likely to contain several tactical categories with one or two dominant ones. For instance, of the 20 symmetrically-distinct 4-piece versus 5-piece 3-move forced avoidances shown in Figure 6-6a, 15 of them belong to a dominant tactical category, while the remaining 5 are split between 2 smaller tactical categories with 3 and 2 members, respectively. Similarly, even the very large set of 1065 root board diagrams, in which both players have placed 4 pieces and all pieces remain in play, has a dominant tactical category that contains 1042 of the root board diagrams, while the remaining 23 board diagrams exhibit 5 less common sets of redescribed action sequences and thus belong to 5 smaller categories.

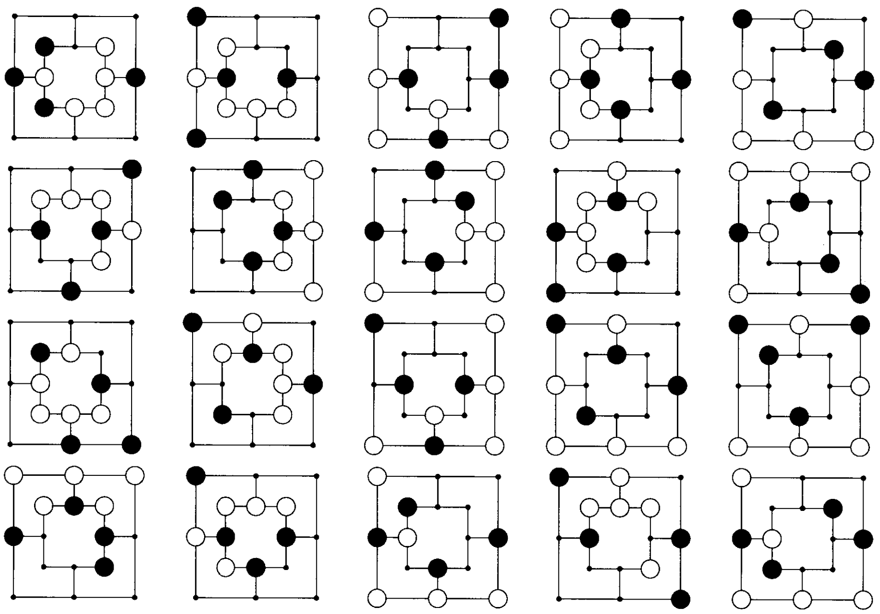
The redescribed action sequences for the root diagrams from Figure 6-6a, which form the 3 tactical categories with 15, 3, and 2 members apiece, are depicted pictorially in Figures 6-6b and 6-6c. The top rows describe the dominant tactical categories, while the bottom rows describe the two-member tactical category. The dominant tactical category applies to board diagrams from Figure 6-6a that contain a blocked one-from an L-shaped mill, where black can block white's intact mill in one move. This is the redescribed context for the dominant tactical category. The redescribed action sequence for white's path to a win is <<W slides out of its mill, W slides back into its mill and captures the B blocking its one-from an L-shaped mill, W slides into its L-shaped mill and captures any piece to win the game>>, and the redescribed action sequence for black's avoidance move is <<B slides to a position that blocks W's mill>>.

Occasionally, a forced avoidance subcategory will contain exactly one tactical category. For instance, the 32 5-piece versus 4-piece 5-move forced avoidances shown in Figure 6-7 all share the same pair of redescribed action sequences. This example describes an interesting strategy in which black can at least temporarily force a draw by refusing to close its mill. As long as black refuses to close its black one-from-a-mill, black prevents white from taking advantage of its white mill, essentially creating a stalemate.

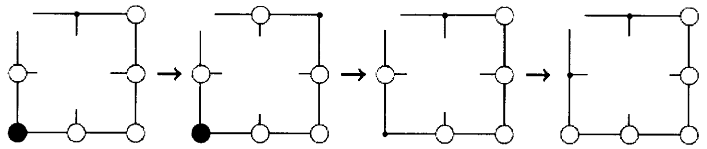
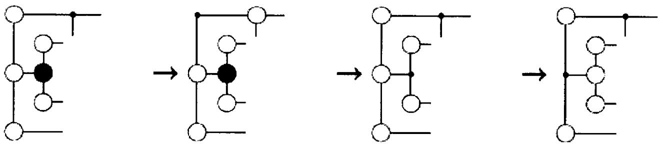
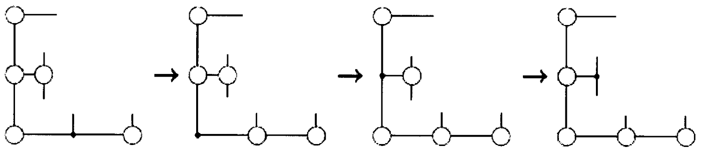
Table 6.6 summarizes the number of tactical categories identified using the 3-, 4-, and 5-move forced avoidance categories. In total, the 27,560 collapsed subtrees form 35 tactical categories, each defined by a different pair of redescribed action sequences, one describing white's path to a win and one describing black's tactic for avoiding white's win. As shown in the previous examples, the root board diagrams in each tactical category share a visual pattern that can be summarized using the concepts from the redescribed action sequences.

6.2.5 Discussion

Gasser created and analyzed endgame databases for nine men's morris, a complex version of five and six men's morris, and showed that nine men's morris is a draw (Gasser, 1996, 1991). It should be noted that Gasser analyzed a version of the game that allowed flying, while I did not. Gasser performed a statistical analysis on the databases and examined some of the longest paths to a win. In his analysis, he remarked that "it is not clear how the realistic positions can be filtered from the

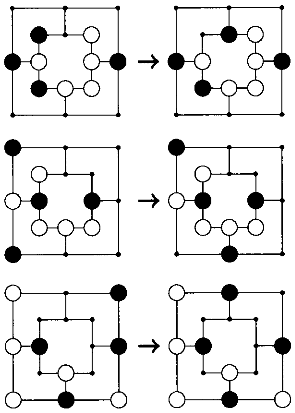
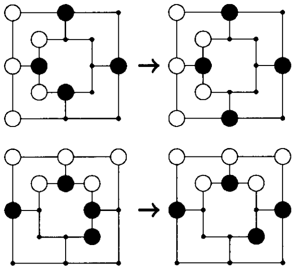
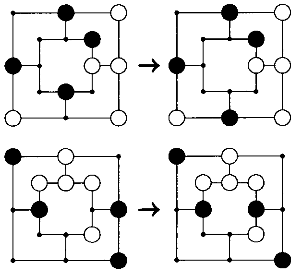


(a) Symmetrically-distinct root board diagrams for the 4-piece versus 5-piece subcategory of forced avoidances that could lead to 3-move wins for white. Black makes the next sliding move, and if black makes a mistake, white can win.

Visual Concept	Abbreviated Redescribed Action Sequence that Leads to a Win for White
Black blocking a one-from-an-L-shaped double mill	
Black blocking a trapezoidal double mill	
Two-from-an-L-shaped double mill	

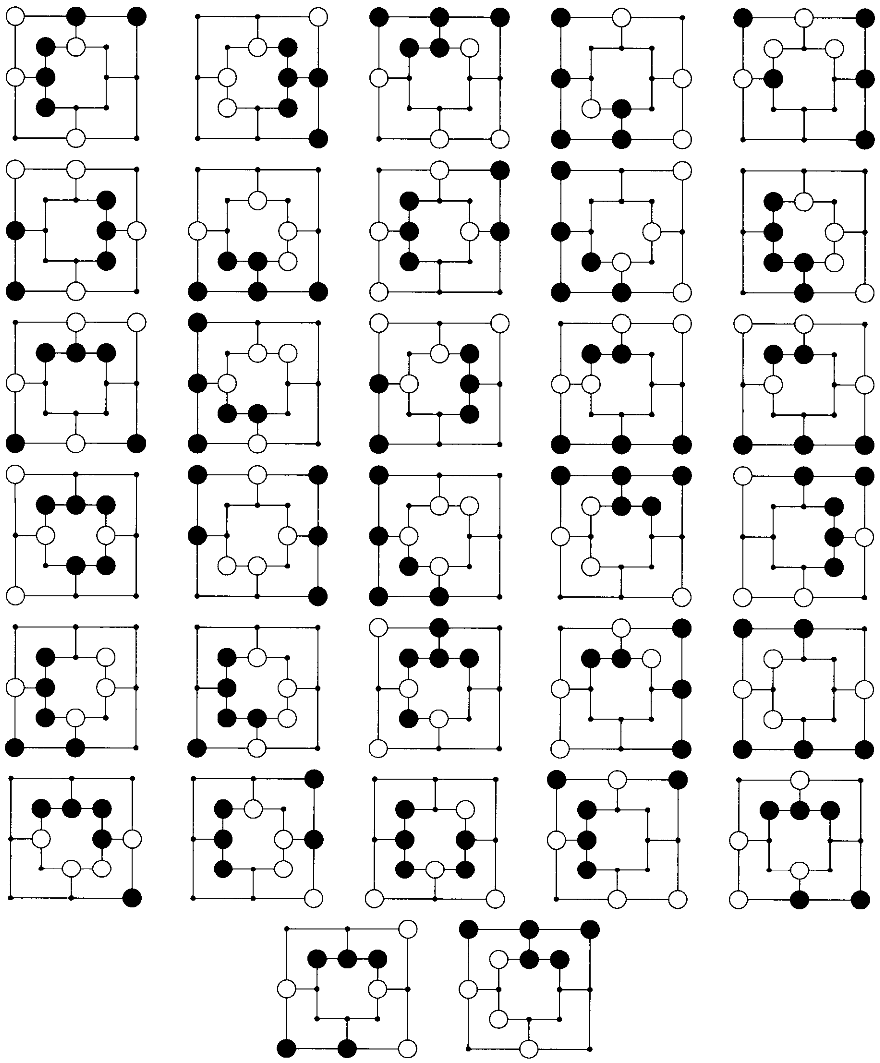
(b) Offensive tactics that white can use to win the game, assuming that black makes a mistake. Each tactic shows only white's moves and highlights how white slides its pieces in and out of instantiations of the visual concepts in the first column. In each case, white captures two black pieces by creating mills during the second and third moves. Black's pieces can be placed anywhere in the hidden part of the board, as long as they are not positioned in a way that would enable black to win in three moves or less.

Figure 6-6: Offensive and defensive tactics for a subcategory of forced avoidances.

Visual Concept	Examples of Black's Defensive Tactic	Description of the Black's Forced Avoidance
Black blocking a one-from-an-L-shaped double mill		Black has one move that will block white's mill.
Black blocking a trapezoidal double mill		Black has one move that will block white's double mill.
Two-from-an-L-shaped double mill		Black has one move that stops white from creating a mill and eventually forming an L-shaped double mill.

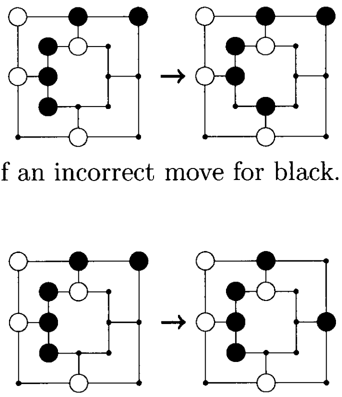
(c) Black's defensive tactics.

Figure 6-6: Offensive and defensive tactics for a subcategory of forced avoidances of 3-move wins for white (continued). (a) The set of 4-piece versus 5-piece forced avoidances where white can win in three moves if black makes a mistake. Black makes the next sliding move. (b-c) White's three offensive tactics and black's three defensive tactics, respectively. Using (row, column) coordinates to label the root board diagrams from part (a), diagrams (1, 4), (2, 2), and (4,1) pair with the tactics shown in the second rows of Tables (b) and (c), diagrams (2,3) and (4,4) pair with the tactics shown in the third rows, and all other diagrams pair with the tactics shown in the top rows.



(a) Board diagrams for the subtree roots of all 5-move forced avoidances with 5 black pieces on the board and 4 white pieces on the board. Black makes the next sliding move.

Figure 6-7: A visual pattern that is easy to identify, given the concepts of a mill and a one-from-a-mill.



(b) An example of an incorrect move for black. Black should not break its mill.

(c) An example of the correct move for black. Here, black selects a move that leaves its mill intact.

Figure 6-7: An example of a visual pattern in the subtree root board diagrams that is easy to identify, given the mill and one-from-a-mill concepts. (a) All of these board diagrams contain a one-from-a-mill for white and a mill for black. Given the concepts of a mill and a one-from-a-mill, this pattern is easy to extract, even though the action sequence leading to a win for white is not obvious. (b) If black starts in one of the states shown in (a) and makes a move that breaks its mill, white can win the game. In this example, white can close its one-from-a-mill during its next turn and capture black's piece at location 12. Then, white can use its mill to capture two more of black's pieces and win the game. (c) Black's optimal avoidance tactic is to keep its mill intact. The resulting state is a forced avoidance for white. If white closes its one-from-a-mill while black's mill is still intact, black can win the game. All of the states in (a) are forced avoidances of a 5-move win for white because black has only one move that does not require breaking its mill.

Path Length	Number of Avoidances	Number of Symmetrically Unique Avoidances	Number of Tactical Categories
3	1008	64	5
4	6088	381	11
5	20464	1282	19
Total	27560	1727	35

Table 6.6: The number of tactical categories identified for each type of forced avoidance. In theory, all important information about the members of a tactical category can be stored using redescribed action sequences and redescribed contexts. This would allow the 27,560 collapsed subtrees to be compressed into 35 trios of redescribed contexts, redescribed offensive tactics, and redescribed defensive tactics.

rest," noted the difficulty of extracting "interesting" positions from the databases, and observed that some optimal play "is clearly beyond human ability." The analysis in this chapter addresses each of these points.

The pattern grouping algorithms provide a mechanism for distinguishing between realistic and unrealistic positions. For example, forcing a win from an arbitrary three-piece versus three-piece state appears to be a daunting task, but in a realistic game, none of these states will ever be reached. The pattern grouping algorithms remove all of these states from the collapsed state space, because once the players reach a three-versus-three state, the game can be won within a move or two. Thus, given even limited lookahead, most three-versus-three states will never occur in a real game.

Gasser analyzed a variation that allowed flying, so more of the three-versus-three states are probably reachable for his version of the game, but the pattern grouping algorithms would still provide a mechanism for distinguishing between realistic and unrealistic states. Realistic states can be filtered by assuming various levels of lookahead. The analysis in this chapter focuses on perfect play with unlimited lookahead, but the pattern grouping algorithms can be used to identify reachable states for any level of lookahead. The likelihood that a state will appear in a real game probably also depends on whether the state appears to move toward a subgoal, such as closing a mill, but filtering states based on whether they are reachable using a realistic degree of lookahead provides an important step toward distinguishing between realistic and unrealistic positions.

This thesis also addresses the issue of identifying "interesting" positions. In this work, an interesting position is one that exhibits instances of visual concepts that can be used to redescribe tactics and remains reachable during perfect play or almost perfect play. Interesting positions are positions within human-interpretable action sequences. These positions not only describe short-term tactics, but may also provide insight into long-term strategies. For instance, the 5-move forced avoidance tactic from Figure 6-7 shows how black can temporarily force a draw by not breaking its mill, while white can continue forcing a draw by not closing its mill. Gasser (1993) noted that forcing a draw in nine men's morris involved cyclical play in which the players could not or would not close a mill, so the 5-move forced avoidance may have revealed a more general strategy for forcing a draw.

Gasser notes that some optimal paths within the nine men's morris state space defy human interpretation. While I do not dispute that some optimal play may be beyond human ability, a significant portion of play may still fall well within the realm of human comprehension. Gasser analyzes many of the longest optimal paths to a win. While these paths may in fact be too difficult for humans to interpret, the analysis of five men's morris suggests that most paths are short to medium in length. The long paths may simply be anomalies.

In five men's morris, all of the short paths analyzed exhibited a human-interpretable tactic. The 1,727 symmetrically-distinct board diagrams associated with 3-, 4-, and 5-move forced avoidances account for more than half of all forced avoidances that lead to a piece capture win for white. Each of these board diagrams fall into one of only 35 tactical categories, thus exhibiting one of only 35 redescribed-context-offensive-tactic-and-defensive-tactic trios. This suggests that a significant portion of the state space does fall within the realm of human comprehension.

In theory, the redescribed action sequences that define tactical categories could be used to compress the information in an endgame database. Assuming that the redescribed action sequences and their redescribed context could be expressed programmatically, visual pattern recognition procedures could identify when a tactical action sequence might apply. When a board diagram appears to match the redescribed context for a particular set of redescribed action sequences, the redescribed action sequences can be applied to quickly verify the match. In the event that a board diagram matches multiple redescribed action sequences, the sequence that leads to a win or loss in the smallest number of moves would take precedence. Encoding game database information in this manner would have the added benefit that states reachable via imperfect play would often lend themselves to the same set of tactics identified by analyzing positions reachable via perfect play.

Storing endgame databases in terms of offensive and defensive action sequences and contexts could compress the databases by several orders of magnitude. Even accounting for symmetry, the number of tactical categories is several orders of magnitude smaller than the number of collapsed subtrees within the 3-, 4-, and 5-move forced avoidance categories. While all avoidance categories may not compress to the same degree, the preliminary results suggest that the overall compression rate would still be significant. In addition, analyses of lose tic-tac-toe and tic-tac-toe suggest that forced avoidance tactics are often special cases of multi-choice avoidance tactics, so many of the multi-choice avoidance categories may share one of the 35 sets of redescribed action sequences identified already.

The major limitation of my approach is the amount of human effort required to identify use-driven concepts and redescribed action sequences. As the number of moves to a win increases, classifying collapsed subtrees based on the optimal winning and forced avoidance tactics becomes more difficult. One way to address this issue would be to intersperse the tactical category identification in the visual space with the application of the pattern grouping algorithms in the state space. This would allow the pattern grouping algorithms to label N+1-move collapsed subtrees based on the tactical action sequences exhibited by their N-move components, which should lead to a better initial separation of the N+1-move tactical categories. Then, analysis in the visual space would would further subdivide the N+1-move collapsed subtrees as needed based on visual concepts and redescribed action sequences. Thus, grouping in the state space would induce classifications in the visual space and vice versa.

Interspersing the tactical category identification with the application of the pattern grouping algorithms provides one step toward automating this approach. Other steps include semi-automated algorithms for converting absolute action sequences into redescribed action sequences and a generative algorithm for identifying useful visual concepts.

The results presented in this chapter suggest that the pattern grouping algorithms and equivalence class principle do a reasonable job of exposing visual concepts in the morris games. Preliminary analysis of the other collapsed subtree categories from both five and six men's morris supports this conclusion. A thorough analysis of this result will be deferred until the final chapter.

Chapter 7

Contributions and Future Work

In this thesis, I have provided evidence that the equivalence class principle can be an effective mechanism for multi-representational concept formation. Based on the semi-automated analysis of five two-player games with state spaces varying in size from fifty-six states to forty-two million states, the pattern grouping algorithms from Chapters 3 and 4 expose high-level state-space structure and optimal strategies. Together, the equivalence class principle and pattern grouping algorithms enable the discovery of use-driven visual concepts, offensive tactics, and defensive tactics in games such as tic-tac-toe and five men's morris. When used in conjuction with the equivalence class principle and pattern grouping algorithms, the action sequence redescription procedure outlined in Chapter 6 provides a mechanism for creating human-interpretable descriptions of tactics. In this chapter, I first explore how my work can be extended and then discuss my contributions and the underlying theoretical questions that I have addressed.

7.1 Areas for Future Exploration

In this section, I describe how my work could be extended to create a general-purpose machine learning algorithm, model human learning, automatically extract human-interpretable tactics from game databases, and expose pattern in various types of networks.

7.1.1 Creating a Multi-Representational Learning Algorithm

While I have explored how the equivalence class principle can expose concepts in games, ascertaining the principle's strengths and limitations will require experimentation in other domains. In future work, the equivalence class principle could be used to create a general-purpose algorithm by combining unsupervised and supervised learning algorithms. For example, given two representations that describe the same set of entities, one with data that could be classified using an unsupervised learning algorithm and one with data that could not, the equivalence class principle would feed the output of the unsupervised learning algorithm into a supervised learning algorithm. The unsupervised learning algorithm would classify entities in the first representation. Then, an application of the equivalence class principle would generate labeled classes in the second representation. A supervised learning algorithm would then use the labeled data to create a classifier and, ideally, identify features in the second representation that correlate with important features in the first representation. The unsupervised learning algorithm applied to the first representation could be replaced by a semi-supervised learning algorithm, depending on the data and the information available. Testing this approach in a variety of domains would help establish the conditions necessary for the equivalent class principle to create connections between representations.

7.1.2 Modeling Human Learning

My motivation for studying the equivalence class principle came from developing a taxonomy of representations and examining how people use multiple representations to learn and perform complex tasks (see Ainsworth, 1999 and de Jong et al., 1998 for reviews on this topic). This effort yielded the insight that when a domain contains multiple representations, the ability to translate between the representations serves several important purposes. Translation between representations facilitates multi-representational problem solving, during which each representation supports operations that would be difficult if not impossible to perform in another representation

(Boshuizen & (Tabachneck-)Schiff, 1998). Translation also disambiguates concepts and facilitates learning in unfamiliar domains (Ainsworth, 1999). (See Appendix B for more information.)

Translation requires building connections between the representations. The equivalence class principle provides a theory that suggests how humans might form these connections. Testing whether the equivalence class principle is a valid model for human learning would require developing a full computational model that incorporates the principle, developing a human-subject experiment that involves building connections between a pair of representations, and showing that the computational model could produce output that matches human subject data.

The procedure for redescribing action sequences from Chapter 6 could also serve as the foundation for a computational model of human learning. If this procedure could be automated, it could become one of the first computational models inspired by Karmiloff-Smith's theory of representational redescription (Karmiloff-Smith, 1995). Again, human-subject experiments would be necessary to establish the model's validity.

7.1.3 Performing Game Analysis

Increasing the Level of Automation

In addition to serving as a testbed for refining the principles of concept formation, the domain of two-player perfect information games provided a rich set of domain-specific research challenges. As I mentioned in Chapter 2, researchers have made numerous attempts to extract human-interpretable strategic information from game databases. In this thesis, I presented a new approach for performing this task, but this approach has the obvious limitation of requiring too much human effort to extract use-driven visual concepts and use them to succinctly describe offensive and defensive tactics.

The next obvious step would be to support an interactive mode of computer-aided discovery to increase the amount of analysis performed by the computer. Expressing tactical patterns as redescribed action sequences is the most time consuming part of the process. Right now, the computer applies all of the pattern grouping algorithms at once and then passes the output to the user for visual analysis. This means that although the user may have developed a redescribed action sequence description for a 5-move path to a win, when the user encounters a 6-move path that builds on the 5-move path, the user must discover this relationship. Because the redescription of the 6-move action sequence often uses the same visual concepts as the redescribed 5-move action sequence, making such dependencies readily available would dramatically speed up the user's discovery process.

An interactive mode of applying the pattern grouping algorithms would allow the user to label collapsed subtrees based on their tactical category and provide an additional degree of automation. Once the user assigns a tactical category to the N^{th}-iteration of collapsed subtrees, the next iteration of pattern grouping algorithms would label the $N + 1^{st}$ set of collapsed subtrees not only based on their attributes in the state space, but also based on the tactical categories attributed to their leaves. Then, applying the equivalence class principle to the output of the $N + 1^{st}$ pattern grouping iteration would classify visual diagrams not just based on the number of moves to the goal, but also based on the tactical categories of the constituent N^{th}-iteration action sequences. This would make it easier for the user to redescribe the action sequences and assign tactical categories to the $N + 1^{st}$ set of board diagrams. This amounts to iteratively using patterns in the state space representation to classify board diagrams in the visual representation, and then using patterns in the visual representation to further classify states in the state space representation.

Once the $N + 1^{st}$ iteration of collapsed subtrees have been labeled with N^{th}-iteration tactical categories, building an algorithm to generate redescribed action sequences and thus assign tactical categories to the $N + 1^{st}$ set of subtrees will be more straightforward. Automating this part of the process would involve providing the learning algorithm with a hypothesis space of visual concepts that could be used to redescribe a set of absolute action sequences. This hypothesis space could be produced using a generative function that creates different piece configurations and weights the piece configurations based on attributes such as size and similarity to concepts already known to be useful within the game of interest. Given a set of absolute action sequences, the algorithm would search for the redescription that could consolidate the largest number of absolute action sequences. Given an N^{th}-iteration redescribed action sequence, creating an $N+1^{st}$-iteration redescribed action sequence would only require redescribing one additional move, thus partitioning the search for an $N+1^{st}$-iteration redescribed action sequence into N+1 one-move subproblems. If necessary, the results could periodically be validated by the user.

Identifying Offensive Strategies in Complex Games

For five and six men's morris, the current pattern grouping algorithms only analyze parts of the state space that lead to a win or loss with perfect play. Because the remaining parts of the state space contain loops, presumably each player should have a strategy that allows them to force a draw. The work on pong hau k'i suggested one approach for identifying offensive strategies that force a draw while remaining in a portion of the state space where the opponent is likely to make a mistake that would cause them to lose. Because of the number of cycles in the morris state spaces, the approach that worked for pong hau k'i is unlikely to succeed in the morris games. Developing pattern grouping algorithms that parse the drawn portion of complex state spaces would be a useful next step. Ideally, such an algorithm would reveal action sequences that would allow a player to force a draw by repeating a short sequence of moves regardless of the opponent's response. Developing pattern grouping algorithms that could extract redescribed action sequences that lead to regions of the state space near where the opponent can lose would be useful as well.

Analyzing Games with Larger State Spaces

As mentioned in Chapter 2 and 6, extracting human-interpretable information in game databases is a challenging task. The results for five and six men's morris suggest that my approach has merit, particularly if it can be automated further. Testing the pattern grouping algorithms and the equivalence class principle on complicated chess endgames, such as the KQKR and KBBKN endgames, would further clarify the strength and weaknesses of this approach. Humans have attempted to analyze these endgames, sometimes with only limited degrees of success, so it would be interesting to see whether my approach could be used to identify human-interpretable tactical patterns (Nunn, 1995, 1994; Roycroft, 1988).

The pattern grouping algorithms described in Section 3.1 could also be applied to games with larger state spaces by separating the state spaces into sections and collapsing the sections one at a time to iteratively perform retrograde analysis. Using the goal-related pattern grouping algorithms to search for paths to other goals, such as piece captures or favorable positions, would also allow my approach to be applied to more complex games. If the action redescription procedure from Chapter 6 could be used to compress information in endgame databases it would allow larger endgame databases to be created.

7.1.4 Applying the Approach to Other Types of Networks

In this thesis, I introduce a set of pattern grouping algorithms specifically designed to analyze state spaces. Developing additional pattern grouping algorithms for other types of networks, such as social networks, may yield insights in other domains. The interactive computer-aided mode of discovery described in Appendix A could be used to develop these algorithms.

7.2 Contributions

Because I have used the domain of games to test and refine principles of use-driven concept formation, I have made contributions both within the domain of games and within the realm of concept formation. Within the domain of games, I developed a semi-automated approach for extracting human-interpretable descriptions of offensive and defensive tactics from game databases. This approach introduces a set of pattern grouping algorithms that identify game-general concepts and expose high-level structure in game state spaces. By applying them to five two-player games with fifty-six states through forty-two million states apiece, I provided preliminary evidence that the pattern grouping algorithms can segment game state spaces in a manner that extracts important strategic information and exposes visual concepts when used in conjunction with the equivalence class principle.

In a detailed case study of five men's morris, I introduced a procedure for identifying use-driven visual concepts and using them to concisely describe tactics. The procedure for creating the concise descriptions of tactics, called redescribed action sequences, builds on Lock and Epstein's work on action sequences and Cazenave's work on patterns with external conditions (Lock & Epstein, 2004; Cazenave, 2001). By applying the procedure to five men's morris, I provided preliminary evidence that this approach can express strategic information in a human-interpretable form and compress the space required for storing information about optimal play by several orders of magnitude. Chapter 6 explores these game-related contributions in more detail.

Within the realm of concept formation, I introduced two semi-automated approaches for identifying use-driven concepts: a semi-automated process for exposing high-level conceptual patterns in networks (see appendix A) and a semi-automated process for using classes in one representation to expose concepts in another. In coordination with the latter approach, I proposed the equivalence class principle as a new inductive bias for learning in a multi-representational context and used the domain of two player games to provide evidence that this mechanism can facilitate concept formation.

To test and refine the equivalence class principle, I used it to extract use-driven visual concepts in five two-player perfect information games. Because I developed and refined the pattern grouping algorithms while analyzing three of the games, pong hau k'i, tic-tac-toe, and to a lesser extent lose tic-tac-toe, the morris games provide the true test of whether the equivalence class principle exposes use-driven visual concepts.

In five men's morris, application of the equivalence class principle and pattern grouping algorithms exposed inside-out symmetry and enabled the compression of 1,727 symmetrically-distinct states into thirty-five groups, each of which exhibited a different set of tactics and contexts. This compression relied on a procedure for redescribing action sequences in terms of use-driven visual concepts, using pattern grouping algorithms to extract offensive and defensive action sequences, and subdividing the pattern grouping algorithm output based on the number of pieces on the board and whether a state belonged to the opening or endgame. In other words, extracting the use-driven concepts required task-specific processing. Regardless, application of the principle did separate game states in a meaningful way that enabled the identification of use-driven concepts and tactics, albeit with a certain level of human processing.

If the equivalence class principle had no merit, one would expect that no level of processing or no consistent procedure for processing would extract useful concepts. Thus, the evidence supports the conclusion that the equivalence class principle has merit. This leads to questions about the conditions under which the equivalence class principle fails, the conditions under which it succeeds, and the level of processing required for the equivalence class principle to yield meaningful results.

Tic-tac-toe provides examples of when the equivalence class principle fails to expose visual concepts. For instance, most draw tree roots do not exhibit a visual pattern. This may be because the roots for paths to wins exhibit visually-salient patterns, and the draw tree roots are simply states that do not exhibit patterns with strategic significance. Alternatively, the draw tree roots might exhibit a set of meaningful patterns, just as the forced avoidance roots within a single five men's morris category exhibit multiple sets of tactics. If this were true, more processing would be required to identify the set of meaningful patterns.

The avoidances of three-move paths to a win in tic-tac-toe share a common pattern in that O can place its mark in a position that creates the context associated with a three-move path to a win, but using visual concepts to describe this pattern starts to become convoluted. The members of this category can more easily be described as a set of special cases. In essence, the board diagrams in this category encode enough information to distinguish them from other tic-tac-toe states, but making this distinction requires processing. The amount of processing required to distinguish between different types of states may simply increase as the length to the goal increases, regardless of the game. This observation might hold in five men's morris as well, suggesting that the extraction of tactical information from visual features may require more processing for states that are further away from goals. Regardless, it suggests that classes of states obtained using the equivalence class principle require varying levels of processing.

Outside of the domain of games, determining a priori how well the equivalence class principle will perform in a particular situation may be like trying to determine a priori whether Occam's razor will hold. Although Occam's razor has guided scientists for centuries, numerous well-documented cases exist in which Occam's razor stalled scientific progress by leading scientists to favor the wrong models (Gernert, 2007; Knowles, 1990). The inability to predict when an inductive bias such as the equivalence class principle or Occam's razor will hold may in fact be a more general feature of inductive biases.

Because the equivalence class principle exploits situations in which two representations express the same information in different ways, the only means for determining whether the principle applies may be to test whether the principle reveals easily detectable correlations. However, the efficacy of the equivalence class principle depends not only on whether two representations encode the same information in different ways, but also on how implicitly or explicitly the representations encode that information. When the equivalence class principle fails to produce easily detectable connections between representations, it is unclear how to determine whether the failure is due to a lack of shared information or due to implicit encoding of the information.

One case in which implicit encoding may be the issue is the notoriously difficult KBBKN endgame. To give the reader a sense of its difficulty, a chess endgame expert spent a year studying this endgame database to determine whether he could teach himself the endgame (Roycroft, 1988). At the end of the year, he still did not understand it well enough to successfully execute a winning strategy from all winning positions.

Chess board diagrams for the KBBKN endgame implicitly encode the paths to a win but may not do so in a manner that facilitates application of the equivalence class principle. With the rules of the game and enough processing power, the board diagrams contain enough information to calculate the optimal path to a win via brute-force search, so the board diagrams do implicitly store this information. However, when information is encoded this implicitly, the equivalence class principle may not be able to extract information that provides a computational advantage over a brute-force search in the endgame database. If, however, the board diagrams encode information about the paths to a win in a way that requires less processing than a brute-force search in the state space, using the equivalence class principle could reveal use-driven concepts that would provide computational savings. While identifying and using such a set of use-driven concepts might require less computational effort than a full brute-force search, identifying the use-driven concepts might still require significant computational effort.

This thesis provides an initial exploration into the strengths and limitations of the equivalence class principle. It provides evidence that the principle can reveal offensive and defensive tactics and strategically-important use-driven concepts in a variety of games. Identifying use-driven concepts currently requires a human to analyze sets of board diagrams and action sequences, but the equivalence class principle provides consistent enough results to warrant further study. Applying it to more games and to a more diverse set of domains will ultimately reveal its power and limitations.

Use-driven concepts were introduced in the first chapter as concepts that make problems in complex domains more tractable. The domain of game analysis provides a task-specific characterization of what it means to be a valuable use-driven concept. In this case, the task of interest is concisely storing tactical information and any concept that helps to perform that task is a use-driven concept. More specifically, a use-driven concept provides a vocabulary for concisely summarizing action sequences and the contexts in which those action sequences apply. The most valuable use-driven concepts provide the highest levels of compression. While a use-driven concept's ability to compress important domain-specific information is probably an effective criteria for ascertaining a concept's value in a variety of situations, the exploration of use-driven concept formation in a diverse set of domains will most likely result in a set of criteria for identifying valuable use-driven concepts.

Appendix A

Using Pattern Grouping Algorithms to Expose Concepts in the State Space

In this appendix, I describe an alternate use case for the pattern grouping algorithms in which the user interactively performs computer-aided discovery. To employ this mode of interactive computer-aided discovery, the user first identifies a frequently-occurring topological pattern in a network and specifies a set of attributes that nodes and edges in the pattern must possess. The computer then collapses and labels all instance of the user-specified pattern, creating a simplified network. After that, the user can identify another frequently-occurring topological pattern in the simplified network, and the computer can in turn collapse instances of that pattern within the simplified network. When the topological patterns identified through this interactive process have meaning in the domain of interest, the process can be seen as identifying concepts. For example, I used this technique to identify two concepts that play an important role in tic-tac-toe: forced avoidances and forced avoidances that lead to a fork to a win.

This type of interactive computer-aided discovery has not been thoroughly studied, but I used it to develop the pattern grouping algorithms and labels described in section 3.1. The discussion in this appendix has been included because one could easily apply this mode of interactive computer-aided discovery to other types of networks, and it seems interesting enough to warrant more study. A more thorough examination of this use case falls outside of the scope of this thesis.

I developed the pattern grouping algorithms in section 3.1 by iteratively applying early versions of the pattern grouping algorithms to small subtrees of the tic-tac-toe state space and examining those subtrees for topological patterns. Through this process, I created specialized versions of the algorithms from section 3.1. Applying the specialized versions to lose tic-tac-toe yielded insights that inspired the generalized versions of the pattern grouping algorithms that I used throughout the thesis. In the remainder of the appendix, I describe this progression and highlight the rediscovery of the forced avoidance concept and the concept of a forced avoidance that leads to a fork to a win.

To begin the computer-aided discovery process, I started with the one-move-to-a-goal grouping algorithm from Section 3.1.1, which I had previously developed while analyzing pong hau k'i. I applied this algorithm to a portion of the tic-tac-toe state space to obtain the simplified state space diagram shown in Figure A-1. This simplified diagram contains several instances of forks to a win. Based on this observation, I wrote the first version of the fork pattern grouping algorithm from Section 3.1.2.[1]

Collapsing the forks in Figure A-1 yields the simplified diagram shown in Figure A-2. When I examined this diagram, I noticed that many nodes contain multiple children with the same label. To create a cleaner version of the state space diagram, I grouped these children into a single node to obtain the diagram shown in Figure A-3. Grouping the children in this manner highlights the fact that a player often has only one move that keeps her opponent from winning. In other words, grouping the children in this manner highlights the concept of a forced avoidance. Based on this observation, I wrote a specialized version of the avoidance pattern grouping algorithm that only

[1] In the interest of keeping the figures readable, I will sometimes show slightly smaller portions of the simplified tic-tac-toe state space than what I considered when creating the algorithms. For example, when I show a diagram that contains a subtree with a root that has four marks on the board, I may have also considered a subtree with a root that has three marks on the board. Larger portions of the state space simply contain more instances of the highlighted topological patterns of interest.

recognizes forced avoidances. Applying this algorithm to the diagram in Figure A-3 results in the diagram shown in Figure A-4a. Collapsing forced avoidances also makes it possible to consider larger portions of the state space at once, as shown in Figure A-4b.

In the simplified state space diagrams from Figure A-4, most uncollapsed nodes lead to draw states. To create a cleaner picture of these state spaces, I created the draw tree pattern grouping algorithm from Section 3.1.4. Applying this algorithm to the diagrams in Figure A-4 yields the diagrams in Figure A-5.

After creating the draw tree pattern grouping algorithm, I generalized the one-move-to-a-goal grouping algorithm to collapse parts of the state space that are one move away from a fork to a win. For example, I collapsed instances in which X moves next and X can create a fork to a win during its next turn. Figure A-6 shows examples of the one-move-to-a-fork pattern. Creating this pattern grouping algorithm was the first step toward creating the fully generalized N-move-to-a-goal grouping algorithm described in Section 3.2.

By collapsing instances of the one-move-to-a-fork pattern, I created the simplified state space diagram shown in Figure A-7, which exposes forced avoidances that enable a player to create a fork. Not having paid much attention to tic-tac-toe before embarking on this project, I was surprised by the importance of this concept within tic-tac-toe. The forced avoidance to a fork enables players to force a win in three moves, and it often enables X to force a win by creating a forced avoidance during its second move of the game.

After creating a pattern grouping algorithm that collapsed forced avoidances that lead to a fork in one move, I generalized the fork and one-move-to-a-goal algorithms so that they could be applied recursively. Thus, I created the initial version of the recursive algorithms described in Section 3.2. In this version, only the N-move-to-a-goal and fork grouping algorithms were applied recursively. The draw tree and forced avoidance grouping algorithms were applied once between the first and second repetition of the other two algorithms.

After applying this sequence of algorithms to lose tic-tac-toe, it became apparent

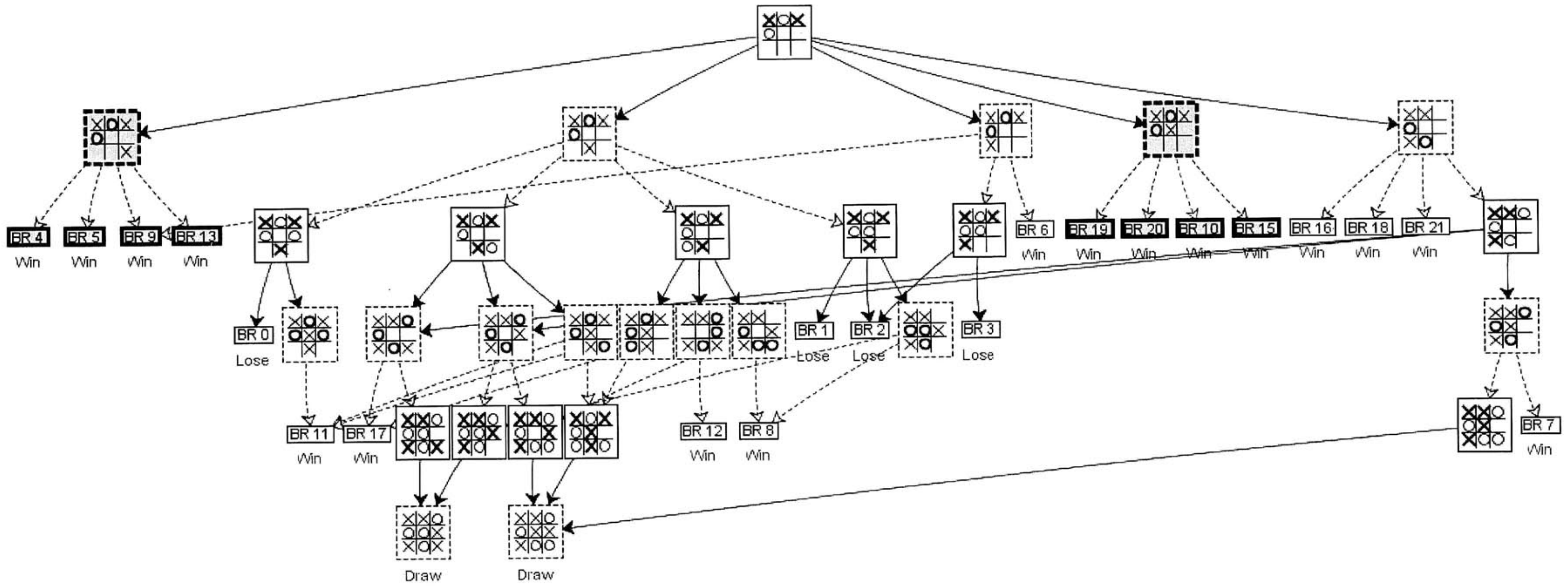


Figure A-1: A portion of the tic-tac-toe state space that contains forks that lead to a win for X. Here, nodes labeled "win" are paths that lead to a win for X in one move and nodes labeled "lose" are paths that lead to a win for O in one move. This view of the state space inspired the fork pattern grouping algorithm.

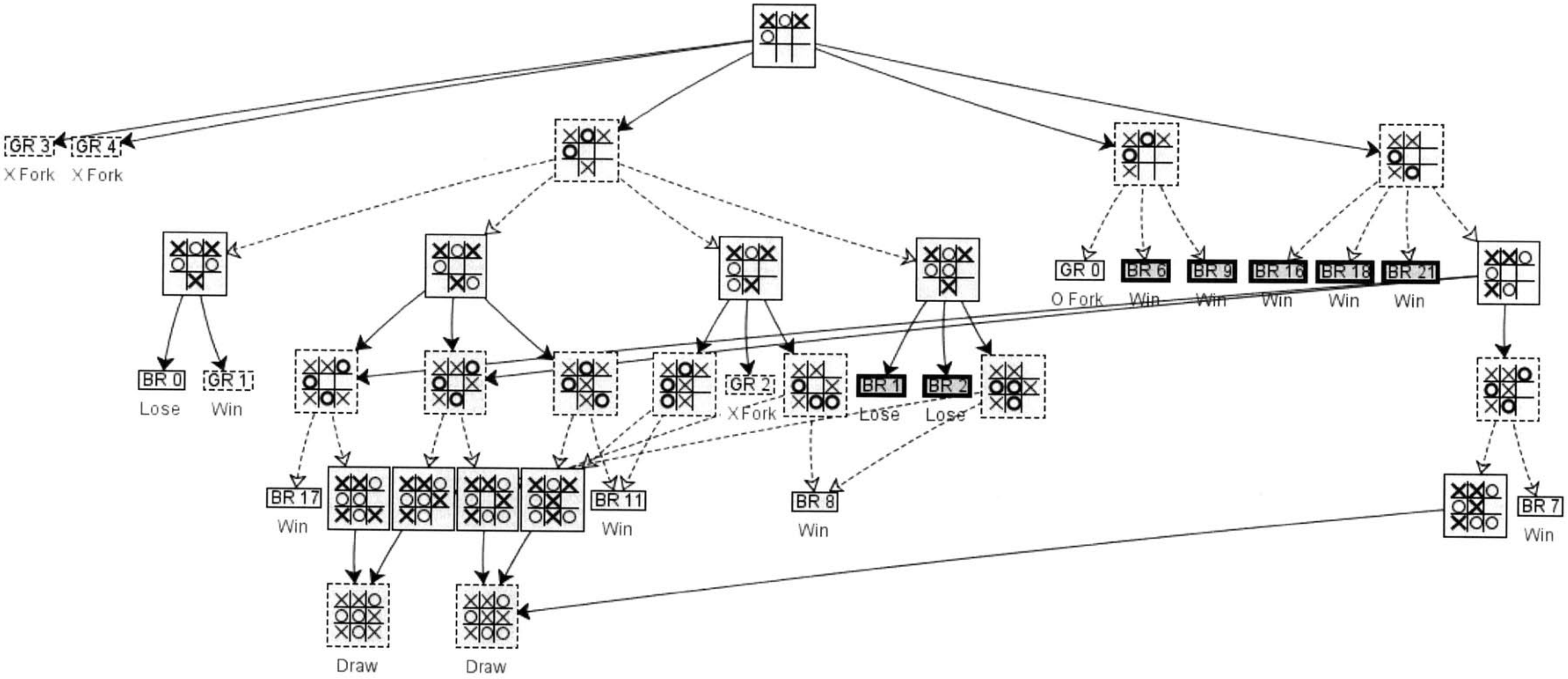


Figure A-2: A subtree of the tic-tac-toe state space after the fork pattern grouping algorithm has been applied. Forks to a win for X are labeled "X Fork" and forks to a win for O are labeled "O Fork." This version of the state space exposes nodes that have multiple children with the same label, which inspired another pattern grouping algorithm.

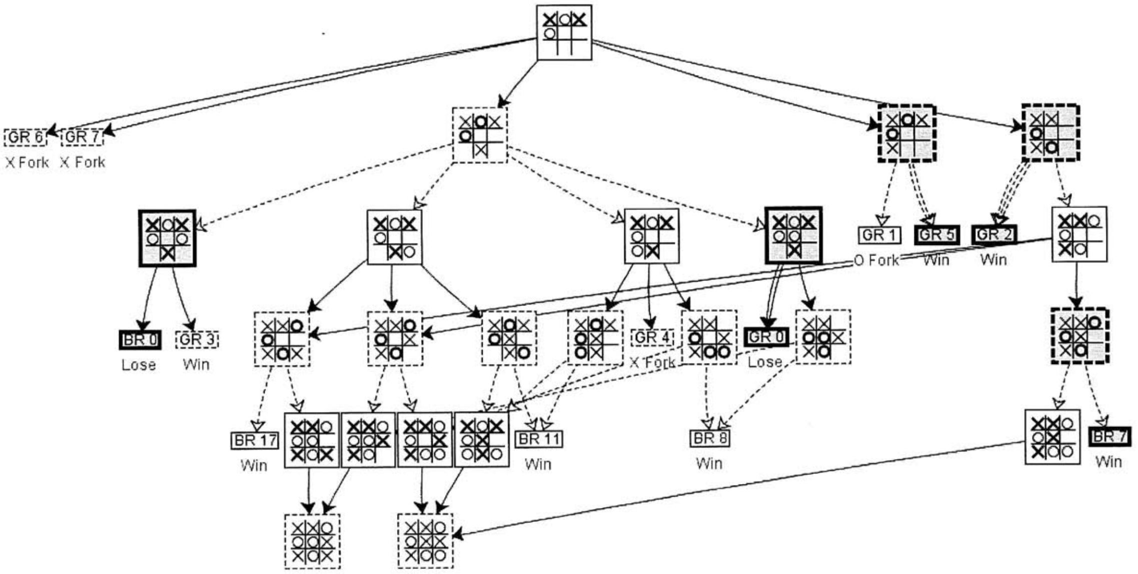
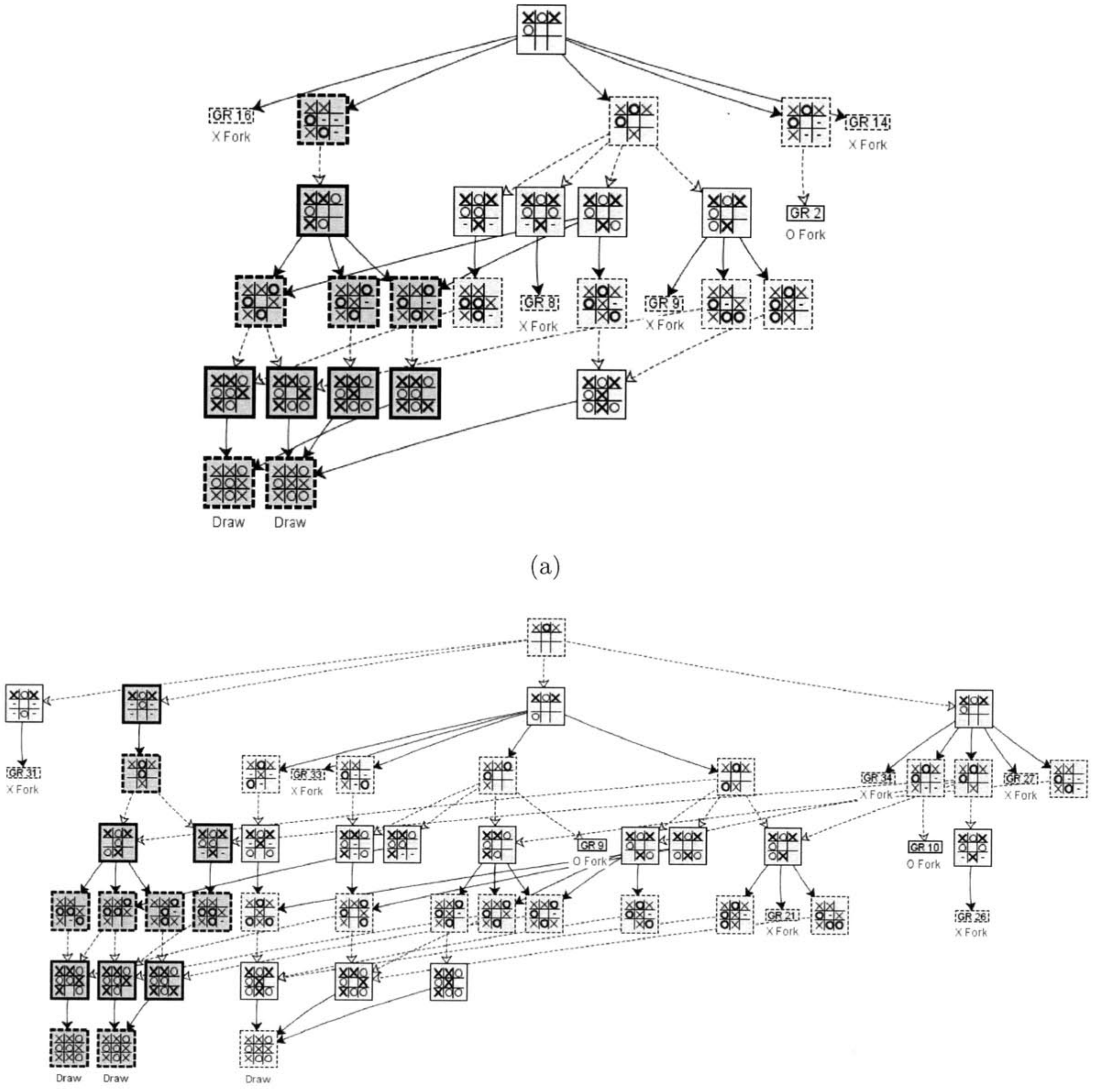


Figure A-3: Collapsing children that share the same "win" or "lose" label exposes the forced avoidance pattern. Here, a node's outline indicates which player moves next. A solid outline indicates that X moves next, and a dotted outline indicates that O moves next. The label "win" again indicates that X wins, while the label "lose" indicates that X loses. In this diagram, a forced avoidance consists of a parent node with two child nodes: one (highlighted) child node that leads to a loss for the player who moves next and one child node that does not lead to a loss for that player. Instances of this pattern inspired the forced avoidance pattern grouping algorithm.



(a)

(b)

Figure A-4: (a) The state space from Figure A-3 after the forced avoidances have been collapsed. Here, forced avoidances are depicted as tic-tac-toe boards where the incorrect moves are marked with a small horizontal line. (b) Collapsing the forced avoidances makes it possible to view larger parts of the state space at once. In both subfigures, the majority of the uncollapsed nodes lead to draw states. These depictions of the state space inspired the draw tree pattern grouping algorithm.

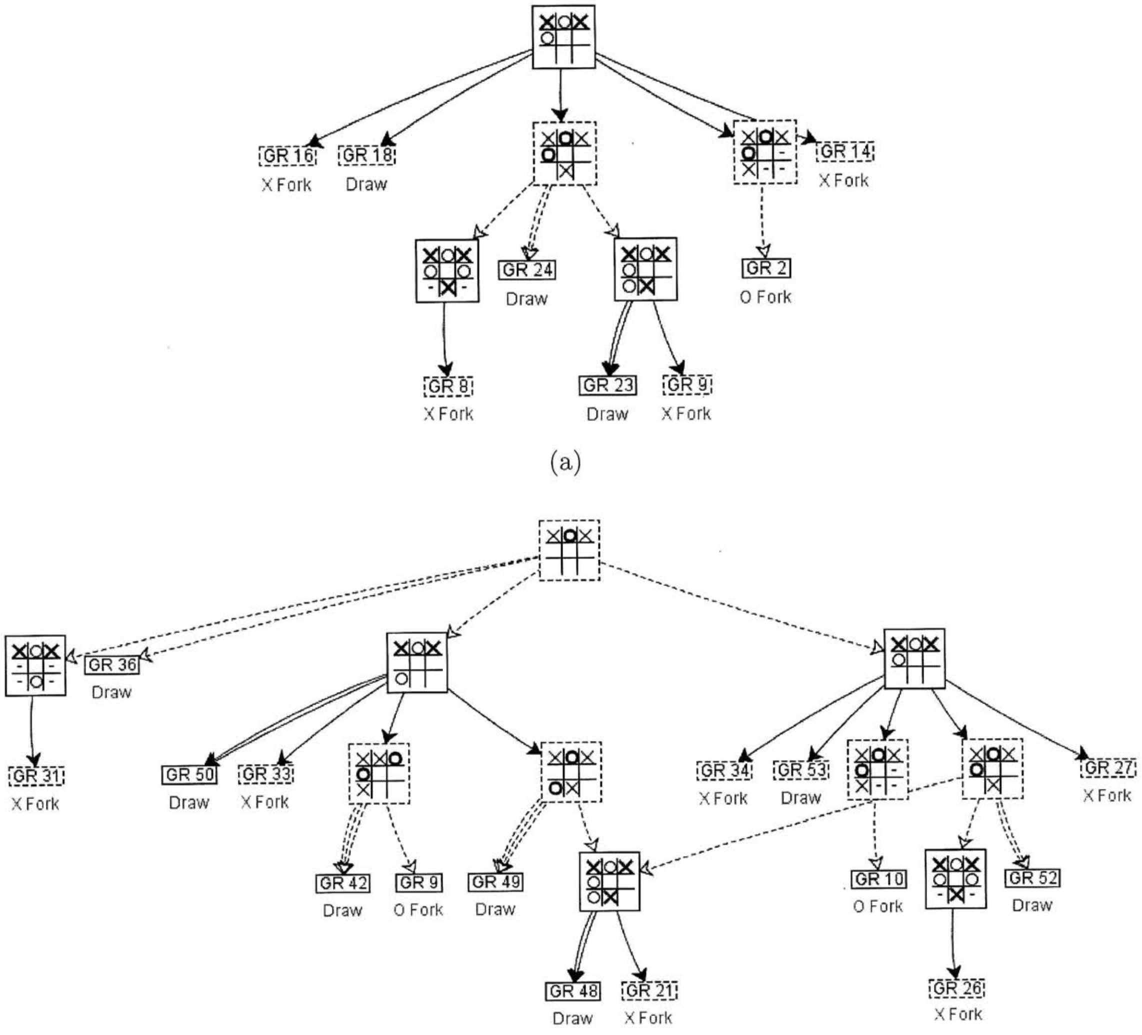


Figure A-5: Collapsing draw trees within the state spaces diagrams from Figure A-4 creates the diagrams shown here. These diagrams highlight decision points that affect the outcome of the game.

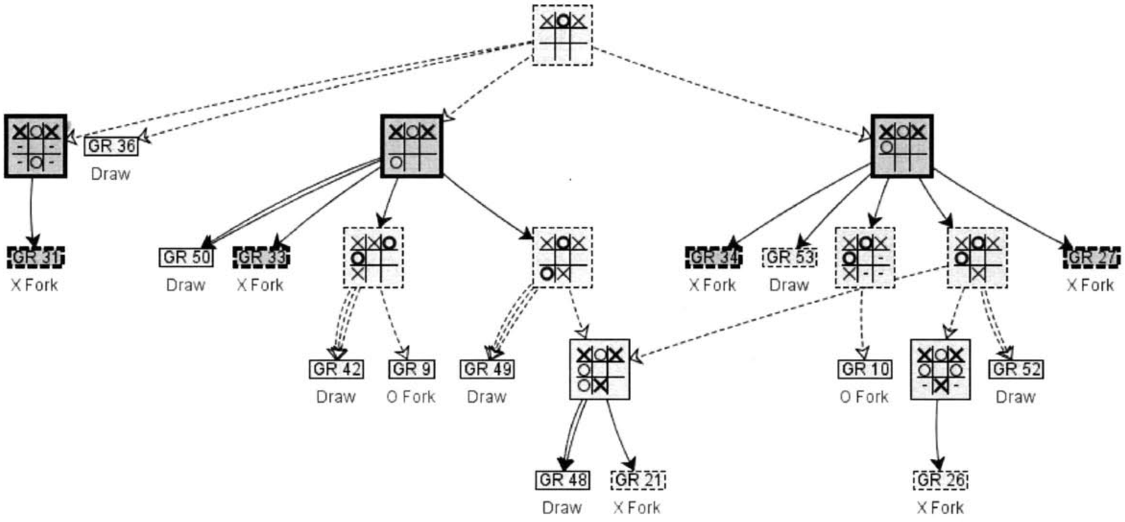


Figure A-6: Each highlighted set of nodes displays a situation in which X can create a fork to a win during its next move. This observation inspired the one-move-to-a-fork pattern grouping algorithm. This algorithm collapses subtrees in which a player can select at least one move that enables her to force a win.

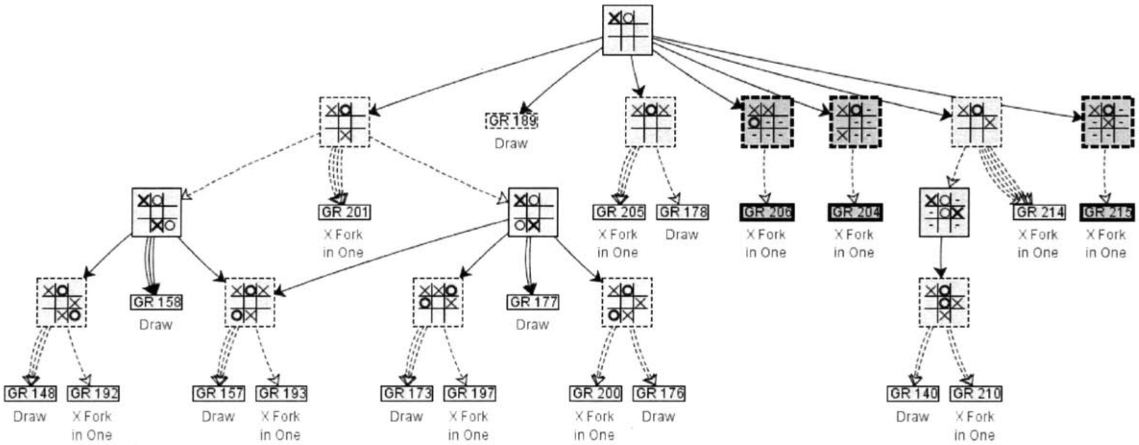


Figure A-7: Collapsing instances of the one-move-to-a-fork pattern reveals forced avoidances that allow a player to create a fork in one move. This is an important concept in tic-tac-toe.

that a forced avoidance is a special case of a more general pattern that involves both forced and multi-choice avoidances. Because lose tic-tac-toe did not have nearly as many forced avoidances as tic-tac-toe, I generalized the forced avoidance pattern grouping algorithm to recognize instances of both forced and multi-choice avoidances. Thus, I created the avoidance pattern grouping algorithm described in Section 3.1.3.

Similarly, early versions of the N-move-to-a-goal and fork-to-an-N-move-win algorithms labeled all collapsed subtrees as paths to a win and forks to a win, without considering subtree features. Subsequent versions of the algorithms differentiated between forced-move and decision-point subtree roots and labeled collapsed subtrees based on the number of moves to a win. I began recursively applying the avoidance pattern grouping algorithm, and I again encountered segmented state space diagrams such as those in Figure A-4 that were dominated with paths leading to a draw. To address this situation, I created a recursive version of the draw tree grouping algorithm that collapsed subtrees after each iteration of the avoidance grouping algorithm and labeled subtrees based on the look-ahead required to force a draw. This final step created the version of the algorithms described in Chapters 3, 5, and 6.

Throughout the development process, exploring partially collapsed versions of the state space exposed topological patterns and game-related concepts such as forced avoidances. It inspired the creation of new pattern grouping algorithms that made it possible to visually explore increasingly large portions of the state space. In the future, this mode of interactive computer-aided discovery could be applied to other domains to identify high-level structure, expose concepts, and hierarchically organize networks.

Appendix B

Symbiotic Sets of Representations

> *The cognitive linking of representations creates a whole that is more than the sum of its parts.... It enables us to see complex ideas in a new way and apply them more effectively.*
>
> – James J. Kaput, 1989

In my work on the equivalence class principle, I attempted to computationally characterize a mechanism for building connections between representations. The ideas behind the equivalence class principle emerged during an effort to taxonomize the space of knowledge representations. This work revealed that humans often create multiple representations to describe the same domain. It also suggested that sets of representations become powerful when one can build connections between and translate between representations in a set. In this appendix, I review previous efforts to characterize the space of representations, highlight research on sets of representations, and propose three computational properties that make sets of interconnected representations powerful.

B.1 Characterizing the Space of Representations

A representation is a formal system that

- Makes certain entities, relationships, and types of information explicit (Marr, 1982; Winston, 1993),
- Exposes constraints inherent in a domain or problem (Winston, 1993),
- May make other information hard to recover (Marr, 1982), and
- Supports a specific set of operations (Larkin & Simon, 1987; Boshuizen & (Tabachneck-)Schiff, 1998).

Most methods used to construct a taxonomy of representations involve analyzing each representation individually and identifying features that make each representation distinct or powerful (see Cox, 1996 and Ainsworth, 2006 for reviews). Researchers have created taxonomies of this sort using "a variety of methods (e.g. intuition, analysis of domain properties, and card sort techniques with subjects), and although there is some overlap between the taxonomies, no one classification is universally accepted. [The taxonomies] differ in domains addressed, the granularity with which representations are described, and the task for which they were created" (Ainsworth, 2006, pp. 10).

For example, Lohse, Biolsi, Walker, and Rueler (1994) analyzed how human subjects classified and rated sixty visual representations. Subjects rated the representations using the following scales:

- spatial versus non-spatial,
- easy versus hard to understand,
- numeric versus non-numeric,
- continuous versus discrete,

- concrete versus abstract,
- nontemporal versus temporal,
- attractive versus unattractive,
- emphasis of wholes versus parts,
- displaying of static structure versus dynamic processes, and
- conveying of large versus small amounts of information.

Using both types of human data, the authors identified eleven types of visual representations, including graphs, tables, process diagrams, and cartograms. The resulting classification system separated some visual representations based on the type of information conveyed and others based on the mechanisms used to convey the information.

Instead of identifying classes of representations, De Jong et al (1998) proposed five dimensions for characterizing representations: perspective, precision, modality, specificity, and complexity. A representation's perspective describes the types of concepts that play a central role within the representation. For example, circuits can be described from a functional perspective that focuses on the circuit's purpose, a behavioral perspective that focuses on each component's input-output behavior, or a physical perspective that focuses on each component's internal physics. Perspective is closely related to a representation's ontology, which is an enumeration of the concepts used by a representation.

Precision describes the level of abstraction and distinguishes between qualitative and quantitative information. Modality describes the form of the representation, such as text, animations, diagrams, graphs, formula, real-life videos, and tables. Specificity (Stenning & Oberlander, 1995) is intended to capture computational differences between representations and explain why one modality is better than another for describing a particular type of information. Finally, complexity measures the amount of information presented using a single representation.

Alpay, Giboin, and Dieng (1998) developed a set of features for describing representations within collaborative settings. Features such as internal vs. external and shared vs. unshared apply specifically to representations used in inter-personal contexts. In contrast, the level of abstraction and degree of permanence apply regardless of the context. The degree of permanence distinguishes between representations that are reused in many contexts (e.g., frameworks) and ones that are built dynamically to describe a particular situation or scenario.

Davis, Shrobe, and Szolovits (1993) proposed five purposes for representations. Representations provide a simplified, compressed characterization of real-world information and offer a set of ontological commitments that dictate how and what to attend to in the world. They also support a mechanism for forming inferences and favor particular types of inferences. Finally, representations support efficient computation and serve as a medium for communication.

Boshuizen and (Tabachneck-)Schijf (1998) break representations into two parts: the format used to record, store, and present the information, and the operators available for modifying the information. This breakdown highlights computational differences between informationally equivalent representations, because two representations that share the same information content but have different operators will have different computational properties.

Larkin and Simon (1987) and Stenning and Obderlander (1995) also offer computational explanations for why different representations lend themselves to different types of tasks. Larkin and Simon introduced the idea of informational equivalence and compared informationally equivalent diagrams and logical representations. They concluded that diagrams provide computational benefits because they localize information and reduce the need for search during problem solving. Stenning and Oberlander highlight the difference in abstraction between diagrams and linguistic representations. For example, the abstract concept of "to the side" can be represented linguistically, but a pictorial representation will display an item either to the left or to the right.

B.2 Related Work on Sets of Representations

My taxonomy of representations highlights a representation's ontology, format, and operators, but instead of attempting to classify or characterize the differences between representations, I seek to define characteristics that a set of representations must share to make it possible to build connections between members of the set. In this section, I review other research on sets of representations.

Ainsworth (1999, 2006) provides an organizational structure for studying sets of related representations based on the roles played by each representation in the set. More specifically, Ainsworth studies the circumstances under which learners benefit from a learning environment that highlights multiple representations within the same domain. She proposed that within a learning or problem-solving environment, sets of representations are beneficial when they perform one or more of the following functions: (1) when they support complementary information or processes, (2) when a familiar representation makes it easier to interpret an unfamiliar representation or removes ambiguity from an otherwise ambiguous representation, or (3) when multiple representations provide different views that lead to a deeper conceptual understanding of a domain.

Although the ability to translate between multiple representations is a noteworthy characteristic of expert behavior (Kozma, Chin, Russell, & Marx, 2000), learning how to connect representations can be challenging. Kozma et al. (2000, pp. 6) suggests that "the meaning of a representation is often generated by coordinating features within and across multiple representations." However, research on multi-representational learning environments indicates that learners find the task of integrating representations challenging, and suggests that the extent to which multi-representational learning environments help learners to understand a domain depends the extent to which learners stop treating the representations in isolation and start building connections between the representations (see (Ainsworth, 2006) for a review). Educational research on multiple external representations seeks to identify the conditions under which multi-representational learning environments promote the creation of connections between representations, but such work is ongoing (Ainsworth, 2006; Goldman, 2003). While the educational research focuses on identifying features within a learning environment that facilitate the creation of connections between representations, the emphasis in my work has been to characterize the features that representations must share to make such connections possible.

B.3 Characterizing What Makes Sets of Representations Powerful

By analyzing approximately one hundred representations used either by humans to study math, science, engineering, or music, or by computers to reason about causality, space, action, or time, I determined that humans tend to exploit sets of representations within the same domain, while computers traditionally use only one type of a representation at a time. I observed that the power inherent in a set of representations lies in the ability to translate between the representations. In this section, I propose a set of computational properties that enable sets of representations to form an interconnected whole that is greater than the sum of its parts.

Within this section, I use *ontology* to describe a representation's core set of concepts. *Format* refers to the medium used to describe a representation. For example, plots and diagrams are examples of visual formats, while equations and logic are examples of grammar-based formats. Finally, I define a *symbiotic set of representations* to be a set of representations whose members

- Share an overlapping set of core concepts,
- Share a translation mechanism, and
- Support complementary operators.

Examples of symbiotic sets include chemical formula, Lewis diagrams, and three-dimensional models in chemistry; plots, difference equations, block diagrams, and Z-transforms in signal processing; force diagrams, plots, and equations in physics;

and cartesian planes, polar coordinate systems, algebraic equations, and geometrical representations[1] in mathematics.

The first of the three criteria requires that representations in a symbiotic set have interrelated ontologies. For example, the cartesian plane representation features concepts such as above, below, and steepness of a curve, while the equation representation features concepts such as coefficients, terms, and addition. Because both representations also share concepts such as slopes and intercepts, these representations have overlapping ontologies. In this case, unshared concepts tend to capture visual aspects of the cartesian plane and algebraic aspects of the equations, while shared concepts highlight connections between the two representations.

The shared concepts support translation, which is the second component of a symbiotic set. In the cartesian plane example, numerical descriptions of a line's slope and intercept within the equation representation translate directly into visual features within the cartesian plane. This correspondence enables one translate an equation of a line into a cartesian plot by identifying the position of the y-intercept, using the slope and y-intercept to plot a second point, and connecting the two points to form a line.

The equation and cartesian plane representations support complementary sets of operators that facilitate mathematical and visual operations, respectively. These complementary sets of operators make it easier to, for example, identify minima and maxima in the cartesian plane representation and compute the sum of two functions in the equation representation. Because the representations have overlapping ontologies, support a translation mechanism, and facilitate different types of operations, they form a symbiotic set of representations that provide interconnected perspectives of mathematical concepts.

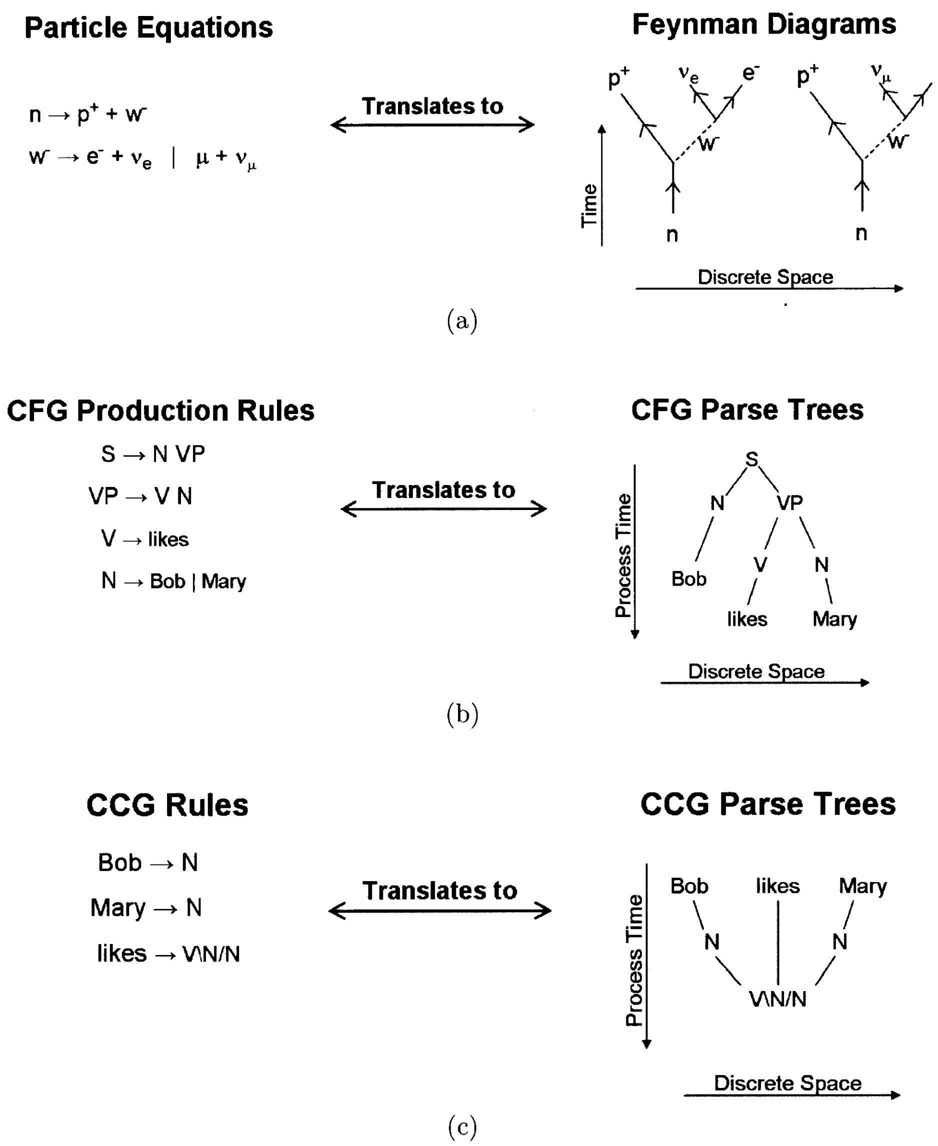
While analyzing sets of representations, I observed that a representation's format can provide a domain-general mechanism for translation. This mechanism relies on creating one-to-one correspondences between elements within each format. For exam-

[1] By a geometrical representation, I mean a representation with an ontology that emphasizes planes, surfaces, and intersections without necessarily situating the surfaces within a coordinate system.

ple, Figure B-1 provides three examples in which a production rule format translates into a tree format. In this example, elements within the production rules map to elements within the trees and provide a mechanism for translating between representations in both physics and natural language.

Some representations provide one-to-one correspondences with the real world and with an abstract representation. These representations create a mechanism for grounding concepts within an abstract representation's ontology. For example, the Dienes blocks shown in Figure B-2 provide a representation that bridges between an object-centric representation of number and a base-ten representation of number. The number of large blocks translates into the ten's digit of the base-ten representation, while the number of disconnected small blocks translates into the one's digit. At the same time, the total number of small blocks provides an object-centric description of the number thirteen. In the same way that Dienes blocks ground the meaning of one's and ten's digits, animations of people throwing balls and cars rolling down inclined planes ground the meaning of abstract concepts in physics. Symbiotic sets of representations often have at least one member that grounds the meaning of abstract concepts used by other members of the set.

The ability to translate between representations sometimes makes it possible to performs tasks that would be impossible using only a single representation. For example, if you wanted to build a circuit implementation of the top block diagram in Figure B-3, but you did not have the correct components, one way to circumvent this problem would be to create an alternate block diagram that uses different components to perform the same function. This task can be accomplished by translating the circuit into an equation representation, manipulating the equation, and translating the new form of the equation into a new block diagram. This translation process relies on one-to-one correspondences between elements in the block diagrams and elements in the equations. Translating from the block diagram representation to the equation representation and performing the processing within the equation representation makes it possible to design equivalent block diagrams. For sufficiently complex circuits, performing the equivalent manipulations entirely in the block diagram rep-



(a)

(b)

(c)

Figure B-1: Three examples of a representation with a production rule format translating into a representation with a tree format. These translations rely on one-to-one correspondences between attributes of the formats and do not depend on domain-specific information.

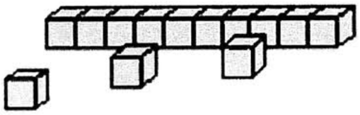


Figure B-2: The Dienes block representation provides a mechanism for translating between objects in the real world and the base-ten representation for number. The single large block translates into a ten's digit of one, while the three separate small blocks translate into a one's digit of three, creating a base-ten description of the number thirteen.

resentation would be impossible because the block diagram representation does not support the correct set of operators.

While studying the development of mathematical representations and representations within artificial intelligence, I observed that the format and ontology of a representation appear to constrain the types of changes that can be made to it when using it as a base for developing a new representation. This agrees with Richard Feynman's observation in his nobel lecture that different representations in physics "suggest different kinds of modifications." For example, first order logic builds on predicate logic by replacing variables with predicates. In predicate logic, variables map to either true or false. In first order logic, these variables are replaced by predicates that take an object as an argument and map that the object-predicate pair to either true or false. This makes it possible to express statements such as hasRedHair(Bob) = true. In this example, grammar constrains what types of changes can be made to predicate logic to create other types of logic. Studying the historical development of representations and examining how their format and ontology constrain the types of incremental changes that can be made when creating a new representation may shed light on the mechanisms that enable representational discovery.

In my efforts to characterize the space of representations, I created a taxonomy of approximately one hundred representations, introduced the concept of a symbiotic set of representations, and identified a set of computational criteria that make it possible to form connections between members of a symbiotic set and thus to create a whole

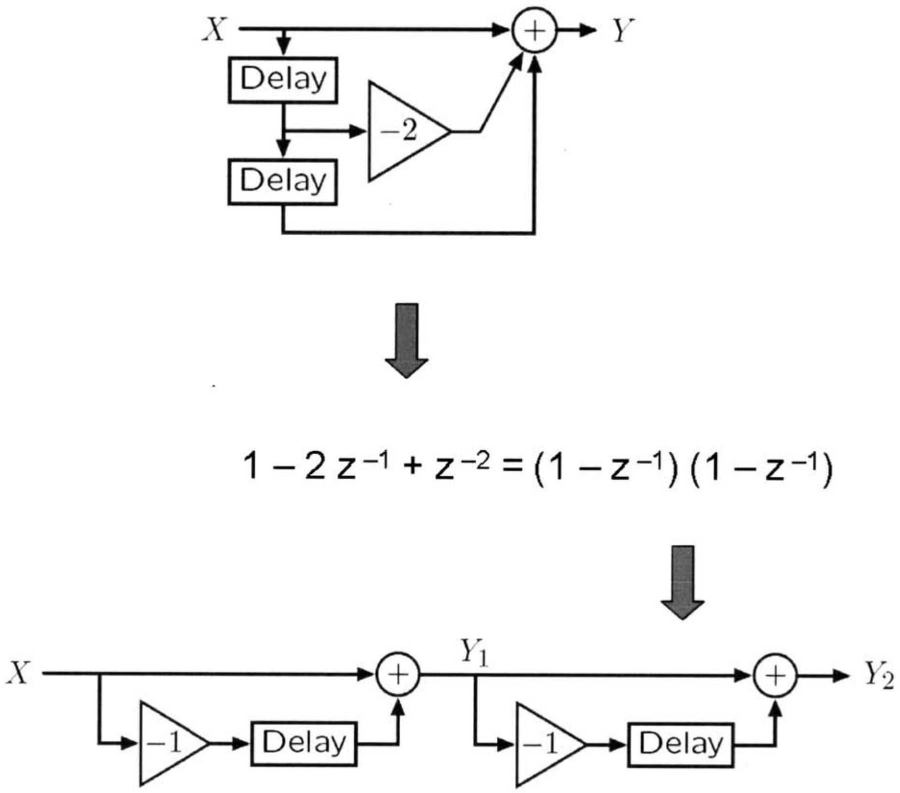


Figure B-3: In signal processing, creating a set of equivalent block diagrams requires translating the first block diagram into an equation, algebraically manipulating the equation into a new form, and translating the new form back into a second functionally equivalent block diagram. The translations involve one-to-one correspondences between parts of the block diagrams and parts of the equations.

that is greater than the sum of its parts. I also discovered a domain-general mechanism for translating between representations by exploiting one-to-one correspondences in a representation's format. Finally, I proposed a new direction for future research on representational discovery.

Bibliography

Ainsworth, S. (1999). The functions of multiple representations. *Computers & Education*, *33*, 131-152.

Ainsworth, S. (2006). Deft: A conceptual framework for considering learning with multiple representations. *Learning and Instruction*, *16*(3), 183-198.

Allen, J. (1989). A note on the computer solution of connect-four. In D. Levy & D. Beal (Eds.), *Heuristic Programming in Artificial Intelligence: The First Computer Olympiad* (p. 134-135). Ellis Horwood Limited.

Allis, L. V. (1988). *A knowledge based approach of connect four. the game is solved: white wins.* Master's thesis, Vrije Universiteit.

Allis, L. V. (1992). Qubic solved again. In H. van den Herik & L. Allis (Eds.), *Heuristic Programming in Artificial Intelligence 3: The Third Computer Olympiad* (p. 192-204). Ellis Horwood.

Allis, L. V. (1994). *Searching for solutions in games and artificial intelligence.* Ph.D. thesis, University of Limburg.

Allis, L. V., van den Herik, H., & Herschberg, I. (1991). Which games will survive? In D. Levy & D. Beal (Eds.), *Heuristic Programming in Artificial Intelligence 2: The Second Computer Olympiad* (p. 232-243). Ellis Horwood Limited.

Alpay, L., Giboin, A., & Dieng, R. (1998). Accidentology: An example of problem solving by multiple agents with multiple representations. In M. W. van Someren, P. Reimann, H. P. A. Boshuizen, & T. de Jong (Eds.), *Learning with Multiple Representations* (p. 152-174). Pergamon.

Bain, M., & Srinivasan, A. (1995). Inductive logic programming with large-scale unstructured data. *Machine Intelligence*, *14*, 233-267.

Bell, R. C. (1979). *Board and table games from many civilizations.* New York, NY: Dover Publications, Inc.

Boshuizen, H. P., & (Tabachneck-)Schiff, H. J. (1998). Problem solving with multiple representations by multiple and single agents: An analysis of the issues involved. In M. W. van Someren, P. Reimann, H. P. A. Boshuizen, & T. de Jong (Eds.), *Learning with Multiple Representations* (p. 137-151). Pergamon.

Burnside, E. S., Davis, J., Chhatwal, J., Alagoz, O., Lindstrom, M. J., Geller, B. M., et al. (2009). Probabilistic computer model developed from clinical data in national mammography database format to classify mammographic findings. *Radiology*, *251*, 663-672.

Buro, M. (1999). From simple features to sophisticated evaluation functions. In H. van den Herik & H. Iida (Eds.), *Lecture Notes in Computer Science: Com-*

puters and Games (Vol. 1558, p. 126-145). Springer Berlin / Heidelberg.

Buro, M. (2003). The evolution of strong Othello programs. In R. Nakatsu & J. Hoshino (Eds.), *Entertainment Computing - Technology and Applications.* Springer.

Cazenave, T. (2001). Generation of patterns with external conditions for the game of Go. In *Advances in Computer Games 9* (p. 277-296).

Coen, M. H. (2005). Cross-modal clustering. In *Twentieth National Conference on Artificial Intelligence (AAAI'05).*

Coen, M. H. (2006). Self-supervised acquisition of vowels in american english. In *Twenty First National Conference on Artificial Intelligence (AAAI'06).*

Coen, M. H. (2007). Learning to sing like a bird: An architecture for self-supervised sensorimotor learning. In *Twenty Second AAAI Conference on Artificial Intelligence (AAAI-07).*

Cohen, D. I. A. (1972). The solution of a simple game. *Mathematics Magazine*, *45*, 213-216.

Collins, G. (1987). *Plan creation: Using strategies as blueprints.* Ph.D. thesis, Yale University, Department of Computer Science.

Cox, R. (1996). *Analytical reasoning with multiple external representaitons.* Ph.D. thesis, University of Edinburgh.

de Jong, T., Ainsworth, S., Dobson, M., van der Hulst , A., Levonen, J., Reimann, P., et al. (1998). Acquiring knowledge in science and mathematics: The use of multiple representations in technology-based learning environments. In M. W. van Someren, P. Reimann, H. P. A. Boshuizen, & T. de Jong (Eds.), *Learning with Multiple Representations* (p. 9-40). Pergamon.

Epstein, S. L. (1991). Deep forks in strategic maps. In D. Levy & D. Beal (Eds.), *Heuristic Programming in Artificial Intelligence 2: The Second Computer Olympiad.* Ellis Horwood.

Epstein, S. L. (2005). Thinking through diagrams: Discovery in game playing. In *Spatial Cognition IV* (p. 260-283). Springer-Verlag.

Epstein, S. L. (June 18, 2010). Personal communication.

Epstein, S. L., & Keibel, J. H. (2002). Learning on paper: Diagrams and discovery in game playing. In *Diagrams* (p. 31-45). Callaway Gardens, GA: Springer Verlag.

Fajtlowicz, S. (1988). On conjectures of Graffiti. *Discrete Mathematics*, *72*, 113-118.

Fang, H., Hsu, T., & Hsu, S. (2002). Construction of chinese chess endgame databases by retrograde analysis. In *CG '00: Revised Papers from the Second International Conference on Computers and Games* (p. 96-114). London, UK: Springer-Verlag.

Finkelstein, L., & Markovitch, S. (1998). A selective macro-learning algorithm and its application to the NxN sliding-tile puzzle. *Journal of Artificial Intelligence Research*, *8*, 223-263.

Fürnkranz, J. (2001). Machine learning in games: A survey. In J. Fürnkranz & M. Kubat (Eds.), *Machines that Learn to Play Games* (pp. 11–59). Huntington, NY, USA: Nova Science Publishers, Inc.

Gasser, R. (1991). Applying retrograde analysis to nine mens morris. In D. Levy &

D. Beal (Eds.), *Heuristic Programming in Artificial Intelligence 2: The Second Computer Olympiad* (p. 161-173). Ellis Horwood Ltd.

Gasser, R. (1993). *Nine men's morris is a draw.* Posting to rec.games.abstract. Available from `http://www.ics.uci.edu/~eppstein/cgt/morris.html`

Gasser, R. (1996). Solving nine mens morris. *Computational Intelligence*, *12*, 24-41.

Gernert, D. (2007). Ockham's razor and its improper use. *Journal of Scientific Exploration*, *21*, 135-140.

Gobet, F., & Charness, N. (2006). Chess and games. In *Cambridge Handbook on Expertise and Expert Performance* (p. 523-538). Cambridge, MA: Cambridge University Press.

Goldman, S. R. (2003). Learning in complex domains: when and why do multiple representations help? *Learning and Instruction*, *13*, 239-244.

Guid, M., Mozina, M., Sadikov, A., & Bratko, I. (2010). Deriving concepts and strategies from chess tablebases. In *Lecture Notes in Computer Science: Advances in Computer Games.* Springer Berlin / Heidelberg.

Iba, G. A. (1989). A heuristic approach to the discovery of macro-operators. *Machine Learning*, *3*, 285-317.

Kaneko, Y. K., T., & Kawai, S. (2003). Automated identification of patterns in evaluation functions. In H. J. van den Herik, H. Iida, & E. A. Heinz (Eds.), *Advances in Computer Games: Many Games, Many Challenges.* Kluwer Academic Publishers.

Karmiloff-Smith, A. (1995). *Beyond modularity: A developmental perspective on cognitive science.* The MIT Press.

Kemp, C. (2007). *The acquisition of inductive constraints.* Ph.D. thesis, MIT.

King, R. D., Muggleton, S. H., Srinivasani, A., & Sternberg, M. J. E. (1996). Structure-activity relationships derived by machine learning: The use of atoms and their bond connectivities to predict mutagenicity by inductive logic programming. *Proceedings of the National Academy of Sciences*, *93*, 438-442.

Knowles, D. (1990). *Explanation and its limits.* Cambridge University Press.

Kojima, T. (1998). *Automatic acquisition of Go knowledge from game records: Deductive and evolutionary approaches.* Ph.D. thesis, University of Tokyo.

Kojima, T., Ueda, K., & Nagano, S. (1997). Flexible acquisition of various types of knowledge from game records: Application to the game of Go. In *In Proceedings of the IJCAI-97 Workshop on Using Games as an Experimental Testbed for AI Research* (pp. 51–57).

Kojima, T., Ueda, K., & Nagano, S. (2000). Flexible acquisition of various types of Go knowledge. In H. I. H.J. van den Herik (Ed.), *Games in AI Research* (p. 221238). Universiteit Maastricht, Maastricht.

Korf, R. E. (1985). Macro-operators: a weak method for learning. *Artificial Intelligence*, *26*, 35-77.

Kozma, R., Chin, E., Russell, J., & Marx, N. (2000). The roles of representations and tools in the chemistry laboratory and their implications for chemistry learning. *Journal of the Learning Sciences*, *9*(2), 105-143.

Kulkarni, D., & Simon, H. A. (1990). Experimentation in machine discovery. In

J. Shrager & P. Langley (Eds.), *Computational Models of Scientific Discovery and Theory Formation.* San Mateo, CA: Morgan Kaufmann.

Landau, B., Smith, L. B., & Jones, S. S. (1988). The importance of shape in early lexical learning. *Cognitive Development*, *3*, 299-321.

Langley, P. (1981). Data-driven discovery of physical laws. *Cognitive Science*, *5*, 31-54.

Langley, P. (1998). The computer-aided discovery of scientific knowledge. In *First International Conference on Discovery Science.*

Larkin, J., & Simon, H. (1987). Why a diagram is (sometimes) worth ten thousand words. *Cognitive Science*, *11*(1), 65-100.

Lenat, D. B. (1977). Automated theory formation in mathematics. In *Fifth International Joint Conference in Artificial Intelligence* (p. 833-842).

Levinson, R., & Snyder, R. (1991). Adaptive pattern-oriented chess. In L. Birnbaum & G. Collins (Eds.), *Proceedings of the* 8th *International Workshop on Machine Learning.*

Lock, E., & Epstein, S. L. (2004). Learning and applying competitive strategies. In *AAAI-04:* 19th *National Conference on Artificial Intelligence* (p. 354-359). San Jose, CA.

Lohse, G. L., Biolsi, K., Walker, N., & Rueter, H. H. (1994, December). A classification of visual representations. *Communications of the ACM*, *37*(12), 36-49.

MacKay, D. J. C. (2003). *Information theory, inference, and learning algorithms.* Cambridge University Press.

Marr, D. (1982). *Vision: A computational investigation into the human representation and processing of visual information.* New York: Henry Holt and Co., Inc.

McGovern, E. A. (2002). *Autonomous discovery of temporal abstractions from interaction with an environment.* Ph.D. thesis, University of Massachusetts Amherst.

Minton, S. (1984). Constraint-based generalization: Learning gameplaying plans from single examples. In *Proceedings of the National Conference on Artificial Intelligence* (p. 251-254).

Minton, S. (1990). Quantitative results concerning the utility of explanation-based learning. *Artificial Intelligence*, *42*(2-3), 363-391.

Mitchell, F., Sleeman, D., Duffy, J. A., Ingram, M. D., & Young, R. W. (1997). Optical basicity of metallurgical slags: new computer based system for data visualisation and analysis. *Ironmaking and Steelmaking*, *24*, 306-320.

Mitchell, T. M. (1980). *The need for biases in learning generalizations* (Technical Report CBM-TR-117). Computer Science Department, Rutgers University.

Muller, M. (2002). Computer Go. *Artificial Intelligence*, *134*, 145-179.

Murphy, G. L. (2002). *The big book of concepts.* Cambridge, MA: The MIT Press.

Nunn, J. (1994). *Secrets of pawnless endings.* Batsford.

Nunn, J. (1995). *Secrets of minor piece endings.* Batsford.

Ratterman, M. J., & Epstein, S. L. (1995). Skilled like a person: A comparison of human and computer game playing. In *Proceedings of the Seventeenth Annual Conference of the Cognitive Science Society* (p. 709-714).

Romein, J. W., & Bal, H. E. (2003). Solving the game of awari using parallel retrograde analysis. *IEEE Computer*, *36*, 26-33.

Roycroft, A. J. (1988). Expert against oracle. *Machine Intelligence*, *11*, 347-373.

Sadikov, A., & Bratko, I. (2006). Learning long-term chess strategies from databases. *Machine Learning*, *63*(3), 329-340.

Schaeffer, J., Burch, N., Bjornsson, Y., Kishimoto, A., Muller, M., Lake, R., et al. (2007). Checkers is solved. *Science*, *317*, 1518-1522.

Sei, S., & Kawashima, T. (1998). Memory-based approach in Go-program Katsunari. In *Complex games lab workshop.*

Spelke, E. S. (1990). Principles of object perception. *Cognitive Science*, *14*, 29-56.

Stenning, K., & Oberlander, J. (1995). A cognitive theory of graphical and linguistic reasoning: Logic and implementation. *Cogntive Science*, *19*(1), 97-140.

Stoutamire, D. (1991). *Machine learning, game play and Go, Technical Report TR 91-128* (Tech. Rep.). Case Western Reserve University.

Thompson, K. (1996). 6-piece endgames. *International Computer Chess Association Journal*, *19*, 215-226.

Valds-Prez, R. E. (1995). Machine discovery in chemistry: new results. *Artificial Intelligence*, *74*(1), 191-201.

van den Herik, H. J., Uiterwijk, J. W. H. M., & van Rijswijck, J. (2002). Games solved: now and in the future. *Artificial Intelligence.*, *134*(1-2), 277-311.

Winston, P. H. (1993). *Artificial Intelligence: 3rd edition.* Reading, MA: Addison-Wesley Publishing Company.

Wolpert, D. H. (1996). The lack of *a priori* distinctions between learning algorithms. *Neural Computations*, *8*, 1341-1390.

Wolpert, D. H., & Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, *1*, 67-82.

Yee, R. C., Saxena, S., Utgoff, P. E., & Barto, A. C. (1990). Explaining temporal-differences to create useful concepts for evaluating states. In *Proceedings of AAAI-90.*

Zaslavsky, C. (1982). *Tic-tac-toe and other three-in-a-row games, from ancient Egypt to the modern computer.* New York: Crowell.

Zytkow, J. M., & Simon, H. A. (1986). A theory of historical discovery: The construction of componential models. *Machine Learning*, *1*, 107-137.